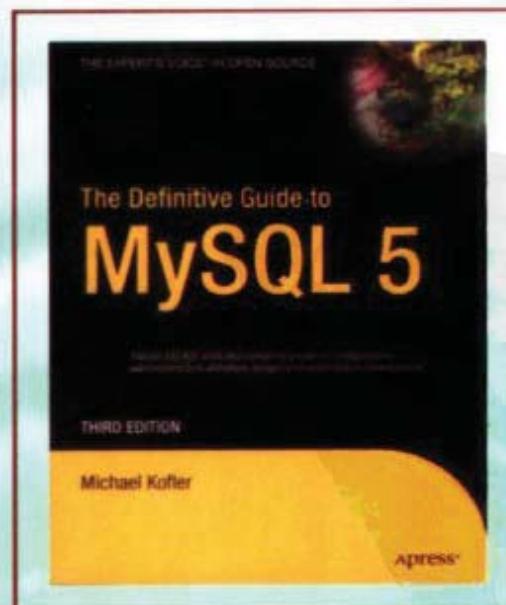


The Definitive Guide to MySQL 5 Third Edition

MySQL 5 权威指南 (第3版)

[奥] Michael Kofler 著
杨晓云 王建桥 杨 涛 等译

- MySQL 经典图书
- 讲述最新版本 MySQL 5
- 覆盖 PHP、Java、VB、C 等主流语言



人民邮电出版社
POSTS & TELECOM PRESS

The Definitive Guide to MySQL 5 Third Edition

MySQL 5 权威指南 (第3版)

“本书的权威名副其实，书中几乎包含了 MySQL 使用者所需的所有知识。作者透彻阐述和有条理地选取、组织素材的才能令人敬佩。”

—— Slashdot.com

“除了下一版之外，你再也找不到比本书更好的 MySQL 资源了。”

—— JavaRanch.com

MySQL 具有适用面广、性能优异、运行稳定、性价比高、技术支持丰富且易于获得的优点，是开源领域应用最广泛的数据
库系统。PHP 或 Perl 与 MySQL 相结合的数据库系统解决方案正在被越来越多的网站所采用。

本书是深受全世界技术人员喜爱的 MySQL 权威图书，内容丰富，结合各种主流程序设计语言提供了大量的实例，从基础开
始，循序渐进地带领读者详细分析和实践每一个步骤，使读者从没有多少 SQL 编程和数据库设计经验的新手，迅速成长为数据
库开发和设计领域的专家！



Michael Kofler 在奥地利格拉茨技术大学获得计算机科学博士学位。他写了很多非常成功的计算机
图书，内容涉及 Visual Basic、Linux、Mathematica 和 Maple 等多种程序设计语言和软件。Kofler 还是 *Definitive
Guide to Excel VBA* 第 2 版的作者。

本书相关信息请访问：[图灵网站](http://www.turingbook.com) <http://www.turingbook.com>

读者 / 作者热线：(010)88593802

反馈 / 投稿 / 推荐信箱：contact@turingbook.com

上架建议 计算机 / 数据库 / MySQL

ISBN 7-115-15337-X



9 787115 153371 >

Apress®

ISBN7-115-15337-X/TP · 5728

定价：79.00 元

TURING 图灵程序设计丛书

MySQL 5权威指南

(第3版)

The Definitive Guide to MySQL 5
Third Edition

[奥] Michael Kofler 著
杨晓云 王建桥 杨涛 等译

 人民邮电出版社
POSTS & TELECOM PRESS

图书在版编目 (CIP) 数据

MySQL 5 权威指南: 第 3 版 / (奥) 科夫勒著; 杨晓云等译. —北京: 人民邮电出版社, 2006.12
(图灵程序设计丛书)

ISBN 7-115-15337-X

I. M... II. ①科...②杨... III. 关系数据库—数据库管理系统, MySQL IV. TP311.138
中国版本图书馆 CIP 数据核字 (2006) 第 117729 号

内 容 提 要

本书全面深入地介绍了 MySQL 的功能, 主要内容包括 MySQL、PHP、Apache、Perl 等组件的安装与功能简介, mysql 等一些重要系统管理工具和用户操作界面的使用, MySQL 数据库系统设计的基础知识与用不同语言设计 MySQL 数据库的过程, 以及 SQL 语法、工具、选项、API 应用指南, 最大限度地帮助读者更快地学习和掌握 MySQL 数据库系统的设计和使用。本书覆盖了 MySQL 5.0, 讨论了新的程序设计接口(如 PHP 5 里的 mysqli) 和新的系统管理工具。

本书是 MySQL 数据库管理员和开发人员的必备参考书。

图灵程序设计丛书

MySQL 5 权威指南 (第 3 版)

-
- ◆ 著 [奥] Michael Kofler
 - 译 杨晓云 王建桥 杨 涛 等
 - 责任编辑 傅志红
 - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
 - 北京顺义振华印刷厂印刷
 - 新华书店总店北京发行所经销
 - ◆ 开本: 800×1000 1/16
 - 印张: 43
 - 字数: 1203 千字 2006 年 12 月第 1 版
 - 印数: 1~4 000 册 2006 年 12 月北京第 1 次印刷

著作权合同登记号 图字: 01-2006-1722 号

ISBN 7-115-15337-X/TP · 5728

定价: 79.00 元

读者服务热线: (010) 88593802 印装质量热线: (010) 67129223

版 权 声 明

Original English language edition, entitled *The Definitive Guide to MySQL 5* Third Edition by Michael Kofler, published by Apress L.P., 2560 Ninth Street, Suite 219, Berkeley, CA 94710 USA.

Copyright © 2006 by Michael Kofler. Simplified Chinese-language edition copyright © 2006 by Posts & Telecom Press. All rights reserved.

本书中文简体字版由 Apress L.P. 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

译者序

MySQL 是使用最广泛的开源数据库系统，它正在数据库市场上重演 Linux 在操作系统领域逐步取得成功的故事。PHP 或 Perl 语言与 MySQL 相结合的数据库系统解决方案正被越来越多的网站所采用，其中又以 LAMP 模式（“Linux + Apache + MySQL + Perl 或 PHP”组合方式）最为流行。MySQL 的突出优点包括：

- **适用面广。**可以在 Windows、Linux、Mac OS X 及各种 UNIX 操作系统上运行，可以用 C、C++、C#、Java、Perl、PHP、Python、Visual Basic 和 Visual Basic.NET 等多种程序设计语言来开发 MySQL 应用程序。在 Linux 领域里，以 MySQL 作为后端数据库引擎的应用项目越来越多：它可以帮助人们更有效率地管理各种日志数据以及电子邮件、MP3 文件、地址名单之类的数据。在 Windows 领域里，借助于 ODBC 接口，MySQL 也能完成类似的任务（在许多场合下，MySQL 提供了更好的技术基础）。
- **性能优异，运行稳定。**MySQL 是一种功能非常强大的关系数据库系统，它的安全性和稳定性足以满足许多应用项目的要求。美国航空航天局、美国洛斯·阿拉莫斯国家实验室（数据量高达 7TB）、Yahoo!、Lycos、索尼、铃木、维基百科等大公司和大机构都采用了 MySQL 来建立它们的后端数据库。从这个意义上讲，选择 MySQL 就等于是让自己与这些优秀的公司站在了同一条起跑线上。此外，MySQL 对硬件性能的要求不那么苛刻，这一点对小公司或个人用户来说特别有优势。
- **性价比高。**MySQL 是一个开源软件产品，采用 GPL 许可证发行，所以绝大多数 MySQL 应用项目都可以免费获得和使用 MySQL 软件。如果用户的 MySQL 项目不符合 GPL 许可证的有关规定，只须支付一些合理的费用就可以获得商业许可证和各种可选的技术支持服务合同。
- **技术支持丰富而且易于获得。**因特网上有着丰富的 MySQL 资源。

与其他的开源数据库系统相比，MySQL 不仅在性能指标方面高出一截，在应用范围和实际装机容量方面也远远领先于竞争对手。MySQL 比其他数据库系统接受过更全面的测试，有着更齐全的文档，有 MySQL 经验的开发人员也相对更多一些。不过，MySQL 目前还无法在所有的方面与一些老牌的商业化数据库系统抗衡。万一用户的项目必须用到某些 MySQL 尚不支持的功能，请在项目的前期可行性研究阶段做出判断和取舍。

本书是一部 MySQL 领域的名著，新版在第 2 版的基础上进行了大量的改写。大部分改动是根据 MySQL 软件从 4.1 版到 5.0 版的变化而做出的。围绕着 MySQL 相关领域做出的改动也有不少，其中包括新的程序设计接口（如 PHP 5 里的 mysqli 接口）和新的系统管理工具。本书从面向应用和面向示例两方面对 MySQL 数据库系统进行了全面系统的介绍，读者无须具备 SQL 编程或数据库设计经验，而书中的示例数据库和示例程序能够帮助那些打算自行开发一个数据库应用软件的读者打下坚实的基础。不过，因为本书没有足够的篇幅从入门开始对每一个论题进行探讨，所以书中有些内容（尤其是与编程有关的内容）需要读者具备相应的基础知识才能从中获得最大收益。

2 译者序

与任何一种现代数据库产品一样，MySQL 的实际应用还需要涉及硬件设备、操作系统、网络与通信等许多方面。这些领域每天都有新的技术和改进诞生。

本书主要由杨晓云、王建桥、杨涛、李东梅翻译，参加翻译工作的还有范精明、许玉新、俞渭明、韩兰、张玉亭、李崧、张雁东、李京山、张立和、张勇、韩文光、孙殿刚、韩东升、陈丽芬、张异宝、卫健、李江卫、丁文学、张斌和汪艳华等同志。

因为译者水平有限，所以书中可能会有一些错误和疏漏，希望能够得到读者的谅解和指正。

前　　言

MySQL 是使用最广泛的开源数据库系统，这主要有以下几个原因：

- MySQL 很快。
- MySQL 很稳定。
- MySQL 很容易学习。
- MySQL 可以在各种主流的操作系统（Windows、Linux、Mac OS X 和各种 UNIX 版本）上运行。
- MySQL 应用软件可以用多种程序设计语言（如 C、C++、C#、Java、Perl、PHP、Python、Visual Basic 和 Visual Basic.NET 等）来编写。
- 网上有详尽的 MySQL 文档，与 MySQL 有关的书籍也非常多。
- 有许多应用项目都允许用户免费使用 MySQL 来开发（在遵守 GPL 许可证制度的前提下）。
- 虽然也有许多商业化的应用软件不受 GPL 许可证的约束，但只需支付一些合理的费用就可以获得商业许可证和各种可选的技术支持服务合同。

MySQL 正在数据库市场上重演 Linux 在操作系统领域逐步取得成功的故事。PHP 或 Perl 语言与 MySQL 相结合的数据库系统解决方案被越来越多的网站所采用。其中又以“Linux + Apache + MySQL + Perl 或 PHP”的组合方式最为流行，这种组合被人们称为 LAMP 模式。MySQL 并不仅仅适用于小型网站，连 Yahoo!、Slashdot 和美国航空航天局等数据量非常大的公司和机构也在使用它。

本书内容

本书面向应用和面向示例对 MySQL 数据库系统进行了全面系统的介绍。读者无需具备 SQL 编程或数据库设计的经验。

本书的入门部分将从如何在运行 Windows 和 Linux 操作系统的计算机上安装 MySQL、Apache、PHP 和 Perl 等软件组件的具体步骤开始展开讨论。此外，还将介绍如何安装需要与 MySQL 配合使用的其他组件。在此基础上的第一个示例将向大家演示 MySQL 和 PHP 的基本用法。

本书的第二部分将对 *mysql*、*mysqladmin*、*mysqldump*、MySQL Administrator、MySQL Query Browser 和 phpMyAdmin 等几个最为重要的系统管理工具和用户操作界面进行介绍，最后一个程序特别适合使用 Web 浏览器以离线方式来完成各种系统管理工作的情况。在这一部分还将专门用一章的篇幅来讨论如何通过 Microsoft Office、Sun StarOffice 和 OpenOffice 访问 MySQL 数据库。

本书的第三部分为读者准备了大量有关数据库语言 SQL、数据库系统的设计思路、存储过程、MySQL 的访问控制系统和多种系统管理工作（如备份、日志和镜像等）的背景资料。

本书的第四部分将重点介绍 PHP 语言，其中有一章内容很长，读者可以学到许多程序设计方面的

技巧。将通过一系列示例程序来演示如何利用 *mysql* 和 *mysqli* (PHP 5 里新增加的软件工具) 程序提供的操作界面去完成各项系统管理任务；还将介绍其他几种程序设计语言，将在专门的章节里对 Perl、Java、C、Visual Basic 6 以及 Visual Basic.NET 和 C# 等语言进行讨论。

本书的主要内容将以参考资料篇（第五部分）作为结束，将对 MySQL 所支持的各种 SQL 命令、常见 MySQL 系统管理工具的功能选项及几种重要的程序设计语言（PHP、Perl、C、Java、ADO.NET）所提供的接口函数进行总结。

最后是附录，其内容涉及术语解释（附录 A），对书里提到的各个示例文件的介绍（附录 B，有关文件都可以从 www.apress.com 网站下载），以及帮助大家进一步掌握 MySQL 的参考读物和建议（附录 C）。

我们相信，本书里的示例数据库和示例程序能够帮助那些打算自行开发一个数据库应用软件的读者打下一个坚实的基础。在这里，预祝大家能够从中获得乐趣和成功。

本书（第3版）新增内容

本书在第 2 版的基础上进行了大量的改写。大部分改动反映了 MySQL 服务器从 4.1 版到 5.0 版的变化。围绕着 MySQL 相关领域做出的改动也有不少，其中包括新的程序设计接口（如 PHP 5 里的 *mysqli*）和新的系统管理工具。最为重要的新增内容如下所示。

MySQL 从 4.1 到 5.0 的变化

- 增加对 Unicode 和其他新字符集的支持（新增和改动之处详见书中各有关内容）。
- 视图（在第 8 章中新增了一个小节）。
- *INFORMATION_SCHEMA* 数据表（在第 9 章中新增了一个小节）。
- 子查询（在第 10 章中新增了一个小节）。
- 经过改进的密码验证机制和访问控制管理方面的新增权限（第 11 章）。
- GIS 函数（新增的第 12 章）。
- 存储过程和触发器（新增的第 13 章）。
- 对 InnoDB 数据表进行维护管理的新办法（在第 14 章中新增了一个小节）。
- 各种新的 SQL 命令、函数、数据类型（新增和改动之处详见书中各有关内容，第 21 章对它们进行了汇总）。
- MySQL 服务器和它的各种系统管理工具的新增功能选项（新增和改动之处详见书中各有关内容，第 21 章对它们进行了汇总）。

相关领域中的变化

- MySQL Administrator 和 MySQL Query Browser（新增的第 5 章）。
- phpMyAdmin 工具中的许多新增功能（新增的第 6 章）。
- OpenOffice / StarOffice 中的数据库接口（在第 7 章中新增了一节）。
- PHP 5 里的 *mysqli* 工具，这是一个面向对象的接口（新增的第 15 章、第 23 章对它们进行了

汇总)。

□ C-API 中的新增函数(第 18 章、第 23 章对它们进行了汇总)。

□ 新的 ADO .NET 驱动程序集 Connector/Net(第 20 章、第 23 章对它们进行了汇总)。

本书没有涉及的内容

在本书的各有关章节里，分别讨论了如何运用 PHP、Perl、C、Java 和 Visual Basic 等多种程序设计语言编写 MySQL 应用软件的问题。那些章节里的内容是在读者对相应的程序设计语言已经比较熟悉的假设下展开的——原因很简单：本书没有足够的篇幅从入门开始对那么多种程序设计语言进行介绍。换句话说，读者只有在自己已经熟练掌握(比如 PHP 语言)的前提下才能从专门讨论 PHP 编程技巧的有关章节里获得最大的收益。

示例程序、源代码

书中所有示例程序的源代码都可以通过 www.turingbook.com 和 www.apress.com 网站下载。

在这本书里，读者会在一些比较长的示例程序清单的开头看到一个如下所示的注释行，它给出了该示例文件在上述网站上的文件名，比如：

```
<!-- php/titleform.php -->
```

出于节约篇幅的考虑，在书中有时只给出了整段程序代码中最精彩的片段。

软件版本问题

MySQL 本身以及各种工具程序、程序设计语言和相关函数库的功能会随着它们各自的每一个新版本的出现而发生变化——这些变化每个星期都有可能发生。下面这份清单是笔者在编写这本书时使用的各种软件的版本明细(对这些软件名称的解释详见书中的适当位置)。

Apache: 2.n 版。

Connector/J: 3.1.7 版和 3.2.0 版。

Connector/ODBC(以前叫做 MyODBC)：3.51.11 版。

gcc: 3.3 版。

Java: 1.4.2 版和 1.5.0 版。

Linux: 本书中的 MySQL 工具和其他程序都已经在 Windows 和 Linux 环境下通过了测试。我们的 Linux 测试环境使用的是 Red Hat Enterprise 4 和 SUSE Professional 9.2 和 9.3 beta 发行版本。

Microsoft Office: Office 2000。

Microsoft Visual Basic、VBA、ADO: 书中的 Visual Basic 程序是在 Visual Basic 6、VBA 6 和 ADO 2.8 环境下开发和测试的。

Microsoft Visual Basic .NET、C#、ADO.NET: 与这几个方面有关的信息主要以 Visual Studio 2005 beta 版(即.NET Framework 2.0)为依据。

Microsoft Windows: Windows 环境下的测试工作全都是在 Windows XP SP2 系统上完成的。从理论上讲，有关信息应该同样适用于 Windows 2000 和未来的 Windows 版本。

MySQL: 5.0.3 版。

OpenOffice: 2.0 beta 版。

Perl: 5.8 版。

PHP: 5.0 版和 5.3 版。

phpMyAdmin: 2.6.1 版。

印刷体例

- SQL 命令、SQL 函数以及 SQL、C、Java、PHP、Perl、Visual Basic 等语言里的各种函数方法、类和关键字主要以斜体字表示（例如：SELECT、*mysql_query*）。
- UNIX/Linux 用户名也以斜体字表示（例如：root、*mysql*）。
- 菜单命令使用首字母大写字体表示（例如：File | Open）。
- 文件名和子目录名使用代码体表示（例如：/usr/local 或 C:\Windows）。
- 程序名和编程命令也以代码体表示（例如：mysql 或 cmd.exe）。
- 程序清单和命令行输入也以代码体表示。

此外，书中的 SQL 命令通常以大写字体写出。这不是一项语法规规定，而是一种习惯。MySQL 在解释 SQL 命令的时候并不区分字母的大小写形式。

在需要给出一个 Windows 目录的时候，通常不会给出一个绝对路径，这是因为它将取决于具体的安装情况。我们将采用以下几种表示方法：

Windows\ 代表 Windows 的安装目录（例如：C:\Windows 或 D:\WinNT4）。

Programs\ 代表 Windows 上的软件安装目录（例如：C:\Programs 或 D:\Program Files）。

Mysql\ 代表 MySQL 的安全目录（例如：C:\Program Files\Mysql）。

命令

本书里将会出现很多命令。通常会对比给出有关命令在 UNIX/Linux 和 Windows 两种环境中的不同写法。下面两条命令就是等价的：

```
root# mysqladmin -u root -h localhost password xxx  
> mysqladmin -u root -h localhost password xxx
```

请大家根据不同的系统提示符来区分这些命令的适用场合（root# 表示适用于 UNIX/Linux，> 表示适用于 Windows）。在输入命令时，只需输入系统提示符后面的的文字（即黑体字部分）。在 UNIX/Linux 系统上，比较长的输入内容可能会被分断为几个输入行。这些输入行是用反斜线字符 (\) 分断的，在书中经常会遇到这样的情况。下面这条命令与上面给出的第一条命令等价：

```
root# mysqladmin -u root -h localhost \  
password xxx
```

请注意，在输入有关命令时别忘了把“xxx”替换为具体的文字（具体到上面这个例子，需要把“xxx”

替换为密码）。为了表明“xxx”是一个哑元参数，将在书中把它写为斜体字。

缩略语

笔者已经尽最大可能不在这本书里使用缩略语，但有些早已约定俗成的缩略语还是会不可避免地反复在书中出现。下表中所列的缩略语在各章的讨论内容里将不再进行解释。

缩略语	含 义
ADO	Active Data Object（活动数据对象），微软公司的数据库函数库
ADO.NET	ADO for .NET（.NET 活动数据对象），与 ADO 不兼容
BLOB	Binary Large Object（二进制大对象），二进制的数据块
GIS	Geographical Information System（地理信息系统）
GPL	GNU Public License（GNU 公共许可证），开源软件普遍采用的一种软件许可证制度
HTML	HyperText Markup Language（超文本标记语言），一种用来描述 Web 文档的计算机语言
InnoDB	这不是一个缩略语。InnoDB 是一家公司的名字，该公司为 MySQL 数据库开发了一种特殊的数 据表格式，该格式的名字也叫做 InnoDB
ISP	Internet Service Provider（因特网接入服务提供商）
MySQL	这是一家公司的名字，该公司开发了 MySQL 数据库系统
ODBC	Open Database Connectivity（开放数据库互连标准），它是一组为了让不同厂商推出的数据库产 品能够相互兼容而制定的标准化数据库访问接口（尤其流行于 Windows 环境下）
PHP	PHP Hypertext Preprocessor（PHP 超文本预处理器），一种用来生成和处理 HTML 页面的脚本程 序设计语言
RPM	Red Hat Packet Manager（Red Hat 软件包管理器），Linux 软件包的一种打包格式
SQL	Structured Query Language（结构化查询语言），一种数据库程序设计语言
URL	Uniform Resource Locator（统一资源定位器），即 http://www.company.com/page.html 格式的 因特网地址
VB	Visual Basic，一种程序设计语言
VBA	Visual Basic for Applications，Microsoft Office 软件包内嵌的程序设计语言
VBA.NET	Visual Basic for Applications for .NET，Microsoft Office 软件包内嵌的程序设计语言

目 录

第一部分 入 门

第 1 章 什么是 MySQL	2
1.1 什么是数据库	2
1.1.1 关系、数据库系统、服务器和客户	2
1.1.2 关系数据库系统与面向对象数据库系统	3
1.1.3 数据表、记录、字段、查询、SQL、索引和键	3
1.2 MySQL	4
1.3 MySQL 的不足	6
1.4 MySQL 的版本编号	7
1.4.1 Alpha、Beta、Gamma、Production (Generally Available)	8
1.4.2 按版本编号排列的 MySQL 功能表	8
1.5 MySQL 的许可证	10
1.5.1 GPL 许可证下的权利和义务	10
1.5.2 开源许可证下的 MySQL 软件	10
1.5.3 商用许可证下的 MySQL 软件	11
1.5.4 MySQL 客户软件开发库 (Connector/ODBC、Connector/J 等) 的商用许可证	12
1.5.5 PHP 项目的客户许可证问题——F(L)OSS 特例	12
1.5.6 MySQL 软件的版本名称	13
1.5.7 MySQL 软件的技术支持合同	14
1.6 MySQL 软件的替代品	14
1.7 小结	15
第 2 章 测试环境	16
2.1 是 Windows 还是 UNIX/Linux	16

2.1.1 MySQL 应用现状 (因特网上的数据库服务器)	16
2.1.2 开发环境	17
2.2 在 Windows 系统上安装 MySQL 和相关软件	17
2.2.1 安装 Apache 2.0	18
2.2.2 安装 MySQL 5.0	19
2.2.3 安装 PHP 5.0	22
2.2.4 安装 Perl	23
2.3 在 SUSE Linux 9.3 系统上安装 MySQL 和相关软件	25
2.3.1 安装 Apache 2、PHP 5 和 Perl	25
2.3.2 安装 MySQL 5.0	27
2.4 在 Red Hat Enterprise Linux 4 系统上安装 MySQL 和相关软件	29
2.4.1 安装 Apache 2	29
2.4.2 安装 MySQL 5	30
2.4.3 编译 PHP 5	30
2.4.4 安装 Perl 5.8	33
2.5 编译 MySQL 软件的开发者版本 (Linux)	33
2.5.1 安装 Bitkeeper	33
2.5.2 下载 MySQL 软件的开发者版本	34
2.5.3 编译 MySQL	34
2.5.4 创建用来管理访问权限的 <i>mysql</i> 数据库	34
2.5.5 MySQL 配置文件和 Init-V 脚本	34
2.5.6 启动 MySQL 服务器	35
2.6 配置 Apache	35
2.6.1 配置文件	35
2.6.2 基本设置	36
2.6.3 对不同子目录的访问权限 (.htaccess)	37

2.7 配置 PHP.....	39
2.8 配置 MySQL.....	41
第 3 章 初级案例研究：MySQL+PHP.....	43
3.1 概述.....	43
3.2 数据库的开发.....	44
3.2.1 启动 mysql 命令行解释器	44
3.2.2 创建数据库.....	45
3.2.3 创建数据表.....	46
3.2.4 为什么要避简就难	47
3.3 调查问卷.....	48
3.4 问卷调查结果的处理和显示.....	49
3.4.1 mysql 界面与 mysqli 界面	49
3.4.2 建立与数据库的连接	49
3.4.3 对数据进行处理并把它存入 数据库.....	49
3.4.4 显示问卷调查的结果	50
3.4.5 程序代码 (results.php)	51
3.4.6 最终生成的 HTML 代码	52
3.5 改进意见.....	52
第二部分 管理工具和用户操作界面	
第 4 章 mysql、mysqladmin 和 mysqldump	56
4.1 mysql.....	56
4.1.1 启动 mysql	57
4.1.2 mysql 的命令行选项	58
4.1.3 交互式使用 mysql	59
4.1.4 UNIX/Linux 环境中 mysql 的使用 技巧	60
4.1.5 Windows 环境下 mysql 的使用 技巧	61
4.1.6 用 mysql 处理 SQL 文件	63
4.2 mysqladmin.....	64
4.3 mysqldump.....	64
第 5 章 MySQL Administrator 和 MySQL Query Browser.....	65
5.1 安装.....	65
5.2 与 MySQL 服务器建立连接	66
5.3 MySQL Administrator.....	67
5.3.1 Server Information 模块 (查看 服务器信息)	68
5.3.2 Service Control 模块 (启动/停止 MySQL 服务器)	68
5.3.3 Startup Variables 模块 (配置启动 参数)	68
5.3.4 User Administration 模块 (用户 管理)	69
5.3.5 Server Connections 模块 (查看 服务器连接信息)	71
5.3.6 Health 模块 (查看系统负载)	71
5.3.7 Server Logs 模块 (查看服务器 日志)	72
5.3.8 Backup 模块 (制作数据库备份)	72
5.3.9 Restore 模块 (用备份恢复数 据库)	73
5.3.10 Replication Status 模块 (查看镜像 机制的工作状态)	74
5.3.11 Catalogs 模块 (对数据库和数据 表进行管理)	74
5.4 MySQL Query Browser	74
5.4.1 SQL 命令的输入和执行	74
5.4.2 对 SELECT 结果里的数据进行 修改	76
5.4.3 SQL 命令的历史记录和书签	77
5.4.4 一次执行多条命令 (脚本)	77
5.4.5 存储过程	77
5.4.6 MySQL Help (帮助文档)	77
第 6 章 phpMyAdmin	78
6.1 phpMyAdmin 的安装与配置	79
6.1.1 安装 phpMyAdmin 文件	79
6.1.2 配置 phpMyAdmin	79
6.1.3 config 身份验证模式	80
6.1.4 http 和 cookie 身份验证模式	82
6.2 用户管理, 保护 MySQL	84
6.2.1 保护 MySQL	85
6.2.2 创建新用户	86
6.3 创建和编辑数据库	88
6.3.1 创建数据库	88

6.3.2 编辑现有的数据表.....	89	7.6.1 建立数据源.....	118	
6.3.3 设置外键规则.....	90	7.6.2 数据的导入.....	118	
6.3.4 数据库设计方案的汇总和存档.....	90	第三部分 基 础 知 识		
6.4 查看、插入和编辑数据.....	91	第 8 章 数据库设计概论 120		
6.5 执行 SQL 命令.....	92	8.1 参考读物	120	
6.6 导入和导出	93	8.2 数据表类型	121	
6.6.1 数据库备份 (SQL 文件)	93	8.2.1 MyISAM 数据表	121	
6.6.2 导出数据表 (CSV 文本文件)	95	8.2.2 InnoDB 数据表	122	
6.6.3 导入数据库或数据表 (SQL 文件)	95	8.2.3 HEAP 数据表	123	
6.6.4 插入数据表数据 (文本文件)	96	8.2.4 临时数据表	124	
6.7 服务器管理	96	8.2.5 其他的数据表类型	124	
6.8 辅助功能	97	8.2.6 数据表文件	125	
6.8.1 为 phpMyAdmin 创建数据库	97	8.3 MySQL 数据类型	126	
6.8.2 SQL 书签和历史记录	98	8.3.1 整数 (xxxINT)	126	
6.8.3 关联/引用关系信息的保存	98	8.3.2 定点数 (DECIMAL)	127	
6.8.4 创建 PDF 格式的数据表关联/ 引用关系图	100	8.3.3 日期与时间 (DATE、TIME、 DATETIME、TIMESTAMP)	128	
6.8.5 格式转换 (数据列内容的另类 显示效果)	101	8.3.4 字符串 (CHAR、VARCHAR、 xxxTEXT)	130	
第 7 章 Microsoft Office 和 OpenOffice/ StarOffice	103	8.3.5 二进制数据 (xxxBLOB 和 BIT)	133	
7.1 安装 Connector/ODBC	103	8.3.6 选项和属性	135	
7.2 Microsoft Access	106	8.4 数据库设计技巧	135	
7.2.1 数据表的导入和导出	107	8.4.1 数据库设计要求	135	
7.2.2 数据库转换器: Access→MySQL (exportsql.txt)	109	8.4.2 起名字的技巧	136	
7.3 Microsoft Excel	110	8.4.3 数据库具体设计工作中的技巧	136	
7.4 安装 Connector/J	112	8.5 规范化	137	
7.4.1 Connector/J	112	8.5.1 起点	137	
7.4.2 安装	112	8.5.2 第一范式	138	
7.5 OpenOffice/StarOffice Base	113	8.5.3 第二范式	139	
7.5.1 与 MySQL 数据库建立连接	113	8.5.4 第三范式	140	
7.5.2 Table 模块	114	8.5.5 规范化理论	141	
7.5.3 Queries 模块	115	8.6 层次关系的处理	143	
7.5.4 Forms 模块、Reports 模块和 其他功能	116	8.6.1 层次关系的处理难点	144	
7.6 OpenOffice/StarOffice 的 Data Source 视图	117	8.6.2 从数据表创建层次关系树	145	
		8.6.3 搜索 <i>categories</i> 数据表里的下级 图书门类	147	
		8.6.4 搜索 <i>categories</i> 数据表里的上级 图书门类	148	

8.7	关系	149	9.2.4	限制查询结果中的数据记录个数 (<i>LIMIT</i>)	173
8.7.1	1:1 关系	149	9.2.5	在使用 <i>LIMIT</i> 关键字确定数据表里的数据记录数 (<i>SQL_CALC_FOUND_ROWS</i> 、 <i>FOUND_ROWS()</i>)	173
8.7.2	1: <i>n</i> 关系	150	9.3	对查询结果进行排序 (<i>ORDER BY</i>)	174
8.7.3	<i>n:m</i> 关系	151	9.3.1	选择一种排序方式	174
8.8	主键和外键	151	9.3.2	试用不同的排序方式	175
8.8.1	主键	151	9.4	筛选数据记录 (<i>WHERE</i> , <i>HAVING</i>)	176
8.8.2	外键	152	9.5	涉及多个数据表的关联查询 (<i>LEFT/RIGHT JOIN</i>)	177
8.8.3	引用一致性 (外键约束条件)	153	9.5.1	两个数据表的关联	178
8.9	索引	156	9.5.2	3 个或更多个数据表的关联	179
8.9.1	普通索引、唯一索引和主索引	157	9.6	合并查询结果 (<i>UNION</i>)	181
8.9.2	全文索引	158	9.7	分组查询, 统计函数 (<i>GROUP BY</i>)	181
8.9.3	查询和索引的优化	159	9.7.1	统计函数	182
8.10	视图	161	9.7.2	统计函数 <i>GROUP_CONCAT()</i>	183
8.11	示例数据库 <i>mylibrary</i> (图书管理)	163	9.7.3	对多个数据列进行 <i>GROUP BY</i> 查询	184
8.11.1	数据库的属性	164	9.7.4	<i>GROUP BY...WITH ROLLUP</i>	184
8.11.2	数据表的属性	164	9.8	修改数据 (<i>INSERT</i> 、 <i>UPDATE</i> 和 <i>DELETE</i>)	185
8.12	示例数据库 <i>myforum</i> (网上论坛)	165	9.8.1	备份数据	185
8.12.1	讨论组数据库: <i>myforum</i>	165	9.8.2	插入数据记录 (<i>INSERT</i>)	186
8.12.2	帖子之间的层次关系	166	9.8.3	修改数据记录 (<i>UPDATE</i>)	188
8.13	示例数据库 <i>exceptions</i> (用于特殊情况的测试)	167	9.8.4	删除数据记录 (<i>DELETE</i>)	188
8.13.1	数据表 <i>testall</i>	167	9.9	创建数据表、数据库和索引	190
8.13.2	数据表 <i>text_text</i>	168	9.9.1	创建数据库 (<i>CREATE DATABASE</i>)	190
8.13.3	数据表 <i>test_blob</i>	168	9.9.2	创建数据表 (<i>CREATE TABLE</i>)	191
8.13.4	数据表 <i>test_date</i>	168	9.9.3	创建索引 (<i>CREATE INDEX</i>)	192
8.13.5	数据表 <i>test_enum</i>	168	9.9.4	变更数据表的结构 (<i>ALTER TABLE</i>)	192
8.13.6	数据表 <i>test_null</i>	168	9.9.5	删除数据库和数据表 (<i>DROP</i>)	193
8.13.7	数据表 <i>test_sort1</i>	168	9.9.6	自动修改数据表设计 (默许的数据列修改)	193
8.13.8	数据表 <i>test_sort2</i>	169	9.9.7	<i>SHOW</i> 命令	194
8.13.9	数据表 <i>importtable1</i> 、 <i>importtable2</i> 、 <i>exporttable</i>	169	9.9.8	<i>INFORMATION_SCHEMA</i> 数据表	
第 9 章	SQL 语言入门	170			
9.1	简介	170			
9.2	简单查询 (<i>SELECT</i>)	171			
9.2.1	确定数据表里有多少条数据记录 (数据行)	172			
9.2.2	确定数据表里有多少条内容不重复的数据记录 (<i>DISTINCT</i>)	172			
9.2.3	限制查询结果中的数据列个数	172			

家族	195
第 10 章 SQL 解决方案	197
10.1 字符串	197
10.1.1 基本函数	197
10.1.2 改变字符集	198
10.1.3 设置客户端字符集	199
10.1.4 模板匹配	200
10.2 日期和时间	201
10.2.1 日期和时间的语法	201
10.2.2 与日期和时间有关的计算	202
10.2.3 UNIX 时间戳	204
10.2.4 地理时区	206
10.3 ENUM 和 SET 数据类型	208
10.3.1 ENUM	208
10.3.2 SET	209
10.4 变量与条件表达式 (IF、CASE)	209
10.4.1 变量	210
10.4.2 IF 查询	211
10.4.3 CASE 分支	211
10.5 在数据表间复制数据	212
10.5.1 利用复制操作创建新数据表	212
10.5.2 把数据复制到现有数据表	213
10.6 统计报表	213
10.6.1 涉及 titles、languages 和 categories 数据表的统计报表	214
10.6.2 月度查询统计报表	215
10.7 子查询	216
10.7.1 语法变体	216
10.7.2 示例	218
10.8 保证数据的一致性	219
10.8.1 找出没有作者的图书	219
10.8.2 找出无效的出版公司引用: 1:n 关系中的无效记录	219
10.8.3 找出作者与图书之间的无效 链接 (n:m 关系)	220
10.9 找出冗余的数据记录	221
10.10 数据表设计方案的改进	221
10.11 对前 n 条或后 n 条记录进行处理	223
10.11.1 数据查询 (SELECT)	223
10.11.2 修改现有记录 (UPDATE 和 DELETE)	224
10.11.3 把全部讨论线程删除到只 剩下最新的 500 个线程	224
10.12 以随机方式选择数据记录	225
10.12.1 通用方法: RAND() 函数	225
10.12.2 自备随机数的数据表	225
10.12.3 利用 id 数据列选择随机 记录	226
10.13 全文索引	226
10.13.1 基础知识	227
10.13.2 图书检索	229
10.13.3 论坛文章检索	231
10.14 锁定	232
10.14.1 语法	232
10.14.2 GET_LOCK 和 RELEASE_ LOCK 函数	233
10.15 事务	233
10.15.1 为什么要使用事务	233
10.15.2 事务的控制	234
10.15.3 事务机制的工作流程	235
10.15.4 事务与锁定	237
10.15.5 事务的隔离模式	238
10.15.6 出错处理	240
第 11 章 访问权限与信息安全	241
11.1 简介	241
11.1.1 客户与 MySQL 服务器之间的 连接	241
11.1.2 访问管理	242
11.2 急救	246
11.2.1 保护 MySQL 安装	246
11.2.2 创建新的数据库和用户	247
11.2.3 授予创建个人数据库的权限	248
11.2.4 忘记 root 密码情况的处理	249
11.3 访问控制机制的内部工作原理	250
11.3.1 两级访问控制	250
11.3.2 权限	250
11.3.3 mysql 数据库	253
11.3.4 user 数据表	254

11.3.5 <i>user.Host</i> 数据列	257	12.3 SQL 示例（冰川数据库）	281
11.3.6 <i>db</i> 数据表和 <i>host</i> 数据表	260	12.3.1 创建数据表	282
11.3.7 <i>tables_priv</i> 和 <i>columns_priv</i> 数据表	261	12.3.2 插入数据	282
11.3.8 <i>procs_priv</i> 数据表	263	12.3.3 查询数据	282
11.4 访问权限的设置工具	263	12.4 SQL 示例（ <i>opengeodb</i> 数据库）	284
11.4.1 使用 <i>GRANT</i> 和 <i>REVOKE</i> 命令改变访问权限	263	12.4.1 数据来源和导入	285
11.4.2 使用 <i>SHOW GRANT</i> 命令查看访问权限	265	12.4.2 导入	285
11.4.3 使用 <i>mysqladmin</i> 程序改变密码	265	12.4.3 对圆形地理区域进行搜索	286
11.5 MySQL 4.1 版本开始的安全密码验证	265	第 13 章 存储过程和触发器	288
11.5.1 升级客户端函数库	266	13.1 为什么要使用存储过程和触发器	288
11.5.2 <i>old-passwords</i> 模式	266	13.1.1 存储过程	288
11.5.3 同时使用旧的和新的密码的操作	266	13.1.2 触发器	289
11.6 建立连接的问题	267	13.2 初识 SP	289
11.6.1 连接困难的可能原因	267	13.2.1 MySQL 命令解释器：mysql 程序	290
11.6.2 错误检查的更多方法	270	13.2.2 MySQL Query Browser	291
11.7 系统安全性	270	13.3 SP 的实现	292
11.7.1 系统级安全措施	270	13.4 SP 的管理	293
11.7.2 重要数据存储的安全保护	271	13.4.1 创建、编辑和删除 SP	293
11.7.3 与 MySQL 服务器有关的安全风险	271	13.4.2 信息安全问题	295
11.7.4 不要使用 <i>root</i> 或 Administrator 权限来运行 MySQL 服务器	271	13.4.3 SP 的备份和恢复	295
11.7.5 网络安全与防火墙	272	13.5 SP 的语法和语言元素	296
第 12 章 GIS 函数	273	13.5.1 基本语法规则	296
12.1 GIS 数据格式	273	13.5.2 调用 SP (<i>CALL</i>)	297
12.1.1 地理坐标的表示方法	273	13.5.3 参数和返回值	298
12.1.2 Well-Known Text 和 Well-Known Binary (OpenGIS)	274	13.5.4 命令的封装 (<i>BEGIN-END</i>)	299
12.2 MySQL 的 GIS 实现	275	13.5.5 分支	301
12.2.1 数据类型	275	13.5.6 循环	301
12.2.2 简单的几何函数	277	13.5.7 出错处理 (出错处理句柄)	302
12.2.3 空间分析函数	279	13.5.8 光标	304
12.2.4 为几何数据创建索引	281	13.6 SP 应用示例	306
		13.6.1 增加新的图书门类	306
		13.6.2 增加一本新图书	307
		13.6.3 确定父门类	308
		13.6.4 按层次结构生成图书门类清单	309
		13.7 触发器	311
		13.7.1 创建触发器	311
		13.7.2 删除触发器	312

13.7.3 实现细节和管理工具	312	14.5.3 变更日志 (update log)	338
13.7.4 功能局限性	312	14.5.4 出错日志、登录日志和慢查询 日志	340
13.7.5 触发器应用示例	313	14.5.5 日志文件的管理	342
第 14 章 管理与服务器配置	314	14.6 镜像机制	342
14.1 基础知识	314	14.6.1 简介	342
14.1.1 MySQL 数据库系统的管理 工具	314	14.6.2 建立镜像机制的主控系统	344
14.1.2 设置 <i>root</i> 密码	315	14.6.3 建立镜像机制的从属系统	346
14.1.3 MySQL 服务器配置文件	316	14.6.4 用 <i>LOAD DATA</i> 命令建立镜像 机制	348
14.1.4 重新启动 MySQL 服务器	316	14.6.5 内部镜像机制	348
14.1.5 MySQL 服务器的基本配置	317	14.6.6 客户端编程	350
14.1.6 目录	317	14.7 管理 MyISAM 数据表	351
14.1.7 通信设置	317	14.7.1 <i>myisamchk</i> 程序	351
14.1.8 默认的数据表格式	318	14.7.2 <i>myisamchk</i> 程序的使用方法	352
14.1.9 新数据表的默认字符集和排序 方式	318	14.7.3 速度优化与内存使用情况	352
14.1.10 地理时区	318	14.7.4 对 MyISAM 数据表进行压缩 和优化	353
14.1.11 出错消息的显示语言	319	14.7.5 修复受损的 MyISAM 数据表	353
14.1.12 SQL 模式	319	14.7.6 压缩 MyISAM 数据表 (<i>myisampack</i> 程序)	354
14.2 备份	321	14.8 InnoDB 数据表的管理	354
14.2.1 备份数据库 (<i>mysqldump</i>)	321	14.8.1 表空间的管理	354
14.2.2 用备份恢复数据库 (<i>mysql</i>)	324	14.8.2 日志文件	359
14.2.3 快速备份 (<i>mysqlhotcopy</i>)	325	14.9 MySQL 服务器的优化	362
14.3 数据库的迁移	327	14.9.1 优化内存管理	363
14.4 导出和导入文本文件	329	14.9.2 查询缓存区	364
14.4.1 文本文件里的特殊字符	329	14.10 ISP 数据库管理	366
14.4.2 字符串、数值、日期/时间、 BLOB 和 NULL 值	330	14.10.1 <i>ssh</i> 工具	366
14.4.3 用 <i>LOAD DATA INFILE</i> 命令 导入	330	14.10.2 <i>phpMyAdmin</i> 工具	366
14.4.4 用 <i>mysqlimport</i> 工具导入	333	14.10.3 实现自定义的 PHP 脚本	366
14.4.5 用 <i>SELECT ... INTO OUTFILE</i> 命令导出	333	14.10.4 自定义: Perl 脚本	367
14.4.6 用 <i>mysqldump</i> 程序导出	335		
14.4.7 用 <i>mysql</i> 程序的批处理模式 导出	335		
14.5 日志	337		
14.5.1 为什么要使用日志	337		
14.5.2 缺点与不足	338		
		第四部分 程序设计	
第 15 章 PHP	370		
15.1 mysql 功能模块	370		
15.1.1 连接 MySQL 服务器	371		
15.1.2 执行 SQL 命令	372		
15.1.3 处理 <i>SELECT</i> 查询结果	373		

15.1.4 事务	376	15.8.5 指向其他结果页面的链接	410
15.1.5 出错处理与查找	376	15.9 处理层次化数据	410
15.2 <i>mysqli</i> 的类、方法和属性	377	15.9.1 代码结构	411
15.2.1 选择编程接口： <i>mysql</i> 还是 <i>mysqli</i>	377	15.9.2 显示图书门类树	413
15.2.2 有效性测试	378	15.9.3 插入一个或多个新图书门类	415
15.2.3 构成 <i>mysqli</i> 接口的类	378	15.9.4 删除一个图书门类及其下级 门类	416
15.2.4 连接 MySQL 服务器	378	15.9.5 搜索上级图书门类	417
15.2.5 执行 SQL 命令	380	15.9.6 搜索下级图书门类	418
15.2.6 处理 <i>SELECT</i> 查询结果 (<i>mysqli_</i> <i>result()</i> 方法)	380	15.10 速度优化	419
15.2.7 一次执行多条 SQL 命令	382	15.10.1 提高代码执行效率的基本 原则	419
15.2.8 带参数的 SQL 命令（预处理 语句）	382	15.10.2 统计信息和性能指标	420
15.2.9 事务	385	15.10.3 示例：高效地生成图书门类 下拉列表	421
15.3 把数据库功能打包为一个类	385	15.11 Unicode	424
15.3.1 使用单独的密码文件提高 安全性	385	15.12 二进制数据（BLOB）和图像	428
15.3.2 使用 <i>MyDb</i> 类实现安全和 方便	386	15.12.1 在数据库里存储图像的基础 知识与编程技巧	429
15.4 把 <i>SELECT</i> 查询结果显示为一个表格	389	15.12.2 在数据库里存储图像的程序 代码	431
15.5 字符串、日期、时间、BLOB 和 <i>NULL</i>	390	15.13 存储过程	434
15.5.1 字符串和 BLOB	390	15.14 SP Administrator	435
15.5.2 日期和时间	391	15.14.1 安装 SP Administrator	435
15.5.3 <i>NULL</i> 值	392	15.14.2 使用 SP Administrator	435
15.6 向关联数据表插入新数据记录	393	15.14.3 SP Administrator 代码	436
15.7 处理来自 HTML 表单的输入数据	393	第 16 章 Perl	441
15.7.1 代码结构	394	16.1 编程技巧	441
15.7.2 创建 HTML 表单	396	16.1.1 <i>DBI</i> 和 <i>DBD::mysql</i> 模块	441
15.7.3 对表单数据进行合法性检查	400	16.1.2 与数据库建立连接	442
15.7.4 把表单数据存入数据库	401	16.1.3 执行 SQL 命令	443
15.7.5 删 除一本图书	403	16.1.4 处理 <i>SELECT</i> 查询结果	445
15.7.6 值得改进的地方	404	16.1.5 字符串、BLOB、日期值、 <i>SET</i> 、 <i>ENUM</i> 和 <i>NULL</i>	449
15.8 分页显示查询结果	404	16.1.6 <i>DBD::mysql</i> 模块特有的方法 和属性	452
15.8.1 代码结构	405	16.1.7 Unicode	453
15.8.2 对图书作者进行搜索	407	16.1.8 事务	454
15.8.3 对图书进行搜索	408	16.1.9 出错处理	454
15.8.4 显示搜索结果	408		

16.2 示例：删除无效的数据记录 (<i>mylibrary</i>)	456	18.3.1 处理 MySQL 配置文件 <i>my.cnf</i>	488
16.3 CGI 示例：图书管理 (<i>mylibrary</i>)	457	18.3.2 处理命令行选项	488
16.3.1 图书检索 (<i>mylibrary-find.pl</i> 脚本)	457	18.4 执行 SQL 命令	489
16.3.2 新图书的简单输入 (<i>mylibrary- simpleinput.pl</i> 脚本)	460	18.4.1 简单的 SQL 命令	489
16.4 CGI Unicode 示例	462	18.4.2 一次执行多条 SQL 命令	490
16.4.1 图书检索 (<i>mylibrary-find- utf8.pl</i> 脚本)	462	18.4.3 预处理语句	493
16.4.2 新图书的输入 (<i>mylibrary- simpleinput-utf8.pl</i> 脚本)	463	18.4.4 字符集设置 (Unicode)	497
第 17 章 Java (JDBC 和 Connector/J)	464	18.5 处理二进制数据和特殊字符	498
17.1 基础知识	464	18.6 出错处理	500
17.1.1 Java 的安装	464	第 19 章 Visual Basic 6/VBA	501
17.1.2 Connector/J 的安装	466	19.1 基础知识和术语	501
17.2 程序设计技巧	468	19.2 Connector/ODBC 选项	503
17.2.1 第一个示例	469	19.3 ADO 程序设计与 Visual Basic 6/VBA	504
17.2.2 与 MySQL 服务器建立连接	470	19.4 与 MySQL 服务器建立连接	506
17.2.3 连接 MySQL 服务器时可能 遇到的问题	472	19.4.1 与 MySQL 服务器建立连接： 使用 DSN	506
17.2.4 执行 SQL 命令	473	19.4.2 与 MySQL 服务器建立连接 (不使用 DSN)	506
17.2.5 处理 SELECT 查询结果	474	19.4.3 ADO 程序设计技巧	509
17.2.6 预处理语句	478	19.4.4 示例：给 <i>titles</i> 数据表增加 一个 <i>authors</i> 数据列	514
17.2.7 事务	479	19.4.5 示例：添加一本新图书	517
17.2.8 批处理命令	479	19.4.6 示例：把图像文件存入和读出 BLOB 数据列	519
17.2.9 二进制数据 (BLOB) 的 处理	480	19.5 转换器：从 Microsoft SQL Server 到 MySQL	521
第 18 章 C 语言	483	19.5.1 <i>mssql2mysql</i> 脚本的特点	521
18.1 MySQL C API (<i>libmysqlclient</i>)	483	19.5.2 对系统的要求	521
18.2 Hello, World	483	19.5.3 缺陷与不足	521
18.2.1 对系统的基本要求	483	19.5.4 使用方法	522
18.2.2 入门级示例	484	19.5.5 设置有关参数	522
18.2.3 编译与链接	485	19.6 <i>VBMYSQLDirect</i>	523
18.2.4 Makefile	486	19.6.1 安装	524
18.2.5 以静态方式绑定 MySQL API 函数	486	19.6.2 应用	524
18.3 与 MySQL 服务器建立连接	487	19.6.3 示例	524
第 20 章 Visual Basic .NET 和 C#	526		
20.1 ADO .NET 与 MySQL 之间的通信	526		

20.1.1 通过 Connector/Net 连接数 据库.....	527	21.2 操作符.....	552
20.1.2 用 ODBC 数据泵连接 数据库.....	530	21.2.1 算术操作符与位操作符.....	553
20.2 编程技巧.....	532	21.2.2 比较操作符.....	553
20.2.1 执行 SQL 命令 (<i>MySql- Command</i> 对象)	532	21.2.3 使用 <i>LIKE</i> 操作符进行模式 匹配.....	553
20.2.2 带参数的 SQL 命令 (<i>MySql- Parameter</i> 对象)	533	21.2.4 使用 <i>REGEXP</i> 操作符进行模式 匹配.....	554
20.2.3 处理离散的 <i>SELECT</i> 查询结果 (<i>ExecuteScalar()</i> 方法)	535	21.2.5 二进制字符串比较.....	554
20.2.4 读取 <i>SELECT</i> 查询结果 (<i>MySqlDataReader</i> 对象)	535	21.2.6 逻辑操作符.....	555
20.2.5 <i>DataSet</i> 、 <i>DataTable</i> 和 <i>MySqlDataAdapter</i> 对象.....	537	21.3 变量和常数.....	555
20.2.6 辅助函数.....	540	21.3.1 变量赋值.....	555
20.2.7 出错处理.....	540	21.3.2 使用和查看变量.....	555
20.2.8 Windows.Form 和 ASP .NET 控件.....	540	21.3.3 全局级系统变量与会话级系统 变量.....	556
20.2.9 事务.....	542	21.4 MySQL 数据类型.....	557
20.3 小例：把新图书记录存入 <i>mylibrary</i> 数据库.....	542	21.5 SQL 命令汇总表（按功能分类）.....	559
20.4 小例：把图像文件存入和读出一个 <i>BLOB</i> 数据列.....	544	21.6 SQL 命令指南（按字母表顺序 排列）.....	561
第五部分 参 资 料			
第 21 章 SQL 语法指南	548	21.7 SQL 函数指南	596
21.1 语法.....	548	21.7.1 算术函数.....	596
21.1.1 对象命名规则.....	548	21.7.2 比较函数、测试函数、分支 函数.....	597
21.1.2 区分字母大小写.....	549	21.7.3 类型转换（投射）.....	597
21.1.3 字符串.....	549	21.7.4 字符串处理.....	597
21.1.4 字符集和排序方式.....	550	21.7.5 日期/时间函数.....	600
21.1.5 数值.....	550	21.7.6 <i>GROUP BY</i> 函数	604
21.1.6 数值和字符串的自动转换.....	550	21.7.7 其他函数.....	605
21.1.7 日期和时间.....	551	21.8 GIS 数据类型与 GIS 函数	606
21.1.8 二进制数据.....	551	21.9 与存储过程和触发器有关的语言 元素	608
21.1.9 二进制数值.....	551		
21.1.10 注释语句.....	551		
21.1.11 SQL 命令末尾的分号	552		
第 22 章 MySQL 工具和选项	610		
22.1 概述.....	610		
22.2 通用选项和配置文件	610		
22.2.1 通用选项	610		
22.2.2 设置配置文件的选项	612		
22.2.3 内存量的表示方法	613		

22.2.4 环境变量（系统变量）	613	23.3 Perl DBI	638
22.2.5 选项设置规则	614	23.3.1 常用的变量名	638
22.3 mysqld 程序（服务器）	614	23.3.2 与 MySQL 服务器建立连接	639
22.3.1 基本选项	615	23.3.3 执行 SQL 命令、处理 SELECT 查询结果	640
22.3.2 与日志和镜像功能有关的 选项	617	23.3.4 出错处理	642
22.3.3 InnoDB 配置选项	618	23.3.5 辅助函数	642
22.3.4 其他选项	620	23.3.6 DBD::mysql 驱动程序中的 MySQL 专用扩展模块	642
22.4 mysqld_safe 脚本（启动 MySQL 服务器）	620	23.4 JDBC (Connector/J)	644
22.5 mysql_install_db 脚本（安装 mysql 数据库）	621	23.4.1 与 MySQL 服务器建立连接	644
22.6 mysql_fix_privileges 脚本（更新 mysql 数据库）	622	23.4.2 执行 SQL 命令	644
22.7 mysql_fix_extensions 脚本 (重命名 MyISAM 文件)	622	23.4.3 处理 SELECT 查询结果 (ResultSet 类)	646
22.8 mysql 程序（SQL 命令解释器）	622	23.4.4 事务	647
22.9 mysqladmin 程序（日常管理）	624	23.5 ADO.NET (Connector/Net)	647
22.10 mysqldump 程序（数据的备份/ 导出）	625	23.5.1 与 MySQL 服务器建立连接	647
22.11 mysqlimport 程序（文本导入、批量 导入）	628	23.5.2 执行 SQL 命令与处理 SELECT 查询结果	648
22.12 mysqlshow 程序（查看信息）	628	23.5.3 利用 DataSet/DataTable 类修改 数据	649
22.13 myisamchk 程序（修复 MyISAM 文件）	629	23.5.4 事务	650
22.14 myisampack 程序（压缩 MyISAM 文件）	630	23.6 C API	650
第 23 章 MySQL API 应用指南	632	23.6.1 数据结构	650
23.1 PHP API (mysql 接口)	632	23.6.2 连接与管理	652
23.2 PHP API (mysqli 接口)	635	23.6.3 执行 SQL 命令及处理 SELECT 查询结果	654
23.2.1 mysqli 类	636	23.6.4 预处理语句	656
23.2.2 mysqli_result 类	637		
23.2.3 mysqli_stmt 类	638		
		第六部分 附录	
附录 A 术语解释		658	
附录 B 本书的配套示例文件		663	
附录 C 参考书目		664	

Part 1

第一部分

入门

■ 本部分内容

- 第1章 什么是MySQL
- 第2章 测试环境
- 第3章 初级案例研究：MySQL+PHP

第1章

什么是 MySQL

本章将先向大家概括地介绍几个在数据库领域里最为重要的基本概念，然后再介绍 MySQL 都有哪些优点和不足，什么是 MySQL，它能做什么，以及它不能做什么。

在对 MySQL 的核心功能做过简要描述之后，还将对 MySQL 软件的许可证问题做全面系统的介绍，包括在哪些场合可以免费使用 MySQL，在哪些场合必须事先获得相应的许可证。

1.1 什么是数据库

在回答本章的核心问题——什么是 MySQL 之前，必须先找到一种共同语言。为此，在这一小节里先对数据库领域里的部分术语做一个简要的、不过分涉及技术细节的解释。如果读者已经与关系数据库打过多年交道，可以跳过接下来的几页。

很难找到比“数据库”这个词的含义更不精确的术语了。数据库可以是某个电子表格程序（如 Excel）里的一份地址清单，可以是某个电信公司用来记录每天数百万次电话接听情况、精确计算出来的单次通话收费、月结话费账单、欠费警告信等信息的日志文件。一个简单的数据库可以是一种单机操作，有关数据只驻留在一台特定的本地计算机里供单个用户使用；而一个复杂的数据库可能有几千位用户在同时使用，有关数据散布在多台计算机和几十个硬盘上。一个数据库可以小到只有几千个字节，也可以大到需要以 TB（1TB=1024GB, 1GB=1024MB, 1MB=1024KB）作为计量单位。

在日常工作中，人们还经常使用“数据库”这个词来称呼各种实实在在的数据、最终的数据库文件、各种数据库系统（如 MySQL 或 Oracle）或某种数据库客户软件（如某个 PHP 脚本或是某个用 C++ 语言编写的程序）。这种多义性经常导致这样一种局面：只要两个人开始谈论数据库方面的话题，就很容易因为彼此对“数据库”这个概念有着不同的理解而产生误会。

1.1.1 关系、数据库系统、服务器和客户

数据库（database）就是一个由一批数据构成的有序集合，这个集合通常被保存为一个或多个彼此相关的文件。这些数据被分门别类地存放在一些结构化的数据表（table）里，而数据表之间又往往会产生种种内在的交叉引用关系。存在于数据表之间的这种关系（relation）使数据库又被称为关系（型）数据库。

为了把事情说得更明白，下面来看一个例子。假设某公司有一个由 3 个数据表构成的数据库，第 1 个数据表保存着该公司的顾客名单数据（姓名、地址等），第 2 个数据表保存着该公司的产品数据，第 3 个数据表保存着该公司的订单数据。在这 3 个数据表之间建立起必要的关系之后，这家公司的员工就可以通过订单数据表去访问另外两个数据表里的数据了（如通过顾客编号和产品代码）。

MySQL、Oracle、Microsoft SQL Server 和 IBM DB2 都是关系(型)数据库系统(relational database system)。除了数据，一个这样的系统还包括用来管理各种关系数据库的程序。一个合格的关系数据库系统不仅要确保各种数据的存储情况安全可靠，还必须能够处理对现有数据进行查询、分析和排序以及对新数据进行保存等诸多命令。而所有这一切不仅会发生在一台独立运行的计算机上，还有可能发生在网络上。因此，在谈论数据库系统的时候，就不得不经常提到数据库服务器(database server)。

有服务器的地方就应该有客户。连接到数据库系统的每一个程序都可以被称为一个数据库客户(database client)。数据库客户的工作是帮助最终用户简便易行地用好数据库。数据库系统的用户没有一个愿意与数据库服务器直接打交道，因为那么做既过于抽象，又过于麻烦(与数据库服务器直接进行交流的事还是交给程序员们去操心好了)。相反，用户普遍期望——他们也有这个权利——能够通过简便易用的数据表和列表清单框等迅速查找到自己想要的数据或是输入新的数据。

数据库客户的类型各种各样，事实上，有不少用户或许从未意识到自己经常使用的某个程序是一个数据库程序，更不用说什么数据库客户了。这类客户的典型例子包括网上论坛里供人们查阅和发布各种消息的HTML页面、提供多个窗口供人们管理各种地址信息和约会安排信息的程序、用来完成某种系统管理工作的Perl脚本等。数据库程序设计的涉及面实在是太广了。

1.1.2 关系数据库系统与面向对象数据库系统

关系数据库已经主宰数据库领域几十年，尤其适用于存储和处理商务数据，它们通常构造为数据表的形式。除了接下来的两个段落，本书将只讨论关系数据库。

另一类数据库被统称为面向对象数据库(object-oriented database)，主要用于存储彼此没有内在联系的数据对象(而不必把它们安排到数据表里去)。虽说面向对象的程序设计语言(如Object-Store、O2、Caché)是近几年来的一个发展方向，但是面向对象的数据库产品在市场上只占相当小的份额。

注意，面向对象的程序设计语言可以用来访问关系数据库，但这并不会把一个关系数据库转变为一个面向对象的数据库。面向对象的数据库系统使人们可以使用某种程序设计语言去直接访问这种程序设计语言所定义的数据对象，还使人们可以在无需进行格式转换的情况下把这类对象存放到数据库里(这有助于保持有关对象的“原汁原味”)。这一点在关系数据库系统里是无法做到的，关系数据库系统里的数据只能被存放在结构化的数据表里。

1.1.3 数据表、记录、字段、查询、SQL、索引和键

前面已经提到过数据表，即用来实际存放有关数据的框架结构。这种数据表里的每一行被称为一条数据记录(data record)，简称“记录”，每条记录的结构和格式是由人们在定义该数据表时决定的。例如，在某个地址簿数据表里，每条记录可能包含着姓氏、名字、街道等多个字段(field)。每个字段对自己所能存储的信息类型又有一定的要求(例如，它必须是一个有着某种特定格式的数字或者是一个字符个数不得超过某个预定义最大值的字符串)。

对一个数据库的描述被称为一个数据库模型(database model)，它是由这个数据库里的全体数据表以及它们所有的字段、关系和索引构成的。数据库模型不仅要定义所涉及的各种数据结构的整体框架，还必须同时给出将存放于此的数据存储格式。

存储在数据表里的数据通常没有特定的顺序(更准确地说，数据表里的数据通常是按照它们被录入或修改时的先来后到顺序排列的)。不过，要想有效率地使用数据库里的数据，根据一项或多项要求为这些杂乱无章的数据创建一份井然有序的清单十分必要。对于一份这样的清单，只包含数据表中

4 第一部分 入 门

全体数据的一个子集往往非常有用。例如，对玩具厂商而言，一份在过去 12 个月内购买过橡皮鸭子玩具的顾客名单显然要比一份全体顾客名单更有实际意义，若是那份名单还已经按照邮政编码的顺序进行过排序，查阅起来就更方便了。

要想创建一份这样的清单，就必须构造并执行相应的查询（query）。这种查询的结果其实也是一个数据表，只不过这个数据表将只存在于计算机的内存（RAM）里而不是硬盘上。

查询是人们用各种 SQL 指令构造出来的，SQL 指令负责具体完成筛选和提取结果数据的工作。SQL 是英文 *Structured Query Language*（结构化查询语言）的缩写，这种语言已发展演变为人们在构造数据库查询命令时的一个标准。不过，因为开发数据库系统的每一家软件厂商都或多或少地在自己的产品里扩充了一些“特色”功能，所以人们当初制定这一标准的初衷——让出自不同厂商的数据库系统能够相互兼容——并没有实现。

随着数据量的增加，数据表会变得越来越大，数据字段——它们通常是按照先来后到的顺序被存入数据表的——有没有适当的索引对查询响应速度的影响也越来越大。索引（index）是一种辅助性的数据表，它们只包含一种信息：原始数据记录的排序情况。索引还经常被人们称为键字或键（key）。

索引有助于加快对数据的访问速度，但同时也存在着一些不尽如人意的弊病。首先，索引会增加数据库文件在硬盘上的空间占用量。其次，索引必须随原始数据同步更新才有实际意义，而这么做当然需要花费时间。换句话说，在读取数据时，索引可以节约时间；但在输入和修改数据时，索引反而会“浪费”时间。从全局观点看，索引是提高了效率还是降低了效率将取决于用户选用了哪一项数据作为索引。

有一种特殊的索引叫做主索引（primary index）或主键（primary key），它们与其他索引的区别在于主索引必须保证每条记录的索引值必须是独一无二的。为了做到这一点，人们通常会简单地使用一个递增的索引编号（ID 编号）作为主索引。主索引在关系数据库里扮演着一个极其重要的角色，它们可以显著加快对数据的访问速度。

1.2 MySQL

MySQL 是一种关系数据库系统。如果读者的朋友当中有 MySQL 的忠实“粉丝”，他肯定会说 MySQL 比其他任何一种数据库系统（包括 Oracle 和 DB2 等商业软件在内）都更快、更可靠和更便宜——总之，比它们都更好。但对这一观点持质疑态度的也大有人在，比较激进的反对者甚至断言 MySQL 根本不能算做是关系数据库系统。在这两种极端看法之间还有许多其他的观点。我们不想加入这场争论，但希望大家能够知道下面两个事实：

- MySQL 用户的数量一直在增加，并且他们当中的绝大多数对 MySQL 都相当满意。对于这些用户来说，MySQL 已经足够好了。
- MySQL 至今仍缺乏一些在其他数据库系统里已司空见惯的功能。如果用户必须用到这些功能，MySQL（至少在目前）不是在选择数据库系统时的最佳选择。MySQL 并不是万能灵药。

接下来将对 MySQL 的一些优点和不足做一次比较深入的探讨。

MySQL 的功能

下面列出了 MySQL 最重要的特点。这一小节是为那些已经具备了一些关系数据库知识的读者准备的。我们将借用一些来自关系数据库领域的专业名词，这可以免去另行定义一大堆术语的麻烦。这里给出的解释不是对有关术语的精确定义，但它们应该可以让数据库新手对正在谈论的话题有足够的

理解。

- **关系数据库系统。**和市场上绝大多数其他的数据库系统一样, MySQL 也是一种关系数据库系统。
- **客户/服务器体系。**MySQL 是一种客户/服务器系统。整个系统由一个数据库服务器 (MySQL) 和任意多个客户 (应用程序) 构成。客户通过与服务器进行通信的方式来完成数据查询和保存修改等操作。客户既可以与服务器运行在同一台计算机上, 也可以运行在另外一台计算机上 (此时将通过一个本地网络或因特网进行通信)。
常见的大型数据库系统 (Oracle、Microsoft SQL Server 等) 几乎都是客户/服务器系统。与它们形成对照的是文件服务器系统 (file-server system), 如 Microsoft Access、dBase 和 FoxPro 等。文件服务器系统的最大不足是在网络上运行时会因为用户人数的增加而变得非常缺乏效率。
- **SQL 兼容性。**正如其名字所表现的, MySQL 支持 SQL (Structured Query Language, 结构化查询语言) 作为自己的数据库语言。SQL 是一种专门用于查询和修改数据库里的数据以及对数据库进行管理和维护的标准化语言。
SQL 有许多种“方言”(可以说有多少种数据库系统, 就有多少种 SQL “方言”)。MySQL 遵守最新的 SQL 标准 (目前是 SQL:2003), 同时又有一些严格的限制和众多的扩展。
可以通过调整 MySQL 服务器的配置开关 sql-mode 使它在行为上与包括 IBM DB2 和 Oracle 在内的多种数据库系统保持最大限度的兼容。(调整 sql-mode 开关只会改变一部分语法定义, 这并不难做到。将在第 14 章对此做更详细的讨论。)
在 <http://www.sql-info.de/mysql/gotchas.htm> 处可以找到一篇很有意思的文章, 那篇文章对 MySQL 与另外几种比较流行的数据库系统的主要区别进行了剖析。
- **子查询。**从 4.1 版开始, MySQL 具备了对以下形式的查询进行处理的能力:

```
SELECT * FROM table1 WHERE x IN (SELECT y FROM table2)
```

(子查询还有很多不同的语法变体形式。)
- **视图。**简单地说, 视图 (view) 与一个被视为数据库对象并且能反映出数据库的某个特定方面的 SQL 查询有关系。MySQL 从 5.0 版本开始支持视图。
- **存储过程。**存储过程 (stored procedure, 简称 SP) 用于处理被存储在数据库系统内部的 SQL 代码。存储过程的常见用途是把一些特定的连续操作步骤——如插入或删除一条数据记录等——简化为一个函数调用。存储过程向编写客户软件的程序员提供了这样一种便利: 在程序里只须调用一个存储过程就可以完成一系列操作, 用不着再直接与数据表打交道了。视图和存储过程对大型数据库项目的管理维护工作有很大帮助。存储过程还可以提高工作效率。MySQL 从 5.0 版本开始支持存储过程。
- **触发器。**触发器是由数据库服务器在一些特定的数据库操作 (*INSERT*、*UPDATE* 和 *DELETE*) 过程中自动执行的一组 SQL 命令。MySQL 从 5.0 版本开始支持触发器 (但并不完备), 厂商承诺将在 5.1 版本提供更多的功能。
- **Unicode。**MySQL 从 4.1 版本开始支持所有常用的字符集, 包括 Latin-1、Latin2 和 Unicode (因为 Unicode 有 UTF8 和 UCS2 两种变体, 所以 MySQL 对 Unicode 的支持也分为两种情况)。
- **用户操作界面。**有许多种方便的用户操作界面可以用来管理 MySQL 服务器。
- **全文搜索。**全文搜索 (full-text search) 简化并加快了对文本字段内单词的搜索操作。如果打算用 MySQL 来保存文本数据 (如网上论坛的帖子内容), 它的全文搜索功能可以帮助用户轻而易举地编写出高效率的搜索函数。

- **镜像复制。** 镜像复制（replication）允许数据库管理员把某个数据库的内容动态地复制到其他计算机。在实际应用中，这么做的理由主要有两个：一是为了避免因为系统故障而中断服务（即使一台计算机出了问题，也可以立刻安排另外一台计算机投入使用）；二是为了加快数据库查询的速度。
- **事务。** 在数据库系统的世界里，事务（transaction）是指把多个数据库操作当做一个整体（块）来对待。数据库系统确保操作要么全都正确地得到执行，要么全都不执行，即使在事务过程中出现了停电、计算机崩溃或其他灾难事件也是如此。这样一来，就不会发生从银行账户汇出了一笔钱款、但对方却因为系统出了某种问题而未能收到这笔钱款的事情了。
事务机制还可以让程序员安全、及时地中止一组命令的执行（并把数据库恢复到这组命令开始执行前的状态）。有了事务机制，程序员在软件开发过程中需要考虑和解决的问题将大为减少。
MySQL 对事务的支持由来已久，只是有不少人不了解这一事实而已。要知道，MySQL 能够以好几种格式来保存数据表。MySQL 数据库系统在默认的情况下会把数据表保存为 MyISAM 格式，这种格式不支持事务。但有几种格式是支持事务的，它们当中最为流行的是 InnoDB 格式，本书会详细讨论这种格式。
- **外键约束。** 它们是程序员为了确保在彼此关联的数据表里没有找不到目标的交叉引用而定义的一些规则。MySQL 数据库系统中的 InnoDB 数据表都支持外键约束。
- **GIS 函数。** MySQL 从 4.1 版本开始支持对二维地理数据进行存储和处理。因此，MySQL 很适合用来开发 GIS（Geographic Information System，地理信息系统）应用程序。
- **程序设计语言。** 在开发 MySQL 应用程序的时候，有一大批 API（application programming interface，应用编程接口）和软件开发库可供选用。以客户端程序为例，可以用（但不限于）C、C++、Java、Perl、PHP、Python 和 Tcl 等许多种语言来编写。
- **ODBC。** MySQL 支持 ODBC 接口 Connector/ODBC。能够在 Microsoft Windows 环境里运行的常用编程语言（Delphi、Visual Basic 等）都可以通过这个接口访问 MySQL 数据库。ODBC 接口也可以在 UNIX 环境里实现，但那并没有太大的必要。
根据个人喜好和具体情况，为微软公司新近推出的.NET 平台开发软件的 Windows 程序员既可以选用由第三方提供的 ODBC 接口组件，也可以选用由微软公司提供的.NET 接口 Connector/.NET。
- **平台独立性。** 不仅 MySQL 客户应用程序可以在多种操作系统下运行，MySQL 本身（即 MySQL 服务器）也可以在多种操作系统下运行。其中最重要的是 Apple Macintosh OS X、Linux、Microsoft Windows 和数不胜数的 UNIX 变体，如 AIX、BSDI、FreeBSD、HP-UX、OpenBSD、NetBSD、SGI Iris 和 Sun Solaris。
- **速度。** 人们普遍认为 MySQL 是一个非常快的数据库程序，有大量的性能测试报告支持这一结论。但希望大家不要盲目相信这类测试报告，不管它们来自何方，都应该抱着清醒的怀疑态度看待它们。

1.3 MySQL 的不足

在这一小节里列出的一部分不足之处可以在 MySQL 开发团队的未来工作计划里查到，另外一部分则是 MySQL 现有功能当中还不够完善的地方。

提示 MySQL软件的文档对部分功能不够完善或者缺少某些功能的事实并没有避而不谈。在MySQL软件文档里有一个文件对“MySQL与各种标准的兼容程度”的话题做了深入浅出的分析，可以在那里找到很多关于“MySQL在哪些地方与有关标准不兼容”的信息。该文档还对一部分不足之处的形成原因做出了解释，并针对一些疑难点提供了规避或补救办法。该文档的URL地址是：<http://www.mysql.com/doc/en/Compatibility.html>。

- 在对默认格式（即 MyISAM 格式）的数据表进行处理时，MySQL 的锁定机制——即暂时禁止对数据库信息的访问或修改——将对整个数据表起作用（数据表锁定）。可以另外选用支持事务的数据表格式（如 InnoDB）来绕过数据表锁定问题，它们支持数据行锁定。
- 在对 MyISAM 数据表进行处理时，MySQL 不能进行热备份。热备份的意思是无须锁定数据表就可以在对数据表进行处理的同时对其进行备份。这个问题的解决方案还是 InnoDB，但此种情况下的热备份功能目前还需要另外花钱购买。
- 许多数据库系统都允许用户自定义数据类型，但 MySQL 目前还不支持这种做法，短期内也没有这方面的计划。
- MySQL 直到现在仍对日益流行的 XML 趋势视若无睹。我们不清楚 MySQL 要到什么时候才能直接处理 XML 数据。与 MySQL 相比，许多商业化的数据库系统在这方面提供了丰富得多的功能，就连 SQL:2003 标准也定义了多项 XML 功能。
- MySQL 确实是一种非常快的数据库系统，但用它来开发实时应用程序的厂商或个人非常少，它至今仍不能提供任何 OLAP 功能。OLAP 是英文 *online analytical processing*（实时分析处理）的缩写，意思是采用一些特殊的方法来管理和分析多维数据。支持 OLAP 的数据库系统通常被称为数据仓库（*data warehouse*）。
- MySQL 从 5.0 版本开始支持存储过程和触发器，但它的这些功能还远谈不上成熟（尤其是在触发器方面）。与商业化数据库系统相比，MySQL 在这方面功能既不够稳定，也不够丰富。
- MySQL 从 4.1.0 版本开始支持的 GIS 功能也存在着同样的问题。商业化数据库系统在这方面提供的功能要比 MySQL 丰富得多。

1.4 MySQL 的版本编号

哪个版本的 MySQL 软件是最新的？它的各个版本都有哪些功能？这类问题就是 MySQL 老手也很难说清楚。本节将提供一些关于 MySQL 版本编号方面的信息。

截止到 2005 年 1 月，MySQL 开发团队已经推出并仍在完善以下 4 个版本系列：

- **MySQL 3.23.n：**该系列里的第一个版本是在 1999 年 8 月发布的 3.23.0 版。从 3.23.32 版（2001 年 1 月发布）开始，MySQL 3.23.n 系列的稳定性得到了人们的公认。这个系列的最新版本是 3.23.58。虽说 MySQL 3.23 近几年来一直没有什么大的创新，但因特网服务提供商们还是很喜欢选用它。未来的 MySQL 3.23.n 系列将不再有功能方面的扩展，新版本将只修补新发现的错误和安全漏洞。
- **MySQL 4.0.n：**该系列里的第一个版本是在 2001 年 10 月发布的 4.0.0 版。从 4.0.12 版（2001 年 1 月发布）开始，MySQL 4.0.n 系列的稳定性得到了人们的公认，建议大家在日常应用中选择这个系列的 MySQL 版本。该系列的最新版本是 4.0.23。与 MySQL 3.23 类似，该系列的未来版本也将只修补漏洞。

- **MySQL 4.1.n:** 该系列里的第一个版本是从 2003 年 4 月开始提供下载的 4.1.0 版。从 4.1.7 版（2004 年 10 月发布）开始，MySQL 4.1.n 系列的稳定性得到了人们的公认，建议大家在日常应用中选择该系列的 MySQL 版本。该系列的最新版本是 4.0.23。与 MySQL 3.23 类，该系列的未来版本也将只修补漏洞。
- **MySQL 5.0.n:** 该系列里的第一个版本是从 2003 年 12 月开始提供下载的 5.0.0 版。该系列的最新版本是 5.0.2，但因为它目前仍处于 Alpha 阶段，所以不建议大家在日常应用中选用。期望 MySQL 5.0.n 到本书出版之时能变得更加稳定。

1.4.1 Alpha、Beta、Gamma、Production (Generally Available)

MySQL 的版本变迁过程可以分为 Alpha、Beta、Gamma 和 Production 4 个阶段。

- **Alpha**。意思是这个版本仍处于开发阶段，还会增加新的功能或是做出重大改动。必须把所有已知的错误全都改好之后才能对外发布 Alpha 版本，但在其内部极有可能还隐藏着未被发现的错误。在 Alpha 版本的测试阶段出现数据损失的可能性非常高。Alpha 版本只适合那些有兴趣尝试 MySQL 最新功能的程序员试用。
- **Beta**。意思是这个版本已基本完成，但是还没有经过全面彻底的测试。Beta 版本一般不会再有太大的改动。
- **Gamma**。意思是比 Beta 版本更稳定一些。这一阶段的主要任务是查找错误并解决它们。
- **Production 或 Generally Available (GA)**。意思是 MySQL 软件的开发团队认为这个版本已经足够成熟和稳定，可以用在较为关键的场合。根据 MySQL 文档，Production 版本上的后期工作以小的修改为主，一般不会再添加新的功能。但以往事实却并非总是如此，有好几个其稳定性已得到人们公认的版本都曾出现过大修大改的情况。尤其是 MySQL 3.23.n 系列，在宣布该版本已经稳定之后（3.23.32），MySQL 开发团队又给它增加了对 InnoDB 和 BDB 数据表的支持（3.23.34），稍后又增加了一些针对 InnoDB 数据表的集成规则（3.23.44）。除了这些大动作，这个系列的 MySQL 版本还曾有过许多次小规模的功能扩展。MySQL 开发团队对这些扩展还是很满意的，但这同时也导致不同 Production 版本之间产生了一些兼容性问题。

在实践中可以掌握这样一个原则：一个全新的 MySQL 版本（即 *n.n.0* 版）总是意味着开发工作正处于 Alpha 阶段，而越来越大的版本编号意味着开发工作进入了 Beta、Gamma 或 Production 阶段。

普通的 MySQL 用户应该只选用已经进入 Production 阶段的 MySQL 版本。如果打算用 MySQL 开发 Web 应用程序，就应该先弄清 ISP（因特网接入服务提供商）使用的是哪一个版本。（ISP 最关心的问题是系统的管理维护工作是否容易进行和系统是否稳定，它们当中的大多数都不会激进地选择最新的版本，而是更喜欢使用老一点儿的版本，而老版本在功能方面往往要比新版本少一些。）

提示 如果读者有兴趣试用一个需要自行完成编译的在开发版本（这意味着它很可能连 Alpha 测试版都算不上），请按照下面这个网页上的指示去下载和编译源代码：<http://dev.mysql.com/doc/mysql/en/installing-source.html>。

1.4.2 按版本号排列的 MySQL 功能表

判断哪个数据库版本可以配合 MySQL 软件的哪个版本使用并不总是一件容易的事情。虽说 MySQL 软件的版本号可以有助于判断某个新增功能的第一次正式出现是在什么时候，但在那之后

又对该功能进行过重大修改或纠错的情况时有发生，新增功能往往需要经历多个版本的完善才会最终稳定下来。另一个值得注意的情况是有些功能只适用于特定的数据表格式。

表 1-1 还对 MySQL 的未来做了一些展望，这些预测是根据 2005 年春季 MySQL 软件文档里的信息整理出来的。

在此必须提醒大家一句：千万不要把表 1-1 里的预测当作承诺。根据以往的经验，新增功能的实际发布日期常常会比预定计划提前或者延误一段时间。

表 1-1 MySQL 功能表（旧版本和预测版本）

功 能	版 本
镜像（动态复制）	3.23
在 MyISAM 数据表中进行全文搜索	3.23
BDB 数据表开始支持事务处理	3.23.34
InnoDB 数据表开始支持事务处理	3.23.34
InnoDB 数据表上的引用集成性检查功能	3.23.34
<i>DELETE</i> 和跨多个数据表的 <i>DELETE</i>	4.0
跨多个数据表的 <i>UPDATE</i>	4.0
<i>UNION</i> （合并多个 <i>SELECT</i> 结果）	4.0
查询缓存区（加快重复执行的 SQL 命令的执行速度）	4.0
嵌入式 MySQL 库	4.0
加密通信（Secure Socket Layer, SSL）	4.0
InnoDB 数据表开始支持热备份（商业化的额外组件）	4.0
适用于客户软件共享函数库的 GPL 许可证（即以前的 LGPL 许可证）	4.0
子查询	4.1
支持 Unicode（UTF8 和 UCS2=UTF16）	4.1
支持 GIS（GEOMETRY 数据类型, R 树索引）	4.1
可变语句（带参数的 SQL 命令）	4.1
<i>GROUP BY</i> 语句增加了 <i>ROOLUP</i> 子句	4.1
<i>mysql.user</i> 数据表采用了更好的口令字加密算法	4.1
允许单个数据表单独存放在一个 InnoDB 表空间文件里	4.1
<i>VARCHAR</i> 类型的数据列可以容纳超过 255 个字符	5.0
引入了 <i>BIT</i> 数据类型	5.0
存储过程（Stored Procedure, SP）	5.0 版前后
触发器（自动执行 SQL 代码）	5.0/5.1
视图	5.0
更节约空间的 InnoDB 表空间格式	5.0/5.1
新的数据库架构管理方案（数据字典, <i>INFORMATION_SCHEMA</i> 数据库）	5.0
<i>FULL OUTER JOIN</i>	5.1 版前后
针对 MyISAM 数据表的引用一致性检查机制	5.1 版前后
无须用户参与的镜像备份功能	5.1 版前后
MyISAM 数据表开始支持热备份	5.1 版前后
只作用于数据列的约束	5.1 版前后
视图	5.1 版前后
XML 支持	计划中，但日期未定
允许用户自行定义新数据类型	目前尚无计划

上表中的信息是根据以下 MySQL 文档的内容整理的：

<http://dev.mysql.com/doc/mysql/en/roadmap.html>
<http://dev.mysql.com/doc/mysql/en/todo.html>
<http://dev.mysql.com/doc/mysql/en/news.html>

1.5 MySQL 的许可证

MySQL 软件的许可证是一个比较复杂有趣的问题。MySQL 是一个开源项目，即 MySQL 软件的源代码是可以自由获得的。从 2000 年 6 月起（即从 3.23.19 版起），MySQL 软件在发行时开始采用 GNU Public License (GPL, GNU 公共许可证)，这就保证了 MySQL 软件今后将一直可以自由——这里所说的“自由”指的是开源意义上的自由——获得和转让。那些基于 MySQL 的商业化应用软件在 GPL 许可证之外还需要获得另外一种许可证，这个问题马上就会讲到。

1.5.1 GPL 许可证下的权利和义务

有不少人把“开源”错误地理解为“免费”，实际情况却并非如此。的确有许多无需支付任何费用就可以获得和使用的 GPL 软件，但这么做是有先决条件的。“开源”并不等于“免费”，它的含义要深远得多。

- 因为源代码可以自由获得，所以当遇到问题时，完全可以不依赖某个具体的软件厂商而自行对其进行修改。
- 在遇到问题时，完全可以自行修补发现的漏洞或给它增加新的功能。当然，也完全可以向该软件的研发团队求助。
- 可以确信，程序代码已经经过许多程序员的阅读和审查，一般不会存在让人大吃一惊的错误或漏洞（例如，有一种名叫 Interbase 的数据库系统被人们发现曾长期存在着一个后门：利用一个硬编码在程序代码里的密码，别人就可以访问 Interbase 数据库里的任何一个数据表）。
- 有权对 GPL 产品做出修改，也有权销售由本人修改后的 new 程序。

GPL 许可证制度除了上面提到的这些好处，还有一些坏处（对商业化应用软件而言）。如果打算在一个 GPL 程序的基础上开发一个商业化产品，就必须公开并允许别人在开源的意义下自由使用改写的那部分源代码。绝大多数商业化软件的开发者是不愿意这么做的。

总地来说，以 GPL 软件为基础开发出来的每一个程序都必须沿用 GPL 许可证（换句话说，GPL 许可证的约束力是可传递的）。

提示 关于开源理念的更多信息，以及 GPL 许可证的全文内容和解释可以在以下网址查到：

<http://www.gnu.org/copyleft/gpl.html>
<http://www.opensource.org/osd.html>

1.5.2 开源许可证下的 MySQL 软件

根据 GPL 许可证的有关规定，可以在以下场合自由使用 MySQL 软件：

- 如果是为了自行开发一个应用软件且不进行商业销售，就可以免费使用 MySQL。可一旦把开发的解决方案销售给别人，许可证问题就来了。这条原则在 MySQL 的官方网站上是这样表述的：“只要你能做到永远也不复制、修改或再发行，就可以永远免费使用 MySQL 软件。”
- 如果使用范围仅限于一个网站的内部，就可以免费使用 MySQL。如果还开发了一个 PHP 应用

程序并需要把它的部分组件安装到 ISP (Internet Service Provider, 因特网接入服务提供商) 的系统里去, 是否把 PHP 代码按 GPL 意义自由公开由本人决定, 可以选择不公开它们。

- 类似地, 对 ISP 而言, 如果只是提供 MySQL 数据库给自己的顾客使用, 就无需为 MySQL 软件支付许可证费用。这是因为 MySQL 软件将只运行在 ISP 的计算机上, 所以有关的应用程序将被认为是属于“仅供内部使用”的范畴。
- 最后, 凡是符合 GPL 许可证或其他与之类似的免费许可证里有关规定的软件开发项目, 都可以免费使用 MySQL 软件。(例如, 假设用户正在为 Linux 开发一个新的、免费的电子邮件客户软件, 那么, 只要想把电子邮件存放到一个 MySQL 数据库里去, 就可以毫无顾虑地这样做。)

1.5.3 商用许可证下的 MySQL 软件

根据 GPL 许可证的有关规定, 下列行为是被禁止的:

- 如果不愿意把改动过的源代码自由公开, 就不应该对 MySQL 软件 (更准确地说是 MySQL 数据库服务器) 进行修改或扩展, 也不应该对外销售修改后的版本或新产品。这些行为是 GPL 许可证所不允许的。
- 在开发一个商业化产品 (如一个财务软件) 时, 如果打算用 MySQL 来实现它的数据库功能, 就必须把这一部分源代码自由公开。如果不愿意这么做, 就不应该使用 MySQL。否则也是 GPL 许可证所不允许的。

如果用户是一名商业软件的开发者并认为 GPL 许可证中的有关规定难以接受, 可以选用某种商业用途的 MySQL 许可证来销售产品 (程序)。这种折衷是很公平的: 有权使用 MySQL, 同时又不必 (或许是因为客观条件不允许, 或许是因为本人不情愿) 按照 GPL 许可证的有关规定公开代码。

MySQL 公司为它们的顾客 (从该公司购买数据库应用软件的人们) 准备了以下两种商用许可证:

- **MySQL Network:** 持有这种许可证的顾客有权在一年的时限内运行一个支持 InnoDB 的 MySQL Pro Certified Server 服务器 (这里所说的“服务器”指的是一台计算机——不管它里面有多少个 CPU), 而允许访问这台服务器的客户可以是任意多个。MySQL Network 许可证的持有者还将获得对一些经过特殊编译的服务器版本 (根据 MySQL 公司的说法, 这些版本非常稳定) 的访问权、由 MySQL 公司提供的电子邮件和电话方式的技术支持及其他一些售后服务。在许可证的有效期内, 对 MySQL 服务器 (甚至包括 4.0 和 5.0 系列版本在内) 的升级是完全免费的。
- **Classic Commercial License:** 在 MySQL Network 许可证出现之后, 曾盛行一时的普通商用 MySQL 许可证就难觅踪迹了 (有兴趣的用户可以通过下面“提示”内容里的最后一个链接了解这一许可证的具体内容)。普通商用 MySQL 许可证没有时效方面的限制, 但它规定持有者只能运行某个特定系列的 MySQL 版本——4.0 版许可证的持有者如果还想运行 5.0 版, 就必须再为 5.0 版另外购买一份许可证。与商业化的数据库系统相比, 普通商用 MySQL 许可证的收费还是很合理的 (目前的价格是: 不带 InnoDB 支持功能为 295 美元; 带 InnoDB 支持功能为 595 美元。一次购买 10 份以上许可证的顾客可以在价格方面获得非常大的折扣)。

MySQL Network 许可证的优势在于丰富的技术支持服务和免费升级条款。普通商用 MySQL 许可证的优势是比较便宜, 只要安装好 MySQL 服务器, 就可以让它永远运行下去。

提示 关于 MySQL 软件的许可证问题，以下网站提供了更详细的信息：

<http://www.mysql.com/company/legal/licensing/>
<http://www.mysql.com/network/>
<https://order.mysql.com/>
<https://shop.mysql.com/?sub=vt&id=software>

1.5.4 MySQL 客户软件开发库 (Connector/ODBC、Connector/J 等) 的商用许可证

除了 MySQL 服务器软件，MySQL 公司还为开发各种应用软件所必需的客户软件开发库准备了几种许可证（还有许多由 MySQL 公司以外的第三方开发的驱动程序包，如 PHP 驱动程序包）。

- **C-API**: 与 C 语言配合使用的应用程序接口。
- **Connector/C++**: 与 C++ 语言配合使用的开发库。
- **Connector/J**: 与 Java 语言配合使用的驱动程序。
- **Connector/MXJ**: 与 Java 语言配合使用的 J2EE MBean 驱动程序。
- **Connector/.NET**: .NET 驱动程序，主要用于开发 Windows 环境下的应用程序。
- **Connector/ODBC**: ODBC 驱动程序，适用于开发 Windows 环境下的各种程序。

与 MySQL 服务器软件的情况类似，这些驱动程序也是采用 GPL 许可证发行的。有许多 MySQL 客户软件开发库（例如 JDBC 接口 Connector/J）的早期版本采用的是 LGPL（Lesser Gnu Public License，弱 Gnu 公共许可证）；LGPL 许可证在商业软件方面的限制相对要少一些。不过，这种情况从 4.0 版本开始发生了变化。

如果打算使用上述驱动程序来开发商业化的应用软件，有件事情必须知道：如果开发的 MySQL 客户程序将被用来访问一个有许可证的 MySQL 服务器，那份服务器许可证的效力将同样适用于准备使用的 MySQL 客户软件开发库。换句话说，为使用 MySQL 客户软件开发库而去购买一份许可证往往没有必要，服务器许可证已经把客户软件开发库的版权自动包括在内了。

不过，如果开发和销售的商业化程序不随 MySQL 服务器一同发行——即把 MySQL 服务器的安装和许可证问题留给顾客去解决，就必须为编写程序时用到的每一个客户软件开发库购买一份许可证。

在 MySQL 公司看来，顾客为 MySQL 服务器购买许可证的情况属于正常业务，顾客单独为 MySQL 客户软件开发库购买许可证的情况则属于特例。MySQL 客户软件开发库的商用许可证其实是 MySQL 公司为了维护自己的权益而采取的一种保护措施。MySQL 公司当然不希望看到这样的局面：商业软件的开发者以不向顾客提供 MySQL 服务器为由拒绝从 MySQL 公司购买许可证，而商业软件的使用者只须从因特网下载一份 MySQL 服务器的 GPL 版本就可以解决配套问题。如果真是那样，MySQL 公司将无法从第三方的商业化 MySQL 产品的开发项目上获得任何收益。

1.5.5 PHP 项目的客户许可证问题——F(L)OSS 特例

在推出 MySQL 4.0 版的前后，MySQL 公司遇到了这样一个难题：MySQL 客户软件开发库采用的是 GPL 许可证，而某些开源项目不受 GPL 许可证的约束。例如，PHP 项目没有采用 GPL 许可证，它采用的是另外一种比 GPL 宽松得多、对商业化软件的限制非常少的开源许可证。如果还要求 PHP 必须按照 GPL 许可证的有关规定去使用 MySQL 客户软件开发库，就意味着整个 PHP 项目必须全盘接受 GPL 许可证的约束。PHP 开发者显然不愿意这样做，否则就不会采用另一种开源许可证了。为了解决这个问

题, MySQL 公司定义了一个特例来允许 PHP 项目把 MySQL 客户软件开发库作为 PHP 的一部分。

MySQL 公司为不采用 GPL 许可证的开源项目定义的各种特例统称为“FLOSS 许可证特例”。FLOSS 是英文 Free/Libre and Open Source Software(自由和开源软件)的缩写。MySQL 公司的官方网站经常把这个缩写单词进一步简写为 FOSS。简单地说,如果某个开源项目打算采用的许可证是 FOSS 特例之一,该项目就可以不受 GPL 许可证的约束而把 MySQL 客户软件开发库集成到它们的代码里去。这些许可证特例的详细信息可以在以下网址查到:

<http://www.mysql.com/company/legal/licensing/faq.html>
<http://www.mysql.com/company/legal/licensing/foss-exception.html>

在过去,Red Hat 公司一直认为这些许可证特例不够完备,所以该公司出品的 Linux 发行版本,如 Red Hat Enterprise (RHEL) 和 Fedora Core 等多年来只收录了早期的 MySQL 3.23.n 系列版本。但 Red Hat 公司在推出 Red Hat Enterprise 4.0 (2005 年 2 月) 和 Fedora Core 4 (2005 年 4 月) 时改变了它们以前的做法,这两种 Linux 发行版本收录的都是 MySQL 4.1 版本,而该版本是现时期最为稳定的 MySQL 版本。

1.5.6 MySQL 软件的版本名称

从 MySQL 4.0 版起,MySQL 软件的自由 (GPL 意义上) 版本和商用版本开始分别使用不同的名字,如表 1-2 所示。

表 1-2 MySQL 软件的版本名称

名 称	许可证	用户评价
MySQL Standard Community Edition	GPL	所有的默认功能(包括 InnoDB)都可以被认为是稳定的
MySQL Max Community Edition	GPL	与 MySQL Standard 相同,但部分附加功能和数据表驱动程序还未达到完全成熟地步
MySQL Classic	商用	与 MySQL Standard 相近,但没有 InnoDB 支持
MySQL Pro	商用	与 MySQL Standard 相近,有 InnoDB 支持(因而支持事务处理)
MySQL Pro Certified Server	商用	与 MySQL Pro 相近,但只能在 MySQL Network 许可证下使用,这个系列的 MySQL 软件的稳定性和安全性是最高的

对于“正宗”的 GPL 应用软件,建议大家选用 MySQL Standard 版本作为搭档。MySQL Max 版本提供的附加功能只有在那些具有比较特殊用途的应用程序里才能发挥最大作用。在现阶段(5.0 版),MySQL 软件的 Standard 和 Max 版本的主要区别在于是否支持 BDB 数据表和是否对客户/服务器连接进行 SSL 加密。这些功能只有在它们足够稳定之后才会被添加到 Standard 和商用版本里去。可以用 SHOW VARIABLES 命令来查看自己的 MySQL 版本都支持哪些功能。下面是在 MySQL Standard 5.02 版本下执行这条命令的结果:

```
mysql> SHOW VARIABLES LIKE 'have%';
```

Variable_name	Value
have_archive	NO
have_bdb	NO
have_compress	YES
have_crypt	YES
have_csv	NO
have_example_engine	NO

```

have_geometry      YES
have_innodb        YES
have_isam          NO
have_ndbcluster    NO
have_openssl       NO
have_query_cache   YES
have_raid          NO
have_rtree_keys    YES
have_symlink       YES

```

在商用应用方面，选择 Classic 或 Pro 版本的标准很简单——是否需要 InnoDB 支持（Pro 版本的许可证费用比较昂贵）。Classic 和 Pro 版本目前还不能从 www.mysql.com 网站下载，必须先获得一份许可证才能获得和使用它们。

MySQL Pro Certified Server 是 2005 年初推出的新产品，它是 MySQL Network 许可证的组成部分。根据 MySQL 公司向用户做出的承诺，随 MySQL Network 许可证提供的 MySQL 软件是最稳定和最安全的版本。

如果想知道自己使用的 MySQL 软件是哪一个版本，可以执行以下命令：

```
mysql> SHOW VARIABLES LIKE 'version';
```

Variable name	Value
version	5.0.2-alpha-standard

上面的输出结果表明，在系统上运行的是 Standard 5.0.2 版本。该版本缺少 Max 版本所提供的附加功能，也不允许用于商业用途。

注解 不要把 MySQL Max 和 MaxDB 混为一谈。后者原来叫做 SAP-DB 数据库系统，MySQL 公司在 2005 年接管和收购了这个品牌。MySQL 数据库系统和 MaxDB 数据库系统有许多相似之处，但也有着重大的区别。本书只讨论 MySQL，不讨论 MaxDB。

1.5.7 MySQL 软件的技术支持合同

不管使用的 MySQL 软件是一个商用版本还是一个 GPL 版本，都可以与 MySQL 公司签署一份技术支持合同。这份合同是从 MySQL 公司获得技术支持服务并分享 MySQL 软件未来开发成果的保证。

提示 MySQL 软件商用许可证的详细内容及相关费用的支付办法可以通过 <http://www.mysql.com/support> 网页查到。为 MySQL 软件提供商业化技术支持服务的网上资源和公司名单可以通过 <http://solutions.mysql.com> 网页查到。

1.6 MySQL 软件的替代品

MySQL 软件有许多种替代品，尤其是在准备承受（高昂的）许可证费用或者是需要支持某种必不可少的硬件设备时。在替代产品当中，IBM DB2、Informix、Microsoft SQL Server 和 Oracle 是最常见的几种。

如果正在开源领域中寻找 MySQL 的替代品，PostgreSQL 大概是最值得考虑的候选了。但有必要提醒大家一句：MySQL 和 PostgreSQL 各有各的支持者，他们之间关于这两种产品孰优孰劣的争论与其说是一种理智的探讨，不如说是一场宗教战争。客观地讲，PostgreSQL (<http://www.postgresql.org/>) 提供的功能要比 MySQL 丰富（尽管 MySQL 正在缓慢地缩小着这种差距），但 PostgreSQL

在速度和稳定性方面要逊色于 MySQL（尽管 PostgreSQL 在这方面的口碑已经大为改观）。

在小型数据库解决方案（这往往意味着较少的资金成本）方面，SQLite(<http://www.sqlite.org/>)很值得考虑。这是一个体积不大（约 250KB 的二进制代码）、但有能力支持绝大多数初级数据库功能的函数库。SQLite 即使对于商业应用程序也是免费的，它特别适合用来开发将单机运行的数据库解决方案，不太适合用来开发网络化的应用软件。目前流行的几种 PHP 版本大都集成有 SQLite 组件。

MySQL 软件还有一些是从以往的商业化数据库系统转变为开源软件的替代品，它们当中最出名的是 Firebird(<http://firebird.sourceforge.net>)，其前身是由 Inprise/Borland 公司推出的 Interbase。

最后介绍一下，刚才提到的 MaxDB 数据库，以前叫做 SAP-DB (<http://www.sap.com/products/maxdb/>)，也是一个不错的选择。除了由 MySQL 公司负责其维护和销售以外，MaxDB 和 MySQL 数据库系统并没有什么瓜葛。MaxDB 是一种专业级的大型数据库系统，因而有着大型（商业化）数据库系统的共同弊病：难以安装、管理和维护；用户数量少；因特网上的文档和求助资源不够多。

1.7 小结

MySQL 是一种功能非常强大的关系型客户服务器数据库系统。它的安全性和稳定性已足以满足许多应用程序的要求，而且有着非常高的性价比（这不仅是因为 MySQL 本身是免费的，还因为它对硬件性能的要求不那么苛刻）。这些优点使 MySQL 成为因特网数据库领域里事实上的标准之一。

别的不说，在 Linux 领域里，以 MySQL 作为后端数据库引擎的应用程序正越来越多：MySQL 可以帮助人们更有效率地管理各种日志数据以及电子邮件、MP3 文件、地址名单等数据。在基于 Windows 操作系统的环境里，MySQL 同样扮演着类似于喷气引擎的角色（在许多场合，MySQL 提供了更好的技术基础），因为有了 ODBC 接口，MySQL 现在已经可以在 Windows 操作系统里完成这些任务了。

与其他的开源数据库系统相比，MySQL 不仅在性能指标方面高出一截，在使用范围方面也远远领先于其他竞争对手。MySQL 比其他数据库系统经受过更全面的测试，有着更齐全的文档，有 MySQL 经验的开发人员也相对更多一些。

不过，与商业化数据库系统领域里的大块头们相比，MySQL 目前还无法在所有的方面与之抗衡。如果所负责的项目必须用到某些 MySQL 尚不支持的功能，那还是不要选用 MySQL 为好。

最终的决定要取决于对“MySQL 对我的应用程序来说是不是个最佳选择”这个问题的回答，而这里可以告诉大家的是：对这个问题做出肯定回答的不仅有数以百万计的网站开发人员，还有 Associated Press、Citysearch、Cox Communications、Los Alamos National Labs（数据量高达 7TB）、Lycos、NASA、Sony、Suzuki、Wikipedia 和 Yahoo！等大公司和大机构。从这个意义上讲，选择 MySQL 就等于是让自己与这些优秀的公司站在了同一条起跑线上。

第2章

测试环境

本章的讨论重点是如何在一台本地计算机上为 MySQL 建立一个测试环境。因为 MySQL 通常不是单独运行，而是需要与其他一些程序配合使用，所以还将讨论 Apache、PHP 和 Perl 等软件的安装和配置问题。

本章将分别讨论 MySQL 软件在 Windows 和 Linux 两种操作系统环境里的安装步骤。此外，还将涉及一些比较高深的话题，其中包括如何在 Windows 环境下对 MySQL 服务进行管理、如何在 Linux 环境下对 MySQL 软件的某个开发测试版本进行编译等。

2.1 是 Windows 还是 UNIX/Linux

“在哪一种操作系统环境下运行 MySQL 更好”是一个很容易引起争论的问题，不同的阵营会给出不同的答案。但希望大家能够以客观平和的心态来回答这个问题，并以同样的心态来评价这本书。作为一个事实，MySQL 本身以及绝大多数与 MySQL 配合使用的软件，如 Apache、PHP、Perl 等，都是首先在 UNIX/Linux 环境下开发出来，然后才被移植到 Windows 环境的。

2.1.1 MySQL 应用现状（因特网上的数据库服务器）

在现实中，或者说在因特网上的数据库服务器当中，刚才提到的那些程序几乎都运行在 UNIX/Linux 环境下。从全世界的范围来看，MySQL 及相关软件的应用以实现网上数据库为主。根据这一事实，可以有把握地说，MySQL 及相关软件在 UNIX/Linux 环境下的版本有着更高的质量——因为它们随时都在经受各种实际应用测试，所以这些软件的 UNIX/Linux 版本中存在的错误和漏洞往往更迅速地被人们发现、纠正和弥补。

一般来说，部署在 UNIX/Linux 环境下的软件程序往往有着更高的运行效率。得出这一结论的原因并不是因为相信了“Windows 比 UNIX/Linux 速度慢”这一说法（笔者不想在这里讨论这个问题），而是因为这样一个事实：不同的操作系统在它们所采用的进程和线程模型方面有着相当大的差别。UNIX/Linux 编程模型对 Apache 和 MySQL 等软件进行优化的工作不仅开始得最早，进行得也最全面彻底，而 Windows 在这方面落后了；这就使这些软件的 UNIX/Linux 版本在速度方面取得了较大的领先优势。

因此，刚才提到几种软件的开发团队普遍倾向于让自己的程序运行在 UNIX/Linux 环境而不是 Windows 环境下。

注解 在提到Windows的时候，一般是指Windows 2000/XP/2003。虽说MySQL也可以在Windows 9x/Me系统上运行，但这类系统的安全性根本不足以阻挡未经授权的访问者。如果没有特别注明，这本书里的有关测试都是在Windows XP系统上进行的。

2.1.2 开发环境

如果用户手里的数据库应用程序还处于开发阶段，情况就有所不同了。在开发过程中，用户使用的是一个测试环境，而这个环境通常只有用户本人或用户的团队才能访问，不太可能因为访问量很大、访问者的成分很复杂而导致系统在安全或效率方面出现问题。这样一来，因为 MySQL 软件的 UNIX/Linux 版本和 Windows 版本有着很好的兼容性，先在 Windows 环境下开发出一个（比如说）讨论组软件、再把这个解决方案完整地移植到 ISP 的 Linux 服务器上去的做法就很值得考虑。

当然，如果用户最喜欢的文本编辑器只能在 Windows 下运行而测试环境中又至少有两台计算机，那么最好把其中一台计算机安装为 Linux 系统并把 MySQL 和 Apache 等软件安装在这台计算机里，把另一台计算机安装为 Windows 系统并用它来完成设计数据库、创建脚本文件等具体开发工作。

一般来说，对软件的开发工作而言，争论 Windows 和 UNIX/Linux 孰优孰劣并没有多大的意义，用户应该选择自己最熟悉的操作系统。

话虽如此，人们对于是否应该使用 Windows 来开发软件的争论也不是没有道理。例如，如果某个项目全部是在 UNIX/Linux 环境下开发的，等到了这个项目的部署阶段，开发者此前获得的经验将特别有助于把它部署在 UNIX/Linux 服务器（这一点在访问权限控制和安全防护方面表现得尤为突出）。

再比如说，同样的系统功能在 UNIX/Linux 环境下和在 Windows 环境下的具体实现或部署机制往往会有差异，开发者必须提前考虑到这类差异才能确保项目的成功。例如，在 UNIX/Linux 环境下，通过程序代码（如一个 PHP 或 Perl 脚本）就可以发送电子邮件，而 Windows 环境没有提供这方面的标准化编程接口。

2.2 在 Windows 系统上安装 MySQL 和相关软件

本节将介绍如何在 Windows 系统上分别单独安装 Apache 2、PHP 5、Perl 5.8 和 MySQL 5.0 等几个软件。分别单独安装这些软件的好处是可以选择这些组件的具体版本，清晰地掌握自己的计算机里都安装了哪些程序以及它们的具体配置情况。这对今后的系统维护工作和软件升级工作会有很大的帮助。

注解 使用一个包含了所有组件的软件包来完成安装工作当然会比较容易。在这类软件包当中，最有名的是 *The Saint WAMP (TSW)* 和 *XAMPP for Windows*:

<http://www.sourceforge.net/projects/yawamp/>
<http://www.apachefriends.org/de/xampp-windows.html>

但使用这种软件包来安装有关软件有两个弊病：其一，除了真正需要的Apache、PHP、Perl和MySQL等组件，这类套装软件包往往还额外包含许多根本用不到的组件；其二，截止到本书付印之时，收录在这类套装软件包里的MySQL软件都不是5.0版本，用户只能获得出现较早、但相对更为稳定的MySQL 4.1或4.0版本。

2.2.1 安装 Apache 2.0

1. 禁用IIS

如果使用的 Windows 操作系统是一个服务器版本，首先应该禁止运行其 Microsoft Internet Information Server（简称 IIS）组件。按标准配置（80 端口）安装的 Apache 软件在同时运行着 IIS 的计算机上无法工作（具体原因笔者也不太清楚）。

查看或禁用 IIS 的方法是：执行菜单命令 Settings（设置）| Control Panel（控制面板）| Add or Remove Programs（添加或删除程序）| Add/Remove Windows Components（添加/删除 Windows 组件），然后（如有必要）在 Windows Component Wizard（Windows 组件安装向导）窗口里取消选中 IIS 复选框，就可以终止并删除该程序了。

2. 安装Apache

Apache 软件的 Windows 版本可以从 www.apache.org 网站（及其各种镜像网站）上下载。从这些网站把 apache_2.n_xxx.msi 文件下载到自己的计算机上以后，在 Windows Explorer（Windows 的“资源管理器”）窗口里双击这个文件就可以启动安装程序。屏幕上会出现一个对话框（如图 2-1 所示），要求输入几个基本的配置参数。必须在该对话框里给出一个域名（例如 *sol*）和一个服务器名（例如 *uranus.sol*）；如果计算机将作为独立的测试计算机使用，这些信息必不可少。一般来说，系统会在这几个文本框里自动填上适当的参数值。如果不清楚应该为计算机填写什么样的名字，可以执行菜单命令 Settings（设置）| Control Panel（控制面板）| System（系统）| Computer Name（计算机名）去查看一下。还必须给出一个 Web 服务器管理员（也就是用户本人）的电子邮件地址并决定是否要使用 80 号端口。

安装工作完成后，Apache 将自动配置为 Windows NT/2000/XP 下的一项系统服务并立刻启动运行。将在 Windows 任务条的左半边看到一个代表着 Apache 工作状态的小图标（一个表明这个程序正在运行的绿色小箭头）。可以利用这个图标去启动和停止 Apache 的运行。

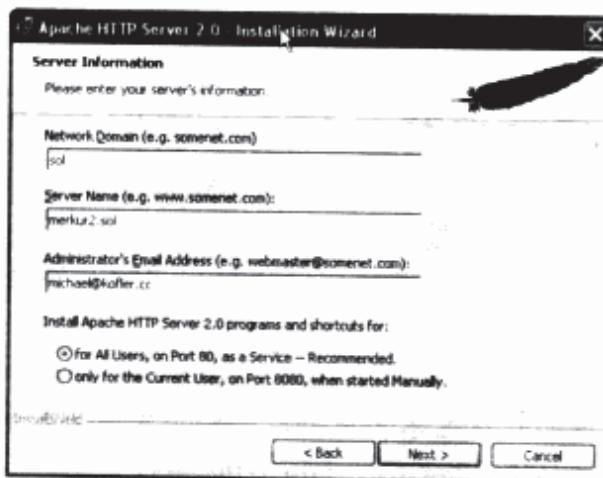


图 2-1 安装 Apache 软件

3. 启动和停止Apache

可以通过菜单命令 Programs（程序）| Apache HTTP Server（Apache HTTP 服务器）| Control Apache Server（控制 Apache 服务器）| Start（启动）和 | Stop（停止）启动和停止 Apache。执行菜单命令 Programs（程序）| Apache HTTP Server（Apache HTTP 服务器）| Configure Apache Server（配置 Apache 服务器）| Edit HTTPD.CONF（编辑 HTTPD.CONF 文件）将启动记事本 Notepad 程序，显示 Apache 的配置情况。

作为判断 Apache 是不是真的在运行的第一项测试，请在 Web 浏览器里打开 `http://localhost` 页面。应该能够看到 Web 浏览器的默认启动页面。接下来，请检查计算机里有没有以下两个文件/子目录（对 Apache 软件的默认安装而言）：Apache 的配置文件 `C:\Programs\Apache Group\Apache2\conf\httpd.conf`；Apache 存放网页文件的地方 `C:\Programs\Apache Group\Apache2\htdocs`。

2.2.2 安装 MySQL 5.0

安装 MySQL 5.0 的 ZIP 压缩文档可以在 `http://dev.mysql.com/downloads/mysql/5.0` 处找到。这份压缩文档只包含一个名为 `setup.exe` 的文件。运行 `setup.exe` 文件，屏幕上将出现一个安装助手对话框。除了把“安装类型”（Setup Type）设置为 Typical（典型安装）以外，这个对话框里的其他选项一般不需要修改。

在把 MySQL 服务器安装到子目录 `C:\Programs\MySQL\MySQL Server 5.0` 之后，安装程序会询问是否想为 `mysql.com` 程序创建一个账户。一般来说，可以单击 Skip Sign-Up 选项跳过这一步骤。

1. MySQL 配置助手

MySQL 配置助手非常有用，它会在安装工作结束后自动运行（如图 2-2 所示）。如有必要，也可以通过菜单命令 Programs（程序）| MySQL | MySQL Server 5.0 | MySQL Server Instance Config Wizard 启动这个助手。这个助手提供了两项功能：一是创建 MySQL 配置文件；二是给 MySQL 服务器设置一个密码。

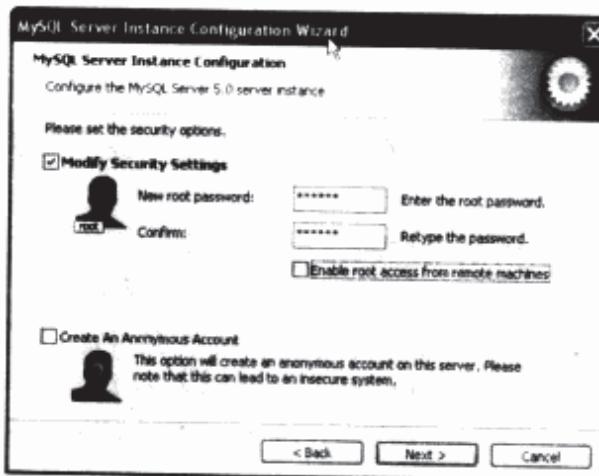


图 2-2 MySQL 配置助手

下面是在选择 Detailed Configuration（详细配置）时 MySQL 配置助手需要设置的一些内容。（请慎重做出选择。如果有不明白的地方，选择 MySQL 配置助手推荐的默认配置是最保险的。）

- **Server Type（服务器类型）**。这里有 Developer、Server 和 Dedicated Server 等几种选择，它们决定着 MySQL 服务器将试图保留多少主内存。主内存越大，服务器的速度就越快，但对其他程序的影响也会越大。对 Web 开发者来说，Developer 选项是最佳选择。（如果曾做过一些性能测试，就会知道 Server 或 Dedicated Server 选项最适用于大数据库）。
- **Database Usage（数据库用途）**。这里有 Multifunctional、Transactional、Only 和 Non-transactional Only 等几种选择，它们决定着 MySQL 服务器将支持哪几种数据表类型（MyISAM、InnoDB 或两种都支持）。默认选择 Multifunctional，用户可以根据自己的具体情况改变。
- **InnoDB Tablespace Setting（InnoDB 表空间设置）**。应该在这里为 InnoDB 数据库文件指定一个存放地点（它们的默认存放地点是 MySQL 安装目录）。

- **Concurrent Connections (并发连接)**。这个参数决定了同时打开的数据库连接的数量。对于一个运行在一台用于软件开发的计算机上的 MySQL 服务器来说，这个数字一般不会太大，这时候选择 **Decision Support** (20 个连接) 应该没什么问题。但对那些在某个热门网站上全速工作的 MySQL 服务器来说，并发连接的个数可能非常大，这时应该选择 **Online Transaction Processing** 选项。
 - **Enable TCP/IP Networking (激活 TCP/IP 组网功能)**。在 Windows 环境里，MySQL 服务器与应用程序的通信有两种方式：一是通过命名管道 (named pipe)；二是通过网络协议 TCP/IP。MySQL 配置助手给出的推荐配置是激活 TCP/IP 并使用 3306 号端口。在这里，如果没有特殊理由，应该接受 MySQL 配置助手的推荐配置。
 - **Default Character Set (默认字符集)**。在把文本数据保存到数据库里时，MySQL 支持现有的各种字符集。但如果在数据库的设计阶段没有明确地选择另外一种字符集，MySQL 会使用默认字符集来处理文本数据（这句话的隐含意思是：这项设置对今后的选择并无限制作用）。因为本书的所有示例使用的都是 *latin1* 字符集，所以选择它作为默认字符集比较好。
 - **Install as Windows Service (安装为 Windows 服务)**。MySQL 服务器可以作为一个*.exe 文件或者是一项 Windows 服务来启动。后一种安排比较方便和安全。这里接受 MySQL 配置助手推荐的默认设置。
 - **Security Options (安全选项)**。这或许是最重要的一个配置了，需要决定允许哪些人来连接 MySQL 服务器以及是否必须正确输入一个密码才能建立连接。图 2-2 中的设置是最安全的(也是笔者最推荐的)：*root* 用户是唯一的用户，而且必须正确输入密码才能连接数据库。该 *root* 用户将成为数据库的系统管理员，并且只允许他在本地计算机上登录（不允许他从网络里的另一台计算机登录）。除此之外，没有任何匿名账户（这里所说的“匿名账户”指的是无须密码就可以连接数据库的账户）。
- 当然，作为 MySQL 数据库的系统管理员，随时都可以创建新的用户并允许他们访问 MySQL 数据库，也可以随时创建新的 MySQL 数据库供用户使用。

最后，为了测试 MySQL 安装是否成功，使用菜单命令 Programs (程序) | MySQL | MySQL Server 5.0 | MySQL Command Line Client 启动 MySQL 的命令行解释器程序 mysql.exe，然后输入在配置 MySQL 服务器时设置的密码。如果一切顺利，应该在输入窗口里看到 MySQL 的输入提示符。接下来，执行 status 命令，应该看到如图 2-3 所示的结果。



```

MySQL Command Line Client
Enter password: welcome
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 5.0.3-beta-nt
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> status
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe Ver 14.9 Distrib 5.0.3-beta-nt for Win32 (ia32)

Connection id:          2
Current database:       -
Current user:           root@localhost
SSL:                   Not in use
Using delimiter:        ;
Server version:         5.0.3-beta-nt
Protocol version:        10
Connection:              localhost via TCP/IP
Server characterset:    latin1
Db:                     -
Client characterset:    latin1
Conn. characterset:     latin1
TCP port:               3306
Uptime:                 26 sec

Threads: 1 Questions: 12 Slow queries: 0 Opens: 0 Flush tables: 0 Open tables: 1 Queries per second avg: 0.000
mysql>

```

图 2-3 测试 MySQL 软件是否安装成功

2. MySQL服务：手动设置、卸载、启动和停止

安装过程会把 MySQL 默认地设置为一项 Windows 服务，让它在开机时自动启动。如果没有特殊的原因，最好不要改变这种配置。但万一需要改变这一配置，可以参考以下建议：

- **MySQL System Tray Monitor(MySQL 系统托盘监控器)**：执行菜单命令 Start(开始) | Programs (程序) | MySQL System Tray Monitor 将启动一个小程序，这个程序将在 Windows 条的右半边（显示系统时间的地方）显示一个小图标。通过这个图标，可以用鼠标右键启动和停止 MySQL 服务。
- **MySQL 系统管理员**：MySQL 数据库系统的系统管理员可以利用本书第 5 章介绍的 Service Control (服务控制) 模块来设置、卸载、启动和停止 MySQL 服务。
- **Windows System Control**：在 Windows 系统上，菜单命令 Settings (设置) | Control Panel (控制面板) | Administrative Tools (系统管理工具) | Service (服务) 将打开一个用来管理各项 Windows 服务的窗口。通过这个窗口，不仅可以启动和停止各项服务，还可以完成许多其他的操作（如图 2-4 所示）。

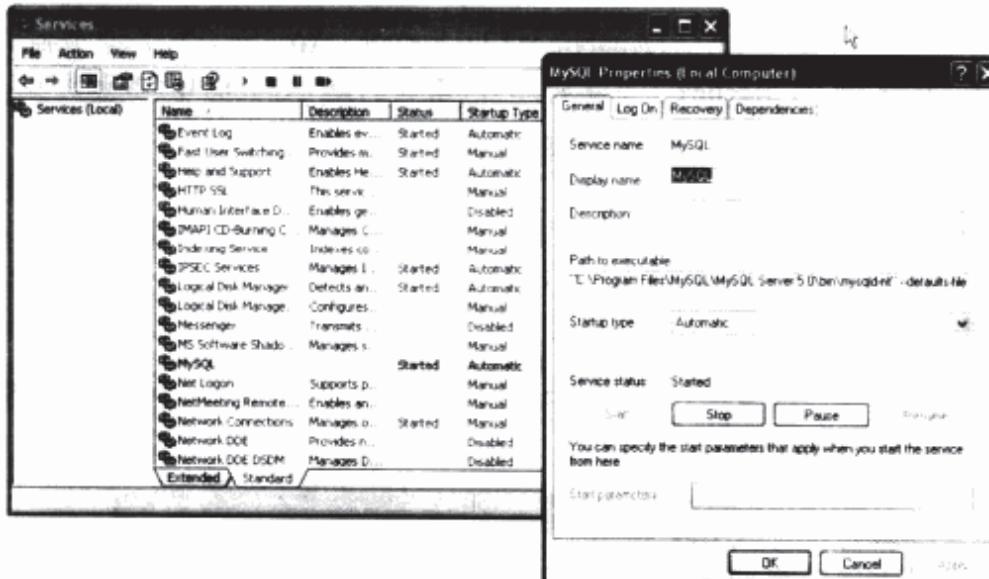


图 2-4 管理 Windows 服务

- **命令**：最后，还打开一个输入请求窗口去执行这些系统管理任务。如果想把 MySQL 安装为一项 Windows 服务，执行下面这两条命令即可。mysqld 命令将以 MySQL 为服务名安装好这项服务，这条命令必须全部写在同一行上：

```
> cd C:\Programs\MySQL\MySQL Server 5.0\bin
> mysqld --install MySQL
      --defaults-file="C:\Programs\MySQL\MySQL Server 5.0\my.ini"
```

上述命令将把 MySQL 安装为一项 Windows 服务，但不会立刻启动它。在这以后，这项服务将在系统开机时自动启动、在系统关机时自动停止。如果想以手动方式启动或停止这项服务，可以执行下面两条命令：

```
> net start MySQL
> net stop MySQL
```

如果想卸载这项服务，可以使用 mysqld 命令的--remove 选项，如下所示：

```
> mysqld --remove MySQL
```

2.2.3 安装 PHP 5.0

可以在 <http://www.php.net/downloads.php> 网址处找到两种格式的 PHP 最新版本：一种是 ZIP 压缩文档，一种是一个安装程序。安装程序必须在 IIS 下运行，因为本书使用 Apache 作为 Web 服务器，所以应该下载 ZIP 压缩文档。

下面是 PHP 5.0 的安装步骤：

(1) 把 ZIP 文档的内容解压缩到选定的某个子目录里（以下讨论将假设这个目录是 C:\php5）。

(2) 把下面代码中的黑体字部分插入到 Apache 服务器的配置文件 httpd.conf 里去。在 httpd.conf 文件里插入这些内容的最佳做法是增加一条 *LoadModule* 指令，该文件通常会有许多条这样的指令。

```
# Changes in C:\Programs\Apache Group\Apache2\conf\httpd.conf
...
LoadModule php5_module "c:/php5/php5apache2.dll"
AddType application/x-httpd-php .php
PHPIniDir "C:/php5"
...
```

(3) 复制 C:\php5\php.ini-recommended 文件，把新文件命名为 C:\php5\php.ini。

(4) 执行菜单命令 Programs (程序) | Apache HTTP Server (Apache HTTP 服务器) | Control Apache Server (控制 Apache 服务器) | Restart (重新启动)，重新启动 Apache 服务器。只要对 php.ini 或 httpd.conf 进行了改动，就应该重新启动 Apache，这是一个原则。

以上步骤将把 PHP 安装为 Apache 的一个扩展模块，并让它随 Apache 一起启动。如果想检查一下 PHP 是否安装成功，可以先用某个文本编辑器在子目录 C:\Programs\Apache Group\Apache2\htdocs 里创建一个名为 phptest.php 的文件，并写入以下内容：

```
<?php phpinfo(); ?>
```

然后用 Web 浏览器打开 <http://localhost/phptest.php> 页面。如果一切顺利，应该看到如图 2-5 所示的 PHP 状态信息。

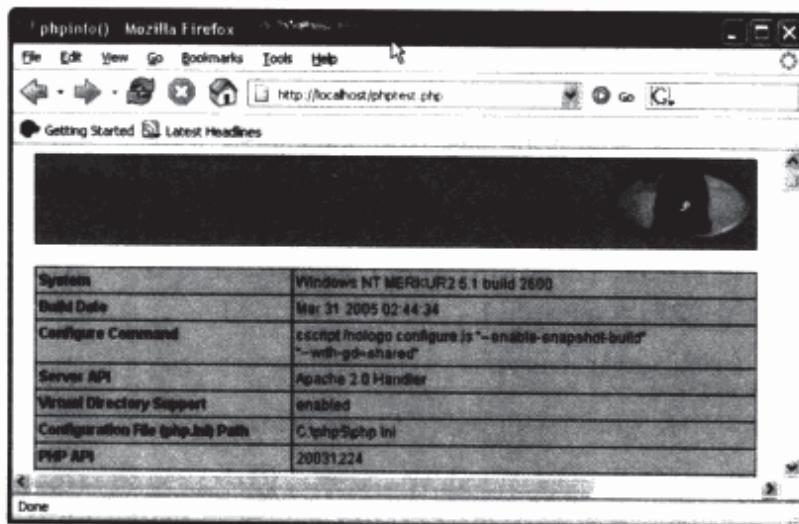


图 2-5 PHP 测试页面

配置 PHP 的 MySQL 扩展接口

作为安装 PHP 的最后一个步骤，必须把 PHP 配置成能够访问 MySQL 服务器。PHP 必须能够找到随 PHP 一起安装的两个 DLL 文件。必须依次完成以下步骤。

(1) 把 C:\php5\libmysql.dll 文件复制到 Windows 安装目录, 该目录通常是 C:\Windows 或 C:\Winnt。

(2) 用文本编辑器 (如记事本 Notepad) 打开 php.ini 文件, 修改下面的黑体字部分或插入它们:

```
; changes in c:\php5\php.ini
...
extension_dir = "c:/php5/ext"
extension = php_mysql.dll
extension = php_mysqli.dll
```

菜单命令 Edit (编辑) | Find (查找) 可以帮助用户快速找到这些行。请注意, 黑体字部分里的分隔符是斜线字符 (“/”) 而不是反斜线字符 (“\”)。在 extension = . . . 开始的语句前绝不应该有分号, 否则那条语句就将被当作一行注释而不是一条可执行语句。

(3) 重新启动 Apache 服务器, 使做出的修改生效。这可以通过菜单命令 Programs (程序) | Apache HTTP Server (Apache HTTP 服务器) | Control Apache Server (控制 Apache 服务器) | Restart (重新启动) 来完成。

如果遇到问题, 请再次加载 PHP 测试页面 (如图 2-5 所示)。首先检查 Configuration File (php.ini) Path (第 6 行) 给出的文件名是不是 C:\php5\php.ini。如果不是, 把刚才编辑好的 php.ini 复制到这个测试页面的指定存放地点。(根据笔者的经验, 最有可能出现的配置错误就是把 PHP 的配置文件放错了地方, 而这将导致做出的修改无法生效。)

接下来, 仔细检查 PHP 测试页面里的其他内容是不是全都正确无误。现在应该在这个测试页面里看到对 mysql 和 mysqli 接口参数的描述 (如图 2-6 所示)。

mysql		
Active Persistent Links	0	
Active Links	0	
Client API version	4.1.7	

mysqli		
Client API version	4.1.7	

图 2-6 PHP 测试页面上对 mysql 和 mysqli 接口的描述信息

2.2.4 安装 Perl

有一家名叫 ActiveState 的公司把 Perl 软件的最新 Windows 版本打包在它们的 ActivePerl 软件包里,

可以从 www.activestate.com 网站免费下载 MSI 文件格式的 ActivePerl 软件包。(还可以在这个网站上找到许多与 Perl 有关的商业化产品。) 在 Windows ME 和 Windows 2000 系统上, 只需双击 MSI 文件就可以开始安装工作。如果使用的 Windows 版本比较“古老”, 可能需要先安装 Windows 安装程序才能采用双击 MSI 文件的办法来安装其他软件。(Windows 安装程序的下载链接可以在 www.activestate.com 网站上找到, 最快捷的办法是用关键字 *System Requirement* 搜索一下。)

1. 测试Perl是否安装成功

安装工作结束后, 如果想检查一下 Perl 是否安装成功, 创建一个名为 test.pl 的测试文件, 并在这个文件里写入以下内容:

```
#!/usr/bin/perl -w
print "Hello world!\n";
print
Please hit Return to end the program!";
$wait_for_return = <STDIN>;
```

然后, 双击这个文件执行。这个程序将在命令窗口里显示“Hello World!”, 并在按回车键时结束运行。如果双击这个文件没有效果, 则打开一个输入请求窗口, 然后输入并执行以下命令(别忘了把这条命令中的 directory_with_my_perl_files 替换为真实的路径):

```
C:\> perl C:\directory_with_my_perl_files\test.pl
```

2. 安装Perl语言的MySQL支持组件

安装了 Perl 并不意味着已经完成了全部工作。Perl 现在还不能与 MySQL 通信, 还需要再安装以下两个 Perl 扩展模块:

- DBI: 英文 *database interface* (数据库接口) 的缩写。DBI 模块是一种通用的数据库编程接口。
- DBD-MySQL: 英文 *database driver* (数据库驱动程序) 的缩写。DBD-MySQL 模块是对于 MySQL 数据库系统的 DBD。

这两个模块都可以用 ppm 程序(Perl 的软件包管理器)来安装, 该程序已被包括在 ActivePerl 软件包里了。这个程序可以在输入请求窗口里启动。它要求计算机已经连上了因特网。下面是用 ppm 程序安装上述两个模块的具体操作步骤(以 ActivePerl 5.8.6 版本为例):

```
C:> ppm
PPM - Programmer's Package Manager version 3.1.
Type 'help' to get started.
PPM> install dbi
Install 'dbi' version 1.46 in ActivePerl 5.8.6.811.
Downloaded 540637 bytes.
Extracting 73/73: blib/arch/auto/DBI/Driver_xst.h
Installing C:\Programs\Perl\site\lib\auto\DBI\dbd_xsh.h
Installing C:\Programs\Perl\site\lib\auto\DBI\DBI.bs
...
Successfully installed dbi version 1.46 in ActivePerl 5.8.6.811.
PPM> install DBD-Mysql
Install 'DBD-Mysql' version 2.9003 in ActivePerl 5.8.6.811.
Downloaded 178968 bytes.
Extracting 17/17: blib/arch/auto/DBD/mysql/mysql.lib
Installing C:\Programs\Perl\site\lib\auto\DBD\mysql\mysql.bs
Installing C:\Programs\Perl\site\lib\auto\DBD\mysql\mysql.dll
...
```

请注意, DBD-MySQL 2.9003 与最新的 MySQL 版本不兼容。DBD-MySQL 2.9003 使用的 MySQL 客户软件开发库的版本比较老, 所以只能与 MySQL 4.1 或 5.0 配合使用, 并且还需要事先把 MySQL 服务器配置成使用老式的、不够安全的身份验证机制才行(对应的 MySQL 配置选项是 old-passwords;

详见第 11 章)。

还好,现在可以从因特网上找到 DBD-MySQL 模块的最新版本。以下命令将把最新版本的 DBD-MySQL 模块安装到系统上:

```
ppm> remove DBD-Mysql
Remove 'DBD-Mysql' version 2.9003 from ActivePerl 5.8.6.811.

...
ppm> install http://theoryx5.uwinnipeg.ca/ppms/DBD-mysql.ppd
Install 'DBD-mysql' version 2.9005_3 in ActivePerl 5.8.6.811.
=====
Downloaded 625824 bytes.
Extracting 37/37: blib
Installing C:\Programs\Perl\site\lib\auto\DBD\mysql\mysql.bs
Installing C:\Programs\Perl\site\lib\auto\DBD\mysql\mysql.dll
...
Successfully installed DBD-mysql version 2.9005_3 in ActivePerl 5.8.6.811.
```

提示 如果是使用HTTP代理上网的,就必须在执行ppm安装命令之前执行下面这条命令(别忘了把命令中的 myproxy.com:8080 替换为真实的代理地址): set http_proxy = http://myproxy.com:8080/。

此外,如果计算机上安装了防火墙,还需要确认perl.exe程序是否有权访问因特网。如有必要,修改或增加一条防火墙规则。

可以用search pattern命令来搜索可供选用的Perl软件包。例如, search mysql命令将把文件名里包含mysql字样(不管它是出现在文件名的中间、开头还是末尾)的Perl软件包全部查找出来。

DBD-MySQL 模块将假设用户的计算机里已经安装了 MySQL 客户软件开发库。这个函数库是 MySQL 软件包的一个组成部分,它的默认存放地点是 C:\Programs\MySQL\MySQL Server 5.0\bin\libmysql.dll 文件。为了让 Perl 能够找到这个库文件,必须把这个库文件复制到 Windows 安装目录里去或是在环境变量 *PATH* 里增加一个新路径 C:\Programs\MySQL\MySQL Server 5.0\bin\ (详见第 4 章)。

将在第 16 章详细介绍如何测试 Perl 环境和如何使用 Perl 语言编写 MySQL 应用程序。

2.3 在 SUSE Linux 9.3 系统上安装 MySQL 和相关软件

SUSE Linux 9.3 已经包含了 Apache 2、PHP 4.3/PHP 5、Perl 5.8 和 MySQL 4.1 等软件。但为了与本书里的示例保持兼容,需要安装 PHP 5 和 MySQL 5。在接下来的两个小节里,将先介绍如何安装 Apache 2、PHP 5 和 Perl 5.8,再介绍如何安装从 dev.mysql.com 网站下载回来的 MySQL 软件包最新版本。

2.3.1 安装 Apache 2、PHP 5 和 Perl

在默认的情况下,SUSE Linux 9.3 将把 PHP 4.3 和 MySQL 4.1 安装到计算机里,但我们需要的是 PHP 5 和 MySQL 5.1,所以要对 SUSE 的默认安装步骤做一些改变。这里有两个关键之处:一是要按照以下步骤来安装 PHP 5(注意:不是 PHP 4.3);二是不要让 SUSE 发行版本安装它自带的 MySQL 软件包,以免以后还得卸载它们。

(1) 选择 Apache 模块。执行菜单命令 System(系统)|YaST 开始安装。执行菜单命令 Software(软件)|Install Software(安装软件)启动模块,把 Filter(过滤器)设置为 Selections(手动选择),

然后在下拉列表里选择“Simple Webserver with Apache2”选项。

(2) 弃选 PHP 4 软件包。把 Filter (过滤器) 改为 Search (搜索), 关键字选为 *php4*。把搜索出来的软件包全部弃选掉。

(3) 选择 PHP 5 软件包。把搜索关键字改为 *php5*, 把搜索出来的软件包——除名为 *php5-devel* 的软件以外——全部选中。

(4) 弃选 MySQL 软件包。把搜索关键字改为 *mysql*, 把搜索出来的软件包——除名为 *mysql*、*mysql-client* 和 *mysql-shared* 的那 3 个软件以外——全部选中(那 3 个软件是 MySQL 4.1 的组件, 将在下一小节介绍如何安装 MySQL 5.0。)

(5) 选择 Perl 语言的 MySQL 驱动程序。在以 *mysql* 作为关键字的搜索结果里, 会看到一个名为 *perl-DBD-mysql* 的软件包。这个软件包是 Perl 访问 MySQL 所必需的(Perl 本身已经随 SUSE Linux 9.3 安装好了)。

(6) 解决软件包之间的依赖关系。单击 Check Dependencies (检查依赖关系) 按钮。YaST 将显示一条警告信息说 *php5-mysql*、*php5-mysqli* 和 *perl-DBD* 等软件包需要 *mysql-shared* 软件包。选择 Ignore this Conflict (忽略此冲突)。(将在稍后手动安装 MySQL 5.0 时解决这个冲突。)

(7) 查看汇总信息。现在, 为了检查上述选择是否准确无误, 把 Filter (过滤器) 改为 Summary (汇总)。YaST 会把它准备安装的软件包(名字前面有一个带对勾的小方框)和不安装的软件包(名字前面有一个“do not enter”标志)列成清单显示出来; 如图 2-7 所示。

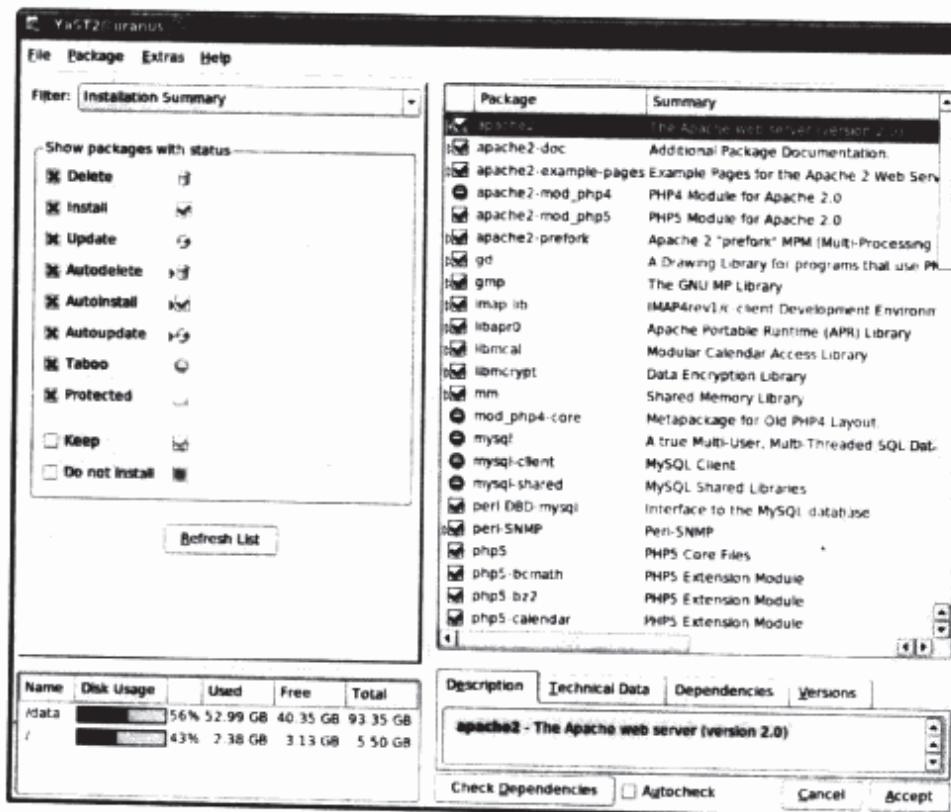


图 2-7 用 SUSE Linux 发行版本中的 YaST 工具来安装软件包

(8) 安装软件包。单击 Accept (接受) 按钮, 被选中的所有软件包就将被依次安装到计算机里。YaST 可能会显示一个对话框说还需要安装其他的软件包(如 *perl-DBD*)。

在默认的情况下, Apache 服务器不会在系统开机时自动启动。它需要以 *root* 用户的身份执行以下命令才能启动:

```
root# /etc/init.d/apache2 start
```

如果想让 Apache 服务器在系统开机时自动启动，就要多执行一条命令：

```
root# insserv apache2
```

并给出两个文件/子目录路径：配置文件，/etc/apache2/*；网页文件，/srv/www/htdocs。

1. 测试Apache和PHP是否安装成功

如果想测试 Apache 服务器是否真的在运行，用 Web 浏览器打开页面 `http://localhost`。应该看到一个 Apache 测试页面。

如果想检查 PHP 是否工作正常，先用一个文本编辑器在/srv/www/htdocs 子目录里创建一个名为 `phptest.php` 的文件，并把以下内容写入该文件：

```
<?php phpinfo(); ?>
```

然后用一个 Web 浏览器打开 `http://localhost/phptest.php` 页面。应该可以看到 PHP 测试页面（如图 2-8 所示）。

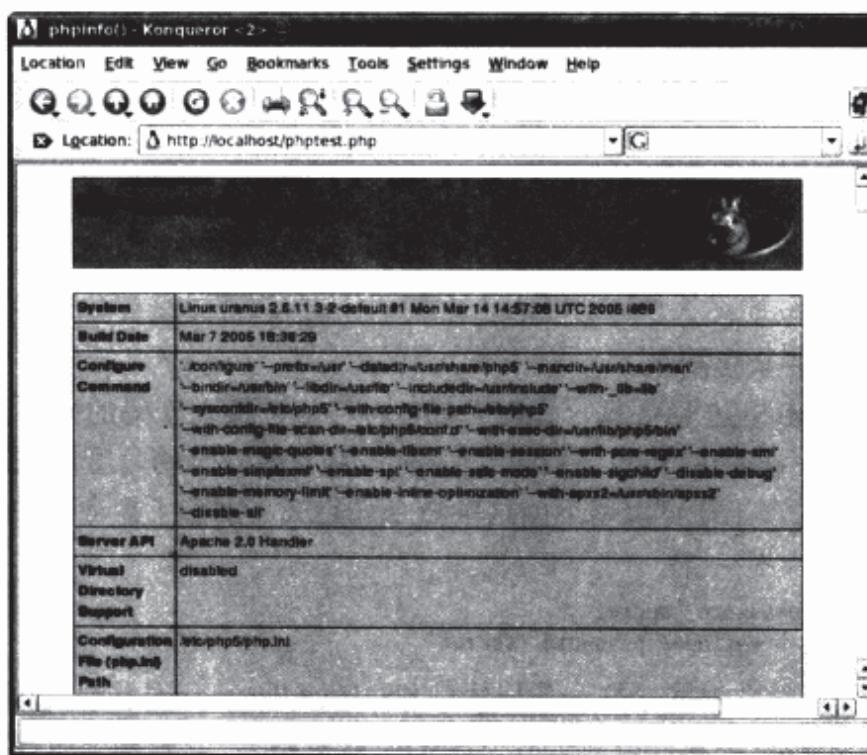


图 2-8 PHP 测试页面

2. 测试Perl是否安装成功

将在第 16 章详细介绍如何测试 Perl 环境和如何使用 Perl 语言编写 MySQL 应用程序。

2.3.2 安装 MySQL 5.0

为了安装 MySQL 服务器，首先要从以下站点下载 RPM 文件格式的最新版本：<http://dev.mysql.com/downloads/mysql/5.0.html>。RPM 是 Linux 软件的一种常见打包格式。

表 2-1 里的软件包只是安装 MySQL 服务器、MySQL 客户和 MySQL 共享库最起码的必需品。安装命令如下所示，因为页面宽度的问题，用反斜线字符 (\) 把这条命令分成了 3 行。安装成功后，系统将提示为 MySQL 服务器的 `root` 用户设置一个密码：

```
root# rpm -i MySQL-server-5.0.n-0.i386.rpm \
      MySQL-client-5.0.n-0.i386.rpm \
      MySQL-shared-5.0.n-0.i386.rpm
PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
To do so, start the server, then issue the following commands:
/usr/bin/mysqladmin -u root password 'new-password'
/usr/bin/mysqladmin -u root -h uranus password 'new-password'
See the manual for more instructions.
```

表 2-1 RPM MySQL 软件包

软件包名称	说 明
MySQL-server-n.rpm	MySQL 服务器
MySQL-client-n.rpm	MySQL 客户程序 (mysql、mysqladmin 等)
MySQL-shared-n.rpm	其他程序访问 MySQL 时需要的共享库
MySQL-devel-n.rpm	用 C 语言编写 MySQL 应用程序时需要的函数库和头文件
MySQL-embedded-n.rpm	一个特殊版本的 MySQL 服务器, 它可以集成到其他程序里去 (只适用于 C 语言程序员)
MySQL-bench-n.rpm	用来进行各项性能测试的脚本
MySQL-Max-n.rpm	MySQL 服务器的 Max 版本, 它支持更多的数据表格式, 并包含着一些尚处于开发和测试阶段的新功能

MySQL 服务器在安装后会立刻启动运行。如果想让数据库服务器在系统开机时自动启动运行, 执行下面这条命令:

```
root# insserv mysql
```

为 MySQL 服务器建立安全机制

对于一个刚安装好的 MySQL 服务器, 任何人都可以在无需输入密码的情况下登录为它的 *root* 用户并不受限制地对所有的数据库进行读、写和删除操作。从系统安全的角度考虑, 为 *root* 用户设置一个密码是绝对有必要的。这项工作可以用下面这条命令来完成 (别忘了把命令中的 *hostname* 替换为计算机的真实名字):

```
root# mysqladmin -u root password 'secret'
root# mysqladmin -u root -h hostname password 'secret'
```

在执行这条命令时, 系统可能会返回这样一条出错信息: *Host 'hostname' is not allowed to connect to this MySQL server* (主机 '*hostname*' 不允许访问这台 MySQL 服务器)。如果发生这种情况, 就说明在安装 MySQL 期间输入的主机名 (它将被存入 Access 数据库) 没有域名部分。这个问题的解决办法是: 重新启动 mysql 程序, 用 *UPDATE* 命令刷新 *mysql.user* 数据表, 然后用 *mysqladmin* 命令修改第二个 *root* 密码。下面是完成上述过程所需要执行的命令。当然, 必须把命令中的 *uranus.sol* 和 *uranus* 替换为计算机的真实名字, 一次带有域名部分, 另一次不带域名部分。

```
root# mysql -u root -p
Enter password: *****
mysql> USE mysql;
mysql> UPDATE user SET host="uranus.sol" WHERE host="uranus";
mysql> FLUSH PRIVILEGES;
mysql> exit
root# mysqladmin -u root -h uranus password 'secret'
```

对 MySQL 访问控制机制的详细讨论请参见本书的第 11 章。MySQL 的 Access 数据库里主机名不正确的问题与 Linux 系统的网络配置有关, 第 11 章对此做了详细解释。

2.4 在 Red Hat Enterprise Linux 4 系统上安装 MySQL 和相关软件

Red Hat Enterprise Linux 4 已经包含了 Apache 2、PHP 4.3、Perl 5.8 和 MySQL 4.1 等软件。但为了与本书的示例保持兼容，需要安装 PHP 5 和 MySQL 5。根据这一要求，将在接下来的几个小节里逐步地带领大家完成自定义安装。

- 安装 Apache2（Red Hat 发行版本自带）。
- 安装 MySQL 5（下载自 dev.mysql.com 网站）。
- 编译 PHP 5。
- 安装 Perl 5.8（Red Hat 发行版本自带）。

以下步骤假设用户在安装 Red Hat Enterprise Linux 4 的时候既没有安装任何一种上述软件，也没有安装 PHP 4.3 和 MySQL 4.1。如果已经安装了它们，使用 `system-config-packages` 或 `rpm -e` 命令卸载掉。

笔者的安装已在 Red Hat Enterprise Linux 4 ES 环境下通过了测试。这里给出的信息应该同样适用于其他的 RHEL 版本和（增加几条限制之后的）Fedora（Fedora Core 4 已经包含了 PHP 5，无需再由用户去编译它）。

2.4.1 安装 Apache 2

Apache 2 要用 `system-config-packages` 程序安装，启动该程序的菜单命令是 Application（应用程序）| System Settings（系统设置）| Add/Remove Application（添加/删除软件）。窗口出现后，先选中 Web Server 选项，再单击它旁边的 Details 链接。在弹出的对话框里，取消选中 `php-ldap` 和 `php` 软件包（如图 2-9 所示），然后单击 Update 按钮开始安装。

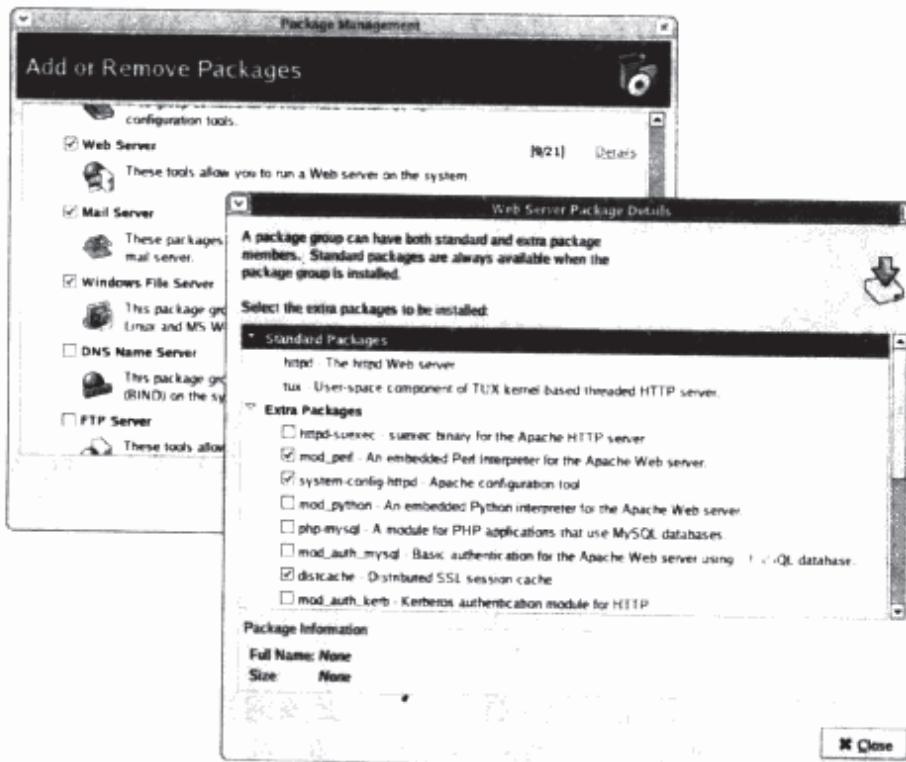


图 2-9 在 Red Hat Enterprise Linux 4 系统上安装 Apache

安装工作结束后，执行下面这条命令即可启动 Apache：

```
root# /etc/init.d/httpd start
```

如果想让 Apache 服务器在系统开机时自动启动，就要多执行两条命令：

```
root# chkconfig --add httpd
root# chkconfig --level 35 httpd on
```

如果想测试 Apache 的 Web 服务器是否真的在运行，请用 Web 浏览器打开页面 `http://localhost`。应该可以看到一个 Apache 测试页面。还需要给出两个文件/子目录路径：配置文件，`/etc/apache2/*`；网页文件，`/srv/www/htdocs`。

2.4.2 安装 MySQL 5

MySQL 软件包的安装步骤与 SUSE Linux 下的情况相似。简单地说，先从 `dev.mysql.com` 网站下载包含着 MySQL 服务器、MySQL 客户和 MySQL 共享库的软件包，再用 `rpm -i` 命令安装它们即可。

```
root# rpm -i MySQL-server-5.0.n-0.i386.rpm \
      MySQL-client-5.0.n-0.i386.rpm \
      MySQL-shared-5.0.n-0.i386.rpm
```

用 Red Hat 发行版本安装的 MySQL 服务器会在每次系统开机时自动启动运行，因而没必要使用前面小节里给出的 `insserv` 命令（顺便说一句，在 Red Hat 发行版本里，与 `insserv` 命令等价的命令是 `-addmysql`）。

接下来，用下面的命令给 MySQL 数据库系统的 `root` 用户设置一个密码。当然，应该把命令里的 `hostname` 替换为计算机的真实名字。

```
root# mysqladmin -u root password 'secret'
root# mysqladmin -u root -h hostname password 'secret'
```

MySQL 5.0.3 与 RHEL 4 默认安装的 SELinux 组件不兼容，这将导致许多命令（例如 `SHOW DATABASES`）无法使用。目前尚不清楚 MySQL 服务器的新版本或是 SELinux 策略的新版本能不能解决这个问题。如果遇到了这方面的问题，就必须为了 MySQL 服务器而禁用 SELinux，具体做法是：以 `root` 用户的身份执行 `system-config-securitylevel` 命令，进入 SELinux 对话框，在 SELinux Policy (SELinux 策略) 部分找到并选中 SELinux service protection | Disable SELinux protection for MySQL daemon 选项。然后，重新启动 MySQL 服务器让这个改动生效。

2.4.3 编译 PHP 5

1. 安装开发工具

在开始编译之前，需要把必要的开发工具（编译器、`make` 程序等）安装好。这些开发工具的安装工作说来并不复杂：启动 `system-config-packages` 程序，选好有关的软件包，然后安装软件包即可。

不过，在笔者的测试系统上，这项看似轻而易举的工作带来了许多麻烦：笔者以前用 `up2date` 程序对系统进行的一次升级与这些开发软件包发生了明显的冲突，而 `system-config-packages` 程序又无法解决软件包之间的依赖关系问题。

经过一番摸索之后，终于用一条 `up2date` 命令把所有必要的程序都安装好了。那几个软件包的最新版本以及它们所依赖的其他软件包都是从因特网上安装的。该命令需要提前安装好 `up2date` 工具并能够高速上网。根据计算机里已经安装了哪些软件包，或许还需要在下面这条命令里再增加几个其他的软件包：

```
root# up2date gcc g++ make autoconf gettext binutils bison flex \
libxml2 libxml2-devel libjpeg libjpeg-devel \
libpng libpng-devel libpng10 libpng10-devel \
gd gd-devel httpd-devel
```

注意，根据具体安装的 MySQL 版本挑选一个适用的 MySQL-devel 软件包和开发辅助文件，这个软件包可以从 dev.mysql.com 网站下载、用 rpm 命令来安装：

```
root# rpm -i MySQL-devel-5.0.n.i386.rpm
```

2. 下载并安装 PHP 5 的源代码

被打包为一个*.tar.bz2 文档的 PHP 5 源代码可以在下面的站点上找到：

```
http://www.php.net/downloads.php
```

接下来的工作将使用 /usr/local/src 子目录来完成。如果想以某个普通用户的身份来完成后面的操作，要趁现在这个机会改变这个子目录的属主，如下所示：

```
root# chown username.username/usr/local/src
```

现在，把 PHP 5 的源代码解压缩到这个子目录里：

```
user$ cd /usr/src/linux
user$ tar xjvf php-5.0.n.tar.bz2
user$ cd php-5.0.n
```

3. 编译 PHP 5

在正式编译 PHP 5 的源代码之前，还需要用 ./configure 命令为 PHP 设置一些编译选项。如果还想把其他一些 PHP 扩展模块包括到本次编译任务中来，要用相应的编译选项添加它们。下面给出的 configure 命令已经足以满足本书示例的需要。如果具体使用的 configure 命令能够写在一行，就不必非得用反斜线字符（“\”）把它分为好几行。

下面是对将要用到的一些 configure 命令选项的说明：-prefix 选项负责设置 PHP 5 的安装位置；-with-apxs2 选项用于把 PHP 5 编译为 Apache 2 的一个扩展模块，紧随其后的路径是该 Apache 模块的存放地点；-with-libxml-dir 选项给出了 XML 库的存放位置；类似地，-with-zlib-dir 选项给出了 zlib 库的存放位置。

-with-mysql 选项的意思是编译出来的 PHP 将使用传统的 mysql 接口（详见第 15 章）。紧随其后的路径是 MySQL 软件的安装目录，从 dev.mysql.com 网站下载的预编译 MySQL 版本都以 /usr 作为其安装目录。

-with-mysqli 选项的意思是把新的 mysqli 接口集成到 PHP 中（详见第 15 章）。紧随其后的文件是 MySQL-devel 软件包的组成部分之一。这涉及一个用来给出 MySQL 版本信息及其安装位置信息的脚本。

--with-xxx 和 -enable-xxx 选项用来激活 PHP 的各种附加功能。configure 命令还有许多其他的命令选项，用 ./configure -help 命令可以查看到一份比较完整的清单。下面是 configure 命令的一个用法示例：

```
user$ ./configure --prefix=/usr/local/php5 \
--with-apxs2=/usr/local/apache2/bin/apxs \
--with-libxml-dir=/usr/lib \
--with-zlib --with-zlib-dir=/usr/lib \
--with-mysql=/usr --with-mysqli=/usr/bin/mysql_config \
--with-jpeg-dir=/usr --enable-exif \
--with-gd --enable-soap --enable-sockets \
--enable-force-cgi-redirect --enable-mbstring
```

如果 configure 命令返回的出错信息说它未能找到某些程序或函数库，必须先用 up2date 命令安装好它们，然后再次执行 configure 命令。

把有关的编译选项都配置好以后，执行 make 命令开始编译。这个过程将花费好几分钟。如果在编译完成后又执行 configure 命令改变了某些编译选项，在再次执行 make 命令之前必须先用 make clean 命令删除前一次的编译结果。

```
user$ make
```

提示 在编译过程中，如果出现了很多提示 libmysql 库里有多重定义 (multiple-defined) 的出错信息，就说明 configure 命令生成的 Makefile 文件里有错误（这种问题在笔者进行的测试中曾多次发生，其具体原因尚不清楚）。

解决这个问题的办法是：在执行 make 命令之前，用一个文本编辑器打开 Makefile 文件，并在其 中找到 EXTRA_LIBS=... 语句。在这条语句里会发现 lmysqlclient 出现了两次。删除其中之一，保存 Makefile 文件，然后再次执行 make 命令。

最后，必须把新编译出来的 PHP 模块安装到一个能够让 Apache 找到的地方。这需要以 root 用户的身份执行 make install 命令：

```
root# make install
```

按照刚才的配置，PHP 5 将被安装到 /usr/local/php5 子目录里。用来存放配置文件 php.ini 的位置是 /usr/local/php5/lib/ 子目录。在默认的情况下，这个文件是不存在的——PHP 5 将按照它的默认配置来运行。与 PHP 有关的 *.ini 文件的模板可以在 /usr/local/src/php5.n 子目录里找到。

4. 改变Apache配置，重新启动Apache

接下来的工作是修改 Apache 服务器的配置文件 /etc/httpd/conf/httpd.conf，让 Apache 能够找到和使用 PHP 5 模块。修改工作很简单，用一个文本编辑器打开 httpd.conf 文件，把下面代码中的黑体字部分添加到里面即可：

```
# changes in /usr/local/apache2/conf/httpd.conf
...
LoadModule php5_module modules/libphp5.so
AddType application/x-httpd-php .php
...
### Section 2: ...
```

然后，重新启动 Apache 服务器，这样它才可以使用那个模块：

```
root# /etc/init.d/httpd restart
```

如果想测试 PHP 是否工作正常，在 /var/www/html/ 子目录里创建一个名为 phptest.php 的文件，并把以下内容写入该文件；这个文件对 Apache 账户 nobody 必须是可读的：

```
<?phpinfo(); ?>
```

现在，用 Web 浏览器打开 http://localhost/phptest.php 页面。应该看到如图 2-10 所示的画面。

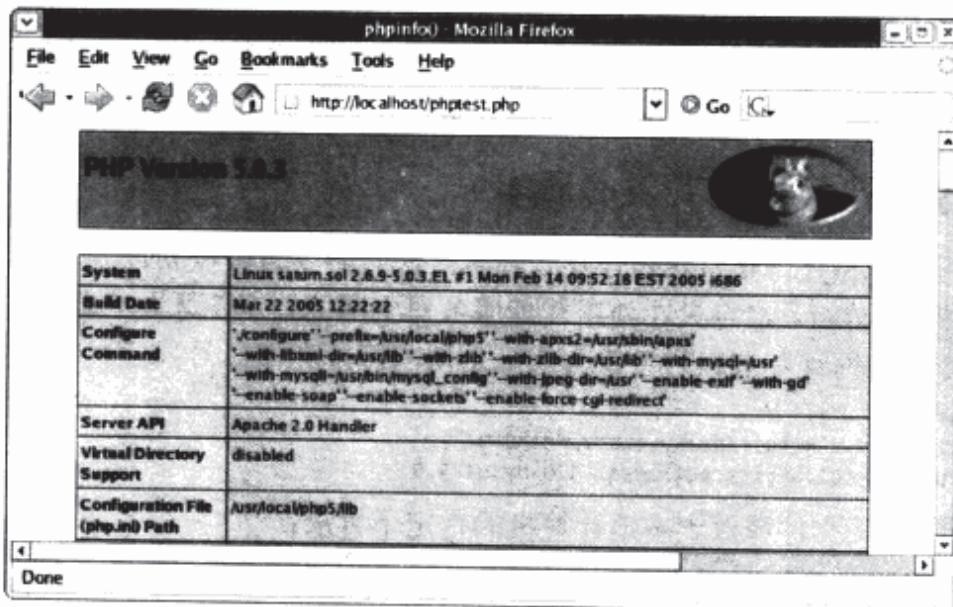


图 2-10 Red Hat Enterprise Linux 4 环境下的 PHP 测试页面

2.4.4 安装 Perl 5.8

在默认的情况下，Red Hat Linux 发行版本将自动安装 Perl 5.8，但 Perl 和 MySQL 通信所必需的 perl-DBI 和 perl-DBD-MySQL 模块不包括在内。其实 Red Hat Linux 发行版本已经收录了这些软件包，但问题是这里不应该使用 system-config-packages 程序来安装它们——那么做会把 MySQL 4.1 也安装到系统里去。这里将使用 rpm -i 命令来安装它们。这两个软件包在 Red Hat Enterprise ES 4 的第二张光盘上。插入光盘，然后执行以下命令：

```
root# cd /mnt/cdrom/RedHat/RPMS
root# rpm -i perl-DBI-*.rpm
root# rpm -i perl-DBD-MySQL-*.rpm
```

第 16 章将详细讨论如何测试 Perl 环境和如何使用 Perl 语言编写 MySQL 应用程序。

2.5 编译 MySQL 软件的开发者版本 (Linux)

本节是为那些打算自行编译 MySQL 软件最新测试版本的 MySQL 高手准备的。虽说在 Linux 环境里自行编译一个软件并不困难，但还是要提醒大家：这种做法只适用于特定场合以及那些不愿意浪费好几个月的时间去等待 MySQL 的下一个正式版的专家们。MySQL 5.0 系列每隔半年就会发布一个新的测试版本。最新的测试版本或许还不够稳定，但多少会修正一些错误和增加一些新功能。因此，有不少 MySQL 高手不愿意等待下一个正式版本，而是更喜欢自行编译最新的测试版本。

现在，如果计算机里已经安装了 MySQL 的一个老版本，使用 phpMyAdmin 或 mysqldump 工具为所有的数据库制作好备份。然后，终止 MySQL 服务器的运行（用/etc/init.d/mysql stop 命令）并卸载现有的 MySQL 安装。

2.5.1 安装 Bitkeeper

tar 软件包格式的测试版 MySQL 源代码在因特网上是找不到的。MySQL 公司采用了一种名为 Bitkeeper 的版本控制系统来管理 MySQL 测试版本的源代码。因此，应该先从 <http://www.bitmover.com/cgi-bin/download.cgi> 网站下载 Bitkeeper 并安装到计算机上。Bitkeeper 必须

经过注册才能使用，但注册是免费的。以下命令将把 Bitkeeper 安装到 /usr/local/bk 子目录：

```
root# chmod u+rwx bk-<version>-x86-glibc23-linux.bin
root# ./bk-<version>-x86-glibc23-linux.bin /usr/local/bk
```

2.5.2 下载 MySQL 软件的开发者版本

以下命令适用于 MySQL 5.0 系列的开发者版本。这些命令将假设 *user* 用户有权对 /usr/src 子目录进行写操作。注意，*export* 命令只在使用代理服务器上网的情况下才是必需的；如果真是如此，别忘了把 *export* 命令中的网址和端口号替换为实际使用的网址和端口号。

```
user$ cd /usr/src/
user$ export http_proxy="http://proxy.firma.de:8080/"
user$ bk clone bk://mysql.bkbits.net/mysql-5.0 mysql-5.0
```

下面两条命令将把源代码文件释放到本地计算机的指定子目录：

```
user$ cd /usr/src/mysql-5
user$ bk -r edit
```

在第一次启动 *bk* 程序的时候，它会对自己的许可证进行验证（按 Q 键停止显示许可证文本，再按 Y 键接受许可证条款）。通过许可证验证之后，*bk* 会详细地列出它曾复制、删除和替换过的文件。

2.5.3 编译 MySQL

以下命令将完成对 MySQL（带 InnoDB 支持）源代码的编译工作：

```
user$ aclocal; autoheader; autoconf; automake
user$ (cd innobase; aclocal; autoheader; autoconf; automake)
user$ ./configure
user$ make
root# make install
```

2.5.4 创建用来管理访问权限的 *mysql* 数据库

在第一次启动 MySQL 服务器之前，必须创建一个名为 *mysql* 的数据库，用于管理 MySQL 数据库系统的访问权限。以下命令将假设有关的数据库文件都存放在 /usr/local/mysql 子目录里，并且 MySQL 服务器使用的账户名是 *mysql*。（如果 MySQL 已经在计算机上运行，可以继续使用它的数据库。这时不要执行下面两条命令。）

```
root# ./scripts/mysql_install_db -ldata=/usr/local/mysql
root# chown -R mysql /usr/local/mysql
```

2.5.5 MySQL 配置文件和 Init-V 脚本

为了启动 MySQL 服务器，需要一个 Init-V 脚本。在 MySQL 软件的源代码文件当中，可以在子目录 support-files/mysql.server 下为这个脚本找到一个适当的模板文件。还需要一个 /etc/my.cnf 文件来配置各种 MySQL 参数，这个文件的模板也可以在 mysql.server 子目录下找到：

```
root# cp support-files/mysql.server /etc/init.d/mysql
root# cp support-files/my-medium.cnf /etc/my.cnf
```

现在，需要在这两个文件里做一些小改动，详见下面两段代码里的黑体字内容。先说 /etc/my.cnf 文件，它负责设置 MySQL 客户和 MySQL 服务器来进行本地通信的套接字。在默认情况下，MySQL 服务器将使用 /tmp/mysql.sock 作为套接字；在安装 PHP 之后，这个套接字通常需要改为 /var/lib/mysql/mysql.sock。

```
# Changes in /etc/my.cnf
...
[client]
socket = /var/lib/mysql/mysql.sock
...
[mysqld]
socket = /var/lib/mysql/mysql.sock
...
```

/etc/init.d/mysql 脚本负责启动 MySQL 服务器。需要在这个文件里给出 MySQL 服务器的二进制可执行文件的存放位置和 MySQL 数据库文件的存放位置。(如果 MySQL 已经在计算机上运行, 可以继续使用它的数据库。此时, 就必须把 *datadir* 变量设置为现有的数据库子目录。一般来说, 这个子目录是/var/lib/sys。)

```
# Changes in /etc/init.d/mysql
...
basedir=/usr/local
datadir=/usr/local/mysql
...
```

2.5.6 启动 MySQL 服务器

启动 MySQL 服务器需要执行以下命令:

```
root# /etc/init.d/mysql start
```

如果想让 MySQL 服务器在每次系统开机的时候自动启动运行, 需要执行下面两条命令之一:

```
root# insserv mysql          (适用于SUSE)
root# chkconfig -add mysql   (适用于Red Hat、Fedora、Mandrakelinux等)
```

2.6 配置 Apache

2.6.1 配置文件

几乎所有的 Apache 设置都是由配置文件 httpd.conf 完成的。这个文件的具体存放位置因操作系统而异:

- **Windows:** Programs\Apache Group\Apache2\conf\httpd.conf
- **Red Hat Enterprise Linux:** /etc/httpd/conf/httpd.conf/etc/httpd/conf.d/*.conf
- **SUSE Linux:** /etc/apache2/httpd.conf/etc/apache2/*.conf/etc/apache2/conf.d/*.conf/etc/sysconfig/apache2

在 Red Hat 和 SUSE 环境下, Apache 服务器的配置参数分散在几个文件里, 这些文件又分散在上面列出的子目录里。如果想修改某个参数, 最快捷的办法是利用 grep 程序把它所在的文件搜索出来。

SUSE 用户请特别注意, 有些*.conf 文件的内容会在 Apache 服务器每次启动后发生变化。因此, 如果需要改变 SUSE 环境下的 Apache 配置, 那就要么直接在/etc/sysconfig/apache2 文件里修改, 要么专门创建一个文件(比如/etc/apache2/httpd.conf.local)来保存改动。如果采用后一种办法, 为确保 Apache 服务器在启动时会读取配置文件里的内容, 还需要把它的文件名追加到 /etc/sysconfig/apache2 文件中的变量 APACHE_CONF_INCLUDE_FILES 的尾部。

注意 配置变化只有在Apache服务器重新启动后才会生效。在Windows系统上, 必须使用斜线字符(“/”)作为Apache配置文件里的分隔符, 不能使用反斜线字符(“\”)。

2.6.2 基本设置

这是一本关于 MySQL 的书籍, 所以无法拿出足够的篇幅把 Apache 服务器的配置细节全部介绍给大家, 只能借此机会把 `httpd.conf` 文件里最为重要的几项基本设置简要地介绍如下:

- **ServerName:** Apache 服务器主机的网络名。网络上的其他计算机需要使用这个名字访问 Apache 服务器 (Web 服务器)。如果这项设置省略, Apache 服务器会试着为自己确定一个网络名。
- **DocumentRoot:** Apache 服务器将到该参数指定的子目录里寻找网页文件 (*.html、*.php 等)。表 2-2 对这项参数在不同安装环境下的默认设置进行了汇总。

表 2-2 用来存放 Web 页面文件的子目录

安装环境	DocumentRoot 位置
Windows	C:\Programs\Apache Group\Apache2\htdocs
Red Hat / Fedora (默认安装)	/var/www/html
SUSE Linux (默认安装)	/srv/www/htdocs

- **LoadModule:** 如果 Apache 被编译成可以在运行时实时加载各种模块 (实际情况也几乎总是如此), LoadModule 指令就必不可少。这些指令的格式如下所示:

```
LoadModule php5_module modules/libphp5.so      # for Linux
LoadModule php5_module "c:/php5/php5apache2.dll" # for Windows
```

- **AddType:** AddType 指令的作用是在文件标识符和应用程序类型之间建立关联关系。例如, 正是因为有了下面的设置, Apache 才会知道 *.php 文件应该用 PHP 解释器执行:

```
AddType application/x-httpd-php .php
```

- **PHPIniDir:** 这项设置将被传递给 PHP 解释器, 并告诉它到哪里去读取 `php.ini` 文件。这个选项本应该让 PHP 的配置工作变得简单一些, 但这往往会因为一些微不足道的理由而达不到目的, 如 PHP 无法找到 `php.ini` 文件、找到的是这个文件的过时版本, 等。PHPIniDir 选项是从 Apache2 开始才引入的, 目前只有 Windows 环境下的 Apache 版本用到它:

```
PHPIniDir "C:/php5"
```

- **AddDefaultCharset:** 这项设置的作用是让 Apache 知道网页文件使用的哪一种字符集。

- **<Directory "xxx">:** 这是一些由多条语句构成的代码块, 每个代码块对应着一个 Web 目录并对该目录的各种属性做出描述。`<Directory />` 块包含 `DocumentRoot` 参数指定的 Web 根目录的默认设置和其他一些配置参数。`<Directory>` 块的结束标记是 `</Directory>`。下面是一个例子:

```
<Directory /> or <Directory "directory ...">
...
AllowOverride AuthConfig FileInfo
Option Indexes
</Directory>
```

`<Directory>` 块内的各种选项控制着允许或禁止对这个子目录做什么、哪些人可以访问它等。这里只介绍两个最常用的选项:

- **AllowOverride:** 有时人们不希望 Apache 总是按照 `.htaccess` 文件所定义的规则去访问所有的 Web 目录, 而是希望它能够按照针对不同 Web 目录定义的规则——这些规则都保存在对应的子目录里——完成操作 (请参见 2.6.3 节)。要想实现这一想法, 就必须设置 `AllowOverride`

选项，这个选项的作用是让 Apache 明白用户希望它能随机而变。可供选用的设置值有 None（不允许 Apache 随机而变，这通常是默认设置）、All（让 Apache 全面接受所有的改动）；它们还可以再与 AutoConfig（身份验证）、FileInfo（语言和字符集设置、出错信息文件等）、Indexes、Limited（访问权限）和 Options 等设置值进行组合。

- **Options Indexes:** 如果从 Web 浏览器传来的地址是 `http://computername/directory/`, Apache 将自动根据相应的 DirectoryIndex 配置项（见下一条目）做出的规定返回 `index.html`、`index.php` 或其他网页。如果 DirectoryIndex 配置项指定的文件不存在，Apache 就会根据 Options Index 配置项的定义列出一份内容清单。这种做法在软件开发过程中很有用，但在软件正式投入使用后却可能是一个安全漏洞（除了 Index，还有许多其他的关键字可以与 Options 搭配使用）。
- **DirectoryIndex:** 利用这个选项，可以指定一个或多个文件供 Apache 在 Web 浏览器传来的地址是一个子目录路径时把这些文件中的某一个返回给 Web 浏览器。这个选项最常用的设置值之一是 `index.html index.php main.php default.php`。这样，当 Web 浏览器传来的地址是 `http://computername/directory/` 时，Apache 就会在子目录 `directory` 里搜索这些文件并把它找到的第一个文件返回给 Web 浏览器。

Web 页面文件的默认子目录

表 2-2 对 Web 页面文件（*.html、*.php 等）的默认存放位置（子目录）进行了汇总。可以把自己的 PHP 示例存放到这个默认子目录或是在其中创建的下级子目录里。

请注意，Linux 环境中的 Apache 服务器有它自己专用的账户（这个账户在 Red Hat 环境里是 `apache`，在 SUSE 环境里是 `wwwrun`）。对一个网页文件来说，只有把它的访问权限设置成同时允许某位用户和 Apache 读取，那个用户才能从自己的 Web 浏览器里看到这个网页的内容。可以用 UNIX 命令 `chown` 和 `chmod` 来改变有关文件的访问权限。例如，`chmod a+x *` 命令将使任何一位用户都能读取所有的文件。

2.6.3 对不同子目录的访问权限（.htaccess）

一旦 Apache 开始运行，本地网络或因特网上的每一个人都将可以访问 Apache 能够处理的所有 Web 页面。但这种情况并不总是人们所希望的，比如说，软件开发团队肯定不想让存放在一台开发用计算机上的东西被无关人员看到。有以下 3 种办法可以对网页的访问权限加以限制：

- **利用 httpd.conf 文件提供保护。** 在 `httpd.conf` 文件里，可以对某特定子目录里的网页都允许人们从哪些地址来访问做出限制。因为篇幅的限制，就不在这里详细讨论这一机制了。
- **利用防火墙提供保护。** 如果在开发用计算机上运行着一个防火墙，就可以通过配置这个防火墙把外部世界（这里所说的“外部世界”指的是“通过网络”）与这台计算机上的 Apache 隔离开来。这样一来，这台计算机上的网页就只能从本地计算机访问了。
- **为各子目录分别设置一个密码。** 这种保护措施的具体做法是：在需要保护的子目录里创建一个名为 `.htaccess` 的文件，然后设置只有使用正确的用户名和密码通过身份验证的人才能访问这个子目录里的内容。在与一个这样的子目录打交道时，Apache 将最先处理 `.htaccess` 文件，它会把一个登录对话框返回给那些使用 Web 浏览器访问这个子目录里的文件的网上用户。

在下面的讨论中，将重点介绍上述最后一种保护措施。它还可以用来保护某个向公众开放的子目录不被来自因特网的未知用户们访问。PHP 程序 `phpMyAdmin` 就是一个这方面的典型例子，它是许多

网站用来管理 MySQL 服务器的主要工具（参见第 6 章），如此重要的程序显然不应该允许随便什么人都可以访问。

1. 密码文件

建立密码保护机制的第一步是创建一个密码文件。出于安全方面的考虑，.htaccess 文件不保存密码数据，密码都保存在另外一个文件里，而该文件应该放在某个不允许无关人员访问的 Web 目录下。在下面的例子里，密码文件的名字是 site.pwd。

密码文件是用 Apache 工具程序 htpasswd 生成的（如果运行的是 SUSE 和 Apache 2，这个命令将是 htpasswd2。）选用-c（创建）选项，给出密码文件的名字和一个用户名。该程序将提示输入密码，然后生成密码文件。密码文件里的用户名是明文，密码是加密文本。

```
> htpasswd -c site.pwd myname
New password: *****
Re-type new password: *****
Adding password for user myname
```

往现有密码文件里添加一组用户名和密码也要执行 htpasswd 程序，但这次不需要-c 选项：

```
> htpasswd site.pwd name2
New password: *****
Re-type new password: *****
Adding password for user name2
```

如此生成的密码文件的内容如下所示：

```
myname:$apr1$gp1.....$qljDszVJOSCS.oBoPJtS/
name2:$apr1$A22.....$OV0iNc1FcXgNsruT9c6Iq1
```

提示 如果想在 Windows 系统上执行 htpasswd.exe 程序，需要打开一个命令窗口（命令提示符），并把该程序的完整路径名放在一对双引号里（通常是“Program\Apache Group\Apache\bin\htpasswd.exe”）。

如果想在 ISP 的计算机里的某个子目录里创建或扩大一个这样的密码文件，就要使用 ssh 或 ssh 程序去访问 ISP 的计算机（为了在那台计算机上执行 htpasswd 命令）。许多 ISP 提供有其他的配置帮助服务。

2. .htaccess 文件

如果某个 Web 目录里存在着一个名为 .htaccess 的文件，Apache 就将执行该文件里的所有配置设置。.htaccess 文件的语法和 httpd.conf 文件的完全一样，但可以在 .htaccess 文件里使用的配置选项数量较少，而且仅限于几个针对子目录的选项。

只要把一个内容如下所示的 .htaccess 文件放入某个子目录，则该子目录（及其所有的下级子目录）就都将受到保护：

```
AuthType Basic
AuthUserFile "c:/Programs/Apache Group/Apache2/htdocs/test/site.pwd"
AuthName "myrealm"
Require valid-user
```

AuthUserFile 语句给出密码文件的完整文件名。AuthName 语句用来划定密码的保护范围，有了它，就用不着在每次访问受 .htaccess 文件保护的子目录时都必须登录。只要登录进入了由 AuthName 语句划定的保护范围内的一个子目录，就可以随意访问这个范围内的其他子目录而无须再次登录。

如果想让自己只登录一次就能访问多个受 .htaccess 文件保护的子目录，就必须在这些子目录

中的.htaccess文件里使用同样的AuthName字符串（相当于把那几个子目录组织为一个域）。反过来说，如果给不同的子目录起了不同的域名，那每访问它们当中的一个都必须重新登录一次。

Require valid-user语句的意思是，每一种用户名和密码组合都必须是合法有效的登录信息。还可以利用这条语句限定只允许特定的用户进行登录，如下所示：

```
Require user myname name2 name3
```

关于用户身份验证机制和.htaccess配置方面的细节讨论可以在以下（但不限于）网址查到：

<http://apache-server.com/tutorials/ATUsing-htaccess.html>
<http://www.apacheweek.com/features/userauth>

注意 必须在Apache服务器的主配置文件httpd.conf里激活.htaccess机制才能让.htaccess文件生效。如果在httpd.conf文件中的<Directory>块里对某个Web目录或默认Web目录（“/”）做出了AllowOverride None设置，那些Web目录（及其下级子目录）里的.htaccess文件都将无法生效。不仅如此，若是AllowOverride语句里的选项与.htaccess文件的设置不一致，即使激活了.htaccess机制，也有可能触发一个错误并进而导致有关Web子目录里的部分文件无法被访问。

如果想让某个Web子目录中的.htaccess文件生效，就必须在httpd.conf文件中相应的<Directory>块里写出AllowOverride All或AllowOverride AutoConfig语句。

提示 Windows Explorer不能以.htaccess作为文件名创建文件，它会认为这个文件名是不合法的。但可以先把文件命名为htaccess.txt，再在输入请求窗口里用RENAME htaccess.txt.htaccess命令更改文件名。

有不少Windows用户不理解为什么这个文件名的第一个字符必须是一个英文句号。其实这种做法在UNIX/Linux环境里相当常见，它是创建一个隐藏文件最简便的办法——在默认的情况下，文件管理器不会把这种文件列在文件清单里。

3. 访问一个受保护的子目录

如果用Web浏览器去访问一个被存放在某个受.htaccess文件保护的Web目录里的文件，就会在Web浏览器里看到如图2-11所示的登录对话框。

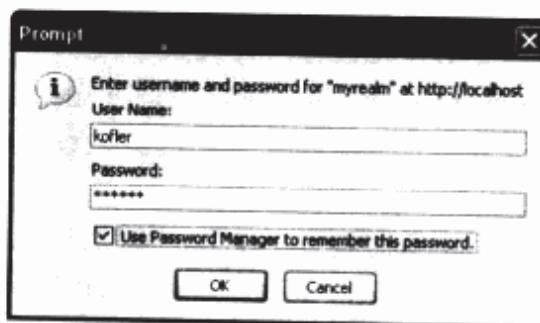


图2-11 访问一个受.htaccess文件保护的Web目录

2.7 配置PHP

PHP解释器的配置参数和选项都保存在php.ini文件里。表2-3列出了文件的常见存放位置。

表 2-3 PHP 配置文件的存放位置

安装环境	php.ini 文件的存放位置
Windows	C:\php5\php.ini
Linux (用户自行编译 PHP)	/usr/local/php5/lib/php.ini
Red Hat / Fedora (默认安装)	/etc/php.ini
SUSE (默认安装)	/etc/php5/php.ini

在 Windows 环境下, php.ini 文件的存放位置一般是由 Apache 配置文件 httpd.conf 中的 PHPIniDir 选项设置的 (从 Apache 2 开始)。如果这个选项默认, PHP 解释器将在 Windows 目录里搜索 php.ini 文件。

注解 如果 PHP 解释器无法找到 php.ini 文件或该文件根本就不存在, PHP 解释器将使用默认的配置设置。如果不清楚 php.ini 文件到底存放在何处, 可以利用 PHP 测试页面——该页面由 PHP 函数 phpinfo() 生成 (如图 2-5 所示) ——去寻找它。

在 Windows 环境下, php.ini 文件的模板可以在 PHP 安装目录 (例如 C:\php5\) 里找到。在 Linux 环境下, 由用户自行安装的 PHP 会把 php.ini 文件的模板保存在 /usr/local/src/php-5.n/ 子目录里。如果要修改 php.ini 文件, 以 PHP 软件包自带的 php.ini-recommended 文件作为蓝本将是个很不错的出发点。

1. 基本设置

这是一本关于 MySQL 的书籍, 所以无法拿出足够的篇幅把 php.ini 文件里的所有选项全都介绍一遍。只能借此机会把 php.ini 文件里最为重要的几项基本设置简要地介绍一下 (下面这些选项的设置值是这本书所使用的):

- **display_error=On:** 这项设置的作用是当某个 PHP 脚本执行出错时显示一条出错消息。这种安排在软件开发阶段非常有用。已经接入因特网的服务器最好禁用这项设置——出错消息往往会展露一些编程方面的小秘密, 它们很可能成为系统的安全漏洞 (在开发一个 PHP 页面的过程中, 如果在 Web 浏览器里看到的不是预定结果而是一个空白页面, 则极可能是由 display_error 设置造成的。)
- **error_reporting=E_All:** 这项设置能够让用户看到所有的 PHP 出错消息和警告消息。
- **magic_quotes_gpc=On:** 这项设置会影响到网页之间传递的数据 (Get/Post/Cookie), 单引号 ('), 双引号 ("), 反斜线 (\) 和零字节将被替换为转义序列 “\”、“\”、“\\” 和 “\0”。对转义序列 (MySQL 文档称之为 *Magic Quotes*) 的详细讨论请参见第 15 章。我们之所以会把这个选项设置为 *On*, 是因为它是几乎所有的 ISP 都在使用的默认设置, 和 ISP 们保持一致可以让我们开发出来的代码更具普遍适用性。
- **default_charset:** 这个选项通常不需要设置 (直接使用默认配置)。把它设置为 iso-8859-1 是为了配合本书的部分示例。*default_charset* 选项控制着 PHP 解释器是否对 HTML 页面文件里属于其他字符集的字符进行转换。
- **max_execution_time=30:** 这个选项的作用是: 如果某个 PHP 脚本在给定时间内 (以秒为单位) 没有执行完毕, 就终止它的执行。
- **max_input_time=60:** 这个选项的作用是: 如果 PHP 等待输入数据的时间超过了给定长度 (比如在某个客户向 PHP 程序上传文件的时候), 就断开与那个客户的连接。

- **file_uploads=On**: 这个选项的作用是让 PHP 能够对文件上传操作进行处理。
- **upload_max_filesize=2M**: 这个选项对文件上传操作中的最大文件长度做出了限定。
- **post_max_size=8M**: 这个选项对通过一个 (POST) 表单发送给 PHP 程序的最大数据集长度做出了限定。
- **memory_limit=8M**: 这个选项对一个 PHP 脚本所能占用的最大内存量做出了限定。

2. PHP 扩展模块

PHP 扩展模块在 PHP 内置功能的基础上向人们提供了更多的功能。哪些功能已直接内置在 PHP 里、哪些功能被认为是扩展的，要取决于 PHP 是如何编译的。

- 如果是在 UNIX/Linux 环境下自行编译的 PHP，通常会把需要用到的功能全都直接集成到 PHP 解释器里（通过适当的 ./configure 选项）。如果是这样，就不需要在 php.ini 文件里对扩展模块进行配置了。
- 在某些 Linux 发行版本里（例如 SUSE），PHP 已高度模块化，每个扩展模块都可以作为一个独立的软件包来单独安装。如果这样，同样不需要关心 php.ini 文件，因为 Linux 发行版本的软件包管理子系统会把这方面的工作管理好。
- 如果使用的是 Windows 系统，将极有可能需要与 PHP 扩展模块打交道。除最核心的功能以外，PHP 解释器的 Windows 版本几乎没有任何额外的功能，所以 Windows 下的 PHP 扩展模块特别多。这些扩展模块都是以扩展库（下级子目录 ext 中的*.dll 文件）的形式提供的。

如果想使用扩展模块，就必须在 php.ini 文件里对变量 extension_dir 进行设置，让它指向用来存放 PHP 扩展模块的子目录，如下所示：

```
extension_dir = "c:\php5\ext\"
```

除此之外，还必须在 php.ini 文件里用 extension 指令列出要使用的扩展模块。下面是对于 OpenSSL 模块的 extension 指令：

```
extension=php_openssl.dll
```

php.ini 文件往往包含着许多条这种 extension 指令，但它们的前面都有一个分号 (;)。在 php.ini 文件里，分号代表着一条注释。如果需要用到某个模块，把与之对应的 extension 指令前面的分号去掉即可。在 UNIX/linux 环境下，PHP 扩展模块的文件标识符是 .so，不是 .dll。

注解 有许多 PHP 扩展模块还需要访问其他函数库，必须把它们安装到一个能够让 PHP 解释器找到的地方。在 Windows 环境下，这意味着必须把有关的扩展模块和函数库复制到 Windows 安装目录或 Windows 系统目录。

例如，PHP 扩展模块 php_mysql.dll 还需要调用 libmysql.dll 库文件里的函数。这个库文件已被收录在最新的 PHP 版本里，但因为它的默认存放位置是 PHP 安装目录，所以 PHP 解释器找不到它。因此，必须在重新启动 Apache 之前把 libmysql.dll 库文件复制到 Windows 安装目录。如果没有完成这一操作就去重新启动 Apache，Apache 将不会启动——它会返回一条出错消息说无法加载某些个 DLL。

2.8 配置 MySQL

有许多选项可以通过一个配置文件来设置和改变。表 2-4 列出了 MySQL 配置文件的常见存放位置。

表 2-4 MySQL 配置文件的存放位置

安装环境	php.ini 文件的存放位置
近期 Windows 版本	C:\Programs\MySQL\MySQL Server n.n\my.ini
早期 Windows 版本	C:\my.cnf 或 Windows\my.ini
Linux	/etc/my.cnf

所有的 MySQL 选项都有默认值，所以即使没有配置文件或者是不加任何修改地使用一个已经存在的配置文件，也可以让 MySQL 运行起来。对 MySQL 配置文件的语法和各种选项的详细介绍参见第 14 和第 22 章。

启动 MySQL 服务器

必须重新启动 MySQL 服务器才能 MySQL 配置文件里做出的修改生效。

- **Windows:** 在 Windows 环境下，重新启动 MySQL 服务器最简单的办法是使用 MySQL 控制台（MySQL System Tray Monitor）。这个控制台在 Windows 任务栏上有一个小图标（靠近系统时钟），可以通过它来执行 Shutdown Instance（停止服务器）和 Start Instance（启动服务器）命令。如果 MySQL 控制台无法运行，通过菜单 Start（开始）| Program（程序）| MySQL 也可以启动这个程序。如果这个程序已经在运行但它的图标没有出现在 Windows 任务栏上，可能是任务栏的设置有问题。用鼠标右键单击任务栏，执行 Taskbar 命令，取消选中 Hide inactive icon（隐藏不活跃的图标）。
- **Linux:** 在 UNIX/Linux 环境下启动 MySQL 服务器要简单得多，一条简短的命令就够用了。重新启动 MySQL 服务器的工作可以用相应的 Init-V 脚本来完成：先用 *stop* 参数执行它一次，再用 *start* 参数执行它一次。根据具体使用的 Linux 发行版本，脚本的名字是 mysql 或 mysqld。

```
root# /etc/init.d/mysql[d] restart
```

第 3 章

初级案例研究：MySQL+PHP



要 想掌握一种数据库或软件开发系统，最好的办法莫过于通过一个面面俱到的例子来学习。在这一章里，将结合 MySQL 和 PHP 来创建一个网上调查站点。

在某种程度上，这是一个非常简单的案例，相信不少读者就算不使用 MySQL 也可以轻而易举地实现同样的结果。从大的方面讲，希望大家能够通过这个案例对 MySQL 和脚本程序设计语言（案例中使用的是 PHP 语言）之间相辅相成的关系有一个直观的了解和重视。在细节方面，希望大家能够领会在数据库应用程序的整个开发过程中——从开始设计数据库到最终完成——都需要注意哪些问题。

3.1 概述

我们的网上调查站点由两个页面构成。第一个页面（vote.html 文件）是一份问卷，问卷上只有一个问题：What is your favorite programming language for developing MySQL applications？在这个问题的下面列出了几种程序设计语言供人们投票选择（如图 3-1 所示）。在选择了一种语言并单击 OK 按钮之后，将看到一个如图 3-2 所示的调查结果页面（results.php 文件）。

图 3-1 调查问卷

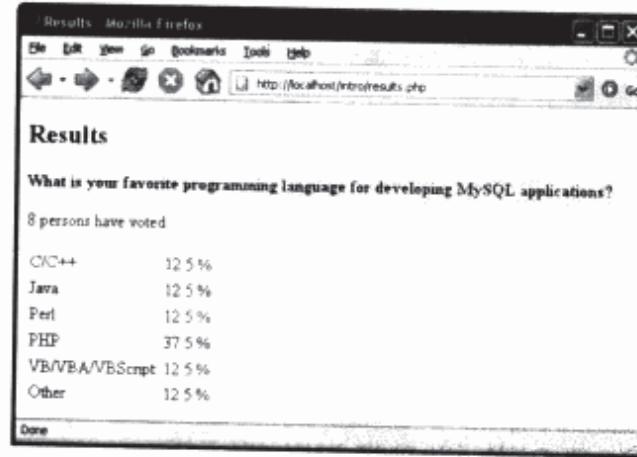


图 3-2 调查结果

准备工作

读者可以到笔者的个人网站（www.kofler.cc）体会一下这个网上调查站点的效果。但为了保证学习质量，最好能在自己的计算机上重现这个案例。如果决定这么做，将需要一个由 Apache + MySQL + PHP 构成的测试环境，这个环境必须允许做以下几件事：

- 创建一个新的数据库（即必须有足以执行 *CREATE DATABASE* 命令的权限）。

□ 把文件移动到 Web 服务器上的某个子目录里去。

□ 执行 PHP 脚本文件。

在一台本地计算机上建立一个符合上述要求的测试环境的具体步骤见第 2 章。

彻底掌握这个案例需要具备一些数据库方面的背景知识。如果很少或从未接触过数据库系统并因此而在理解问题的来龙去脉时遇到了困难，千万不要失去信心。本书的第 8 章对关系数据库系统进行了介绍，第 9 章对数据库查询语言 SQL 的使用方法进行了解释。

这个案例使用的程序设计语言是 PHP，用这种语言编写的代码很容易看懂，即使从未学习过 PHP 语言的读者也不会遇到什么困难。但读者应该大致了解一些关于如何把脚本语言函数嵌入 HTML 的知识。（由微软公司开发的 Active Server 页面也是基于这一思路的。）

3.2 数据库的开发

为了把网上调查结果保存到 MySQL 数据库，首先需要创建一个数据库，再在数据库里创建一个数据表。（同一个 MySQL 数据库可以包含多个数据表。具体到这个案例，只须一个数据表就够用了。软件开发项目的要求越复杂，需要用到的数据表就越多，它们之间的关联关系也将越复杂。）

3.2.1 启动 mysql 命令行解释器

下面将要进行的两项工作：创建一个数据库和创建一个新的数据表，需要能够与 MySQL 通信。在 UNIX/Linux 环境下，执行 mysql 命令就可以启动 mysql 命令行解释器；在 Windows 环境下，可以通过菜单命令 Program（程序）| MySQL | MySQL Server n.n | MySQL Command Line Client 来启动它。

注解 如果 mysql 程序无法启动运行并返回一条 Access denied for user xy（用户 xy 的权限不够）出错信息，原因有两个：一是用户 xy 根本无权访问 MySQL；二是用户 xy 必须给出一个正确的密码才能访问 MySQL。不管什么原因，都应该使用 -u name 和 -p 选项再次执行 mysql 命令：

```
> mysql -u username -p
Enter password: xxx
```

除了这两个选项，mysql 程序还有几十种其他的选项。本书的第 4 章对比较重要和常用的 mysql 命令行选项进行了介绍，第 22 章对所有的 mysql 命令行选项进行了汇总。关于 MySQL 信息安防机制的背景知识（访问控制、密码、权限）可以参见第 11 章。

MySQL 和 mysql 之间的区别有很多人都搞不清楚：MySQL 说的是数据库服务器，它通常会在系统开机时自动启动并在后台保持运行（本章内容将假设这个服务器已经启动运行了），这个服务器程序的名字因操作系统而异，在 UNIX/Linux 环境下叫做 mysqld，在 Windows 环境下叫做 mysqld.exe 或 mysqld-nt.exe。

与这些形成对照的是命令行解释器 mysql 和 mysql.exe。这些程序是人们在需要以交互方式对 MySQL 数据库系统进行维护或管理时使用的软件工具。mysql 程序的任务是把人们以交互方式发出的命令传递给 MySQL 服务器，再把那些命令的执行结果显示在屏幕上。

将在第 5 章和第 6 章对 mysql 程序的替代品，其中包括 MySQL Administrator 和简便易用的 HTML 操作界面 phpMyAdmin——进行介绍。

通过 mysql 程序输入的命令将被传输到数据库服务器去执行（如图 3-3 所示）。可以用 STATUS 命令来检查自己是否已经连接上了数据库服务器，这个命令的执行结果将返回许多关于数据库的状态信

息（比如它的版本编号）。

```
mysql> STATUS;
mysql Ver 14.7 Distrib 5.0.2, for Win95/Win98 (i32)
Connection id:          2
Current database:
Current user:           ODBC@localhost
SSL:                   Not in use
Using delimiter:        ;
Server version:         5.0.2-alpha
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    latin1
Conn.  characterset:    latin1
TCP port:               3306
Uptime:                 12 days 1 hour 26 min 20 sec1

Threads: 2  Questions: 3099  Slow queries: 0  Opens: 6
Flush tables: 1 Open tables: 0  Queries per second avg: 0.003
```

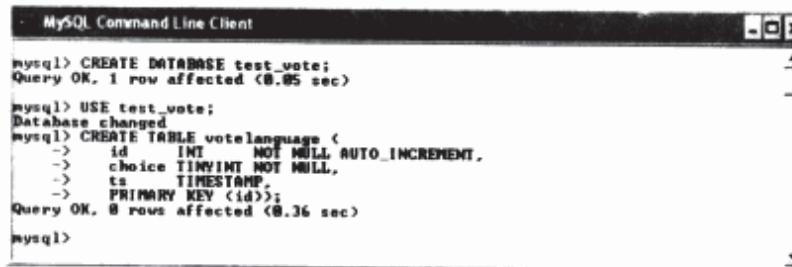


图 3-3 MySQL 命令行解释器的窗口画面

如果在启动 mysql 程序或执行 *STATUS* 命令时遇到了麻烦，最有可能的原因是数据库服务器尚未启动或者是没有足够的访问权限。如果遇到的是 MySQL 服务器安装、配置和启动方面的问题，请参阅第 2 章；如果遇到的是访问权限和信息安防方面的问题，请参阅第 11 章。

3.2.2 创建数据库

这个案例需要创建一个名为 *test_vote* 的新数据库，这项工作可以通过在 mysql 程序里执行 *CREATE DATABASE* 命令来完成。注意，必须用一个分号（;）来结束这个命令。下面两行代码中的黑体字部分是必须由用户输入的内容：

```
mysql> CREATE DATABASE test_vote;
Query OK, 1 row affected (0.09 sec)
```

下面看一下由 mysql 返回的响应信息。MySQL 服务器会把自己管辖范围内的所有数据库列成一份清单，并在其内部把这份清单保存为一个 MySQL 数据表，而输出信息“1 row affected”表明：在那份清单里，有一行数据发生了变化。现在还不必考虑到底发生了什么样的变化，只要知道这意味着 *CREATE DATABASE* 命令已经得到了正确的执行就可以了。

在这里选择 *test_vote* 作为新数据库的名字是有目的的。根据 MySQL 访问控制机制的默认设置，每个用户都有权在本地计算机上使用以单词“*test*”开始的名字来创建新的数据库。如果不是 MySQL 系统管理员（只是一位很多事情都要求助于系统管理员的普通用户），选用 *test_xy* 形式的名字来创建自己的数据库肯定会省下许多的求助电子邮件或求助电话。

使用 *test_xy* 形式的名字来创建数据库的弊病是：本地计算机上的任何一位用户都可以编辑、修改

甚至删除你的数据库。这对这个简单的案例来说不是什么大问题，但对一个实际应用程序来说就太不安全了。如果想为自己的数据库提供更多的信息安防保护，请参阅本书的第 11 章。

3.2.3 创建数据表

test_vote 数据库创建好了，但它现在还不能保存任何信息。信息需要用数据表来保存。如果想在 *test_vote* 数据库里创建一个新的数据表，还需要再执行一条 *CREATE TABLE* 命令。

在执行 *CREATE TABLE* 命令的时候，必须告诉它想让它在哪个数据库里创建新的数据表，这项工作要用 *USE* 命令来完成。*USE* 命令的用途是为后续命令指定一个默认数据库，让它们知道应该对哪个数据库进行处理或操作。(要知道，除了刚刚创建的 *test_vote* 数据库，MySQL 服务器很可能还管理着许多其他的数据库。)

```
mysql> USE test_vote;
Database changed
mysql> CREATE TABLE votelanguage (
    ->     id      INT      NOT NULL AUTO_INCREMENT,
    ->     choice  TINYINT NOT NULL,
    ->     ts      TIMESTAMP,
    ->     PRIMARY KEY (id));
Query OK, 0 rows affected (0.75 sec)
```

从未接触过数据库系统的读者可能会对 *CREATE TABLE* 命令感到有些陌生。别管它，先把上面给出的黑体字命令输入进去。(可以用回车键来断开那些命令行，在最后一行用分号结束整个命令。分号的作用是告诉 mysql 程序“整个命令已输入完毕”。)

如果在输入命令的时候打错了字，MySQL 会立刻发现它并用一条出错消息通知用户。如果真的遇到这种情况，将需要重新输入整个命令。使用上、下箭头键可以迅速到达出错位置去做出修改。

如果打错了字但 MySQL 还是接受了输入的 *CREATE TABLE* 命令(出现这种情况的原因是，虽然打错了字或者说犯了一个语义错误，但它在语法上仍是正确的)，使用 *DROP TABLE votelanguage* 命令删除定义有误的数据表，然后重新输入正确的 *CREATE TABLE* 命令。

用 *CREATE TABLE* 命令来创建 *votelanguage* 数据表只是办法之一，完全可以使用其他的数据库管理工具，如 MySQL Administrator 或 phpMyAdmin，来完成这项工作。当然，那么做的先决条件是已经把那些工具安装在了计算机里，并且知道它们的使用方法。mysql 程序的易用性比较差是人所共知的事实，但它的好处是可以把曾经执行过的步骤详细地记录下来，让用户有机会改正错误。

现在来解释一下实际发生了哪些事情。刚才输入和执行的 *CREATE TABLE* 命令创建出了一个新的数据表，这数据表有 *id*、*choice* 和 *ts* 3 个数据列。如果把一些数据(网上问卷调查结果)填入这个数据表，这个数据表的内容就将显示为如下所示的样子：

<i>id</i>	<i>choice</i>	<i>ts</i>
1	4	20050114154618
2	5	20050114154944
3	4	20050114154953
4	3	20050114154954
5	3	20050114154957
6	6	20050114155012
7	3	20050114155021
8	1	20050114155027
...		

这些数据的含义是：第一个回答问卷问题的人选择的程序设计语言是 PHP，第二个人选择的是 VB，第三个人选择的又是 PHP。接下来的几个人分别选择了 Perl、Other、Perl 和 C。那个名为 *id* 的数据列

包含着一个递增的标识编号，这使每行数据（它们被统称为“数据集”）都有一个独一无二的编号。名为 *choice* 的数据列保存着人们在回答调查问卷时做出的选择（经过了编码），数字 1~6 分别对应着 C、Java、Perl、PHP、VB 等几种程序设计语言和 Other (“其他”)。名为 *ts* 的数据列记录着人们回答调查问卷的时刻（比如说，第一个回答问卷问题的人是在 2005 年 1 月 14 日 15 点 46 分 18 秒填写好 *vote.html* 页面的）。

注解 与其他专业领域一样，数据库领域也有自己的术语。比如说，数据库专业人员会把上面数据表里的每一行称为一条数据记录（*data record*），把它的每一列（例如 *id* 和 *choice* 列）称为一个字段（*field*）。

如果只是为了生成一个由 *id* 和 *choice* 两个数据列构成的数据表，下面这条命令就已经足够了：

```
CREATE TABLE votelanguage (id INT, choice TINYINT, ts TIMESTAMP);
```

这条命令将把 *id* 列的数据类型声明为 *INT*，把 *choice* 列的数据类型声明为 *TINYINT*。这些声明意味着，从理论上讲，在超出整数变量 *id* 的表示范围之前，最多可以有 2^{31} （即 2 147 483 648）个人参与问卷调查（如果把 *id* 列声明为 *UNSIGNED INT* 类型，这个数字将翻一番）。再看 *choice* 列，它可以提供 2^{16} 种选择。（本书将在第 8 章对 *INT*、*TINYINT* 以及其他 MySQL 数据类型进行讨论。）

看到这儿，或许会有一些读者对本章的内容安排提出质疑：既然有这么简单的办法可以得到同样的结果，为什么不先介绍它而是让大家使用那么复杂的 *CREATE TABLE* 命令呢？是这样的，这种复杂与简单之间的区别正是好的和坏的数据库设计方案之间的区别。

在前一个 *CREATE TABLE* 命令里，为 *id* 数据列定义了 *AUTO_INCREMENT* 属性。这么做的效果是 MySQL 会在每一条新记录里自动插入一个正确的 *id* 值。这样一来，在保存问卷调查结果的时候，只要处理好 *choice* 字段即可，*id* 字段有 MySQL 替我们照料。这个属性将确保数据记录的编号是递增的和独一无二的，而这一点对数据表的使用和管理工作有着非常重要的意义。

在前一个 *CREATE TABLE* 命令里，还为 *id* 和 *choice* 数据列定义了 *NOT NULL* 属性。这个属性的作用是确保 *id* 和 *choice* 这两个字段的取值有实际意义。具体到这个案例，首先，空数据记录 (*NULL*) 将不允许存入数据表；其次，*choice* 字段没有取值的数据记录也不允许存入数据表（*id* 字段由 MySQL 负责处理，它肯定会有取值）。总之，这个属性可以防止非法或无效的数据记录被存入数据表。（读者不妨试试看能不能让 MySQL 接受一个非法的数据记录，它肯定会拒绝接受并显示一条出错信息。）

PRIMARY KEY 属性（见 *id* 数据列）的效果是让 MySQL 使用 *id* 数据列来标识各条数据记录。这其实是最先定义 *id* 数据列的原因，但因为 MySQL 还没有聪明到可以猜出人类思想的地步，所以还需要通过 *PRIMARY KEY* 属性把我们想让它做的事情精确无误地告诉它。（从现在起，将在讨论这个案例的时候把 *id* 字段改称为主键（primary key）。）一个数据表是否有主键对 MySQL 访问这个数据表里的数据记录的速度有着决定性的影响，对那些相互关联的数据表来说就更是如此。作为一个原则，应该为自己的每一个数据表定义一个唯一的主键（在这样做的时候，要尽可能地把含有 *AUTO_INCREMENT* 属性的 *INT* 字段定义为主键）。

3.2.4 为什么要避简就难

有些读者可能会认为笔者是在把一个简单的问题复杂化，因为它完全可以用一个简单的办法来解决。这个案例所需要的只不过是 6 个计数器而已，创建一个如下所示的数据表就足够了：

id	counter
1	2
2	0
3	7
4	9
5	2
6	1

上面这个数据表的内容可以这样来解释：在回答问卷问题的人群当中，有 2 个人喜欢用 C 语言，没人喜欢用 Java，有 7 个人喜欢用 Perl，有 9 个人喜欢用 PHP。每当有人回答说他喜欢某种程序设计语言的时候，给相应的计数器加上一个 1 就行了。这么一来，数据库总共才需要 6 个数据行；不必顾虑性能问题，内存占用量又几乎是零——这有多好！再说，总共不过 6 个计数器，一个小小的文本文件就已经足以容纳它们。总而言之，这个调查问卷案例根本用不着 MySQL。

就事论事地说，上面提出的这个方案已经解决了我们的问题，但也就仅此而已了。可是，这个方案可以说没有任何扩展的余地，一旦项目要求发生变化，就不得不重新设计一个新的解决方案。你们是否考虑过，如果今后想为每位参与者提供一个发表评论的机会该怎么办？如果想让参与者必须先注册再回答问卷问题（为了防止人们重复投票干扰调查结果）该怎么办？如果想把每位参与者的答题时刻和 IP 地址记录下来（还是为了防止人们重复投票干扰调查结果）又该怎么办？

如果也曾考虑过上面这些问题，就一定会赞同笔者提出的数据库设计方案。这么一来，在需要把解决方案扩展到可以解决上面那几个问题的时候，用不着对程序结构做什么修改，只要给 *votelanguage* 数据表增加一两个数据列就行了。从这个意义上讲，如果在第一眼看到笔者提出的解决方案时觉得它有点儿避简就难，那不过是因为笔者想让它有最大的可扩展性——虽然它只是本书的第一个案例，但这种为今后的扩展留出余地的思维方式却是每一个程序员都应该具备的。

3.3 调查问卷

正如在本章开篇时提到的那样，这个案例项目由两个网页构成。第一个页面 (*vote.html*) 包含着调查问卷，它全部由 HTML 代码实现（没有 PHP 代码）。第二个页面 (*results.php*) 要完成两项任务：一是对问卷调查结果进行处理，二是显示调查结果。

下面是用来实现调查问卷的 HTML 代码，把它们弄明白对理解 3.4 节里的问卷调查结果处理代码将会有很大的帮助。这段代码中最重要的是问卷表单元素的 *name* 和 *value* 属性。6 个单选按钮的名字都是 *vote*，它们的 *value* 属性分别是 1~6 之间的整数值。OK 按钮的名字是 *submitbutton*，它的 *value* 属性值是字符串“OK”。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- php/vote/vote.html -->
<html><head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1" />
  <title>MySQL-poll</title>
</head>
<body>
<h2>MySQL-poll</h2>

<p><b>What is your favorite
programming language for developing MySQL applications?</b></p>

<form method="POST" action="results.php">
  <p>    <input type="radio" name="vote" value="1" />C/C++
  <br /><input type="radio" name="vote" value="2" />Java
  <br /><input type="radio" name="vote" value="3" />Perl
```

```

<br /><input type="radio" name="vote" value="4" />PHP
<br /><input type="radio" name="vote" value="5" />
      ASP[.NET] / C# / VB[.NET] / VBA
<br /><input type="radio" name="vote" value="6" />
      another language
</p>
<p><input type="submit" name="submitbutton" value="OK" /></p>
</form>
<p>Go directly to the <a href="results.php">result</a>.</p>
</body>
</html>

```

3.4 问卷调查结果的处理和显示

若要显示问卷调查结果，可以直接访问 results.php 页面（比如说，通过问卷页面上的链接），也可以在提交问卷表单后查看。在前一种情况下，系统将只显示问卷调查结果的当前状态；在后一种情况下，系统还将对提交的数据进行处理并把它存入数据库。

3.4.1 mysql 界面与 mysqli 界面

从 PHP 5 开始，PHP 语言同时支持 mysql 和 mysqli 这两种与 MySQL 进行通信的界面。

- mysql 界面是以前的 PHP 版本都支持的界面，它可以与几乎所有的 MySQL 版本配合工作。在开发这个案例时使用的就是 mysql 界面，所以编写的程序在老版本的 PHP 和 MySQL 环境下也可以运行。
- mysqli 界面相对要新一些，它是一种面向对象的界面，只能在 PHP 5 和 MySQL 4.1（或更高的版本）环境中使用。将在第 15 章学习使用 mysqli 界面。

3.4.2 建立与数据库的连接

results.php 文件的开头部分是一些 HTML 代码，随后是负责完成第一项任务——建立一条与 MySQL 服务器的连接——的 PHP 代码。这个任务是由 PHP 函数 *mysql_connect* 具体完成的，这个函数需要传递 3 个参数，即：

- 用户名：
- 密码：
- 计算机名（主机名）。

如果把 MySQL 安装在了本地计算机上并且尚未给它设置密码，可以简单地传递一个空字符串作为密码参数的值。计算机名参数只有在 Web 服务器（即 Apache）与 MySQL 服务器不在同一台计算机上运行的情况下才是必须给出的。

mysql_connect 函数将返回一个标识代码，把这个代码保存在变量 *link* 里。在这个案例里，只在需要进行“无法与数据库建立连接”错误处理时才会用到 *link* 变量。一旦发生这种错误，系统将显示一条出错信息，然后通过 *exit* 语句立刻终止 PHP 代码的执行。如果一切顺利，就将由 *mysql_select_db* 函数打开指定的数据库。（MySQL 服务器可以同时管理多个数据库。具体到这个案例，*mysql_select_db* 函数的作用是让所有的后续命令把 *test_vote* 数据库当做默认的处理对象。）

3.4.3 对数据进行处理并把它存入数据库

results.php 脚本的第二项任务是对调查问卷里的数据进行处理并把它们存入数据库。那些数据

将被传递给全局字段 `$_POST` 的 `submitbutton` 和 `value` 元素(请把它们与 `vote.html` 页面里的 `name` 和 `value` 属性进行对比)，我们编写了一个名为 `array_item` 的辅助函数来读出这个字段。

如果 `results.php` 脚本是因为人们在 `vote.html` 页面里回答了问卷问题之后单击 `OK` 按钮而被调用的，变量 `submitbutton` 的取值就将是字符串“`OK`”。为避免意外，`results.php` 脚本还对变量 `vote` 的值进行了一些合法性检查。(比如说，说不定会有人没在问卷上做出选择就提交了表单。) 如果变量 `value` 的值是合法的(是 1 到 6 之间的一个整数)，这个值就会被保存到数据库里去。用来完成这个操作的 SQL 命令如下所示：

```
INSERT INTO votelanguage (choice) VALUES (3)
```

上面这条 `INSERT INTO` 语句将把一条新的数据记录(一个新的数据行)插入到 `votelanguage` 数据表里，表达式(`choice`)和 `VALUES (3)`命令的意思是把数据记录的 `choice` 字段的值设置为 3。因为 `id` 字段由 MySQL 负责处理(参见前面对 `AUTO_INCREMENT` 属性的讨论)，所以在这个案例里只有 `choice` 字段需要关照。当然，根据人们在回答问卷问题时选择的程序设计语言，这个“3”应该被替换为 `vote` 变量的值。这条 SQL 语句将由 PHP 函数 `mysql_query` 传递到 MySQL 服务器去执行。

提示 如果从未与 SQL 打过交道，因而对 SQL 命令的语法不熟悉，不要害怕，本书的第 9 章对 SQL 语言做了详细的讨论。本章只用到了两条 SQL 命令(`INSERT` 和 `SELECT`)，它们的用途从字面上就应该能看懂。

`votelanguage` 数据表的 `id` 和 `ts` 数据列由 MySQL 自动填写，它会把一个递增的、独一无二的编号数字填入 `id` 字段，把当前时间填入 `ts` 字段。

3.4.4 显示问卷调查的结果

不管是否回答了调查问卷里的问题，都可以查看以前的调查结果。(如果回答了问卷问题，看到的调查结果将包含着自己的贡献。)

要做的第一件事是检查一下 `votelanguage` 数据表里有没有数据。(当这份问卷第一次出现在因特网上时，因为还没有人投过票，所以在 `votelanguage` 数据表里没有数据。) 这种检查可以用如下所示的 SQL 查询来完成：

```
SELECT COUNT(choice) FROM votelanguage
```

这条 SQL 语句也将由 `mysql_query` 函数传递到 MySQL 服务器去执行。但这次把查询结果的一个链接保存到了变量 `result` 里。一般来说，`SELECT` 命令的执行结果将是一个数据表。具体到上面这条 `SELECT` 语句，作为其执行结果的数据表只包含一项数据，也就是只有一行一列。(顺便说一句，`result` 变量的值只是一个 ID 数字，这个数字将被其他 `mysql_xxx` 函数当做一个参数来使用。对查询结果本身进行具体管理是 PHP 的任务。)

为了对查询结果进行处理，这里使用了 `mysql_result($result, 0, 0)` 函数，它将从查询结果数据表中把位于第一行第一列的数据提取出来。(PHP 函数在处理数组时把 0 作为第一个下标。)¹

在 `votelanguage` 数据表不为空的情况下，接下来的循环语句将为每一种程序设计语言分别计算出它们在这个网上调查中的得票率。这里使用的 SQL 查询与上面那条 `SELECT` 语句很相似，只是这次统计的是那些 `choice` 字段值等于某给定数字(比如 3)的数据记录有多少个而已，如下所示：

1. 这里原书误写成“MySQL 函数”——`mysql_result()` 应该是一个 PHP 函数。——译者注

```
SELECT COUNT(choice) FROM votelanguage WHERE choice = 3
```

这里又一次用到了 *mysql_result* 函数。为了得到精确到两位小数的百分比数字，这里还进行了一些必要的运算。

3.4.5 程序代码（results.php）

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- php/vote/results.php -->
<html><head>
<meta http-equiv="Content-Type"
      content="text/html; charset=iso-8859-1" />
<title>Survey Result</title>
</head><body>
<h2>Survey Result</h2>
<?php

$mysqlhost="localhost";
$mysqluser="root";
$mysqlpasswd="";
$mysqldbname="test_vote";
// Create connection to the database
$link =
@mysql_connect($mysqlhost, $mysqluser, $mysqlpasswd);
if ($link == FALSE) {
    echo "<p><b>Unfortunately, a connection to the
database cannot be made. Therefore, the results cannot be
displayed at this time. Please try again later.</b></p>
</body></html>\n";
    exit();
}
mysql_select_db($mysqldbname);

// if questionnaire data are available:
// evaluate + store
function array_item($ar, $key) {
    if(array_key_exists($key, $ar)) return($ar[$key]);
    return(''); }

$submitbutton = array_item($_POST, 'submitbutton');
$vote = array_item($_POST, 'vote');

if($submitbutton=="OK") {
    if($vote>=1 && $vote<=6) {
        mysql_query(
            "INSERT INTO votelanguage (choice) VALUES ($vote)");
    }
    else {
        echo "<p> Not a valid selection. Please vote
again. Back to
<a href=\"vote.html\">questionnaire</a>.</p>
</body></html>\n";
        exit();
    }
}

// display results
echo "<p><b> What is your favorite programming language
for developing MySQL applications?</b></p>\n";

// Number of votes cast
$result =
```

```

mysql_query("SELECT COUNT(choice) FROM votelanguage");
$choice_count = mysql_result($result, 0, 0);

// Percentages for the individual voting categories
if($choice_count == 0) {
    echo "<p>No one has voted yet.</p>\n";
}
else {
    echo "<p>$choice_count individuals have thus far taken part
        in this survey:</p>\n";
    $choicetext = array("", "C/C++", "Java", "Perl", "PHP",
        "VB/VBA/VBScript", "Andere");
    print("<p><table>\n");
    for($i=1; $i<=6; $i++) {
        $result = mysql_query(
            "SELECT COUNT(choice) FROM votelanguage " .
            "WHERE choice = '$i'");
        $choice[$i] = mysql_result($result, 0, 0);
        $percent = round($choice[$i]/$choice_count*10000)/100;
        print("<tr><td>$choicetext[$i]:</td>");
        print("<td>$percent %</td></tr>\n");
    }
    print("</table></p>\n");
}
?>
</body>
</html>

```

3.4.6 最终生成的 HTML 代码

如果对 PHP 不怎么熟悉，阅读由 results.php 脚本最终生成的 HTML 代码（从 Web 浏览器里看到的问卷调查结果页面就是由这些代码生成的，如图 3-2 所示），肯定会对看懂上面的程序代码清单有所帮助。

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- results.php -->
<html><head>
<meta http-equiv="Content-Type"
      content="text/html; charset=iso-8859-1" />
<title>Survey Results</title>
</head><body>
<h2>Survey Results</h2>
<p><b>What is your favorite programming language
    for the development of MySQL applications?</b></p>
<p>17 people have thus far taken part
    in this survey:</p>
<p><table>
<tr><td>C/C++:</td><td>17.65 %</td></tr>
<tr><td>Java:</td><td>11.76 %</td></tr>
<tr><td>Perl:</td><td>17.65 %</td></tr>
<tr><td>PHP:</td><td>41.18 %</td></tr>
<tr><td>ASP[.NET] / C# / VB[.NET] / VBA:</td><td>11.76 %</td></tr>
<tr><td>Other:</td><td>0 %</td></tr>
</table></p>
</body></html>

```

3.5 改进意见

□ 布局：在这个案例里，没在窗口布局方面下什么功夫。从技术角度讲，用彩色直方图来显示问卷调查结果并不困难，但那与本书的主题（MySQL）并没有多大关系。如果有兴趣了解怎样才能把 HTML 页面弄得丰富多彩、怎样使用 PHP 的图象处理函数库进行编程等问题，有很

多这方面的图书及网上资源可以参考。

- **把调查问卷和调查结果显示在同一个页面上：**把这次网上调查活动的各个环节——显示调查问卷、处理问卷答案、显示调查结果——全部用一个 PHP 脚本来实现并不是不可能。可是在投票方式的民意调查活动中不让人们在回答问卷之前看到以前的调查结果比较好。
- **在问卷中多提供一些选择：**如果想让这次调查活动变得更有趣或收集更多的信息，在问卷中多提供一些选择将是个不错的主意。就拿这个案例来说，可以在问卷里增加一个文本框供人们输入其他的程序设计语言，还可以增加一个文本框供人们发表评论意见或是输入他们的职业背景信息。可是，不知读者是否想过，问的问题越多，收集到的虚假信息也就越多；如果参与者不想回答某个必须回答的问题，往往会随便选择一个答案。
- **允许多重选择：**从 PHP 编程的角度讲，允许同一个参与者在调查问卷里一次选择多种程序设计语言并不困难，只须在 HTML 表单里把单选按钮改为多选按钮就可以做到这一点。可是，在数据库的设计方面，为适应这一要求而必须对数据库设计方案做出的修改相当复杂：我们现在必须让 *votelanguage*¹ 数据表里的数据记录能够把那几种程序设计语言的任意组合保存起来。作为一种简单但缺乏效率的解决方案，我们可以在 *votelanguage* 数据表里增加几个数据列，让它们分别对应着一种程序设计语言，然后用 1 或 0 来代表人们是否把票投给了某种程序设计语言。如果不希望 *votelanguage*² 数据表占用太多内存空间的话，还可以把一个多重选择保存为一种二进制位组合——这需要用到 MySQL 提供的 *SET* 数据类型。不过，*SET* 数据类型虽然可以减少空间占用量，但处理起来却不容易：用来保存和处理问卷调查结果的程序代码将会变得相当复杂。
- **防止他人干扰调查活动或滥用调查结果：**绝大多数网上民意调查活动的目的不是为了收集信息而是为了传播某种观点或推销某种产品，许多网站和个人都把这种活动当做一种提高网页访问量的手段来使用。在这些网站和个人看来，防止他人干扰调查活动或滥用调查结果并没有多大的必要。话虽如此，如果确实需要一些保护措施的话，还是有一些办法可以供选择的：
 - 可以把一个随机生成的 cookie 放到参与者的计算机里去并同时把它存入数据库。这么一来，万一有人试图重复投票，就可以轻而易举地把他识别出来。（这个办法的弊端是：有许多因特网用户不喜欢 cookie，他们会把进入自己计算机的 cookie 随手删掉或根本不让 cookie 进入自己的计算机。）
 - 可以要求人们必须先用他们的电子邮件地址和密码进行注册，只有经过注册的人才允许进行投票。（这个办法的弊端是：因特网用户普遍对注册没什么耐心，为了能够回答调查问卷而愿意把自己的电子邮件地址公开的因特网用户恐怕会很少。）

一般来说，想彻底杜绝他人干扰调查活动或是滥用调查结果是不现实的。无数事实告诉我们，只要“坏人”肯下功夫，就可以突破或绕过任何一种信息安防机制。

提示 读者可以在笔者与 Hernd Oggie 合著的 *PHP 5 & MySQL* (Addison-Wesley 出版公司, 2005 年) 一书中找到大量与上述改进思路有关的参考示例。如果想搭建一个功能齐全的网上调查站点的话，现在就有许多现成的、用 PHP 语言开发的解决方案可以选择。在另起炉灶之前，最好先到因特网上的那些著名 PHP 站点看一看，如 <http://phpsurveyor.sourceforge.net/>。

1. 原书误写为 votes。——译者注
2. 原书误写为 votes。——译者注

Part 2

第二部分

管理工具和用户操作界面

本部分内容

- 第4章 mysql、mysqladmin 和 mysqldump
- 第5章 MySQL Administrator 和 MySQL Query Browser
- 第6章 phpMyAdmin
- 第7章 Microsoft Office 和 OpenOffice/StarOffice

第4章

mysql、mysqladmin 和 mysqldump

MySQL 是一种客户/服务器体系的数据库系统，它的服务器部分在启动运行后没有人机界面，所以终端用户无法直接使用 MySQL，只能通过各种 MySQL 客户程序或 Web 站点访问数据库、录入新数据和备份数据。

上述任务既可以用本章将要介绍的 mysql、mysqladmin 和 mysqldump 命令来完成，也可以用后面两章将要介绍的 MySQL Administrator、MySQL Query Browser 和 phpMyAdmin 工具完成。（还可以在因特网上找到许多其他的数据库管理工具，它们有些是免费的，有些是收费的商业化软件。但因为没有足够的篇幅，本书只能舍弃那些工具。）

mysql、mysqladmin 和 mysqldump 命令都是只支持文本模式，所以只能在 UNIX/Linux 系统的输入窗口或控制台窗口里执行。它们既没有菜单，也没有其他形式的助手机制。这种安排乍看起来似乎是一种缺陷，但在许多场合却是一种优势：首先，这些命令可以在网络上使用（如通过一个 ssl 控制台）；其次，它们很适合用来以自动方式完成各种系统管理工作（通过脚本编程）；最后，这些程序都很短小精悍，完全可以在多个窗口里同时打开它们以快速验证某种操作思路。对有经验的 MySQL 用户来说，这些程序就像一把瑞士军刀对于登山者那么重要。

提示 本章对 mysql、mysqladmin 和 mysqldump 等工具的介绍是很肤浅的。这些工具还有许多其他的功能，我们会在有需要的时候在本书的其他段落里再进行描述。例如，这些工具还提供了许多用于导入/导出文件和制作备份的操作选项，这些操作将集中在第 14 章加以讨论。第 22 章对这些工具的所有操作选项和功能进行了汇总。

4.1 mysql

在第 3 章里，介绍了 mysql 工具作为一个简单的命令解释器的使用方法，大家想必也都掌握了在文本模式下使用这个工具去执行 SQL 命令和查看结果数据表的技能。本节将对 mysql 工具在 MySQL 数据库系统里的其他用法和使用技巧进行介绍。

在继续前进之前，想再次提醒大家一句：mysql 是一个相对比较短小的程序，其用途仅限于把数据库操作命令发送给 MySQL 服务器和把那些命令的执行结果显示给用户。mysql 不是 MySQL 服务器，它只是供人们用来与 MySQL 服务器进行通信的一种客户端程序。

4.1.1 启动 mysql

- **UNIX/Linux 环境：**在 UNIX/Linux 系统中，只须简单地在控制台窗口里输入 mysql 命令再按回车键就可以启动这个工具。本章将要介绍的其他命令行工具（mysqladmin、mysqldump）也要用这个办法来启动。这几种工具的程序文件通常都安装在/usr/bin 子目录里，所以用户在启动的时候一般用不着再给出完整路径¹。
- **Windows 环境：**在 Windows 系统上，mysql 和其他一些命令工具大都存放在 C:\Programs\MySQL\MySQL Server n.n\bin 子目录里，但这个子目录往往不是系统变量 PATH（该变量的用途是为可执行程序定义搜索路径）的一部分。启动 mysql 程序的办法主要有以下几种：
 - 最简便的办法是执行菜单命令 Programs（程序）| MySQL | MySQL Server n.n | MySQL Command Line Client，其效果相当于使用 -u root -p 选项来启动 mysql 程序。在它被启动之后，mysql 将提示为 root 用户输入密码。上述菜单命令既不允许使用其他的选项来启动 mysql，也不允许执行 mysql 以外的其他命令。
 - 第二种办法是先用菜单命令 Programs（程序）| Accessories（附件）| Command Prompt（命令提示符）打开输入请求窗口，然后输入 mysql 程序的完整路径：
 > C:\Programs\MySQL\MySQL Server n.n\bin\mysql.exe
 - 如果想少打些字，可以利用 Windows 资源管理器和拖放操作把 mysql 程序的名字复制到输入请求窗口里。或者，还可以先用 CD 命令切换到 C:\Programs\MySQL\MySQL Server n.n\bin 子目录里。
 - 为了今后考虑，最方便的办法是对系统变量 PATH 进行扩展，让它把 MySQL 工具程序的路径也包括进去。具体做法是：执行菜单命令 Start（开始）| Settings（设置）| Control Panel（控制面板）| System（系统），选择 Advanced（高级）选项卡，双击选中 PATH 变量，然后把 MySQL 的 bin 子目录添加进去（如图 4-1 所示）。注意，PATH 变量里的路径名必须用分号隔开。这样做了之后，在任何一个命令窗口（不管其中的当前子目录或当前驱动器是什么）直接输入 MySQL 命令名就可以启动相应的程序了。

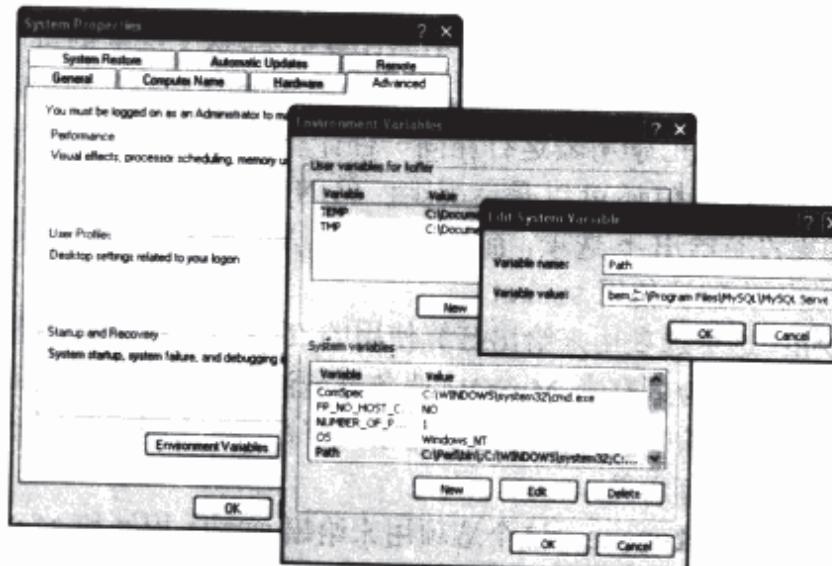


图 4-1 设置 Windows 系统变量

1. 原文在这里多了一句“See Figure 4-1”，但这个图只与 Windows 有关，所以省略了。——译者注

4.1.2 mysql 的命令行选项

在启动 mysql 工具的时候，有许多选项可供选择。连接 MySQL 服务器通常只需要用到两个选项，即 -u name 和 -p。下面描述了这两个和另外几个常用的选项，还给出了一些背景知识。这些选项并不是 mysql 独有的，包括 mysqladmin、mysqldump 在内的绝大多数 MySQL 命令工具都支持这些选项。

□ **-u name 或 -user=name:** 这个选项用来给出打算使用的 MySQL 用户名。根据 MySQL 的具体配置情况，系统里可能会有多个 MySQL 用户（请参见第 11 章）。如果从未创建过 MySQL 用户或者如果将要执行的是系统管理任务，就要在这里给出 *root* 作为用户名。*root* 用户在所有的 MySQL 安装里都存在，它就是所谓的“数据库系统管理员”。

如果删除 -u 选项，那么在 UNIX/Linux 系统下会使用当前登录名；而在 Windows 系统下则会使用 ODBC，只有在 MySQL 服务器上已经创建了这些用户名，登录才有可能实现。

□ **-p:** 这个选项的作用是让 mysql 在启动时提示人们输入密码。如果用 -u 选项指定的 MySQL 用户名有密码，就必须同时给出这个选项。

相反，如果命令中带有 password="xxx"，则用户可以指定密码。但这样做不太安全，因为命令会以纯文本格式显示在进程或任务栏内。

□ **-h computername 或 -host=computername:** mysql 程序不必与 MySQL 服务器运行在同一台计算机上也能与之进行通信。如果真的如此，就必须用 -h 选项给出 MySQL 服务器的主机名或 IP 地址。

使用 -h 选项必须满足一些先决条件。首先，名为 *computername* 的计算机必须是可以通过网络连通的；可以提前用 ping *computername* 命令来检查情况是否真的如此。其次，TCP/IP 3306 端口必须是可用的。如果在计算机与 MySQL 服务器主机之间有防火墙，应该先确认防火墙没有阻断这个端口上的通信再发出 mysql 命令；第三，远程的 MySQL 服务器必须被配置成允许使用本地计算机去连接它，但这不是 MySQL 服务器的默认配置。将在第 11 章详细讨论 MySQL 数据库系统的访问权限及其正确配置。

□ **-protocol=name:** 这个选项用来给出打算使用的通信协议。这个选项一般不需给出，由 mysql 程序自行选用的通信协议往往就是正确的。

如果 mysql 程序和 MySQL 服务器运行在不同计算机上，这个选项的可取值将只有 tcp 一个（使用 TCP/IP 协议进行通信）。如果已经给出了 -h 选项，mysql 将默认使用这个协议。

如果 mysql 程序和 MySQL 服务器运行在同一台计算机上，那么，根据 MySQL 服务器的具体配置，这个选项的可取值还可以是 socket（UNIX/Linux）、pipe（Windows）或 memory（Windows）。如果没有给出 -h 选项，mysql 程序在 Windows 环境下将默认使用 tcp 协议，在 UNIX/Linux 环境下将默认使用 socket。

□ **-P n 或 -port=n:** 这个选项用来给出打算使用的 TCP/IP 端口号。必须给出这个选项的场合可以说非常少，这是因为只有当 MySQL 服务器无法使用其默认通信端口（3306 号端口）时才需要指定另外一个通信端口。此外，这个选项只有在 mysql 程序和 MySQL 服务器使用 TCP/IP 协议进行通信的时候才有意义。

□ **-default-character-set=name:** 这个选项用来给出 mysql 程序和 MySQL 服务器进行通信时使用的字符集。这个字符集应该与在输入窗口（Windows）或控制台窗口（UNIX/Linux）里默认使用的字符集保持一致。MySQL 支持的字符集有以下几种：*latin1*（ISO-8559-1）、*latin2*（ISO-8559-2）、*utf8*（Unicode）和 *cp850*（西欧使用的 DOS 字符集）。

□ **databasename:** 作为 mysql 命令行的最后一个参数（注意：不是命令行选项），可以把打算使用的数据库的名字告诉 mysql 程序。则该数据库将成为所有后续命令的默认数据库。在 mysql 程序启动之后，随时都可以用 *USE tablename* 命令改变这个默认数据库。

示例：下面这条命令将尝试连接正在计算机 *uranus* 上运行的 MySQL 服务器。如果连接成功，名为 *mylibrary* 的数据库将成为默认数据库，双方通信时将使用 Unicode (*utf8*) 字符集。

```
> mysql -u root -p -h uranus --default-character-set=utf8 mylibrary
Password: *****
```

4.1.3 交互式使用 mysql

用 -u、-p 等选项启动 mysql 程序之后，就可以输入并执行 SQL 命令了。SQL 命令可以延续好几行，但必须用分号 (;) 结束。如果在启动 mysql 程序时没有指定一个默认数据库，执行的第一条 SQL 命令应该是 *USE tablename*。

```
> mysql -u root -p
Enter password: xxx
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 248 to server version: 5.0.2-alpha-standard
Type 'help;' or '\h' for help. Type '\c' to clear the buffer
mysql> USE mylibrary;
Database changed
mysql> SELECT titleID, title FROM titles LIMIT 10;
+-----+-----+
| titleID | title
+-----+-----+
| 11 | A Guide to the SQL Standard
| 52 | A Programmer's Introduction to PHP 4.0
| 19 | Alltid den där Annette
| 51 | Anklage Vatermord
| 78 | Apache Webserver 2.0
| 3 | Client/Server Survival Guide
| 63 | Comédia Infantil
| 77 | CSS-Praxis
| 86 | Dansläraren Återkomst
| 80 | Darwin's Radio
+-----+-----+
10 rows in set (0.01 sec)
```

注意 如果使用不当，mysql 程序有可能造成很大破坏。例如，*DELETE FROM tablename* 命令将删除 *tablename* 数据表里的全部数据，而这种删除是无法恢复的。如果想预防这类致命错误，就要在启动 mysql 程序时加上 *--i-am-a-dummy* 选项（意为“我是个新手”）。此后，如果发出的 *UPDATE* 和 *DELETE* 命令里没有关键字 *WHERE* 或 *LIMIT*，mysql 程序就会拒绝执行。此外还有其他一些安全措施。

除了 SQL 命令，还可以使用由 mysql 提供的一些命令（如 *source filename*）。这些命令既不需要以分号结束，也不区分字母的大小写（SQL 命令也如此）。这些命令每个都有一个双字符的简写形式；比如说，*help* 命令可以简写为 *\h*。简写命令只能出现在命令行的中间或末尾，如果是在行首，则必须使用它们的完整形式。表 4-1 只列出了几个最重要的命令，这些命令的完整清单参见第 22 章。

表 4-1 最重要的 mysql 命令

简写形式	命 令	说 明
\c	clear	放弃正在输入的命令
\h	help	显示一份命令清单
\q	exit 或 quit	退出 mysql 程序。UNIX/Linux 用户还可以使用 Ctrl+D 快捷键
\s	status	查看 MySQL 服务器的状态信息
\T [f]	tee [filename]	把输入和输出记载到指定文件里
\t	noteee	停用 tee 功能。此后，随时可以用 tee 或\T 命令把输入和输出继续记到刚才的指定文件里，用不着再给出它的文件名
\u db	use database	另行指定一个默认数据库
\. fn	source filename	读取并执行某给定文件里的 SQL 命令。那些命令必须以分号隔开

1. 简化输入

□ **放弃正在输入的命令。**在输入一条比较长的命令时，出现打字错误是在所难免的。在这种情况下，放弃正在输入的命令重头再来往往会是更好的选择。这在 mysql 程序里很容易做到：不管光标位置在哪里，只须输入\c 再按回车键就可以退出当前命令行。不过，万一光标位置正处于某个已经用单引号或双引号括了起来的字符串的中间，\c 就将被认为是那个字符串的一部分而起不到退出当前命令行的作用。

□ **历史记录。**mysql 可以记住人们曾经输入过的命令（即使是在程序中断之后也是如此）。这些命令可以用键盘上的上、下箭头键调出。

2. 退出mysql程序

使用命令 *exit*、*quit*、*\q* 和快捷键“Ctrl+D”（仅限于 UNIX/Linux 环境）都可以退出 mysql 程序。Windows 用户还有一个更简单的办法可用：直接关闭相应的命令窗口。

4.1.4 UNIX/Linux 环境中 mysql 的使用技巧

□ **滚动区域。**在 UNIX/Linux 环境下，mysql 程序通常运行在 shell 窗口里（如图 4-2 所示）。这些 shell 窗口程序（xterm、console、gnome-terminal 等）都可以把一定行数的屏显内容保存在一个临时缓存区里，而这个行数还允许调整。只要把这个行数设置为一个足够大的值，就可以利用滚动条查看自己以前输入的命令，还可以用鼠标把它们复制到剪贴板。

```

File Edit View Terminal Title Help
[redacted] -> mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.2-alpha-standard
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT * FROM titles LIMIT 10;
Query OK, 0 rows affected (0.02 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> SELECT titleID, title FROM titles LIMIT 10;
+-----+-----+
| titleID | title |
+-----+-----+
| 11 | A Guide to the SQL Standard |
| 52 | A Programmer's Introduction to PHP 4.0 |
| 19 | Alltid den där Annette |
| 51 | Anklage Vatneord |
| 78 | Apache Webserver 2.0 |
| 3 | Client/Server Survival Guide |
| 63 | Cowabba Infantil |
| 77 | CSS-Praxis |
| 86 | Denaläkarens Återkomst |
| 80 | Darwin's Radio |
+-----+-----+
10 rows in set (0.00 sec)

mysql> 
```

图 4-2 在一个 Linux 控制台窗口里使用 MySQL 命令解释器

- **键盘快捷键。**在绝大多数的操作系统下，mysql 程序都必须依赖 readline 函数库才能工作。这就使 UNIX/Linux 环境里的许多文本编辑快捷键也可以用来编辑在 mysql 程序里的输入（例如，快捷键 Ctrl+K 可以从光标的当前位置删除到行尾，Ctrl+Y 可以恢复刚刚删除的文本等）。这些快捷键中的绝大多数都与它们在 Emacs 编辑器里的用途和用法相一致。
- 此外，在输入数据表或数据列的名字时，如果先输入它们的前几个字符再按下 Tab 键，系统还会自动补足剩下的部分。
- **个人配置文件。**在 UNIX/Linux 环境下，可以把自己常用的设置参数和选项（如用户名和密码）保存到一个名为~/.my.cnf 的用户级配置文件里去。具体地说，把适用于所有客户工具的选项集中安排在[client]部分里，把只适用于 mysql 程序的选项集中安排在[mysql]部分里。下面就是一个这样的例子（创建用户级配置文件的细节步骤参见第 22 章）。

```
# Options for all MySQL tools
[client]
user=username
password=xxxx
# Options for mysql
[mysql]
database=mydatabase
```

因为这个文件里包含着明文形式的密码，所以应该将其隐藏不让别人偷看到：

```
user$ chmod 600 ~/.my.cnf
```

字符集问题

现在，有许多主流的 Linux 发行版本都把 *utf8* 作为它们的默认字符集；或者使用 *latin1*(ISO-8859-1) 或 *latin2*(ISO-8859-2) 字符集。如果 mysql 程序和 MySQL 服务器在通信时使用的字符集不是同一种，就会造成某些不常用的特殊字符无法正确显示。这类问题基本上都可以用以下两种方法来解决。

- 最简单的办法是：像往常一样启动 mysql 程序，启动后立刻执行 SQL 命令 *SET NAMES 'name'*，命令中的 *name* 是控制台窗口能够支持的某种字符集的名字。MySQL 支持的字符集在数量上要比 UNIX/Linux 操作系统支持的少很多，但选择 *latin1* 或 *utf8* 字符集应该不会有什么问题。（MySQL 不支持 *latin9* 字符集，应该把它替换为 *latin1* 字符集。这两种字符集的差异是欧元符号。）
- 另一种办法是在 mysql 程序的启动命令里增加一个`-default-character-set=name` 选项，但要把这里的 *name* 替换为打算使用的字符集的名字。

有一个简单的办法可以查出 mysql 程序和 MySQL 服务器在通信时使用的是什么字符集：在 mysql 程序里执行 *status* 命令：字符集信息出现在 *Server* 和 *Conn. character set* 行上。

4.1.5 Windows 环境下 mysql 的使用技巧

在 Windows 环境下，mysql 通常运行在输入窗口（这种窗口有时也被称为“命令窗口”或“DOS 窗口”）。输入窗口的许多属性都是可配置的。用鼠标右键单击输入窗口的标题栏就可以进入它的配置对话框（如图 4-3 所示）。

- **颜色。**Windows 的命令窗口通常是黑底白字，看时间长了眼睛可能会不太舒服。如果更喜欢白底黑字的显示效果，在 *Color*（颜色）选项卡里做出相应的修改即可。
- **窗口尺寸。**用一个尺寸是 80×25 字符的命令窗口去查看 *SELECT* 命令的查询结果往往会有许多内容看不到。*Layout*（布局）选项卡可以在这方面提供一些帮助，可以通过这个选项卡对命令窗口的尺寸做出一些调整。

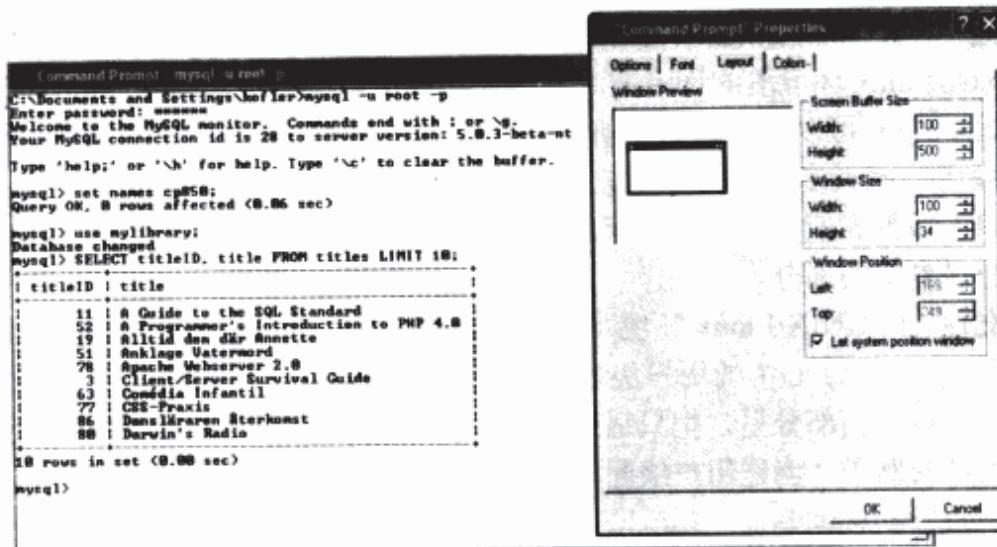


图 4-3 在 Windows 环境下配置 MySQL 命令解释器

- **滚动区域。**窗口越大，显示的东西就越多，但也就仅此而已了。这里所说的滚动区域指的是可以通过 Layout (布局) 选项卡加以调整的 Screen Buffer Size (屏幕缓冲区尺寸)，这项设置对应的是虚拟窗口的尺寸。人们在命令窗口里看到的东西只是整个虚拟窗口的一部分，可以用拖动滚动条的办法去查看自己以前执行过的命令及其结果。
- **剪贴板。**Windows 的命令窗口允许人们利用剪贴板对屏显文本进行复制和粘贴。只须在命令窗口选取一个方形区域再按回车键，就可以把它复制到剪贴板上；在命令窗口里单击鼠标右键，就可以把剪贴板上的东西粘贴回来。不过，必须先在 Option(选项)选项卡里激活 QuickEdit Mode (快捷编辑模式) 选项才能使用这些便捷功能。
在关闭命令窗口的配置对话框时，系统会询问：刚才做出的改动是仅适用于当前的命令窗口、还是要把它们保存起来；后一种选择将影响到所有的命令窗口。建议大家选择后一种，这将使以后打开的命令窗口都采用在此设置的属性。

字符集问题

MySQL 服务器使用的默认字符集通常是 *latin1* 或 *utf8*，但 Windows 命令窗口使用的字符集却是与 DOS 兼容的 *cp850*。因此，会有许多特殊字符（比如 àäèéöüß）在命令窗口里无法正确显示。更糟糕的是，在 Windows 命令窗口里用 SQL 命令 *INSERT* 或 *UPDATE* 存入数据库的数据记录将沿用 DOS 字符集，等以后用另一种程序（比如 phpMyAdmin）去查看这些数据记录时，就会有许多特殊字符无法正确显示。有以下几种方法可以解决这类问题：

- 最简单的办法是：像往常一样启动 mysql 程序，启动后立刻执行 SQL 命令 *SET NAMES cp850*。这将使 MySQL 服务器在与 mysql 程序进行通信时使用 *cp850* 字符集，确保 MySQL 服务器能按照要求接受所有的字符串。
- 在 mysql 程序的启动命令里增加一个 *-default-character-set=cp850* 选项也可以获得同样的效果。不过，系统可能会返回一条错消息：Character set 'cp850' is not a compiled character and is not specified in the ... file ('cp850' 字符集没有经过编译，也没有在……文件里指定)。这条出错消息的含义是：mysql 找不到对 *cp850* 字符集做出描述的文件。解决这个问题的办法是在 mysql 程序的启动命令里再增加一个 *--character-sets-dir=C:/Programs/MySQL/MySQL Server 5.0/share/characters* 选项，务必要把这个选项中的路径替换为自己的 MySQL 字符集路径。这个办法的缺点是需要手工输入很多字。

□ 第三种办法是在 Windows 命令窗口所使用的字体和字符集上动脑筋。

因为 Windows 命令窗口所使用的 Raster Fonts 字体与绝大多数字符集都不兼容，所以必须改用别的字体才行。用鼠标右键单击 Windows 命令窗口的标题栏，选中 Properties（属性），然后把字体设置为 Lucida Console。

接下来，在启动 mysql 程序之前，还需要选择一种正确的字符集。如果 MySQL 服务器使用的默认字符集是 *latin1*，请在命令窗口里输入并执行 CHCP 1252¹命令以启用 *Codepage 1252*。（*Codepage* 是 Windows 对字符集的称呼。*Codepage 1252* 大致对应于 *Latin1* 字符集。）完成这些准备工作之后，重新启动 mysql 程序。

如果 MySQL 服务器使用的默认字符集是 *utf8*，执行 CHCP 65001 命令。（Windows 中的 *Codepage 65001* 对应于 Unicode 字符集 *utf8*。）

有一个简单的办法可以查出 mysql 程序和 MySQL 服务器在通信时使用的是什么字符集：在 mysql 程序里执行 *status* 命令；字符集信息出现在 *Server* 和 *Conn. character set* 行上。

4.1.6 用 mysql 处理 SQL 文件

通过前面的学习，大家想必都已经知道如何利用 mysql 程序去以交互方式执行 SQL 命令了，但是是否知道 mysql 程序还可以用来处理存放在文件里的 SQL 命令呢？这其实非常简单，只须按如下所示启动 mysql 程序即可：

```
> mysql [options] databasename < file.sql
```

上面这条命令将以 *databasename* 数据库为操作对象执行 *file.sql* 文件所包含的全部 SQL 命令。请注意，这种 SQL 批处理文件不必一定以 *.sql* 作为后缀名；文件的内容才是最关键的，再就是这类文件里的命令都必须以分号（;）隔开才行。（可以在这类文件里用 *DELIMITER symbol* 命令定义一个不同的分隔符，而这对那些包含着 *CREATE FUNCTION*、*CREATE PROCEDURE* 或 *CREATE TRIGGER* 语句的 SQL 批处理文件来说是必然的——分号是这几种语句的组成部分之一。）

读入数据库备份

在实际工作中，很可能会经常遇到一些由 mysqldump 命令生成的数据库备份文件，这类文件多以 **.sql* 文件的面目出现。为了读取这类文件，通常需要先用 mysqladmin 命令创建一些新的数据库以便容纳数据。下面两条命令演示了这一过程。（我们马上就会讨论到 mysqladmin 和 mysqldump 程序的用法。）

```
> mysqladmin -u root -p create dbname
Password: *****
> mysql -u root -p dbname < backupfile.sql
Password: *****
```

提示 由最新版本的 mysqldump 程序生成的数据库备份包含有为备份文件设置字符集的 SQL 命令，所以不必担心字符串集方面的问题。

如果数据库备份是用 mysqldump 程序的早期版本或者是另外一种程序（比如 phpMyAdmin）制作的，字符集信息很可能会缺失。为了正确地读入一个这样的备份文件，必须在 mysql 程序的启动命令里增加一个 *-default-character-set= ...* 选项。如果拿不准应该选用哪一种字符集，那就使用 *latin1* 好了；这个字符集是 MySQL 4.0 系列版本的默认字符集，而人们在安装 MySQL 软件时改用其他字符集的情况不多见。

1. 原书在这里误写为“CHCP 65001”。——译者注

```
> mysql -u root -p --default-character-set=latin1 name < backup.sql
```

用最新版本的phpMyAdmin工具制作的数据库备份使用utf8作为它们的默认字符集。

4.2 mysqladmin

mysqladmin程序有助于完成许多种系统管理任务，如创建或删除一个数据库、修改密码等。它的命令语法很简单：

```
> mysqladmin [options] admincommand
```

mysqladmin用来连接数据库服务器的选项与mysql完全一样（-u、-p、-h等）。*admincommand*是mysqladmin将要执行的任务。在此只介绍3个最重要的命令：*create*，创建数据库；*delete*，删除数据库；*password*，改变由-u选项给出的MySQL用户的密码。

```
> mysqladmin -u root -p create newdatabase
Password: *****
> mysqladmin -u root -p drop testdatabase
Password: *****
Dropping the database is potentially a very bad thing to do.
Any data stored in the database will be destroyed.
Do you really want to drop the 'testdatabase' database [y/N] y
Database "'testdatabase'" dropped
> mysqladmin -u root -p password "new password"
Password: old password
```

4.3 mysqldump

mysqldump程序的用途是为数据库创建备份，在它生成的结果文件里包含着创建必要的数据表和把有关数据插入这些数据表的SQL命令。下面是mysqldump命令的语法：

```
> mysqldump [options] dbname > backupfile.sql
```

mysqldump用来连接数据库服务器的选项与mysql完全一样（-u、-p、-h等）。此外，还可以通过许多其他的选项来调控备份工作的细节，但那些额外的选项对数据库的日常备份工作来说并不是必要的。

备份出来的结果文件是一个文本文件。最新版本的mysqldump程序使用utf8作为自己的默认字符集。如果想使用另外一种字符集，就必须用--default-character-set=...选项来设置。

第 5 章

MySQL Administrator 和 MySQL Query Browser

MySQL Administrator 和 MySQL Query Browser 是两种相对较新的 MySQL 用户界面。就像它们的名字所暗示的那样, MySQL Administrator 的侧重点是帮助人们完成各种系统管理任务, MySQL Query Browser 的侧重点则是帮助人们编写和调试 SQL 命令。

与其他同类软件工具(比如将在第 6 章介绍的 phpMyAdmin)不同, MySQL Administrator 和 MySQL Query Browser 是由 MySQL 公司一手开发的, 这使得这两个程序多少带有一点儿名门正宗的色彩。它们分别取代了早期 MySQL 版本中的 WinMySQLAdmin 和 MySQL ControlCenter 工具(这两种工具的开发工作已正式宣告终止)。

本章的讨论重点是 MySQL Administrator 1.0.19 版和 MySQL Query Browser 1.1.5 版, 它们都是 2005 年 2 月才发布的最新版本。虽然 MySQL 公司认为它们都已经相当稳定(它们都是 Generally Available 版本), 但是人们还是不断地发现它们存在着种种明显的问题和不稳定性。换句话说, 这两个程序的功能都很丰富, 可是随时都有可能发生崩溃。因此, 如果打算安装这两个程序, 就一定要到 dev.mysql.com 网站上去看看有没有最近推出的新版本。

提示 MySQL Administrator 和 MySQL Query Browser 程序使许多种系统管理任务和 SQL 命令调试大为简化。不用说, 熟悉 MySQL、系统管理、数据库设计、信息安全机制和 SQL 语法将是用好这两个程序的先决条件, 第 8 章~第 14 章将深入介绍这些基础知识。本章内容属于介绍性质, 目的是为了对这些话题有所领略, 因而不会涉及很多的细节或背景。

5.1 安装

- **Windows:** 这两个程序的*.msi 安装文件(Microsoft Installer)可以从 dev.mysql.com 网站自由(按照 GPL 许可证对自由的解释)下载。安装过程很简单, 双击下载到的*.msi 安装文件即可。在默认的情况下, 系统将把有关文件安装到子目录 C:\Programs\MySQL 里。安装工作结束后, 使用菜单命令 Start(开始)|Programs(程序)|MySQL|Name 就可以启动它们。
- **Linux(i386 兼容的 RPM 系统):** 这两个程序已被收录在了最新的 Linux 发行版本里, 可以用 Linux 发行版本自带的软件包管理工具(如 SUSE 发行版本中的 YaST)来安装它们。如果用户的 Linux 发行版本没有收录这两个程序, dev.mysql.com 网站上有适用于 i386 兼容处理器的预编译 RPM 软件包供人们下载。下载*.rpm 文件之后, 执行 rpm -i packagename.rpm 命令即可完成安装。安装工作结束后, 执行 mysql-administrator 或 mysql-query-browser

命令就可以启动这两个程序。

- **Linux (i386 兼容的其他软件包系统)**：如果正在使用的是采用其他软件包管理机制的 Linux 发行版本（比如 Debian），需要找到这两个程序预编译好的 tar.gz 软件包。这种软件包的安装工作比较复杂。下载有关文件之后，先要把它们解压缩到/usr/local 子目录里：

```
root# cd /usr/local
root# tar xzf mysql-query-browser-<version>-linux.tar.gz
root# tar xzf mysql-administrator-<version>-linux.tar.gz
```

然后，在这两个程序的启动脚本里对变量 **MYPATH** 做如下修改（黑体字部分）：

```
...
# change in /usr/local/mysql-query-browser/bin/mysql-query-browser
...
MYPATH=/usr/local/mysql-query-browser/bin
# change in /usr/local/mysql-administrator/bin/mysql-administrator
...
MYPATH=/usr/local/mysql-administrator/bin
```

最后，创建两个如下所示符号链接，让系统上的每一个用户都能方便快捷地启动这两个程序：

```
root# cd /usr/bin/X11
root# ln -s /usr/local/mysql-administrator/bin/mysql-administrator .
root# ln -s /usr/local/mysql-query-browser/bin/mysql-query-browser .
```

- **UNIX/Linux/Mac OS X**：如果使用的不是 i386 兼容系统（比如 PowerPC）或者使用的是另外一种 UNIX 变体，就只能自行编译这两个程序了。这两个程序的源代码和安装指南都可以在 dev.mysql.com 网站上找到。

适用于 Max OS X 操作系统的稳定版本在 2005 年 2 月时还未出现，但应该很快就会发布。使用苹果电脑的朋友如果不打算试用测试版本，就只能用别的系统管理工具再坚持一段时间了。

5.2 与 MySQL 服务器建立连接

在启动 MySQL Administrator 或 MySQL Query Browser 程序之后，必须先与 MySQL 服务器建立连接（如图 5-1 所示）才能开始使用这两个程序去干活。如果没有特意说明，本节讨论内容将同时适用于这两种程序，因为它们采用了同样的机制来与 MySQL 服务器建立连接。

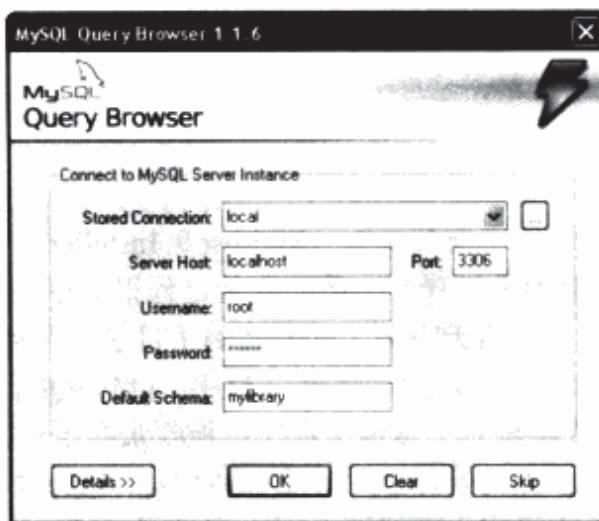


图 5-1 MySQL Query Browser 的连接对话框

需要向图 5-1 所示的对话框提供以下几项信息：

- **Stored Connection:** 在 Windows 环境下, 这项信息等于是为接下来填写的连接参数起个名字。这样一来, 等以后再启动这两个程序的时候, 就可以直接使用这个名字调出有关的连接数据而用不着再把所有的参数都重新填写一遍。在需要通过这两个程序去访问多个 MySQL 服务器的时候, 这个字段将会非常有实用价值。
在 Linux 环境下, 把连接数据保存起来要比较麻烦一些。在填写好所有的参数之后, 将需要先从一个列表框里选择 Save This Connection (保存这条连接) 选项, 再在弹出的对话框里为新连接起一个名字。
- **Server Host:** 将需要在这里给出 MySQL 服务器的主机名。如果它是一台本地计算机, 填写 *localhost* 即可。
- **Port:** 在默认的情况下, 新连接将使用 TCP/IP 协议进行通信, 而此种情况下的 MySQL 服务器将使用 3306 号端口。不要把这个字段随手设置为其他的端口号, 除非试图连接的 MySQL 服务器正在使用其他的端口进行通信。
- **Username 和 Password:** 需要在*这里*给出一个 MySQL 用户名 (比如 *root*) 和与之相关联的密码。基于信息安全方面的考虑, 在这里输入的密码不会随其他连接数据一同保存起来, 只能在每次建立连接时亲手输入它。
如果嫌麻烦, 这里有一个简单的解决办法: 在第一次建立起连接之后, 执行菜单命令 Tools (工具) | Options (选项) | General Options (常规选项), 然后激活 Store Password (保存密码) 选项。(请注意, 这么做是要冒风险的, 因为谁都不能保证别人找不到密码文件。)
- **Default Schema (仅适用于 MySQL Query Browser):** 这个输入字段是为了给将要执行的 SQL 命令指定一个默认数据库用的。这项输入是可选的; 即使把这个字段留做空白, 也可以在稍后通过菜单命令 File (文件) | Change Default Schema (改变默认数据库) 选择一个数据库。
- **Details:** 只有极少的特殊场合才需要在这个字段输入一些可选的连接参数。比如说, 如果打算连接的 MySQL 服务器出于信息安全方面的考虑不使用 TCP/IP 协议进行通信, 那么 Windows 用户将需要在此选择 Named Pipe (命名管道) 并给出一个命名管道名, Linux 用户则需要在此选择 Socket File (套接字文件) 并给出一个套接字文件名。(对这几个术语的解释见第 11 章。)

提示 MySQL Administrator能够从本地计算机启动MySQL服务器。具体做法是: 在连接对话框 (如图 5-1 所示) 里按键盘上的 Ctrl 键, 这将把 OK 按钮变成 Skip 按钮, 就可以跳过连接对话框 (连接对话框只有在 MySQL 服务器已经运行的情况下才有用) 直接进入 MySQL Administrator 了。在 MySQL 服务器启动之前, MySQL Administrator 只有少数几种功能是可用的。

5.3 MySQL Administrator

MySQL Administrator 可以帮助人们管理 MySQL 服务器 (如加载数据、统计已经打开的连接有多少条、查看服务器的工作状态等) 和执行多种系统管理任务。这个程序既可以对本地数据库服务器进行管理, 也可以对网络上的远程数据库服务器进行管理。

这个程序的许多功能只能用于某种特定的场合, 这些场合包括:

- 只能用于对本地服务器进行管理的场合;
- 只允许有足够的访问权限的用户 (*root*) 使用;
- 只允许有足够的操作系统访问权限的用户使用。

例如，只有在登录为 *root* 用户（UNIX/Linux）或系统管理员（Windows）的场合才能启动或停止一台本地 MySQL 服务器的运行。

MySQL Administrator 的众多功能分散在几大功能模块里，使用者需要先通过一个列表来选择一个功能模块，再通过一系列选项卡来使用各项功能。在接下来的几个小节里，我们将对这些功能模块做一个简单的介绍。

5.3.1 Server Information 模块（查看服务器信息）

可以通过这个模块查看 MySQL 服务器的工作状态、网络连接信息、MySQL 服务器的版本、客户端信息以及关于 MySQL Administrator 所使用的客户软件开发库的信息（如图 5-2 所示）。

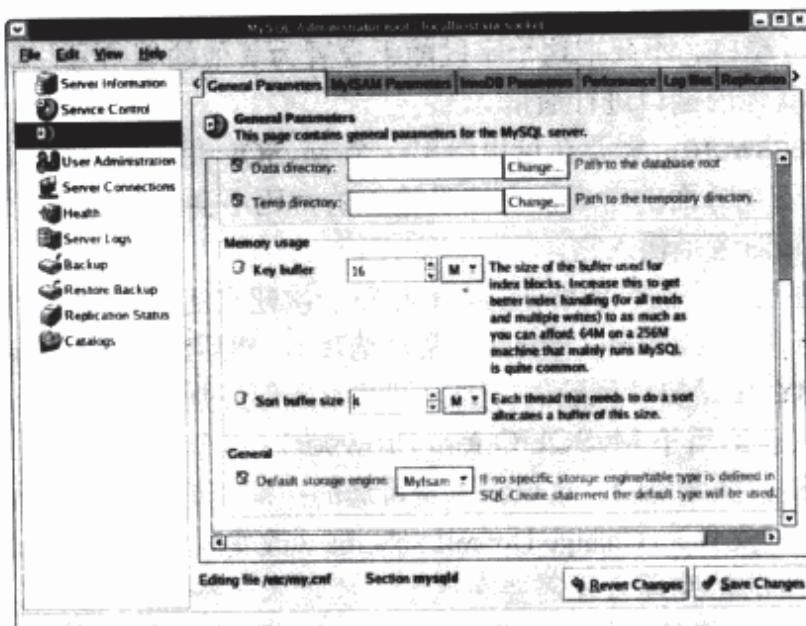


图 5-2 用 MySQL Administrator 查看服务器信息

5.3.2 Service Control 模块（启动/停止 MySQL 服务器）

可以通过这个模块来启动和停止 MySQL 服务器的运行，但必须同时满足两个条件：一是仅限于控制本地 MySQL 服务器（对网络中的远程 MySQL 服务器无效）；二是 MySQL Administrator 本身必须是由某位拥有系统管理员权限的用户启动的。在笔者进行的测试中，MySQL Administrator 程序的 Linux 版本在启动和关闭 MySQL 服务器时发生了崩溃，Windows 版本没有出现这样的问题。

在 MySQL Administrator 程序的 Windows 版本里，可以在 Configure Service（服务配置）选项卡里对许多选项进行设置。这个程序的 Linux 版本没有提供这个选项卡，只能在 Startup Parameters（启动参数）选项卡里设置一些选项（见 5.3.3 节）。

5.3.3 Startup Variables 模块（配置启动参数）

在这个模块里有许多选项卡（如图 5-3 所示），可以通过它们方便地对 MySQL 配置文件/etc/my.cnf（UNIX/Linux）或 C:\Programs\MySQL\MySQL Server n.n\my.ini（Windows）里的许多选项进行修改。MySQL 配置文件是一个文本文件，它里面包含着无数的选项，将在第 14 章和第 22 章对其中最重要的一些选项进行描述。这些选项控制着 MySQL 服务器的行为，但 MySQL 服务器只在启动时才会去读取它们。因此，对 MySQL 配置文件的修改只有重新启动一次 MySQL 服务器才会生效。

如果 MySQL 配置文件尚不存在，系统会在有人进行过一次查询之后自动创建它。对这个文件进行修改的先决条件是 MySQL Administrator 具有这个文件的写权限。万一 MySQL Administrator 找不到这个文件，就必须要由用户去搜索它了。

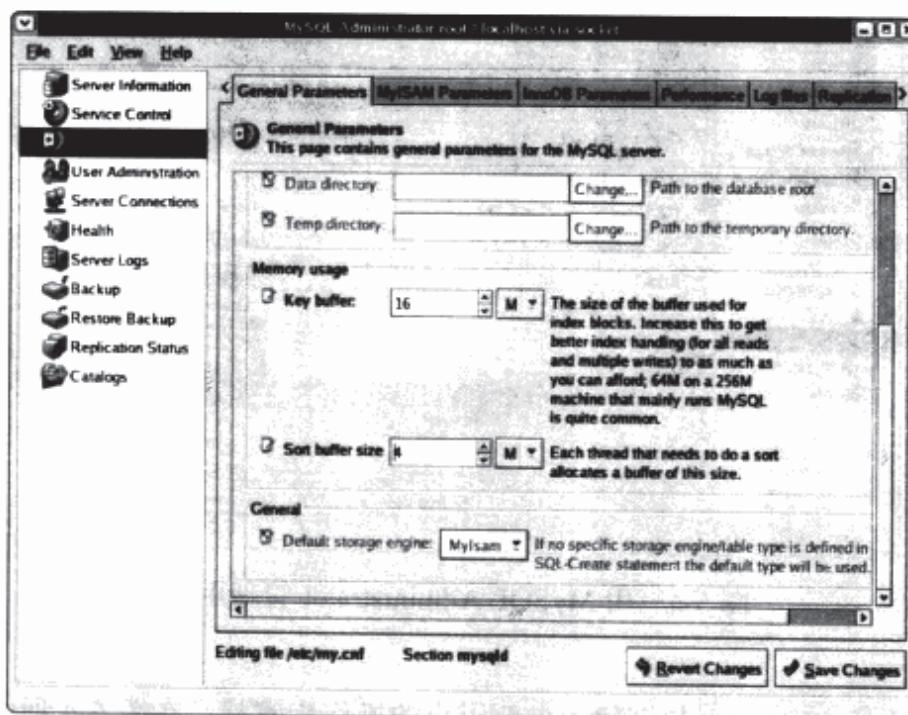


图 5-3 修改 MySQL 服务器的启动参数

MySQL Administrator 在每个选项的前面会显示一个小按钮，这些按钮可以告诉用户某个选项是否已被激活（红色的 X 标记）。未被激活的选项不会出现在配置文件里；对于这一部分选项，MySQL 服务器将使用内部的默认值。

5.3.4 User Administration 模块（用户管理）

这个模块可以帮助人们创建新的 MySQL 用户和改变现有用户的访问权限。这些工作需要对 MySQL 的信息安防机制有一个清晰的了解（参见第 11 章）。User Administration 模块在某些方面与类似的系统管理工具（比如 phpMyAdmin）有着较大的差异，很值得我们深究一下它的使用细节。

1. 为多个用户/主机组合修改密码

MySQL 的访问控制机制可以根据某位用户在连接 MySQL 服务器时具体使用的客户机位置来决定应该把哪些权限授予这位用户。这一点在 MySQL Administrator 工具里的体现是每位用户的名下都有一个主机名列表，表中列出的主机就是允许这位用户用来访问 MySQL 服务器的计算机。比如说，在图 5-4 里，可以看到用户 *root* 允许从名为“localhost”、“saturn.sol”¹ 的计算机去连接 MySQL 服务器。

从信息安全的角度讲，用户名和主机名的每一种组合都应该有它自己的密码。但在实践中，同一用户在多台主机上使用同一个密码的情况十分常见。MySQL Administrator 工具提供了一种简单的办法来把同一位用户在所有可用主机上的密码修改为同一个。具体做法是：在图 5-4 所示的对话框里，只选中某个用户名，不要再选主机名（即不选择@*name*），然后修改该用户的密码。这样，该用户在所有可用主机上的密码就一次性地被修改成同一个了。

1. 原文内容与图中文字不符。——译者注

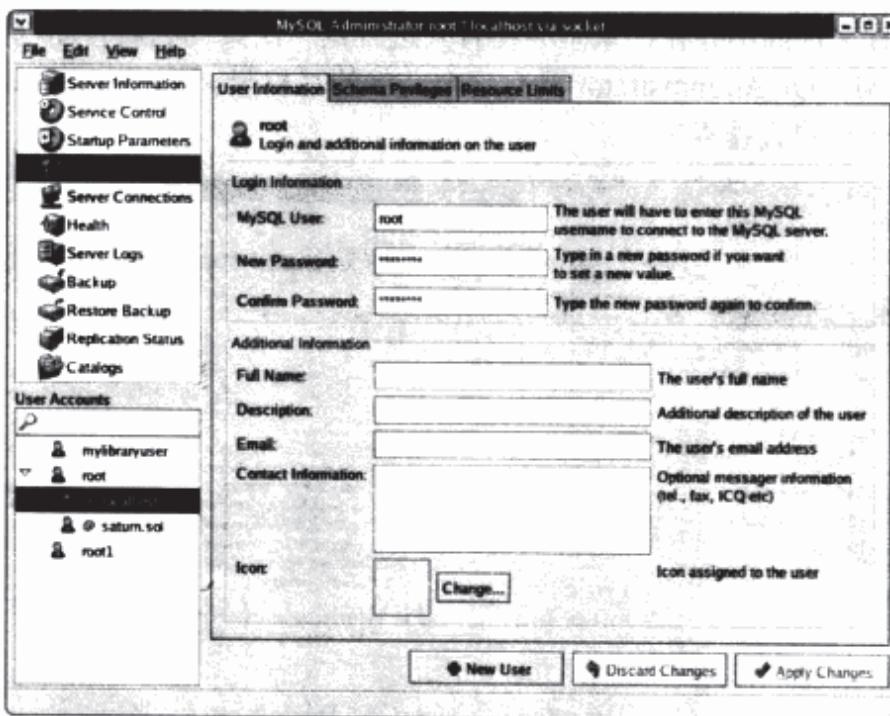


图 5-4 用 MySQL Administrator 管理用户

2. 创建新用户

使用 MySQL Administrator 工具创建一个新用户的操作步骤是：在图 5-4 所示的对话框里，单击 New User（新用户）按钮，输入新的用户名和密码，最后单击 Apply Changes（应用）按钮加以确认。这里要特别提醒大家注意一点：MySQL Administrator 在创建新用户的时候会把%用做主机名，而这种做法的效果是新用户将有权从网络中的任意位置去访问 MySQL。（“%”是一个用来匹配任意主机名的通配符。它在 MySQL Administrator 里是无法直接看见的，只有查看 *mysql.user* 数据表才能知道 MySQL Administrator 在创建新用户时做了些什么。）

可是，如此宽松的访问控制很容易导致严重的安全问题，所以应该限制新用户只能使用特定的计算机访问 MySQL。具体做法是：在图 5-4 所示的对话框里用鼠标右键单击新用户的用户名，通过弹出菜单执行 Add Host（添加主机）命令，输入预定的主机名（如 *localhost*）。这样，在 *mysql.user* 数据表里，在新用户的名下将会有两个主机名设置项，一个是%，另一个是 *localhost*。接下来，在图 5-4 所示的对话框里用鼠标右键再次单击新用户的用户名，通过弹出菜单执行 Remove Host（删除主机）命令，将删掉主机名“%”。

3. 设置操作权限

MySQL 把操作权限分为“全局权限”（访问权限）和“对象权限”两大类，前者适用于全体数据库，后者只适用于特定的数据库、数据表和数据列。在默认的情况下，MySQL Administrator 在查看对象权限的时候只显示到数据库级别（见图 5-4 中的 Schema Privileges 选项卡）。如果想查看全局权限以及数据表和数据列级别的对象权限，就必须执行菜单命令 Tools（工具）| Options（选项）| Administrator（系统管理员）（Windows 环境）或 File（文件）| Preferences（首选项）| Administrator（系统管理员）（UNIX/Linux 环境），打开配置对话框，选择相应的选项。

把操作权限授予某个用户或者收回它们的具体做法是：先在相应的选项卡选中想要设置的对象（比如说，某个数据库），然后在 Available Privileges（可用权限）列表里选中有关的权限，再单击箭头图案的按钮把那些选中的权限放到 Assigned Privileges（已分配权限）列表里，如图 5-5 所示。

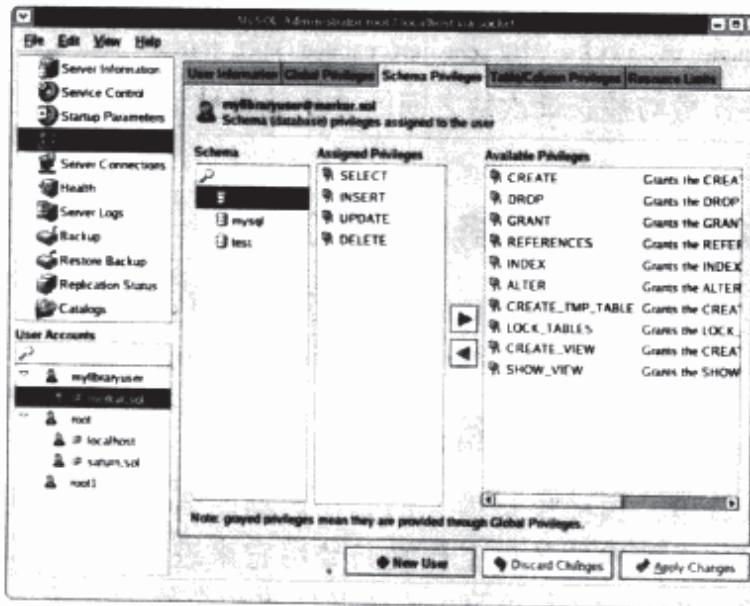


图 5-5 用 MySQL Administrator 管理访问权限

5.3.5 Server Connections 模块（查看服务器连接信息）

这个模块有以下两个选项卡：

- Thread (线程) 选项卡。** 这个选项卡把所有活跃的 MySQL 线程列成一份清单（相当于 SQL 命令 *SHOW PROCESSLIST*）显示出来。
- User Connection (用户连接) 选项卡。** 这个选项卡把所有活跃的 MySQL 用户和他们的线程列成一份清单显示出来。这些数据来自 SQL 命令 *SHOW PRIVILEGES*，唯一的区别是这份清单的内容是按照用户名分组排列的。这份清单可以告诉我们哪些用户打开的数据库连接最多。

5.3.6 Health 模块（查看系统负载）

这个模块有以下 4 个选项卡，可以通过它们了解 MySQL 服务器的使用和负载情况：

- Connection Health (连接负载) 选项卡。** 这个选项卡上有 3 个图表，它们给出的信息包括：当前活跃连接的数量、网络通信流量、每秒处理的 SQL 查询次数等。这些图表对那些负荷沉重的 MySQL 服务器很有用，但这种事一般只会发生在比较热门的 MySQL 服务器上，开发用途的计算机一般不会出现负荷过大的情况。
- Memory Health (内存消耗情况) 选项卡。** 这个选项卡给出了两个重要的内存区域——Query Cache（查询缓存区）和 Key Buffer（索引缓冲区）——的使用情况。
- Status Variables (状态变量) 和 Server Variables (服务器变量) 选项卡。** 这两个选项卡使用了一些数据表来管理各种各样的状态信息。

这个模块特别值得一提的地方是可以让人们只需付出很小的努力就定制出丰富的图形显示效果。我们可以利用鼠标右键添加新的图表、甚至是新的选项卡。（MySQL Administrator 工具的 Windows 版本比它的 Linux 版本提供了更多的方便，并且还允许人们对现有图形进行定制。）

在定制一个图表时，可以使用 Status Variables（状态变量）和 Server Variables（服务器变量）选项卡里的所有变量。变量名必须放在方括号里。如果在某个变量名的前面加上一个可选的“^”字符，系统在生成图表时使用的将是这个变量自上次刷新有关图表以来的相对值，而不是这个变量的绝对值。比如说，我们可以写出一个如下所示的 Query Cache Hitrate（查询缓存区命中率）的计算公式：

```
(^[_Qcache_hits] /  
(^[_Qcache_hits] + ^[_QCache_inserts] + ^[_QCache_not_cached])) * 100
```

这样，这个缓存区的被命中次数与被查询次数之间的关系就将被计算和表示为一个百分比数字（0~100之间的某个值）。

5.3.7 Server Logs 模块（查看服务器日志）

这个模块可以让用户方便地查看 MySQL 服务器的日志数据，但前提是有关的日志功能必须已被激活。与日志有关的选项可以通过 Startup Parameters 模块的 Log Files（日志文件）选项卡修改。

5.3.8 Backup 模块（制作数据库备份）

从模块的名字就可以看出，该模块是供人们制作数据库备份用的。用这个模块来制作数据库备份的步骤比较繁琐，偶尔一两次的备份工作没必要用它来进行，它最适合那些需要定期制作备份的场合：必须先通过 New Project（新项目）选项卡创建一个新的备份项目，在 Project Name（项目名称）字段里给新项目起一个名字。这个项目包含着进行备份所需要的所有参数设置：对哪些个数据库进行备份、使用哪些选项进行备份、什么时候进行备份（这个选项是可选的）等。

1. 设置备份的范围

备份工作可能涉及一个或多个数据库。在默认的情况下，选中的数据库里的所有数据表将全部被备份下来，但可以在 Backup Content（备份内容）列表里对数据表进行取舍（如图 5-6 所示）。



图 5-6 在 MySQL Administrator 里定义一个备份项目

2. 备份选项

在 Advanced Options（高级选项）选项卡里，可以设置备份工作的细节。下面是一些比较重要的参数或选项。

- Lock All Tables (锁定所有的数据表)**。激活这个选项的效果是在备份工作开始之前，用 *LOCK* 命令给所有的数据表都加上锁，以免它们所包含的数据在备份过程中被其他用户改变。这种做法的好处是可以让用户获得一份稳定的 MyISAM 数据表备份，但代价是整个数据库在备份过程中无法使用，而备份工作还必须等到所有的数据表都收到 *LOCK* 命令之后才能真正开始。

(如果有其他用户正在使用打算备份的数据库，所有的 *LOCK* 操作都必须暂时取消。)

- Single Transaction (单个事务)**。激活这个选项的效果是整个备份工作将被视为一个事务。这种做法的好处是可以让用户获得一份稳定的 InnoDB 数据表备份，但代价是这么大的一个事务会给数据库在备份期间的查询响应速度带来负面影响。
- Normal Backup (常规备份)**。激活这个选项的效果是每个数据表将依次分别备份。这是最有效率的办法，但缺点是数据库有可能会在备份过程中被其他用户修改，而这又可能导致数据表里的数据彼此矛盾。
- Complete Backup (完全备份)**。激活这个选项的效果是：把被选中的数据库里的所有数据表全都备份下来。此时，在 Backup Project (备份项目) 选项卡里对数据表做出的取舍无效。
- No CREATEs**。激活这个选项的效果是备份结果里不包含 *CREATE TABLE* 命令。
- No EXTENDED INSERTs**。激活这个选项的效果是为数据表里的每条记录分别生成一条 *INSERT* 语句。(出于节约空间和时间的考虑，默认的做法是为尽可能多的记录生成一条 *INSERT* 语句。)
- Add DROP TABLE (添加 DROP TABLE 命令)**。激活这个选项的效果是：在每条 *CREATE TABLE* 语句的前面加上一条 *DROP TABLE* 命令。这样一来，等以后用这个备份去恢复数据库的时候，数据库里的已有数据表将先被删除、再被重新创建出来。
- Complete INSERTs (完整的 INSERT 语句)**。激活这个选项的效果是：备份结果中的 *INSERT* 命令将包含着各个数据列的名字(如 *INSERT INTO tablename (colname1, colname2) VALUES (...)*)。
- ANSI QUOTES**。激活这个选项的效果是：数据表和数据列的名字将用与 ANSI 标准兼容的双引号 ("name") 而不是 MySQL 惯用的单引号 ('name') 括起来。
- Disable Keys**。激活这个选项的效果是：等以后使用这个备份恢复数据库时，在恢复过程中暂时停止更新各种索引。在备份/恢复一个大数据表的时候，这可以节约不少时间。

3. 进行备份

把有关选项全部设置好以后，先单击 Save Project (保存项目) 按钮把项目设置保存起来，再单击 Execute Backup Now (进行备份) 按钮就可以开始备份了。系统将提示用户为备份结果文件输入一个文件名。这是一个文本文件，其内容是一些 UTF8 编码的 SQL 命令。由 MySQL Administrator 生成的备份文件与 mysqldump 命令生成的备份文件很相似，但 MySQL Administrator 并没有调用 mysqldump 命令，它使用的是属于它自己的备份代码。

4. 备份工作自动化

可以通过 MySQL Administrator 提供的 Schedule (计划任务) 选项卡让备份工作定期执行。备份结果文件将被写入一个选定的子目录。根据在这个选项卡里的设置，操作系统会在预定时刻使用一个特殊的选项自动启动 MySQL Administrator 去执行备份任务。

5.3.9 Restore 模块 (用备份恢复数据库)

这个模块的用途是用以前制作的备份来恢复数据库。在 Restore Content (恢复内容) 选项卡里，在选好备份文件之后，可以对应该恢复哪些数据表进行取舍(默认做法是全部恢复)。

在绝大多数场合，备份的数据表来自哪一个数据库，就应该把它们恢复到哪一个数据库。但在某些场合，可能需要把数据表恢复到另外一个数据库里去。如果确实想这么做，在 MySQL Administrator 程序的 Linux 版本里，需要在 Restore Selected Tables In (把数据表恢复到……) 字段里输入那个数据库的名字；在 Windows 版本里，这个字段的名字是 Target Schema (目标数据库)。

注意 应该只用这个模块去恢复用 MySQL Administrator 程序制作的备份文件，不要用它来恢复用其他工具（phpMyAdmin、mysqldump 等）制作的备份文件。如果备份文件不是用 MySQL Administrator 程序制作的，Restore Backup（恢复备份）模块将无法正常工作。其他备份工具在这方面的要求不那么严格。

5.3.10 Replication Status 模块（查看镜像机制的工作状态）

如果用 MySQL Administrator 连接的 MySQL 服务器是某个镜像系统的组成部分，就可以通过这个模块查看到该镜像系统的工作状态。与镜像系统有关的背景知识可以在第 14 章找到。可以通过 MySQL Administrator 程序的 Startup Parameters（启动参数）模块对镜像系统的有关参数进行配置。

5.3.11 Catalogs 模块（对数据库和数据表进行管理）

虽然名字叫做 Catalogs（目录），但这个模块的真正用途是对数据库和数据表进行管理（如图 5-7 所示）。在这个模块里，可以快速查看现有数据库、数据表、索引等的定义和使用情况。

这个模块其实是为了创建新数据表和修改现有数据表而设计的。不过，在笔者测试过的版本里，这些功能既难以使用，又不够稳定。在情况改变之前，还是使用 *CREATE TABLE* 和 *ALTER TABLE* 命令或是使用 phpMyAdmin 来设计数据库更有效率。

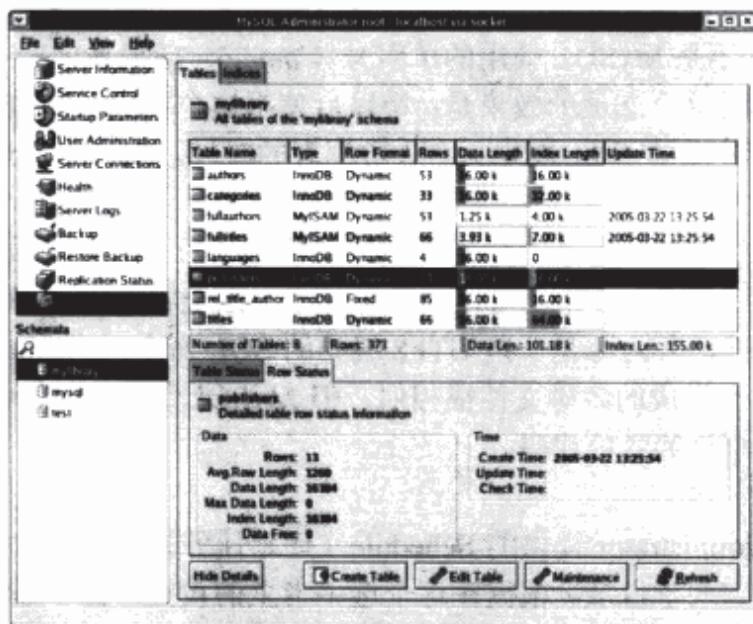


图 5-7 用 MySQL Administrator 查看 mylibrary 数据库的数据表属性

5.4 MySQL Query Browser

MySQL Query Browser 的主要任务是帮助人们构造和调试 SQL 命令。除此之外，这个程序还提供了一些辅助性的功能：可以把数据插入数据表、对数据表里的数据进行修改、对规则表达式进行调试（SQL 操作符 REGEX 会用到规则表达式）、阅读 MySQL 帮助文档等。

5.4.1 SQL 命令的输入和执行

学习使用 MySQL Query Browser 的最佳办法是启动这个程序并输入一条简单的 SQL 命令。SQL

第 5 章 MySQL Administrator 和 MySQL Query Browser 75

关键字将自动显示为彩色文字。单击 Execute (执行) 按钮就可以执行输入的命令。命令的执行结果将显示为一个数据表 (如图 5-8 所示)。

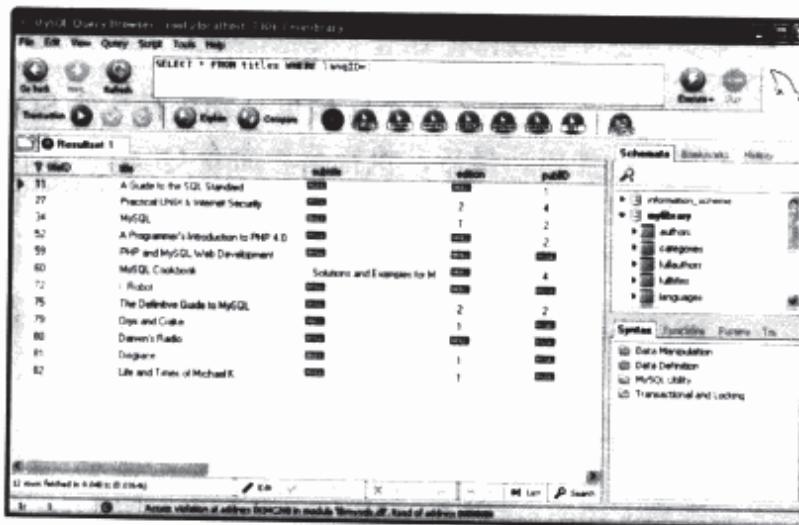


图 5-8 与 MySQL Query Browser 的第一次接触

提示 有一些键盘快捷键可以让 MySQL Query Browser 用起来更方便、更有效率。

- Ctrl+回车键：执行当前 SQL 命令（与 Execute (执行) 按钮作用相同）。
- Ctrl+Shift+回车键：执行命令，但结果将显示在一个新的窗口里。这么做的好处是还可以看到前一条或前几条命令的执行结果。
- F11 键：改变窗口布局并加大 SQL 命令输入区。这在输入一个很长的 SQL 命令时非常有用。如果再次按下 F11 键，窗口将恢复原样。
- F12 键：关闭窗口其他部分，只保留结果显示区。这在查看大量结果的时候非常有用。如果再次按下 F12 键，窗口将恢复原样。

□ **用鼠标单击方式执行 SQL 命令。** MySQL Query Browser 提供的各种箭头按钮可以让用户只须打最少的字就能“组装”出 SQL 命令来。请看下面的例子。

- 单击 SELECT 按钮。
- 单击数据表 *titles* 中的 *title*、*subtitle* 和 *edition* 数据列。
- 单击 WHERE 按钮。
- 单击数据表 *titles* 中的 *langID* 数据列。
- 用键盘输入 WHERE 条件 “=2”。
- 单击 ORDER 按钮。
- 单击数据表 *titles* 中的 *title* 数据列。

根据以上鼠标动作，MySQL Query Browser 将构造出一条如下所示的命令，只须再单击 Execute 按钮就可以执行它了：

```
SELECT t.title, t.subtitle, t.edition, t.catID FROM mylibrary.titles t
WHERE t.langID=2 ORDER BY t.title
```

上面这条命令中的 *t* 是数据表 *mylibrary.titles* 的一个假名。这个假名也可以用... *FROM mylibrary.titles AS t ...* 子命令来定义。MySQL Query Browser 不使用可选的 *AS*，这让不少 SQL 老手感到不习惯。熟悉 SQL 语言的 MySQL 用户往往更喜欢自己动手从键盘输入命令。

- **导出 SELECT 结果。**可以用菜单命令 File (文件) | Export Result (导出结果) 把刚执行完的 SELECT 命令的结果导出为 CVS、HTML、XML 或 Excel 格式的文件。
- **对 SELECT 结果进行比较。**MySQL Query Browser 的特长之一是它可以对两条 SELECT 命令的结果进行比较。具体做法是：先用快捷菜单命令 Split Tab Vertically (纵向拆分窗口) 把结果数据表分割为上、下两个区域。然后在两个区域里分别执行一条 SQL 命令，这两条 SQL 命令必须是对同一些数据列进行操作。最后，执行菜单命令 Query (查询) | Compare Resultsets (比较结果集)。根据比较结果，数据记录将被显示为白色（同时出现在两个结果集里）、绿色（只出现在第一个结果集里）或红色（只出现在第二个结果集里），如图 5-9 所示。

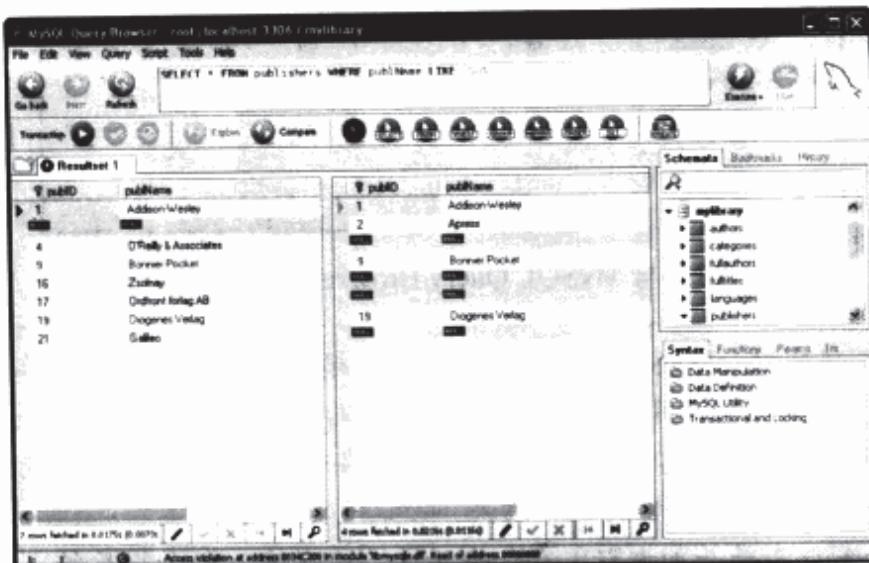


图 5-9 对比两条 SELECT 命令的结果

- **BLOB 数据列。**BLOB 数据列的内容无法直接显示。但可以在另外一个编辑窗口里查看它们或者是把它们存放到一个文件。
- **分析 SQL 命令的执行过程。**单击 Explain (解释) 按钮就可以看到刚执行完的 SQL 命令的执行细节信息。MySQL 高手可以从中发现刚才的命令在执行时都用到了哪些索引，但普通用户要想看懂它们就没那么容易了（参见第 8 章）。
- **事务。**除了刚才提到的 SELECT 命令，还可以在 MySQL Query Browser 的帮助下快速执行 UPDATE、INSERT、DELETE 等许多命令，只是这些命令的结果数据表将会是一片空白而已。如果正在使用 InnoDB 数据表，还可以通过按钮 Start (开始)、Commit (执行事务) 或 Rollback (撤销事务) 把这些命令当做事务来执行。
- **同时打开多个结果显示区。**可以通过菜单命令 File (文件) | New Query Tab (新查询窗口) 打开一个 SQL 命令输入区和结果显示区。这在需要并行调试多条 SQL 命令时会很有用。

5.4.2 对 SELECT 结果里的数据进行修改

如果某个简单的 SELECT 查询只针对一个数据表（没有使用 JOIN 子命令）且没有使用 GROUP BY 子命令或任何统计函数，它的查询结果将允许修改。会在结果数据表的底部看到一个 Edit (编辑) 按钮，单击它就可以开始修改了；这个按钮只在查询结果允许修改时才会出现。

如果想修改某数据表字段，必须先双击以选中它。添加的新记录将简单地追加到数据表的末尾。如果想删除某个记录（即数据行），用鼠标右键单击 Delete Row (删除数据行) 命令即可。最后，必

须单击 **Apply Changes** (应用) 才能让刚才做出的修改生效。

5.4.3 SQL 命令的历史记录和书签

MySQL Query Browser 会把所有执行过的命令自动保存到一个名为 History(历史记录) 的清单里, 可以通过主窗口右边的滚动窗口查看这个清单 (如果那个滚动窗口足够宽)。可以用鼠标把命令从历史记录里拖放到 SQL 命令输入区, 还可以用一个双击动作执行它。

可以用 **Ctrl+B** 组合键给自己常用的命令起一个名字并把它保存到 Bookmark (书签) 清单中。这个清单的位置也在主窗口右边的滚动窗口那里。

5.4.4 一次执行多条命令 (脚本)

用菜单命令 **File** (文件) | **New Script Tab** (新脚本) 打开一个 SQL 脚本输入区。可以在那里输入多条 SQL 命令——别忘了用分号 (;) 隔开它们。接下来, 既可以一次执行全部这些命令 (按下 **Execute** (执行) 按钮), 也可以一条一条地依次执行它们 (通过 **Script** (脚本) 菜单)。

这些脚本多由几种常用的 SQL 命令构成, 它们与存储过程 (stored procedure; 参见下一小节) 并不是一回事。可以通过 **File** (文件) 菜单把这些命令保存为*.sql 文件, 但历史记录和书签功能对它们是不可用的。

5.4.5 存储过程

MySQL Query Browser 还提供了一些可以帮助用户输入存储过程 (stored procedure) 的功能, 将在第 13 章对这些功能做进一步讨论。(存储过程是由多条 SQL 命令构成的、由 MySQL 服务器直接控制的一些函数。简单地说, 存储过程是 MySQL 服务器的特色功能之一, SQL 脚本是 MySQL Query Browser 的特色功能之一。)

5.4.6 MySQL Help (帮助文档)

MySQL Query Browser 还可以用来阅读 MySQL 帮助文档 (如图 5-10 所示)。这份帮助文档可以通过主窗口右边的 Syntax (语法) 和 Function (函数) 滚动窗口进行检索, 其内容来自 <http://dev.mysql.com/doc/> 处的 MySQL 在线文档。

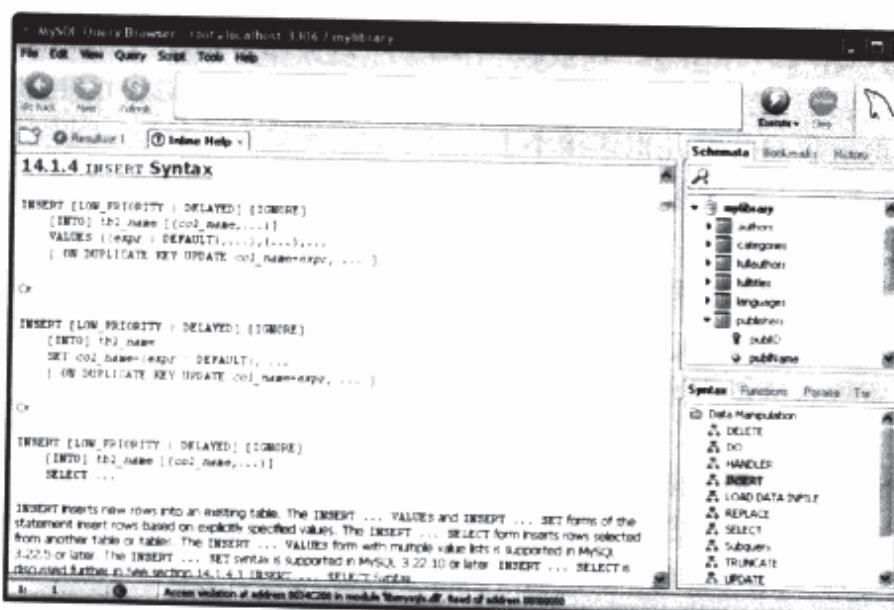


图 5-10 用 MySQL Query Browser 阅读 MySQL 帮助文档

第6章

phpMyAdmin



phpMyAdmin 程序可能是最受欢迎的 MySQL 系统管理工具。它可以用来创建、修改、删除数据库和数据表；可以用来创建、修改或删除数据记录；可以用来导入和导出整个数据库；还可以用来完成许多其他的系统管理任务。

虽然 phpMyAdmin 有众多功能，但这些功能当中到底有哪些是用户能用的要取决于 phpMyAdmin 在 MySQL 服务器上有哪些访问权限。在用户自己的本地测试系统上，可以让 phpMyAdmin 以 *root* 权限运行，这样用户就可以顺理成章地可以访问任何一个数据库和执行任何一种系统管理任务。可是，当使用 phpMyAdmin 去访问一个属于别人（如 ISP）的 MySQL 服务器时，访问权限往往仅限于用户自己的数据库。

phpMyAdmin 由一系列 PHP 脚本构成。使用 phpMyAdmin 的前提条件是系统里还安装了 Web 服务器（通常是 Apache）和 PHP。与其他 PHP 程序一样，以下事实对 phpMyAdmin 也不例外：脚本在 Web 服务器上执行，但由脚本生成的 HTML 页面却可以从任何地方访问。这对 phpMyAdmin 来说其实是件大好事，这一事实使用户得以在本地计算机上使用 phpMyAdmin 去管理运行在另外一台计算机（比如 ISP 的计算机）上的 MySQL 服务器。这是因为，从 ISP 的立场看，由一系列脚本构成的 phpMyAdmin 是一个本地程序，因此它有权对 MySQL 服务器进行任何访问。

与此形成鲜明对照的是在第 5 章介绍的 MySQL Administrator 和 MySQL Query Browser 程序，只有在本人已经有权通过网络去访问一个或多个远程数据库（当然，那应该是些 MySQL 数据库）的情况下，才能使用它们去管理一个非本地的 MySQL 服务器。可是，出于信息安全方面的考虑，绝大多数 ISP 会把他们的 MySQL 服务器配置为只允许本地用户去访问数据库（请注意，这里所说的“本地”指的是 ISP 眼里的本地，不是用户眼里的本地，也不是通过网络）；这也正是为什么许多软件在自己的计算机里很好用，可上了网却哪儿也连接不上。

在这一章里，将先讲解怎样安装和配置 phpMyAdmin 2.6.1，然后再对这个工具最重要的几项功能进行介绍。请注意，因为这个版本与 MySQL 5.0 不太兼容，所以还不能通过它去使用或管理 MySQL 5.0 的某些新增功能，如视图、存储过程、触发器等。此外，phpMyAdmin 也不支持始见于 MySQL 4.1 版本的二维地理数据管理功能（MySQL 公司称为“GIS 功能”）。不过，随着 MySQL 5.0.n 的日益流行，phpMyAdmin 程序的新版本应该离用户不远了。

提示 phpMyAdmin 使 MySQL 服务器上的许多管理工作大为简化，但用好这个工具的前提是必须对 MySQL 服务器的维护和管理、数据库设计、信息安全问题、SQL 语法等概念足够熟悉。对这些问题的讨论可以参见本书的第 8 章~第 14 章。

6.1 phpMyAdmin 的安装与配置

本节内容的讨论前提是已经把 Apache 和 PHP 提前安装在了将要安装 phpMyAdmin 程序的计算机上，而且 PHP 已经可以与 MySQL 进行通信（参见第 2 章）。

6.1.1 安装 phpMyAdmin 文件

可以在 <http://www.phpmyadmin.net/> 网站找到*.zip 或*.tar.bz2 文档形式的 phpMyAdmin 软件包。这个文档里的文件必须全部复制到 Apache 的 DocumentRoot 目录下的某个子目录里。根据所使用的操作系统，这个子目录应该是 Programs\Apache Group\Apache 2\htdocs、/var/www/html 或 /src/www/htdocs。解压缩后的 phpMyAdmin 要占用大约 8MB 的空间。

- **本地 Windows 计算机。** 在 Windows 系统上，用 WinZip 等 ZIP 文件工具对下载回来的文档进行解压缩；Windows XP 用户也可以使用 Windows Explorer（资源管理器）内建的文件解压缩功能。把文件解压缩操作生成的 phpMyAdmin-version 子目录重新命名为一个简单点的名字（在以后的示例里，将使用 phpmyadmin 作为这个子目录的名字）。
- **本地 Linux 计算机。** 在 Linux 系统上，先切换到 DocumentRoot 目录，然后用 tar 命令对下载回来的文档进行解压缩。

```
root# cd <documentroot directory>
root# tar xjf phpMyAdmin-version.tar.bz2
root# mv phpMyAdmin-version phpmyadmin
```

有一些 Linux 发行版本已经收录了 phpMyAdmin，只须安装相应的软件包即可。请注意，发行版本自带的 phpMyAdmin 版本不一定是最新的。

- **ISP 的 Web 服务器。** 在一台非本地的 Web 服务器上安装 phpMyAdmin 会比较麻烦（假设那台服务器主机还没有安装 phpMyAdmin）。这里又分为两种情况：
 - **允许 shell 登录的 ISP。** 先从 [phpmyadmin.net](http://www.phpmyadmin.net/) 网站把 phpMyAdmin-version.tar.bz2 文件下载到本地计算机，再用一个 FTP 客户程序把这个文件上传到 ISP 那里。接下来，用 ssh 命令登录到 ISP 主机并执行上面 3 条 Linux 命令。
 - **不允许 shell 登录的 ISP。** 如果可以通过 ssh 命令去访问在这类 ISP 的账户，必须先在自己的本地计算机上把 phpMyAdmin 软件包解压缩到一个任意的子目录里，然后再用一个 FTP 客户程序把子目录整个上传到 ISP。因为有大约 8MB 的 500 多个文件需要上传，所以会花费不少的时间。别忘了把 FTP 客户程序设置为使用二进制模式（不是文本模式）进行上传。

6.1.2 配置 phpMyAdmin

在可以使用 phpMyAdmin 之前，还需要在 phpMyAdmin 的安装目录里对 config.inc.php 文件中的几个选项做一些设置。根据 MySQL 服务器所使用的身份验证模式，需要修改的选项也会有所不同。将在接下来的两个小节里详细讨论这些不同。

- **config 身份验证模式（密码以明文形式保存在 config.inc.php 文件里）。** 这是最简单的情况。只需要把 MySQL 用户名和密码直接输入到 config.inc.php 文件即可。如果只是在一个本地测试系统上试用 phpMyAdmin 或者如果在 ISP 那里有一个专用账户，建议选用这种模式。
- **http 和 cookie 身份验证模式。** 在这两种模式下，用户必须先在一个登录窗口里通过注册才能使用 phpMyAdmin 程序。这种做法有两个明显的好处：首先，因为 MySQL 密码没有出现在

config.inc.php 文件里，所以身份验证过程更加安全；其次，允许为不同的用户设置不同的密码并让他们在通过注册后只能对自己的数据库进行管理。这两种身份验证模式尤其适合那些需要管理许多个账户的 ISP。

6.1.3 config 身份验证模式

注意，以下示例中的 config.inc.php 文件片段只给出了必须修改的部分（改动量最小）。根据实际情况，也许还需要对其他一些选项做出修改。如果没有特别注明，只有黑体字部分是需要修改的。

- **phpMyAdmin 的地址。**首先，把变量 \$cfg['PmaAbsoluteUri'] 设置为将被用来运行 phpMyAdmin 程序的主机的网络地址。可以使用一个计算机名来设置这个变量，但前提是那台计算机必须在网络中真实存在（笔者在自己的测试网络里使用的计算机名是 *uranus.sol*）。只有当 phpMyAdmin 程序与 MySQL 服务器是安装在同一台计算机里、因而不需要通过网络去访问 MySQL 服务器的时候，才允许把这个变量设置为 *localhost*。在参照以下代码去修改自己的 config.inc.php 文件的时候，千万不要忘记把主机名 *uranus.sol* 和子目录 *phpmyadmin* 替换为自己的主机名和 phpMyAdmin 安装路径（注意字母的大小写和最末尾的“/”字符）。

```
# changes in config.inc.php
...
# must contain the computer name and the phpMyAdmin installation directory:
$cfg['PmaAbsoluteUri'] = 'http://uranus.sol/phpmyadmin/';
...
```

也可以把变量 \$cfg['PmaAbsoluteUri'] 设置为空；在大多数场合，phpMyAdmin 有能力自行确定正确的设置。不过，这会使 phpMyAdmin 不停地显示一条警告信息，如果想避免这个问题，就需要在 config.inc.php 文件里把变量 \$cfg['PmaAbsoluteUri_DisableWarning'] 设置为 TRUE。

- **MySQL 用户名和密码。**接下来，需要告诉 phpMyAdmin 怎样去连接 MySQL 服务器。一般来说，只有密码是必须修改的，其他参数可以保持原样。注意，如果还没有为 MySQL 用户名设置密码，在参照以下代码修改自己的 config.inc.php 文件时就必须把密码字符串设置为空。如果修改 config.inc.php 文件的目的是为了让自己能够使用某个 ISP 主机上的 phpMyAdmin 程序，请把以下代码里的用户名 *root* 替换为 ISP 分配的 MySQL 用户名。如果 ISP 的 Web 服务器和 MySQL 服务器没有安装在同一台计算机上，也许还需要把以下代码里的主机名 *localhost* 替换为 MySQL 服务器的真实地址或主机名。

```
$cfg['Servers'][$i]['host']      = 'localhost'; // MySQL computer name
...
$cfg['Servers'][$i]['auth_type'] = 'config';    // must be 'config'
$cfg['Servers'][$i]['user']     = 'root';        // MySQL user
$cfg['Servers'][$i]['password'] = 'xxx';         // MySQL password
...
```

- **几个可选的通信参数。**如果打算访问的 MySQL 服务器运行在 UNIX/Linux 环境下并且被配置成只能通过一个套接字文件（不使用 TCP/IP 协议）进行访问，就必须把 config.inc.php 文件里的 *connect_type* 变量设置为 *socket*。如果担心 phpMyAdmin 找不到套接字文件，就把它文件名也在 config.inc.php 文件里写清楚好了，如下所示：

```
$cfg['Servers'][$i]['connect_type'] = 'socket'; // access over socket file
$cfg['Servers'][$i]['socket']      = '';        // socket file name
```

请注意，只要没有在 config.inc.php 文件里把 *host* 变量设置为 *localhost*，phpMyAdmin 就将自动使用 TCP/IP 协议去访问 MySQL 服务器。此时，作为一个可选设置项，可以在

config.inc.php 文件里指定一个 TCP/IP 连接端口供 phpMyAdmin 与 MySQL 服务器建立连接时使用。

```
$cfg['Servers'][$i]['port']      = '3306'; // port number
$cgi['Servers'][$i]['connect_type'] = 'tcp'; // access over TCP/IP
```

注解 在 RHEL 4 环境下，必须使用 TCP/IP。这是因为 RHEL 4 中的 SELinux 默认配置不允许 Apache 访问 MySQL 套接字文件。

□ **mysql 或 mysqli 接口。**从版本 5 开始，PHP 可以使用 *mysql* 或 *mysqli* 两种接口（模块）与 MySQL 通信。第 15 章将对这两种接口进行详细介绍，而现在只要记住：如果 MySQL 服务器的版本是 4.1 或更高，就应该使用 *mysqli* 扩展模块，这是因为它的速度更快，安全性也更好：

```
$cfg['Servers'][$i]['extension'] = 'mysqli'; // mysqli 接口
```

1. 试用phpMyAdmin

现在，可以试试 phpMyAdmin 能不能正常启动了：在 Web 浏览器里输入地址 `http://localhost/phpadmin/index.php` 或 `http://computername/index.php`。在 phpMyAdmin 的启动页面上，可以对窗口文字的语言（比如英语）和 phpMyAdmin 的窗口布局做出设置。

phpMyAdmin 的启动页面应该是如图 6-1 所示，根据 phpMyAdmin 的具体安装版本和它在 MySQL 服务器上的访问权限，用户看到的启动页面可能会与图 6-1 有所不同。如果一切顺利，不妨趁这个机会为这个启动页面创建一个书签，以后就用不着再输入这个地址了。

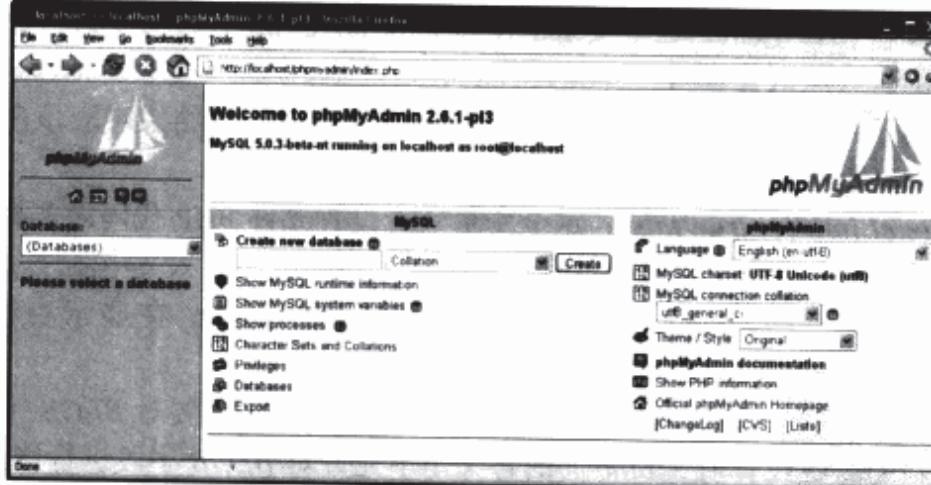


图 6-1 phpMyAdmin 的启动画面

如果在 phpMyAdmin 的启动页面上看到一条出错消息说使用了一个多字节字符集但 PHP 扩展模块 *mbstring* 不可用，则还需要再去安装那个模块。在绝大多数 Linux 发行版本里，只需安装相应的软件包即可解决这一问题。在 Windows 环境下，这个扩展模块其实早已安装好了，只是还没有被激活而已；只需在 `php.ini` 文件里找到 `extension=php_mbstring.dll` 语句并把它前面的分号去掉即可。无论是在 UNIX/Linux 环境下还是在 Windows 环境下，都必须重新启动 Apache 才能让修改生效。

2. 保护phpMyAdmin目录

如果 phpMyAdmin 安装在 ISP 里，还需要解决一个信息安全隐患：如果有其他的网上冲浪者猜中了用户的 phpMyAdmin 目录名（这个目录名往往是有规律可循的），他就可以像用户本人一样使用 phpMyAdmin 去修改或删除数据库。

可以用一个`.htaccess`文件来防止别人通过Web浏览器去访问`phpMyAdmin`目录。这么做的效果是：如果有人（包括用户本人在内）试图使用一个Web浏览器去查看用户的`phpMyAdmin`目录里的PHP文件，系统就将要求他进行登录并提供正确的密码。`.htaccess`机制的用户名和密码与上机账户和MySQL访问控制机制都没有任何关系，对这一机制的详细介绍请参见第2章中的有关内容。）

另一个必须考虑的信息安全隐患是：MySQL密码在`config.inc.php`文件里是未经加密的明文，而某些ISP允许用户通过FTP来访问Web文件。对于这种情况，`.htaccess`文件所提供的保护将无能为力，必须另想办法来确保`config.inc.php`文件不会被别人以匿名FTP的方式读取。

3. 用`phpMyAdmin`访问多个MySQL服务器

完全可以把用来访问多个MySQL服务器的多组主机名、用户名和密码设置项组合放入同一个`config.inc.php`文件，这将使得这些MySQL服务器的主机名出现在`phpMyAdmin`¹启动页面上的一个列表框里，选择其中之一就可以让`phpMyAdmin`连接与之对应的MySQL服务器。

可以用`config.inc.php`文件里的`$cfg['ServerDefault']`变量指定一个MySQL服务器作为`phpMyAdmin`在启动时将自动与之连接的默认服务器。如果把这个变量设置为0，`phpMyAdmin`在启动时将只显示刚才提到的列表框。

此外，如果不希望在那个列表框里看到枯燥机械的MySQL服务器主机名（比如`localhost`）而是一些有意义的文字的话，还可以用`$cfg['Servers'][$i]['verbose']`变量显示另一个字符串。

4. 其他配置选项

在`config.inc.php`文件里，除了MySQL连接参数，还可以对许多选项进行设置。下面是那些选项当中最为重要的几个和它们的默认设置值。

- **`$cfg['Confirm'] = true/false`**: 这个变量决定着`phpMyAdmin`是否会在执行数据删除操作之前要求用户加以确认（默认设置值：`true`）。
- **`$cfg['MaxRow'] = n`**: 这个变量告诉`phpMyAdmin`在显示查询结果时应该在每个页面上显示多少条数据记录（默认设置值：30）。
- **`$cfg['ExecTimeLimit'] = true/false`**: 这个变量负责设置允许PHP代码最多执行多长时间（默认设置值是300s）。这项设置对那些需要较长时间才能完成的导入/导出命令影响最大。
- **`$cfg['ShowBlob'] = true/false`**: 这个变量告诉`phpMyAdmin`在显示查询结果时是否要把`xxxBLOB`类型的字段也显示出来。（默认设置：`false`）
- **`$cfg['ProtectBinary'] = false/blob/all`**: 这个变量决定着是否要对`BLOB`字段的内容进行写保护。这个选项的默认设置值（'blob'）的含义是不允许对`BLOB`字段进行修改，'all'的含义是不允许对数据表的任何内容（不管它是什么数据类型）进行修改，'false'的含义是允许对数据表的任意内容进行修改。

如果想改变`phpMyAdmin`主窗口中的左窗口宽度（这个宽度的默认值是200像素），用一个文本编辑器打开`themes/themename/layout.inc.php`文件并修改其中的`$cfg['leftWidth']`变量即可。在左窗口里显示数据库列表框（供用户选择数据库）和当前数据库的数据表名单。注意，别忘了把文件路径中的`themename`替换为某个真实的窗口主题名称（如`original`或`darkblue_orange`）。

6.1.4 http 和 cookie 身份验证模式

6.1.3节介绍的`config`身份验证模式（`$cfg['Servers'][$i]['auth_type'] = 'config'`）最容易配置，但缺

1. 原书此处误为“MySQL”。——译者注

点一是缺乏灵活性，无法让多位用户分别管理他们各自的数据库；二是不够安全，用户名和密码都以明文形式出现在 config.inc.php 文件里。如果想让多位用户能够分别管理他们各自的数据库，就应该考虑下面将要介绍的两种身份验证模式。

1. 前提条件

- 本节内容的讨论前提是使用的至少是 phpMyAdmin 2.6.1 和 MySQL 4.1.2 版本。这两个程序的较早版本虽然也提供了 *http* 和 *cookie* 模式的身份验证功能，但必须提前创建一个“多余的”MySQL 用户才能使用。那个额外的 MySQL 用户必须有权对 *mysql.user*、*mysql.db* 和 *mysql.tables_priv* 3 个数据库进行读操作，它的用户名和密码（分别对应于配置变量 *controluser* 和 *controlpassword*）必须在 config.inc.php 文件里定义；具体配置步骤见 phpMyAdmin 帮助文档 Document.html 中的 *Using authentication modes* 小节。
- *http* 身份验证模式要求 PHP 解释器运行为一个 Apache 模块（而不是 CGI 程序）。比较新的 PHP 版本都符合这一要求。
- *http* 身份验证模式还要求 phpMyAdmin 目录不能受到 .htaccess 机制的保护。（否则 *http* 身份验证模式与 .htaccess 机制会发生冲突。）
- *cookie* 身份验证模式是 *http* 身份验证模式的补充，不能使用 *http* 身份验证模式的场合（如 Web 服务器是 Microsoft Internet Information Server 时）都可以使用它。*cookie* 身份验证模式要求用户必须允许来自 phpMyAdmin 的 cookie 进入自己的计算机。

2. http 身份验证模式

如果想让 phpMyAdmin 使用 *http* 身份验证模式，首先需要在 config.inc.php 文件里做出如下所示（黑体字部分）的修改：

```
# in config.inc.php
$cfg['Servers'][$i]['controluser'] = '';           // empty since phpMyAdmin
$cfg['Servers'][$i]['controlpass'] = '';           // version 2.6.1
$cfg['Servers'][$i]['auth_type'] = 'http';          // http authentication
$cfg['Servers'][$i]['user'] = '';                  // empty!
$cfg['Servers'][$i]['password'] = '';              // empty!
```

接下来，还需要在 config.inc.php 文件里对 *host*、*connect_type*、*port*、*extension* 等配置变量进行设置，具体做法请参见 6.1.3 节里的有关内容。完成设置之后，当启动 phpMyAdmin 时，屏幕上将弹出一个如图 6-2 所示的 Web 浏览器对话框，需要在这个对话框里输入 MySQL 用户名和密码；phpMyAdmin 在启动后只能在该 MySQL 用户的权限范围内执行各种操作。

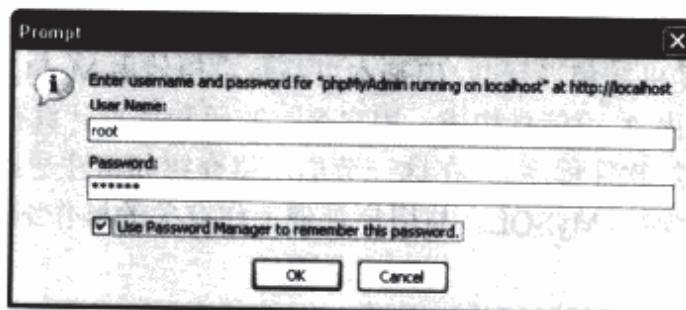


图 6-2 *http* 身份验证模式下的登录对话框

http 身份验证模式比直接把用户名和密码明文写在 config.inc.php 文件里身份验证模式要安全得多。可是，Web 浏览器与 Web 服务器之间通过 *http* 协议传输的登录信息仍然是明文。如果 Web 服务器支持 *https* 协议，就可以获得更大的安全保障——因为登录数据在传输前将会被加密。

注解 如果在登录时遇到问题，phpMyAdmin 2.6.1会向Web浏览器返回一个空白的页面，此时将无法立刻重新进行登录；解决这个问题的办法是先关闭Web浏览器再重新打开它。另外，在注销后也将无法立刻重新进行登录；解决这个问题的办法还是重新启动Web浏览器。

3. cookie身份验证模式

如果想让 phpMyAdmin 使用 *cookie* 身份验证模式，除了必须修改 config.inc.php 文件里的 *auth_type* 语句外，还必须向 *blowfish_secret* 参数提供一个字符串。在把登录数据（用户名和密码）存储为本地计算机上的 cookie 之前，系统将会使用这个字符串对它们进行加密。

```
# in config.inc.php
$cfg['blowfish_secret']      = 'xxxxxx';
...
$cfg['Servers'][$i]['controluser'] = '';
$cfg['Servers'][$i]['controlpass'] = 'xxx'; // version 2.6.1
$cfg['Servers'][$i]['auth_type']   = 'cookie'; // cookie authentication
$cfg['Servers'][$i]['user']       = '';
$cfg['Servers'][$i]['password']   = ''; // empty!
```

登录输入字段将被显示在 phpMyAdmin 的启动页面（如图 6-3 所示）。



图 6-3 *cookie* 身份验证模式下的登录对话框

6.2 用户管理，保护 MySQL

MySQL 的访问权限管理机制相当复杂，但基本思路是希望使用 MySQL 的每一位用户和每一个程序都必须有一个账户。账户由 3 项信息构成：用户名、密码和客户计算机名。（这里所说的“客户”指的是希望与 MySQL 通信的软件程序。）在这一节里，将介绍如何注册新的 phpMyAdmin 用户，如何修改访问权限，以及如何保护 MySQL。与用户管理工作有关的操作细节和背景知识可以参见本书的第 11 章。

注解 一定要让那些拥有 *root* 权限的用户只能从本地计算机（不能通过网络）使用 phpMyAdmin 去管理 MySQL 用户，否则会给系统带来极大的隐患。ISP 在安装 MySQL 服务器的时候必须采取有效措施防止其他用户使用这个账户。（如果不是这样，应该立刻改用另一个 ISP。）由 ISP 分配给用户的 MySQL 账户不应该允许用户改变自己的 MySQL 访问权限。

6.2.1 保护 MySQL

对一个 MySQL 数据库系统而言，它至少需要满足以下几项要求才可以被认为是安全的：

- 不存在没有密码的用户。
- 不存在没有用户名的用户（若是 *User* 列里有空白项，就意味着任何人都可以登录）。
- 不存在其 *Host* 字段的取值是%的用户（%意味着用户可以从网络或因特网里的任何地点登录）。
- 访问权限不受任何限制（各种权限设置字段的取值都是 Y）的用户数量越少越好（最理想的情况是只有 *root* 用户一个）。这种用户在 phpMyAdmin 的用户管理窗口里很显眼：它们的 Global Privileges 字段的取值是 All Privileges。

在 Linux 环境下，MySQL 在安装完成后可以识别 4 种用户名和主机名组合（如表 6-1 所示），这些组合没有一种有密码保护。换句话说，这 4 种组合都违反了上述安防原则。

表 6-1 bind_param 的数据类型

用户名（用户）	主机名（主机）	访问权限
<i>root</i>	<i>localhost</i>	不受任何限制
<i>root</i>	<i>computer name</i> （计算机名）	不受任何限制
空白（用户可以使用任意名字登录）	<i>localhost</i>	受很大限制
空白（用户可以使用任意名字登录）	<i>computer name</i> （计算机名）	受很大限制

从信息安全的角度考虑，用户名是空白的两个设置项必须删除，那两个 *root* 设置项应该加上密码。这些工作都可以通过 phpMyAdmin 的权限管理窗口（如图 6-4 所示）来完成；在 phpMyAdmin 的启动页面上单击 Privileges 选项卡就可以进入这个窗口。

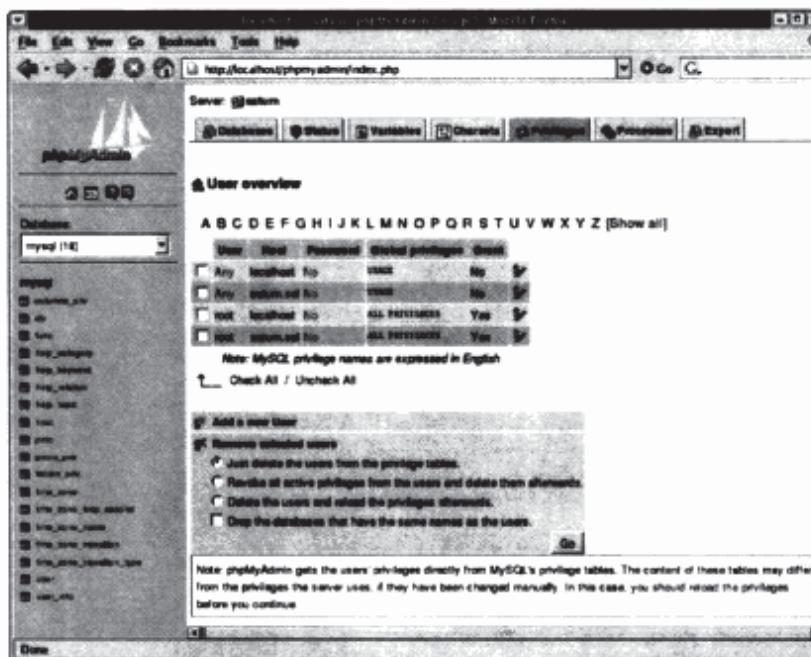


图 6-4 用 phpMyAdmin 来管理用户和访问权限

- **删除不安全的用户设置项。**先在 User overview（用户名单）里把打算删除的用户设置项（在图 6-4 里，这包括两个 Any 用户）全部选中，然后单击 OK 按钮删除它们即可。
- **给用户设置项加上密码保护。**先选中一个用户名，然后单击位于最后一列的 Edit（编辑）符号，

就会打开一个用户权限设置窗口，该用户的所有权限全都汇总在这个窗口里。这个窗口提供了很多功能，而我们在此最感兴趣的是密码设置功能。为避免打字错误造成严重后果，需要输入两遍密码。

注解 在修改了 *root* 用户在 *localhost* 主机上的密码之后，phpMyAdmin会在用户转往其他页面时显示一条出错信息“Access Denied”（“访问被拒绝”）。为了让phpMyAdmin重新开始工作，需要把新设置的密码输入config.inc.php文件(*config*身份验证模式)或重新进行一次登录(*http*和*cookie*身份验证模式)。

6.2.2 创建新用户

从信息安全的角度考虑，应该把 *root* 用户只用于系统管理。如果准备开发一个新的 MySQL 应用程序，应该为开发工作专门创建一个新的 MySQL 用户并只把刚够完成有关任务的权限分配给它；这里的原则是：只要新用户能读取和修改某个特定的数据库就足够了。

□ 创建新用户。为了创建新用户，在 phpMyAdmin 的启动页面里打开 Privileges 选项卡，再在 Privileges 选项卡上单击 Add a new user 链接。在接下来的页面里(如图 6-5 所示)，在 User name (用户名) 下拉列表框里输入新的用户名。在 Host 下拉列表框里，通常应该选择 Local (本地) 选项，phpMyAdmin 会自动地把 localhost 添加到旁边的文本框里。这意味着新用户将只能从本地计算机去访问 MySQL 服务器，这对 PHP 应用程序来说通常已经足够了。最后，必须输入两次密码 (这个密码与 *root* 用户的密码越不一样越好！)

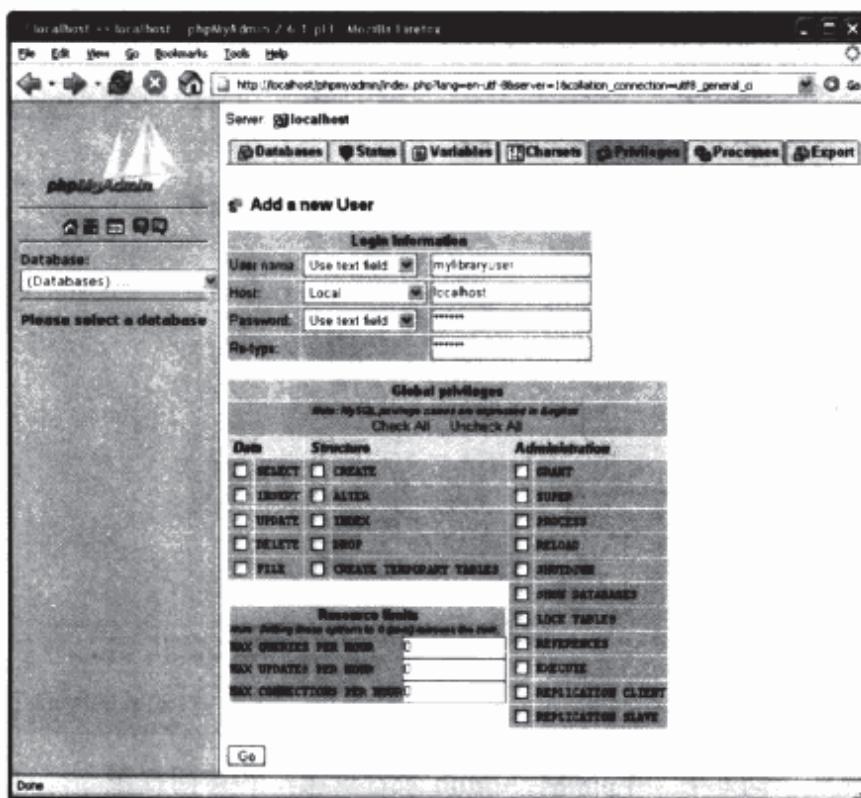


图 6-5 创建一个新用户

不要选中任何一个全局权限设置项！(将在下一个步骤为这个新用户分配足以访问某个特定数据库的权限，所以没有必要在这里向他授予全局权限。)

- **为新用户分配数据库级权限。**单击前一个页面里的OK按钮将转到一个新页面，可以在这里对新用户的访问权限进行细致的设置。在Database-specific privileges（数据库级权限）部分，先在列表框里选中这个新用户需要访问的数据库（如图6-6所示）。这个数据库必须已经存在才会出现在列表框里。如果它此时还不存在，可以在这个文本框输入它的名字。

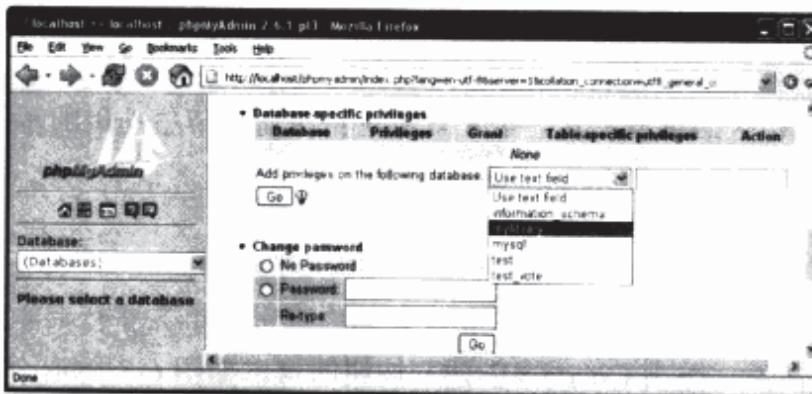


图6-6 为新用户选择一个数据库

在列表框里选中一个数据库之后，将立刻进入另一个页面；通过这个页面对新用户在给定数据库上的各种权限进行设置（如图6-7所示）。下面是关于这些权限的含义的简单说明。

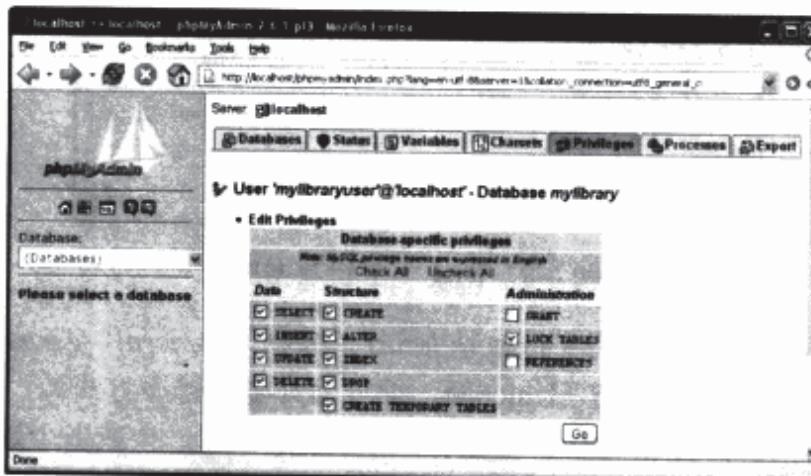


图6-7 设置数据库级操作权限

- **SELECT、INSERT、UPDATE 和 DELETE**权限永远是应该分配给用户的，这样才能让用户（以及相关的PHP页面）读取和编辑数据表。
- **CREATE、ALTER、INDEX、DROP 和 CREATE TEMPORARY TABLES**等权限允许用户改变数据库的结构。要不要把这几项权限授予用户需要根据具体情况来决定。
- **GRANT**权限允许用户把他或她自己的权限转授给其他用户。从信息安全的角度考虑，这项权限通常不应该授予给用户。
- **LOCK TABLES**权限将允许用户使用LOCK命令锁定数据表。这项权限只对极少数的特殊场合才是必不可少的；比如说，用户需要把多条相互关联的SQL当做一一个单元来执行，并且不希望有其他用户在这些命令的执行过程中修改有关的数据。作为LOCK TABLES权限的一种替代品，InnoDB数据表和事务操作往往能把事情做得更好。
- **REFERENCES**权限目前既没有见于文字记载，也没有被实际实现。

数据表级权限的基本用途是配合数据库级权限来实现这样一种效果：先通过数据库级权限禁止用户对数据库进行某种操作，再通过数据表级权限允许用户对数据库里的某个特定的数据表进行该种操作。需要设置或修改数据表级权限的情况极其少见，我们就不在这里对它们进行介绍了。

6.3 创建和编辑数据库

学习本节内容需要对数据库的设计工作以及 MySQL 数据表和 MySQL 数据类型已经有了基本的了解。可以在第 8 章查到相关的背景知识。第 8 章还对将在这一节里使用的示例数据库（*mylibrary* 数据库）做详细的描述。

6.3.1 创建数据库

用 phpMyAdmin 创建新数据库非常简单：先在它的启动页面里为新数据库输入一个名字并选择一种排序方式（sort order），然后单击 Create（创建）按钮即可，如图 6-8 所示。这里所说的“选择一种排序方式”其实就是挑选一种字符集。



图 6-8 创建一个新数据库

对美国和西欧用户而言，最重要的字符集是 *latin1*（细分为 *general*、*german1*、*german2*、*danish*、*spanish* 和 *swedish* 等排序方式）和 *utf8*（*Unicode*，细分为许多种排序方式）。

刚创建的新数据库是空的，不包含任何数据表。下一步是创建一个或多个数据表（如图 6-9 所示）：从数据库列表框里选中那个数据库，进入该数据库的 Structure（结构）页面。（在创建一个新数据库的时候，会被系统自动带到这个页面里来。）

图 6-9 创建新数据表

在图 6-9 所示的页面里，首先需要给新数据表起一个名字，还需要为新数据表定义一些数据列（字段）。如果拿不准应该定义多少个数据列，按照宁多勿少的原则定义；数据表里有几个多余的数据列并不是什么大问题。

现在，phpMyAdmin 会给出一个有着无数输入字段的表单，需要在这个表单里为数据表的每一个数据列（字段）提供以下几项数据。

- Field:** 数据列的名字。
- Type:** 数据类型，如 *INT* 或 *VARCHAR*。
- Length/Values:** 这个字段对绝大多数的数据类型来说是可选的。只有 *VARCHAR* 类型的数据列必须设置一个最大字符长度（最多可以是 255 个字符）。
- Collation:** 设定此数据列使用的字符集和排序方式（仅在不想让此数据列使用在创建这个数据表时选择的默认字符集和排序方式的时候才需要设置这个字段）。
- Attributes:** 这个字段通常留为空白。对于一个整数类型的数据列，如果能肯定它包含的值都是非负数的话，可以在这个字段里填上 *Unsigned*。
- Null:** 这个字段决定着此数据列是否可以包含空值 *NULL*。
- Default:** 可以在这里为此数据列设置一个默认值。等以后添加数据记录的时候，如果没有给这个字段输入另外一个值，系统将把默认值自动填入这个字段。
- Extra:** 这个字段只有一个可取值 *AUTO_INCREMENT*。这项设置一般只对主键（primary key）字段才有意义。
- Index Options:** 这里有 5 种选择：主索引（primary index）、普通索引（regular index）、唯一索引（UNIQUE 索引）、无索引、全文本索引（full-text index）。它们决定着应该为这个数据列创建哪些索引。

在数据列定义块的下面有一个 Table Comments 文本框，供用户输入一些注释；许多人喜欢在这里对数据表的内容做个简单的说明。在它旁边的下拉列表框里，需要选择一种数据表类型（MyISAM 或 InnoDB）。最后，单击 Save（保存）按钮结束数据表的设计过程。

6.3.2 编辑现有的数据表

如果想改变某个数据表的结构，先要在 phpMyAdmin 里选中相应的数据库，然后再选中相应的数据表。对数据表结构的改动可以通过以下几个 phpMyAdmin 设置页面来进行。

- Structure（结构）页面：**可以通过这个页面增加或删除数据列和索引。
- Operation（操作）页面：**可以通过这个页面重新命名数据表、改变数据类型、把这个数据表移动或复制到另一个数据库里去，等等。
- Empty（清空）：**可以通过这个页面快速删除数据表里的全部记录，但数据表本身不会被删除。

为多个数据列创建一个索引

如果只为某一个数据列创建一个新索引，只须在数据表的 Structure（结构）设置页面里单击相应的索引符号即可。可如果想为一个以上的数据列创建一个索引的话，必须先在 Structure（结构）页面的 Indexes（索引）部分写出那些数据列的个数，单击 OK 按钮，然后在接下来的页面里选定数据列和索引类型（如图 6-10 所示）。是否给新索引起名是可选的；如果没有给它起名字，系统会自动挑选一个被索引的数据列的名字作为它的名字。

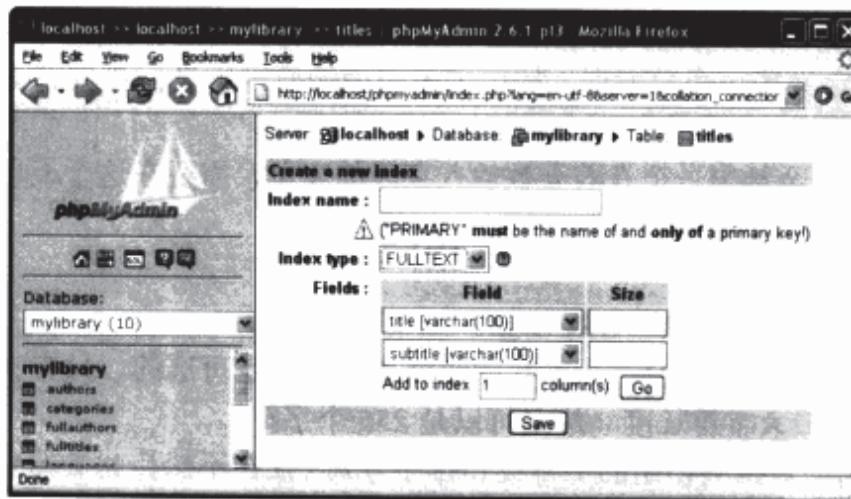


图 6-10 创建涉及两个数据列的全文本索引

6.3.3 设置外键规则

如果你正在使用 InnoDB 数据表，可以用一些一致性规则（术语称为“外键规则”，详见第 8 章）来设置和保持 InnoDB 数据表之间的关系。具体做法是：先选中一个包含着一个或多个外键（foreign key，即指向其他数据表的字段）的数据表，然后单击 Relation View（关系视图）链接。

在接下来的页面里，可以让每一个外键指向另一个数据表里的主键（如图 6-11 所示）。例如，让字段 *titles.catID* 指向字段 *categories.catID*。在定义了这样一条外键规则之后，*titles* 数据表就不会引用一个并不存在的 *categories* 数据表记录项了。

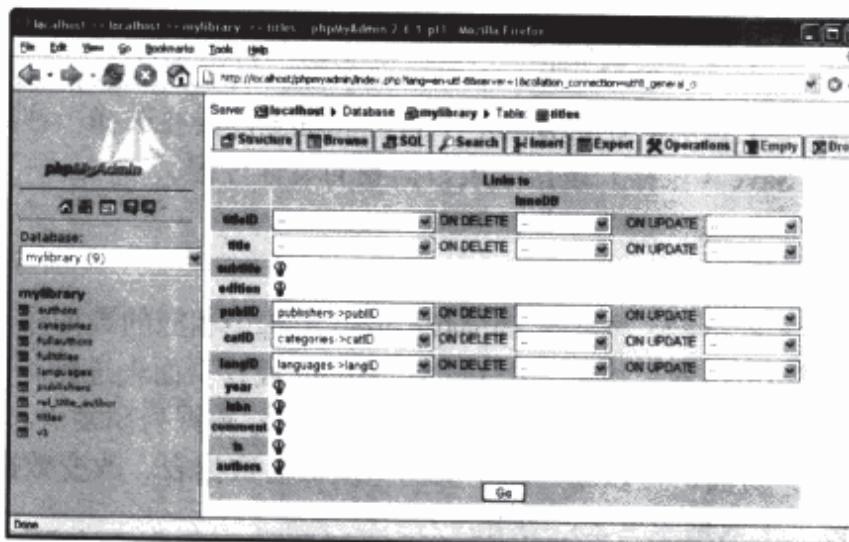


图 6-11 设置外键规则

不仅如此，还可以通过 *ON DELETE* 和 *ON UPDATE* 字段告诉 MySQL 在执行违反了一致性规则的 *DELETE* 和 *UPDATE* 命令时应该怎样做。默认的行为是触发一条出错信息并拒绝执行那条命令。但在许多场合，把受影响的数据记录全部删除(*CASCADE*)或者是把受到影响的外键字段设置为 *NULL* (*SET NULL*) 或许更好——只需根据具体情况选择相应选项即可。

6.3.4 数据库设计方案的汇总和存档

在新数据库的设计工作全部完成之后，肯定希望把它的设计方案完整地记录下来，phpMyAdmin

可以帮助用户实现这个想法：单击位于数据库名字左边的小图标，再单击 Data Dictionary（数据字典）链接，就可以看到一个如图 6-12 所示的页面，phpMyAdmin 会把该数据库里的所有数据表以及它们最重要的属性汇总在那个页面里。

提示 phpMyAdmin 还可以把数据表之间的关系绘制成一张图并把它导出为一个 PDF 文件。不过，要想使用这项功能，必须先创建一些辅助性的数据表并把 phpMyAdmin 配置成能够访问那些数据表。这些事将在本章后面的内容里详细讨论。

Field	Type	Null	Default
authID	int(11)	Yes	0
titleID	int(11)	Yes	0
timestamp	timestamp	Yes	CURRENT_TIMESTAMP

Field	Type	Null	Default
pubID	int(11)	Yes	NULL
title	varchar(100)	Yes	
author	varchar(100)	Yes	
year	tinyint(4)	Yes	NULL
genre	int(11)	Yes	NULL
content	text	Yes	
authors	varchar(255)	Yes	
timestamp	timestamp	Yes	CURRENT_TIMESTAMP

图 6-12 mylibrary 数据库的结构

6.4 查看、插入和编辑数据

对于那些已经存储了一些数据的数据库，可以在 phpMyAdmin 的帮助下轻松地浏览它里面的数据表。具体做法是：先选中想要查看的数据表，再单击 Browse（浏览）页面，phpMyAdmin 就会把这个数据表里的前 30 条记录显示出来。如果单击了某个数据列的标题栏，phpMyAdmin 就会根据这个数据列对显示内容进行排序（如图 6-13 所示）。通过<<、<、>和>>按钮可以快速前进到第一页、前一页、后一页和最后一页。如果想同时看到更多的数据列，可以单击位于数据表左上角的“T”字形符号，让那些文字较长的字段缩短一些。

id	title	author	year	genre	content	timestamp
11	A Guide to the SQL Standard	NULL	1	0	1007 NULL	NULL 2005-02-28 13:34:22
10	MySQL & MySQL	NULL	4	34	NULL NULL NULL	NULL 2005-02-28 13:34:22
9	MySQL	NULL	3	34	NULL 2005-02-28 13:34:22	NULL
8	Web Application Development with PHP 4.0	NULL	0	0	NULL 2005-02-28 13:34:22	NULL
7	Object-Oriented Database Guide	NULL	0	1	NULL 1997 NULL	NULL 2005-02-28 13:34:22
6	2 The Definitive Guide to Excel VBA	NULL	2	3	NULL 2000 NULL	NULL 2005-02-28 13:34:22
5	1 User Authentication, Verification, and Authorization	NULL	0	0	NULL NULL NULL	NULL 2005-02-28 13:34:22
4	3 Object-Oriented Database	NULL	0	1	NULL 1997 NULL	NULL 2005-02-28 13:34:22
3	2 The Definitive Guide to Excel VBA	NULL	0	2	NULL 2000 NULL	NULL 2005-02-28 13:34:22
2	The Definitive Guide to MySQL	NULL	0	34	NULL 2005-02-28 13:34:22	NULL
1	User	NULL	0	0	NULL NULL NULL	NULL 2005-02-28 13:34:22

图 6-13 查看数据表的内容

在这个页面上，既可以一次删除一条数据记录（按钮）或编辑它（Edit 按钮），也可以先选取多条数据记录再把它们当做一组来进行编辑或删除。

Insert（插入）页面可以每次输入两条新记录（如图 6-14 所示）。注意，带有 *AUTO_INCREMENT* 属性（多见于 ID 字段或主键字段）和 *TIMESTAMP* 属性的字段通常用不着由用户来提供数据，MySQL 会为它们自动提供一个适当的值。

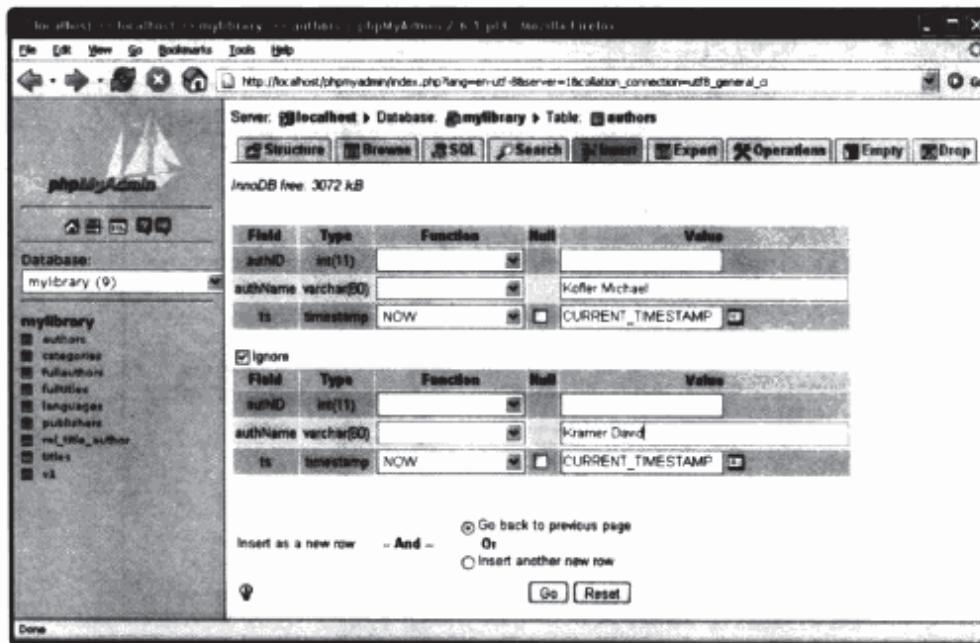


图 6-14 输入一条新记录

在图 6-14 所示的输入表单里，还可以通过 Function（函数）下拉列表框选择一个 SQL 函数。这么做的效果是：MySQL 将先用这个函数对随后输入的字符串进行处理，然后再把函数计算结果存入数据表。例如，如果数据表里有一个数据列是用来存放加密密码的，就应该把 *PASSWORD('abc')* 而不是 '*abc*' 存入这个数据列。

提示 phpMyAdmin 不适合用来输入大量的记录，在实际工作中也很少有这个必要。在正常情况下，把数据存入数据库应该是 PHP 程序的工作。如果希望快速输入一些数据供测试使用，建议采用本章稍后将要介绍的办法：用一个文本编辑器或电子表格程序来录入数据，把它们保存为一个文本文件，最后再导入该文件。

6.5 执行 SQL 命令

在选中数据库或数据表之后，可以在 phpMyAdmin 的 SQL 页面里输入一条 SQL 命令并执行它。如果是一条 *SELECT* 命令，其执行结果将被显示在一个如图 6-13 所示的页面里。

phpMyAdmin 在这里提供了一个非常实用的功能：用户可以再打开一个窗口来输入 SQL 命令（如图 6-15 所示）。打开这个窗口的办法很简单，在 phpMyAdmin 主窗口左半部分的 SQL 按钮（数据库清单上面的那一排按钮中的一个）上单击即可。如果想通过 phpMyAdmin 的数据列编辑区往一个现有的数据表里插入一些新的数据行，这个窗口最方便不过了。

还可以一次输入并执行多条 SQL 命令，但千万记得要用分号把这些命令分隔开。

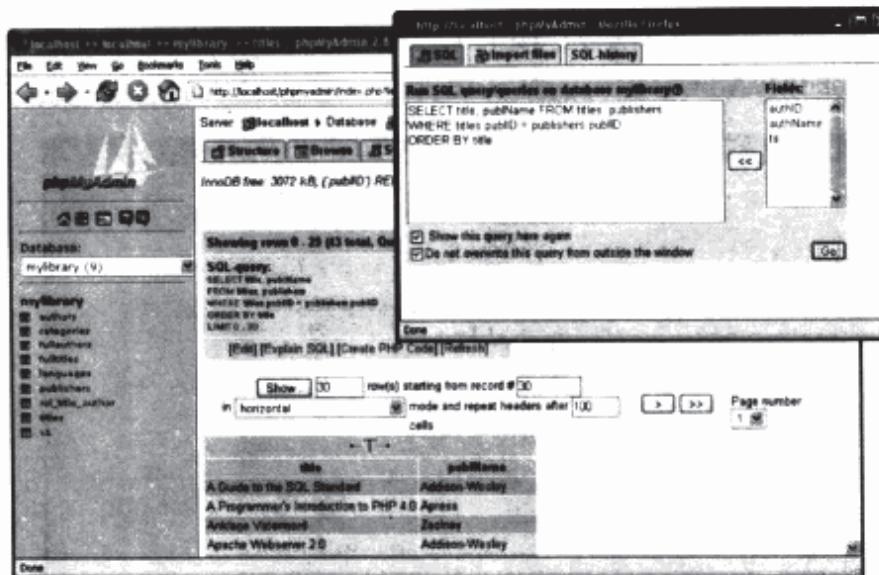


图 6-15 执行 SQL 命令

注解 phpMyAdmin有一些高级功能需要用户提前为它创建一个数据库（见本章后面的内容）才可以使用。如果已经为phpMyAdmin创建好了一个这样的数据库，它就会把曾经执行过的SQL命令都记录到所谓的“历史记录”里去，可以通过它的SQL-History页面方便地查看和使用最近执行过的命令。

6.6 导入和导出

phpMyAdmin 提供了几种方法来导入/导出数据表或整个数据库。本节将对这些功能进行集中介绍。

注解 因为PHP脚本的最长运行时间是受限制的，所以超大数据库和超大数据表的导入/导出操作很容易半途而废。可以去试试在config.inc.php文件里加大这个时间（\$cfg['ExecTimeLimit']=n）的办法，但能不能有效果就很难说了——服务器端的PHP配置往往因为信息安全方面的考虑而忽略这项设置。对于这个问题，最根本的解决办法是修改PHP配置文件php.ini里的max_execution_time变量。除此之外，服务器端的PHP配置往往还会对上传文件的最大长度做出限制（php.ini文件里的upload_max_filesize变量），这也是在导入/导出超大数据库和数据表时必须考虑的一个因素。

如果运行PHP的计算机不在用户的控制范围内（例如，它运行在ISP的计算机上），php.ini文件也就不会在用户的控制范围内。（虽然可以请求ISP修改某些变量，但很可能得不到肯定的回答。）如果没有更好的办法，就应该考虑把导入/导出操作分成几个较小的部分（例如，以数据表为单位而不是以整个数据库为单位）来进行。

另一种选择是用mysqldump命令来导出数据库、用mysql命令来导入数据库。

6.6.1 数据库备份（SQL 文件）

进行数据库备份的目的是为了把某个数据库完整地保存下来，以便在日后有必要时按原样恢复它（甚至是恢复到另一台计算机上去）。首先，选中打算备份的数据库，切换到 Export（导出）页面，在

那里选中 SQL 单选按钮（如图 6-16 所示）。接下来，把要备份的数据表全部选中并修改有关的选项。如果不确定，可以使用 phpMyAdmin 提供的默认设置。

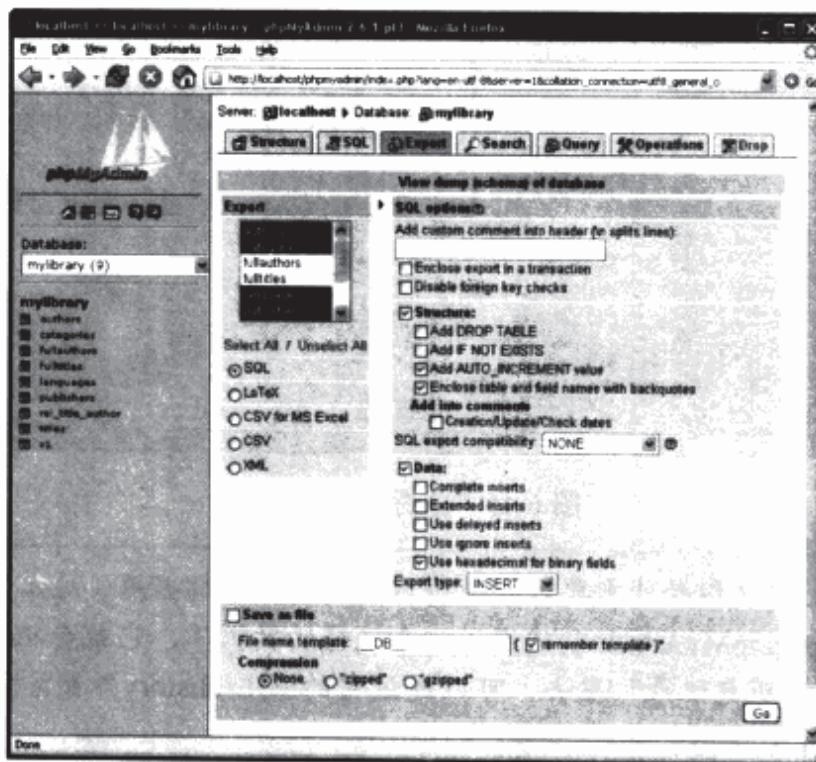


图 6-16 数据库备份

在单击 OK 按钮之后，phpMyAdmin 将创建出一个包含着无数的 *CREATE TABLE* 和 *INSERT* 命令的新页面。现在可以选取这些命令，把它们复制到一个文本编辑器并保存。更简单的办法是直接选择 Save as file（保存为文件）选项。phpMyAdmin 将把数据传输到 Web 浏览器并打开它的 Save（保存）对话框，以便把传输来的数据存入一个本地*.sql 文件。（也可以选择以压缩格式执行上传操作，这将把 SQL 命令保存为一个*.gz 或*.zip 文件。）

下面是对几个最重要的导出选项的说明。

- Disable foreign key check**。如果正在导出带有外键规则的 InnoDB 数据表，应该激活这个选项。这将使 phpMyAdmin 在 SQL 文件的开头加上 *SET FOREIGN_KEY_CHECKS=0* 语句，在末尾加上 *SET FOREIGN_KEY_CHECKS=1* 语句。这将在以后的数据库恢复过程中禁用外键规则检查功能。这不仅可以加快导入操作的速度，还可以避免因为没有按照正确的顺序创建数据表而导致导入操作执行出错。
- Structure options**。这些选项控制着 *CREATE TABLE* 命令的构造情况。比如说，可以在 *CREATE TABLE* 命令的前面插入 *DROP TABLE* 命令以删除可能已经存在的同名数据表。
- Data options**。这些选项控制着 *INSERT* 命令的构造情况。*Complete INSERT* 意味着必须在每条 *INSERT* 命令里把所有数据列的名字写出来，而这通常是多余的。
Extended INSERT 意味着可以用一条 *INSERT* 命令插入多个数据记录。这种做法更有效率，但有可能导致 *INSERT* 语句过长而使它们在以后的导入操作无法处理（相应的出错消息是 *got a packet bigger than 'max_allowed_packet' bytes*）。
也可以在 Export type 列表框里选择 *UPDATE* 或 *REPLACE* 命令而不是 *INSERT*。（这种做法只在打算刷新现有数据时才有意义。）

注意，作为备份结果的 SQL 文件是一个 UTF8 (Unicode) 文件。

6.6.2 导出数据表 (CSV 文本文件)

CSV (*comma-separated value*, 用逗号隔开的数据) 文件其实就是文本文件，只是它们所包含的数据都用逗号或另一种可选的字符隔开了而已。绝大多数电子表格程序都可以毫无问题地导入这种文件。作为一个原则，最好是把每个数据表分别导出为一个 CSV 文件，而不是把同一个数据库里的全体数据表一次性地导出为一个 CSV 文件。

这里需要考虑字符集的问题。在默认设置下，phpMyAdmin 2.6 无法对 CSV 文件的字符集进行设置，只能创建 Unicode 文件 (UTF8)，而这种文件在被导入 Excel 的时候会引起许多问题。还好，解决这个问题的办法很简单，只要在 config.inc.php 文件里修改一个变量即可，如下所示：

```
# in config.inc.php
$cfg['AllowAnywhereRecoding'] = TRUE;
```

从现在起，导出对话框的底部会出现一个列表框，可以在这个列表框里选择需要的文本模式。

6.6.3 导入数据库或数据表 (SQL 文件)

以 SQL 命令形式导出的数据库或数据表（不管它是用上面刚介绍的办法导出的还是用 mysqldump 程序命令导出的）可以用 phpMyAdmin 的 SQL 页面来导入。下面是从一个*.sql 文件导入一个数据库的基本步骤。

- (1) **创建一个数据库。** 在 phpMyAdmin 的启动页面上，创建一个新的、空白的数据库。
- (2) **选择一个 SQL 文件。** 新数据库将被自动选为当前的默认数据库。切换到 SQL 页面（如图 6-17 所示），单击 Browse (浏览) 按钮找到并选中准备导入的那个本地*.sql 文件，该文件包含着 CREATE TABLE 和 INSERT 命令（这些命令必须是以分号隔开的）。

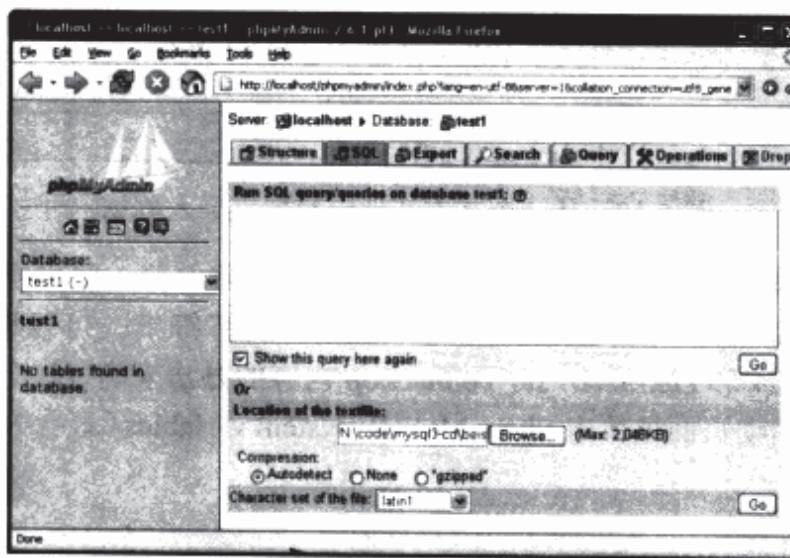


图 6-17 从一个 SQL 文件导入数据库

- (3) **选择字符集。** 在默认的情况下，phpMyAdmin 将假设它正在处理的*.sql 文件使用的是 utf8 字符集。如果事实并非如此，请对字符集设置做出相应的修改。（本书各示例数据库的*.sql 备份文件使用的都是 latin1 字符集。）

- (4) **开始导入。** 单击 OK 按钮，被选中的*.sql 文件就将从本地文件系统被读出，并通过因特网传

输给 phpMyAdmin 去执行。

6.6.4 插入数据表数据（文本文件）

phpMyAdmin 可以从一个文本文件导入某个数据表的内容。这个数据表必须已经存在并且有着足够的数据列和正确的数据类型。为了导入数据，先切换到这个数据表的 Structure 页面，然后单击 Insert data from a textfile to table（从文本文件向数据表插入数据）链接。这个链接可能不太容易找到，它位于 Structure 页面的底部。

在接下来的页面里，选中要导入的文件并给出文件里的数据是如何分隔的、各个字段是如何标识的等。图 6-18 里的设置可以成功地导入一个内容如下所示的典型 CSV 文件：

```
"1";"Linux";"Installation, Konfiguration, Anwendung";"5";"1";"57";
"2";"2000";NULL;NULL
"2";"The Definitive Guide to Excel VBA";NULL;NULL;"2";"3";
NULL;"2000";NULL;NULL
```

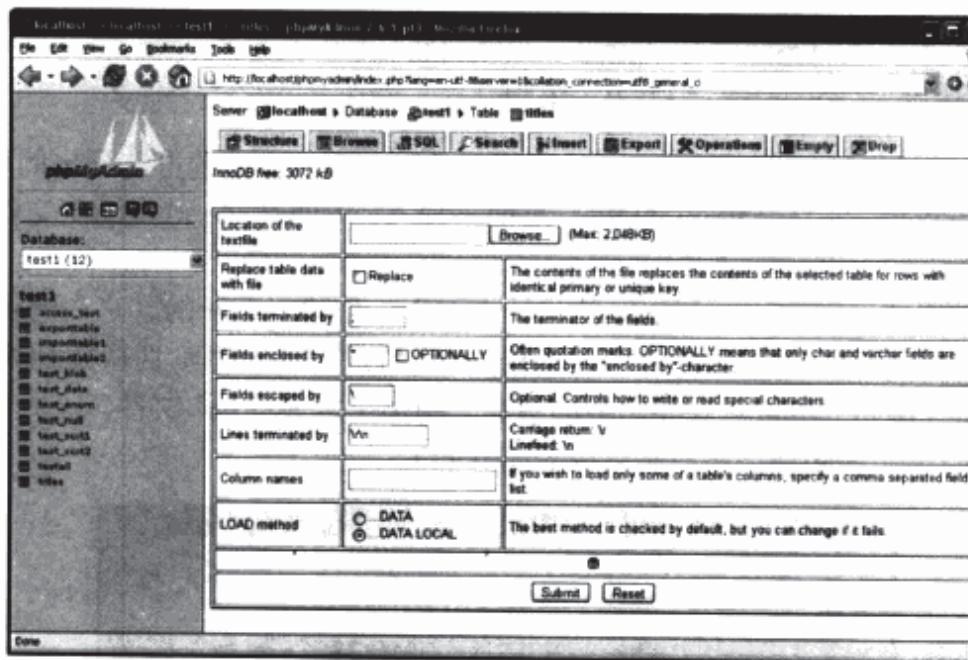


图 6-18 从一个文本文件向数据表导入数据

- **字符集问题。** phpMyAdmin 2.6 不允许为导入操作设置字符集。要想导入成功，文本文件就必须使用与 MySQL 服务器和数据库连接一模一样的字符集。可以通过查看 MySQL 变量 `character_set_server` 和 `character_set_connection` 内容的办法来确定这个字符集，具体做法是在 phpMyAdmin 的启动页面里单击 Show MySQL system variables（查看 MySQL 系统变量）链接。

6.7 服务器管理

Database（数据库）和 Table（数据表）页面里的 Operations（操作）窗格包含着用来重新命名和复制数据库和数据表以及把数据表传输到另一个数据库的命令。

单击 phpMyAdmin 窗口顶部的 Home 图标可以快速返回 phpMyAdmin 的启动页面。如果有足够的 phpMyAdmin 登录权限，就可以在这个页面上看到用来执行各种系统管理任务的链接。

- **Show MySQL runtime information:** 显示 MySQL 服务器的工作状态（相当于 `SHOW STATUS` 命令）。

- **Show MySQL system variables:** 显示 MySQL 系统变量的设置情况(相当于 *SHOW VARIABLES* 命令)。
- **Show processes:** 显示当前进程表(相当于 *SHOW PROCESSLIST* 命令)。如果有足够的权限,还可以终止这些进程。
- **Character sets and collation:** 显示 MySQL 服务器支持的所有字符集和排序方式(相当于 *SHOW COLLATION* 命令)。
- **Privileges:** 显示全体 MySQL 用户名单。可以从这个页面创建新的用户、改变现有用户的权限和删除用户。
- **Databases:** 显示全体数据库清单。在这个页面上单击 *Activate database Statistics* 链接可以了解到各个数据库的大小。密码锁图案的 *Check privileges* 链接也很实用,可以通过它了解到哪些 MySQL 用户有权访问哪些数据库。
- **Export:** 这个链接通往在 6.6 节介绍的导出对话框(如图 6-16 所示)。唯一的区别是现在可以同时选中多个数据库进行备份。

6.8 辅助功能

在这一节将介绍的 phpMyAdmin 功能有一个共同的要求,那就是必须提前创建几个数据表供 phpMyAdmin 存放有关的信息。因此,本节将从创建这些数据表的必要步骤开始展开讨论。然后,将学习到如何以其他办法完成这个工作。(下面将要介绍的 phpMyAdmin 辅助功能和配置指令适合高级 phpMyAdmin 用户使用,普通 phpMyAdmin 用户请不要随便尝试。)

6.8.1 为 phpMyAdmin 创建数据库

在下面的讨论中,假设用户使用的是 phpMyAdmin 程序的 *config* 身份验证模式(*\$cfg['Servers'][\$i]['auth_type'] = 'config'*),即用户名和密码是以明文形式存放在 config.inc.php 文件里的。如果使用的是 *cookie* 或 *http* 身份验证模式,在操作细节方面会与下面给出的有所不同(这些功能在 *cookie* 或 *http* 身份验证模式下的使用方法可以在 phpMyAdmin 文档里查到,但很难阅读):

(1) **创建名为 phpMyAdmin 的数据库。**只要有可能,就应该把供 phpMyAdmin 程序使用的数据表放在一个专用的数据库里,这个数据库的名字设为 *phpMyAdmin*。现在,先使用这个名字创建一个新的、空白的数据库。(如果没有创建新数据库所需要的权限,可以在下一个步骤里把 phpMyAdmin 辅助数据表放在任何一个现有的数据库里。)

(2) **创建必要的数据表。**用文本编辑器打开 phpMyAdmin 安装目录里的 scripts/create_tables_mysql_4_1_2+.sql 文件。如果正在使用的 MySQL 版本低于 4.1.2,必须使用 create_tables.sql 文件。

从文本编辑器里把所有的 *CREATE TABLE* 命令复制到新数据库的 SQL 页面里并执行,它们将创建出以下几个 phpMyAdmin 数据表:*pma_bookmark*、*pma_column_info*、*pma_history*、*pma_pdf_pages*、*pma_relation*、*pma_table_coords* 和 *pma_table_info*。(但不要执行也包含在那个*.sql 文件里的 *DROP DATABASE*、*CREATE DATABASE*、*USE* 和 *GRANT* 命令。)

(3) **修改 config.inc.php 文件。**现在,必须对 config.inc.php 文件做出一些修改。这里的关键是如下所示的那两行,它们所设置的用户名和密码将被用来访问包含着 phpMyAdmin 辅助数据表的数据库。在这里输入与 phpMyAdmin 基本配置相同的用户名和密码:

```
// in config.inc.php
$cfg['Servers'][$i]['controluser'] = 'root';           // MySQL user name
$cfg['Servers'][$i]['controlpass'] = 'xxx';
```

如果没有把 phpMyAdmin 辅助数据表创建在专用的数据库 *phpMyAdmin* 里而是创建在了另外一个数据库里，就必须把下面这行语句中的 *phpMyAdmin* 替换为那个数据库的名字：

```
$cfg['Servers'][$i]['pmadb']      = 'phpmyadmin'; // DB name
```

6.8.2 SQL 书签和历史记录

如果经常在 phpMyAdmin 里执行 SQL 命令，总是重复输入一些同样的命令就会让人觉得很麻烦。其实 phpMyAdmin 是可以把这些命令保存在一个专门为准备的数据表里的，只须对 config.inc.php 文件做一个小改动，告诉 phpMyAdmin 应该把最近执行过的 SQL 命令或命令序列保存到哪个数据表里，就可以享受到这种便利了：

```
# in config.inc.php
$cfg['Servers'][$i]['bookmarktable'] = 'pma_bookmark';
$cfg['Servers'][$i]['history']      = 'pma_history';
```

从现在起，将可以给正在 phpMyAdmin 程序的 SQL 窗口里处理的 SQL 命令起个名字保存起来（这就是所谓的“书签”功能）。保存起来的命令可以通过 phpMyAdmin 程序的 SQL-history 对话框去查看、执行或编辑（如图 6-19 所示）。注意，这个历史记录只能显示用 Submit（提交）或 View only（仅供查看）按钮执行/编辑过的 SQL 命令。

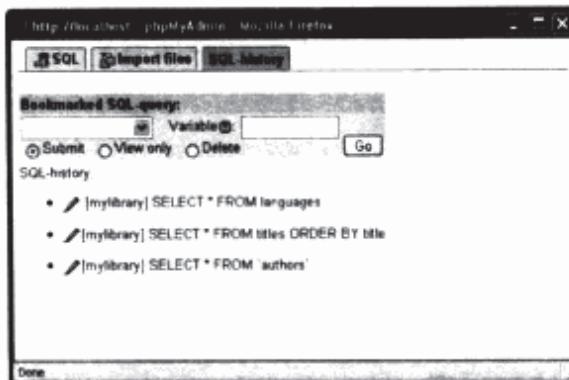


图 6-19 SQL 书签和历史记录

6.8.3 关联/引用关系信息的保存

phpMyAdmin 单靠它自己是没有能力判断各有关数据表都是通过哪些个字段关联在一起的（即使已经为 InnoDB 数据表定义了外键规则也是如此）。不过，这类信息可以由用户来设置，这样可以让日后的数据表编辑工作大为简化。需要在 config.inc.php 文件里做一个小改动：

```
# in config.inc.php
$cfg['Servers'][$i]['relation'] = 'pma_relation';
$cfg['Servers'][$i]['table_info'] = 'pma_table_info';
```

现在，将可以通过任何一个数据表的 Structure（结构）页面上的 Relation View（关系）链接进入一个对话框，并在对话框中把每一个外键字段与另一个数据表里某个字段关联起来。（对于已经定义了外键规则的 InnoDB 数据表，可以省略这一步。）不仅如此，还可以设置当前数据表里的哪些个数据列将在查看另一个引用了当前数据表的数据表时伴随着那个数据表里的数据列一同显示出来。

为了帮助大家更好地理解这个问题，来看一个例子。在本书的示例数据库 *mylibrary* 里，*titles* 数

据表里的 *pubID*、*catID* 和 *langID* 字段分别与 *publishers*、*categories* 和 *languages* 数据表相关联；可以在图 6-20 中的 Link to (关联到) 部分看到这些关联关系。注意画面最下方的 Choose Field to display 下拉列表框，框里的 title (“书名”) 是 *titles* 数据表中的一个数据列的名字。这样，如果另一个数据表引用了 *titles* 数据表，当查看那个数据表时，将看到来自 *titles.title* 字段的书名而不是来自 *titles.titleID* 字段的数字编号。

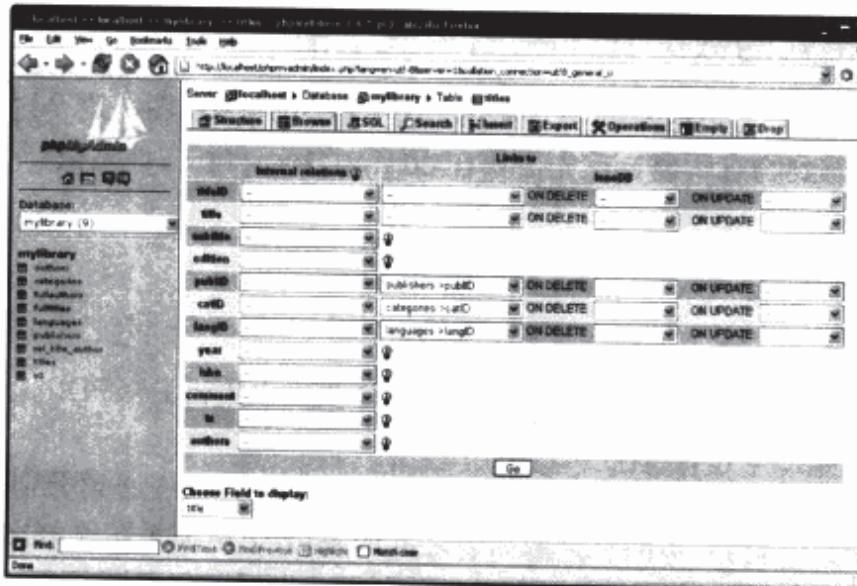


图 6-20 mylibrary 数据库中从 *titles* 数据表到其他数据表的引用关系

如果某个数据库里的数据表很多，为全体数据表输入这种“额外”的关联/引用关系信息的确有些麻烦，但这种麻烦会在以后使用 phpMyAdmin 处理这个数据库时带来许多方便。

- 查看某个数据表的内容时，它的外键字段都将被显示为一个链接。当单击一个这样的链接时，与之关联的数据表里的有关数据记录就会被显示出来。
- 当在某个数据表里修改一条记录或插入一条新记录时，用不着在外键字段里输入一个 ID 编号，只须通过一个下拉列表框从与之关联的字段里挑选一个即可（如图 6-21 所示）。这种下拉列表框里的内容将重复显示两次：第一次按 ID 字段排序，第二次按相关字段名排序。

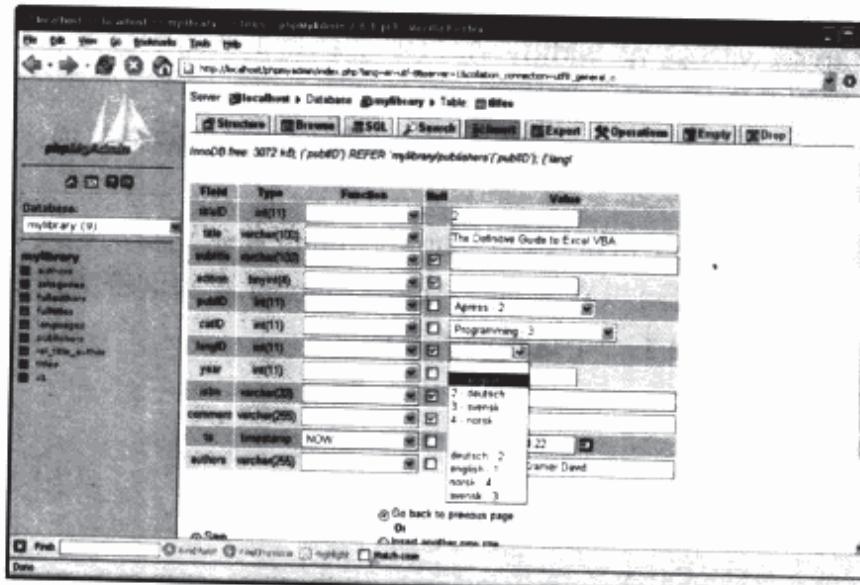


图 6-21 下拉列表框简化了外键字段的输入和编辑工作

- 可以通过 Table (数据表) 页面的 Structure (结构) 子页面里的 Data Dictionary (数据字典) 链接查看到关于数据库结构的详细信息，这些信息很适合打印出来存档（作为数据库文档的一部分）。
- 可以创建一个 PDF 文件，以图文并茂的方式把数据表的结构和所有的相关信息保存为一份文档。

6.8.4 创建 PDF 格式的数据表关联/引用关系图

还可以用 phpMyAdmin 为数据库绘制一张数据表关联/引用关系图。这个过程虽说有点儿麻烦，但在看到结果时肯定会感到物有所值。首先需要在 config.inc.php 文件里做一个小改动：

```
# in config.inc.php
$cfg['Servers'][$i]['table_coords'] = 'pma_table_coords';
$cfg['Servers'][$i]['pdf_pages'] = 'pma_pdf_pages';
```

现在，就可以按照以下几个步骤去创建这种 PDF 文档了。

- (1) 在 Database (数据库) 页面的 Structure (结构) 子页面上单击 Edit PDF (编辑 PDF) 链接。
- (2) 为 PDF 文件命名并选中该文件进行编辑。
- (3) 选中想记载到文档的所有数据表（因为某些原因，每个数据表必须分别选取）。
- (4) 为了在 Structure (结构) 页面里把数据表之间的关联/引用关系图绘制出来，必须为每一个数据表在这个页面里设定一个起始坐标。在一个 A4 页面上，坐标取值范围是：X 坐标，从 0 (左) 到 750 (右)；Y 坐标，从 0 (上) 到 600 (下)。
- 如果 Web 浏览器足够好，还可以利用鼠标拖放操作直观地把数据表摆放到屏幕上的适当位置（通过 Toggle scratchboard (切换草稿板) 按钮进行切换）。
- (5) 以下选项用来设置 PDF 文档的格式。
 - Show grid: 以纵横交错的坐标网格作为显示背景。这在绘图布局阶段很有用。
 - Show color: 设置数据表之间的连线颜色。（提醒：彩色线条在黑白打印机上的输出效果往往不是很好。）
 - Show dimension of Tables: 在 PDF 文档的坐标系统里显示每一个数据表的占地尺寸。这在绘图布局阶段很有用。
 - Display all tables with same width: 在 PDF 文档里把所有的数据表显示为同样的宽度。在某些场合，图表宽度统一的文档比较容易阅读。
 - Data Dictionary: 这个选项决定着要不要把对数据表属性的细节描述集成到 PDF 文档里去（每页一个数据表）。

完成设置后，单击 OK 按钮生成 PDF 文档；phpMyAdmin 将把这份文档发送到 Web 浏览器里去。根据 Web 浏览器的配置情况，或者可以立刻通过 Adobe Acrobat 看到这份文档，或者需要先把它保存为一个本地文件再说。图 6-22 给出了一个这样的关系图。

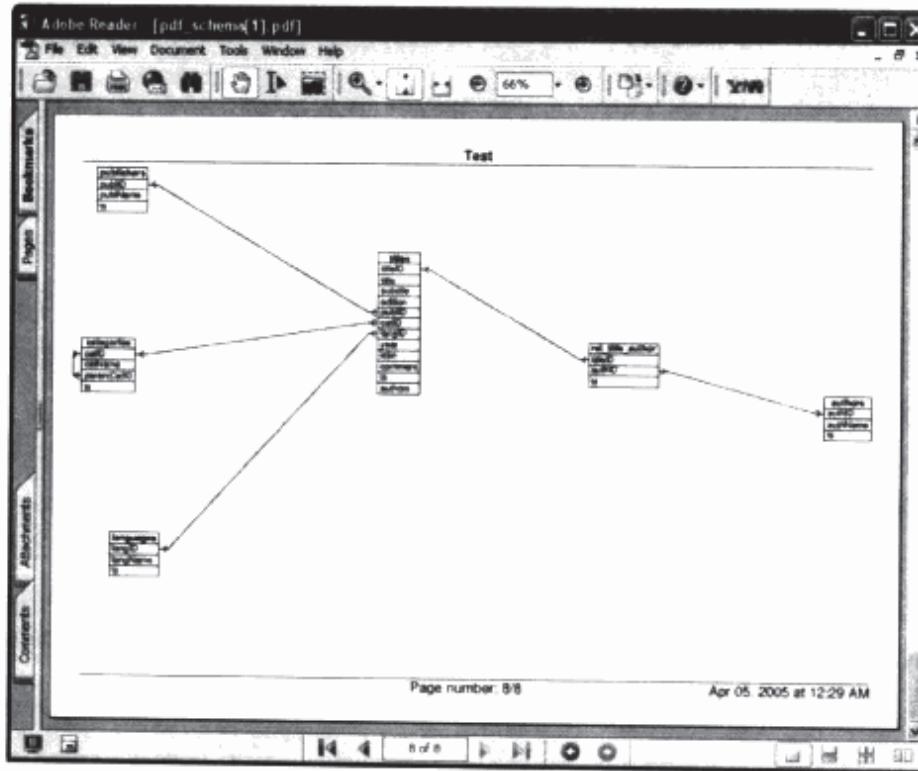


图 6-22 mylibrary 数据库的数据表关联/引用关系图

6.8.5 格式转换（数据列内容的另类显示效果）

phpMyAdmin 可以根据给定规则对有关数据列的内容做一些格式转换。这为数据表提供了一些神奇的显示效果。

□ 内嵌图片：如果 BLOB 数据列里保存的是图片 (JPEG/PNG)，这些图片可以直接显示在 Table (数据表) 页面。图 6-23 中所示数据表的 thumbnail 数据列里存放着一些 JPEG 图片；这是为该数据列设置了 MIME 类型 *image/jpeg* 和格式转换规则 *image/jpeg inline* 之后的显示效果。

#	Image	Model	Orientation	Date/Time	ExposureTime	FNumber	Flash	ExposureBiasValue	WhiteBalance	Thumbnail
1	Canon	Canon DIGITAL SLR 40D	1	2004-08-05 15:40:29	1/250.5	45/10	16.00	0		
2	Canon	Canon DIGITAL SLR 40D	1	2004-08-07 09:46:17	1/317.9	32/10	16.00	0		
3	Canon	Canon DIGITAL SLR 40D	1	2004-08-07 10:09:00	1/317.9	26/10	16.00	0		
4	Canon	Canon DIGITAL SLR 40D	1	2004-08-07 11:14:04	1/403.5	71/10	16.00	0		

图 6-23 带有内嵌图片的 Table 页面

- **超文本链接。**如果文本字段里包含着 Web 网址或文件名，它们可以在 Table（数据表）页面上显示为去往那些网站或文件的超文本链接。

为了让 phpMyAdmin 知道应该把格式转换规则存放到哪一个数据表里，我们需要在 config.inc.php 文件里做一个小改动：

```
# in config.inc.php
$cfg['Servers'][$i]['column_info'] = 'pma_column_info';
```

在这之后，如果想改变某个数据列的属性，phpMyAdmin 提供了 3 个新的输入字段：

- **MIME/Type:** 你可以通过这个列表框为数据列选择一个适当的数据类型（比如 *image/jpeg* 或 *text/plain*）。请注意，这个列表框里的 *auto-detect* 选项不管用。
- **Transformation options:** 可以通过这个字段更细致地设置格式转换工作。比如说，可以在根据格式转换规则 *text/plain link* 而显示出来的地址链接前面再加上一个前缀。
- **ISBN:** 在 *mylibrary* 数据库的 *titles* 数据表里有一个名为 *ISBN* 的字段。如果为这个字段设置了 MIME 类型 *text/plain*、格式转换规则 *text/plain link* 和细节选项 '<http://www.amazon.com/exec/obidos/ASIN/>'（注意，有单引号），phpMyAdmin 就会根据 ISBN 数据列的实际取值生成一个指向 [amazon.com](http://www.amazon.com) 网站的超文本链接。

第7章

Microsoft Office 和 OpenOffice/StarOffice

在实践中，经常会遇到 Microsoft Office 程序需要访问一个数据库的情况，比如利用电子表格进行计算、把数据显示为图表、把数据插入一份邮件合并文档等。在这一章里，将介绍如何在 Microsoft Office、OpenOffice/StarOffice 和 MySQL 之间创建一条连接。这里有以下两种解决方案可供选择：

- ODBC (Open Database Connectivity)。ODBC (数据库通用互连标准) 是一种很流行 (尤其是在 Windows 环境下) 的机制，它是人们为了让应用软件能够以一种统一的方式去访问各种数据库系统而定义的一套编程接口。这个方案只需为数据库系统安装一个所谓的 ODBC 驱动程序就可以实现，ODBC 驱动程序的作用是在 ODBC 系统和数据库之间创建一个接口。MySQL 数据库系统的 ODBC 驱动程序叫做 Connector/ODBC，它可以从 <http://dev.mysql.com> 网站免费下载。
- JDBC (Java Database Connectivity)。JDBC (数据库 Java 互连标准) 是 ODBC 的 Java 版本。在 UNIX/Linux 环境下，JDBC 和 Connector/J (MySQL 数据库系统的 JDBC 驱动程序) 的组合构成了从办公软件 StarOffice 和 OpenOffice 访问 MySQL 数据库最简单的解决方案。(ODBC 也有适用于 UNIX/Linux 环境的版本，但是有许多发行版本没有收录它)。

7.1 安装 Connector/ODBC

Connector/ODBC 3.51 (以前叫 MyODBC) 是 MySQL 数据库系统的 ODBC 驱动程序。这个驱动程序支持 ODBC 3.5.1 版本里定义的全部功能 (资料来源：在线手册 *Complete Core API and Level 2 Features*)。为了保证它的安装效果，应该从 MySQL 公司的官方网站获得 Connector/ODBC。就目前而言，Connector/ODBC 可以在以下地址找到：<http://dev.mysql.com/downloads/connector/odbc/3.51.html>。

安装过程很简单，只要把 myodbc3.dll 和 myodbc3S.dll 这两个 DLL 文件复制到 Windows\System32 子目录并加以注册就行了。Connector/ODBC 没有用到 libmysql.dll 函数库 (虽然别的 MySQL 应用软件经常用到它)，所以可以不安装它。

如果想检查一下 Connector/ODBC 是否安装成功，可以去查看 ODBC 驱动程序对话框 (如图 7-1 所示)；这个对话框可以通过菜单命令 Settings (设置) | Control Panel (控制面板) | Administrative Tools (系统管理工具) | Data Sources (数据源) 打开。

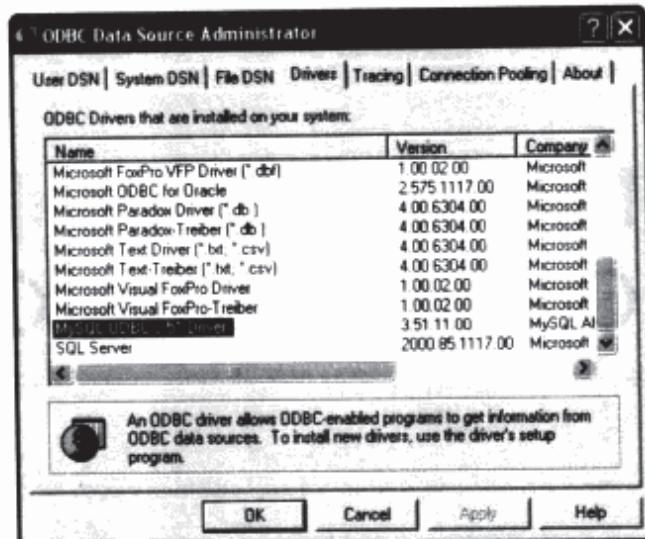


图 7-1 ODBC 驱动程序清单

注解 ODBC也可以在UNIX环境下使用，所以Connector/ODBC也有适用于UNIX/Linux环境的版本。但在本书里，只讨论Connector/ODBC在Windows环境下的应用。

设置 ODBC 数据源

反复输入同样的数据库连接参数（用户名和密码等）是件枯燥乏味的事情，ODBC 提供的数据源（data source）定义机制可以帮助我们减少这种烦恼。一旦设置了数据源，只须用鼠标单击一个数据源的名字（data source name, DSN）就可以快速建立起一条与数据库服务器的连接。

有许多带有 ODBC 接口的程序只是把 DSN 当做一种创建数据库连接的简便方法，但包括 Access 和 Excel 在内的另外一些程序却只有通过 DSN 才能建立一条 ODBC 连接。换句话说，如果要让 Access 或 Excel 创建一条与 MySQL 服务器的 ODBC 连接，就必须先设置好数据源。

DNS 通常是在 ODBC Data Source Administrator（ODBC 数据源管理员）对话框里设置的。（有不少 Windows 程序有自己的 DNS 设置对话框，但除了外观有所不同以外，在本质上并没有什么区别。）这个对话框可以通过菜单命令 Start（开始）| Settings（设置）| Control Panel（控制面板）| Administrative Tools（系统管理工具）| Data Sources(ODBC)（ODBC 数据源）打开，可以在该对话框对 3 种 DNS 做出设置，它们是：User DSN、System DSN 和 File DSN。

- **User DSN:** 这种 DSN 只有它们的定义者才能使用。
- **System DSN:** 这种 DNS 允许本地计算机上的全体用户使用。User DSN 和 System DSN 的设置数据都由 ODBC 内部管理，用户是查看不到的。
- **File DSN:** 这些 DSN 其实是一些*.dsn 文件，它们以文本格式保存着与数据源建立连接所需要的所有设置数据。这些文件通常存放在 Program Files\Common Files\ODBC\Data Sources 子目录里。从某种意义上讲，File DSN 相对要更加透明一些，因为用户想了解的信息就明明白白地写在那里。

如果想访问一个以上的 MySQL 数据库，必须为它们当中的每一个分别设置一个数据源。

定义一个新数据源的第一步是在 User DSN 或 System DSN 选项卡里单击 Add（添加）按钮并在随后出现的对话框里选择一种 ODBC 驱动程序。具体到这里的讨论，正确的选择是 MySQL。双击 ODBC 驱动程序名将打开一个如图 7-2 所示的对话框，需要在这里做出以下设置：

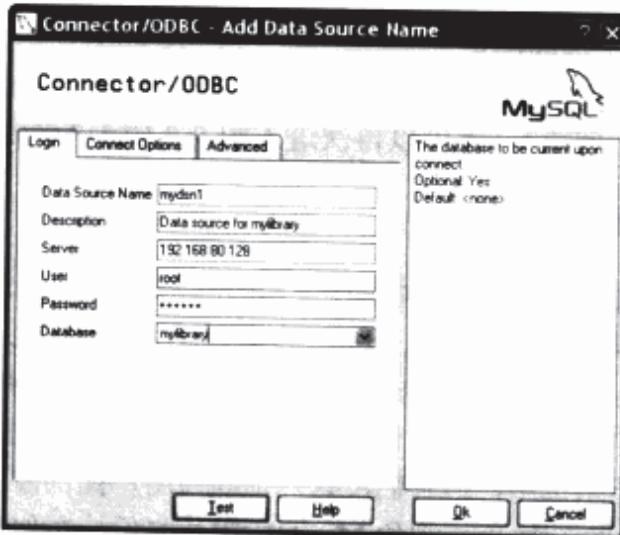


图 7-2 Connector/ODBC 的基本设置

- **Data Source Name:** 给新数据源命名（ODBC 程序将根据这个名字去寻找和访问数据源）。选择一个容易记忆并且有意义的名字，比如说，可以把与这个数据源对应的数据库的名字或是这个数据源的用途嵌在它的名字里。
 - **Server:** 给出 MySQL 服务器的主机名或 IP 地址。如果它是本地计算机，那就填上 *localhost*。
 - **User 和 Password:** 将在第 11 章讨论的 MySQL 访问控制机制对 Connector/ODBC 同样有约束力：事实上，没有一个 MySQL 程序能够例外。（如果无法建立一条 Connector/ODBC 连接，访问权限不足往往是最有可能是原因。）
需要在 User 和 Password 文本框里输入将用来访问数据库的用户名和密码。不过，是否应该这么做却是个值得考虑的问题。首先，这么做的后果是其他人也可以使用这个 DSN 去访问用户的数据库——只要用鼠标选中它就行；其次，密码没有经过加密，所以很容易被别人看到。为了避免这种信息安防风险，可以把这两个文本框空着不填。以后，每当用户本人（或别人）使用这个 DSN 去连接 MySQL 服务器时，如图 7-2 所示的对话框就出现在屏幕上，必须输入一个正确的用户名和密码才能继续工作。从保护信息安全的方面讲，这是个简单实用的办法。但水平一般的用户却很可能对此感到困惑。如果他们只是搞不清楚应该输入哪一个用户名和密码也就罢了，万一在这个对话框的其他输入字段里胡乱输入，可能会弄出难以收拾的结局来。
 - **Database:** MySQL 数据库的名字（比如，如果打算去访问这本书里的示例数据库，就应该输入 *mylibrary*）。
 - **Connection Options:** 如果这个新数据源没有使用默认的 3306 号端口，就需要在这个选项卡的 Port（断开）字段里给出正确的 IP 端口号。在绝大多数场合，这个字段无须填写。这个选项卡上的其他输入字段一般也用不着填写。
请注意，若是 MySQL 和 Connector/ODBC 没有运行在同一台计算机上，并且在两台计算机之间有一道防火墙，就必须打开 3306 号端口；否则，MySQL 与 Connector/ODBC 之间将无法通信。
 - **Advanced:** 这个选项卡上有许多可供设置的细节选项，将在第 19 章讨论 ADO 编程技术时再进行介绍。目前只需记住要选中 *Don't optimize column width* 和 *Return matching rows* 复选框即可。这两个选项是确保 Microsoft Access 和 Excel 能够与 MySQL 正确通信所必需的。
- 设置工作告一段落之后，单击 Test（测试）按钮对当前设置能不能让 Connector/ODBC 与 MySQL

服务器成功地建立起连接做一个快速测试。

如果以后需要修改某个 DSN 的设置，先打开 ODBC Data Source Administrator 对话框，再双击那个 DSN（或 Configure（配置）按钮），就可以再次进入图 7-2 所示的对话框。

7.2 Microsoft Access

Access 和 MySQL 是两个彼此截然不同的程序。Access 有非常友好的用户操作界面，大大简化了数据库的设计工作和数据库程序的开发工作。但是，如果有 3 个或 4 个以上的用户同时去访问数据库，Access 的速度将会变得相当慢。因此，Access 不适合用来开发基于因特网的应用程序。

与 Access 相比，MySQL 在多用户环境里的运行效率和安全性要高得多。不过，虽说现在已经有了许多种 MySQL 用户操作界面可供选用，但它们没有一个能向用户提供像 Access 那样的方便。

如果把这些因素都考虑进来，让 Access 和 MySQL 相互取长补短似乎是个很不错的主意。虽然存在着许多的局限性，但把 Access 和 MySQL 结合起来之后的应用前景还是相当光明的。

- 可以把 Access 数据表导入 MySQL，然后在 MySQL 里对它们做进一步编辑。
- 可以把 MySQL 数据表导入 Access 或是在 Access 里创建一个 MySQL 数据表的链接，然后 Access 里它们做进一步编辑。
- 可以把来自 Access 和来自 MySQL 的数据可以相互混杂、结合在一起。
- 可以用 Access 去设计新数据库，然后再把整个数据库导入 MySQL。
- 可以通过 Access 的用户操作界面去修改 MySQL 数据表的内容。
- 可以先利用 Access 去编写和调试比较复杂的查询语句，然后再把最终的 SQL 代码放到脚本里供 MySQL 用户使用。
- 可以创建和打印数据库报表。
- 可以使用在 Access 里开发的 VBA 代码去自动处理来自 MySQL 数据库的数据。
- 可以利用 Access 提供的方便功能去创建 MySQL 数据库的数据表关联/引用关系图。

注解 本节内容的讨论前提是读者对 Access 已经相当熟悉。对 Access 的介绍不是本书的讨论范围，有需要的读者请自行参阅其他相关书籍。本节中的示例已全部在 Access 2000 环境下通过了测试。

1. 问题

千万不要把目标定得高不可攀。Access 从来都不是 MySQL 的一种用户操作界面，今后也不会是。因此，如果在集成这两种程序的过程中遇到了种种困难，不要大惊小怪。

- Access 和 MySQL 都有一些自己独有的数据类型。这些数据类型会给数据库和数据表的导入/导出操作带来许多麻烦。比如说，MySQL 数据表里的 *ENUM* 和 *SET* 字段在被导入 Access 后会变成普通的文本字段。再比如说，*FLOAT* 类型的单精度浮点数在这两种程序里的定义是不一样的，如果打算用 Access 去处理 MySQL 数据表，就应该尽可能地使用 *DOUBLE* 类型的双精度浮点数。

还有，如果把一个 MySQL 数据表导入 Acees 之后又导出回 MySQL，这个 MySQL 数据表中可能会有许多属性发生了变化。

- MySQL 目前只能在 InnoDB 格式的数据表之间创建链接。在 Access 里创建的数据表关联/引用关系（包括各种外键规则）在数据表被导入 MySQL 之后往往会丢失。
- MySQL 使用的 SQL “方言”与 Access 使用的不完全一致。在 Access 环境里开发的 SQL 查询

很可能需要做些细小的修改才能在 MySQL 环境里正常工作。

2. 注意事项

- 如果打算在 Access 里通过数据表链接对 MySQL 数据表里的数据进行修改，就必须在定义 ODBC 数据源(也就是 DSN)的时候启用 Return Matching Rows 选项。如果使用的是 Access 2.0，那就还需要启用 Simulate ODBC 1.0 选项。
- 如果打算在 Access 里通过数据表链接对某个 MySQL 数据表的数据进行修改，就必须给这个 MySQL 数据表配上一个 *TIMESTAMP* 字段。(还好，这个字段除了需要由用户去定义以外，其他的事用不着操心；MySQL 和 Access 都能自动地把数据记录的最近一次修改时间存入这个字段。Access 需要这项信息去区分被修改过和没有被修改过的数据。)
- 所有的 MySQL 数据表都必须配有一个主索引(它通常是一个带有 *AUTO_INCREMENT* 属性的 *INT* 字段)。
- 数据类型不兼容是许多问题的根源：
 - MySQL 数据表使用的是 *DOUBLE* 而不是 *FLOAT*，使用的是 *DATETIME* 而不是 *DATE*。
 - MySQL 数据表要避免使用 *BIGINT*、*SET* 和 *ENUM* 数据类型。
 - Access 在处理 MySQL 数据表中的 *BLOB* 字段时偶尔会出现一些问题，它会认为自己是在和一个 OLE 对象打交道。

提示 在使用 Access 通过数据表链接去处理一个 MySQL 数据表的时候，如果在修改一条数据记录之后看到了 #deleted 提示，就表明上面列出的条件没有得到全部满足。

在 Access 里开发好一个查询之后，当把它的 SQL 代码拿到 MySQL 里去使用时，往往会遇到一些兼容性方面的问题。(Access 使用的 SQL “方言”与 MySQL 使用的不完全一致。)下面这个办法应该能够解决这类问题：用 Access 通过数据表链接打开那个 MySQL 数据表，在 Access 里执行菜单命令（此时 Access 的查询命令构造窗口要保持在打开状态）：Query (查询) | SQL Specific | Pass-through。这条菜单命令的作用是告诉 Access 在构造查询命令时不要使用自己的“方言”，必须严格遵守 SQL 语言标准。

7.2.1 数据表的导入和导出

1. 导入型数据表与链接型数据表

如果打算在 Access 里处理 MySQL 数据，有两种方案可供选择：一是先把数据表导入 Access 再对其进行处理；二是在 Access 和 MySQL 数据表之间创建一个链接。

- 对于链接型数据表，可以插入或修改数据，但数据表本身仍由 MySQL 负责管理。在这个方案里，Access 只是一个操作界面，不能用它去改变数据表的结构和属性（如添加或删除一个数据列）。这个方案的亮点主要有两个：一是数据表链接的创建速度非常快（与数据表的大小无关），二是数据仍存放在 MySQL。
- 对于导入型数据表，因为它们已经成为了 Access 数据库的一部分，所以在处理和操作方面没有任何限制，就算是改变它们的结构或属性，也属于 Access 的问题。导入型数据表现在都保存在 Access 数据库文件里，可以在 Access 的“势力范围”内对它们进行任何操作。不过，在日后把导入型数据表“归还”给 MySQL 的时候可能会带来点儿麻烦：有几种 Access 数据类型无法自动转换为 MySQL 数据类型，需要由以手工方式来处理。

2. 把MySQL数据表导入Access

把 MySQL 数据表导入 Access 的第一步是打开一个现有的 Access 数据库或者创建一个全新的(空白)数据库。接下来, 执行菜单命令 File (文件) | Get external data (获取外部数据) | Import (导入), 并在弹出的文件选择对话框里选中 ODBC 文件类型, 这将打开 Select Data Source (选择数据源) 对话框。在这个对话框的 Machine data sources (机器数据源) 选项卡里列出了所有的 User DSN 和 System DSN, 需要在这里选择要使用的 DSN。屏幕上将出现图 7-2 所示的对话框, 提示输入用户名和密码(如果在刚才设置 ODBC 数据源的时候已经设置了用户名和密码, 这个对话框将不出现)。

如果能与 MySQL 建立起连接, 就会在接下来的对话框(如图 7-3 所示)里看到与给定 DSN 相对应的数据表里的数据表名单。把想导入的数据表全部选中(Ctrl+单击), 再单击 OK 按钮, 导入工作就开始了。如果数据表比较大, 导入过程可能需要花费一些时间。(Access 将为那些数据表创建一份本地复制并把它们保存到一个 Access 数据库文件里去。)

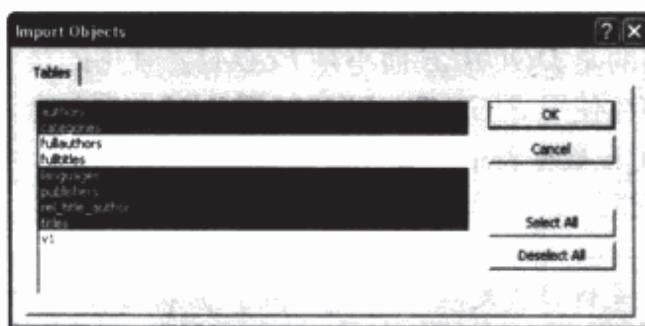


图 7-3 把 MySQL 数据表导入 Access: 挑选数据表

注意, 有许多数据列属性(比如 *AUTO_INCREMENT*)会在导入过程中丢失, 索引也有可能发生变化(例如, 一个 MySQL 中的 *UNIQUE* 索引在 Access 里变成一个允许有关字段重复取值的索引)。

3. 在Access里为MySQL数据表创建链接

创建数据库链接的办法与导入 MySQL 数据表基本一样, 唯一的区别是此时需要执行菜单命令 File (文件) | Get external data (获取外部数据) | Import (导入) | Link tables (链接数据表)。

注意 如果在 MySQL 里改变了某个数据表的属性(比如说, 给它增加了一个数据列), 就必须在 Access 里刷新与这个数据表的连接。具体做法是: 执行菜单命令 Tools (工具) | Database Utilities (数据库工具) | Linked Table Manager (链接型数据表管理器), 然后在弹出的对话框里选择需要刷新的数据表。

4. 把Access数据表导出给MySQL

如果把 MySQL 数据表导入 Access 之后对它进行了修改, 或者如果在 Access 里开发出一个完整的数据库之后想把它转移到 MySQL, 将需要把我们在上一节里做的事情反过来做一遍。

首先, 在 Access 的数据库窗口里选取有关的数据表, 然后执行菜单命令 File (文件) | Export (导出) 并在弹出的文件选择对话框里选中 ODBC 文件类型。在接下来的 Export (导出) 对话框里, 需要给出数据表在 MySQL 环境下的名字。(如果还想继续使用数据表在 Access 里的名字, 直接单击 OK 按钮即可。)

Access 的数据表导出功能总的来说还是不错的, 但有些地方还是需要人工参与。下面是从 Access 导出数据表时需要特别注意的几个基本问题:

- 在 Access 里定义的所有索引（包括主索引在内）都将丢失。
- 没有把 Access 中的数据列属性 *Required* 翻译为 MySQL 中的 *NOT NULL* 属性。
- 没有把 Access 中的数据列属性 *Autonumber* 翻译为 MySQL 中的 *AUTO_INCREMENT* 属性。
- 某些数据类型翻译得不正确或不够好。比如说，把 Access 中的 *Currency* 类型翻译成 MySQL 中的 *DOUBLE* 类型——这里的最佳选择应该是 *DECIMAL*。总之，应该仔细检查 MySQL 数据表有没有这些错漏。

注意，Access 每次只能导出一个数据表。如果想导出一个完整的数据库，就必须为每一个数据表重复一次上述过程。

7.2.2 数据库转换器：Access→MySQL (exportsql.txt)

因为 Access 在向 MySQL 导出数据表时存在着上面列举的种种问题，有位名叫 Pedro Freire 的程序员编写了一个能够把一个完整的 Access 数据库写入一个*.sql 文件的数据库转换器，它生成的*.sql 文件可以直接用做 mysql 命令的输入。这个转换器的质量比 Access 自备的 ODBC Export 功能要好很多。这个转换器的程序文件 exportSQL.txt 可以从以下地址免费下载：<http://www.cunergi.net/exportsql>。

使用这个转换器来导出 Access 数据库的具体步骤可以在它的程序代码里找到，把它归纳如下：

- (1) 在 Access 里打开准备导出的数据库。
- (2) 按下 Alt+F11 组合键进入 VBA 编辑器，用菜单命令 Insert (插入) | Module (模块) 插入一个新模块，并把 exportSQL.txt 文件整个地复制到新模块里去。注意，VBA 编辑器只能用于导出 Access 2000 数据库。如果准备导出的数据库是用较低版本的 Access 创建的，就必须在 Access 的数据库窗口里打开一个新模块并把有关代码插入到这个模块里去。
- (3) 接下来要在程序代码里对导出选项做一些必要的修改。这些选项在程序代码里被定义为一些常数 (*Private Const name = ...*)，它们控制着导出操作的各有关参数。在绝大多数场合，只需要修改 *ADD_SQL_FILE* 和 *DEL_SQL-FILE* 两个常数，它们给出了导出操作的结果文件名。
- (4) 按下 F5 键开始进行导出。（如果是在“宏”对话框里，必须选择名为 *exportSQL* 的过程。）
- (5) 在导出过程中，可能会看到一些警告消息，它们大多与 Access 和 MySQL 在数据类型方面的不兼容有关，必须单击 OK 按钮同意操作继续进行。这些警告消息也会出现在结果文件里。（受影响的语句行前面有#Warning 标志。）
- (6) 在 MySQL 里创建一个空白的新数据库（用 *CREATE DATABASE* 命令）。
- (7) 执行 esql_add.txt 文件里的 SQL 命令。这可以用如下所示的 mysql 命令来完成：

```
mysql:  
> mysql -u root -p databasename < C:\tmp\esql_add.txt  
Enter password: xxxxx
```

如果遇到问题，可能需要用一个文本编辑器对 esql_add.txt 文件做些修改。在某些场合，对这个转换器的程序代码做一些小修改是必要的。在开始第二次尝试之前，必须先把因第一次尝试半途而废而残留在数据库里的数据全部删除干净；这时候，Access 数据库导出操作生成的第 2 个文件 esql_add.txt 就派上用场了：

```
> mysql -u root -p databasename < C:\tmp\esql_del.txt  
Enter password: xxxxx
```

1. 问题

- Access 允许一些特殊字符出现在数据表和数据列的名字里，但 MySQL 不接受这些字符。解决

这个问题的办法是修改 exportSQL.txt 文件里的有关程序代码。下面的修改（黑体字部分）将使 exportSQL.txt 把那些特殊字符自动替换为下划线（“_”）：

```
Private Function conv_name(strname As String) As String
    ...
    Select Case Mid$(str, 1, 1)
        Case " ", Chr$(9), Chr$(10), Chr$(13), "-", "("
        ...
    End Select
    conv_name = str
End Function
```

- 有些国家使用逗号作为浮点数里的小数点，但 MySQL 不支持这种格式。这个问题可以用以下代码来解决：

```
Sub exportSQL()
    ...
    Select Case crs.Fields(cfieldix).Type
        ...
        Case Else
            sqlcode = sqlcode & conv_str(str(crs.Fields(cfieldix).Value))
    End Select
End Sub
```

- MySQL 要求 *PRIMARY KEY* 数据列必须具备 *NOT NULL* 属性，但 exportSQL.txt 往往无法百分之百地做到这一点，这会在数据库里留下隐患。这个问题的解决办法是用一个文本编辑器对导出结果文件 esql_add.txt 里有关的数据列定义进行改正。

2. 其他的MySQL/Access数据库转换器

除 exportSQL.txt 以外，笔者未能详细测试的 MySQL/Access 数据库转换器（正、反两个方向的都有）还有很多，比如 importsq1 和 MyAccess。这些工具有许多是免费的，另一些则是商业化软件。可以在以下网址找到对这类程序的介绍：<http://solutions.mysql.com/>。

7.3 Microsoft Excel

Excel 能够引起 MySQL 用户兴趣的地方是它的数据分析能力。有了 Connector/ODBC 的帮助，完全可以把 MySQL 数据导入 Excel 工作表进行分析、创建图表等。图 7-4 所示的例子是一份基于 *mylibrary* 数据库的电子表格，它可以告诉我们各家出版公司出版了多少本给定门类的图书。（因为数据记录的个数相对很少，所以这里的分析结果没有多少权威性。如果能为 *mylibrary* 数据库收集到大量的图书信息，这份表格就可以让人们轻而易举地看出各家出版公司在哪一个领域最专业。）

catName	pubName	Auszahl - titleID	Gesamtergebnis
Computer books	Addison-Wesley	3	6
Databases	Apress	2	2
Linux	Galileo	2	4
MySQL	Hanser	1	6
PHP	Markt u. New Riders	1	4
Programming	O'Reilly &	1	5
Relational Databases		1	1
SQL		1	1
Visual Basic		1	3
Visual Basic .NET		3	36
Gesamtergebnis		22	36

图 7-4 一份 Excel 电子表格

用 MS Query 导入数据

不管是想把 MySQL 数据直接插入一份 Excel 工作表（菜单命令是 Data（数据）| Get External Data（获取外部数据）| New Database Query（新建数据库查询）），还是准备根据外部数据创建一份电子表格或图表，Excel 都将启动一个名为 MS Query 的辅助程序。这个程序是 Excel 与外部数据库之间的接口，它还可以帮助用户设置各种导入和查询参数。

在启动后，这个程序将首先显示一个 Choose Data Source（选择数据源）对话框以选择 DNS，然后打开 Query Wizard（查询向导）对话框来引导用户构造一个数据库查询命令。构造数据库查询命令的第一个步骤是选择需要用到的数据表或数据表字段（如图 7-5 所示），接下来的两个步骤分别是设置过滤规则（对应于 WHERE 条件）和设置排序方式。

如果对自己在这 3 个步骤里做出的设置很有把握，现在就可以单击 Finish（完成）按钮退出 MS Query 程序了。不过，建议最好先单击 View data or Edit query（查看数据或编辑查询）选项看看实际的效果。单击这个选项后，MS Query 将进入调试模式，并把查询命令的文本显示出来以便对它进行优化（如图 7-6 所示）。这也是把 MS Query 未能识别出来的数据表关联/引用关系告诉它的好机会；具体做法是：把 ID 字段从一个数据表拖放到另一个数据表，然后在弹出的对话框里对关系属性做一些必要的设置即可。

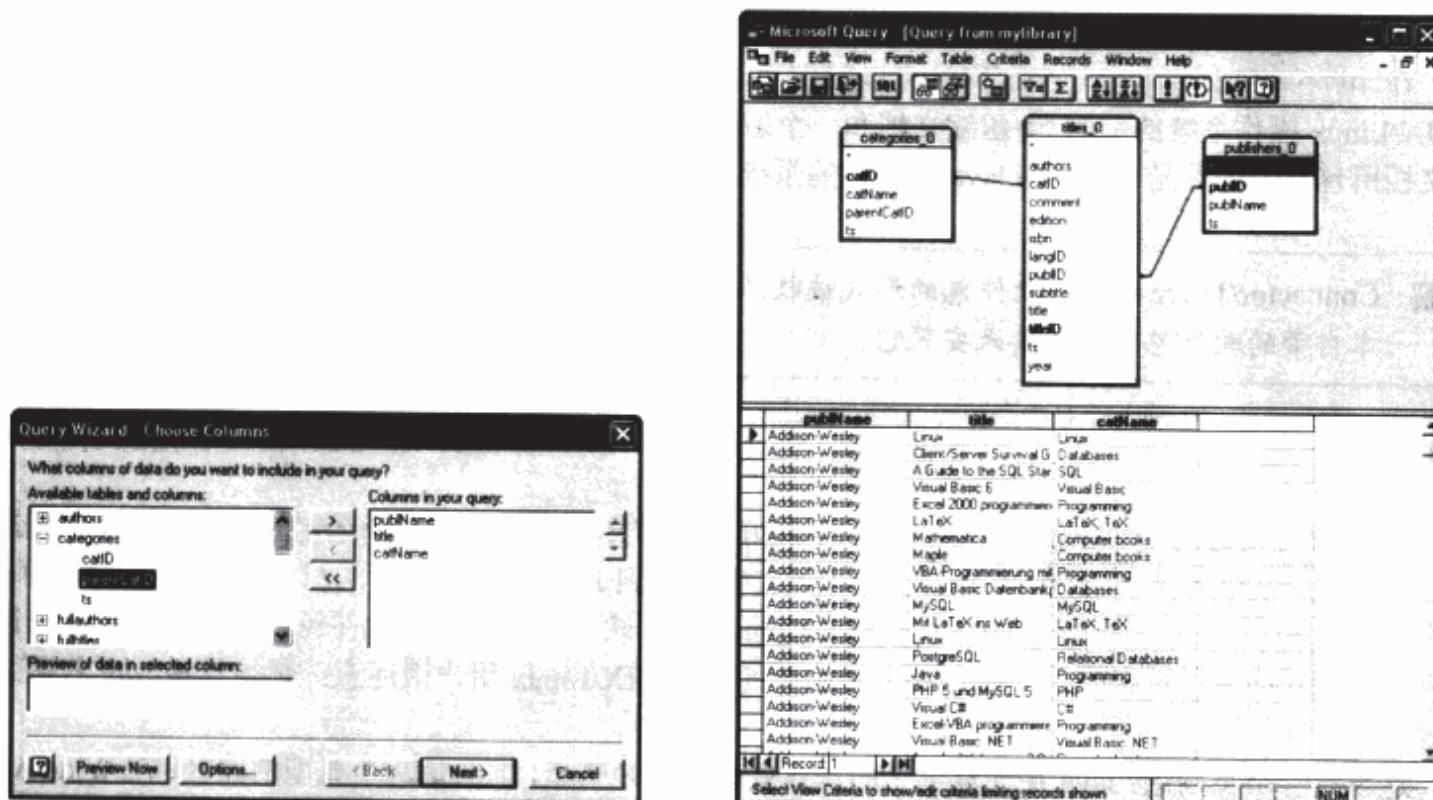


图 7-5 在 MS Query 里选中一个数据列

图 7-6 MS Query 的开发模式

提示 Excel 在导入关于数据的信息时常常会出现一些问题。SQL 函数 `CONCAT()` 可以帮我们解决这些问题当中的一部分。在需要用这个函数来格式化某个数据列的时候，先双击这个数据列打开 MS Query 程序的 Edit Column（编辑数据列）对话框，然后在输入字段 Field（字段）里输入 `CONCAT(...)` 就可以了。比如说，对于 `table.birthdate` 数据列，应该输入 `CONCAT(table.birthdate)`。

7.4 安装 Connector/J

OpenOffice 和 StarOffice 既可以通过 ODBC 访问 MySQL，也可以通过 JDBC 访问 MySQL。如果希望在 Windows 环境下使用 ODBC 进行工作，就必须安装 Connector/ODBC 并设置一个数据源。但在 UNIX/Linux 环境下，JDBC 往往是更好的选择，这是因为 JDBC 在这些操作系统上更容易安装和设置。

JDBC（Java Database Connectivity，数据库 Java 互连标准）是 Java 系统的组成部分之一。本节内容的讨论前提是计算机上已经安装了 Java 运行环境，如 Sun 公司的 J2SE Runtime Environment 5.0 版。这个运行环境是执行 Java 程序所必需的。（OpenOffice 和 StarOffice 不是纯粹的 Java 程序，但它们用到了一些 Java 函数。）如果需要获得一个适用于操作系统的 Java 运行环境，可以从以下网址下载它（用 JRE 做关键字搜索一下）：<http://java.sun.com/j2se/1.5.0/download.jsp>。

7.4.1 Connector/J

JDBC 本身不包含适用于 MySQL 数据库系统的驱动程序。这个任务是由 Connector/J 承担的，可以从 <http://dev.mysql.com/> 网站免费下载 Connector/J（它使用的是 GPL 许可证）。在本书的有关示例中，我们使用的是成熟稳定的 Connector/J 3.1.7 版本（最新的 3.2.n 版本在 2005 年 3 月仍处于 alpha 测试阶段）。

在 <http://dev.mysql.com/> 网站可以找到不同版本的 Connector/J，每一个版本都有一个适用于 UNIX/Linux 操作系统的 tar.gz 压缩文档和一个适用于 Windows 操作系统的 ZIP 压缩文件。这两种压缩文档所包含的文件是一样的：Java 与具体的系统平台无关。

注解 Connector/J 已经以一个软件包的形式被收录在了许多的 Linux 发行版本里，可以用这些发行版本自带的软件包管理工具来安装它。

7.4.2 安装

Connector/J 的安装工作相当简单：先把下载好的压缩文档解压缩到一个选定的子目录，再把释放出来的 mysql-connector-java-n.n.n-bin.jar 文件复制到 Java 安装目录下的/lib/ext 子目录即可。这个*.jar 文件已经包含了所有的驱动代码。Connector/J 软件包里的其他文件包含的是源代码、帮助文档之类的东西，不是运行这个驱动程序所必需的。（UNIX/Linux 用户请注意：必须具备 root 权限才能复制*.jar 文件。）

这里有一个小问题：Java 在计算机上的具体安装位置是哪里。下面是两个典型的路径：在 Windows 环境下为 C:\Programs\Java\jre1.5.0_01\lib\ext；在 UNIX/Linux 环境下为 /usr/lib/jvm/jre-1.4.2-sun/lib/ext。如果使用的是 OpenOffice 2.0，可以利用菜单命令 Options(选项)|OpenOffice.org|Java 打开一个对话框来确定 Java 的具体安装位置（如图 7-7 所示）。

注意，OpenOffice 必须经过重新启动才能加载新安装的 Connector/J 驱动程序，这也包括任务栏上的 Quickstart 程序。

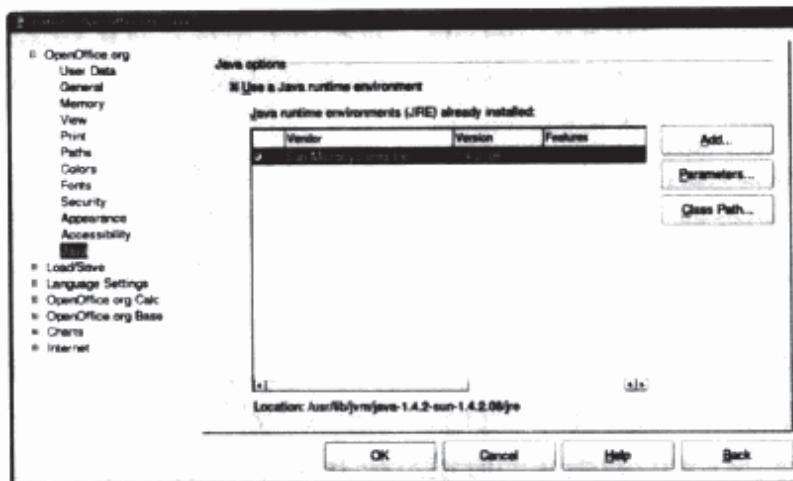


图 7-7 用 OpenOffice 确定 Java 的安装路径

7.5 OpenOffice/StarOffice Base

OpenOffice 2.0 和 StarOffice 8.0 里的一项重大改进是增加了数据库组件 Base。这个组件不仅提供了在 OpenOffice/StarOffice 里创建各种定制数据库的可能性，还大大简化了在 OpenOffice/StarOffice 里管理和使用外部数据库时的操作。不过，在这一节里的讨论内容将只限于如何使用 Base 去访问 MySQL 数据库的基本步骤。（出现较早的 OpenOffice 1.1 和 StarOffice 7.0 也具备数据库功能，但它们都深藏在 Data（数据）菜单里，用起来不太方便。将在本章后面介绍 OpenOffice/StarOffice Calc 程序时进一步介绍 Data（数据）菜单里的有关细节。）

7.5.1 与 MySQL 数据库建立连接

为了打开一个新的数据库项目，执行菜单命令 File（文件）| New（新建）| Database（数据库）并选择 Connect to an existing database 和数据库类型 MySQL。接下来的步骤是在 ODBC 和 JDBC 之间做出选择。如果希望使用 Connector/J，请在这里选择 JDBC。

在第(3)步里，可以单击 Test class（测试）按钮（如图 7-8 所示）去检查一下 Connector/J 是否已经安装成功。如果是，就可以输入数据库的名字和 MySQL 服务器的计算机名了（通常是 *localhost*）。

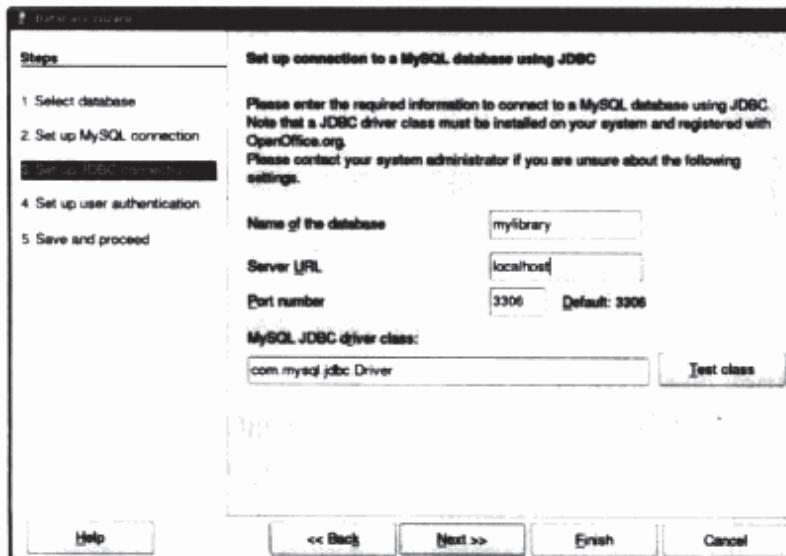


图 7-8 创建一条与 MySQL 数据库的连接

在第(4)步里，需要给出一个 MySQL 用户名。如果这个用户带有密码保护（应该如此），请启用 Password Required（必须提供密码）选项。现在，可以单击 Test Connection（测试连接）按钮去检查一下是否一切顺利。系统将提示输入密码。

在退出 Database Wizard（数据库向导）之前，还需要给出一个文件名。Base 文件的内容最初只有连接参数，以后会增加一些查询、表单和报表；但真正的数据仍存放在 MySQL 服务器那里。换句话说，在 Base 文件里自始至终不会有任何来自数据库的东西，它里面只有用来访问和管理数据库的对象。正是因为这个原因，Base 文件的尺寸往往不会很大。

接下来，OpenOffice/StarOffice 的数据库组件 Base 终于出现了。在 Base 的主窗口里，可以看到 4 个模块，分别是 Table（数据表）、Queries（查询）、Forms（表单）和 Reports（报表）（图 7-9）。在第一次使用 Base 去访问某个 MySQL 数据库的时候，在 Base 里只能看到和使用这个 MySQL 数据库里的现有数据表——因为还没有创建过任何查询或其他数据库对象，这些东西需要把它们创建出来之后才能从 Base 里看到和使用。如果在选中某个数据表之后想预览一下它的格式和内容，在主窗口右下位置的 Document（文档）列表框里做出相应的选择即可。

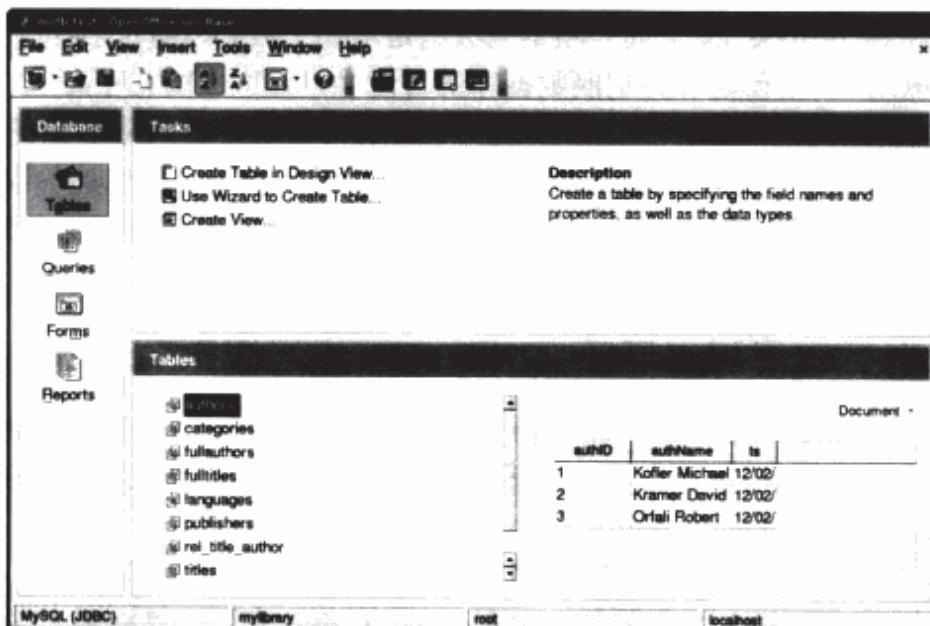


图 7-9 OpenOffice/StarOffice 的数据库组件：Base

7.5.2 Table 模块

进入 Table（数据表）模块后，会看见一份现有数据表和视图的清单。从这里出发，可以执行许多如下操作：

- **打开数据表。** 双击某个数据表的名字就可以打开它。可以浏览这个数据表，添加、修改和删除记录（如图 7-10 所示）。还可以用 Ctrl 或 Shift 键配合鼠标选取一些数据行并把它们拖放到 OpenOffice/StarOffice 的其他组件里去。
- **创建新数据表。** 可以使用简单的数据表编辑器（OpenOffice 称为 Design（设计）视图）或者使用向导去创建新数据表。向导提供了一些现成的数据表模板，如果其中之一能满足要求，就可以省下不少时间。

pubID	pubName	ts
1	O'Reilly & Associates	12/02/04 06:36 PM
2	Apress	12/02/04 06:36 PM
3	New Riders	12/02/04 06:36 PM
4	O'Reilly & Associates	12/02/04 06:36 PM
5	Hanser	12/02/04 06:36 PM
9	Bonnier Pocket	12/02/04 06:36 PM
16	Zsolnay	12/02/04 06:36 PM
17	Ordförande förlag AB	12/02/04 06:36 PM
19	Diacones Verlag	12/02/04 06:36 PM
20	Markt und Technik	12/02/04 06:36 PM
21	Galileo	12/02/04 06:36 PM
23	dpunkt	12/02/04 06:36 PM
24	Sybex	12/02/04 06:36 PM
<AutoField>		
Record 1 of 13		

图 7-10 用 Base 来查看和编辑一个数据表

- 改变数据表的显示效果。通过快捷菜单里的 Edit（编辑）命令打开这个数据表的 Data View 窗口，可以添加、编辑或删除数据列，定义或删除索引等。但这里要提醒大家一句：MySQL 支持的许多功能都不能或不应该在这个窗口里使用。要知道，Base 并不是专为 MySQL 而开发的，它还需要与其他的数据库系统保持兼容，所以它的数据表编辑能力仅限于各种数据库系统都支持的几种常用功能。
- 把数据复制到剪贴板。用快捷菜单里的 Copy（复制）命令将把当前数据表里的全部记录复制到剪贴板。在这之后，可以把那些数据插入到 OpenOffice/StarOffice 的其他组件或是插入到其他的文本编辑器里去（RTF 或 HTML 格式）。
- 创建新视图。这个操作将把用户带到稍后将要介绍的查询编辑器去，在那里构造一个查询，数据库系统会在这个查询的基础上创建出一个新的视图并把它保存到数据库里去（详见第 8 章有关内容）。
- 删除数据表或视图。还可以通过快捷菜单来删除数据库对象。

注意 以上所有操作将直接修改 MySQL 数据库而不仅仅是有关数据的本地复制。如果在 Base 里删除了一个数据库，它就再也无法恢复地消失了。

7.5.3 Queries 模块

Queries（查询）模块是 Base 里最让人感兴趣的部分，因为它可以帮助人们——尤其是 SQL 语言的初学者——方便快捷地构造各种查询命令。在 Base 里构造查询命令的办法有 3 种：在 Design（设计）视图里、使用一个向导、在 SQL 视图里。不过，向导只能用来构造只涉及一个数据表的查询，对涉及两个或更多个数据表的查询无能为力。在 SQL 视图里，只能直接输入 SQL 代码，可这种事就算不用 Base 也能做得很好。因此，下面将重点讨论 Design（设计）视图里的查询编辑器，它和我们在本章前面介绍的 MS Query 程序有许多相似之处。

首先，在查询设计窗口里，双击选中需要用到的所有数据表，然后利用鼠标拖放操作把数据表之间的关联字段链接起来。比如说，在图 7-11 里，*publishers* 和 *titles* 数据表之间的关联字段是 *publID*。为了让 Base 知道有这样一个链接，需要用鼠标把 *publishers* 表里的 *publID* 字段拖放到 *titles* 表里的 *publID* 字段上（也可以按反方向把 *publID* 字段从 *titles* 表拖放到 *publishers* 表）。

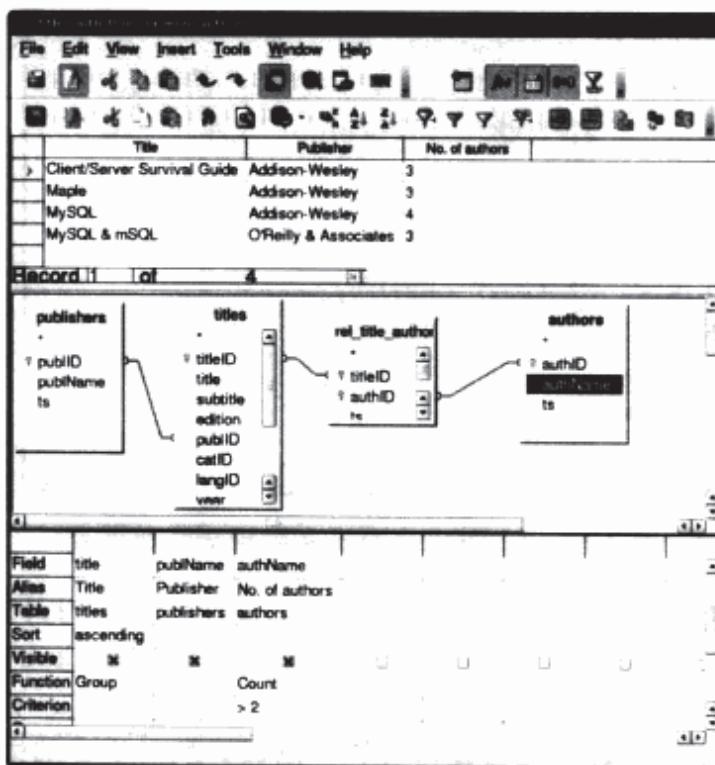


图 7-11 创建并测试查询

在默认的情况下，Base 会把数据表之间的关联关系创建为所谓的 INNER JOINS。这意味着在链接时只有那些在两个数据表里都出现的字段才会被考虑。如果其中一个数据表里有 *NULL* 字段，就需要一个 LEFT JOIN 或 RIGHT JOIN。关系属性可以通过在链接线上双击鼠标的办法来设置。

提示 为了让Base能够为多个MySQL数据表之间的链接关系生成正确的SQL代码，必须在执行菜单命令Edit（编辑）| Database（数据库）| Advanced Settings（高级设置）打开的对话框里取消选中Use the syntax {OJ} for outer joins复选框。

完成这些准备工作之后，剩下的事就很简单了：连续双击选定需要的结果数据列、给它们加上一个标签（也叫“假名”，这里指的是数据列在查询结果中的标题文字），如有必要还可以设置一下排序方式和分组/统计函数。在图 7-11 中的查询里，查询结果按书名（图中的 Title 列）分组并为每一组统计了作者人数（图中的 No. of authors 列）。在这个结果里，只有作者人数是两个以上的图书才会被显示出来。

单击 Execute query（执行查询）按钮将返回查询的结果。如果想知道自己构造出来的查询生成了什么样的 SQL 代码，单击 Design view（设计视图）按钮就可以在设计模式和 SQL 模式之间进行切换。

7.5.4 Forms 模块、Reports 模块和其他功能

Base 还通过它的 Forms（表单）模块、Reports（报表）模块和 Tools（工具）菜单提供了许多其他的功能，但因为本书的主题是 MySQL 而不是 Base，所以下面只对它们做一个非常简单的介绍。注意，这些功能的开发工作目前仍未取得圆满成功，最好只在要求比较简单的场合里使用它们。

在 Forms（表单）模块里，可以创建输入和编辑表单（如图 7-12 所示）。与直接处理数据表相比，这些表单有助于简化输入和搜索数据工作。Base 提供了两种创建表单的办法：通过 Design view 窗口

或使用一个向导。如果对 Base 不熟悉，那还是使用向导比较稳妥；通过 Design view 窗口的办法需要用户具备足够丰富的实践经验，即便如此，它使用起来还是很复杂。（提示：可以通过菜单命令 View（视图）| Toolbars（工具栏）启动 Database（数据库）工具栏，其中有许多在创建表单时需要用到的东西。）

catName	Computer books
catID	1
> Linux Installation,	5
The Def.	3
Client/S	2
Web Ap	8
MySQL	0
MySQL	34
A Guide	6
Visual EProgrammik	51
Excel 20	4
PHP - C Webserver-	8
Nennen	9
Alltid da	9

图 7-12 一个按门类显示图书目录的简单表单

Reports（报表）模块提供了一个向导来帮助人们编制和打印数据库报表，它可以为数据表、视图或查询生成相当漂亮的打印文档。尽管如此，无论是在报表创建方面还是在自身结构方面，Reports 模块与 Microsoft Access 仍有着较大的差距。Reports 模块只能对单个数据库对象里的数据进行处理。如果想把来自多个数据表的关联数据打印出来，就必须先构造出一个适当的查询才行。在报表生成后再对它进行修改几乎没有可能。虽说可以用快捷菜单里的 Edit（编辑）命令把报表打开为一份 Writer 文档，但在那里能做的修改仅限于非常初级的排版操作（比如文本的字号）。由于报表的内部结构无法通过菜单命令或工具栏去访问，比较高级的编辑操作都无法进行。

执行菜单命令 Tools（工具）| Relationship（关系）本应该打开一个编辑器让用户补上 Base 没能识别出来的数据表关联/引用关系上，但对 MySQL 数据库使用这个命令却会返回一条“*The database does not support relations!*”（这个数据库不支持关系）出错消息——这与事实显然不符：MySQL 本来就是关系数据库。

执行菜单命令 Tools（工具）| User Administration（用户管理）可以打开一个对话框让用户对 MySQL 用户进行管理，但这个对话框里的信息几乎都是不正确的。根据笔者个人的经验，这个对话框引起的问题要比它解决的问题多。不要使用它！（用 phpMyAdmin 工具去管理 MySQL 用户更好。）

7.6 OpenOffice/StarOffice 的 Data Source 视图

除相对独立的 Base 组件以外，OpenOffice/StarOffice 中的其他组件都有个所谓的 Data Source（数据源）视图。这个视图可以用 F4 键或菜单命令 View（视图）| Data Source（数据源）打开和关闭，它可以帮助用户从外部数据库把数据输入 Calc、Writer 等程序。

提示 OpenOffice 1.1 和 StarOffice 7.0 里也有 Data Source (数据源) 视图。这个视图在数据表和查询的管理方面提供了与新 Base 组件类似的功能。这些功能都隐藏在一个晦涩难懂的界面后面，而且其中一些编写得还不够好。

7.6.1 建立数据源

如果使用的是 OpenOffice 2/StarOffice 8，就可以在 Data Source (数据源) 视图里看到系统已知的全部 Base 数据源。创建一个新的数据源的菜单命令是 File (文件) | New (新建) | Database (数据库)。如有必要，可以通过菜单命令 Tools (工具) | Options (选项) | Base () | Database (数据库) 打开的对话框改变数据源名称 (DSN) 和 Base 文件之间的对应关系。

在 OpenOffice 1.1/StarOffice 7.0 里，可以用快捷菜单命令 Administrate Data Sources (管理数据源) 启动一个向导。进入向导之后，单击 New data source (新数据源) 按钮，然后在第一个对话框里选择数据库类型 MySQL，在第二个对话框里选中 Connector/J 选项。接下来的配置工作要比在 Base 里复杂：必须按照 *localhost:3306/dbname* 的格式来给出数据源的 URL；例如，如果 MySQL 服务器没有运行在本地计算机上，就要把 *localhost* 替换为真实的计算机名。当然，别忘了把 *dbname* 替换为打算访问的数据库的名字（比如 *mylibrary*）。

7.6.2 数据的导入

在 Database (数据库) 视图里，可以选取一个数据表或一个查询并用鼠标双击动作执行它。现在，就可以把整个数据表或者是用 Ctrl 或 Shift 键配合鼠标选取的语句块插入到电子表格或文本文档，如图 7-13 所示。

The screenshot shows two windows side-by-side. The top window is the 'Data Source' view, showing a tree structure on the left with 'Bibliography', 'oodb-test', 'Queries', and 'Tables'. Under 'Tables', 'authors' is selected. The main area displays a table with columns 'authID', 'authName', and 'ts'. The data includes records 1 through 12. Record 3 is highlighted. The bottom window is a Calc spreadsheet with a single sheet named 'Sheet 1 / 3'. The data from the table in the Data Source view is pasted into the spreadsheet, starting at cell A4. The data in the spreadsheet matches the table above.

authID	authName	ts
1	Kofler Michael	12/02/04 06:36 PM
2	Kramer David	12/02/04 06:36 PM
3	Orfali Robert	12/02/04 06:36 PM
4	Herkey Dan	12/02/04 06:36 PM
5	Edwards Jeri	12/02/04 06:36 PM
6	Ratschiller Tobias	12/02/04 06:36 PM
7	Gertken Till	12/02/04 06:36 PM
8		
9		
10		
11		
12	Yarger Randy Jay	12/02/04 06:36 PM

图 7-13 Data Source (数据源) 视图 (上) 和 Calc 表单 (下)

Part 3

第三部分

基础 知识

本部分内容

- 第 8 章 数据库设计概论
- 第 9 章 SQL 语言入门
- 第 10 章 SQL 解决方案
- 第 11 章 访问权限与信息安全
- 第 12 章 GIS 函数
- 第 13 章 存储过程和触发器
- 第 14 章 管理与服务器配置

第8章

数据库设计概论



一切数据库应用程序的第一阶段工作都是对数据库进行设计，这项工作的好坏对应用程序执行效率的高低、前期编程和后期维护工作的难易程度以及能否在今后灵活修改设计方案等问题将产生巨大深远的影响。在设计阶段埋下的隐患会在其后给开发者和使用者带来无穷的烦恼和痛苦。千万不要掉以轻心！这里没有什么捷径。数据库设计方案的好坏与设计者的知识和经验是否丰富有着很大的关系，本章的篇幅虽然不能算短，但也只能向大家提供一些基本的出发点而已。

在这一章里，将重点讨论关系数据库的基本概念、总结 MySQL 所支持的数据类型和数据表类型，并以一个名为 *mylibrary* 的数据库为例，展示在数据库设计工作中应该遵循的基本原则和步骤。（*mylibrary* 数据库的用途是管理图书信息，这本书中的许多示例都用到了这个数据库里的 *authors* 和 *publishers* 数据表。）除此之外，本章还在数据库索引和数据一致性规则（外键约束条件）的正确使用方面进行了探讨。

提示 编写本章的目的是为了提供一个从事数据库设计工作的理论基础，不涉及如何创建新数据库、如何创建新数据表等具体操作。但如果读者有兴趣动手实践一番，有两种方法可以完成这些操作。

- 最简单的办法是使用 MySQL Administrator 或 phpMyAdmin 等工具（详见本书第 5 章和第 6 章），它们可以通过鼠标操作来定义新数据表的各种属性。
- 用 SQL 命令（如 `CREATE TABLE name ...;`，详见本书第 9 章）创建数据库和数据表。这种做法相对比较繁琐，但好处是可以把有关命令写入 PHP 脚本再执行，这样的脚本在需要创建一些临时数据表的时候会很有用。

8.1 参考读物

独立于具体的数据库系统而专门探讨数据库设计理论和介绍 SQL 语言的图书实在是太多了，而评价这些图书优劣好坏的标准更是千变万化。下面是几本笔者个人认为值得一读的图书。

- Joe Celko: *SQL for Smarties*, Morgan Kaufmann Publishers 出版公司, 1999 年。这不是一本为 SQL 初学者写的书。由于 MySQL 与 ANSI-SQL/92 标准并不完全兼容，所以书中有许多例子目前还不能在 MySQL 数据库系统里套用。尽管如此，这部示例丰富的著作仍不失为一本关于 SQL 的最佳参考书。
- Judith S. Bowman 等: *The Practical SQL Handbook*, Addison-Wesley 出版公司, 2001 年。
- Michael J. Hernandez: *Database Design for Mere Mortals*, Addison-Wesley 出版公司, 2003 年。这本书前半部分略显冗长，但后半部分却写得非常精彩。

- Peter Gulutzan 和 Trudy Pelzer: *SQL-99 Complete, Really*, R&D Books 出版公司, 1999 年。这本书经常被人们提起, 但笔者对这本书并不熟悉。

如果舍不得拿自己辛苦挣来的钱去买书、可现在又对数据库设计非常感兴趣, 可以到网上去找找由 Fernando Lozanow 编写的关于关系数据库设计入门的文章, 见 <http://www.edm2.com/0612/mysql7.html>。

8.2 数据表类型

在创建一个新的 MySQL 数据表时, 可以为它设置一个类型; 这种做法在其他数据库系统里是不多见的。MySQL 支持多种数据表类型, 它们各自有各自的特点和属性, 其中最重要的 3 种类型是 MyISAM、InnoDB 和 HEAP。

如果在创建一个数据表时没有设置其类型, MySQL 服务器将会根据它的具体配置情况在 MyISAM 和 InnoDB 之间做出选择; 默认的数据表类型由 MySQL 配置文件里的 default-table-type 选项决定。

在这一节里, 我们将对 MySQL 所能支持的各种数据表类型以及它们各自的属性和适用场合做一个简单的介绍。

8.2.1 MyISAM 数据表

MyISAM 数据表类型的特点是成熟、稳定和易于管理。只要没有特殊理由选择其他的类型, 就应该选用这个类型。这种数据表类型在 MySQL 内部又细分为以下两种, MySQL 服务器将自行选择它认为最适当的一种来使用:

- **MyISAM Static (静态 MyISAM)** : 如果数据表里的数据列各自都有预先定义好的固定长度, MySQL 服务器将自动选择这种数据表类型。这种数据表的数据存取效率非常高, 而且即使对数据表的修改非常频繁(这里指的是有大量的 *INSERT*、*UPDATE* 和 *DELETE* 命令)也是如此。还有, 这种数据表类型的安全性相当高, 即使出现文件受损或其他问题, 数据记录的提取和恢复工作也比其他类型的数据表容易。

- **MyISAM Dynamic(动态 MyISAM)** : 如果在数据表的定义里有且只有一个 *VARCHAR*、*xxxTEXT* 或 *xxxBLOB* 字段, MySQL 将自动选择这种数据表类型。与静态 MyISAM 类型相比, 这种类型的突出优点是数据表的空间需求量往往小得多: 存储字符串和二进制对象所需要的字节数仅仅是它们的实际长度(再加上几个字节的开销)。

这就使数据记录很可能不都是同样的长度。这样一来, 如果记录被修改了, 它们在数据库文件里的存储位置就可能发生改变并在原先的位置留下一个空洞。于是, 在数据库文件里, 同一条记录的各个字段就不一定是存储在一个连续的字节块里, 而是会散布在各处。当被编辑的数据表变得越来越碎片化时, 数据的存取时间就会变得越来越长。因此, 这种类型的数据表需要人们经常地使用 SQL 命令 *OPTIMIZE TABLE* 或者是某个优化工具来进行碎片整理(*myisamchk*, 见第 14 章)。

- **MyISAM Compressed (压缩 MyISAM)** : 动态和静态 MyISAM 数据表都可以用 *myisamchk* 工具压缩。这种压缩的效果往往可以使数据表的空间占用量减少到原来的一半以下(与数据表的具体内容有关)。虽说以后在读取数据记录时需要对它们进行解压缩, 但在某些场合, 数据表的访问速度甚至会变得更快——这在“低速硬盘+高速 CPU”的系统上体现得尤其明显。

压缩 MyISAM 数据表的最大缺点是不能再对它们进行修改（即它们是只读数据表）。

8.2.2 InnoDB 数据表

除了上面介绍的 MyISAM 格式，MySQL 还支持一种名为 InnoDB 的数据表格式。可以把 InnoDB 看做是 MyISAM 的一种更新换代产品，它至少增加了以下几种新功能：

□ **事务。** InnoDB 数据表里的数据库操作可以被执行为一个事务。这将允许把几条有着内在逻辑关系的 SQL 命令当做一个整体来执行。如果在执行时发生错误，所有的命令（而不仅仅是触发错误的命令）都将失效，就好像从未执行过这些命令一样。除此之外，事务机制还可以改善数据库应用程序的安全性。

MySQL 支持 ANSI-SQL/92 标准里定义的全部 4 种事务级别（*READ UNCOMMITTED*、*READ COMMITTED*、*REPEATABLE READ*、*SERIALIZABLE*）（详见第 10 章）。

□ **数据行级锁定机制。** 在执行一个事务的时候，InnoDB 数据表的驱动程序使用的是它自己内建的数据行级锁定机制（不是 MySQL 提供的数据表级锁定机制）。也就是说，在事务过程中，数据表是不会被锁定的，其他用户仍可以访问它，被锁定的只是正在接受事务处理的数据记录（而 MyISAM 数据表在执行 *LOCK TABLE* 命令期间会被整个锁定）。如果有许多用户正在同时对一个大数据表进行修改，数据行级锁定机制将会大大提高人们的工作效率。

InnoDB 驱动程序能够自动识别“死锁”现象（两个进程各自占用着一项对方需要的资源，同时又在等待对方先释放所占用的资源，结果是谁也不能继续执行）并自动终止两个进程中的一个。

□ **外键约束条件。** 如果在数据表之间定义了关系，InnoDB 驱动程序将自动保证数据表的引用一致性在执行过 *DELETE* 命令之后也能保持。也就是说，不可能出现数据表 A 里的一条记录引用了数据表 B 里一条已经不复存在的记录的问题。用数据库的术语来讲，这一功能叫做外键约束条件。

□ **崩溃恢复。** 在发生崩溃后，InnoDB 数据表能够迅速地自动恢复到一个稳定可用的状态（前提是计算机的文件系统没有被破坏）。笔者没有对这个功能进行过测试。

InnoDB 数据表驱动程序从 MySQL 3.23.34 版本开始一直是 MySQL 的一个集成组件。这个驱动程序的研发工作和收费技术支持服务来自一家名为 InnoDB 的独立公司 (<http://www.innodb.com>)。

1. 问题和缺陷

如果 InnoDB 数据表只有优点、没有缺点，人们肯定会把 MyISAM 数据表扔进垃圾堆而全盘接受 InnoDB 数据表。但事实却并非如此。

□ **表空间的管理。** MyISAM 数据表驱动程序把每个数据表分别保存在它自己的文件里，这些文件会根据实际情况增大或缩小。InnoDB 数据表驱动程序却是把所有的数据和索引都保存在一个表空间（tablespace）里。表空间由一个或多个文件构成，它们形成了一个虚拟的文件系统。这些文件在被创建之后只能增大，不能缩小。如果想复制某个 InnoDB 数据表，把 MySQL 服务器停下来复制有关文件的办法是行不通的。因此，在管理 InnoDB 数据表时，*mysqldump* 命令的使用频率要比管理 MyISAM 数据表的时候高很多。

□ **数据记录的长度。** InnoDB 数据表中的单条数据记录最多可以占用 8000 个字节的空间。这一限制并不包括 *TEXT* 和 *BLOB* 数据列，它们只有前 512 个字节是随其他数据列一起存储在数据库里的，超过这个长度的数据将被存放在表空间的其他页面。

- **存储空间占用量。** InnoDB 数据表的空间占用量要比同样内容的 MyISAM 数据表大很多（最多时会有后者的两倍）。
- **全文索引。** InnoDB 数据表不支持全文索引（full-text index）。
- **GIS 数据。** InnoDB 数据表不能用来保存二维地理数据。
- **COUNT 问题。** 因为支持事务，InnoDB 数据表驱动程序在统计一个数据表里的记录个数时经常会遇到困难，所以在 InnoDB 数据表上执行 *SELECT COUNT(*) FROM TABLE* 命令的速度要比在 MyISAM 数据表上慢很多。这个问题应该尽快解决。
- **数据表锁定。** InnoDB 驱动程序在执行事务时使用的是它自己的锁定算法。因此，应该尽量避免使用 *LOCK TABLE...READ/WRITE* 命令。应该使用 *SELECT...IN SHARE MODE* 或 *SELECT...FOR UPDATE* 命令——它们将只锁定个别记录而不是锁定整个数据表。厂家已经计划在 MySQL 的未来版本里增加为 InnoDB 量身定做的 *LOCK TABLE...IN SHARE MODE* 和 *LOCK TABLE...IN EXCLUSIVE MODE* 命令。
- **mysql 数据表。** 用于管理 MySQL 访问权限的 mysql 数据表不能被转换为 InnoDB 数据表。它们必须是 MyISAM 格式。
- **许可证费用。** 在商用 MySQL 许可证里增加 InnoDB 支持将收取双倍的费用。（许可证费用只针对开发商业化软件产品的个人和公司。开源软件、内部使用软件和普通网站使用免费的 MySQL 版本就足够了。详见第 1 章。）

提示 InnoDB 数据表和 MyISAM 数据表在细节方面的优劣对比可以参见以下网址：

<http://dev.mysql.com/doc/mysql/en/innodb-restrictions.html>。

2. 选择 MyISAM 还是 InnoDB

可以把数据库里的不同数据表设置为不同的类型。也就是说，MyISAM 数据表和 InnoDB 数据表可以同时存在于同一个数据库里，这就使用户能够根据每一个数据表的内容数据和具体用途为它们分别选定最佳的数据表类型。

如果希望以最节约空间和时间的方式来管理数据表，MyISAM 数据表就应该是首选。从另一个方面讲，如果应用程序需要用到事务，需要更高的安全性，或者需要允许很多用户同时修改某个数据表里的数据，InnoDB 数据表就更值得考虑。

哪一种数据表类型的响应速度更快？这个问题是没有标准答案的。从理论上讲，事务操作需要花费更多的时间，InnoDB 数据表需要占用更多的硬盘空间，所以 MyISAM 似乎更有优势。但 InnoDB 数据表可以让用户避免使用 *LOCK TABLE* 命令，在某些特定的应用场合反而会是更优化的选择。

此外，系统的硬件配置（尤其的内存的大小）、MySQL 配置文件里的参数设置及其他一些因素对应用程序的速度也有着很大的影响。既然如此，在这里就只能送给大家这样一句忠告：如果速度对应用程序来说很关键，应该模拟实际应用情况在这两种数据表类型上进行过测试之后再做出选择。

8.2.3 HEAP 数据表

HEAP 数据表只存在于内存中（不是硬盘上）。它们使用了一个散列索引（hash index），所以数据记录的存取速度非常快。HEAP 数据表的主要用途是充当临时数据表；请参见 8.2.3 节对临时数据表的讨论。

与普通的数据表相比，HEAP 数据表在功能上受到了许多限制，其中最重要的有：不允许使用 *xxxTEXT* 和 *xxxBLOB* 数据类型；只允许使用 = 和 <=> 操作符来搜索记录（不允许使用 <、>、<= 或 => 操作符）；不支持 *AUTO_INCREMENT* 属性；只允许对 *NOT NULL* 数据列进行索引。

HEAP 数据表适用于数据量相对较小、但对访问速度要求很高的场合。请注意，因为 HEAP 数据表只存在于内存，所以一旦 MySQL 服务器停止运行，HEAP 数据表也就消失了。从这个意义上讲，HEAP 数据表是一种临时性的数据表。但它与特意使用 *CREATE TEMPORARY TABLE* 命令或是 MySQL 为了保存中间结果而临时创建的数据表是有区别的：HEAP 数据表对于来访问同一个数据库的其他 MySQL 连接是可见的，在连接意外中断时也不丢失¹。HEAP 数据表的最大长度由 MySQL 配置文件里的 *max_heap_table_size* 参数决定。

8.2.4 临时数据表

这里所说的临时数据表指的是特意使用 *CREATE TEMPORARY TABLE* 命令或是 MySQL 为了保存中间结果而临时创建的数据表²，其数据表类型可以是我们前面介绍的任何一种。这种数据表在 MySQL 服务器意外掉电时不一定会丢失，但在 MySQL 服务器正常关机、本次 MySQL 连接正常结束或意外中断时都将全部丢失！此外，这种数据表对于访问同一个数据库的其 MySQL 连接是不可见的，两个不同的用户可以在同一个数据库里使用相同的名字创建临时数据库而不会发生冲突。（注意与 HEAP 数据表的区别。）

临时数据表并不是一种新的数据表类型，它们的数据表类型可以是前面介绍的任何一种，只不过在用途上是临时的而已。需要特意使用 *CREATE TEMPORARY TABLE* 命令去创建一个临时数据表的场合并不多，人们在提到“临时数据表”时更多地是指 MySQL 为了保存 *SELECT* 查询的中间结果而自动创建的数据表。

临时数据表与其他 MySQL 数据表是分开保存的，MySQL 会把它们存放到一个临时子目录里去。这个临时子目录在 Windows 环境下通常是 C:\Windows\Temp，在 UNIX/Linux 环境下通常是 /tmp 或 /var/tmp 或 /usr/tmp；这个子目录可以在 MySQL 服务器启动时设置。

8.2.5 其他的数据表类型

MySQL 还支持其他一些数据表类型，下面列出了其中最重要的几种。这些数据表类型只在 MySQL 的 Max 版本或者是由用户自行编译的版本里才有。如果想知道自己的 MySQL 版本都支持哪几种数据表类型，可以用 *SHOW ENGINES* 命令来查看。

□ **BDB 数据表。** BDB 数据表是最早具备事务支持能力的 MySQL 数据表类型。但随着 InnoDB 数据表驱动程序的日益成熟，BDB 数据表已经没有什么用武之地了。

□ **ARCHIVE 数据表（压缩数据表，始见于 MySQL 4.1）。** 这种数据表类型是为了保存和备份海量数据而设计的。这种数据表类型的优点是在保存数据记录之前会先对数据记录进行压缩。*ARCHIVE* 数据表只适合用来保存不再需要修改的数据记录（它允许用户执行 *INSERTR* 命令，但不允许执行 *UPDATE* 和 *DELETE* 命令）。

ARCHIVE 数据表不能建立索引，所以每执行一条 *SELECT* 命令，就必须读取一遍全部的数据记录。因此，这种数据表类型仅适用于数据访问量非常少的场合。

1. 原文作者在这里漏掉了一些非常关键的话，译者给加上了。——译者注

2. 原文作者在这里漏掉了一些非常关键的话，这一句是译者给加上了。——译者注

- **CSV 数据表 (文本格式的数据表, 始见于 MySQL 4.1)**。CSV 数据表里的记录都保存在文本文件里, 数据之间用逗号隔开, 如 ““123”,*I am a character string*”。CSV 数据表不能建立索引。
- **NDB 数据表 (MySQL 集簇, 始见于 MySQL 4.1)**。NDB 数据表类型是集成在 MySQL Max 版本里的 MySQL 集簇功能中的一种。NDB 的意思是 *network database* (网络数据库)。这种数据表类型支持事务, 最适合用来建设数据分布在大量计算机上的网络数据库。使用这种数据表类型的前提是必须有多台安装了 MySQL Max 版本的联网计算机并配置使它们支持集簇操作。这方面的细节信息可以参见 <http://dev.mysql.com/doc/mysql/en/ndbcluster.html>。
- **FEDERATED 数据表 (外部数据表, 始见于 MySQL 5.0)**。这种数据表类型能够让用户去访问外部数据库里的数据表, 而那个数据库系统可以位于本地网络中的另一台计算机上。就目前而言, 外部数据库必须也是一个 MySQL 数据库, 但未来的 MySQL 版本可能会允许使用这种数据表类型与其他品牌的数据库系统建立连接。
FEDERATED 数据表类型还有许多值得完善的地方: FEDERATED 数据表上的事务和查询都无法用 Query Cache 工具优化; 不能对外部数据表的结构进行修改 (但数据记录可以)。换句话说, 不能对 FEDERATED 数据表执行 *ALTER TABLE* 命令, 但可以执行 *INSERT*、*UPDATE* 和 *DELETE* 命令。

MySQL 文档里有一章是专门介绍各种 MySQL 数据表类型的, 可以在以下网址找到它: <http://dev.mysql.com/doc/mysql/en/storage-engines.html>。

8.2.6 数据表文件

可以在启动 MySQL 服务器时为数据库文件指定一个存放位置 (这个位置在 UNIX/Linux 环境下通常是 /var/lib/mysql 子目录, 在 Windows 环境下通常是 C:\Programs\MySQL\MySQL Server n.n\data 子目录)。下面给出的文件路径都是相对于这个子目录而言的。

每个数据表都有一个*.frm 定义文件, 同一个数据库的*.frm 文件统一存放在以这个数据库名字命名的子目录里: data/dbname/tablename.frm。这个文件的内容是数据表的结构定义 (数据列的名字、数据类型等)。

从 MySQL 4.1 版本开始, MySQL 在每一个数据库子目录里增加了一个与整个数据库有关的 db.opt 文件: data/dbname/db.opt。这个文件的内容是整个数据库的结构定义和设置。

MySQL 还将为每个 MyISAM 数据表创建两个文件: 一个是 data/dbname/tablename.MYD 文件, 用来存放 MyISAM 数据表的数据; 另一个是 data/dbname/tablename.MYI, 用来存放 MyISAM 索引 (数据表的全部索引)。

根据 MySQL 配置文件中 innodb_file_per_table 选项的设置情况, InnoDB 数据表既可以各自存为一个文件, 也可以统一存放在一个所谓的表空间 (tablespace) 里。表空间的存放位置和名字由配置设置决定。MySQL 现在的默认安排是把 InnoDB 数据表的数据和索引存放在 data/dbname/tablename.ibd 文件里, 把表空间和撤销日志 (undo log) 存放在 data/ibdata1、-2、-3 等文件里, 把 InnoDB 日志数据存放在 data/ib_logfile0、-1、-2 等文件里。

如果用户还为数据表定义了触发器 (trigger, 详见第 13 章), MySQL 现在的做法是把它们的代码存放在 data/dbname/tablename.TRG 文件里, 但这个路径在未来的 MySQL 版本里可能会发生变化。

8.3 MySQL 数据类型

每个数据表至少会有一个数据列，而用户必须为每个数据列分别定义一个适当的数据类型。在一节里，将对 MySQL 所支持的数据类型做一个概括的介绍。

8.3.1 整数 (xxxINT)

在默认的情况下，*INT* 数据类型既包括正数，也包括负数。但如果给 *INT* 数据列定义了 *UNSIGNED* 属性，它的取值范围将仅限于正数。这里要特别提醒大家注意：对 *UNSIGNED* 数据列做减法计算的返回值仍将是一个 *UNSIGNED* 整数，而这可能导致虚假或让人困惑的结果。

TINYINT 数据类型的取值范围是 -128~+127。如果使用了 *UNSIGNED* 属性，它的取值范围就将变成 0~255。如果用户试图存入的数据值超出了数据类型的取值范围，MySQL 就会简单地把它替换为最大可取值或最小可取值，具体情况如表 8-1 所示。

表 8-1 MySQL 支持的整数数据类型

MySQL 数据类型	含 义
<i>TINYINT(m)</i>	8 位整数（1 个字节， -128~+127）；可选参数 <i>m</i> 给出的是 <i>SELECT</i> 查询结果中的数据列宽度（Maximum Display Width，最大显示宽度），对取值范围没有影响
<i>SMALLINT(m)</i>	16 位整数（2 个字节，从 -32 768~+32 767）
<i>MEDIUMINT(m)</i>	24 位整数（3 个字节，从 -8 388 608~+8 388 607）
<i>INT(m)、INTEGER(m)</i>	32 位整数（4 个字节， -2 147 483 648~+2 147 483 647）
<i>BIGINT(m)</i>	64 位整数（8 个字节， $\pm 9.22 \times 10^{18}$ ）
<i>SERIAL</i>	<i>BIGINT AUTO_INCREMENT NOT NULL PRIMARY KEY</i> 的简写

在一个整数字段的定义里，可以选择为它定义一个数据列宽度，比如 *INT(4)*。这个参数在书里一般写为 *M* 参数（Maximum Display Width，最大显示宽度），它可以帮助 MySQL 和各种用户操作界面把查询结果以一种整齐易读的格式显示。

注解 对于 *INT* 数据类型，*M* 参数既不影响它的取值范围，也不影响数据值的位数。如 *INT(4)* 这样的定义并不影响把大于 9999 的数值存入数据列。不过，在一些很少见的特定场合里（例如，当 MySQL 在执行一些需要借助于临时数据表才能完成的复杂查询时），临时数据表里的数值有可能会被截短并导致最终结果不正确。

1. *AUTO_INCREMENT* 整数

如果给某个数据表中的一个整数数据列定义可选的 *AUTO_INCREMENT* 属性，那么当用户向这个数据表插入一条新记录时，MySQL 就会自动地把这个整数数据列的当前最大取值加上 1 之后赋值给新记录中的这个整数字段。*AUTO_INCREMENT* 属性的常见用法是定义数据表的主键字段。

使用 *AUTO_INCREMENT* 属性需要注意以下几个问题：

- 这个属性必须与 *NOT NULL*、*PRIMARY KEY* 或者 *UNIQUE* 属性同时使用。
- 每个数据表最多只能有一个 *AUTO_INCREMENT* 数据列。
- MySQL 的这种 ID 值自动生成机制只在用户使用 *INSERT* 命令插入新记录、并且没有为 ID 字段明确地给出一个值或 *NULL* 时才起作用。如果用户给出了一个具体的值并且这个值还没有

在 ID 列里出现过, MySQL 就将使用这个 ID 值生成一条新数据记录。

- 如果想知道 MySQL 在刚插入数据记录里生成的 *AUTO_INCREMENT* 值是多少, 在执行完 *INSERT* 命令之后 (但还是在本次连接或本个事务里), 立刻执行 *SELECT LAST_INSERT_ID()* 命令。
- 如果 *AUTO_INCREMENT* 计数器到达了它的最大值 (不同的整数数据类型有不同的最大值), 将不再继续递增, 数据记录的插入操作也将随之无法继续进行。对 *INSERT* 和 *DELETE* 操作非常频繁的数据表来说, 哪怕数据记录远少于 20 亿条, 也有可能出现 32 位整数类型的 *AUTO_INCREMENT* 计数器被耗尽的情况。在定义这种数据表的时候, 最好使用 *BIGINT* 数据列。

2. 二进制数据 (*BIT*和*BOOL*)

MySQL 关键字 *BOOL* 是 *TINYINT* 的同义词。MySQL 5.0.2 及以前的版本里的 *BIT* 也是如此。从 5.0.3 版开始, *BIT* 变成了一种可以存储多达 64 位二进制数值的新数据类型。

3. 浮点数 (*FLOAT*和*DOUBLE*)

从 MySQL 3.23 版开始, *FLOAT* 和 *DOUBLE* 类型一直分别对应着 IEEE 标准所定义的单精度浮点数和双精度浮点数, 绝大多数程序设计语言都支持这两种数据类型。

在显示/打印时, *FLOAT* 和 *DOUBLE* 数值的数字个数可以用 *m* 和 *d* 两个可选参数来设置, 其中 *m* 是十进制数字的总个数, *d* 是小数点后面的数字个数。表 8-2 对浮点数据类型进行了总结。

表 8-2 MySQL 支持的浮点数数据类型

MySQL 数据类型	含 义
<i>FLOAT(m, d)</i>	单精度浮点数, 8 位精度 (4 字节)。参数 <i>m</i> 和 <i>d</i> 是可选的。 <i>m</i> 是十进制数字的总个数, <i>d</i> 是小数点后面的数字个数。在插入数据的时候, 数值将做必要的舍入。超出范围的数值将被替换为最大可取值。可以用 <i>SHOW WARNING</i> 命令来查看警告内容
<i>DOUBLE(m, d)</i>	双精度浮点数, 16 位精度 (8 字节)
<i>REAL(m, d)</i>	<i>DOUBLE</i> 的同义词

参数 *m* 只影响数值的显示效果, 对数值的精确度没有影响。参数 *d* 则不同, 小数点后面的数字需要按照它来舍入。比如说, 如果试图把 123456.789877 保存到一个 *DOUBLE(6, 3)* 数据列里, 实际存放的将会是 123456.790。

注解 MySQL 要求浮点数必须使用通用计数法写出, 即小数点必须是一个句点, 不能是逗号 (有部分欧洲国家使用逗号作为小数点)。MySQL 服务器永远以这种写法向客户返回查询结果。非常大或非常小的数值将使用科学计数法写出 (如 1.2345678901279e+017)。

如果想让浮点数显示为其他格式, 有两种办法: 一是在 SQL 查询里使用 *FORMAT* 函数; 二是用某种客户程序设计语言 (PHP、Perl 等) 编写一个脚本来改变浮点数的显示效果。

8.3.2 定点数 (*DECIMAL*)

MySQL 在把数据保存为 *FLOAT* 和 *DOUBLE* 数据类型时会自动进行必要的舍入, 如果因此而带来的误差不可接受 (比如在处理财务数据的时候), 就应该选用整数类型 *DECIMAL*。*DECIMAL* 数据类型以字符串的形式来保存数据, 并且不允许使用指数形式, 所以将占用更多的空间, 可以表示的数据范围也比较小。表 8-3 对 MySQL 支持的定点数据类型进行了总结。

表 8-3 MySQL 支持的定点数数据类型

MySQL 数据类型	含 义
<i>DECIMAL(p, s)</i>	定点数，以字符串形式保存：数字个数不限（每位数字占用一个字节，再加上 2 个字节的开销）
<i>NUMERIC、DEC</i>	双精度浮点数，16 位精度（8 字节）

参数 *p* 和 *s* 分别设定了数据值的数字总个数（表示范围，最大值是 65）和小数点后面的数字个数（精确度，最大值是 30）。比如说，*DECIMAL(6, 3)* 的可表示范围是 -999.999~999.999。MySQL 在内部把定点数保存为二进制格式：先把定点数分成小数点前面和小数点后面两个部分，并为它们各自分配 4 个字节，这 4 个字节最多可以表示 9 位数字。也就是说，*DECIMAL(6, 3)* 和 *DECIMAL(18, 9)* 都将实际占用 8 个字节。如果还有多出来的数字，就再分配 1 个字节来存储它们（每个字节容纳 2 位数字，但每 4 个字节容纳 9 位数字）。这么计算下来，*DECIMAL(19, 9)* 将实际占用 9 个字节。

8.3.3 日期与时间 (*DATE*、*TIME*、*DATETIME*、*TIMESTAMP*)

表 8-4 对 MySQL 用来存储时间值的数据类型进行了总结。

表 8-4 MySQL 支持的日期和时间数据类型

MySQL 数据类型	含 义
<i>DATE</i>	'2003-12-31' 格式的日期值，取值范围：1000-01-01~9999-12-31（3 个字节）
<i>TIME</i>	'23:59:59' 格式的时间值，取值范围：±838:59:59（3 个字节）
<i>DATETIME</i>	'2003-12-31 23:59:59' 格式的 <i>DATE</i> 加 <i>TIME</i> 组合
<i>YEAR</i>	年份，取值范围：100~2155（1 个字节）

1. 日期/时间数据的合法性检查

在较早的 MySQL 版本里，*DATE* 和 *DATETIME* 数据类型只进行很少的类型检查：月份值允许是 0~12 之间的任意数字，日期值允许是 0~31 之间的任意数字，但保证数据正确无误是客户端程序的责任。（比如说，为了存储不完整或不确定的日期，允许客户端程序提供 0 作为月份值或日期值）。

从 5.02 版开始，MySQL 对日期和时间数据的合法性检查变得严格了——只有它认为合法的数据才能进入数据库，但仍允许使用 0 作为月份值或日期值，如 '0000-00-00'。

对日期和时间数据进行的合法性检查由 MySQL 系统变量 *sql_mode* 控制（详见第 14 章），表 8-5 对这个变量的设置值与合法性检查严格程度的关系进行了总结。

表 8-5 *sql_mode* 设置

设置值	含 义
<i>ALLOW_INVALID_DATES</i>	允许使用显然不正确的数据作为日期/时间值（如 '2005-02-31'）
<i>NO_ZERO_DATE</i>	'0000-00-00' 不再是一个合法的日期时间值
<i>NO_ZERO_IN_DATE</i>	不允许使用 0 作为月份值或日期值

2. *TIMESTAMP* 的特点

在各种日期/时间数据类型当中，*TIMESTAMP* 是一个非常特殊的角色。这个类型的字段会在数据记录的其他字段被修改时自动刷新，从效果上看，这个字段里的日期/时间值其实就是数据记录的最后一次修改时间。由于这一特点，*TIMESTAMP* 类型的字段几乎总是被用于内部管理用途而不是被用来

存储“真正的”数据——虽然那是不可能。

有许多需要与客户端程序或函数库（比如 Connector/ODBC）进行交互的数据库操作，只有在数据库里的每一个数据表都有一个 *TIMESTAMP* 数据列的时候才能正常工作。MySQL 在对数据进行内部管理的时候也经常需要知道它们的最后一次修改时间。

如果想让 MySQL 自动刷新 *TIMESTAMP* 字段，就不能在修改数据记录时给这个字段设置一个具体的值或 *NULL*，这样 MySQL 才会把当前时间插入到这个字段里。

如果同一个数据表里有一个以上的 *TIMESTAMP* 数据列，MySQL 将自动刷新它们当中的第一个，但如果用户在修改数据记录时明确地为第一个 *TIMESTAMP* 数据列给出了一个具体的值，则顺延到第二个；以此类推。

从 MySQL 4.1.3 开始，*TIMESTAMP* 数据列增加了两个属性，它们可以让用户更好地调控 MySQL 对的 *TIMESTAMP* 数据列的刷新行为。表 8-6 对这两个属性的组合效果进行了总结。

表 8-6 *TIMESTAMP* 的变体

设 置	含 义
<i>TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP</i>	在创建新记录和修改现有记录时都对这个数据列进行刷新
<i>TIMESTAMP</i>	同上
<i>TIMESTAMP DEFAULT CURRENT_TIMESTAMP</i>	在创建新记录时把这个字段设置为当前时间，但以后修改时不再刷新它
<i>TIMESTAMP ON UPDATE CURRENT_TIMESTAMP</i>	在创建新记录时把这个字段设置为 0，以后修改时刷新它
<i>TIMESTAMP DEFAULT 'yyyy-mm-dd hh:mm:ss' ON UPDATE CURRENT_TIMESTAMP</i>	在创建新记录时把这个字段设置为给定值，以后修改时刷新它

注意 不要选用 *TIMESTAMP* 数据列来保存“真正的”日期/时间数据，那么做的最佳选择是 *DATETIME* 数据类型。

如果偶尔在修改某条记录的时候不想让 *TIMESTAMP* 数据列自动刷新，就必须像下面这样明确地给出一个日期/时间值：

```
UPDATE tablename SET col='new value, ts=ts;
```

在 MySQL 4.0 及更早的版本里，*TIMESTAMP* 值的格式是 *YYYYMMDDHHMMSS* 而不是像现在这样的 *YYYY-MM-DD HH:MM:SS*。这种格式的 *TIMESTAMP* 数据在处理时可能会导致兼容问题。

如果更喜欢这种老格式，就需要在有关命令里加上一个 0（就像这样： *SELECT ts+0 FROM table;*）。

□ **微妙。**在未来 MySQL 版本里，*TIMESTAMP* 数据类型可能会增加一个微妙部分。其实相应的语法早就定义出来了 (*2005-12-31 23:59:59.nnnnnnn*)，用来处理这种数据的 *MICROSECOND* 函数也有了不少种，但现有的 MySQL 版本（包括 MySQL 5.0.2 在内）还不能存储这种数据。

3. 日期/时间数据的处理和排版

MySQL 在返回一个日期值时使用的是 '2005-12-31' 格式，但 *INSERT* 和 *UPDATE* 命令还可以接受其他一些格式——只要有关数据符合“年/月/日”的顺序并且是一个数值即可。对于以两位数字给出的年份值，MySQL 将做出如下解释：70~99 代表着 1970 年~1999 年，00~69 代表着 2000 年~2069 年。

如果需要对查询结果进行排版，有几个 MySQL 函数可以用来处理日期/时间值的格式。在这些函数当中，以 *DATE_FORMAT()* 的用法最为灵活，下面是它的一个用法示例：

```
SELECT DATE_FORMAT(birthdate, '%Y %M %e') FROM students
1977 September 3
1981 October 25
...
```

提示 第 10 章将对 *DATE_FORMAT()* 及其他一些 MySQL 用来处理日期/时间数据的函数进行介绍，大家将在那里学习到如何根据不同的地理时区对时间值进行转换。与日期/时间有关的所有函数将被汇总在本书的第 21 章。

8.3.4 字符串 (*CHAR*、*VARCHAR*、*xxxTEXT*)

表 8-7 对 MySQL 用来存储字符串值的数据类型进行了总结。

CHAR 类型的字符串其长度是有严格限制的。不管字符串的实际长度是多少，*CHAR(20)* 字段在每条记录里都将占用 20 个字节。（字符串的前导空格在存储前将被去掉。比较短的字符串在尾部加空格补足——MySQL 在读出数据时会自动删除这些空格。但这么做的后果是 MySQL 数据库不能保存尾部确实有空格字符的字符串。）

表 8-7 MySQL 支持的字符串数据类型

MySQL 数据类型	含 义
<i>CHAR(n)</i>	固定长度的字符串，最多 255 个字符
<i>VARCHAR(n)</i>	可变长度的字符串，最多 255 个字符（MySQL 4.1 及以前： <i>n</i> <256；MySQL 5.0.3 及以后： <i>n</i> <65 535）
<i>TINYTEXT</i>	可变长度的字符串，最多 255 个字符
<i>TEXT</i>	可变长度的字符串，最多 $(2^{16}-1)$ 个字符
<i>MEDIUMTEXT</i>	可变长度的字符串，最多 $(2^{24}-1)$ 个字符
<i>LONGTEXT</i>	可变长度的字符串，最多 $(2^{32}-1)$ 个字符

VARCHAR 和 *xxxTEXT* 类型的字符串其长度是可变的，它们占用的存储空间由它们的实际长度决定。

VARCHAR 和 *TINYTEXT* 类型看起来似乎一样——两个都是可以存储多达 65 535 个字符的字符串数据类型，但在一些细节方面却是有区别的：*VARCHAR* 数据列的最大字符长度（0~65 535 之间的某个整数）必须在声明数据表时设置，超长的字符串将在存储前被截短；*xxxTEXT* 数据列根本不允许为它设置一个最大长度（唯一的限制是特定文本类型本身的最大长度）。

注意 在创建新数据表的时候，MySQL 经常会把数据列的定义改成一种对 MySQL 来说更有效率的形式（详见第 9 章）。MySQL 文档把这种自动方式的修改称为默许的数据列修改（silent column changes），它们对 *CHAR* 和 *VARCHAR* 数据列都有影响：

- 如果 *n*<4，*VARCHAR(n)* 将被修改为 *CHAR(n)*；
- 如果 *n*>3 并且在同一个数据表里还有其他的 *VARCHAR*、*TEXT* 或 *BLOB* 数据列，*CHAR(n)* 将被修改为 *VARCHAR(n)*。如果数据表里只有固定长度的数据列，*CHAR(n)* 将不发生变化。

□ **VARCHAR 的新特性。** MySQL 5.0 对 *VARCHAR* 数据类型做了两项重大改进。（在笔者测试的

MySQL 5.0.3 版本里, *VARCHAR* 数据类型的新特性只在 MyISAM 数据表里起作用, 在 InnoDB 数据表里则不会。)

- 在 MyISAM 数据表里, *VARCHAR* 数据列的最大长度现在是 65 535 个字节 (以前是 255 个字节), 但最大字符个数还要取决于具体的字符集——许多字符集要用一个以上的字节来表示一个字符。
- *VARCHAR* 值的前导空格和尾部空格现在可以存入数据表了, 即 *INSERT INTO tablename (varcharcolumn) VALUES ('abc')* 命令现在将把 'abc' (注意, 有一个前导空格和一个尾部空格) 保存到数据列里。(过去, MySQL 会把 *VARCHAR* 值的尾部空格去掉——这违反了 ANSI 标准中的有关规定。)
- **BINARY 属性。**如果给 *CHAR* 和 *VARCHAR* 类型的数据列加上可选的 *BINARY* 属性, MySQL 就会把它们视同 *BLOB* 数据列来处理。*BINARY* 属性给用户带来的好处是: MySQL 在排序时将把字符视为二进制数 (而不是字母), 而这样就可以把字母的大小写形式区分开。这在没有引入 *BINARY* 属性时是做不到的。与千变万化的正宗 (*garden-Variety*) 字符串相比, 二进制字符串的内部管理工作更简单, 需要花费的时间也更短。

1. 与字符集有关的基本概念

所有的文本数据列都允许使用 *CHARACTER SET charactersetname COLLATE sortorder* 属性来指定一种字符集和排序方式。字符集回答了使用什么样的编码来表示各种字符的问题。128 个英文 ASCII 字符在绝大多数字符集里的编码都是一致的 (如字母 A 的编码是 65), 问题主要集中在如何表示各国语言中的特有字符方面。

□ **Latin 字符集。**在过去, 几乎每一种主要的人类语言都有它自己的单字节字符集。随着时间的推移, 有几种 Latin 字符集得到了最广泛的应用, 它们是: 对应于 ISO-8859-1 标准的 *Latin1* 字符集, 收录了西欧国家常用的所有字符 (äöüßåàå等); 对应于 ISO-8859-2 标准的 *Latin2*, 收录了来自中欧和东欧语言的字符; 对应于 ISO-8859-15 标准的 *Latin0* (也叫 *Latin9*) 字符集, 与 *Latin1* 完全一样, 只是增加了一个欧元符号。

这些字符集都存在着这样一个明显的问题: 它们当中没有一个能把所有欧洲语言里的所有字符都收录齐全, 即没有一种 *Latin* 字符集能够适用于整个欧洲。

□ **Unicode 字符集。**为了解决这个问题, 人们开发出了双字节的 Unicode 字符集。两个字节意味着多达 65 535 个字符, 所以 Unicode 字符集不仅覆盖了所有的欧洲语言, 还覆盖了绝大多数亚洲语言。

可是, Unicode 标准只定义了每个字符的编码, 没有定义如何存储这些编码。因此, Unicode 字符集有好几种变体, 其中最重要的是 UCS-2 (universal character set, 通用字符集) 和 UTF-8 (Unicode transfer format, Unicode 传输格式)。

■ UCS-2 (也叫 UTF-16) 字符集代表着一种最简单的解决方案: 每种字符都用两个字节 (16 位) 来表示。人们把这种格式称为 UTF-16 或 UCS-2 格式。Microsoft Windows 的操作系统函数几乎都使用这种表示法。

可是, 这种表示法有两大弊端: 其一, 字符串的空间占用量一下子翻了一倍, 即使字符串里只有欧洲字符、甚至是只有英语 ASCII 字符也是如此; 其二, 字节编码 0 会出现在 Unicode 字符串里的许多地方 (比如说, 在使用英文 ASCII 字符写出来的文本里, 每一个第 2 字节都是 0), 而许多 C 语言程序、电子邮件服务器和其他软件都把 0 字节视为字符串的结束标志。

■ UTF-8 是最流行的 UTF-16 替代品。在这里, 所有的 ASCII 字符 (7 位) 仍像以前那样用第

1位是0的单个字节来表示，所有其他的Unicode字符则用2~4个字节来表示。

这种格式的最大缺点是同一份文档的字节数和字符数没有任何显而易见的关系。但因为它与现有的程序高度兼容和其他一些优点，UTF-8已经成为了一种事实上的标准，UNIX/Linux软件和绝大多数重要的Web应用软件现在都把它作为自己的默认字符集。如果没有特别说明，本书里提到Unicode的时候所指的都是UTF-8格式。

虽然Unicode(UTF8和UTF-16)有很多优点，但现在就全盘接受它似乎还为时过早，这是因为以下两个原因：其一，Unicode字符集与许多常用的单字节字符集不兼容；其二，Web开发工具对Unicode的支持还远未达到完美的境地（这里最薄弱的环节是PHP，而且这种情况在PHP 5.2版推出之前似乎也不会有什么改善）。

2. MySQL的字符集支持能力

在MySQL 4.0及以前的版本里，各种文本字段的字符集和排序方式都由MySQL服务器决定。MySQL服务器的默认设置是使用Latin1字符集和瑞典排序方式。虽然允许在MySQL配置文件里另外指定一种字符集和排序方式，会导致所有的数据库都将受到影响。还有，对字符集和排序方式做出的任何修改都需要重新启动数据库服务器才能生效，启动后还必须重新创建所有的索引。最后，MySQL 4.0及以前的版本根本不支持Unicode字符集。

从MySQL 4.1开始，上述问题得到了极大的改善：现在可以为每一个数据列分别指定一种字符集和排序方式。与此同时，字符集和排序方式的选择范围也比以前大多了，这包括Unicode UTF-8和UCS-2在内。

字符集和相关排序方式的完整清单可以通过执行SQL命令SHOW COLLATION获得，下面是这个命令执行结果的一部分。（请注意，字符集与排序方式不能随意组合。一种字符集只能与一组特定的排序方式配合使用。在下面的清单里，每种字符集的默认排序方式由Default列里的Yes标明。）

Collation	Charset	Id	Default	Compiled	Sortlen
ascii_bin	ascii	65			0
ascii_general_ci	ascii	11	Yes		0
binary	binary	63	Yes	Yes	1
latin1_bin	latin1	47		Yes	1
latin1_danish_ci	latin1	15			0
latin1_general_ci	latin1	48			0
latin1_general_cs	latin1	49			0
latin1_german1_ci	latin1	5			0
latin1_german2_ci	latin1	31		Yes	2
latin1_spanish_ci	latin1	94			0
latin1_swedish_ci	latin1	8	Yes	Yes	1
latin2_bin	latin2	77			0
latin2_croatian_ci	latin2	27			0
latin2_general_ci	latin2	9	Yes		0
latin2_hungarian_ci	latin2	21			0
latin5_bin	latin5	78			0
latin5_turkish_ci	latin5	30	Yes		0
latin7_bin	latin7	79			0
latin7_estonian_cs	latin7	20			0
latin7_general_ci	latin7	41	Yes		0
latin7_general_cs	latin7	42			0
utf8_bin	utf8	83		Yes	1
utf8_czech_ci	utf8	202		Yes	8
utf8_danish_ci	utf8	203		Yes	8
utf8_estonian_ci	utf8	198		Yes	8
utf8_general_ci	utf8	33	Yes	Yes	1
utf8_icelandic_ci	utf8	193		Yes	8

utf8_latvian_ci	utf8	194	Yes	8
utf8_lithuanian_ci	utf8	204	Yes	8
utf8_persian_ci	utf8	208	Yes	8
utf8_polish_ci	utf8	197	Yes	8
utf8_roman_ci	utf8	207	Yes	8
utf8_romanian_ci	utf8	195	Yes	8
utf8_slovak_ci	utf8	205	Yes	8
utf8_slovenian_ci	utf8	196	Yes	8
utf8_spanish2_ci	utf8	206	Yes	8
utf8_spanish_ci	utf8	199	Yes	8
utf8_swedish_ci	utf8	200	Yes	8
utf8_turkish_ci	utf8	201	Yes	8
utf8_unicode_ci	utf8	192	Yes	8
...				

表 8-8 对几种最为重要的字符集和排序方式组合进行了说明。

表 8-8 几种最重要的字符集和排序方式

字符集	排序方式	说 明
<i>latin1</i>	<i>latin1_swedish_ci</i>	瑞典排序方式，在 MySQL 里是 <i>latin1</i> 数据列的默认排序方式； <i>ci</i> 的意思是不区分字母的大小写形式
<i>latin1</i>	<i>latin1_general_ci</i>	通用排序方式，适用于许多种西欧语言，各国的特有字母不考虑在内
<i>latin1</i>	<i>latin1_general_cs</i>	同上，但区分字母的大小写形式：大写字母排在小写字母的前面
<i>latin1</i>	<i>latin1_german1_ci</i>	DIN-1 标准下的德国排序方式（ä=a、ö=o、ü=u、ß=s）
<i>latin1</i>	<i>latin1_german2_ci</i>	DIN-2 标准下的德国排序方式（ä=ae、ö=oe、ü=ue、ß=ss）
<i>utf8</i>	<i>utf8_general_ci</i>	通用排序方式，适用于许多种西欧语言，各国的特有字母不考虑在内；在 MySQL 里是 <i>utf8</i> 数据列的默认排序方式

如果没有为某个数据列指定一个字符集和排序方式，它将使用数据表、数据库或 MySQL 服务器的默认字符集——这取决于哪一个级别上有默认设置。

至于哪一种字符集是数据库的最佳选择，必须根据具体的应用来决定。如果根本用不着西欧语言以外的字符，那就应该选择 *latin1* 字符集。这种字符集在以后的数据处理工作中通常不会有任何问题。

提示 本章还有许多地方谈到了字符集和排序方式，请参见在介绍 SQL 语法、SQL 命令、MySQL 变量和 MySQL 系统配置的有关章节里在这方面的讨论。此外，书中介绍程序设计语言和技术的各章内容还有围绕如何处理 Unicode 字符串这个主题而展开的专题讨论。大家可以通过书后索引中的 Unicode 条目去查找那些章节。

8.3.5 二进制数据 (xxxBLOB 和 BIT)

如果想在数据库里储存二进制数据，*BLOB* (binary large object, 二进制大对象) 数据类型将是最佳选择。*BLOB* 与 *TEXT* 数据类型有着几乎完全一样的属性，唯一的区别是文本数据通常是按照文本模式进行比较和排序的，而二进制数据则是根据它们的二进制编码进行比较和排序的。

是否应该在数据库里存放二进制数据一直是个容易引起争论的问题。有不少人认为应该把二进制数据（如图像）保存为外部文件，数据库只用来保存这些文件的链接。

使用 *BLOB* 数据列的好处是提高了数据库里的数据集成程度（提高了安全性、简化了备份工作、

能够以统一的形式访问所有数据), 缺点是数据库的响应速度往往会大大降低。把短小的数据(字符串、整数等)与长的数据(*BLOB* 和长文本)混合存放在同一个数据表里是一种非常不好的做法, 因为这会导致所有数据记录的存取速度变慢。

此外, *BLOB* 数据在正常操作情况下只能作为一个整体来读出。也就是说, 如果某个 *BLOB* 数据的长度是 800KB, 想直接读取它的最后 100KB 是不可能的。*BLOB* 数据必须以一个整体来读写和传输。

表 8-9 对 MySQL 所支持的二进制数据类型进行了总结。

表 8-9 MySQL 支持的二进制数据类型

MySQL 数据类型	含 义
<i>BIT(n)</i>	二进制数据, <i>n</i> 是二进制位的个数(最大可取值是 64)
<i>TINYBLOB</i>	可变长度二进制数据, 最多 255 个字节
<i>BLOB</i>	可变长度二进制数据, 最多 $(2^{16}-1)$ 个字节
<i>MEDIUMBLOB</i>	可变长度二进制数据, 最多 $(2^{24}-1)$ 个字节
<i>LONGBLOB</i>	可变长度二进制数据, 最多 $(2^{32}-1)$ 个字节

1. *BIT* 数据类型

从 MySQL 5.0.3 开始, *BIT* 数据列的最大宽度达到了 64。*BIT* 数据类型并不是新生事物, 但在较早的 MySQL 版本里, 它只是 *TINYINT(1)* 的一个同义词。

MySQL 5.0.3 还增加了一条用来写出二进制位数据的新语法 *b'0101'*。*SELECT* 查询在遇到 *BIT* 数据列的时候将返回二进制数据, 但笔者测试的客户端程序无法把这些值正确地显示出来。如有必要, 先使用 *SELECT bitcolumn+0* 命令把二进制值转换为整数, 再使用 *SELECT BIN(bitcolumn+0)* 命令把这些整数显示为二进制值。

如果插入 *BIT* 数据列的数值引起了下溢出或上溢出, 所有的二进制位都将被设置为 1。例如, 当把-1、0、1、7、8 依次插入一个 *BIT(3)* 数据列的时候, 实际存放的二进制值将是: *b'111'*、*b'000'*、*b'001'*、*b'111'* 和 *b'111'*。

2. 其他数据类型

ENUM 和 *SET* 数据类型是 MySQL 的一个特色。它们可以帮助 MySQL 服务器以一种非常有效率的方式对涉及字符(串)集合¹的排列组合问题进行处理。

ENUM 数据类型定义的是一个字符串集合, 它的成员最多可以有 65 535 个。数据表中的 *ENUM* 字段的取值只能是这个集合中的某一个成员(不允许是不同成员的一个组合); 相当于数学意义上的“排列”。

SET 数据类型采用了类似的思路, 但允许数据表中的 *SET* 字段的取值是集合成员的任意组合(但数量不得超过 64 个); 相当于数学意义上的“组合”。在内部, 这些字符串分别与 2 的幂(1、2、4、8 等)相对应, 所以字符串的组合就相当于二进制位的组合。因为每个字符串分别对应一个二进制位, 所以 *SET* 数据类型的空间占用量比 *ENUM* 的大。最多可以有 64 个字符串参加组合(此时的空间占用量是 8B)。

这两种数据类型有一些缺点。首先, 用 PHP 来管理它们相当复杂(例如, 如果想知道 *ENUM* 字段的取值是哪一个字符串, 就不得不遍历那个枚举集合); 其次, 绝大多数其他的数据库系统根

1. 千万不要理解为“字符集”。——译者注

本不知道 *ENUM* 和 *SET* 是什么，这在日后需要把 MySQL 数据库移植到其他数据库系统去的时候会把事情弄得非常复杂。因此，多创建一个关联数据表来取代 *ENUM* 和 *SET* 数据列的做法往往更有实用价值。

表 8-10 对 MySQL 特有的几种数据类型进行了总结。

表 8-10 MySQL 特有的几种数据类型

MySQL 数据类型	含 义
<i>ENUM</i>	字符串的排列集合，最多可以有 65 535 个成员（1 或 2 字节；详见第 10 章）
<i>SET</i>	字符串的组合集合，最多可以有 255 个成员（1~8 字节；详见第 10 章）
<i>GEOMETRY</i> 、 <i>POINT</i> 等	二维地理数据对象（始见于 MySQL 4.1；详见第 12 章）

8.3.6 选项和属性

在创建数据列的时候还可以定义许多属性。表 8-11 列出了最重要的选项。注意，许多属性仅适用于特定的数据类型。

表 8-11 重要的数据列属性和选项

MySQL 关键字	含 义
<i>NULL</i>	数据列可以包含 <i>NULL</i> 值（这是默认的设置）
<i>NOT NULL</i>	不允许包含 <i>NULL</i> 值
<i>DEFAULT xxx</i>	如果在输入时没有给出一个具体的值，则使用 <i>xxx</i> 作为默认值
<i>DEFAULT CURRENT_TIMESTAMP</i>	仅适用于 <i>TIMESTAMP</i> 数据列，在创建新记录时保存当前时间
<i>ON UPDATE CURRENT_TIMESTAMP</i>	仅适用于 <i>TIMESTAMP</i> 数据列，在修改记录时（ <i>UPDATE</i> ）保存当前时间
<i>PRIMARY KEY</i>	把数据列定义为主键
<i>AUTO_INCREMENT</i>	自动输入一个序列编号。 <i>AUTO_INCREMENT</i> 属性仅适用于整数类型的数据列，而且必须与 <i>NOT NULL</i> 属性和 <i>PRIMARY KEY</i> 或 <i>UNIQUE</i> 属性同时使用
<i>UNSIGNED</i>	本数据列里的数据都将是无符号整数。警告：无符号整数的数学运算结果也将是一个无符号整数
<i>CHARACTER SET name [COLLATE sort]</i>	仅适用于字符串数据列，指定一种字符集和一种可选的排序方式

MySQL 不允许使用函数来设置默认值；例如，不允许使用 *DEFAULT RANDOM()* 来设置某个数据列的默认值——不管多想让 MySQL 自动生成一个随机数插入该数据列。MySQL 也不支持为数据列定义数据合法性检验规则，例如不可能只允许 0~100 之间的值存入某个数据列。

8.4 数据库设计技巧

8.4.1 数据库设计要求

一个好的数据库设计方案应该满足以下几项要求：

- 数据表里没有重复冗余的数据。（如果总是在往数据表里重复输入同样的数值或字符串，就意

味着有什么地方出了问题。)

- 数据表里没有 *order1*、*order2*、*order3* 等数据列 (*order* 在这里的意思是“订单”)。要知道，就算定义了 10 个这样的数据列，也迟早会发生某个用户想要订购 11 件商品的事情。
- 全体数据表的空间占用总量越小越好。
- 使用频率高的数据库查询都能以简单高效的方式执行。(注意：在数据库里的记录还比较少的时候，人们可能觉察不出什么；经常要等到数据库里有了成千上万条记录的时候，才会发现它没能满足这一要求，可那时往往已经不可能再对数据库设计方案进行修改了。)

以上要求只是些总体性原则，在接下来的 8.5 节里还提出了一些对数据库设计工作的细节要求。所有这些要求都同样重要，只不过这里给出的比较容易理解和遵循而已。

8.4.2 起名字的技巧

- MySQL 对数据列的名字不区分字母的大小写形式，但对数据库和数据表的名字却区分。因此，至少是在给数据库和数据表起名时，应该以一种统一的模式来使用大写和小写字母。(在本书的示例当中，笔者在命名数据库和数据表的时候只使用了小写字母。)
- 数据库、数据表和数据列的名字最多可以是 64 个字符长。
- 在名字里要避免使用特殊字符(如 äöü)。MySQL 允许使用所有的字母和数字字符，但不同的操作系统和不同的 Linux 发行版本所使用的默认字符集往往也不同，而对默认字符集等系统设置进行修改往往会导致一些难以预料的后果。
- 数据列和数据表的名字应该有意义。要尽可能地让数据列的名字可以准确地反映出它们的内容和用途。例如，*authName* 这样的名字就要比 *name* 好。
- 按照一定规范系统地给数据列起名有助于减少粗心产生的错误。喜欢选择像 *author_name* 这样的名字还是喜欢选择像 *authorname* 这样的名字并不重要，关键是应该一直保持这种风格。
- 类似地，还应该在选用单数名词还是选用复数名词作为名字的问题上做出统一的决定。笔者在给数据表和数据列起名字的时候从不选用复数名词。这里并没有什么对错之分，但如果数据表里有一半名字是单数名词、另一半名字是复数名词，恐怕时间一长自己都会糊涂。

8.4.3 数据库具体设计工作中的技巧

用最短的时间把一大堆杂乱无章的数据组织为一些井井有条的数据表并不是件容易的事情。下面是数据库设计领域的新手们应该牢记于心的一些建议。

- 从一批数量相对较少的测试性数据入手去尝试着把它们纳入一个或多个数据表。(如果测试性数据太少，有些设计问题就可能发现不了；但如果太多，又难免会让用户在设计阶段就浪费很多的时间。)
- 在第一次尝试时最好不要立刻就去创建和使用真正的 MySQL 数据表，应该先在 Excel 或 OpenOffice Calc 等电子表格程序里用一些工作表把 MySQL 数据表勾勒出来(如图 8-1 所示)。这么做的好处是可以在一个相对简单得多的环境里开展工作。在这个阶段，应该把注意力放在要把哪些数据安排到哪些数据表和哪些数据列，还不到考虑数据列格式和索引等数据库设计细节的时候。

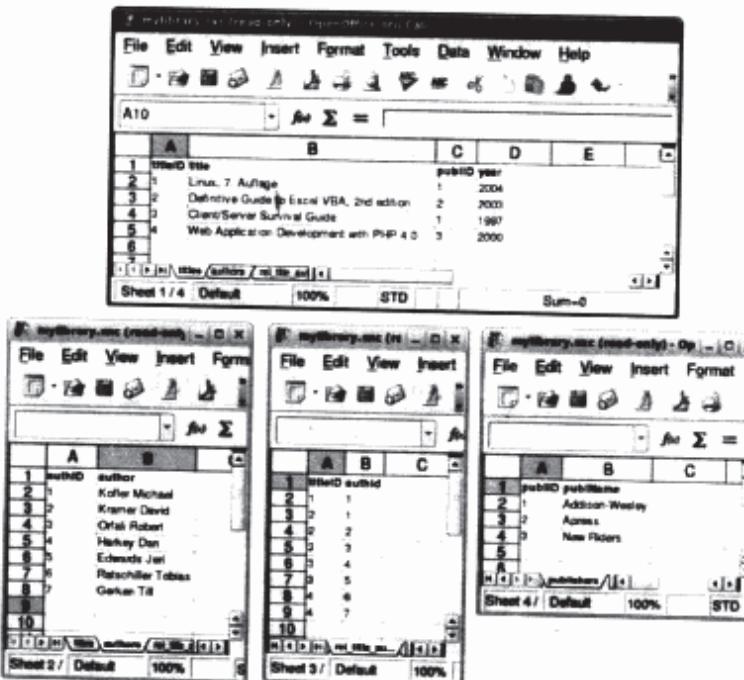


图 8-1 利用一个电子表格程序来设计数据库

8.5 规范化

写过书的人总是喜欢谈论书的事情，笔者也不例外——本节的示例都是和书打交道，这一点还要请大家谅解。这一节的目标是创建一个小型数据库来存放关于书的信息：书名、出版商、作者、出版日期等。当然，这些数据并不是必须要用一个数据库来保存不可，完全可以像下面这样把它们以文本格式保存为一份清单：

Michael Kofler: *Linux, 7th edition*. Addison-Wesley 2004.

Michael Kofler, David Kramer: *Definitive Guide to Excel VBA, Second Edition*. Apress 2003.

Robert Orfali, Dan Harkey, Jeri Edwards: *Client/Server Survival Guide*. Addison-Wesley 1997.

Tobias Ratschiller, Till Gerken: *Web Application Development with PHP 4.0*. New Riders 2000.

这份清单既简明又方便，所有必要的信息都在这儿了。看到这里，也许有人会问：把这些东西用一个字处理软件存为一份文档不就行了，为什么要费力气把它们存放到一个数据库里去呢？

既然准备把它当做例子介绍给大家，笔者当然有充足的理由：想从这份清单里查找些东西是不难，但想把它组织为另外一种形式——比如按作者列出所有图书或者是按书名列处所有图书——就不可能了。

在本章接下来的内容里，将一步一步地对结果数据库 *mylibrary* 进行设计和改进。最终完成的数据库可以在 Apress 出版公司的网站下载。顺便说一句，*mylibrary* 数据库是本书许多示例的基础。

8.5.1 起点

请先对自己说：没有什么事情比把这份清单转换为一个数据库表更简单了。（为了节约空间，在设计 *mylibrary* 数据库的时候将使用书名和作者名的缩略写法。）

表 8-12 是最容易想到的，但相信大家一眼就能发现它存在着许多不足。别的先不说，把作者的人数限定在 3 个就有欠考虑。如果有一本书的作者人数超过了 3 个怎么办？最简单的办法当然是继续增

加作者数据列直到 *authorN*，可这个 *N* 又该是多少呢？要知道，大多数图书的作者人数不会很多，而多出来的作者数据列只是白白地浪费空间。

表 8-12 图书数据库：初次尝试

<i>title</i>	<i>publName</i>	<i>year</i>	<i>authName1</i>	<i>authName2</i>	<i>authName3</i>
Linux	Addison-Wesley	2004	Kofler M.		
Definitive Guide ...	Apress	2003	Kofler M.	Kramer D.	
Client/Server ...	Addison-Wesley	1997	Orfali R.	Harkey D.	Edwards E.
Web Application ...	New Riders	2000	Ratschiller T.	Gerken T.	

8.5.2 第一范式

先告诉大家一个好消息：数据库理论家们早已为这类问题总结出了一个通用的解决方案，只需一步一步地三个范式（normal form）的规则应用到自己的数据库上就可以了。下面是第一范式应该遵循的规则（为了照顾不同水平的读者，这里把这些规则从数据库理论家们使用的专业语言翻译成了日常语言）：

- 内容相似的数据列必须消除（“消除”的办法是再创建一个数据表来存放它们）。
- 必须为每一组相关数据分别创建一个数据表。
- 每条数据记录必须用一个主键来标识。

先看第 1 条规则，它显然适用于这个例子里的 *authorN* 数据列。

第 2 条规则在这里似乎没有用武之地——将存入这个数据库的数据都与图书有关，有一个数据表好像就已经够用了（但稍后将会看到事实并非如此）。

第 3 条规则其实是一条实践经验，它的意思是数据表里的每一个数据行都应该包含一个独一无二的标识符作为索引。（注意，并不是只有整数才能充当主键，这里只有“独一无二”是必须满足的条件。但从时间和空间效率的角度考虑，主键应该尽可能地小和整齐。这样，整数当然要比长度变化不定的字符串更适合充当主键。）

按照第 1 条和第 3 条规则改进示例数据表之后，它应该变成如表 8-13 所示的样子。

表 8-13 图书数据库：第一范式

<i>titleID</i>	<i>title</i>	<i>publName</i>	<i>year</i>	<i>authName</i>
1	Linux	Addison-Wesley	2004	Kofler M.
2	Definitive Guide ...	Apress	2003	Kofler M.
3	Definitive Guide ...	Apress	2003	Kramer D.
4	Client/Server ...	Addison-Wesley	1997	Orfali R.
5	Client/Server ...	Addison-Wesley	1997	Harkey D.
6	Client/Server ...	Addison-Wesley	1997	Edwards E.
7	Web Application ...	New Riders	2000	Ratschiller T.
8	Web Application ...	New Riders	2000	Gerken T.

多位作者需要多个数据列来保存的问题已经解决了。现在，无论作者的人数有多少，都能把他们的名字存入数据表。但是，每增加一位作者，数据列 *title*、*publName* 和 *year* 的内容就不得不重复出现一次。肯定还有更好的办法！

8.5.3 第二范式

下面是第二范式的规则：

- 只要数据列里的内容出现重复，就意味着应该把数据表拆分为多个子表。
- 拆分形成的数据表必须用外键关联起来。

如果对数据库术语还不熟悉，就肯定会对外键（foreign key）这个词感到陌生。这个术语相当于人们日常用语里的“交叉引用”，因为每一个外键都分别指向另一个数据表里的某个数据行。对程序员而言，这个词相当于“指针”；对网民而言，它相当于“链接”。

在表 8-13 里，可以看到几乎每个数据列里都有重复内容，而导致这种冗余的根源显然是用来存放图书作者姓名的 *authorName* 列。把作者姓名拆分出来存入另外一个数据表的第一次尝试得到了表 8-14 和表 8-15。

表 8-14 *titles* 数据表：第二范式

<i>titleID</i>	<i>title</i>	<i>publName</i>	<i>year</i>
1	Linux	Addison-Wesley	2004
2	Definitive Guide	Apress	2003
3	Client/Server	Addison-Wesley	1997
4	Web Application	New Riders	2000

表 8-15 *authors* 数据表：第二范式

<i>authID</i>	<i>titleID</i>	<i>authName</i>
1	1	Kofler M.
2	2	Kofler M.
3	2	Kramer D.
4	3	Orfali R.
5	3	Harkey D.
6	3	Edwards E.
7	4	Ratschiller T.
8	4	Gerken T.

在 *authors* 数据表里，第 1 个数据列存放着顺序递增的 *authID* 值，这些 *authID* 值将作为这个数据表的主键。第 2 个数据列承担起了提供外键的任务，而那些外键分别指向（或者说引用）*titles* 数据表里的某个数据行。例如，*authors* 数据表里的第 7 行可以告诉用户：*Ratschiller T.* 是书名编号为 *titleID=4* 的书（它的书名是 *Web Application...*）的作者之一。

第二步范式，第二次尝试

现在的改进结果还远谈不上最优，因为人名 *Kofler M.* 在 *author* 数据表里出现了两次。千万不要以为这是个小问题——随着这个数据库所收录的图书数量的增加，只要某位作者写过一本以上的书，现在看到的这种冗余就会加剧。

唯一的解决方案是对 *author* 数据表再次进行拆分：把 *titleID* 列分离出去，创建第 3 个数据表来保存关于哪些图书有哪些作者的信息。这 3 个数据表就是表 8-16~表 8-18。

表 8-16 *titles* 数据表：第二范式

<i>titleID</i>	<i>title</i>	<i>publName</i>	<i>year</i>
1	Linux	Addison-Wesley	2004
2	Definitive Guide	Apress	2003
3	Client/Server	Addison-Wesley	1997
4	Web Application	New Riders	2000

表 8-17 *authors* 数据表：第二范式

<i>authID</i>	<i>authName</i>
1	Kofler M.
2	Kramer D.
3	Orfali R.
4	Harkey D.
5	Edwards E.
6	Ratschiller T.
7	Gerken T.

表 8-18 *rel_title_author* 数据表：第二范式

<i>titleID</i>	<i>authID</i>
1	1
2	1
2	2
3	3
3	4
3	5
4	6
4	7

这一步相当困难和抽象的改进，因为 *rel_title_author* 数据表里根本没有现实世界里的内容。这样的数据表完全不适合以非自动化方式进行管理，但对计算机和 MySQL 之类的软件来说，处理这种数据正是它们的看家本领。现在，假设想查 *Client/Server...* 一书的作者都有谁，MySQL 将先从 *titles* 数据表里查出这本书的 *titleID*，再从 *rel_title_author* 数据表找出所有包含着这个编号的数据记录，并根据其中的 *authorID* 编号把作者的名字全部查出来。

注解 可能会有读者对 *rel_title_author* 数据表里没有一个类似 *rel_title_author_ID* 的 ID 列感到奇怪，但这并不是疏忽，这样的数据列通常是可以省略的，因为 *titleID* 和 *authorID* 的组合已经是一个最优的主键了。（各种关系数据库系统都允许人们把多个数据列的组合定义为主键。）

8.5.4 第三范式

第三范式只有一条规则：与主键没有直接关系的数据列必须消除（“消除”的办法是再创建一个数据表来存放它们）。

在这个例子里，*titles* 数据表里有一个 *publisher* 列，但出版公司的名字与图书的名字并没有必然

的联系。既然它们是彼此独立的两组数据，就应该把它们分离开。当然不能无视每一本书都必须对应着一家出版公司的事实，但这并不意味着必须把出版公司的全名在 *titles* 数据表里写出来；这个问题用一个外键（或者说“一个引用”、“一个指针”或“一个链接”）就足以解决了，如表 8-19 和 8-20 所示。

表 8-19 *titles* 数据表：第三范式

<i>titleID</i>	<i>title</i>	<i>publID</i>	<i>year</i>
1	Linux	1	2001
2	Definitive Guide	2	2003
3	Client/Server	1	1997
4	Web Application	3	2000

表 8-20 *publishers* 数据表：第三范式

<i>publID</i>	<i>publName</i>
1	Addison-Wesley
2	Apress
3	New Riders

author 和 *rel_title_author* 数据表不需要进行第三范式，它们仍保持刚才的样子。现在，图书数据库总共有 4 个数据表。

如果在第一范式时就仔细考虑过第 2 条规则（“必须为每一组相关数据分别创建一个数据表”），是可以节省一些中间环节的。但那么做会让这个例子失去教学演示意义。事实上，在实际工作中，人们往往要等到数据库里已经有了足够的测试数据之后才会注意到种种冗余现象，才会清楚地知道怎样拆分数据表最合适。

提示 最终的 *mylibrary* 数据库要比现在看到的更复杂一些：*titles* 表增加了几个字段来保存图书的副标题、内容简介或注释评论；增加了一个 *languages* 数据表来存放图书写作语言清单，并相应地在 *titles* 表里增加了一个 *langID* 字段来指明每本书的写作语言；最后，增加了一个 *categories* 数据表来存放图书门类（计算机类、文学类等）清单并相应地在 *titles* 数据表里增加了一个 *catID* 字段来指明每本书的所属门类。

8.6 节将对 *categories* 数据表的结构进行分析。*mylibrary* 数据库的完整定义见 8.11 节。

8.5.5 规范化理论

1. 如果想学习更多的理论知识

上面介绍的关系数据库三个范式是著名学者 E. F. Codd 最先提出来的，后人又在此基础上对大到数学集合理论、小到关系数据库设计细节等诸多方面进行了研究和探索。

有些探讨数据库设计问题的图书在三个范式基础上又增加了三个范式，但这些范式在实践中重要性不大。范式及其相应规则只是把刚才做过的事情描述得更细致而已。在笔者看来，那些充斥着“实体”、“属性”等专业术语又彼此雷同的教条不仅没有把事情说清楚，反而会让人们更看不出它们与关系数据库有什么联系。

如果对这个话题的细节感兴趣，就应该找一本探讨数据库设计理论和实践的好书来读（比如笔者

在本章开篇推荐的那几本书)。

2. 如果是个急性子的初学者

笔者已经尽了最大努力来把这本书里的第一个三范式的例子写得简明直观，但这对某些读者来说理论性可能还是强了些。说老实话，范式对数据库设计领域的新手不见得会有很大的帮助——那些规则本身并不复杂，但正确地理解和实践它们就没那么简单了。因此，为了帮助大家在刚起步时少遇到些坎坷，下面介绍一些可以立即上手的方法。

- 在设计数据库的时候，一定要给自己以充足的时间。(如果等到数据库里已经充满了数据、当配套的客户端程序也已经开发完成的时候，才猛然发现数据库设计方案还需要修改，那么将要花费的时间和精力可就太大了。)
- 如果发现自己给数据列起的名字里有序号，比如 *name1*、*name2* 或者是 *object1*、*object2*，请提高警惕。这种现象几乎总是意味着还有更好的解决方案没有想到——再多创建一个数据表。
- 在第一时间向数据库里输入一些测试用途的数据，而且要尽可能地多包括一些特殊情况。如果数据库里的数据出现了冗余，即同一个数据列里多次出现了同样的内容，往往是应该把数据表拆分成两个（甚至更多个）新数据表的提示信号。
- 注意发现和运用各个数据表之间的关联/引用关系。
- 掌握 SQL 语言（见第 9 章和第 10 章）。缺少 SQL 编程经验的人是很难拿出一个优秀数据库设计方案的。把信息存入数据库的目的是为了让更多人能够使用 SQL 查询命令把它们再迅速准确地查出来，只有了解了 SQL 查询命令的涉及范围，才能找出把数据分门别类地存入数据库的最佳办法。
- 找个示例数据库作为借鉴（本书和其他关于数据库的书籍里都有这样的示例数据库）。

提示 <http://www.phpbuilder.com/columns/barry20000731.php3> 上有一个非常好的数据库设计实例教程。

3. 范式的优缺点

范式再怎么好，也只是一种手段。它可以帮助人们又快又好地设计数据库，但不能代替人们思考。在某些场合，机械地按照范式把所有的冗余都消除干净并不能获得最佳效果。

范式的缺点：数据表的个数越多，把从网页上的表单输入的数据分门别类地存入这些数据表的复杂性就越大。这种复杂性不仅会给程序员带来烦恼，也会给最终用户带来不便（他们不得不一个接一个地填写表单）。

不仅如此，数据表的个数越多，从中提取相关数据生成查询结果的复杂性也越大。为了提高查询的效率，适度的冗余有时反而是必要的。从多个数据表提取数据往往要比从单个数据表提取数据来得慢；这对那些已不再需要或是很少需要修改但经常需要对复杂查询做出快速响应的数据库来说更是如此。（数据库的世界里有一个领域叫做“数据仓库”。在设计数据仓库的时候，人们往往会有意识地增加一些冗余度以获得更好的响应速度。不过，MySQL 数据库系统至少在目前还不是人们在建设数据仓库时的首选，所以对这一特殊应用领域里的细节问题也就不多加介绍。）

范式的优点：冗余意味着存储空间的浪费。这种浪费在硬盘的单体容量已经达到 400GB 的今天似乎并不是什么大问题，但事实却是一个大数据库往往同时也是一个慢数据库（至少在数据库的大小超出了计算机内存容量的时候会如此）。

一般而言，严格按照范式设计出来的数据库能够提供最丰富、最灵活的查询选项。（人们往往要

等到必须使用一种新的查询或是必须对数据进行一种新的分类时才会真正意识到这一点，但可惜的是这些新需求几乎总是出现在数据库已投入运行几个月之后，那时再去改动数据库设计方案的代价是令人难以承受的。)

8.6 层次关系的处理

在最终的 *mylibrary* 数据库里有一个名为 *categories* 的数据表，它的作用是帮助人们按门类（文学类、儿童书籍类等）来组织和管理图书。之所以会把这个数据表单独拿出来在一节里进行讨论，是因为图书的门类是一种典型的层次化关系。从数据库设计工作的角度看，让每条门类记录的 *parentCatID*¹ 字段指向它的父门类即可。（把根门类 *All Books* 的 *parentCatID*² 字段设置为 *NULL* 值——在管理和使用 *categories* 数据表的时候，千万要把这个特例考虑进去。）表 8-21 是图书门类的层次结构图，表 8-22 是这个层次结构的数据表形式。

表 8-21 *categories* 数据表的示例数据

All books
Children's books
Computer books
Databases
Object-oriented databases
Relational databases
SQL
Programming
Perl
PHP
Literature and fiction

表 8-22 表 8-21 所示层次关系的数据表形式

catID	CatName	parentCatID
1	Computer books	11
2	Databases	1
3	Programming	1
4	Relational databases	2
5	Object-oriented databases	2
6	PHP	3
7	Perl	3
8	SQL	2
9	Children's books	11
10	Literature and fiction	11
11	All books	NULL

1. 原书在这里误写为“parentID”。——译者注

2. 原书在这里误写为“parentID”。——译者注

8.6.1 层次关系的处理难点

如果单从表 8-21 和表 8-22 来看，把层次关系转化为数据表并不困难，而勾勒出来的数据表看上去还很精致，但在这种表面现象的背后还有许多难题在等待着用户。例如，我们根本无法只用一个简单的 *SELECT* 查询就把某给定门类的父门类或子门类全都查出来——必须通过客户端程序执行多个查询才能构造出这些层次。这方面的编程技巧将在第 15 章里介绍。也可以利用 MySQL 提供的一些存储过程（stored procedure）来管理这种层次化的关系（详见第 13 章）。

提示 本章的讨论重点是数据库设计而不是 SQL 编程技术，但这两个话题有着千丝万缕的联系，很难把它们割裂开。事实上，如果 SQL 技能还不足以把想要查找的数据从数据表里提取出来，想拿出一个优秀的数据库设计方案就只能是一句空谈。

没有 SQL 编程经验的读者应该先去钻研一下第 9 章再回到这里继续阅读——以下内容完全可以搬到第 9 章并冠以“高级数据库设计技巧”之类的标题。

要想通过数据表去管理和使用层次关系，还需要解决许多难题，而这些难题几乎都与 SQL 语言不允许递归查询这一事实有关。以 *mylibrary* 数据库里的 *categories* 数据表为例：

- 只用一个查询就把某给定门类的所有上级门类全部查出来是不可能的。

例题：给定图书门类 *Relational databases* (*parentCatID*=2)，请利用 SQL 查询创建一个内容为 *Computer books* → *Databases* → *Relational databases* 的列表。

解析：*SELECT * FROM categories WHERE catID=2* 命令只能查到 *Databases* 门类，但查不到 *Computer books* 门类，后者在图的图书层次关系里比给定图书门类 *Relational databases* 高两级。因此，还需要再执行一条 *SELECT * FROM categories WHERE catID=1* 命令。肯定会想到这可以用某种程序设计语言（Perl、PHP 等）中的循环语句来实现，可刚才说的是“只用一个查询……”——不管选用哪一种程序设计语言，一条 SQL 指令都是绝对无法完成这个任务的。

- 把完整的数据表还原为层次关系形式（树状结构）也是个难点。还是必须执行多个查询才能完成这个任务。

- 把属于某给定门类¹的图书全部查出来是一项相当复杂的任务。

例题：请利用 SQL 查询把属于给定门类 *Computer books* 的图书全部查出来。

解析：*SELECT * FROM titles WHERE catID=1* 命令只能查出直接属于 *Computer books* 门类的图书，但查不到属于 *Databases*、*Relational databases*、*Object-oriented databases* 等门类的图书。还需要再执行 *SELECT * FROM titles WHERE catID IN (1, 2, …)* 命令，其中 1, 2, … 是子门类的 ID 编号——这道例题的真正难点是如何确定这些编号。

顺便说一句，就算是最低层次的门类，也必须多执行一条查询才能确认它下面再也没有任何子门类了——在进行这一查询之前，不可能知道给定门类是不是一个最低层次的门类。只有这样，设计出来的程序才是完备的！

- 在这个相对比较简单的层次关系里，不可能把同一个子门类与两个或更多个父门类关联起来。

例题：无。

解析：在这个例子里，SQL 语言被放在了 *Databases* 门类下。可是，把 SQL 语言放在 *Programming*

1. 原书这里是“higher categories”，即“较高层次的门类”；但这种说法是不完备的，见其下面的“顺便说一句……”。

门类下也很合乎逻辑。换句话说，如果让 *SQL* 同时成为 *Databases* 和 *Programming* 门类的子门类将是最合理的。

- 把层次关系表示为数据表最怕的是留下循环引用隐患。错误的输入数据是导致循环引用的根源，但只要是由人输入的数据（或者是由人编写的程序），就有可能出现错误。如果发生循环引用，绝大多数数据库系统都会陷入无限循环，而预防和解决这种问题的措施往往非常复杂。

上述问题尽管都很困难，但幸好还都有办法可以解决。从上面的讨论可以看出，层次关系往往会导致这样一种局面：即便是一个相对简单的问题也需要执行许多条 SQL 查询命令才能回答，而且整个过程相当慢。如果不使用层次关系（更准确地说，把层次关系限制在最多两级），上面提到的绝大多数问题都可以避免。如果必须使用两级以上的层次关系，增加一些数据列或数据表来提供更多关于层次关系的信息（参见 8.6.2 节），将有助于让层次关系的解决方案变得简单一些。

8.6.2 从数据表创建层次关系树

如果认真阅读过 8.5 节，就应该知道冗余是不好的。它会导致存储空间不必要的浪费，增加数据库内部管理工作的负担。

可是，在某些场合，为了提高应用程序的执行效率，冗余反而是一种相当简明的解决方案。

希望下面的介绍能够让大家明白数据库设计其实是一件多方妥协和折中的事情。在数据库领域，通往同一个目标的道路往往会有好几条，而选择这些道路中的任何一条都意味着做出了这样或那样的妥协。做出什么样的妥协最有利在很大程度上要取决于数据库的具体使用情况：什么类型的查询发生的最为频繁？数据需不需要频繁修改？

当需要把 *categories* 表显示为一个如表 8-21 所示的层次关系树形式时，就更有必要做出一些妥协了。正如在前面曾提到过的，对数据进行这类处理就意味着要与无数的 SQL 查询命令和/或复杂的客户端代码打交道。

可供选择的解决方案之一是在 *categories* 数据表里再增加两个数据列：一个是 *hierNr* 列，它负责给出各条数据记录将出现在层次关系树的第几行（假设同属一个层次的数据记录将按 *catName* 数据列取值的字母表顺序排序）；另一个是 *hierNrIndent* 列，它负责确定数据记录在显示时的缩进量。表 8-23 所示是增加了这两个数据列之后的 *categories* 数据表，表中的数据对应着前面的表 8-21 所示的层次关系树。

表 8-23 增加了 *hierNr* 列和 *hierNrIndent* 列的 *categories* 表

<i>catID</i>	<i>CatName</i>	<i>parentCatID</i>	<i>HierNr</i>	<i>hierIndent</i>
1	Computer books	11	2	1
2	Databases	1	3	2
3	Programming	1	7	2
4	Relational databases	2	5	3
5	Object-oriented databases	2	4	3
6	PHP	3	9	3
7	Perl	3	8	3
8	SQL	2	6	3
9	Children's books	11	1	1
10	Literature and fiction	11	10	1
11	All books	NULL	0	0

这种安排的好处可以通过在 mysql 程序里执行下面这条简单查询的办法得到证明。下面先解释一下这里用到的 SQL 函数：*CONCAT()* 把两个字符串合并为一个；*SPACE()* 生成指定个数的空格字符。*AS* 关键字给整个表达式起一个新的假名 *category*。

```
SELECT CONCAT(SPACE(hierIndent*2), catName) AS category,
       hierNr, hierIndent
  FROM categories ORDER BY hierNr
    category          hierNr  hierIndent
      All books        0        0
      Children's books 1        1
      Computer books   2        1
      Databases        3        2
      Object-oriented databases 4        3
      Relational databases 5        3
      SQL               6        3
      Programming      7        2
      Perl              8        3
      PHP               9        3
      Literature and fiction 10       1
```

读者现在可能会问：*hierIndent* 和 *hierNr* 的值究竟是怎么来的？我们将通过下面的例子（增加一个新的图书门类 *Operating systems*）来演示如何在 *categories* 数据表里插入一条新的数据记录。

(1) 新图书门类 *Operating systems* 将被插入到 *Computer books* 门类的下一级层次，而与后者有关的数据是已知的：*catID=1, parentCatID=11, hierNr=1, hierIndent=1*。

(2) 现在，需要在 *Computer books* 门类的下一级层次里找到将出现在新插入的 *Operating systems* 门类后面的第一个现有门类（具体到这个例子，是 *Programming*）。真正感兴趣的是这条记录的 *hierNr* 值。下面先简单解释一下将要用到的 SQL 命令：

WHERE parentCat_ID=1: 找出 *Computer books* 门类的下一级层次里的所有记录（具体到这个例子，它们是 *Databases* 和 *Programming*）。

catName>'Operating Systems': 只选取将出现在新门类 *Operating systems* 之下的那些数据记录。

ORDER BY catname: 对查找出来的记录进行排序。

LIMIT 1: 把最终结果限定为第一条记录。

```
SELECT hierNr FROM categories
 WHERE parentCatID=1 AND catName>'Operating Systems'
 ORDER BY catName
 LIMIT 1
```

上面这个查询的返回结果将是 *hierNr=7*。这样就知道了新插入的数据记录将使用这个层次编号。在这之前，还需要先给所有 *hierNr>=7* 的现有记录的 *hierNr* 值加上 1，把 *hierNr=7* 的位置空出来留给新门类 *Operating systems*。

(3) 步骤(2)里的查询有可能不会返回任何结果，会出现这种情况的场所有两种：一是新门类的上一级层次没有记录项（新门类将成为根门类 *All books* 的一个直接下级）；二是现有门类在按字母表排序时全都出现在新门类的前面（如果打算在 *Computer books* 门类的下一个层次里插入的新图书门类是 *Software engineering*，就会遇到这种情况）。

在这两种场合下，需要继续搜索这样一条记录：它的 *hierNr* 值大于已知门类 *Computer books* 的 *hierNr* 值、但它的 *hierIndent* 值小于或等于已知门类 *Computer books* 的 *hierIndent* 值。这样查找出来的记录将是层次与已知门类 *Computer books* 相同或更高的下一个图书门类。

```
SELECT hierNr FROM categories
 WHERE hierNr>1 AND hierIndent<=1
 ORDER BY hierNr LIMIT 1
```

这个查询的返回结果将是 *hierNr=10*（对应于 *Literature and fiction* 门类）。也就是说，新记录将使用 10 作为自己的 *hierNr* 编号，而所有 *hierNr>=10* 的现有记录的 *hierNr* 值都必须加上 1，把 *hierNr=10* 的位置空出来留给新门类。

(4) 如果步骤(3)里的查询还是没有返回任何结果的话，就意味着必须把新记录插入到整个层次结构的最后一行。此时需要知道 *hierNr* 数据列的当前最大值，而这可以用下面的查询命令轻松查出：

```
SELECT MAX(hiernr) FROM categories
```

(5) 给现有记录的 *hierNr* 值加 1 的工作可以用下面这条命令来完成（请注意：因为这个例子不需要进行步骤(3)，所以下面这条命令对应于，步骤(2)。在需要进行步骤(3)的场合——仍以插入 *Software engineering* 门类为例——将需要把 *hierNr>=7* 相应地改为 *hierNr>=10*）：

```
UPDATE categories SET hierNr=hierNr+1 WHERE hiernr>=7
```

现在，终于可以插入新记录了。把新记录的 *parentCatID* 字段设置为已知门类 *Computer books* 的 *CatID* 值；把新记录的 *hierNr* 字段设置为在步骤(2)里查出来的“7”；把新记录的 *hierIndent* 字段设置为已知门类 *Computer books* 的 *hierIndent+1*：

```
INSERT INTO categories (catName, parentCatID, hierNr, hierIndent)
VALUES ('Operating systems', 1, 7, 2)
```

可见，*categories* 数据表里的新增数据列 *hierNr* 和 *hierIndent* 简化了读操作，但插入操作还是那么复杂。不过，更复杂的是对现有层次关系进行修改。例如，假设打算修改某个现有图书门类的名字，这很可能会改变它在按字母表排序的图书门类清单里的位置。这种修改不仅会影响到这条记录本身，还会波及到许多其他的记录，*categories* 表里会有相当多的记录需要调整 *hierNr* 值。看到冗余带来的麻烦了吧！

综上所述，必须决定对读操作进行优化还是对写操作进行优化读者。在这本书里，笔者在最终的 *categories* 表里舍弃了 *hierNr* 和 *hierIndent* 列。这首先是出于教学方面的考虑：将在第 13 章（存储过程）和第 15 章（PHP）向大家介绍许多能够以最大效率去处理各种层次关系数据的编程技巧。其次，笔者一直认为，越是简单的数据库设计，在实际应用中的长远效果就越好。最后，层次关系虽然处理起来很复杂，但数据总量往往都不是很大——就算是 amazon.com 网站的图书数据库，设置一两千个图书门类也足够用了。一般来说，只有在那些大得多的数据表上才会出现严重的效率问题。

8.6.3 搜索 *categories* 数据表里的下级图书门类

现在，不妨假设需要从 *titles* 数据表里把属于 *Databases* 门类的图书全部查找出来。注意，只查找 *catID=2* 的记录是无法完成这个任务的，必须把属于 *Relational database*、*Object-oriented database* 和 *SQL* 门类（也就是 *catID* 等于 4、5、8 的记录）的图书都查找出来才算圆满。这些图书门类只在有关字段的具体取值方面有所不同，在其他方面并没有什么差异，所以可以用同样的算法来搜索它们。在下面的讨论里，将不特意区分具体的图书门类而是把它们当做同样的事物来对待。

第一种解决方案是只利用现有的 *titles* 和 *categories* 数据表来搜索那些图书，这需要解决两个问题。第一，在开始搜索 *titles* 数据表之前，必须先把与本次搜索有关的 *catID* 值全部查出来。这可以用一系列 *SELECT* 查询来完成，但不打算在这里深入探讨这个话题（参见第 13 章和第 15 章）。第二，从 *titles* 数据表里把与刚查到的 *catID* 值相匹配的数据记录全部搜出来。

这个方案可以完成搜索图书的任务，只是过程相当复杂，它不仅需要编写多条 SQL 查询命令，还需要编写一些客户端代码。

另一种解决方案是先创建一个新的（冗余的）数据表来保存图书门类之间的层次关系，把每一个图书门类的 *categories* 记录都与它所有上级门类的 *categories* 记录关联起来。这个新数据表——不妨把它命名为 *rel_cat_parent*——由两个数据列构成：*catID* 和 *parentID*（如表 8-24 所示）。这个数据表可以告诉我们，*Relational database* 门类 (*catID=4*) 总共有 3 个上级门类，分别是 *All books*、*Computer books* 和 *Databases* (*parentID=11, 1, 2*)。

表 8-24 *rel_cat_parent* 数据表里的一些数据记录

<i>catID</i>	<i>parentID</i>
1	11
2	1
2	11
3	1
3	11
4	1
4	2
4	11
	..

这一解决方案的最大弊端是必须让 *rel_cat_parent* 数据表与 *categories* 数据表的每一次修改保持同步，但万幸的是这种同步很容易实现。

作为所付出努力的回报，*Databases* 门类都有哪些下级门类的问题现在变得非常容易回答：

```
SELECT catID FROM rel_cat_parent
WHERE parentID=2
```

接下来，如果想把属于 *Databases* 门类以及属于它的各个下级门类的图书全部查出来，只须再执行一条如下所示的查询命令即可。关键字 *DISTINCT* 在这里必不可少，否则这个查询将返回许多重复雷同的书名：

```
SELECT DISTINCT titles.title FROM titles, rel_cat_parent
WHERE (rel_cat_parent.parentID = 2 OR titles.catID = 2)
    AND titles.catID = rel_cat_parent.catID
```

这里又一次遇到了查询效率与数据库设计原则发生冲突的情况：*rel_cat_parent* 数据表里的数据没有一项是不能直接从 *categories* 数据表里查到的，但这个冗余的数据表却让数据查询工作变得既简明又高效。在经过一番犹豫之后，笔者决定在最终的 *mylibrary* 数据库里舍弃 *rel_cat_parent* 数据表。

8.6.4 搜索 *categories* 数据表里的上级图书门类

在这里遇到的问题刚好与 8.6.3 节中的问题相反：给定图书门类都有哪些上级门类？比如说，如果给定的图书门类是 *Perl* (*catID=7*)，那么它的直接上级门类将是 *Programming*、然后是 *Computer books*、最后是 *All books*。

利用上一节定义的 *rel_cat_parent* 数据表，这个问题可以用以下简单的查询来回答：

```
SELECT CONCAT(SPACE(hierIndent*2), categoryName) AS category
FROM categories, rel_cat_parent
WHERE rel_cat_parent.catID = 7
    AND categories.catID = rel_cat_parent.parentID
ORDER BY hierNr
```

```

category
All books
Computer books
Programming

```

不过，要是没有 *rel_cat_parent* 数据表可供利用，就必须在一个循环里通过执行一系列 *SELECT* 命令的办法沿着 *categories.parentCatID* 字段进行上溯，直到这个字段的值是 *NULL* 为止。

8.7 关系

在将数据库转换为范式的过程中，需要把一系列数据表关联起来。这种关联用数据库术语来说就是“关系”。两个数据表之间的关联/引用关系可以细分为以下 3 种：

- **1:1 关系。** 两个数据表之间的“1:1 关系”是这样的：第一个数据表里的每一条数据记录分别对应着第二个数据表里的一条数据记录，同时第二个数据表里的每一条数据记录也分别对应着第一个数据表里的一条数据记录。在实际工作中，这种关系相当少见，因为两个有着 1:1 关系的数据表完全可以合并为一个。
- **1:n 关系。** 两个数据表之间的“1:n 关系”是这样的：第一个数据表里的一条数据记录可以对应着第二个数据表里的多条数据记录（比如说，同一个供货商可以收到多份订单），但反过来却不能成立（比如说，同一份订单不可能交给多个供货商）。读者可能还曾听说过“n:1 关系”，但那不过是从反方向看过来的“1:n 关系”而已。
- **n:m 关系。** 两个数据表之间的“n:m 关系”是这样的：第一个数据表里的一条数据记录可以对应着第二个数据表里的多条数据记录，同时第二个数据表里的一条数据记录也可以对应着第一个数据表里的多条数据记录。比如说，同一份订单可以包括好几种商品，同一种商品又可以出现在多份不同的订单里。图书和作者之间的关系也是一种 n:m 关系：几位作者可以合著一本书，而每位作者又可能参加了好几本书的编写工作。

8.7.1 1:1 关系

1:1 关系的起源通常是因为人们把某个数据表拆分成了两个使用同一个主键的数据表。这种情况在下面这个例子里体现得最明显：某公司有一个用来保存员工个人资料的数据表，这个数据表里存放着每位员工的姓名、任职部门、出生日期、进厂日期等信息。出于某种考虑，该公司决定把这个数据表拆分为两个，一个是 *personnel* 数据表，用来存放需要经常查询和用不着保密的信息；另一个是 *personnel_extra* 数据表，用来存放不经常使用和需要保密的信息。

按 1:1 关系对数据表进行拆分的理由不外乎两种。第一种理由是出于信息安全方面的需要。就拿上面的例子来说，那家公司只须简单地限制无关人员对 *personnel_extra* 数据表进行访问就可以达到既不泄露员工隐私、又不影响日常工作的目的。（这种谨慎在 MySQL 数据库系统里并不是很有必要，因为 MySQL 可以给数据表里的每一个数据列分别设置一个不同的访问权限。从 MySQL 5.0 开始，信息的安全保密问题还可以通过创建一些视图的办法来解决。）

第二种理由是为了提高数据库系统的响应速度。如果某个数据表里有很多个数据列、但其中只有少数几个是经常被查询的，把那几个常用的数据列拿出来存放为另一个数据表将大大有助于提高查询工作的效率。（在理想的情况下，常用的数据表最好是完全由一些固定长度的数据列构成，这种数据表比那些数据列长度可变的数据表更容易管理，查询效率也更快。对各种 MySQL 数据表类型的介绍可以在本章前半部分内容里找到。）

按 1:1 关系对数据表进行拆分的最大弊端是，系统会因为需要确保两个数据表里的数据保持同步而增加资源开销。

8.7.2 1:n 关系

1:n 关系是两个数据表之间最常见的关系，只要在第一个数据表（所谓“细节数据表”）的数据记录里有一个特定字段引用了——无论是何种引用方式——第二个数据表（所谓“主控数据表”）里的数据列，就会形成 1:n 关系。

1:n 关系是通过键字（key）字段发生的。主控表里的数据列由一个主键（primary key）标识；细节表里则包含着一个外键（foreign key）字段，该字段的内容指向着主控表。下面是一些例子。

- **mylibrary 数据库。** *titles* 数据表与 *publishers* 数据表之间存在着一个 1:n 关系，*publishers* 表是主控表，主键是 *publishers.pubID*。

每家出版公司（1:n 关系中的“1”）可以出版多本图书（1:n 关系中的“n”）。

mylibrary 数据库里还存在着另外两个 1:n 关系：一个是 *titles* 数据表与 *languages* 数据表之间（字段 *langID*），另一个是 *titles* 数据表与 *categories* 数据表之间（字段 *catID*）。

- **某商业应用软件中的订单数据表。** 在细节表里存放着所有已经处理过的订单，这个数据表通过一个外键字段（“顾客”字段）引用了主控表的一个数据列（“顾客”数据列），从而形成了 1:n 关系。

每位顾客（1:n 关系中的“1”）可以下达多份订单（1:n 关系中的“n”）。

- **网上论坛的消息帖子数据表。** 在细节表里存放着网上论坛收到的消息帖子的数据（标题、内容、日期、作者、讨论组等）。它通常对应着两个主控表：一个是“讨论组”主控表，其内容是一份该论坛上的全体讨论组名单；另一个是“作者”主控表，其内容是一份该论坛允许在本论坛的某个讨论组里发表帖子的网友名单。

每位作者（1:n 关系中的“1”）可以发表任意多份帖子（1:n 关系中的“n”）。每个讨论组（1:n 关系中的“1”）可以包含任意多份帖子（1:n 关系中的“n”）。

- **某数据库里的音乐光盘数据表。** 在细节表里存放着该数据库所收录的每一套 CD 光盘的数据（光盘名称、表演者、套装光盘个数等）。它通常对应着两个主控表：一个是“表演者”主控表，其内容是一份该数据库所收录的全体表演者名单；另一个是“唱片公司”主控表，其内容是一份唱片公司名单。

每位表演者（1:n 关系中的“1”）可以出现在任意多张音乐光盘上（1:n 关系中的“n”）。每家唱片公司（1:n 关系中的“1”）可以发行任意多张唱片（1:n 关系中的“n”）。

注解 在创建数据库的时候，人们往往会有意识地给两个有着 1:n 关系的数据表里的相应数据列起上一个同样的名字。这种做法很有条理并有助于人们认清 1:n 关系，但并不是必要的。

主键和外键还可以出现在同一个数据表里。这时候，该数据表里的某一条数据记录将引用同一个数据表里的另一条数据记录。这种情况多发生在需要表示一种层次关系的场合。下面是几个这样的例子：

- **mylibrary 数据库。** *categories* 数据表里的每一个图书门类通过 *parentID* 字段指向另一个图书门类（或 NULL）。

- **员工个人资料表。** 在这个数据表里，每条员工记录（除了老板的）都通过一个“上级主管”

字段指向该名员工的顶头上司。

- **网上论坛的消息帖子数据表。**在这类数据表里，每份帖子都通过一个“跟帖标题”字段表明自己是对哪份帖子做出的后续回应（用术语来说，就是“指向层次关系中的上一级”）。
- **某数据库里的音乐光盘数据表。**音乐有很多风格，每种风格又细分为一些具体的表现形式。在这类数据表里，每张唱片光盘可以通过一个“音乐风格”字段表明自己属于哪一种音乐风格（比如说，“街舞”是“爵士”音乐的一种；“小提琴”是“管弦乐”中的一种）。

8.7.3 n:m 关系

n:m 关系在人类社会中十分常见，但在设计数据库的时候，需要为每两个有着 n:m 关系的数据表多定义一个辅助数据表，并利用这个辅助数据表把一组 n:m 关系转化为两组 1:n 关系。下面是一些例子：

- **mylibrary 数据库。**在这个数据库里，书名和作者之间存在着 n:m 关系，而这里是通过 *rel_title_author* 数据表把这个关系建立起来的。
还可以再对 *rel_title_author* 数据表进行一些扩展，例如增加一个字段来表明作者们的署名顺序（不仅仅是按字母表排序）等。
- **某商业应用软件中的订单数据表。**为了在订单和订购商品之间建立起关系，需要增加一个辅助数据表来记载 y 号订单购买了多少件 x 商品。
于是，这个辅助数据表应该由 *articleID*（商品号）、*orderID*（订单号）、*quantity*（数量）——或许还有一个 *price*（价格）——等几个数据列构成。把“价格”也包含在这个辅助数据表里的理由是：商品价格会随着时间和许多其他因素发生变化，顾客订单里的实际成交价与厂家的最新报价很可能会不一样；这个辅助数据表里的“价格”是最新报价，而订单数据表里的“价格”是实际成交价。
- **学院管理、考试表。**为了记录哪些学生已经通过了哪些考试以及考试成绩，有必要在“学生”数据表和“考试”数据表之间再建立一个数据表。

8.8 主键和外键

数据表之间抽象的关联/引用关系是依靠具体的主键（primary key）和外键（foreign key）建立起来的。本节将对这两个概念做深入细致的解释。在讨论过程中，将不可避免地会涉及到一些 SQL 命令，对这些命令详细介绍请参见第 9 章和第 10 章。

8.8.1 主键

主键的任务是帮助 MySQL 以最快的速度把一条特定的数据记录在数据表里的位置确定下来（比如说，在一个有着几百万条记录的数据库里找到 *id=314159* 的那条记录的位置），而这种寻址操作在一切需要从多个数据表提取数据的场合都会发生，简单地说，就是发生得非常频繁。

包括 MySQL 在内的绝大部分数据库系统都允许人们在创建数据表时把多个字段的组合定义为一个主键。但不管它是由单个字段还是由多个字段构成的，主键都必须满足以下条件：

- 主键必须是唯一的，任意两条数据记录里的主键字段绝不允许是同样的内容。
- 主键应该是紧凑的，理由有两个：

首先，为了加快寻址定位时的搜索速度，主键都必须有索引（即“主索引”）。主键字段越紧

凑，主索引上的管理工作效率就越高。因此，整数类型就要比非固定长度的字符串类型更适合作为主键。

其次，主键字段的内容几乎总是被用做另一个数据表里的外键；外键越紧凑，效率也就越高。

(人们为数据表建立各种关系的目的之一是为了避免冗余数据浪费存储空间，如果主键字段和/或外键字段占用的空间过大，就失去它们应有的意义了。)

在绝大多数的数据库系统上，使用 32 位或 64 位整数作为主键字段并让数据库系统为它们自动生成一组序列编号（1、2、3、…）的做法已经成为一种标准化的行为，这么做的好处是程序员和用户都无需再去考虑怎样才能为每一条新记录找到一个独一无二的主键值的问题了。

在 MySQL 数据库系统里，主键字段要用如下所示的语句来声明：

```
CREATE TABLE publishers
  (publID INT NOT NULL AUTO_INCREMENT,
   othercolumns ...,
   PRIMARY KEY (publID))
```

如果把上面这条 SQL 命令翻译成日常用语，它的意思是：整数字段 *publID* 不允许包含 *NULL* 值；它的内容将由数据库系统自动生成（除非在这个字段里明确地插入了另外一个具体的值）；这个字段将成为一个主键；MySQL 会为它自动创建一个索引以加快搜索速度。这一切将确保新输入的数据记录里的 *publID* 字段有一个唯一的值。

对于那些 *INSERT* 和/或 *UPDATE* 操作比较频繁的数据表，应该选用 *BIGINT* (64 位整数) 而不是 *INT* (32 位整数) 作为主键字段的数据类型。

注解 主键字段的名字对寻址搜索工作没有任何作用。把本书示例中的主键字段一般都命名为 *id* 或 *tablenameID*，偶尔会使用 *no* 或 *nr* (意思是“编号”) 组合，比如 *customerNr*。

8.8.2 外键

外键的任务是引用另一个数据表的某条记录，只不过这种引用关系是在构造数据库查询命令的时候而不是在声明数据表或数据列的时候定义的，如下所示：

```
SELECT titles.title, publishers.publName FROM titles, publishers
WHERE titles.publID = publishers.publID
ORDER BY title
```

上面这条命令将生成一份按字母表排序的书名清单，清单中的第 2 列将给出每本书的出版公司。这条查询命令的执行结果如下所示：

<i>title</i>	<i>publName</i>
Client/Server ...	Addison-Wesley
Definitive Guide ...	Apress
Linux	Addison-Wesley
Web Application ...	New Riders

上面这条语句中的关键是 *WHERE titles.publID=publishers.publID* 子句，正是它在 *titles* 数据表和 *publishers* 数据表之间建立起了完成这个查询所必需的关系。用查询命令来“链接”两个数据表还有其他一些办法，将在第 9 章讨论。

外键在数据表的定义声明里毫无特殊之处。在 MySQL 看来，外键不过是数据表里的又一个普通字段。在声明一个外键的时候，既不必使用特殊的保留字，也不必为它创建索引（在实际工作中，需要搜索外键字段值的场合几乎没有）。不过，在声明一个外键字段的时候有一点需要大家特别注意：千万不

要给外键字段加上 *AUTO_INCREMENT* 属性，因为外键字段指向哪条数据记录应该由用户来决定而不是由 MySQL 自动生成的序列编号来决定。还有一点也需要大家特别注意：外键字段的数据类型应该尽可能与主键字段保持一致；否则，*WHERE* 条件的求解过程将非常慢。

```
CREATE TABLE titles
  (othercolumns ...,
  publisherID INT NOT NULL)
```

是否需要给外键字段加上 *NOT NULL* 属性要根据具体情况来决定：在大多数场合下，应该给外键字段加上 *NOT NULL* 属性，以防止那些缺少必要数据的不完整记录被插入数据库；如果不关心出版公司而只是想把图书信息存入数据库，就不应该使用 *NOT NULL* 属性。

8.8.3 引用一致性（外键约束条件）

现在，如果从 *mylibrary* 数据库的 *authors* 数据表里把作者 Kofler 删掉，再去执行那些会涉及到 *Linux* 和 *Definitive Guide* 两本书的 SQL 查询命令时就会遇到这样一个问题：*rel_title_author* 数据表里编号为 1 的 *authID* 在 *authors* 数据表里已经不存在了。用数据库领域的语言来讲，这种情况叫做“数据库的引用一致性已遭到破坏”。

作为数据库的开发者，有责任确保这类事件不可能发生。因此，在删除一条记录之前，必须先去检查在其他数据表里是否存在对这条记录的引用。

完全依靠程序员来防止这类事件的发生并不总是那么保险，所以许多数据库系统都有一些用来保证引用一致性不会被破坏的规则。只要数据库里发生了变化，所谓的“外键约束条件”（一致性规则）就会去检查是否有数据表之间的交叉引用关系受到影响。根据外键的具体声明情况，数据记录删除操作的最终结果不外乎两种：一是不允许执行这一操作（返回一条出错信息）；二是把其他数据表里与之相关的数据记录也全部删掉。具体采用哪一种做法要由数据本身来决定。

MySQL 也提供了这样一种控制机制，但该机制目前仅适用于 InnoDB 数据表。下面的 SQL 代码片段演示了如何在声明一个外键字段时给它加上一致性规则（外键约束条件）：

```
CREATE TABLE titles
  (column1, column2, ...,
  publID INT,
  FOREIGN KEY (publID) REFERENCES publishers (publisherID)
```

上面这条命令把 *titles.publID* 字段明确地定义为一个指向主键 *publishers.publisherID* 的外键。在此基础上，还可以利用 *ON DELETE*、*RESTRICT* 和 *ON DELETE CASCADE* 等几个选项对数据库系统在它的引用一致性遭到破坏时应该如何响应做出进一步的设置。

```
CREATE TABLE titles
  (column1, column2, ...,
  publID INT,
  FOREIGN KEY (publisherID) REFERENCES publishers (publID)
  )
```

上面这条命令把 *titles.publID* 字段明确地定义为一个指向主键 *publishers.publisherID* 的外键。一致性规则对这两个数据表将产生以下影响：

- 在 *titles* 数据表里，既不能使用一个在 *publishers* 数据表里不存在的 *publID* 编号去插入一本新图书，也不能在 *titles* 数据表里把现有的 *publID* 编号改为一个在 *publishers* 数据表里不存在的值。
- 在 *publishers* 数据表里，如果某个 *publID* 编号已被 *titles* 数据表引用，将不能删除它或者是用

UPDATE 命令改变它。

上面这条命令还会影响到插入/删除数据记录时的操作顺序：如果想把一本新书存入 *titles* 数据表，但它的出版公司在 *publishers* 数据表里没有记录，必须先在 *publishers* 数据表里插入出版公司的记录，然后才能在 *titles* 数据表里插入新书的记录。如果想从 *publishers* 数据表里删除一个出版公司，必须先把该公司出版的图书从 *titles* 数据表里全部删除。其他操作不受影响。

1. 语法

下面是为外键字段 *table1.column1* 设置外键约束条件的基本语法：

```
FOREIGN KEY [name] (column1) REFERENCES table2 (column2)
  [ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT}]
  [ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT}]
```

根据如上所示的语法，外键约束条件可以被命名为 *name*（可选）；受外键约束条件约束的外键是 *table1.column1*；受外键约束条件约束的另一方是 *table2.column2* 字段，该字段必须配有索引。（在许多场合，*column2* 字段会是 *table2* 数据表的主键，但并非必须如此。）

此外，外键约束条件中的 *table1* 和 *table2* 允许是同一个数据表，此时 *column1* 字段引用的将是同一个数据表里的 *column2* 字段。*mylibrary* 数据库里的 *categories* 数据表就属于这种情况，它的 *parentID* 字段引用了同一个数据表里的 *CatID* 字段，而这将形成层次关系。

2. 外键约束条件的作用

ON DELETE 子句是可选的，它定义了试图从 *table2* 数据表里删除一条被 *table1* 数据表引用的记录时数据表驱动器应该如何采取行动。这里有以下 4 种选择。

RESTRICT: 这是默认行为。*DELETE* 命令将引起一个错误，但那条记录不会被删除。（*DELETE* 命令引起了一个错误并不意味着正在执行的事务会半途而废，它只表明 *DELETE* 命令没有被执行而已。必须像往常一样用 *COMMIT* 或 *ROLLBACK* 命令来终止这个事务。）

SET NULL: *table2* 数据表里的记录将被删除，*table1* 数据表里受其影响的所有记录的 *column1* 字段将全部被设置为 *NULL*。这条规则的生效前提是 *table1.column1* 字段可以取值为 *NULL*。以 *titles/publishers* 数据表为例，这意味着如果从 *publishers* 数据表删除了 *x* 出版公司，*titles* 数据表里由这家出版公司出版的所有图书的 *publID* 字段都将被设置为 *NULL*。

CASCADE: *table2* 数据表里的记录将被删除，*table1* 数据表里受其影响的所有记录将同时被删除。

以 *titles/publishers* 数据表为例，这意味着如果从 *publishers* 数据表删除了 *x* 出版公司，*titles* 数据表里由这家出版公司出版的所有图书也将被删除。

NO ACTION: *table2* 数据表里的记录将被删除，对 *table1* 数据表不采取任何行动。也就是说，引用一致性遭到破坏是可以忍受的。这个动作没有什么实用意义，因为它与没有定义外键约束条件时的情况完全一样。

可选的 *ON DELETE* 子句也提供了上述 4 种动作供人们选择（默认行为也是 *RESTRICT*）。*UPDATE* 规则针对的是 *table2* 数据表里现有记录的修改操作。*RESTRICT*、*SET NULL* 和 *NO ACTION* 3 个选项在 *ON UPDATE* 子句里的动作含义与它们在 *ON DELETE* 子句里的相同。

CASCADE 选项在 *ON UPDATE* 子句里的动作含义与它在 *ON DELETE* 子句里的有所不同：对 *table2* 数据表里的 *column2* 字段的修改将被传递到 *table1* 数据表里的 *column1* 字段。以 *titles/publishers* 数据表为例，这意味着如果在 *publishers* 数据表里对 *x* 出版公司的 *publID* 进行了修改，*titles* 数据表里由这家出版公司出版的所有图书的 *publID* 字段将全部被刷新为那个新 *publID* 值。

注解 外键约束条件并不能阻止删除整个数据表。比如说，完全可以执行`DROP TABLE publishers`命令，虽然这破坏了引用一致性。

3. 设置外键约束条件的前提

设置和使用外键约束条件必须满足以下几个前提。如果这些条件没有得到满足，外键约束条件将不会生效，MySQL 会返回一条出错信息，其内容通常是 *Error 1005: Can't create table xxx (errno: 150)*（无法创建 xxx 数据表）。（导致 MySQL 返回这条出错信息的原因有时相当微不足道；比如在输入数据列的名字时出现打字错误。）

- 数据列 `table1.column1` 和 `table2.column2` 都必须配有一个普通索引。`FOREIGN KEY` 语句不会自动创建这个索引，必须在 `CREATE TABLE` 命令里或随后的 `ALTER TABLE` 命令里自行创建。`table2.column2` 字段通常是 `table2` 数据表的主键字段，但并非必须如此。
如果使用的是多字段索引 (`INDEX(columnA, columnB)`)，外键约束条件中的关键字字段必须出现在第一个字段。否则，就必须为关键字字段再多创建一个索引。
- 数据列 `table1.column1` 和 `table2.column2` 的数据类型必须匹配到无需进行数据类型转换就可以直接进行比较的程度。最有效率的做法是把这两个字段都声明为 `INT` 或 `BIGINT` 类型——注意，此时这两个数据列的正负符号类型必须相同（或者都是 `SIGNED`，或者都是 `UNSIGNED`）。
- 如果还定义了可选规则 `ON DELETE/UPDATE SET NULL`，就必须允许 `table1.column1` 字段取值为 `NULL`。
- 外键约束条件必须从一开始就得到满足：如果数据表里已经有了一些数据，就可能会有个别记录不能满足外键约束条件的要求。如果真是这样，`ALTER TABLE` 命令将返回一个 1216 号错误 *A foreign key constraint fails*（“外键约束条件创建失败”）。换句话说，必须先改正那些不符合要求的个别记录才能成功创建外键约束条件。

提示 如果在创建外键约束条件时遇到问题，可以用 `SHOW INNODB STATUS` 命令去查看更详细的出错原因信息。

4. 找出不符合外键约束条件的数据记录

在看到 1216 号错误时，肯定会马上想到这样一个问题：“怎样才能把不符合要求的记录找出来？”下面这条简单的 `SELECT` 命令将从 `titles` 数据表里把 `titles.publID` 字段的值在 `publisher.publID` 字段里没有对应值的所有记录全部查找出来：

```
SELECT titleID, publID FROM titles
WHERE publID NOT IN (SELECT publID FROM publishers)
```

<u>titleID</u>	<u>publID</u>	<u>publishers.publID</u>
66	99	NULL

根据上面的查询结果，`titleID=66` 的图书对应着 `publisher` 数据表里 `publID=99` 的出版公司，可是 `publisher` 数据表里根本没有这样一个 ID 编号。解决这个问题的办法有两种：其一是补上一个 `publID=99` 的出版公司；其二是删除 `titleID=66` 的图书。（还可以在 `titles` 数据表里把不符合要求的 `publID` 字段设置为 `NULL`，但在实际工作中往往不允许对两个链接着的数据表这样做。）

5. 删除外键约束条件

如果想删除一个外键约束条件，可以执行 `ALTER TABLE` 命令：

```
ALTER TABLE tablename DROP FOREIGN KEY foreign_key_id
```

可以用 *SHOW CREATE TABLE* 命令查出需要删除的索引的 *foreign_key_id*。注意，如果此时正使用着镜像功能，删除外键约束条件可能会引起问题。这是因为外键约束条件在镜像系统和原始系统上的名字可能不一样。

6. 临时禁用外键约束条件检查

SET foreign_key_checks=0 命令可以暂时关闭对外键约束条件的自动检查功能。有时，为了加快大数据表备份的读入速度，关闭这种检查还是有必要的。*SET foreign_key_checks=1* 命令将重新启用这些规则。在这些规则被暂时禁用期间，不会检查对数据库的改动。如果在此期间有违反了这些规则的改动，因此而导致的错误将不会被自动纠正。

7. *mylibrary* 数据库上的外键约束条件

mylibrary 数据库完全由 InnoDB 数据表构成。笔者为这些数据表之间的所有关系都定义了必要的外键约束条件，如表 8-25 所示。

表 8-25 *mylibrary* 数据库上的外键和被引用键

外键	被引用键
<i>titles.publID</i>	<i>publishers.publID</i>
<i>titles.langID</i>	<i>languages.langID</i>
<i>titles.catID</i>	<i>categories.catID</i>
<i>categories.parentCatID</i>	<i>categories.catID</i>
<i>rel_title_author.titleID</i>	<i>titles.titleID</i>
<i>rel_title_author.authID</i>	<i>authors.authID</i>

8.9 索引

为了在一个数据表里检索某个特定的记录，或者提取一系列数据记录以生成一个排序表格，MySQL 必须把这个数据表里的所有数据记录都搜索一遍。下面是一些有关的 *SELECT* 命令（有关细节见第 9 章）：

```
SELECT column1, column2 ... FROM table WHERE column3=12345
SELECT column1, column2 ... FROM table ORDER BY column3
SELECT column1, column2 ... FROM table WHERE column3 LIKE 'Smith%'
SELECT column1, column2 ... FROM table WHERE column3 > 2000
```

如果数据表的体积比较庞大，在遇到这样一些查询的时候性能就会显著下降。有个简单的办法可以解决数据表性能下降的问题：为查询操作所涉及的数据列（在上面的例子中，就是 *column3*）创建并使用一个索引。

索引是一种特殊的文件（InnoDB 数据表上的索引是表空间的一个组成部分），它们包含着对数据表里所有记录的引用指针。（数据库索引与这本书的索引有着同样的功用。如果想知道对某个特定话题的讨论都在书中的哪些地方出现过，书后的索引可以让读者无需把整本书从头到尾地翻看一遍就能把它们查出来，既省时间又省事。）

注意 索引不是万能灵药！它们可以加快数据检索操作，但会使数据修改操作变慢。每修改一条数据记录，索引就必须刷新一次。为了在某种程度上弥补这一缺陷，许多 SQL 命令都有一个 *DELAY_KEY_WRITE* 选项。这个选项的作用是暂时制止 MySQL 在该命令每插入一条新记录和每

修改一条现有记录之后立刻对索引进行刷新，对索引的刷新将等到全部记录插入/修改完毕之后再进行。在需要把许多新记录插入某个数据表的场合，*DELAY_KEY_WRITE*选项的作用将非常明显。

索引的另一个明显缺陷是它们会在硬盘上占用相当大的空间。因此，应该只为最经常查询和最经常排序的数据列建立索引。注意，如果某个数据列包含着许多重复的内容，为它建立索引就没有太大的实际效果——在遇到这类问题的时候，首先应该对照在8.5一节里给出的有关原则去检查数据库设计方案是否最优。

从理论上讲，完全可以为数据表里的每一个字段分别创建一个索引，但 MySQL 把同一个数据表里的索引总数限制为 16 个。（MySQL 还允许为多个字段的组合创建索引，这种索引对于涉及多个字段的查询/排序操作——如 *WHERE country='Austria' AND city='Graz'*——特别有帮助。）

1. InnoDB数据表的索引

与 MyISAM 数据表相比，索引对 InnoDB 数据表的重要性要大得多。在 InnoDB 数据表上，索引不仅会在搜索数据记录时发挥作用，还是数据行级锁定机制的基础。“数据行级锁定”的意思是在事务操作的执行过程中锁定正在被处理的个别记录，不让其他用户进行访问。这种锁定将影响到（但不限于）*SELECT...LOCK IN SHARE MODE*、*SELECT...FOR UPDATE* 命令以及 *INSERT*、*UPDATE* 和 *DELETE* 命令（在第 10 章将对 MySQL 的事务处理机制做进一步讨论）。

出于效率方面的考虑，InnoDB 数据表的数据行级锁定机制实际发生在它们的索引上，而不是数据表自身上。显然，数据行级锁定机制只有在有关的数据表有一个合适的索引可供锁定的时候才能发挥效力。

2. 限制

- 如果 *WHERE* 子句的查询条件里有不等号 (*WHERE column != ...*)，MySQL 将无法使用索引。
- 类似地，如果 *WHERE* 子句的查询条件里使用了函数 (*WHERE DAY(column) = ...*)，MySQL 也将无法使用索引。
- 在 *JOIN* 操作中（需要从多个数据表提取数据时），MySQL 只有在主键和外键的数据类型相同时才能使用索引。
- 如果 *WHERE* 子句的查询条件里使用了比较操作符 *LIKE* 和 *REGEXP*，MySQL 只有在搜索模板的第一个字符不是通配符的情况下才能使用索引。比如说，如果查询条件是 *LIKE 'abc%'*，MySQL 将使用索引；如果查询条件是 *LIKE '%abc'*，MySQL 将不使用索引。
- 在 *ORDER BY* 操作中，MySQL 只有在排序条件不是一个查询条件表达式的情况下才使用索引。（虽然如此，在涉及多个数据表的查询里，即使有索引可用，那些索引在加快 *ORDER BY* 操作方面也没什么作用。）
- 如果某个数据列里包含着许多重复的值，就算为它建立了索引也不会有很好的效果。比如说，如果某个数据列里包含的净是些诸如“0/1”或“Y/N”等值，就没有必要为它创建一个索引。

8.9.1 普通索引、唯一索引和主索引

1. 普通索引

普通索引（由关键字 *KEY* 或 *INDEX* 定义的索引）的唯一任务是加快对数据的访问速度。因此，应该只为那些最经常出现在查询条件（*WHERE column=...*）或排序条件（*ORDER BY column*）中的数据列创建索引。只要有可能，就应该选择一个数据最整齐、最紧凑的数据列（如一个整数类型的数据

列) 来创建索引。

2. 唯一索引 (Unique索引)

普通索引允许被索引的数据列包含重复的值。比如说, 因为人有可能同名, 所以同一个姓名在同一个“员工个人资料”数据表里可能出现两次或更多次。

如果能确定某个数据列将只包含彼此各不相同的值, 在为这个数据列创建索引的时候就应该用关键字 *UNIQUE* 把它定义为一个唯一索引。这么做有两个好处: 一是简化了 MySQL 对这个索引的管理工作, 这个索引也因此而变得更有效率; 二是 MySQL 会在有新记录插入数据表时, 自动检查新记录的这个字段的值是否已经在某个现有记录的这个字段里出现过了; 如果是, MySQL 将拒绝插入那条新记录。也就是说, 唯一索引可以保证数据记录的唯一性。事实上, 在许多场合, 人们创建唯一索引的目的往往不是为了提高访问速度, 而只是为了避免数据出现重复。

3. 主索引

在前面已经反复多次地强调过: 必须为主键字段创建一个索引, 这个索引就是所谓的“主索引”。主索引与唯一索引的区别是, 前者在定义时使用的关键字是 *PRIMARY* 而不是 *UNIQUE*。

4. 外键索引

如果为某个外键字段定义了一个外键约束条件(一致性规则, 参见 8.8 节), MySQL 就会定义一个内部索引来帮助自己以最有效率的方式去管理和使用外键约束条件。

5. 复合索引

索引可以覆盖多个数据列, 如像 *INDEX(columnA, columnB)* 索引。这种索引的特点是 MySQL 可以有选择地使用一个这样的索引。如果查询操作只需要用到 *columnA* 数据列上的一个索引, 就可以使用复合索引 *INDEX(columnA, columnB)*。不过, 这种用法仅适用于在复合索引中排列在前的数据列组合。比如说, *INDEX(A, B, C)* 可以当做 *A* 或 *(A, B)* 的索引来使用, 但不能当做 *B*、*C* 或 *(B, C)* 的索引来使用。

6. 索引的长度限制

在为 *CHAR* 和 *VARCHAR* 类型的数据列定义索引时, 可以把索引的长度限制为一个给定的字符个数(这个数字必须小于这个字段所允许的最大字符个数)。这么做的好处是可以生成一个尺寸比较小、检索速度却比较快的索引文件。在绝大多数应用里, 数据库中的字符串数据大都以各种各样的名字为主, 把索引的长度设置为 10~15 个字符已经足以把搜索范围缩小到很少的几条数据记录了。

在为 *BLOB* 和 *TEXT* 类型的数据列创建索引时, 必须对索引的长度做出限制: MySQL 所允许的最大索引长度是 255 个字符。

8.9.2 全文索引

文本字段上的普通索引只能加快对出现在字段内容最前面的字符串(也就是字段内容开头的字符)进行的检索操作。如果字段里存放的是由几个、甚至是许多个单词构成的大段文字, 普通索引就没什么作用了。这种检索往往以 *LIKE %word%* 的形式出现, 这对 MySQL 来说很复杂, 如果需要处理的数据量很大, 响应时间就会很长。

这类场合正是全文索引(full-text index)可以大显身手的地方。在生成这种类型的索引时, MySQL 将把在文本中出现的所有单词创建为一份清单, 查询操作将根据这份清单去检索有关的数据记录。全文索引既可以随数据表一同创建, 也可以等日后有必要时再使用下面这条命令添加:

```
ALTER TABLE tablename ADD FULLTEXT(column1, column2)
```

有了全文索引, 就可以用 *SELECT* 查询命令去检索那些包含着一个或多个给定单词的数据记录了。下

下面是这类查询命令的基本语法：

```
SELECT * FROM tablename  
WHERE MATCH(column1, column2) AGAINST('word1', 'word2', 'word3')
```

上面这条命令将把 *column1* 和 *column2* 字段里有 *word1*、*word2* 和 *word3* 的数据记录全部查找出。

注解 InnoDB数据表不支持全文索引。

第10章将对与全文检索功能有关的SQL语法以及各种应用示例做详细的介绍和分析。

8.9.3 查询和索引的优化

只有当数据库里已经有了足够多的测试数据时，它的性能测试结果才有实际参考价值。如果在测试数据库里只有几百条数据记录，它们往往会在执行完第一条查询命令之后就被全部加载到内存里，这将使后续的查询命令都执行得非常快——不管有没有使用索引。只有当数据库里的记录超过了1000条、数据总量也超过了MySQL服务器上的内存总量时，数据库的性能测试结果才有意义。

在不确定应该在哪些个数据列上创建索引的时候，人们从 *EXPLAIN SELECT* 命令那里往往可以获得一些帮助。这其实只是简单地给一条普通的 *SELECT* 命令加上一个 *EXPLAIN* 关键字作为前缀而已。有了这个关键字，MySQL 将不是去执行那条 *SELECT* 命令，而是去对它进行分析。MySQL 将以表格的形式把查询的执行过程和用到的索引（如果有的话）等信息列出来。

在 *EXPLAIN* 命令的输出结果里，第1列是从数据库读取的数据表的名字，它们按被读取的先后顺序排列。*type* 列指定了本数据表与其他数据表之间的关联关系 (*JOIN*)。在各种类型的关系当中，效率最高的是 *system*，然后依次是 *const*、*eq_ref*、*ref*、*range*、*index* 和 *All*。（*All*的意思是：对于上一级数据表里的每一条记录，这个数据表里的所有记录都必须被读取一遍——这种情况往往可以用一个索引来避免。）

possible_keys 数据列给出了 MySQL 在搜索数据记录时可以选用的各个索引。*key* 数据列是 MySQL 实际选用的索引，这个索引按字节计算的长度在 *key_len* 数据列里给出。比如说，对于一个 *INTEGER* 数据列的索引，这个字节长度将是 4。如果用到了复合索引，在 *key_len* 数据列里还可以看到 MySQL 具体使用了它的哪些部分。作为一般规律，*key_len* 数据列里的值越小越好（意思是更快）。

ref 数据列给出了关联关系中另一个数据表里的数据列的某个名字。*row* 数据列是 MySQL 在执行这个查询时预计会从这个数据表里读出的数据行的个数。*row* 数据列里的所有数字的乘积可以让我们大致了解这个查询需要处理多少种组合。

最后，*extra* 数据列提供了与 *JOIN* 操作有关的更多信息，比如说，如果 MySQL 在执行这个查询时必须创建一个临时数据表，就会在 *extra* 列看到 *using temporary* 字样。

提示 *EXPLAIN* 命令提供的信息很有用，但想把它们看明白却必须具备一定的 MySQL 和数据库经验才行。可以在以下网址处的 MySQL 文档里找到更多这方面的信息：

<http://dev.mysql.com/doc/mysql/en/query-speed.html>

<http://dev.mysql.com/doc/mysql/en/explain.html>。

在以下网址可以找到一篇 OpenOffice 格式的 MySQL 速度优化指南：

<http://dev.mysql.com/tech-resources/presentations/>

<http://dev.mysql.com/Downloads/Presentations/OSCON-2004.sxi>。

1. 示例1

下面这个查询将生成一份所有图书和它们的所有作者的不排序名单。所有的 ID 数据列都配有主索引。

```
USE mylibrary
EXPLAIN SELECT * FROM titles, rel_title_author, authors
  WHERE rel_title_author.authID = authors.authID
    AND rel_title_author.titleID = titles.titleID
```

table	type	key	key_len	ref	rows	Extra
titles	ALL	authName	60	NULL	53	Using index
rel_title_author	ref	authID	4	authors.authID	1	Using index
authors	eq_ref	PRIMARY	4	rel_title_author.titleID	1	

从上面的 *EXPLAIN* 命令执行结果可以看出：*titles* 数据表里的所有记录将最先被读取，读取时使用的索引是 *authName*。（这实际上并没有必要，因为这条 *SELECT* 命令不要求对查询结果进行排序。）接下来，在 *rel_title_author* 数据表的 *authID* 索引和 *authors* 数据表的主索引的帮助下，与另外两个数据表的链接被建立了起来。这些数据表都有效果最佳的索引，整个查询的每一步都有索引可用。

为了节约篇幅，这里省略了 *EXPLAIN* 命令执行结果中的一些数据列，其中包括 *possible_keys* 数据列。这个数据列的内容是在各个查询步骤里可供选用的索引的清单。*key* 数据列指出 MySQL 实际选用的索引是哪一个。

2. 示例2

下面这个查询将生成一份由一家特定的出版公司出版的所有图书（以及它们的作者）的清单。这份清单将按书名排序。和示例 1 里的情况一样，所有的 ID 数据列都配有索引。此外，在 *titles* 数据表里，*title* 和 *pubID* 数据列也有索引：

```
EXPLAIN SELECT title, authName
  FROM titles, rel_title_author, authors
 WHERE titles.publID=2
   AND titles.titleID = rel_title_author.titleID
   AND authors.authID = rel_title_author.authID
 ORDER BY title
```

table	type	key	key_len	ref	rows	Extra
titles	ref	publIDIndex	5	const	4	Using where; Using filesort
rel_title_author	ref	PRIMARY	4	titles.titleID	2	Using index
authors	eq_ref	PRIMARY	4	rel_title_author.authID	1	

从上面的 *EXPLAIN* 命令执行结果可以看出：数据表都有效果最佳的索引，整个查询的每一步都有索引可用。有一点很有意思：虽然 *title* 数据列上有一个索引，但这份书名清单（*ORDER BY title*）显然是在外部完成排序的。这或许是因为 *title* 记录是根据查询条件 *publID=2* 被最先选取出来的，因而无法再利用 *title* 索引对它进行排序的缘故。

3. 示例3

这个示例使用的 *SELECT* 查询命令与示例 2 一样，但这一次 *titles.publID* 数据列上没有索引。结果是 *titles* 数据表里总共 53 条记录都必须被读取一遍。这里虽然用到了 *authName* 索引，但它对加快查询没有任何帮助。*Extra* 数据列里的 *Using temporary* 表明，MySQL 为了保存中间结果而创建了一个

临时数据表。

table	type	key	key_len	ref	rows	Extra
titles	index	authName	60	NULL	53	Using index; Using temporary; Using filesort
rel_title_author	ref	PRIMARY	4	titles.titleID	4	Using index
authors	eq_ref	PRIMARY	4	rel_title_author.authID	4	Using where

8.10 视图

视图（view）使得人们可以为一个或多个数据表定义一个特殊的表现形式。视图在行为上与数据表没有什么区别——可以使用 *SELECT* 查询命令去查询数据，还可以（但要取决于视图的具体定义）使用 *INSERT*、*UPDATE* 和 *DELETE* 命令修改数据。

在 MySQL 5.0 及以后的版本里都可以使用视图。本节内容的讨论前提是对于 *SELECT* 命令（参见第 10 章）和各种 MySQL 访问权限都已经有了足够的了解。注意，phpMyAdmin 2.6（它是本书写作之时的最新版本）还不能用来处理视图。

人们创建和使用视图的基本理由有两个：

□ **安全**。有时候，可能不想让某个特定的数据库用户有权对某个数据表进行任何访问。这方面的一个典型例子是公司里的“员工个人资料”数据表（这里不妨假设这个数据表里存放着所有员工的信息），有些数据（如姓名和电话号码）应该让全体用户都可以访问，但另外一些数据（如生日和薪水）就不应该是这样了。

解决方案是，把应该让全体用户都可以访问的那些数据列创建为一个视图，然后通过设置 MySQL 访问权限让某些用户只能访问这个视图、但不能访问这个视图背后的数据表。

□ **方便**。在许多应用程序里，人们经常需要执行同样的查询去根据某种要求从一个或多个数据表里收集数据。让那么多的用户或程序员重复输入同一个复杂的 *SELECT* 命令显然不是最佳办法，于是，作为数据库管理员，可以定义一个视图为大家提供方便。

视图的定义

视图相当于一个虚拟的数据表，而这个虚拟数据表的内容是某个 *SELECT* 查询命令的执行结果。有了这个认识，视图需要用一条 *SELECT* 命令来定义这件事也就不足为怪了。下面两个示例为 *mylibrary* 数据表定义了两个视图并给出了两个视图里的 5 条记录。

```

CREATE VIEW v1 AS
  SELECT titleID, title, subtitle FROM titles
  ORDER BY title, subtitle
  SELECT * FROM v1 LIMIT 5
    titleID  title          subtitle
      11  A Guide to the SQL Standard    NULL
      52  A Programmer's Introduction ...  NULL
      19  Altid den där Annette       NULL
      51  Anklage Vatermord           Der Fall Philipp Halsmann
      78  Apache Webserver 2.0         Installation, ...
CREATE VIEW v2 AS
  SELECT title, publname, catname FROM titles, publishers, categories
  WHERE titles.publid=publishers.publid

```

```

AND titles.catID = categories.catID
AND langID=2
SELECT * FROM v2 ORDER BY title LIMIT 5

title          publname      catname
Anklage Vatermord Zsolnay       Literature and fiction
Apache Webserver 2.0 Addison-Wesley Computer books
CSS-Praxis      Galileo        Computer books
Ein perfekter Freund Diogenes Verlag Literature and fiction
Excel 2000 programmieren Addison-Wesley Programming

```

注意，必须具备 *Create View* (创建视图) 权限才能执行 *CREATE VIEW* 命令。第 11 章将对 MySQL 中的访问权限以及它们的管理问题进行讨论。

1. 在视图里修改数据记录

能不能在某个视图里使用 *INSERT*、*UPDATE* 和 *DELETE* 命令（或者说这个视图是不是可刷新的）要取决于当初用来定义这个视图的 *SELECT* 命令。可刷新的视图需要满足以下几个条件：

- 当初用来定义视图的 *SELECT* 命令不得包含 *GROUP BY*、*DISTINCT*、*LIMIT*、*UNION* 或者 *HAVING* 等子命令。
- 如果某个视图里的数据来自一个以上的数据表，那它几乎总是不可刷新的。
- 视图应该包含主键索引、唯一索引、外键约束条件所涉及的全部数据列。如果视图里没有或缺少这样的数据列，就将由 MySQL 选项 *updateable_views_with_limit* 来决定是允许刷新并同时返回一条警告消息（默认设置）、还是不允许刷新并触发一个错误（设置为 0）。

2. 视图选项

下面是 *CREATE VIEW* 命令的完整语法：

```

CREATE [OR REPLACE] [ALGORITHM = UNDEFINED | MERGE | TEMPTABLE]
VIEW name [(columnlist)] AS select command
[WITH [CASCADED | LOCAL] CHECK OPTION]

```

下面是对有关选项的简单解释：

- *ON REPLACE*: 把一个现有视图替换为新视图，并且不返回出错消息。
- *ALGORITHM*: 新视图的内部表示形式。不过，MySQL 文档在笔者写本小节时还没有对这个选项做出具体定义。在默认的情况下，MySQL 总是把这个选项设置为 *UNDEFINED* (可以通过 *SHOW CREATE TABLE viewname* 命令来查看)。
- *WITH CHECK OPTION*: 只有当用来创建这个视图的 *SELECT* 命令的 *WHERE* 条件满足时才允许对这个视图里的记录进行修改。*WITH CHECK OPTION* 选项只在视图允许刷新的时候才起作用。

变形 *WITH LOCAL CHECK OPTION* 将影响从其他视图派生出来的视图(这是允许的)。*LOCAL* 的意思是只考虑本条 *CREATE VIEW* 命令的 *WHERE* 条件，不考虑父级视图的 *WHEER* 条件。

变形 *WITH CASCADE CHECK OPTION* 的含义刚好与 *WITH LOCAL CHECK OPTION* 相反：本视图以及所有父级视图的 *WHERE* 条件都必须考虑。如果既没有给出 *CASCADE*、也没有给出 *LOCAL*，默认设置将是 *CASCADE*。

3. 查看视图的定义

就像 *SHOW CREATE TABLE name* 命令可以查看某个数据表的创建命令的 SQL 代码那样，*SHOW CREATE VIEW name* 命令可以对视图做同样的事情。必须具备 *Create View* (创建视图) 权限才能执行 *CREATE VIEW* 命令。

```
SHOW CREATE VIEW v1

CREATE ALGORITHM=UNDEFINED VIEW `mylibrary`.`v1` AS
SELECT `mylibrary`.`titles`.`titleID`,
       `mylibrary`.`titles`.`title`,
       `mylibrary`.`titles`.`subtitle`
  FROM `mylibrary`.`titles`
 ORDER BY `mylibrary`.`titles`.`title`,
          `mylibrary`.`titles`.`subtitle`
```

4. 删除视图

SHOW TABLES 命令将返回一个全体数据表和视图的清单，但不能使用 *DROP TABLE* 命令去删除视图。必须使用 *DROP VIEW viewname* 命令来做这件事。

8.11 示例数据库 *mylibrary* (图书管理)

本章以 *mylibrary* 数据库为例对数据库设计工作的几个方面展开了讨论，但一直没有机会对这个样板数据库进行总结，所以可能会有部分读者对它的各有关细节，如数据表、字段、索引、数据类型等，还没有一个全面的了解。因此，本节对 *mylibrary* 数据库的所有属性进行汇总。这些属性的汇总通常被称为数据库设计方案或数据库大纲 (database schema)。

图 8-2 是 *mylibrary* 数据库的数据表关联/引用关系图。虽然图中没有标出数据列的数据类型和建有索引的字段，但那并不妨碍用户一目了然地看出各数据表的主键字段和数据表之间的关联/引用关系。(本书的数据库关系图都是用 OpenOffice 套装软件中的 Query Designer 工具绘制的。)

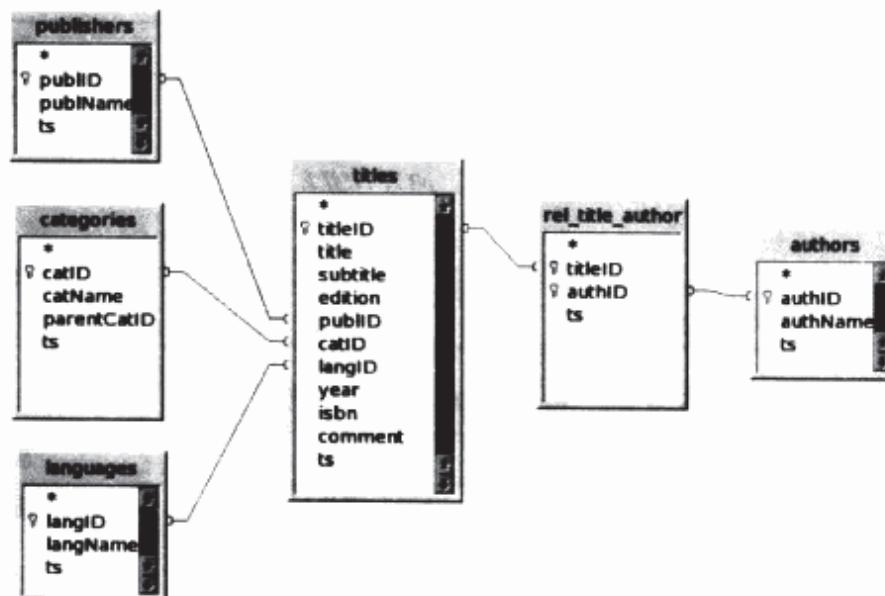


图 8-2 *mylibrary* 数据库的数据表关联/引用关系图

注解 在本书的配套文件当中，可以找到一个包含着 *mylibrary* 数据库的完整定义和一些测试用数据的 *.sql 文件。如果想把这个文件读入系统，请先在 phpMyAdmin 程序里创建一个空白的数据库，并把它的默认字符集设置为 *Latin1*，然后切换到 SQL 页面去加载这个 *.sql 文件。当然，也可以用下面两条命令来创建这个数据库：

```
> mysqladmin -u root -p create mylibrary
> mysql -u root -p mylibrary < mylibrary.sql
```

8.11.1 数据库的属性

mylibrary 数据库及其中的所有文本字段使用 *Latin1* 作为默认字符集，使用 *latin1_general_ci* 作为默认排序方式。

8.11.2 数据表的属性

mylibrary 数据库里的数据表都是 Innodb 数据表，这些数据表的字段、数据类型、属性和索引分别在表 8-26~表 8-31 中给出。本章前面的内容已经对这些数据表里的大部分数据列做了介绍，这里就不再多做解释了。注意，每个数据表都有一个 *ts* 数据列，MySQL 将把数据记录的最后一次修改时间自动记录到这个数据列里。如果想通过 ODBC/ADO 程序来访问某个数据库，就必须在这个数据库的数据表里定义一个 *TIMESTAMP* 数据列，*mylibrary* 数据库就属于这种情况。

表 8-26 *authors* 数据表的属性

字 段	数据类型	属 性	索引、外键约束条件
<i>authID</i>	<i>INT</i>	<i>NOT NULL</i> <i>AUTO_INCREMENT</i>	主键
<i>authName</i>	<i>VARCHAR(60)</i>		键
<i>ts</i>	<i>TIMESTAMP</i>		

表 8-27 *categories* 数据表的属性

字 段	数据类型	属 性	索引、外键约束条件
<i>catID</i>	<i>INT</i>	<i>NOT NULL</i> <i>AUTO_INCREMENT</i>	主键
<i>catName</i>	<i>VARCHAR(60)</i>	<i>NOT NULL</i>	键
<i>parentCatID</i>	<i>INT</i>		键
<i>ts</i>	<i>TIMESTAMP</i>		外键 <i>categories.catID</i>

表 8-28 *languages* 数据表的属性

字 段	数据类型	属 性	索引、外键约束条件
<i>langID</i>	<i>INT</i>	<i>NOT NULL</i> <i>AUTO_INCREMENT</i>	主键
<i>langName</i>	<i>VARCHAR(60)</i>	<i>NOT NULL</i>	键
<i>ts</i>	<i>TIMESTAMP</i>		

表 8-29 *publishers* 数据表的属性

字 段	数据类型	属 性	索引、外键约束条件
<i>publID</i>	<i>INT</i>	<i>NOT NULL</i> <i>AUTO_INCREMENT</i>	主键
<i>publName</i>	<i>VARCHAR(60)</i>	<i>NOT NULL</i>	键
<i>ts</i>	<i>TIMESTAMP</i>		

表 8-30 *rel_title_author* 数据表的属性

字 段	数据类型	属 性	索引、外键约束条件
<i>authID</i>	<i>INT</i>	<i>NOT NULL</i>	主键、外键 <i>authors.authID</i>
<i>titleID</i>	<i>INT</i>	<i>NOT NULL</i>	主键、键、外键 <i>titles.titleID</i>
<i>ts</i>	<i>TIMESTAMP</i>		

表 8-31 *titles* 数据表的属性

字 段	数据类型	属 性	索引、外键约束条件
<i>titleID</i>	<i>INT</i>	<i>NOT NULL</i> <i>AUTO_INCREMENT</i>	主键
<i>title</i>	<i>VARCHAR(100)</i>	<i>NOT NULL</i>	键
<i>subtitle</i>	<i>VARCHAR(100)</i>		
<i>edition</i>	<i>TINYINT</i>		
<i>publID</i>	<i>INT</i>		键、外键 <i>publishers.publID</i>
<i>catID</i>	<i>INT</i>		键、外键 <i>categories.catID</i>
<i>langID</i>	<i>INT</i>		键、外键 <i>languages.langID</i>
<i>year</i>	<i>INT</i>		
<i>isbn</i>	<i>VARCHAR(20)</i>		
<i>comment</i>	<i>VARCHAR(255)</i>		
<i>ts</i>	<i>TIMESTAMP</i>		

8.12 示例数据库 *myforum* (网上论坛)

本书中的大部分示例使用的都是 *mylibrary* 数据库，本章里谈论的也一直是它。可是，一个如 *mylibrary* 这么小的数据库是很难把所有的 SQL 概念都演示清楚的，所以这里还准备了两个示例数据库，一个是将在本节介绍的 *myforum* 数据库，另一个是将在 8.13 节介绍的 *exceptions* 数据库。

myforum 数据库取材于一个真实的 PHP 应用程序，但由于本书的篇幅有限，所以没有在这本书里对它的程序代码进行介绍和分析，毕竟这里关注的只是这个数据库的设计方案。*myforum* 数据库包含 1000 多条的数据记录，非常适合用来测试 MySQL 的全文检索功能（详见第 10 章）。

exceptions 数据库没有配套的程序代码，提供这个示例数据库的目的是为了让用户能够直观地对一些不常用的 MySQL 数据类型及各种排序方式进行测试。

8.12.1 讨论组数据库：*myforum*

留言簿、讨论组和其他一些能够让用户创建各种文本、在网站上发表言论的 MySQL 应用程序很受人们的欢迎。数据库是讨论组的基础之一，而示例数据库 *myforum* 可以让用户了解这一切是如何实现的。这个数据库由 3 个数据表构成。

- *forums* 数据表：这个数据表的内容是一份全体讨论组的清单，每个讨论组都使用一种特定的工作语言（英语或德语）。
- *users* 数据表：这个数据表的内容是一份已经在数据库里注册、因而允许在讨论组里发表言论

的全体用户名单，里面记载着每一位注册用户的登录名、密码和电子邮件地址。这个数据表的 *authent* 列包含着一个随机的字符串，这个字符串将在有新用户注册时通过电子邮件发送给该用户，只有在该用户单击了电子邮件里的“我要注册”链接（这个链接对应着 *active* 列）并正确输入了这个随机字符串时，新账户才会正式启用。

- *messages* 数据表：这个数据表用来存放所有的讨论帖子，里面记录着每份帖子的 *Subject*（主题）、实际内容、*forumID*、*userID* 和一些内部管理用途的信息。从数据库设计工作的角度看，*messages* 数据表在 *myforum* 数据库里的 3 个数据表当中是最有意思的，本节将重点介绍。

这个数据库里的数据表全部是 MyISAM 数据表，这是因为目前只有 MyISAM 格式的数据表才支持全文检索功能。这个数据库里的所有字符串数据列都使用 *Latin1* 作为默认字符集，使用 *latin1_general_ci* 作为默认排序方式。图 8-3 是这个数据库的数据表关联/引用关系图。

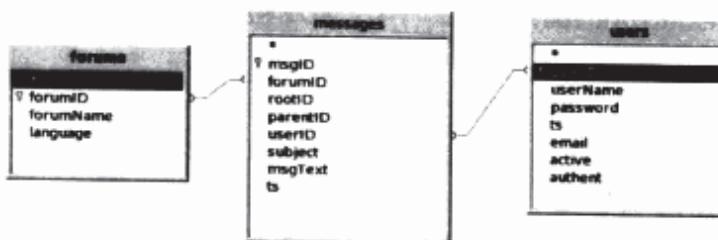


图 8-3 *myforum* 数据库的数据表关联/引用关系图

myforum 数据库已被收录在了本书的配套文件里，其中包含着 3000 多份选自 www.kofler.cc 网站的讨论帖子。（在这本书中的有关示例里，隐去了帖子作者的名字。）这个数据库非常适合用来测试各种全文检索功能（详见第 10 章）。

注解 *messages* 数据表还是一个全文索引会占用大量存储空间的好例子。在笔者为大家准备的测试数据库里，*messages* 数据表包含着大约 3000 份帖子，字符总数大约是 150 万个。这些帖子本身需要大约 1.7MB 的存储空间，它们的全文索引需要再消耗 1.4MB 的存储空间，即整个数据表的空间占用量差不多是它实际内容的两倍！

8.12.2 帖子之间的层次关系

就像在 *mylibrary* 数据库里看到的情况一样，在 *myforum* 数据库里也需要和层次关系大打交道。讨论组的最大特点之一是讨论线程呈典型的层次关系，而这些层次关系的起源是每个帖子都可能会有一些跟帖。表 8-32 给出了 A、B、C、D、E 5 位用户之间的某次讨论所形成的层次关系。

表 8-32 讨论线程

讨论内容（帖子的标题）
A: How do I sort with MySQL Tables? (17.1.2005 12:00)
B: first answer (17.1.2005 18:30)
A: thanks! (17.1.2005 19:45)
C: better suggestion (19.1.2005 10:30)
D: second answer (18.1.2005 3:45)
A: I don't understand that (18.1.2005 9:45)
D: the same explanation, but with more detail (18.1.2005 22:05)
E: third answer (18.1.2005 19:00)

因为演示的需要，在表 8-32 里给每个帖子分别加上了一个标题。在现实生活中，讨论线程中的跟帖往往没有自己的标题，在它们的标题行上一般只有讨论线程的名字（第一份帖子的名字）。

表 8-32 中的层次关系可以表示为如表 8-33 所示的数据库格式。不妨在此假设用户 A 到 E 的 *userID* 编号依次是 201~205，而论帖的 *msgID* 编号从 301 开始顺序递增。（在现实生活中，同一个讨论线程中的帖子很少会有顺序递增的 *msgID* 编号——绝大多数论坛按先来后到的顺序给帖子编号，顺序递增的下一个编号可能早已分配给其他讨论线程中的帖子了。）

表 8-33 *messages* 数据表：把讨论线程表示为数据记录

<i>msgID</i>	<i>forumID</i>	<i>rootID</i>	<i>parentID</i>	<i>userID</i>	<i>subject</i>	<i>ts</i>
301	3	301	NULL	201	How do I...	2005-01-17 12:00
302	3	301	301	202	first answer	2005-01-17 18:30
303	3	301	302	201	thanks!	2005-01-17 19:45
304	3	301	301	204	second answer	2005-01-18 3:45
305	3	301	304	201	I don't understand...	2005-01-18 9:45
306	3	301	301	205	third answer	2005-01-18 19:00
307	3	301	304	204	the same...	2005-01-18 22:05
308	3	301	302	203	better suggestion	2005-01-19 10:30

讨论帖子之间的层次关系通过 *parentID* 数据列得到了体现，每条讨论帖子记录的 *parentID* 字段指向自己的上一级论帖。如果 *parentID* 字段里的值是 *NULL*，就表明该讨论帖子开始了一个新的讨论线程。

数据列 *rootID* 指向讨论线程中的第 1 个论帖。这个数据列不允许包含 *NULL* 值。对于讨论线程中的第一个帖子，其 *msgID*=*rootID*。数据列 *rootID* 里的信息其实是冗余的，这是因为完全可以通过 *parentID* 字段为任何一个帖子找到它所在的讨论线程中的第一个帖子，但增加 *rootID* 数据列的好处是可以简化这一任务并大大提高整个论坛的效率：有了它，确定某个讨论线程中的讨论帖子总数、汇总某个讨论线程中的全部讨论帖子等工作只需一条简单的 *SELECT* 查询命令就可以完成了。

8.13 示例数据库 *exceptions*（用于特殊情况的测试）

在刚开始使用一种新的数据库系统、程序设计语言或者是还不太熟悉的 API 去开发一个应用程序时，如果能有一个简单的测试数据库可以对各种特殊情况进行快速测试将非常实用。在这本书里，将使用下面介绍的 *exceptions* 数据库对书中的示例代码、书中用到的各种 API 以及书中进行的各种导入/导出操作进行测试。这个数据库里的数据表可以提供以下（但不限于）测试素材：

- 各种 MySQL 数据类型的数据列，包括 *xxxTEXT*、*xxxBLOB*、*SET* 和 *ENUM* 等数据类型在内。
- *NULL* 值。
- 包含各种特殊字符的 *TEXT* 和 *BLOB* 数据。
- 全部 255 个文本字符（编码 1~255）。

在接下来的内容里，将对 *exceptions* 数据库各数据表的结构和内容做一个总结性的介绍。在这个数据库里，数据列的名字表明了它们的数据类型（比如说，数据列 *a_blob* 的数据类型是 *BLOB*）；除 *id* 数据列以外，所有其他的数据列都允许出现 *NULL* 值。

8.13.1 数据表 *testall*

这个数据表里的数据列分别是下面这些最重要（但不是全部）的 MySQL 数据类型。

数据列: *id*、*a_char*、*a_text*、*a_blob*、*a_date*、*a_time*、*a_timestamp*、*a_float*、*a_decimal*、*a_enum*、*a_set*。

8.13.2 数据表 *text_text*

这个数据表可以用来测试各种文本数据类型和有关操作。

数据列: *id*、*a_varchar* (最大长度是 100 个字符)、*a_text*、*a_tinytext*、*a_longtext*。

8.13.3 数据表 *test_blob*

这个数据表可以用来测试各种二进制数据类型和有关操作(导入、导出、用客户程序读取/存入等)。

□ 数据列: *id*、*a_blob*。

□ 内容: 这个数据表初始只有一条记录 (*id=1*)，该记录的数据内容是一个长度为 512 个字节的二进制数据块，块中的二进制数据是十进制数值 0, 1, 2, …, 255, 0, 1, 2, …, 255 的二进制编码。

8.13.4 数据表 *test_date*

这个数据表可以用来测试各种日期/时间数据类型和有关操作。

□ 数据列: *id*、*a_date*、*a_time*、*a_datetime*、*a_timestamp*。

□ 内容: 这个数据表初始只有一条记录 (*id=1*)，该记录的数据内容依次是“2005-12-31”、“23:59:59”、“2005-12-31 23:59:59”。

8.13.5 数据表 *test_enum*

这个数据表可以用来测试 *SET* 和 *ENUM* 数据类型以及有关操作。

□ 数据列: *id*、*a_enum* 和 *a_set* (字符集合{'a', 'b', 'c', 'd', 'e'})。

□ 内容:

<i>id</i>	<i>a_enum</i>	<i>a_set</i>
1	a	a
2	e	b,c,d
3		
4	NULL	NULL

8.13.6 数据表 *test_null*

这个数据表可以用来测试 MySQL 和/或客户程序能不能把 *NULL* 值 (第一条记录) 与一个空白字符串 (第二条记录) 区分开。

□ 数据列: *id*、*a_text*。

□ 内容:

<i>id</i>	<i>a_text</i>
1	NULL
2	
3	'a text'

8.13.7 数据表 *test_sort1*

这个数据表可以用来测试 *Latin1* 和 *UTF-8 (Unicode)* 字符集上的各种排序方式 (*ORDER BY*)。这个数据表由 3 个数据列构成: 数据列 *id* 包含着 33~126、再从 161~255 的序号数字; 数据列 *latin1char*

包含编码等于 *id* 字段值的 *Latin1* 字符；数据列 *utf8char* 包含着编码等于 *id* 字段值的 Unicode 字符。我们故意没有给后两个 *CHAR* 数据列配上索引。

内容：

```
id    latin1char  utf8char  
...  
65    'A'        'A'  
66    'B'        'B'  
...
```

8.13.8 数据表 *test_sort2*

这个数据表也是为了测试各种排序方式而创建的。它也由 3 个数据列构成：*id*、*latin1text* 和 *utf8text*（后两个数据列的数据类型都是 *VARCHAR(100)*，它们分别对应于 *Latin1* 和 *UTF8* 字符集）。这一次，文本数据列的内容都是些单词，如 *abc*、*Abc*、*ABC*、*Bar*、*Bär*、*Bären*、*Barenboin*、*bärtig*、*Ärger* 等。可以在第 9 章里找到几个使用了数据表 *test_sort1* 和 *test_sort2* 的例子。

8.13.9 数据表 *importtable1*、*importtable2*、*exporttable*

这 3 个数据表容纳着一些用来测试各种文本文件导入/导出操作的数据。将在第 14 章对这几个数据表以及 MySQL 的各种导入/导出功能做进一步的讨论。

第9章

SQL 语言入门



本章将向大家简要地介绍一些 SQL 语言（Structured Query Language，结构化查询语言）的基本知识。这种语言的主要用途是构造各种数据库系统操作指令，用来修改或删除各种数据库对象的查询和命令当然也包括在内。在这些命令当中最重要的是 *SELECT*、*INSERT*、*UPDATE* 和 *DELETE* 命令，它们也是这一章的讨论重点，我们将通过大量示例来演示它们的用法。

作为本章内容的延续，还在第 10 章介绍一些常见 SQL 问题的“药方”，其内容涉及如何处理字符串、如何处理日期/时间数据、如何构造和使用子查询命令（对 *SELECT* 查询结果做进一步查询）、如何使用事务、如何进行全文检索等。

9.1 简介

在第 8 章描述的那几个示例数据库是本章所有示例的共同基础。如果想在自己的系统上试用那几个示例数据库，必须先把它们下载到自己的测试环境里。（所有的源代码都可以通过 www.apress.com 网站的 Downloads 页面下载。）我们在本章各小节的开始或者是示例的开头用一条 *USE databaseName* 命令来表明这个示例使用了哪一个示例数据库。

对于比较简单的 SQL 命令，测试它们的最佳办法是使用 MySQL 自带的命令解释器，即 *mysql* 程序。在 UNIX/Linux 环境下，这个程序可以用 *mysql* 命令来启动；在 Windows 环境下，可以通过菜单命令 Program(程序)|MySQL|MySQL Server n.n|MySQL Command Client 来启动它。（注意，在 *mysql* 程序里执行 SQL 命令的时候千万不要忘记在命令的末尾加上一个分号。本章中的示例都省略了这个分号，因为其他种类的 MySQL 客户端程序都不要求这么做。也就是说，在 SQL 命令末尾加分号的做法不是 MySQL 的语法。关于 *mysql* 程序的更多信息可以在第 4 章找到。）

对于又长又复杂的 SQL 命令，*mysql* 程序用起来就不是那么方便了。这时候，最好选用一个第三方的 MySQL 客户端操作界面（比如 MySQL Query Browser 或 phpMyAdmin）来编辑和调试 SQL 命令；这类程序的优点是它们会把查询结果显示在一个整齐易读的表格里。

注解 在这本书里，SQL 命令名和 MySQL 关键字总是以大写字母写出；数据库、数据表和数据列的名字是以小写字母写出，但偶尔会有大小写字母混用的时候（比如 *ColumnName*）。除数据库和数据表的名字以外，MySQL 不区分字母的大小写形式。也就是说，只需要且必须在输入数据库和数据表的名字时注意区分字母的大小写，这一点希望大家千万要注意！MySQL 的详细命名规则可以在第 21 章找到。

DML、DDL 和 DCL

SQL 命令可以分为以下 3 大类别。

- **DML (Data Manipulation Language, 数据处理语言)**：这类命令主要包括 *SELECT*、*INSERT*、*UPDATE*、*DELETE* 以及另外几个用来从数据表读出数据、把数据存入数据表或是对数据表里的现有记录进行修改的命令。这类命令是本章的讨论重点。
- **DDL (Data Definition Language, 数据定义语言)**：这类命令主要包括 *CREATE TABLE*、*ALTER TABLE* 等用来定义和改变数据库结构的命令。我们将在本章结尾部分对这类命令中的几个常用命令进行介绍。
- **DCL (Data Control Language, 数据控制语言)**：这类命令主要包括 *GRANT*、*REVOKE* 以及另外几个用来帮助人们设置和调整 MySQL 访问控制机制的 SQL 命令。我们将在第 11 章对它们做详细的介绍。

为了方便大家查阅，我们把所有的 SQL 命令按字母表顺序汇总在第 21 章。

9.2 简单查询 (*SELECT*)

*SELECT * FROM tablename* 大概是最简单的数据库查询命令了。这个查询命令将返回给定数据表里的全部数据记录。星号字符 (*) 在这里的含义是：本次查询操作将涉及给定数据表里的全体数据列（如图 9-1 所示）。

```

Command Prompt mysql -uroot -p
mysql> USE mylibrary;
Database changed
mysql> SELECT * FROM publishers;
+-----+-----+-----+
| publID | publName | ts          |
+-----+-----+-----+
|     1 | Addison-Wesley | 2004-12-02 18:36:58 |
|     2 | Apress      | 2004-12-02 18:36:58 |
|     3 | New Riders  | 2004-12-02 18:36:58 |
|     4 | O'Reilly & Associates | 2004-12-02 18:36:58 |
|     5 | Hanser      | 2004-12-02 18:36:58 |
|     9 | Bonnier Pocket | 2004-12-02 18:36:58 |
|    16 | Zsolnay      | 2004-12-02 18:36:58 |
|    17 | Ordfront förlag AB | 2004-12-02 18:36:58 |
|    19 | Diogenes Verlag | 2004-12-02 18:36:58 |
|    20 | Markt und Technik | 2004-12-02 18:36:58 |
|    21 | Galileo      | 2004-12-02 18:36:58 |
|    23 | dpunkt       | 2004-12-02 18:36:58 |
|    24 | Sybex       | 2004-12-02 18:36:58 |
+-----+-----+-----+
13 rows in set (0.02 sec)

mysql>

```

图 9-1 在 MySQL 命令解释器 (mysql 程序) 里执行 SQL 命令

```

Use mylibrary
SELECT * FROM publishers
publID  publName
_____
1 Addison-Wesley
2 Apress
3 New Riders
4 O'Reilly & Associates
5 Hanser

```

注解 在 *SELECT* 命令里允许不出现数据库或数据表的名字——即 *SELECT* 命令的 *FROM* 子句可以省略，如 *SELECT 2*3*。此时 *SELECT* 命令将自己的执行结果返回为一个单行单列的小表格。*SELECT* 命令的这种用法多见于需要确定某个 MySQL 变量或函数的具体内容的场合（比如说，用 *SELECT NOW()* 命令来确定当前时间）。

9.2.1 确定数据表里有多少条数据记录（数据行）

有时候，并不需要查看数据记录的具体内容，而只是想知道某个数据表里到底包含了多少条数据记录。对于这种情况，可以使用以下查询：

```
SELECT COUNT(publID) FROM publishers
```

COUNT(publID)

5

注意，在如上所示的查询命令里，并不是只能使用 *publID*——完全可以把它替换为 *publishers* 数据表的任何一个数据列的名字（或者是代表全体数据列的星号字符“*”）。无论使用的是什么，在经过 MySQL 的优化之后，这种查询将只返回给定数据表里的数据记录个数，而不是去真正读取它们的内容。

9.2.2 确定数据表里有多少条内容不重复的数据记录（DISTINCT）

如果数据表里的某个数据列有完全相同的内容，而读者却只想知道这个数据列的内容不重复的数据记录有多少个，就必须使用关键字 *DISTINCT*。比如说，如果想知道 *titles* 数据表涉及多少家不同的出版公司，就需要使用如下所示的查询命令：

```
SELECT COUNT(DISTINCT publID) FROM titles
```

COUNT(DISTINCT publID)

11

这个查询的结果不必与上一个查询的同期¹结果丝毫不差。*publishers* 数据表完全有可能还收录了一些从未出版过图书（至少是 *titles* 数据表所收录的那些图书）的出版公司。

下面是一个利用 *DISTINCT* 关键字在 SQL 命令里进行一些简单计算的例子：计算每本图书平均有几位作者。在这里，*COUNT(*)* 将返回 *rel_title_author* 数据表（这个数据表把图书和作者关联在了一起）的数据记录总数，*COUNT(DISTINCT titleID)* 将返回这个数据表里不同图书的总数：

```
SELECT COUNT(*) / COUNT(DISTINCT titleID) FROM rel_title_author
```

COUNT(*) / COUNT(DISTINCT titleID)

1.27

9.2.3 限制查询结果中的数据列个数

在很多时候，并不需要看到一个数据表里的全部数据列。这时，需要在 *SELECT* 命令里明确写出出现在查询结果中数据列的名字（而不是使用星号“*”）：

```
SELECT publName FROM publishers
```

publName

Addison-Wesley
Apress
New Riders
O'Reilly & Associates
Hanser

1. 原书作者显然没有注意到这样一个细节：既然提到了上一个查询，就应该让这两个查询的结果保持“5”>“11”的关系——比如参照图 9-1 把“5”改为“13”。这里只能猜测这两次查询是在不同时期进行的——所以特意加上了“同期”二字：上一个查询发生在数据库里数据还不那么多的时候，这个查询发生在数据库里已经有了许多数据的时候。——译者注

提示 如果能预见到某个查询将返回大量的数据记录，就应该在 *SELECT* 命令里只写出感兴趣的数据列的名字（千万不要为了偷懒而只输入一个星号 “*”）。对查询结果中的数据列个数加以限制的好处是可以大大提高查询工作的效率：MySQL 服务器不必去提取用不着的数据、客户程序不必去申请用不着的内存、网络不必传输用不着的数据。

9.2.4 限制查询结果中的数据记录个数 (*LIMIT*)

不仅可以在查询结果中的数据列个数加以限制，还可以对查询结果中的数据记录个数加以限制。假设 *titles* 数据表收录了 100 000 本图书，但只需要用到其中前 10 条记录（比如说，把它们显示在一个 HTML 页面里），对另外 99 000 条记录也进行查询就只会浪费 CPU 时间、内存和网络带宽。为了避免这种浪费，应该在 *SELECT* 命令里加上一个 *LIMIT n* 子命令对查询结果中的数据记录个数加以限制。下面这条命令将只返回 *titles* 数据表里的两条记录：

```
USE mylibrary
SELECT title FROM titles LIMIT 2

title
Client/Server Survival Guide
Definitive Guide to Excel VBA
```

在这之后，如果想查看下两条记录，就需要再进行一次查询，但这次应该使用一个 *LIMIT offset n* 子命令，这里的 *offset* 是从数据表里的第 1 条数据记录开始算起的偏移值。（警告：MySQL 在计算数据记录的偏移值时永远从 0 开始。因为偏移值 *m* 意味着要跳过前 *m* 条记录，而这种计数又是从 0 开始算起的，所以有关操作实际将从数据表里的第 $(m+1)$ 条记录开始进行。）¹ 具体到下面这个查询，它将把数据表中的第 3 和第 4 条记录提取出来：

```
SELECT title FROM titles LIMIT 2, 2

title
Linux
Web Application Development with PHP 4.0
```

9.2.5 在使用 *LIMIT* 关键字确定数据表里的数据记录数 (*SQL_CALC_FOUND_ROWS*, *FOUND_ROWS()*)

包含 *LIMIT* 关键字的 *SELECT* 命令只能让用户得到一个部分结果，但在某些场合（尤其是在需要逐页地显示全部数据的时候），提前知道 *SELECT* 命令总共会返回多少条记录是很有必要的。

从 MySQL 4.0 开始，如果在一条带有 *LIMIT* 关键字的 *SELECT* 命令里增加一个 *SQL_CALC_FOUND_ROWS* 选项，就可以在下一条查询命令里用 SQL 函数 *FOUND_ROWS()* 查出这条 *SELECT* 命令在不带 *LIMIT* 关键字的情况下会返回多少条记录。

下面的第一条 *SELECT* 查询命令将返回 *titles* 数据表里按字母表顺序排在最前面的 3 条记录，第二条 *SELECT* 命令将指出 *titles* 数据表里总共有多少条记录：

```
SELECT SQL_CALC_FOUND_ROWS title FROM titles ORDER BY title LIMIT 3
```

1. 为了避免与 *LIMIT offset n* 里的 *n* 发生误会，把原文这里的几个 *n* 改为了 *m* 并且用“第 $(m+1)$ 条”消除了可能引起的误会。——译者注

```
title _____
A Guide to the SQL Standard
Alltid den där Annette
Client/Server Survival Guide

SELECT FOUND_ROWS()

FOUND_ROWS() _____
26
```

在一些比较复杂的查询里，专门使用一条 *SELECT* 命令去统计有关数据记录的总数可能会花费不少的时间。这时候，*SQL_CALC_FOUND_ROWS* 选项和 *FOUND_ROWS()* 函数就可以帮上大忙。不过，因为使用 *SQL_CALC_FOUND_ROWS* 选项会让 MySQL 无法对 *LIMIT* 查询进行某些特定的优化，所以在此建议大家只在有必要使用 *FOUND_ROWS()* 函数的时候才使用 *SQL_CALC_FOUND_ROWS* 选项。

9.3 对查询结果进行排序 (*ORDER BY*)

如果没有设置，*SELECT* 命令就不会对它们所返回的查询结果进行排序。也就是说，如果想让查询结果按某种顺序排列，就必须明确地使用一个 *ORDER BY column* 子命令做出必要的设定。下面的命令将根据 *mylibrary* 数据库里的 *authors* 数据表返回一份按字母表顺序排序的作者名单：

```
SELECT authName FROM authors ORDER BY authName
```

```
authName _____
Bitsch Gerhard
Darween Hugh
Date Chris
DuBois Paul
Edwards E.
Garfinkel Simon
Gerken Till
Harkey D.
Holz Helmut
...
```

如果更喜欢按倒序排列的查询结果，那就需要在 *ORDER BY* 子句里加上一个 *DESC* 关键字 (*descending*, 倒序的)，如 *ORDER BY authName DESC*。最终的排序效果还要取决于在定义数据表时为有关数据列设定的排序方式是什么。

注解 从某种意义上讲，即使没有 *ORDER BY* 子句，*SELECT* 查询结果的排列顺序也不是随机的——它们将按照在数据表里的存储顺序排列。不过，作为众多数据库用户中的一员，用户对这个顺序恐怕没有什么影响力，所以千万不要想当然地认为 *SELECT* 命令的查询结果会按照某种有规律的方式排序。如果是新创建的数据表，查询结果通常会按 *id* 编号顺序排列（如果 *id* 是一个 *AUTO_INCREMENT* 字段的话），可一旦有人修改或是删除过其中的数据记录，这个顺序就会被打乱。千万不要想当然！
如何以一种真正的随机方式选取数据记录的办法见第 10 章。

9.3.1 选择一种排序方式

在需要对字符串进行排序的时候，MySQL 首先选用的将是人们在创建数据表时为有关数据列设置的排序方式；如果那个数据列上没有排序方式，MySQL 将选用数据表上的排序方式；如果数据表

上也没有排序方式，MySQL 将选用数据库上的排序方式；如果数据库上还没有排序方式，MySQL 将回过头来选用那个数据列所使用的字符集的默认排序方式。（一个给定字符集的默认排序方式可以用 SQL 命令 *SHOW CHARACTER SET* 命令查看。）

如果希望临时改用另外一种排序方式对查询结果进行排序，可以在 *SELECT* 命令里用 *COLLATE* 子命令来进行设置，但此时有多少种排序方式可供选择（可以用 *SHOW COLLATE* 命令查看）还要取决于有关数据列具体使用的字符集。这种临时改用其他排序方式的做法将导致 MySQL 在排序时不使用任何索引。换句话说，对于一个大数据表，临时改用其他排序方式对查询结果进行排序将又慢又没有效率！

```
SELECT authName FROM authors
ORDER BY authName COLLATE latin1_german2_ci
```

也可以永久性地改变某个数据列的排序方式，这会使有关索引将被自动更新：

```
ALTER TABLE authors MODIFY authName VARCHAR(60)
CHARACTER SET latin1 COLLATE latin1_german2_ci
```

如果给定数据列上的可用排序方式里没有所需要的，可以用 *CONVERT* 命令为这个数据列里的数据临时指定另外一种字符集。但必须明白这样做的后果：这对一个大数据表来说将是一个需要花费大量时间的过程。在下面的例子里，*SELECT* 命令将先把作者姓名从 *latin1* 字符集转换为 *utf8* 字符集，然后再按照 Polish（波兰）排序方式对它们进行排序：

```
SELECT authName FROM authors
ORDER BY CONVERT(authName USING utf8) COLLATE utf8_polish_ci
```

9.3.2 试用不同的排序方式

在这里，建议大家找个机会利用示例数据库 *exceptions* 里的 *test_sort1* 和 *test_sort2* 数据表提前试用一下 MySQL 所支持的各种排序方式。下面的命令提供了一些例子。应该先用 *SHOW COLLATION* 命令获得一份可用排序方式的清单再进行这种测试。为了节约篇幅，下面的 *SELECT* 查询结果没有以表格形式给出，无法显示的字符也都删掉了。

在下面 4 条命令的执行结果里，国际符号和一些特殊字符在排列顺序上的差异很容易看出：

```
USE exceptions
```

```
SELECT latin1char FROM test_sort1
ORDER BY latin1char COLLATE latin1_general_ci
```

```
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A a
À à Á Á à Á ä Á å Á æ b B c C ç Ç d D ? ð E e Ë è É é ê Ë ë F
f g G h H I i i Í í i ï Í j J K k L l M m N n Ñ ñ Ò ò Ó ó Õ ó Õ ô
Ó ô Õ ö Ø Ø P p Q q R S s ß T t u U Ú ú Ú ü Ú ÿ V v W w X x Y
? ? ý Z z ? ? [ \ ] ...
```

```
SELECT latin1char FROM test_sort1
ORDER BY latin1char COLLATE latin1_german1_ci
```

```
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ Á á
à á á á Á Á Á Á Á á Á Á æ b B c C ç Ç d D é É è É õ õ õ õ õ õ õ õ õ
g h H í í I i i i Í Í j J K k L l M m N n Ñ ñ Ò ò Ó ó Õ ó Õ ô
ó ô ô Ø Ø P p Q q R S s t T Ú ú Ú ü Ú ÿ V v W w X x Y ? ?
z Z [ \ ] ^ _ { | } ...
```

```
SELECT latin1char FROM test_sort1
ORDER BY latin1char COLLATE latin1_swedish_ci
```

```
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ á á
```

```

à a Ä Å À à A b B ç c Ç C D d ð ? e è E É Ë é è ë F f G g H Í
í I Í ï i ï I J j K k L l m M ñ Ñ n N ó Õ Õ o õ ô P p Q q
r R S s T t u Ú Ù Õ Õ U ù Ú V v W w X Y Ü ? y ü ? z Z à Á [ Á é Ä \ 
ä ö ] ö ^ ` { | } ...
SELECT utf8char FROM test_sort1
ORDER BY utf8char COLLATE utf8_general_ci
! " # $ % & ' ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ a à
á â ä å Ä Å à à Á Á b B c Ç ç C d ð e E É è é Ë ê F f G g H
h Í í Í ï i ï i ï j J k k l L m M ñ Ñ n N ó Õ Õ o õ ô Õ Õ ö ö
P p Q q R r S s T t u Ú Ù Õ Õ U ù Ú V v W w X y ? y Y ? z Z [ \
] ^ _ ` { | } ...

```

在下面的几个例子里，可能需要仔细观察才能找出它们之间的差异。比如说，在下面的第 2 个例子里（DIN-1 标准下的 German（德国）排序方式），*Bar* 出现在 *Bär* 的前面；但在第 3 个例子里（DIN-2 标准下的 German 排序方式），*Bär=Baer* 却出现在了 *Bar* 的前面。在 Swedish（瑞典）排序方式下，因为字符“Ä”、“Å”和“Ö”是瑞典字母表里的最后几个字符，所以 *Ärger* 出现在了排序结果的最末尾。在最后一个例子里，我们看到 Unicode 排序方式 *utf8_general_ci* 与 DIN-1 标准下的 German 排序方式完全一样——至少对这里给出的那些单词来说是如此。

```

SELECT latin1text FROM test_sort2
ORDER BY latin1text COLLATE latin1_general_ci
abc Abc ABC Ärger Bar Barenboim Bär Bären bärtig

SELECT latin1text FROM test_sort2
ORDER BY latin1text COLLATE latin1_german1_ci
abc Abc ABC Ärger Bar Bär Bären Barenboim bärtig

SELECT latin1text FROM test_sort2
ORDER BY latin1text COLLATE latin1_german2_ci
abc Abc ABC Ärger Bär Bären bärtig Bar Barenboim

SELECT latin1text FROM test_sort2
ORDER BY latin1text COLLATE latin1_swedish_ci
abc Abc ABC Bar Barenboim Bär Bären bärtig Ärger

SELECT utf8text FROM test_sort2
ORDER BY utf8text COLLATE utf8_general_ci
abc Abc ABC Ärger Bar Bär Bären Barenboim bärtig

```

9.4 筛选数据记录（WHERE, HAVING）

在很多时候，用户感兴趣的并不是某个数据表里的全部记录，而只是它们当中能够满足某一种或某几种条件的记录。这类条件要用 *WHERE* 关键字来给出。在第一个例子里，只想查出其姓氏的第一个字母落在从“M”¹到“Z”区间内的图书作者的名字：

```
SELECT authName FROM authors WHERE authName>='M'
```

<u>authName</u>
Orfali R.
Pohl Peter
Ratschiller Tobias

1. 原文此处误写为“L”。——译者注

Reese G.

...

在第2个例子里，使用了 *LIKE* 操作符来比较字符串。这个查询将把其姓名里包含着字母序列 *er* 的作者查出来。在操作符 *LIKE* 后面的条件表达里，字符“%”是代表任意字符串的通配符。

```
SELECT authName FROM authors WHERE authName LIKE '%er%'
```

<i>authName</i>
Bitsch Gerhard
Gerken Till
Kofler Michael
Kramer David
Pohl Peter
Ratschiller Tobias
Schmitt Bernd
Yarger R.J.

注意 使用 *LIKE* 操作符进行的比较操作在比较大的数据表上往往非常慢，一是因为 MySQL 需要读取和分析数据表里的全部数据记录，二是因为这类查询无法用索引来优化。使用了 *LIKE* 操作符的查询往往可以改写为一个利用全文索引进行的全文检索查询。

枚举型的¹比较操作可以用操作符 *IN* 来实现：

```
SELECT authID, authName FROM authors
WHERE authID IN (2, 7, 12)
```

<i>authID</i>	<i>authName</i>
2	Kramer David
7	Gerken Till
12	Yarger R.J.

查询条件并非只能由关键字 *WHERE* 引导，还可以使用关键字 *HAVING* 来引导查询条件。如果它们同时出现，MySQL 将优先执行 *WHERE* 子句，而 *HAVING* 子句只能用来对 *SELECT ... WHERE ...* 查询的结果（中间结果）做进一步筛选。*HAVING* 关键字的优点是查询条件还可以作用于有关字段的数学计算结果（比如某个 *GROUP BY* 查询中的 *SUM(columnXy)* 函数）；在 9.7.2 节里有一个例子。

对 MySQL 来说，*HAVING* 子句不像 *WHERE* 条件子句那么容易优化，而这意味着我们应该尽可能地避免使用 *HAVING* 子句——除非找不到与之等价的 *WHERE* 子句。

注解 MySQL 不支持 *colname = NULL* 这样的用法——如果打算对包含 *NULL* 值的数据记录进行搜索，就必须使用 *ISNULL(colname)* 函数。

9.5 涉及多个数据表的关联查询 (*LEFT/RIGHT JOIN*)

到目前为止，我们在本章给出的 *SELECT* 查询示例全都是对单个数据表进行查询的。但是，在关系数据库里，往往需要与多个彼此相关的数据表打交道，需要用 *SELECT* 命令对多个数据表里的数据进行查询。这种涉及多个数据表的关联查询需要使用 *JOIN* 语法来构造。

1. 原文这里是“对大量数据值进行的”，但我们认为“枚举型的”更好。——译者注

9.5.1 两个数据表的关联

下面这个查询本想从 *titles* 和 *publishers* 数据表创建一份所有图书 (*titles* 数据列) 及其出版公司 (*publName* 数据列) 的清单，但这次尝试显然是失败了：

```
USE mylibrary
SELECT title, publName FROM titles, publishers
```

title	publName
Client/Server Survival Guide	Addison-Wesley
Definitive Guide to Excel VBA	Addison-Wesley
Linux	Addison-Wesley
Web Application Development with PHP 4.0	Addison-Wesley
Client/Server Survival Guide	Apress
Definitive Guide to Excel VBA	Apress
Linux	Apress
Web Application Development with PHP 4.0	Apress
Client/Server Survival Guide	New Riders
Definitive Guide to Excel VBA	New Riders
Linux	New Riders
Web Application Development with PHP 4.0	New Riders
...	

MySQL 返回的是图书和出版公司的所有组合。因为示例数据库并不大，所以从这些组合里把想要的东西挑出来还不算困难。但试想一下，如果这个数据库里收录了 10 000 本图书和 500 家出版公司，它们的组合就会有 500 万种。

如果想让涉及多个数据表的查询返回有意义的结果，就必须准确无误地告诉 MySQL 怎样把来自那几个数据表的数据关联在一起。表达这种关联关系的办法之一是给 *SELECT* 命令增加一个 *WHERE* 子句。具体到我们这个例子，已经知道 *titles* 和 *publishers* 数据表是通过 *publID* 字段相关联的，但因为这个字段在两个数据表里的名字一模一样（都是 *publID*），所以必须用 *table.column* 的形式把它们区分开：

```
SELECT title, publName FROM titles, publishers
WHERE titles.publID = publishers.publID
```

title	publName
Linux, 5th ed.	Addison-Wesley
Definitive Guide to Excel VBA	Apress
Client/Server Survival Guide	Addison-Wesley
Web Application Development with PHP 4.0	New Riders
MySQL	New Riders
MySQL & mSQL	O'Reilly \& Associates
...	

有许多办法可以获得同样的结果，其中之一是用 *LEFT JOIN ... ON* 语法来表明这两个数据表之间的关联关系：

```
SELECT title, publName
FROM titles LEFT JOIN publishers
ON titles.publID = publishers.publID
```

还有一种办法是使用关键字 *USING* 来给出两个数据表之间的关联字段，此时两个数据表里的关联字段必须是相同的名字（如本例中的 *publID*）；否则就不能使用这个办法：

```
SELECT title, publName
FROM titles LEFT JOIN publishers
USING (publID)
```

9.5.2 3个或更多个数据表的关联

当查询所涉及的数据表超过两个时，事情就变得更有意思了。下面的查询命令将返回一份所有图书及其所有作者的清单，同一本书有多位作者的将在清单里重复出现。图书名称（*titles* 数据表）和图书作者（*authors* 数据表）之间的关系是通过 *rel_title_author* 数据表建立的，所以这个查询将涉及 3 个数据表。筛选结果数据记录的查询条件是：*titles* 和 *rel_title_author* 数据表里的 *titleID* 字段值相等，*authors* 和 *rel_title_author* 数据表里的 *authorID* 字段值相等。

```
SELECT title, authName
  FROM titles, rel_title_author, authors
 WHERE titles.titleID = rel_title_author.titleID
   AND authors.authID = rel_title_author.authID
 ORDER BY title
```

title	authName
A Guide to the SQL Standard	Date Chris
A Guide to the SQL Standard	Darween Hugh
Alltid den där Annette	Pohl Peter
Client/Server Survival Guide	Orfali R.
Client/Server Survival Guide	Harkey D.
Client/Server Survival Guide	Edwards E.
Definitive Guide to Excel VBA	Kofler Michael
Definitive Guide to Excel VBA	Kramer David
Excel 2000 programmieren	Kofler Michael
Jag saknar dig, jag saknar dig	Pohl Peter
LaTeX	Kopka Helmut
Linux für Internet und Intranet	Holz Helmut
Linux für Internet und Intranet	Schmitt Bernd
Linux für Internet und Intranet	Tikart Andreas
Linux, 5th ed.	Kofler Michael
Maple	Kofler Michael
Maple	Komma Michael
Maple	Bitsch Gerhard
...	

下一个例子就更复杂了：生成一份出版公司和图书作者的清单，即这个查询将告诉用户哪些作者写的书被哪些出版公司出版了。出版公司（*publishers* 数据表）和图书作者（*authors* 数据表）之间的关系也是通过 *rel_title_author* 数据表建立的，所以这里总共将涉及 4 个数据表。

因为 *mylibrary* 数据库所收录的作者当中有几位在同一家出版公司出版过不止一本书，所以如果只是把这几个数据表简单地关联起来，将会出现同一种“作者-出版公司”组合在查询结果里重复出现的现象。因此，在这个查询命令里还使用了 SQL 关键字 *DISTINCT* 来确保重复的数据记录只输出一次。

```
SELECT DISTINCT publName, authName
  FROM publishers, titles, rel_title_author, authors
 WHERE titles.titleID = rel_title_author.titleID
   AND authors.authID = rel_title_author.authID
   AND publishers.publID = titles.publID
 ORDER BY publName, authName
```

publName	authName
Addison-Wesley	Bitsch Gerhard
Addison-Wesley	Darween Hugh
Addison-Wesley	Date Chris
Addison-Wesley	Edwards E.
Addison-Wesley	Harkey D.
...	
Apress	Kofler Michael
Apress	Kramer David

New Riders	DuBois Paul
New Riders	Gerken Till
New Riders	Ratschiller Tobias
...	

关联查询的语法

如果注意过 MySQL 文档里 *SELECT* 命令语法之下的 *FROM* 子句的细节，就会发现这个子句有许多彼此差不多完全一样的变体；我们把 *FROM* 子句的各种变体汇总在了表 9-1 和表 9-2 里。

表 9-1 无条件关联（数学意义上的完全组合，几乎没有实际用途）

语法变体
(1) <i>FROM table1, table2</i>
(2) <i>FROM table1 JOIN table2</i>
(3) <i>FROM table1 CROSS JOIN table2</i>
(4) <i>FROM table1 INNER JOIN table2</i>
(5) <i>FROM table1 STRAIGHT_JOIN table2</i>

表 9-2 有条件关联

语法变体
(6) <i>FROM table1, table2 WHERE table1.xyID = table2.xyID</i>
(7) <i>FROM table1 LEFT [OUTER] JOIN table2 ON table1.xyID = table2.xyID</i>
(8) <i>FROM table1 LEFT [OUTER] JOIN table2 USING (xyID)</i>
(9) <i>FROM table1 NATURAL [LEFT [OUTER]] JOIN table2</i>
(10) <i>FROM table2 RIGHT [OUTER] JOIN table1 ON table1.xyID = table2.xyID</i>
(11) <i>FROM table2 RIGHT [OUTER] JOIN table1 USING (xyID)</i>

表 9-1 中，第（1）项到第（4）项都意味着 MySQL 将依靠它自己去找出有关数据的最佳访问序列；表 9-1 中，第（5）项与前 4 项的区别在于 MySQL 将不对从数据表提取有关数据的顺序进行优化。第（6）项的含义是：在对两个数据表进行关联查询的时候，只提取那些满足筛选条件——即两个给定字段的取值相等——的记录。

表 9-2 中，第（7）项到第（9）项的语义相同：为第一个（左）数据表 (*table1*) 里的每一条记录生成一条结果记录，即使第二个（右）数据表 (*table2*) 里的关联字段取值为 *NULL* 也必须如此（所谓的“左关联”）。比如说，如果把 *titles* 数据表用做 *table1*，把 *publishers* 数据表用做 *table2*，那么 *LEFT JOIN* 将为 *titles* 数据表里的每一本书生成一条结果记录，即使 *publishers* 数据表里没有与之对应的出版公司也必须如此。（关键字 *OUTER* 是可选的，但有它和没有的效果完全一样。）

表 9-2 中，第（10）项和第（11）项分别对应着第（7）项和第（8）项，但把 *table1* 和 *table2* 的角色刚好调换了过来（所谓的“右关联”）。不过，MySQL 文档提出了这样的建议：为了最大限度地与其他品牌的数据库系统保持兼容，最好不要使用 *RIGHT JOIN*——应该把它改写为 *LEFT JOIN*（把数据表的左、右顺序调换一下）。

在细节方面，两个 *USING* 变体都要求两个数据表里的 *ID* 字段有相同的名字。*NATURAL* 变体将自动使用两个数据表里名字相同的字段进行关联，这意味着两个数据表里除了 *ID* 字段之外不应该再有任何名字相同的字段。

注意 在 *LEFT JOIN* 查询里，数据表的左、右顺序非常关键。以 *titles* 和 *publishers* 数据表为例：

□ *titles LEFT JOIN publishers* 可以返回没有出版公司的图书，但没有出版过书的出版公司

不行。

□ *publishers LEFT JOIN titles* 可以返回没有出版过书的出版公司，但没有出版公司的图书不行。

与许多其他的SQL“方言”不同，MySQL不支持关键字*FULL JOIN*（全关联），*FULL JOIN*返回的是*LEFT JOIN*（左关联）和*RIGHT JOIN*（右关联）的并集。未来的MySQL 5.1系列版本可能会增加这一功能。

9.6 合并查询结果（UNION）

从MySQL 4.0开始，可以用关键字*UNION*把两个或更多个*SELECT*查询命令合并在一起，而最终的结果是各次查询结果的顺序排列：

```
SELECT command UNION [ALL] SELECT command ...
```

下面的例子演示了同一个数据表上的两个*SELECT*命令的结果是如何合并在一起的。（当然，不同数据表上的查询也可以合并，但需要保证各次查询结果的数据列在个数和数据类型上都是一样的。否则，MySQL将把最终的结果数据全部转换为第一条*SELECT*命令的数据类型。）

```
USE mylibrary
SELECT * FROM authors WHERE authName LIKE 'b%'
UNION
SELECT * FROM authors WHERE authName LIKE 'g%'
authID authName
22 Bitsch Gerhard
26 Garfinkel Simon
7 Gerken Till
```

一般来说，重复的记录将从最终结果里被自动消除。MySQL只有在使用了*UNION ALL*的时候才会把最终结果里重复的记录保留下来。

还可以把个别的*SELECT*命令用圆括号括起来，这样，可以对每次查询以及最终结果做出*LIMIT*和*ORDER BY*设置。下面这个查询先从数据表*tbl1*和*tbl2*里分别选取了最多10条记录并把它们合并起来，然后又从这些（最多20个）记录里选取了前5个显示了出来：

```
(SELECT * FROM tbl1 ORDER BY colA LIMIT 10)
UNION
(SELECT * FROM tbl2 ORDER BY colA LIMIT 10)
ORDER BY colTimestamp LIMIT 5
```

注解 目前，*UNION*是MySQL里唯一的集合分隔符。MySQL没有提供关键字*MINUS*和*INTERSECT*，许多其他的数据库系统提供了这两个关键字。

9.7 分组查询，统计函数（GROUP BY）

在当初把图书信息保存为*mylibrary*数据库里的数据记录时，我们还为它们划分了一些门类。现在，如果想知道不同的门类都包含着哪些图书，将发现下面这个查询能帮上忙。（警告：没被归入任何门类的图书将根本不会出现在这个查询的结果里。如果想把所有的图书都列出来，必须使用*FROM titles LEFT JOIN categories ON titles.catID=categories.catID*这样的关联查询。）

```
USE mylibrary
SELECT catName, title FROM titles, categories
WHERE titles.catID = categories.catID
```

ORDER BY catName, title	
catName	title
Children's books	Alltid den där Annette
Children's books	Jag saknar dig, jag saknar dig
Computer books	LaTeX
Computer books	Linux, 5th ed.
Computer books	Maple
Databases	Client/Server Survival Guide
Databases	Visual Basic Datenbankprogrammierung
Programming	Definitive Guide to Excel VBA
Programming	Visual Basic
SQL	A Guide to the SQL Standard

9.7.1 统计函数

如果想知道每个门类里有多少本书，可以在刚才生成的清单里数一数：2本儿童类图书、3本计算机类图书、2本关于数据库的图书等。

有没有想过会有一个更自动化的办法来做这件事情呢：*GROUP BY name* 语句可以按一个给定数据列的每个成员对查询结果进行分组统计，最终看到的是一个分组汇总表。不过，简单地给上面那个查询加上 *GROUP BY catName* 子句的效果并不十分理想：比如说，*catName='Children's books'* 门类的那两本书将只剩下一本，能看到哪一本要由排序方式来决定——别忘了，就算没有指定排序方式，MySQL 也会按照默认排序方式显示查询结果。

显然，单独使用 *GROUP BY* 并没有真正达到目的（除非确实想把每个门类里的所有图书全部列举出来，但那样也有更简单的办法）。SQL 语言提供的各种统计函数可以配合 *GROUP BY* 语句实现这一功能，可以在 *SELECT* 命令里用 *COUNT()*、*SUM()*、*MIN()* 和 *MAX()* 等函数去告诉 MySQL 用户想看到的查询结果到底是什么。也正是因为有了这些函数，*GROUP BY* 子句才真正成为了一种有用的工具，就像下一个例子所演示的那样：使用 *COUNT()* 函数来统计每个门类有多少本图书。

这里新出现了关键字 *AS*，它给查询结果中的第二列起了一个名字叫 *nrOfItems*。如果不使用 *AS*，这个列的名字将是 *COUNT(itemID)*，在查询结果里看到这样的列标题多少有点儿让人摸不着头脑，在查询命令里引用这个列的时候（比如在 *ORDER BY* 子句里）还需要多输入几个字。

SELECT catName, COUNT(title) AS nrOfItems	
FROM titles, categories	
WHERE titles.catID = categories.catID	
GROUP BY catname	
ORDER BY catname	
catName	nrOfItems
Children's books	2
Computer books	3
Databases	2
Programming	2
SQL	1

正如在前面的内容里提到的那样，如果确实想把所有的图书门类——不管数据库里有没有收录该种门类的图书——都列出来，就必须像下面这样使用关键字 *LEFT JOIN* 去关联这两个数据表。顺便说一句，这次的排序效果是图书数量最多的门类显示在最前面。

```
SELECT catName, COUNT(title) AS nrOfItems
FROM categories LEFT JOIN titles ON titles.catid = categories.catid
GROUP BY catname
ORDER BY nrOfItems DESC
```

<i>catName</i>	<i>nrOfItems</i>
Computer books	5
MySQL	4
Programming	3
Children's books	3
PHP	2
Databases	2
Literature and fiction	2
LaTeX, TeX	1
SQL	1
Object-oriented databases	0
All books	0
Relational Databases	0
Perl	0

提示 SQL语言还允许把`IF()`函数放在一个统计函数里，比如说，如果想让`COUNT()`和`SUM()`等函数只对符合某种条件的记录进行统计，就需要这么做。我们将在第10章给出一些这种用法的示例。

9.7.2 统计函数 `GROUP_CONCAT()`

统计函数`GROUP_CONCAT()`是从 MySQL 4.1 版开始引入的，它的作用是把一些字符串归为一个分组，而这让人们得以实现许多富于创造力的想法——就像下面这个例子所演示的那样：把作者人数在一个以上的图书列举出来，同时把作者们的姓名按字母表顺序也列举出来。

在这个例子里，如果不强调对作者姓名进行排序，只需把有关数据列的名字简单地用做`GROUP_CONCAT()`函数的参数变量就行了，MySQL 将按照这个数据列的默认排序方式把作者们的姓名字符串归为一个分组，字符串之间用逗号隔开。我们在这里使用了`ORDER BY`关键字来明确地指定一种排序方式，又使用了一个`SEPARATOR`关键字来明确地指定一个分隔符：

```
SELECT title,
       GROUP_CONCAT(authname ORDER BY authname SEPARATOR ', ')
          AS authors,
       COUNT(authors.authID) AS cnt
  FROM authors, titles, rel_title_author
 WHERE authors.authID = rel_title_author.authID
   AND titles.titleID = rel_title_author.titleID
 GROUP BY titles.titleID
 HAVING cnt>1
 ORDER BY title
```

<i>title</i>	<i>authors</i>	<i>cnt</i>
A Guide to the SQL Standard	Darween Hugh, Date Chris	2
Client/Server Survival Guide	Edwards Jeri, Harkey Dan, ...	3
Linux für Internet und Intranet	Holz Helmut, Schmitt Bernd, ...	3
Maple	Bitsch Gerhard, Kofler Michael, ...	3
Mit LaTeX ins Web	Goosens Michael, Rahtz Sebastian	2
MySQL	Kofler Michael, Kramer David	2
MySQL & mSQL	King Tim, Reese Georg, Yarger ...	3
PHP 5 und MySQL 5	Kofler Michael, Öggl Bernd	2
PHP and MySQL Web Development	Thomson Laura, Wellington Luke	2
The Definitive Guide to ...	Kofler Michael, Kramer David	2
The Definitive Guide to ...	Kofler Michael, Kramer David	2
Visual C#	Frank Eller, Kofler Michael	2
Web Application Development ...	Gerken Till, Ratschiller Tobias	2

9.7.3 对多个数据列进行 GROUP BY 查询

GROUP BY 还可以用来对多个数据列进行分组。下面这个查询将按语言和图书门类对图书进行分组统计。这里需要指出的是，没被归入某种语言或门类的图书都不会出现在查询结果里。

```
SELECT, langName, catName, COUNT(*)
FROM titles, languages, categories
WHERE titles.catID = categories.catID
    AND titles.langID = languages.langID
GROUP BY langName, catName
```

<i>langName</i>	<i>catName</i>	COUNT(*)
deutsch	C#	1
deutsch	Children's books	1
deutsch	Computer books	5
deutsch	Databases	1
deutsch	LaTeX, TeX	2
deutsch	Linux	4
deutsch	Literature and fiction	7
deutsch	MySQL	1
deutsch	PHP	3
deutsch	Programming	4
deutsch	Relational Databases	1
deutsch	Visual Basic	1
deutsch	Visual Basic .NET	3
english	Computer books	1
english	Literature and fiction	2
english	MySQL	3
english	PHP	2
english	Science Fiction	3
english	SQL	1
norsk	Literature and fiction	1
svensk	Children's books	2
svensk	Literature and fiction	9

9.7.4 GROUP BY...WITH ROLLUP

从 MySQL 4.1 开始，*GROUP BY* 语法又增加了一个新的关键字 *WITH ROLLUP*。如果 *GROUP BY* 子句里只有一个数据列，加上 *WITH ROLLUP* 关键字的效果是 MySQL 将在查询结果的最后一行将自动增加一条总数统计记录，这条记录的 ID 字段取值或者说这条记录的名字永远是 *NULL*。如下所示：

```
SELECT langName, COUNT(*)
FROM titles, languages
WHERE titles.langID = languages.langID
GROUP BY langName WITH ROLLUP
langName      COUNT(*)
deutsch          34
english          12
norsk            1
svensk           11
NULL             58
```

WITH ROLLUP 关键字在多个数据列上的效果更让人感兴趣。在这种场合，*GROUP BY* 将为查询结果里的第一列分组统计一个阶段性总和（相当于“小计”，第 2 列里的有关数字相加），最后再为全体记录统计一个最终的总和（相当于“总计”，第 2 列里的所有数字相加）。在下面的例子里，“小计”和“总计”记录都以黑体字突出显示。

```

SELECT langName, catName, COUNT(*)
FROM titles, languages, categories
WHERE titles.catID = categories.catID
    AND titles.langID = languages.langID
GROUP BY langName, catName WITH ROLLUP
langName  catName          COUNT(*)
deutsch   C#                1
deutsch   Children's books  1
deutsch   Computer books   5
deutsch   Databases         1
deutsch   LaTeX, TeX        2
deutsch   Linux              4
deutsch   Literature and fiction  7
deutsch   MySQL              1
deutsch   PHP                3
deutsch   Programming       4
deutsch   Relational Databases 1
deutsch   Visual Basic      1
deutsch   Visual Basic .NET  3
deutsch   NULL               34
english   Computer books   1
english   Literature and fiction  2
english   MySQL              3
english   PHP                2
english   Sience Fiction    3
english   SQL                1
english   NULL               12
norsk     Literature and fiction  1
norsk     NULL               1
svensk   Children's books  2
svensk   Literature and fiction  9
svensk   NULL               11
NULL     NULL               58

```

9.8 修改数据 (*INSERT*、*UPDATE* 和 *DELETE*)

9.8.1 备份数据

在开始在数据库里“胡闹”之前，应该先为有关的数据表或是整个数据库制作一个备份，这样万一数据库搞乱了，还可以把数据库恢复到它原来的状况。可以选用 `mysqldump` 之类的专用工具（见第 4 章），也可以直接使用 SQL 命令来制作备份；本节将要讨论的只是众多办法中的一种。

提示 MySQL新手经常会因为考虑不周而在使用 *UPDATE* 和 *DELETE* 命令修改或删除记录时误改、误删了有用的东西。为了把失误造成的损失降到最低，应该在启动 `mysql` 程序时加上 `--i-am-a-dummy`（中文意思是“我是个新手”）选项：`mysql` 程序拒绝执行不带 *WHERE* 子句的 *UPDATE* 和 *DELETE* 命令。

1. 为数据表制作一个副本

下面这条命令将创建一个名为 *newtable* 新数据表并把 *table* 数据表里的数据全部拷贝到这个新数据表里去。新数据表里的数据列定义与老数据表的完全一样，但偶尔会出现一些意外，比如说，个别数据列上的 *AUTO_INCREMENT* 属性可能会丢失（但 *AUTO_INCREMENT* 字段的数据值会保存下来）、新数据表里没有任何索引等：

```
CREATE TABLE newtable SELECT * FROM table
```

2. 用数据表副本恢复数据表

下面两条命令中的第一条将把数据表 *table* 里的现有记录全部删除，第二条命令将把保存在 *newtable* 数据表里的数据全部插入到 *table* 数据表里去（在插入过程中，原始的 *AUTO_INCREMENT* 字段值将得到恢复）¹。

```
DELETE FROM table
INSERT INTO table SELECT * FROM newtable
```

如果不想继续保留这个副本，把 *newtable* 数据表删掉即可：

```
DROP TABLE newtable
```

3. 为整个数据库制作一个备份

可以用 *mysqldump* 程序（它在 Windows 系统上的文件名是 *mysqldump.exe*）把一个完整的数据库备份为一个文本格式的文件。注意，*mysqldump* 是一个独立的程序而不是 *mysql* 程序里的一个命令；也就是说，不能在 *mysql* 程序里执行 *mysqldump* 命令，必须像下面这样在一个 shell 或命令窗口里启动它：

```
user$ mysqldump -u loginname -p dbname > backupfile
Enter password: xxxx
```

4. 用数据库备份恢复数据库

没有与 *mysqldump* 程序功能相对的数据恢复工具，需要依靠 *mysql* 命令去读入一个用 *mysqldump* 程序制作的数据库备份文件。备份文件将被用做输入源，而目标数据库（如本例中的 *dbname* 数据库）必须已经存在：

```
user$ mysql -u loginname -p dbname < backupfile
Enter password: xxxx
```

当然，也可以在交互模式下重建这个数据库：

```
user$ mysql -u root -p
Enter password: xxxx
mysql> CREATE DATABASE dbname; -- if dbname does not yet exist
mysql> USE dbname;
mysql> SOURCE backupfile;
```

9.8.2 插入数据记录 (*INSERT*)

可以用 *INSERT* 命令往数据表里插入一条新记录。在数据表名字的后面，必须先写出各有关数据列的名字（有默认值、允许包含 *NULL* 值以及带有 *AUTO_INCREMENT* 属性的数据列不必写出），然后再写出将被插入的数据。

在下面的例子里，将在 *mylibrary* 数据库的 *titles* 数据表里插入一条新的数据记录。这里只写出了两个数据列（*title* 和 *year*），剩下的数据列都由 MySQL 负责照顾：MySQL 会在 *titleID* 字段插入一个新的 *AUTO_INCREMENT* 值，在其他字段放上它们各自的默认值或 *NULL*——因为在创建 *titles* 数据表的时候已经给这几个字段设置了默认值或是允许它们包含 *NULL* 值。

```
USE mylibrary
INSERT INTO titles (title, year)
VALUES ('MySQL', 2005)
```

不写出数据列的名字也可以插入新记录，但此时必须为所有的数据列——按照它们在数据表里的先后顺序——都提供一个数据值。以 *title* 数据表为例，必须按顺序为以下这些数据列提供数据值：

1. 原文这里直译是“原始的 *AUTO_INCREMENT* 字段值不会被改变”，不够准确。——译者注

titleID、*title*、*subtitle*、*edition*、*publID*、*catID*、*langID*、*year*、*isbn*、*comment* 和 *ts*。

在下面的命令里，为好几个数据列提供的都是 *NULL* 值，而 MySQL 实际存入这些数据列的也大都是 *NULL* 值，除了两个例外：MySQL 将为 *titleID* 数据列自动生成一个新的 *AUTO_INCREMENT* 值（因为它带有 *AUTO_INCREMENT* 属性），而实际存入 *ts* 数据列的则是系统的当前日期和时间（因为它是一个 *TIMESTAMP* 类型的数据列）。

```
INSERT INTO titles
VALUES (NULL, 'deleteme', '', 1, NULL, NULL, NULL, 2005, NULL, NULL)
```

INSERT 命令的另外一种语法允许人们用一条命令插入多条新的数据记录，如下所示：

```
INSERT INTO table (columnA, columnB, columnC)
VALUES ('a', 1, 2), ('b', 12, 13), ('c', 22, 33), ...
```

向关联数据表插入新记录

向几个彼此关联的数据表插入新记录往往需要执行一条以上的 *INSERT* 命令。比如说，为了把一本新书收录到 *mylibrary* 数据库里，至少需要在 *titles* 和 *rel_title_author* 两个数据表中各插入一条新记录。如果这本新书的作者（可能不止一位）或出版公司还没有被收录在 *mylibrary* 数据库里，那至少还需要在 *publishers* 和 *authors* 数据表里各插入一条新记录。

事情并不是那么简单，因为需要在 *INSERT* 命令里写出的数据可能会有一些是在刚开始的时候还不知道的。比如说，为了把一本由一家新出版公司出版的新书收入数据库，必须在往 *titles* 数据表里插入新记录的时候给出新出版公司的 *publID* 编号，可它是一个 *AUTO_INCREMENT* 值，在还没有把这家新出版公司收录到数据库里之前，是不可能知道 MySQL 为它自动生成的编号值到底是多少的。

既然如此，能不能查出最新生成的 *AUTO_INCREMENT* 值到底是多少就成为了解决问题的关键。办法当然是有的，而且还不止一种，其中最为简明易用的非 *LAST_INSERT_ID()* 函数莫属，这个函数的唯一功能就是返回 MySQL 为上一条 *INSERT* 命令生成的 *AUTO_INCREMENT* 值。*LAST_INSERT_ID()* 是一个面向连接的函数，只对本次连接会话有效。这句话有两层含义：①在本次连接中，这个函数的返回值将永远是因为执行的上一条 *INSERT* 命令而生成的 *AUTO_INCREMENT* 值，即使还有其他用户也在使用 *INSERT* 命令往同一个数据表里插入新记录也是如此；②如果与 MySQL 服务器之间的连接断开了（不管什么原因），就算立刻重新连接 MySQL 服务器并执行这个函数，它的返回值也可能发生变化（除非在此期间没有人往刚才使用的数据表里插入任何新记录）。¹

以下命令将把一本新书收录到数据库里，它包含不少新信息（3位新作者、1家新出版公司）。这本书是 Randy Yanger、George Reese、Tim King 合著的 *MySQL & mSQL*, O'Reilly 出版公司 1999 年出版。

```
INSERT INTO publishers (publName) VALUES ('O'Reilly & Associates')
SELECT LAST_INSERT_ID()
        4      ---- publID for the publisher
INSERT INTO authors (authName) VALUES ('Yanger R.')
SELECT LAST_INSERT_ID()
        12     ---- authID for the first author
INSERT INTO authors (author) VALUES ('Reese G.')
SELECT LAST_INSERT_ID()
        13     ---- authID for the second author
INSERT INTO authors (author) VALUES ('King T.')
SELECT LAST_INSERT_ID()
        14     ---- authID for the third author
INSERT INTO titles (title, publID, year)
VALUES ('MySQL & mSQL', 4, 1999)
SELECT LAST_INSERT_ID()
```

1. 原文这里说的不够细致，而这一细节是每一本数据库书都会特别说明的。——译者注

```

    9 <--- titleID for the book
INSERT INTO rel_title_author
VALUES (9, 12), (9, 13), (9,14)

```

当然，以手动方式输入这么多的命令在测试阶段还可以试试，在实际工作中还这样就太麻烦了。在实际工作中，查询和计算 *ID* 值的工作应该交给一个 PHP 或 Perl 脚本或是交给一个 Java 程序来负责，这个脚本/程序的另一项任务是确保那些已被收录在了数据库里的作者和出版公司不会因为种种失误而被重复存入。（在本书的第 15 章里有一个用 PHP 语言编写的这种输入处理程序。）

9.8.3 修改数据记录（*UPDATE*）

我们可以用 *UPDATE* 命令来修改数据库里的现有数据记录。这个命令的常用语法如下所示：

```

UPDATE tablename
SET column1=value1, column2=value2, ...
WHERE columnID=n

```

这个语法的含义是：在 *tablename* 数据表里把 *columnID* 字段值等于 *n* 的那条数据记录的 *column1*、*column2* 字段分别修改为 *value1*、*value2*。在下面的例子中，将把图书 *Linux* (*titleID=1*) 的名字修改为 *Linux, 6th ed.*：

```

USE mylibrary
UPDATE titles SET title='Linux, 6th ed.' WHERE titleID=1

```

不带 *WHERE* 子句的 *UPDATE* 命令将对给定数据表里的全体数据记录做出修改（此处要小心）。比如说，下面这条命令将把所有图书的出版日期全部改为 2005 年：

```
UPDATE titles SET year=2005
```

在 *UPDATE* 命令里还允许进行少数几种数学运算。比如说，假设在 *titles* 数据表里还存放着图书的价格，下面这条命令将把所有图书的价格上调 5%：

```
UPDATE title SET price=price*1.05
```

1. 编辑排序清单里的数据记录（*UPDATE ... ORDER BY ... LIMIT*）

如果只想对满足某种排序条件的前 *n* 条或后 *n* 条数据记录进行修改，可以给 *UPDATE* 命令加上必要的 *ORDER BY* 和 *LIMIT* 子句（仅适用于 MySQL 4.0 及更高版本）。下面这条命令将把 *tablename* 数据表里对 *name* 字段按字母表排序时的前 10 条数据记录的 *mydata* 字段设置为零：

```
UPDATE tablename SET mydata=0 ORDER BY name LIMIT 10
```

2. 更新关联数据表里的数据记录

还可以在一条 *UPDATE* 命令里对多个数据表里的相关记录做出修改（仅适用于 MySQL 4.0 及更高的版本）。下面这条命令将对数据表 *table1* 里的一些数据记录的 *columnA* 字段做出修改，新数据来自 *table2.columnB* 数据列，这两个字段之间的关系通过两个数据表里的同名 *ID* 字段 *table1ID* 建立（在第 10 章还有一个更有普遍适用意义的多数据表 *UPDATE* 命令示例）：

```

UPDATE table1, table2
SET table1.columnA = table2.columnB
WHERE table1.table1ID = table2.table1ID1

```

9.8.4 删除数据记录（*DELETE*）

说 *DELETE* 命令是语法最简单的命令应该不会有人提出异议，它的用途就更没有什么可说的了：

¹. 原书中的这段示例代码有错误！——译者注

把满足 *WHERE* 条件的记录全部删掉。不需要给出任何数据列的名字（事实上，这也不允许），因为整个数据记录都将被删得不留痕迹。下面这条命令将从 *titles* 数据表里删除一条记录：

```
USE mylibrary
DELETE FROM titles WHERE titleID=8
```

注意 不带 *WHERE* 子句的 *DELETE* 命令会把数据表里的所有记录全部删掉！MySQL 没有提供任何可以用来恢复被删除记录的“撤销”机制。不过，*DELETE* 命令不会删除数据表本身以及里面的数据列定义、索引定义等。如果想让某个数据表“消失”，就必须执行 *DROP TABLE* 命令。

1. 删除关联数据表里的数据记录

在 MySQL 3.23 及以前的版本里，每条 *DELETE* 命令只能对一个数据表进行操作。从 MySQL 4.0 开始，*DELETE* 命令新增加了如下所示的语法：

```
DELETE t1, t2 FROM t1, t2, t3 WHERE condition1 AND condition2 ...
```

这个语法将把数据表 *t1* 和 *t2* 里满足 *WHERE* 条件的所有数据记录全部删掉，而 *WHERE* 条件可以是数据表 *t1*、*t2* 和 *t3* 之间的任何关联/引用关系。（简单地说，*DELETE* 命令将只从出现在关键字 *FROM* 之前的数据表里删除数据记录，但在 *WHERE* 条件里允许使用任何一个出现在关键字 *FROM* 之后的数据表的名字。）

下面这条命令例子将从 *title* 数据表里把由作者 *Kofler Michael* 写的书全部删掉。为了找出需要删除的图书记录，在这里用到了一个涉及 *titles*、*rel_title_author* 和 *authors* 等 3 个数据表的 *WHERE* 子句：

```
USE mylibrary

DELETE titles FROM titles, rel_title_author, authors
WHERE titles.titleID=rel_title_author.titleID
  AND authors.authID=rel_title_author.authID
  AND authors.authName='Kofler Michael'
```

下面这条命令不仅将从 *titles* 数据表里把由作者 *Kofler Michael* 写的书全部删掉，还将从 *authors* 数据表里把这位作者本人也删掉，*rel_title_author* 数据表里与作者 *Kofler Michael* 和他写的书有关的所有记录也将同时被删得干干净净。这两条命令之间的唯一区别是出现在关键字 *FROM* 之前的数据表个数不一样：在上面那条命令里只有 1 个，而下面这条命令里是 3 个：

```
DELETE titles, rel_title_author, authors
  FROM titles, rel_title_author, authors
 WHERE titles.titleID=rel_title_author.titleID
   AND authors.authID=rel_title_author.authID
   AND authors.authName='Kofler Michael'
```

在 *DELETE* 命令里，数据表之间的关联关系也可以通过 *JOIN* 操作符来建立。

注意 上面给出的两条示例命令在语法上都是正确的，但在执行它们时却会看这样的出错消息：*a foreign key constraint fails*（意思是“违反了一个外键约束条件”）。这是因为当初在创建 *rel_title_author* 数据表时曾为它定义过两个外键约束条件：其一用来确保 *rel_title_author* 数据表仍在引用的图书记录里，不会从 *titles* 数据表里被删除；其二用来确保 *rel_title_author* 数据表仍在引用的作者记录里，不会从 *authors* 数据表里被删除。因此，必须最先删除 *rel_title_author* 数据表里的有关记录才行，而这又需要单独使用一条 *DELETE* 命令才能做到。

有几种办法可以绕过这个困难。

首先，可以通过 *SET foreign_key_check=0* 命令暂时关闭 MySQL 的外键约束条件检查机制，等执

行完`DELETE`命令之后再用`SET foreign_key_check=1`命令重新启用该功能。`(SET`是一个独立的SQL命令。)

其次，可以在定义外键约束条件时加上`ON DELETE CASCADE`选项，但这有可能把导致其他数据表里与外键相关联、但与本次删除操作无关的数据记录也被自动删掉。

第三种办法是彻底抛弃外键或者是从一开始就使用MyISAM数据表（MyISAM数据表不支持数据一致性规则）。

2. 删除排序清单里的数据记录（`DELETE ... ORDER BY ... LIMIT`）

在`UPDATE`命令里也可以使用`ORDER BY`和`LIMIT`子句（仅适用于MySQL 4.0 及更高版本），这使我们可以有选择地删除某个数据表的前`n`条或后`n`条记录。在这么做的时候，可以选用任何一种排序方式。下面这条命令将从`authors`数据表里把最新插入或修改过的一条记录删掉（这里利用了`ts`数据列里存放的最近一次修改时间）：

```
DELETE FROM authors ORDER BY ts DESC LIMIT 1
```

9.9 创建数据表、数据库和索引

在实际工作中，人们往往更喜欢使用MySQL Administrator、phpMyAdmin或其他的管理工具来创建或改变数据库和数据表，因为它们可以减少人们在构造各种`CREATE`和`ALTER`命令时的麻烦和失误。

在某些场合，直接使用`mysql`程序去创建或改变数据库和数据表会更加简便快捷。还有些时候，或许不得不使用一个客户端程序（比如通过一个PHP脚本）去改变数据库。这一切都要求用户必须熟悉`CREATE`和`ALTER`命令的语法。在下面的内容里，将只讨论`CREATE`和`ALTER`命令最重要的语法变体，它们最重要的选项汇总在第21章。

注解 不管选用的是哪一种工具，在动手之前，请务必把访问权限问题弄清楚。如果MySQL服务器是按照安全模式配置的而你又不是MySQL数据库系统的管理员，那将根本无法创建一个新的数据库。对MySQL访问权限和信息安全问题的讨论见第11章。

对于现有的数据库或数据表，可以在SQL代码里使用`SHOW CREATE DATABASE name`或`SHOW CREATE TABLE name`命令去查看¹数据库或数据表的定义声明。

9.9.1 创建数据库（`CREATE DATABASE`）

必须先创建数据库，才能够在这个数据库里创建数据表。创建数据库的操作将在硬盘上创建出一个子目录来。为了创建一个名为`mylibrary`的数据库，需要使用如下所示的命令：

```
CREATE DATABASE mylibrary
```

还可以在创建数据库时“顺便”为它指定一个默认的字符集和排序方式，如下所示：

```
CREATE DATABASE mylibrary
DEFAULT CHARACTER SET latin1 COLLATE latin1_general_ci
```

下面这条命令将把`mylibrary`数据库设置为默认数据库，而在此之后发出的SQL命令都将对这个数据库进行操作：

1. 原文这里是“创建”，显然错误。——译者注

```
USE mylibrary
```

9.9.2 创建数据表 (CREATE TABLE)

新数据表要用 *CREATE TABLE* 命令来创建。这个命令的基本语法如下所示：

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tblname (
    colname1 coltype coloptions reference,
    colname2 coltype coloptions reference...
    [ , index1, index2 ... ] )
[ ENGINE = MyISAM|InnoDB|HEAP ]
[ DEFAULT CHARSET = csname [ COLLATE = colname ]]
```

CREATE TABLE 命令中的大部分内容都是对新数据表里的数据列（名字、数据类型、属性）和索引（索引的名字和用圆括号括起来的被索引数据列的名字，比如 *publIDIndex(publID)*）的定义和描述。为了让大家对这个语法有一个直观的认识，下面来看一个例子。下面的命令将生成 *titles* 数据表：

```
CREATE TABLE titles (
    titleID INT          NOT NULL AUTO_INCREMENT,
    title   VARCHAR(100) NOT NULL,
    subtitle VARCHAR(100),
    edition TINYINT,
    publID  INT,
    catID   INT,
    langID  INT,
    year    INT,
    isbn    VARCHAR(20),
    comment VARCHAR(255),
    ts      BLOB,
    PRIMARY KEY (titleID),
    KEY publIDIndex (publID),
    KEY langID (langID),
    KEY catID (catID),
    KEY title (title),
    CONSTRAINT titles_ibfk_1 FOREIGN KEY (publID)
        REFERENCES publishers (publID),
    CONSTRAINT titles_ibfk_2 FOREIGN KEY (langID)
        REFERENCES languages (langID),
    CONSTRAINT titles_ibfk_3 FOREIGN KEY (catID)
        REFERENCES categories (catID)
ENGINE = InnoDB
DEFAULT CHARSET = latin1 COLLATE = latin1_german1_ci
```

请注意，上面这条命令定义了几个外键约束条件 (*CONSTRAINT ...*)，这需要假设 *categories*、*languages* 和 *publishers* 等几个数据库已经存在。否则，就不能在创建 *titles* 数据表的时候定义这些外键约束条件——可以先创建这个数据表，然后等那几个数据表创建出来之后再用 *ALTER TABLE* 命令把这些外键约束条件增加上。

用SELECT结果来创建新数据表

MySQL 允许我们把 *SELECT* 命令的查询结果直接转存为一个新的数据表（在实际工作中，人们经常用这个办法来临时保存某个复杂查询的中间结果），具体做法很简单：先写出 *CREATE TABLE* 和新数据表的名字，再像往常一样接着写出那条 *SELECT* 命令就可以了。下面这条命令将把 *catID=1* 的所有图书——也就是 *Computer books*——复制到新数据表 *computerbooks*：

```
USE mylibrary
```

```
CREATE TABLE computerbooks
SELECT * FROM titles WHERE catID=1
```

如此创建的新数据表可以立刻用于查询，具体到这个例子，可以用下面这条 *SELECT* 命令来确认想要

的数据已经在新数据表里了：

```
SELECT title FROM computerbooks
```

title
Linux, 5th ed.
LaTeX
Mathematica
Maple
Practical UNIX & Internet security

等不再需要 *computerbooks* 数据表的时候，随时都可以用下面这条命令把它删掉：

```
DROP TABLE computerbooks
```

MySQL 能够根据 *SELECT* 命令的查询结果来确定和设置新数据表里数据列属性，但偶尔会出现新、老数据表在某些细节上略有差异的情况。

9.9.3 创建索引 (CREATE INDEX)

索引既可以随数据表一同创建，也可以在稍后用 *CREATE INDEX* 或 *ALTER TABLE* 命令来创建，这 3 个办法里的索引描述语法是完全一样的。下面的 3 条命令有着完全一样的效果，它们都将为 *titles* 数据表里的 *title* 数据列创建出一个名为 *idxtitle* 的索引。（当然，只须执行其中的一条命令就够了。如果又执行了一条命令，MySQL 会报告说该索引已经存在。）

```
CREATE TABLE titles (
    titleID ..., title ..., publID ..., year ...,
    PRIMARY KEY \dots,
    INDEX idxtitle (title)
)
CREATE INDEX idxtitle ON title (title)
ALTER TABLE titles ADD INDEX idxtitle (title)
```

SHOW INDEX FROM tablename 命令将生成一份现有索引的清单。现有的索引可以用 *DROP INDEX indexname ON tablename* 命令删除。

如果在创建索引时想让它只对被索引字段的（比如说）前 16 个字符而不是全部字符进行索引，可以使用下面这样的命令进行变更（如果索引已经存在，先删除，再重新创建）：

```
ALTER TABLE titles ADD INDEX idxtitle (title(16))
```

9.9.4 变更数据表的结构 (ALTER TABLE)

ALTER TABLE 命令可以用来改变一个数据表的许多设计细节，如增加/删除一些数据列、改变数据列的属性（比如数据类型）、定义和删除各种索引等。下面这条命令把 *titles* 数据表里的 *title* 数据列的最大字符个数增加到了 150 个字符：

```
ALTER TABLE titles CHANGE title title VARCHAR(150) NOT NULL
```

这条 *ALTER TABLE* 命令容易让人感到困惑的地方是 *title* 出现了两次，这可不是打字错误：第一次是数据列的当前名字，第二次是数据列的新（但没有改变）名字。

下面是 *ALTER TABLE* 命令最重要的一些语法变体。

1. 增加一个数据列

```
ALTER TABLE tblname ADD newcolname coltype coloptions
    [FIRST | AFTER existingcolumn]
```

2. 修改一个数据列

```
ALTER TABLE tblname CHANGE oldcolname newcolname coltype coloptions
```

3. 删除一个数据列

```
ALTER TABLE tblname DROP colname
```

4. 增加一个索引

```
ALTER TABLE tblname ADD PRIMARY KEY      (indexcols ...)
ALTER TABLE tblname ADD INDEX      [indexname] (indexcols ...)
ALTER TABLE tblname ADD UNIQUE      [indexname] (indexcols ...)
ALTER TABLE tblname ADD FULLTEXT [indexname] (indexcols ...)
```

5. 增加一个外键约束条件

```
ALTER TABLE tblname ADD FOREIGN KEY [idxname]
    (column1) REFERENCES table2 (column2)
```

6. 删除一个索引

```
ALTER TABLE tblname DROP PRIMARY KEY
ALTER TABLE tblname DROP INDEX indexname
ALTER TABLE tblname DROP FOREIGN KEY indexname
```

7. 改变全体文本数据列上的字符集

```
ALTER TABLE tblname CONVERT TO CHARACTER SET charsetname
```

8. 改变数据表的类型 (MyISAM、InnoDB)

```
ALTER TABLE tblname ENGINE typename
```

如果 MyISAM 数据表包含着全文索引或地理数据，从 MyISAM 到 InnoDB 的转换将不能成功。(InnoDB 数据表不支持这些功能。)

如果需要对大量的数据表进行类型转换，UNIX/Linux 环境下的 mysql_convert_table_format 脚本很值得选用。如果没有在这个脚本的命令行上给出任何数据表的名字，它就会把数据库里的所有数据表转换为指定类型，如下所示：

```
root# mysql_convert_table_format [opt] --type=InnoDB dbname [tblname]
```

注意 千万不要修改mysql数据库里数据表的数据表类型！在这些数据表里存放着MySQL的内部管理信息（用户和访问权限），它们必须是MyISAM格式。

9.9.5 删除数据库和数据表 (DROP)

下面两条命令中的第一个用来删除一个数据表，第二个用来删除整个数据库。这种删除是不可逆的：

```
DROP TABLE tblname
DROP DATABASE dbname
```

9.9.6 自动修改数据表设计 (默许的数据列修改)

在创建 (CREATE TABLE) 或修改 (ALTER TABLE) 一个数据表的时候，MySQL 会在特定条件下对这个数据表的设计方案自动做出一些修改，其理由或者是那么做可以让数据表的效率更高，或者是设计思路 MySQL 无法实现。

这里要特别提醒那些从其他数据库系统迁移过来的读者注意：MySQL 在对数据表设计方案自动做出修改时不会给出任何提示，所以一定要用 SHOW CREATE TABLE 命令去检查一下最终的数据表设计方案是不是所想像的样子。在下面的例子里，MySQL 自做主张地把一个 CHAR(20) 数据列变成了一个 VARCHAR(20) 数据列，还给那两个数据列加上了 DEFAULT NULL 属性。

```

CREATE TABLE test1 (col1 VARCHAR(20), col2 CHAR(20))
SHOW CREATE TABLE test1
CREATE TABLE `test1` (
  `col1` varchar(20) default NULL,
  `col2` varchar(20) default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1

```

下面是 MySQL 会对数据表设计方案自动做出的几项重大修改：

- 如果 $n < 4$, *VARCHAR(n)* 数据列将被修改为 *CHAR(n)* 数据列。
- 如果 $n > 3$ 并且在同一个数据表里还有其他的 *VARCHAR*、*TEXT* 或 *BLOB* 数据列的话, *CHAR(n)* 将被修改为 *VARCHAR(n)*。如果数据表里只有固定长度的数据列, *CHAR(n)* 将不发生变化。
- MySQL 允许 *TIMESTAMP* 数据列带有 *NULL* 或 *NOT NULL* 属性, 但它们没有实际的效果。这是因为 MySQL 无法把 *NULL* 值存入 *TIMESTAMP* 数据列。如果把 *TIMESTAMP* 字段的取值设置为 *NULL*, MySQL 实际存入的将是当前时间或“0000-00-00 00:00:00”。
- 即使没有设定, MySQL 也会给 *PRIMARY KEY* 数据列自动加上 *NOT NULL* 属性。
- 如果没有给某个数据列设置默认值, MySQL 就会为它自动定义一个适当的值 (*NULL*、0 或空字符串)。

MySQL 文档把这种自动方式的修改称为默许的数据列修改 (silent column changes), 这方面的详细信息可以在 <http://dev.mysql.com/doc/mysql/en/silent-column-changes.html> 网址处的文章里找到。总之, 在完成数据库的设计和创建工作之后, 一定要仔细检查数据库、数据库里的数据表、数据表里的数据列、数据列的各有关属性是不是都如所想。就是在 MySQL 的日常使用中, 也应该经常去快速检查一下数据表上的索引是不是所想像的样子。人们把这些不是关于数据表里的内容而是关于数据表自身属性的信息称为元数据 (metadata)。

9.9.7 SHOW 命令

查看元数据的传统办法是执行相应的 *SHOW* 命令。比如说, *SHOW DATABASE* 命令将返回一份当前用户可以去访问的全体数据库的清单; *SHOW TABLES* 命令将返回一份当前数据库里的全体数据表的清单; *SHOW COLUMNS* 命令将返回一份当前数据表里的全体数据列的清单等。

表 9-3 对用来查看数据库和数据表元数据的命令进行了汇总。这些命令中的大多数都允许使用 *LIKE pattern* 选项对其返回结果做出筛选。比如说 *SHOW DATABASE LIKE 'test%'* 命令将只把名字以 *test* 开头的数据库全部列出来。下面是一个 *SHOW* 命令的用法示例:

SHOW COLUMNS FROM mylibrary.titles					
Field	Type	Null	Key	Default	Extra
titleID	int(11)		PRI	NULL	auto_increment
title	varchar(100)		MUL		
subtitle	varchar(100)	YES		NULL	
edition	tinyint(4)	YES		NULL	
publID	int(11)	YES	MUL	NULL	
catID	int(11)	YES	MUL	NULL	
langID	int(11)	YES	MUL	NULL	
year	int(11)	YES		NULL	
isbn	varchar(20)	YES		NULL	
comment	varchar(255)	YES		NULL	
ts	timestamp	YES		CURRENT_TIMESTAMP	

这些命令的准确语法可以在第 21 章查到, 在那里还会再介绍几个其他的 *SHOW* 命令。比如说, *SHOW COLLATIONS* 命令将返回一份 MySQL 所支持的全体排序方式的清单, *SHOW WARNINGS* 命令将返回刚刚执行的那条命令所返回的警告信息。

表 9-3 SHOW 命令

命 令	功 能
<code>SHOW DATABASES</code>	返回一份当前用户可以去访问的全体数据库清单
<code>SHOW TABLES FROM dbname</code>	返回一份 <code>dbname</code> 数据库里的全体数据表清单
<code>SHOW [FULL] COLUMNS FROM tablename</code>	返回 <code>tablename</code> 数据表里的全体数据列的详细信息
<code>SHOW INDEX FROM tablename</code>	返回 <code>tablename</code> 数据表的全体索引的信息

9.9.8 INFORMATION_SCHEMA 数据表家族

虽然各种各样的 `SHOW` 命令对人们很有帮助，但它们不符合 SQL:2003 标准。它们各有各的语法，在功能上也有一定的局限性：在 MySQL 5.0 之前的版本里，每一种 `SHOW` 命令进行的查询都无法用功能强大的 `SELECT` 命令去完成，后者只能查询数据库里面的数据，对数据库系统本身的元数据无能为力。

这种情况在 MySQL 5.0 引入了一批 `INFORMATION_SCHEMA` 数据表之后得到了极大的改观，人们现在可以使用 `SELECT` 命令从这些数据表里检索关于数据库、数据表、数据列等的元数据。与 `SHOW` 命令相比，`SELECT` 命令有一整套面面俱到的语法变体可供选用，对那些 `INFORMATION_SCHEMA` 数据表进行查询所返回的信息也比使用 `SHOW` 命令时的丰富得多。下面的命令提供了一个例子：

```
SELECT column_name, ordinal_position, column_default,
       data_type, collation_name
  FROM information_schema.columns
 WHERE table_schema="mylibrary" AND table_name="titles"

column_name  ordinal_position  is_nullable  data_type  collation_name
titleID          1            YES        int         NULL
title           2            YES        varchar    latin1_german1_ci
subtitle         3            YES        varchar    latin1_german1_ci
edition          4            YES        tinyint   NULL
publID           5            YES        int         NULL
catID            6            YES        int         NULL
langID           7            YES        int         NULL
year             8            YES        int         NULL
isbn             9            YES        varchar   latin1_german1_ci
comment          10           YES        varchar   latin1_german1_ci
ts                11           YES        timestamp NULL
```

表 9-4 对这些数据表的内容进行了汇总。表中的 `information_schema` 是一个虚拟数据库而不是一个真实存在的文件。因此，`SHOW DATABASES`、`SELECT * FROM information_schema.schemata`、`USE information_schema` 等命令对这个数据库无法使用。

`information_schema` 数据表不允许修改 (`INSERT`、`UPDATE` 等命令不允许使用)。在访问这些数据表和数据列的时候，无须区分字母的大小写形式。这些数据表的清单可以用 `SHOW TABLES FROM information_schema` 命令来查看，数据列清单可以用 `SHOW COLUMNS FROM information_schema.tablename` 命令来查看；这么看来，`SHOW` 命令还是有它的用处的。

`information_schema` 数据库里的数据表和数据列的名字都符合 SQL:2003 标准中的有关规定，但 SQL:2003 标准所定义的数据表和数据列目前还没有在 MySQL 里全部体现出来。此外，MySQL 还在有关的数据表里增加了一些数据列来保存自己独有的信息，如“数据表类型”列 (`Engine = MyISAM`、`InnoDB` 等)。

表 9-4 *information_schema* 数据表家族

数据表	内 容
<i>information_schema.schemata</i>	所有可用数据库的元数据
<i>.tables</i>	描述了所有数据表的属性
<i>.columns</i>	描述了所有数据列的属性
<i>.statistics</i>	关于数据表索引的统计信息
<i>.views</i>	描述了所有视图的属性
<i>.table_constraints</i>	所有数据表的所有主索引、唯一化索引、外键索引的清单
<i>.key_column_usage</i>	所有索引清单，比 <i>table_constraints</i> 数据表包含更多的细节
<i>.referential_constraints</i>	描述了所有的外键约束条件（但目前尚未实现）
<i>.user_privileges</i>	用户管理和权限
<i>.schema_privileges</i>	全体 MySQL 用户的名单（数据来自 <i>mysql.user</i> 数据表）
<i>.table_privileges</i>	数据库级权限（数据来自 <i>mysql.db</i> 数据表）
<i>.column_privileges</i>	数据表级权限（数据来自 <i>mysql.tables_priv</i> 数据表）
<i>.character_sets</i>	数据列级权限（数据来自 <i>mysql.columns_priv</i> 数据表）
<i>.collations</i>	字符集和排序方式
<i>.collations_character_set_applicability</i>	所有可供选用的字符集清单
<i>.routines</i>	所有可供选用的排序方式清单
<i>.parameters</i>	哪些排序方式可以配合哪些字符集使用
<i>.triggers</i>	存储过程和触发器
	关于所有存储过程的信息（对应于 <i>mysql.proc</i> 数据表的内容）
	关于存储过程的参数的信息（但目前尚未实现）
	所有触发器的元数据（但目前尚未实现）

information_schema 数据表对全体 MySQL 用户开放，各用户的 MySQL 访问权限对它们没有效力。换句话说，只要是某个 MySQL 服务器上的注册用户，就可以使用 *SELECT* 命令去访问该 MySQL 服务器上的所有 *information_schema* 数据表。

第 10 章

SQL 解决方案

本章收录了许多 SQL 小示例，它们有一部分是初级入门水平，还有一部分则是比较高级的应用解决方案。编写本章的目的主要有两个：一是为了让读者能够对 SQL 语言向人们提供的各种可能性有一个比较直观的认识；二是向读者提供一些常见 SQL 问题的解决方案。

本章示例中的绝大多数都以本书第 8 章所介绍的几个示例数据库为基础，有兴趣的读者不妨亲自试用一下这些命令。当然，为了介绍一些比较特殊的技巧，本章中的部分示例也用到了几个临时想像出来的数据库和数据表。

10.1 字符串

在 MySQL 所支持的各类函数当中，用来对查询命令和查询结果中的字符串进行处理的函数占了相当大的比例。以下示例只介绍了这些函数的一部分，对字符串处理类 MySQL 函数的汇总见第 21 章。

10.1.1 基本函数

1. 合并字符串

CONCAT(s1, s2, ...) 函数把给定字符串合并为一个字符串。这个函数在需要合并多个数据列的时候（比如把分别存放在两个数据列里的姓氏和名字合并为一个完整的姓名）很有用：

```
SELECT CONCAT(firstname, ' ', lastname) FROM addresses
```

2. 截取字符串

SUBSTRING(s, pos, n) 函数的返回值是给定字符串 *s* 从位置 *pos*（字符串里的第一个字符的 *pos=1*，而不是 *pos=0*）算起的 *n* 个字符。下面这条命令将从 *mylibrary* 数据库里把所有图书名字的前 10 个字符截取出来：

```
USE mylibrary
SELECT SUBSTR(title, 1, 10) FROM titles
```

熟悉 Basic 语言的读者可以用 *LEFT()*、*RIGHT()* 和 *MID()* 等函数代替 *SUBSTRING()* 函数去读取位于字符串的开头或末尾的子串。本章稍后的内容里有一个这样的例子。

3. 确定字符串的长度

CHAR_LENGTH(s) 函数的返回值是字符串中的字符个数，*LENGTH()* 函数的返回值是字符串的字节长度值。对使用 *latin1* 字符集的字符串来说，这两个函数的返回值永远相等。但如果使用的是 Unicode 字符集或另外一种多字节字符集，单个字符的字节长度往往都大于 1 个字节。

4. 把字符串缩短到一个给定长度

函数 *IF(a, b, c)* 将先对条件表达式 *a* 求值，如果结果为真 (*TRUE*)，则返回 *b*；否则返回 *c*。在下

面这个例子里，将把 *titles* 数据表里的书名（*title* 字段的值）缩短到 30 个字符，不足 30 个字符的书名保持不变；查询结果中的书名将显示为“书名的前 20 个字符 … 书名的后 5 个字符”的形式：

```
SELECT IF(CHAR_LENGTH(title)>30,
          CONCAT(LEFT(title, 20), '...', RIGHT(title, 5)),
          title)
FROM titles AS shorttitle

shorttitle
Linux für Internet u ... ranet
Mathematica
Practical UNIX & Int ... urity
Visual Basic Datenba ... erung
...
```

5. 存储改变的字符串

上面几个例子都是对查询结果中的字符串进行处理。当然，也可以利用字符串函数修改 MySQL 数据表。下面这个例子将把 *mytable* 数据表里的 *mycolumn* 数据列中的 “” 字符（双引号）全部替换为 ‘’ 字符（单引号），具体替换工作由 *REPLACE()* 函数完成：

```
UPDATE mytable SET mycolumn = REPLACE(mycolumn, '"', "'")
```

下面这条命令将在字符串里搜索空格字符。如果找到这个字符，就意味着该字符串是由多个部分（单词）构成的，而这条 *UPDATE* 命令将把该字符串缩短为只包含着第一个单词。这里还用到了一个 *LOCATE()* 函数，它的返回值是匹配模板在字符串里的位置：

```
UPDATE mytable
SET mycolumn = LEFT(mycolumn, LOCATE(" ", mycolumn )-1)
WHERE LOCATE(" ", mycolumn) > 1
```

10.1.2 改变字符集

可以用 *CONVERT(x USING charset)* 函数来改变给定字符串的字符集。下面这条命令将使用 Unicode 字符集返回它的执行结果（本书中的样板数据库使用的是 *latin1* 字符集）。请注意，字符集被改变后的效果在客户端往往看不出来——这种改变只发生在 MySQL 服务器那里，由服务器发送回客户端的查询结果将按照客户端的字符集来显示其中的字符串：

```
SELECT CONVERT(title USING utf8) FROM titles
```

如果想看到 *CONVERT()* 函数的调用效果，先把字符串转换为十六进制形式再显示即可：

```
SELECT HEX(CONVERT('äöü' USING latin1))
E4F6FC
SELECT HEX(CONVERT('äöü' USING utf8))
C3A4C3B6C3BC
```

还可以利用 *CONVERT()* 函数在把数据从数据表里的某个数据列复制到另一个数据列去的同时改变它们的字符集。在下面的例子里，第一条命令给 *titles* 数据表新增加了一个名为 *title_utf8* 的数据列，第二条命令以 Unicode 格式把书名（*titles* 字段值）复制到了新的数据列里：

```
ALTER TABLE titles ADD title_utf8 VARCHAR(100) CHARSET utf8
UPDATE titles SET title_utf8 = CONVERT(title USING utf8)
```

现在，来检查一下 *titles* 和 *title_utf8* 数据列里的书名在字节长度方面有什么差异，以确定保存某一书名需要占用多少字节。与使用 *latin1* 字符集的时候相比，书名里的特殊字符在 *utf8* 格式下需要占用更多的字节：

```

SELECT title,
       LENGTH(title)      AS latinilength,
       LENGTH(title_utf8) AS utf8length FROM titlescopy
HAVING latinilength!=utf8length ORDER BY title

title          latinilength  utf8length
Alltid den där Annette           22        23
Comédia Infantil                 16        17
Dansläraren Återkomst            21        23
Das Haus meiner Väter             21        22
Gengångare                         10        11
Kärleken                           8         9
Linux für Internet und Intranet 31        32
Linux im Büro                      13        14
Nicht alle Eisbären halten Winterschlaf 39        40
PHP - Grundlagen und Lösungen     29        30

```

指定某个字符串的字符集（投射操作符）

在默认的情况下，通过 MySQL 客户端程序（mysql 程序、phpMyAdmin、PHP 脚本等）输入的字符都将使用客户端的默认字符集（参见下一小节）。如果偶尔需要在 SQL 命令里使用另一种字符集的字符或字符串，就必须给它们加上一个投射操作符作为前缀以指定它们的字符集设置。这种投射操作符由一个下划线字符和那个临时字符集的名字构成（如 `_utf8` 或 `_latin1`）：

```
INSERT INTO titles (title) VALUES( _utf8 'title in UTF-8 encoding')
```

10.1.3 设置客户端字符集

在 MySQL 服务器上，不同的数据库、数据表、甚至是数据列可以使用不同的字符集，但绝大多数 MySQL 客户（比如 PHP 脚本、MySQL 命令解释器 mysql 程序等）都不具备这种同时支持多种字符集的能力，每次只能使用一种字符集。这一般不是什么大问题，因为 MySQL 客户软件库可以把从客户发往服务器和从服务器返回客户的字符串自动转换为相应的字符集编码；如果在转换时遇到了无法表示的字符，该字符将被替换为一个问号（“?”）。

客户与服务器之间的字符集转换工作由几个 MySQL 系统变量控制，这些变量列在了表 10-1 里。

表 10-1 MySQL 字符集变量

变 量	含 义
<code>@@character_set_client</code>	客户端使用的字符集
<code>@@character_set_server</code>	服务器的默认字符集
<code>@@character_set_connection</code>	客户与服务器之间的连接正在使用的字符集
<code>@@character_set_result</code>	<code>SELECT</code> 查询结果上的字符集
<code>@@character_set_database</code>	数据库的默认字符集

MySQL 客户（程序）并不是总能猜中到底想使用哪一个字符集。如果有任何疑问，最简单的办法是用下面这条命令去查看一下那几个系统变量的状态：

```
SHOW VARIABLES LIKE 'character_set\_%'
```

Variable_name	Value
<code>character_set_client</code>	<code>latin1</code>
<code>character_set_connection</code>	<code>latin1</code>
<code>character_set_database</code>	<code>latin1</code>
<code>character_set_results</code>	<code>latin1</code>
<code>character_set_server</code>	<code>latin1</code>
<code>character_set_system</code>	<code>utf8</code>

有些 MySQL 客户程序允许在它们的配置文件里或是在启动它们时指定一个字符集，比如说，可以通过`--default-character-set=utf8`选项把`mysql`或`mysqldump`程序的默认字符集设置为 UTF-8。另外一些程序则没有提供这类选项，只能在启动它们之后以手动方式去设置表 10-1 里的那几个系统变量。比如说，如果想处理 UTF-8 数据，就需要发出以下命令：

```
SET @session.character_set_client = 'utf8'
SET @session.character_set_results = 'utf8'
SET @character_set_connection = 'utf8'
```

`SET NAMES 'utf8'`命令的效果与上面这 3 条命令的一样，它一次就可以把 3 个变量都设置好。还有一个办法是`SET CHARACTER SET 'utf8'`命令，它将把`character_set_client`和`character_set_result`变量设置为`utf8`，再把`character_set_database`变量的值传递给`character_set_connection`变量；这样的设置对绝大多数场合来说都足够了。

完成这些准备工作之后，在 SQL 命令里输入的字符串将默认使用 UTF-8 字符集，`SELECT`查询结果里的字符串也将默认使用 UTF-8 字符集。

10.1.4 模板匹配

1. 用`LIKE`操作符进行模板匹配

`LIKE`操作符很适合用来完成简单的模板匹配任务。它支持两个通配符：下划线字符（`_`），匹配任意单个字符；百分号字符（`%`），匹配任意多个（包括 0 个）字符。和普通的字符串比较操作一样，`LIKE`操作符也不区分字母的大小写情况。下面是一个例子：

```
SELECT 'MySQL' LIKE '%sql'
      1
```

下面的`SELECT`命令将返回所有包含着字符串“SQL”的书名（`title`字段值）：

```
USE mylibrary
SELECT title FROM titles WHERE title LIKE '%SQL%'
```

如果字符“`_`”或“`%`”本身需要出现在匹配模板里，必须在它们的前面加上一个反斜线字符（`\`）进行转义，就像`'50%' LIKE '%\%'`这样（这个表达式的求值结果是 1）。如果想用另外一个字符代替反斜线作为转义引导字符，就要使用类似`ESCAPE:'50%' LIKE '%&%' ESCAPE '&'`（这个表达式的求值结果也是 1）的语法。

注解 模板匹配操作在比较大的数据表上执行得非常慢，`LIKE`操作符是如此，`REGEXP`操作符更是如此。在许多场合，用一个全文检索操作来代替模板匹配操作会更有效率。

2. 用`REGEXP`操作符进行模板匹配

`REGEXP`操作符（以及它的同义操作符`RLIKE`）可以用来构造更灵活的匹配模板。与`LIKE`操作符一样，`REGEXP`操作符也不区分字母的大小写情况。下面是这个操作符的几个用法示例。第 21 章有一个表格对`REGEXP`模板匹配操作中最重要的一些问题进行了汇总。

在最简单的例子中，无论字符串`pattern`是否包含在`expr`中，`expr REGEXP pattern`都执行测试。

```
SELECT 'abcabc' REGEXP 'abc',      'abcabc' REGEXP 'cb'
      1, 0
```

匹配模板不必描述整个字符串，但如果确实需要匹配整个字符串，就必须在搜索中使用“`^`”和“`$`”字符，如下所示：

```
SELECT 'abc' REGEXP '^abc$', 'abcabc' REGEXP '^abc$'
      1, 0
```

在 *REGEXP* 表达式里，方括号的含义是“多选一”；连字符用来给出一个字符范围，比如 a-c 代表从 a 到 c（包括 a 和 c）；加号表示至少要有一个字符必须在被匹配的字符串里出现至少一次：

```
SELECT 'cde' REGEXP '[a-c]+', 'efg' REGEXP '[a-c]+'
      1, 0
```

此外，在 *REGEXP* 表达式里，圆括号里的内容是一个不可分割的字符串；花括号里的数字表示它前面的字符串必须连续出现多少次：

```
SELECT 'xabcabcz' REGEXP '(abc){2}', 'xabcyabcz' REGEXP '(abc){2}'
      1, 0
```

只要在被匹配字符串里找到了与匹配模板相匹配的子串——不管具体位置如何，*REGEXP* 表达式的求值结果就将为真（*TRUE*）。

提示 在下面 MySQL 文档附录里能找到大量示例：<http://dev.mysql.com/doc/mysql/en/regexp.html>。

3. 对字符串进行二进制比较

字符串比较操作一般都不区分字母的大小写情况，所以条件表达式 'a'='A' 的求值结果是 1。如果想对字符串进行二进制比较，就必须在操作数（任何一个操作数均可）的前面加上一个关键字 *BINARY*。*BINARY* 是一个“投射”操作符，其作用是改变一个操作数的数据类型（具体效果是把一个数字或一个字符串转换为一个二进制对象）。*BINARY* 既可以在普通的字符串比较操作里使用，也可以在使用了 *LIKE* 和 *REGEXP* 操作符的模板匹配操作里使用：

```
SELECT 'a'='A', BINARY 'a' = 'A', 'a' = BINARY 'A'
      1, 0, 0
SELECT 'abcabc' REGEXP 'ABC', 'abcabc' REGEXP BINARY 'ABC'
      1, 0
```

10.2 日期和时间

这一节里的绝大多数示例都需要假设 *mytable* 数据表里有一个存放着日期/时间数据值的 *ts* 数据列（数据类型是 *DATETIME* 或 *TIMESTAMP* 均可）。

10.2.1 日期和时间的语法

SQL 命令里的日期和时间必须以 MySQL 服务器能够接受的字符串格式给出。下面 3 个查询是等效的，并且对 *DATETIME* 或 *TIMESTAMP* 数据列都适用：

```
SELECT COUNT(*) FROM mytable
WHERE ts BETWEEN '2004-05-16 08:34:07' AND '2005-02-11 00:15:44'
SELECT COUNT(*) FROM mytable
WHERE ts BETWEEN '2004/05/16 08:34:07' AND '2005/02/11 00:15:44'
SELECT COUNT(*) FROM mytable
WHERE ts BETWEEN '20040516083407' AND '20050211001544'
```

注意，*BETWEEN a AND b* 语法里的启始时间 *a* 和结束时间 *b* 也包括在给定范围之内（即相当于 *d>=a AND d<=b*）。

1. 按年、月、日进行分组查询

在实际工作中，经常会有需要按年、月、日对有关数据记录进行某种统计的情况。这类任务当然可以用一系列 *BETWEEN start AND end* 查询来完成，但联合使用 *GROUP BY* 子句和 *COUNT()* 函数去进行分组统计的办法显然更简单明了。

第一个例子是从 *bigvote* 数据库里查出 2005 年里各个月份的投票数字。这里使用了 MySQL 函数 *MONTH()* 和 *YEAR()* 来提取日期/时间值中的年份和月份数字 (*bigvote* 数据库的结构和在第 3 章介绍的 *test_vote* 数据库一样，但收录了大约 5000 条记录)：

```
USE bigvote
SELECT COUNT(*), MONTH(ts) AS m
FROM votelanguage
WHERE YEAR(ts)=2002
GROUP BY m
```

COUNT(*)	m
152	1
227	2
...	...
58	12

如果想要所有年份的统计结果（不光是某一年的），*GROUP BY* 子句中的分组表达式就必须同时包含年份和月份，而这需要用到 *DATE_FORMAT()* 函数（这个函数的语法见第 21 章）：

```
SELECT COUNT(*), DATE_FORMAT(ts, '%Y-%m') AS m_y
FROM votelanguage
GROUP BY m_y
```

count(*)	m_y
214	2002-05
111	2002-06
226	2002-07
...
73	2006-01

2. 日期/时间数据的输出格式

DATE_FORMAT(date, format) 函数和 *TIME_FORMAT()* 函数可以帮助我们按自己的想法为日期/时间数据设置输出格式，下面 3 个例子演示了它们的用法：

```
SELECT DATE_FORMAT('2005-12-31', '%M %d %Y')
December 31 2005
SELECT DATE_FORMAT('2005-12-31', '%D of %M')
31st of December
SELECT TIME_FORMAT('02:17', '%H')
02
```

在 MySQL 里，星期几、月份等的名字一律使用英语单词给出——不管 MySQL 服务器上的语言设置 (*language* 选项) 是什么。可以在这两个函数里使用的格式字符清单见第 21 章。

10.2.2 与日期和时间有关的计算

日期/时间的计算不能简单地使用加法 (+) 或减法 (-) 等操作符来进行，这种计算必须使用特殊的 SQL 函数来完成。下面列出的只是这类函数当中几个最重要的，对它们的全面介绍见第 21 章。

□ ***ADDDATE()*、*DATE_ADD()***：这两个等效的函数用来把一个时间间隔和一个日期 (*DATETIME* 或 *TIMESTAMP*) 相加。

ADDDATE('2005-12-31 6:00', INTERVAL 3 MINUTE) 将返回 *2005-12-31 6:03*。

如果想用任意时间间隔做加法运算，需要使用 *HOUR_SECOND* 语法或使用 *ADDTIME()* 函数：

ADDDATE('2005-12-31 6:00', INTERVAL '3:15:22' HOUR_SECOND) 将返回 *2005-12-31 9:15:22*。

- **SUBDATE()**、**DATE_SUB()**: 对时间间隔做减法，其余同上。
- **ADDTIME()** (始见于 MySQL 4.1): 日期加日期或时间加时间。与 *ADDDATE()* 函数的区别：第 1 个参数现在可以是 *TIME* 数据类型；第 2 个参数不再需要关键字来引导或结束。
ADDTIME('2005-12-31 6:00', '3:15:22') 将返回 *2005-12-31 9:15:22*。
- **SUBTIME()** (始见于 MySQL 4.1): 对时间间隔做减法，其余同上。
- **DATEDIFF()** (始见于 MySQL 4.1): 返回两个日期之间的天数，计算时忽略日期中的小时部分。
DATEDIFF('2005-12-31', '2005-12-28') 将返回 *3*。
DATEDIFF('2005-12-31 00:00:00', '2005-12-28 23:59:59') 也将返回 *3*。
- **TIMEDIFF()** (始见于 MySQL 4.1): 返回两个时间 (*TIME*) 或两个日期 (*DATETIME* 或 *TIMESTAMP*) 的时间差。这个函数不能用于从一个日期减去一个时间的运算。
TIMEDIFF('2005-12-31 12:30', '2005-12-30 19:45') 将返回 *16:45:00*。
- **转换函数**: MySQL 还有许多与时间有关的函数，它们可以用来提取日期/时间数据和计算各种时间间隔: *HOUR()*、*MINUTE()*、*SECOND()*、*TIME_TO_SEC()*、*SEC_TO_TIME()*、*DAYOFYEAR()*、*LAST_DAY()*、*MONTH()*、*WEEK()*、*STR_TO_DATE()*、*UNIX_TIMESTAMP()*、*FROM_UNIXTIME()*、*UTC_DATE()*、*UTC_TIME()* 等。

提示 从 MySQL 4.1 开始，许多 MySQL 时间函数已经有能力对微秒进行处理，但因为目前还没有一种 MySQL 数据类型可以存储微秒，所以这些函数现在还派不上用场。关于这个话题的进一步信息请参见：<http://dev.mysql.com/tech-resources/articles/4.1/time.html>。

一些例子

数据表 *workingtimes* 是下面几个示例的基础，它里面已经存放了一些（比如说）上班和下班时间。

USE test

```
CREATE TABLE workingtimes (
    id INT AUTO_INCREMENT,
    begintime DATETIME,
    endtime DATETIME,
    PRIMARY KEY (id)

INSERT INTO workingtimes (begintime, endtime) VALUES
    ('2005-03-27 7:15', '2005-03-27 18:00'),
    ('2005-03-28 8:00', '2005-03-28 18:00'),
    ('2005-03-29 7:30', '2005-03-29 16:50'),
    ('2005-03-30 7:00', '2005-03-30 17:15')
```

像下面这样去计算工作时间只能是一次失败的尝试，原因很简单：MySQL 把减法操作 *endtime - begintime* 里的时间字符串解释成了整数，所以 '2005-03-27 7:15:00' 就是 20050327071500。对这样的数字做减法运算当然难不倒 MySQL，但运算结果却毫无用处——谁都知道时间值是 60 进制，不是 10 进制：

```
SELECT DATE_FORMAT(begintime, '%Y-%m-%d') AS dt,
       endtime - begintime AS s
  FROM workingtimes
```

<u>dt</u>	<u>s</u>
2005-03-27	108500
2005-03-28	100000
2005-03-29	92000
2005-03-30	101500

得到正确结果的最简单办法是用 *TIMEDIFF()* 函数：

```
SELECT DATE_FORMAT(begintime, '%Y-%m-%d') AS dt,
       TIMEDIFF(endtime, begintime) AS s
  FROM workingtimes
```

<u>dt</u>	<u>t</u>
2005-03-27	10:45:00
2005-03-28	10:00:00
2005-03-29	09:20:00
2005-03-30	10:15:00

像下面这样用 *SUM()* 函数去累计工作时间也是一场失败的尝试。MySQL 把 10:45:00 之类的东西解释成了普通的数字：

```
SELECT SUM(TIMEDIFF(endtime, begintime)) AS sumtime
  FROM workingtimes
```

```
sumtime  
39
```

为了得到正确的结果，需要先把 *TIMEDIFF()* 函数计算出来的时间差用 *TIME_TO_SEC()* 函数转换为秒数，然后把它们累加起来，最后再用 *SEC_TO_TIME()* 函数把它们转换回时间值。*SEC_TO_TIME()* 函数的返回值是 *hh:mm:ss* 格式，小时部分允许是大于 24 的数字；也就是说，它不把小时数转换为天数：

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(TIMEDIFF(endtime, begintime))))
      AS sumtime
  FROM workingtimes
```

```
sumtime  
40:20:00
```

下面这个查询对工作时间超过 10 个小时的天数进行了统计：

```
SELECT COUNT(*) FROM workingtimes
 WHERE endtime >= ADDTIME(begintime, '10:00')
```

```
COUNT(*)  
3
```

10.2.3 UNIX 时间戳

UNIX 操作系统以 1970 年 1 月 1 日作为系统时间的起点。从那时起每过一秒，UNIX 的系统时间计数器就递加一个 1，由此得到的整数被称为“UNIX 时间戳”。MySQL 用户对 UNIX 时间戳感兴趣的地方有两点：首先，它们可以直接拿过来进行计算（加减法的结果算出来就是对的）；其次，PHP 和 Perl 等程序设计语言中的许多函数都以 UNIX 时间戳作为输入参数或输出结果。

注意 UNIX 时间戳与 MySQL 中的 *TIMESTAMP* 时间值完全是两回事。MySQL 在存储一个时间值时使用的内部数据格式与 UNIX 类似，但在执行 SQL 命令的时候会把时间值转换为 '2005-12-31 23:59:59' 这样的字符串。

UNIX 时间戳在存储时其实就是一个整数，因为历史的原因，这个整数是一个 32 位整数。如此

算来，这个整数将在2038年的某一天出现溢出。不过，在1970到2038年之间，UNIX时间戳是一项紧凑的、容易使用的数据值。但未来怎么办？首先，如果把UNIX时间戳看做是一个无符号整数的话，它还有一个符号位可用，而这意味着它发生溢出的日子将推后到22世纪初。更有可能的是UNIX/Linux操作系统在最近几年就会使用一个64位的整数来存放时间戳数值，而这意味着在2038年极有可能不会再发生“千年虫”事件。

1. 转换函数

MySQL 函数 `FROM_UNIXTIME()` 和 `UNIX_TIMESTAMP()` 可以完成 MySQL 日期 (`DATE`、`DATETIME` 和 `TIMESTAMP`) 与 UNIX 时间戳的相互转换：

```
SELECT NOW(), UNIX_TIMESTAMP(NOW())
2004-12-20 11:19:53          1103537993
SELECT FROM_UNIXTIME(1103600000)
2004-12-21 04:33:20
```

2. 应用示例

以下命令使用了 10.2.2 节提到的 `workingtimes` 数据表。为了计算工作时间，这里先用 `UNIX_TIMESTAMP()` 函数把 MySQL 时间值转换成了 UNIX 时间戳：

```
SELECT DATE_FORMAT(begintime, '%Y-%m-%d') AS dt,
UNIX_TIMESTAMP(endtime) - UNIX_TIMESTAMP(begintime) AS s
FROM workingtimes
```

dt	s
2005-03-27	38700
2005-03-28	36000
2005-03-29	33600
2005-03-30	36900

上面给出的计算结果是没有错误的，只是那些秒数让人根本无法看懂。这个问题可以用 MySQL 函数 `SEC_TO_TIME()` 来解决：

```
SELECT DATE_FORMAT(begintime, '%Y-%m-%d') AS dt,
SEC_TO_TIME(UNIX_TIMESTAMP(endtime) - UNIX_TIMESTAMP(begintime)) AS t
FROM workingtimes
```

dt	t
2005-03-27	10:45:00
2005-03-28	10:00:00
2005-03-29	09:20:00
2005-03-30	10:15:00

现在，甚至可以用 `SUM()` 函数来统计工作总时间：下面是对 2005 年 3 月进行统计的结果：

```
SELECT SEC_TO_TIME(SUM(UNIX_TIMESTAMP(endtime) -
UNIX_TIMESTAMP(begintime))) AS sumtime
FROM workingtimes
WHERE begintime>='2005-03-01 00:00:00' AND
begintime<='2005-03-31 23:59:59'
```

sumtime
40:20:00

3. 避免MySQL自动更改TIMESTAMP值

如果某个数据表里有一个 `TIMESTAMP` 数据列，那么当对这个数据表执行一条 `UPDATE` 命令的时候，MySQL 就会自动更改有关记录的 `TIMESTAMP` 字段值。（一般来说，人们创建一个 `TIMESTAMP` 数据列的目的差不多总是为了记载有关数据记录的创建和上一次修改时间。）

可是，也许会有一天在修改一条记录时不想让 MySQL 自动更改它的 *TIMESTAMP* 字段。这时候，必须像下面那样明确地使用 *SET* 命令把 *TIMESTAMP* 字段设置为它们在修改有关数据记录之前的值：这里需要假设那个 *TIMESTAMP* 数据列的名字是 *ts*：

```
UPDATE table SET data='new text', ts=ts WHERE id=123
```

如果对数据表的改动量很大，这里有一个更稳妥的办法供大家参考。首先，给 *table* 数据表增加一个新的 *DATETIME* 类型的数据列（我们不妨假设它的名字是 *oldts*）并把 *ts* 数据列里的 *TIMESTAMP* 值全部复制到它里面去：

```
ALTER TABLE table ADD oldts DATETIME
UPDATE table SET ts=ts, oldts=ts
```

接下来，放心大胆地对数据表进行修改，不必考虑 *ts* 时间的问题。等修改工作全部完成之后，只须再执行下面两条命令就可以把 *ts* 数据列恢复到它原来的样子，并把已经没有用了的 *oldts* 数据列删掉：

```
UPDATE table SET ts=oldts
ALTER TABLE table DROP oldts
```

10.2.4 地理时区

MySQL 用户很少需要关心地理时区的问题，涉及日期和时间的 MySQL 函数几乎都能自动完成本地时区与其他时区的必要转换。MySQL 在存储 *TIMESTAMP* 值时使用的是 UTC（Coordinated Universal Time，全球协调时间）格式，但在执行 SQL 命令的时候会把 *TIMESTAMP* 值转换为本地时区里的时间值。

本节将对 MySQL 数据库系统中的时区管理问题做一个比较详细的讨论，而讨论的重点是如何自行设置地理时区以及这么做会影响到哪些函数和功能。

1. 服务器时区和客户端时区

在 MySQL 4.0 及以前的版本里，只有服务器的时区是可配置的。从 4.1 版本起，MySQL 开始区别对待这两种时区。

● 服务器时区（系统变量`global.time_zone`）

如果在配置 MySQL 服务器（软件）时没有明确地为它指定一个时区，它在启动时将自动使用服务器主机（硬件）上的时区设置。可以用 `SHOW @@global.time_zone` 命令去查看 MySQL 服务器的当前时区设置。这个查询的结果通常是 *SYSTEM*，但这个答案应该说并没有实际的意义：

```
SELECT @@global.time_zone
      SYSTEM
```

还好，还可以通过 `SHOW VARIABLES` 命令去查看 MySQL 服务器主机的时区设置到底是什么（见系统变量 *system_time_zone*，这个变量无法用 *SELECT* 命令查询）。在下面的例子中，MySQL 服务器正运行在 Central European（CET，中欧）时区，“古欧洲”的许多首都如巴黎、维也纳和柏林等城市均处于这个时区当中：

```
SHOW VARIABLES LIKE 'system_time_zone'
      CET
```

下面这个例子利用了函数 *NOW()* 和系统变量 *UTC_TIMESTAMP* 来计算本地系统时间与国际协调时间之间的地理时差：

```
SELECT TIMEDIFF(NOW(), UTC_TIMESTAMP)
      01:00:00
```

拥有 *SUPER* 权限的 MySQL 用户可以随时使用 *SET GLOBAL time_zone = ...* 命令改变 MySQL 服务器的

时区设置，但这么做的前提是 MySQL 服务器没有启用二进制日志机制和/或镜像（动态备份）机制。如果这两种机制之一或全部已被启用，就只能通过先修改 MySQL 配置文件（文件名是 `my.cnf` 或 `my.ini`）里的 `default-time-zone` 选项、再重新启动 MySQL 服务器的办法去改变它的时区设置。

● 客户端时区（系统变量`session.time_zone`）

在默认的情况下，MySQL 客户会自动“抄袭”MySQL 服务器的时区设置。这么做是有道理的，因为 MySQL 服务器和客户通常都运行在同一个时区里。

话虽如此，如果真需要改变客户端的时区设置，`SET time_zone = ...` 命令（注意：这里没有关键字 `GLOBAL`）可以让用户达到目的。可以用 `SHOW @@session.time_zone` 命令去查看本地系统的当前时间。

2. 设置客户端时区

MySQL 经常被人们用来建设网络数据库。这时候，MySQL 服务器和 Web 服务器或者运行在同一台计算机上，或者运行在不同的计算机上，但无论如何都是运行在同一个地理时区里。对数据库用户来说，即使他们散布在世界各地，即使他们在连接和访问 MySQL 服务器时需要调用一些 PHP 或 Java 程序，那些程序也只能运行在与 MySQL 服务器同处一个时区的那台配套的 Web 服务器上，用户不必关心时区管理的问题。

可是，如果是一名 Web 应用程序的开发者，就不能不考虑 MySQL 服务器的时区管理问题，就不能不考虑怎样才能用好 MySQL 服务器的时区管理能力的问题：应该让 Web 用户有机会看到他们自己的当地时间。通过这里的学习，大家想必已经知道这其实是一件很简单的事情：在 Web 用户与 MySQL 服务器建立连接后执行一条必要的 `SET time_zone = ...` 命令；这条 SQL 命令已足以让 Web 用户看到他们自己的当地时间。

在下面的例子里，MySQL 服务器运行在 CET 时区里，所以 SQL 函数 `NOW()` 第一次返回的是 CET 时间。把 `time_zone` 变量调整为 GMT-8 时区之后，`NOW()` 函数返回的将是洛杉矶、西雅图、温哥华等城市的当地时间。

```
SELECT NOW()
2005-12-07 16:29:54
SET time_zone='-8:00'
SELECT NOW()
2005-12-07 07:29:57
```

注意，`SET` 命令只改变 `session.time_zone` 变量，不会改变 `global.time_zone` 变量。用 `SET` 命令做出的时区设置在 MySQL 客户与 MySQL 服务器之间的连接断开后将丢失。

3. 设置 MySQL 服务器时区

□ Windows。如果 MySQL 服务器运行在 Windows 环境下，就只能以 UTC (Coordinated Universal Time, 全球协调时间) 格式来设置系统时区。比如说，“+1:00”是柏林、“0:00”是伦敦、“-5:00”是纽约等。

□ Linux。如果 MySQL 服务器运行在 UNIX/Linux 环境下并经过了适当的配置（参见第 14 章），就可以使用操作系统的时区名来设置 MySQL 服务器的时区。比如说，“European/Berlin”或“CET”是柏林、“European/London”或“Greenwich”是巴黎、“America/New_York”是纽约等。

4. 不同时区的自动转换和计算

表 10-2 列出了会受到时区设置影响的 MySQL 数据类型和函数。

表 10-2 地理时区的调整

功 能	会根据时区设置做出自动调整	不会根据时区设置做出自动调整
存储日期/时间数据	<i>TIMESTAMP</i> 数据列	<i>DATE</i> 、 <i>TIME</i> 和 <i>DATETIME</i>
确定当前时间	<i>NOW()</i> 、 <i>CURDATE()</i> 、 <i>CURRENT_xxx()</i> 、 <i>LOCALTIME()</i> 、 <i>SYSDATE()</i> 等函数	<i>UNIX_TIMESTAMP()</i> 、 <i>UTC_DATE()</i> 、 <i>UTC_TIME()</i> 、 <i>UTC_TIMESTAMP()</i> 等函数
<i>DATETIME</i> 与 UNIX 时间戳之间的相互转换	<i>UNIX_TIMESTAMP(datetime)</i> 、 <i>FROM_UNIXTIME(timestamp)</i>	
日期/时间的计算与输出格式		所有其他的日期/时间函数，如 <i>ADDDATE()</i> 、 <i>SUBDATE()</i> 、 <i>HOUR()</i> 、 <i>MINUTE()</i> 、 <i>DATE_FORMAT()</i> 、 <i>TIME_FORMAT()</i> 、 <i>CONVERT_TZ()</i> 等

5. 不同时区的手工转换

不管 MySQL 客户和服务器的时区设置如何，都可以用 *CONVERT_TZ()* 函数来转换不同时区的时间。这个函数的第一个参数是一个时间值（日期加时间，只给出时间是不行的），第 2 个参数是第一个参数所在的时区，第 3 个参数是准备转换过去的时区。

在下面的例子中，将把一个柏林时间转换为纽约时间：柏林的中午 12 点在纽约是早上 6 点：

```
SELECT CONVERT_TZ('2005-12-31 12:00', 'Europe/Berlin', 'America/New_York')
2005-12-31 06:00:00
```

10.3 ENUM 和 SET 数据类型

第 8 章介绍过的 *ENUM* 和 *SET* 数据类型可以帮助人们高效率地完成两大类任务：从 n 个字符串里选择其中之一（“ n 选 1”问题， $n < 65\ 536$ ）；从 n 个字符串里选择任意多个（“ n 选 m ”问题， $n < 256$ ）。本节将向大家介绍一些 *ENUM* 和 *SET* 数据类型的使用技巧。

10.3.1 ENUM

ENUM 数据类型可以管理一个成员多达 65 535 个的字符串集合，集合中的每个字符串都有一个顺序排列的编号，而 MySQL 实际存入 *ENUM* 数据列的正是这些编号而不是那些字符串本身。这种做法既节约了空间又加快了处理速度。注意，*ENUM* 数据列里的每一个字段只能存入一个字符串——更准确地说是那个字符串的编号，在同一个字段里存入多个字符串是不允许的。（如果打算保存的是多个字符串的一种组合，就应该使用一个 *SET* 数据列。）

在涉及字符串比较操作的查询里，MySQL 不区分 *ENUM* 数据的字母大小写情况。除了在集合里预先定义的那些字符串，空白字符串（注意，不是空字符串）总是可以存入 *ENUM* 字段，但只有不带 *NOT NULL* 属性的 *ENUM* 数据列才允许 *NULL*（注意：这才是空字符串）存入。

ENUM 字段在使用时与其他字符串字段没有什么两样。在下面的例子中，创建了一个包含着一个 *ENUM* 数据列（它的名字是 *color*）的数据表 *testenum*，插入了一个新记录并进行了检索。具体到这个例子，*color* 字段的取值只能是我们预先定义的 5 种颜色之一：

```
CREATE TABLE testenum
(color ENUM ('red', 'green', 'blue', 'black', 'white'))
INSERT testenum VALUES ('red')
SELECT * FROM testenum WHERE color='red'
```

10.3.2 SET

SET 数据类型与 *ENUM* 数据类型有很多相似之处，它们之间的唯一区别是我们可以把属于同一个集合的任意多个字符串的组合存入 *SET* 字段。MySQL 在其内部会对 *SET* 集合里的字符串按 2 的幂次进行编号，这么做的好处是可以利用各种二进制按位比较操作来处理 *SET* 数据（MySQL 也正是这么做的）；弊端是可用空间浪费巨大（一个字节只能对应 8 个字符串），同一个 *SET* 字段最多只能容纳由 64 个字符串构成的一种组合（按每个 *SET* 字段占用 8 个字节计算）。

在把多个字符串的组合存入 *SET* 字段的时候，字符串之间必须用逗号隔开（并且不允许有空格）。在插入新记录时，*SET* 字段里的字符串先后顺序无关紧要；在存储时和在查询结果里，那些字符串将按它们在 *SET* 集合的定义顺序排列显示：

```
CREATE TABLE testset
  (fontattr SET ('bold', 'italic', 'underlined'))
INSERT testset VALUES ('bold,italic')
```

在查询命令里使用“=”操作符来比较 *SET* 数据的意思是两组字符串组合必须完全匹配，查询结果里将只包含满足这一条件的记录。比如说，如果 *testset* 数据表里只有我们刚刚插入的那条 ‘bold,italic’ 记录，下面这个查询将不会返回任何结果：

```
SELECT * FROM testset WHERE fontattr='italic'
```

如果真想查找 *SET* 字段包含着某给定字符串（不管它与其他字符串如何组合）的记录，就应该使用 MySQL 函数 *FIND_IN_SET()*。这个函数将返回给定字符串在 *SET* 集合里的定义顺序（具体到这个例子，‘bold’ 对应着序号 1，‘italic’ 对应着序号 2）：

```
SELECT * FROM testset WHERE FIND_IN_SET('italic', fontattr)>0
```

ENUM 值和 *SET* 值在 MySQL 内部被表示为整数而不是字符串。如果想通过查询去确定这些整数的值，就需要使用 *SELECT x+0 FROM table* 这样命令，这里的 *x* 是 *ENUM* 或 *SET* 数据列的名字。用 *INSERT* 和 *UPDATE* 命令直接把一个整数值存入 *ENUM* 或 *SET* 字段也是允许的。

如果想知道某个 *ENUM* 或 *SET* 集合里都包含着哪些成员字符串，就必须找 *DESCRIBE tablename columnname* 命令来帮忙。该命令将返回一个对 *columnname* 数据列做出详细描述的表格，表格中的 *Type* 列包含着 *ENUM* 或 *SET* 集合的定义声明。

注解 因为 MySQL 在存储和处理 *ENUM* 和 *SET* 数据时使用的不是那些字符串本身而是它们的整数编号值，所以 *ENUM* 和 *SET* 字段的内容在排序时将按照它们在 *ENUM* 或 *SET* 集合里的先后顺序，而不是按照字母表顺序排列。如果想让它们按字母表顺序排列，就必须明确地把它们转换为字符串形式，比如像这样： *SELECT CONCAT(x) AS xstr ... ORDER BY xstr*。

10.4 变量与条件表达式 (*IF*、*CASE*)

本节内容分为两大部分，一是如何在 SQL 命令里定义和使用变量；二是如何在查询命令里构造一些比较简单的 *IF* 和 *CASE* 条件子句。

提示 MySQL 5.0 的新增功能之一是允许人们使用 SQL 语言来编写“子程序”，即所谓的“存储过程”(stored procedure, SP)。MySQL 把自己的 SQL “方言” 扩展到了能够像通用程序设计语言那样

进行编程的地步，但那些语言结构只能用在存储过程里，不能用在普通的SQL命令当中。将在第13章对它们进行讨论。

10.4.1 变量

MySQL 允许人们把简单的值（离散值，不是像 *SELECT* 查询结果那样的集合或列表）保存在变量里。在日常应用里，需要用到 MySQL 变量的时候并不多；但对存储过程来说，变量却是非常重要的 SQL 元素（详见第 13 章）。MySQL 里的变量可以分为 3 大类。

- **普通变量**。这类变量的标志是以字符@开头，它们在 SQL 连接被关闭时将失去内容。
- **系统变量和服务器变量**。这类变量的内容是 MySQL 服务器的工作状态或属性，它们的标志是以两个@@字符开头（比如，@@binlog_cache_size）。

有许多系统变量都有两种形式，一个对应着当前连接（如@@session.wait_timeout），另一个对应着整个 MySQL 服务器（如@@global.wait_timeout，它包含着这个变量的默认值）。
- **存储过程里的局部变量**。这些变量是在存储过程内声明的，只在存储过程内有效。它们没有统一的特殊标志，但变量名必须与数据表和数据列的名字有区别。

在这一节里，将主要处理普通变量。存储过程里的变量将在第 13 章里详细讨论。系统变量和服务器变量将在第 14 章和第 21 章讨论。这 3 种变量的赋值和使用在语法上都是一样的。

1. 变量的赋值

如下所示，对变量进行赋值的办法有好几种。这里要提醒大家注意两件事：第一，*SET* 命令使用的赋值操作符是“=”，*SELECT* 命令使用的是“:=”；第二，下面最后一种变体把多个数据列赋值给了多个变量，这仅适用于 MySQL 5.0 及更高的版本，而且必须——通过适当的 *WHERE* 条件子句或 *LIMIT 1*——保证 *SELECT* 命令将精确地返回且仅返回一条记录。

```
SET @varname = 3
SELECT @varname := 3
SELECT @varname := COUNT(*) FROM table
SELECT COUNT(*) FROM table INTO @varname
SELECT title, subtitle FROM titles WHERE titleID=... INTO @t, @st
```

2. 变量的使用

一旦变量被定义出来，就可以用在任何 SQL 命令里。请注意，MySQL 4.1 及以前的版本区分变量名中的大小写字母，MySQL 5.0 及以后的版本则不区分。

```
SELECT @varname
@varname
123
SELECT * FROM titles WHERE titleID=@varname
```

还可以利用变量进行一些简单的计算。在下面这个例子里，将为 *bigvote* 数据库里的每一种候选 (*choice* 字段) 计算出它的得票率 (*bigvote* 数据库的结构和我们在第 3 章介绍的 *test_vote* 数据库一样，只是收录了更多的记录而已)：

```
USE bigvote
SELECT @total := COUNT(*) FROM votelanguage
@total := COUNT(*)
4807
```

```

SELECT choice, COUNT(*) / @total * 100 AS percentage
FROM votelanguage GROUP BY choice ORDER BY percentage DESC

choice percentage
4    49.19908466819
3    12.87705429581
1    11.75369253172
2    11.37923861036
5    7.53068441855
6    7.26024547534

```

在实际工作中，用 SQL 语言来完成这种计算的情况很少见：对于比较复杂的查询或插入操作，人们更习惯使用 PHP 或 Perl 语言来编程，而那样也就用不着使用 MySQL 变量了（参见第 3 章里的示例）。

10.4.2 IF 查询

本章前面的内容里已经对 *IF()* 函数做了一些介绍，它可以让查询命令根据一个条件表达式的求值结果返回不同的查询结果：

```
IF(condition, result1, result2)
```

MySQL 允许 *IF()* 函数嵌套使用：

```
IF(condition1, result1, IF(condition2, result2a, result2b))
```

如果打算测试某个数据列是否包含着 *NULL* 值，就必须使用 *ISNULL(expr)* 函数。在许多场合，可以用一个 *IFNULL()* 来代替 *IF() + ISNULL()* 的组合。下面两个表达式是等价的，它们都会在 *expr1* 的求值结果是 *NULL* 时返回 *expr2*；否则返回 *expr1*。

```
IFNULL(expr1, expr2) corresponds to IF(ISNULL(expr1), expr2, expr1)
```

10.4.3 CASE 分支

CASE 结构有两种语法变体。第 1 种变体需要判断一个 *IF* 条件：若 *expr=val1*，则返回 *result1*；若 *expr=val2*，则返回 *result2*；等等：

```

CASE expr
  WHEN val1 THEN result1
  WHEN val2 THEN result2
  ...
  ELSE resultn
END

```

第 2 种变体就不是一个条件了，它可以有任意多个条件：若条件 *cond1* 成立，则返回 *result1*；若条件 *cond2* 成立，则返回 *result2*；等：

```

CASE
  WHEN cond1 THEN result1
  WHEN cond2 THEN result2
  ...
  ELSE resultn
END

```

在下面的例子里，将为 *mylibrary* 数据库里收录的英语图书 (*langID=1*) 按字母表顺序生成一份清单，但在生成清单的时候不让 *a*、*an* 和 *the* 参加排序，于是一本名为 *A Programmer's Introduction* 的图书将被排在字母 P 下而不是被排在字母 A 下。要想获得这种效果，就不能直接对书名 (*title* 字段) 进行排序，这里使用了一个 *CASE* 结构来去掉这几个冠词，也就是把书名字符串截短了一点儿：

```

SELECT title FROM titles
WHERE langID=1
ORDER BY
CASE
    WHEN LEFT(title,2)="A "    THEN MID(title,3)
    WHEN LEFT(title,3)="An "   THEN MID(title,4)
    WHEN LEFT(title,4)="The "  THEN MID(title,5)
    ELSE title
END
title
_____
Darwin's Radio
The Definitive Guide to MySQL
Disgrace
A Guide to the SQL Standard
...
Practical UNIX & Internet Security
A Programmer's Introduction to PHP 4.0

```

参照这个示例，可以对德语图书（冠词是 *der*、*die*、*das*）或法语图书（冠词是 *le*、*la*、*les*）或俄语图书（开个小玩笑，俄语里没冠词）做出同样的处理。还可以利用这个技巧把网址或是其他字符串里的 www 或其他子串过滤掉。

注解 像这样的搜索条件会让SQL查询变得非常慢：MySQL将不得不先创建一份所有图书的清单再进行排序，*title*数据列上的现有索引全都帮不上忙。

10.5 在数据表间复制数据

10.5.1 利用复制操作创建新数据表

我们可以用 *CREATE TABLE newtable SELECT ... FROM oldtable* 命令去创建一个名为 *newtable* 的新数据表，并把 *SELECT* 命令从 *oldtable* 数据表里选取的记录全部插入到新数据表里去，MySQL 将根据 *SELECT* 命令所选取的数据在 *newtable* 数据表里自动创建出所有必要的数据列：

```

CREATE DATABASE newlibrary
USE newlibrary
CREATE TABLE publishers SELECT * FROM mylibrary.publishers

```

如果现在用 *SHOW CREATE TABLE oldtable* 和 *SHOW CREATE TABLE oldtable* 命令去对照一下这两个数据表的定义，就会发现：索引、*publID* 数据列的 *AUTO_INCREMENT* 属性以及 *ts* 数据列的一些 *TIMESTAMP* 属性都不见了，数据表的类型也从 InnoDB 变成了 MyISAM：

```

SHOW CREATE TABLE mylibrary.publishers      -- (original)
CREATE TABLE publishers(
    publID  INT(11) NOT NULL AUTO_INCREMENT,
    publName VARCHAR(60) COLLATE latin1_german1_ci NOT NULL default '',
    ts      TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
          ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (publID),
    KEY publName (publName)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_german1_ci

SHOW CREATE TABLE newlibrary.publishers      -- (new)
CREATE TABLE publishers (
    publID  INT(11) NOT NULL default '0',
    publName VARCHAR(60) CHARACTER SET latin1
          COLLATE latin1_german1_ci NOT NULL default '',
    ts      TIMESTAMP NOT NULL DEFAULT '0000-00-00 00:00:00'
) ENGINE=MyISAM DEFAULT CHARSET=latin1

```

数据表结构方面的这些变化都是用户不需要也不想要的。为了避免发生这种问题，更好的办法是先单独使用一条 *CREATE TABLE* 命令去创建一个与老数据表在结构上相同的新数据表，再用 *INSERT INTO ... SELECT ...* 命令把它填满数据。（*CREATE TABLE ... LIKE* 命令始见于 MySQL 4.1。）

```
USE newlibrary
CREATE TABLE publishers LIKE mylibrary.publishers
INSERT INTO publishers SELECT * FROM mylibrary.publishers
```

提示 用 MySQL 的数据导入/导出命令也可以完成数据表的复制/移动操作。将在第 14 章讨论备份问题的时候再介绍这些命令。

10.5.2 把数据复制到现有数据表

为了把数据复制到一个现有的数据表里，可以使用 *INSERT INTO ... SELECT ...* 命令。必须确保数据列的个数、数据类型、先后顺序都准确无误。如果源数据表里的数据列个数比目标数据表里的少，可以在 *SELECT* 部分相应地给出几个 *NULL* 或固定值，比如像... *SELECT NULL, publID, publName, ts, 0, "abc"* 这样：

```
INSERT INTO newlibrary.publishers
SELECT publID, publName, ts FROM mylibrary.publishers
```

这里最容易出问题的地方还是 *AUTO_INCREMENT* 字段：如果源数据表和目标数据表有 ID 编号相同的记录，*INSERT* 命令将报告出错（*Duplicate entry xy for key ...*）。解决这个问题的办法有以下几种：

- 可以使用 *INSERT IGNORE INTO ...* 命令，它不会报告出错，受影响的记录不参加复制。
- 可以用 *REPLACE ...* 命令代替 *INSERT INTO ...* 命令，受影响的记录将在复制过程中被覆盖。
- 可以对 ID 数据列（如本例中的 *publID*）做出一些调整，不让它们发生冲突。这个办法需要假设 *publID* 数据列没被其他数据表引用，或者虽然被引用，但外键会随之变化。

```
SELECT @maxid := MAX(publID) FROM newlibrary.publishers;
INSERT INTO newlibrary.publishers
SELECT publID + @maxid, publName, ts FROM mylibrary.publishers
```

10.6 统计报表

统计报表从一个数据源收集信息并对它们进行分组统计。了解这些表格的最简单的办法是通过一个例子：在 *mylibrary* 数据库里，每本书都有出版公司、图书门类和语言 3 项信息。这里假设希望知道图书门类和语言的每一种组合有多少本书，统计报表提供了一种对数据表里的数据进行统计运算的简单办法。

许多数据库系统提供特殊的 OLAP 命令来创建统计表格。OLAP 代表着“online analytical processing”（实时分析处理）并包括了特殊的方法来管理和处理多维数据，“多维”在这里的意思是“多种不同的特征”，数据就是按照这些特征而被分组的。具备 OLAP 能力的数据库系统经常被人们称为数据仓库（data warehouse）。

可惜的是 MySQL 不支持 OLAP 函数，除了 *GROUP BY*（参见第 9 章）。因此，通常需要使用外部程序来创建统计表格，Microsoft Excel 尤其适合用来完成这项工作（参见第 7 章）。

对于比较简单的统计问题，可以利用 MySQL 中的 SQL 函数来解决。非常实用的是 *SUM()* 和 *IF()* 函数的组合：用 *SUM()* 求出一组数据列的总和，用 *IF()* 为求和操作建立必要的筛选条件。下面两个例子演示了如何使用它们。

10.6.1 涉及 *titles*、*languages* 和 *categories* 数据表的统计报表

在第一个例子里，使用 *mylibrary* 数据库。目标是返回一个统计表格，它可以告诉我们图书门类和语言的每一种组合有多少本书（当然，只统计有这些信息的图书）。初始数据如下所示：

```
USE mylibrary
SELECT title, langName, catName
FROM titles, categories, languages
WHERE titles.catID = categories.catID
AND titles.langID = languages.langID
```

title	langName	catName
A Guide to the SQL Standard	english	SQL
Practical UNIX & Internet Security	english	Computer books
MySQL	english	MySQL
MySQL Cookbook	english	MySQL
PHP and MySQL Web Development	english	PHP
Visual Basic	deutsch	Programming
Excel 2000 programmieren	deutsch	Programming
LaTeX	deutsch	Computer books
Nennen wir ihn Anna	deutsch	Children's books
Mathematica	deutsch	Computer books
Jag saknar dig, jag saknar dig	svensk	Children's books
Hunderna i Riga	svensk	Literature and fiction
Ute av verden	norsk	Literature and fiction
...		

现在将按图书门类来对这个清单进行分组统计。对于每个门类，使用 *SUM(IF(titles.langID = n, 1, 0))* 来统计用一种特定语言写的图书有多少本。*COUNT(*)* 统计各个门类里的图书总数（不分语言）。*WITH ROLLUP* 有这样的效果：*GROUP BY* 会为查询结果中的每一个输出列分别计算一个总计数字（见查询结果的最后一行）：

```
SELECT catName,
       SUM(IF(titles.langID=1, 1, 0)) AS english,
       SUM(IF(titles.langID=2, 1, 0)) AS deutsch,
       SUM(IF(titles.langID=3, 1, 0)) AS svensk,
       SUM(IF(titles.langID=4, 1, 0)) AS norsk,
       COUNT(*)
  FROM titles, categories, languages
 WHERE titles.catID = categories.catID
   AND titles.langID = languages.langID
 GROUP BY catName WITH ROLLUP
```

catName	english	deutsch	svensk	norsk	COUNT(*)
C#	0	1	0	0	1
Children's books	0	1	2	0	3
Computer books	1	5	0	0	6
Databases	0	1	0	0	1
LaTeX, TeX	0	2	0	0	2
Linux	0	3	0	0	3
Literature and fiction	2	7	9	1	19
MySQL	3	1	0	0	4
PHP	2	3	0	0	5
Programming	0	4	0	0	4
Relational Databases	0	1	0	0	1
Science Fiction	3	0	0	0	3
SQL	1	0	0	0	1
Visual Basic	0	1	0	0	1
Visual Basic .NET	0	2	0	0	2
NULL	12	34	11	1	58

这条查询命令有一个很明显的弊病： $SUM(...)$ 的个数固定为 4 个。万一 *languages* 数据列里有不止 4 种语言——比如 20 种或 30 种语言——又该怎么办呢？为了解决这个问题，需要在客户程序里（比如一个 PHP 脚本）为这个查询编写一段 SQL 代码：先获得一份所有语言的清单，再为每种语言构造一个 $SUM(...)$ 函数，最后把它们合并为一个完整的查询命令。

可以用下面这条命令来查出用每一种语言写的图书有多少本：

```
SELECT langName, COUNT(*)
FROM titles, categories, languages
WHERE titles.catID = categories.catID
    AND titles.langID = languages.langID
GROUP BY langName
```

<i>langName</i>	COUNT(*)
deutsch	15
english	5
norsk	1
svensk	5

10.6.2 月度查询统计报表

第二个例子将使用 *votelanguage* 数据表，它的内容是一个简单的网络问卷的投票结果（详见第 3 章中的示例）。我们的目标是把每个月的投票结果统计为百分比数字显示出来，这一方面是为了了解人们的投票情况，另一方面是为了观察有没有人试图操纵投票结果。初始数据大约有 3300 条记录，下面是其中的前 5 条：

```
USE bigvote
SELECT * FROM votelanguage LIMIT 5
```

<i>id</i>	<i>choice</i>	<i>ts</i>
1	4	20010516083407
2	3	20010516083407
3	5	20010516083407
4	4	20010516083407
5	2	20010516083407

每月的投票人数很容易统计出来：

```
SELECT DATE_FORMAT(ts, '%Y-%m') AS mnth, COUNT(*)
FROM votelanguage
GROUP BY mnth
```

<i>mnth</i>	COUNT(*)
2001-05	214
2001-06	111
...	...

每种程序设计语言的得票数可以用一些 $SUM(IF(...))$ 函数查出来：

```
SELECT DATE_FORMAT(ts, '%Y-%m') AS mnth,
    SUM(IF(choice=1, 1, 0)) AS c,
    SUM(IF(choice=2, 1, 0)) AS java,
    SUM(IF(choice=3, 1, 0)) AS perl,
    SUM(IF(choice=4, 1, 0)) AS php,
    SUM(IF(choice=5, 1, 0)) AS vb,
    SUM(IF(choice=6, 1, 0)) AS other,
    COUNT(*)
FROM votelanguage
GROUP BY mnth
```

<i>mnth</i>	<i>c</i>	<i>java</i>	<i>perl</i>	<i>php</i>	<i>vb</i>	<i>other</i>	COUNT(*)
2001-05	20	24	30	110	16	14	214
2001-06	6	13	15	64	7	6	111
...							

现在，如果想知道投票情况的百分比，只须把 *SUM* 表达式除以 *COUNT(*)* 即可。为了让除法结果有可读性，我们给它们乘上 100 并利用 *ROUND()* 函数保留到小数点后面一位数字：

```
SELECT DATE_FORMAT(ts, '%Y-%m') AS mnth,
       ROUND(SUM(IF(choice=1, 1, 0)) * 100 / COUNT(*), 1) AS c,
       ROUND(SUM(IF(choice=2, 1, 0)) * 100 / COUNT(*), 1) AS java,
       ROUND(SUM(IF(choice=3, 1, 0)) * 100 / COUNT(*), 1) AS perl,
       ROUND(SUM(IF(choice=4, 1, 0)) * 100 / COUNT(*), 1) AS php,
       ROUND(SUM(IF(choice=5, 1, 0)) * 100 / COUNT(*), 1) AS vb,
       ROUND(SUM(IF(choice=6, 1, 0)) * 100 / COUNT(*), 1) AS other
  FROM votelanguage
 GROUP BY mnth
```

<i>mnth</i>	<i>c</i>	<i>java</i>	<i>perl</i>	<i>php</i>	<i>vb</i>	<i>other</i>
2001-05	9.3	11.2	14.0	51.4	7.5	6.5
2001-06	5.4	11.7	13.5	57.7	6.3	5.4
...						

10.7 子查询

从 4.1 版本开始，MySQL 开始支持所谓的子查询（*subSELECT*），也就是一组嵌套的 *SELECT* 查询。（有很多其他品牌的数据库系统从它们最早的版本开始就支持子查询。）

提示 如果使用的是 MySQL 的一个老版本，通常可以把一个子查询改写为一个 *JOIN* 查询，但这种改写往往很困难。在一些比较复杂的场合，可能需要提前创建一些临时数据表来保存中间结果。

10.7.1 语法变体

构造子查询命令的办法有许多种。下面是一些最重要的语法变体，具体例子见 10.7.2 节。

□ **SELECT ... WHERE col = [ANY | ALL] (SELECT ...)**。在这个语法变体里，第二条 *SELECT* 命令必须返回一个离散值（一行一列）。这个值将被用在 “*col = ...*” 的比较操作里（这里还可以使用其他的比较操作符，如 “*col > ...*”、“*col <= ...*”、“*col <> ...*” 等）。这个比较操作可以用关键字 *ANY*、*SOME*（它是 *ANY* 的一个同义词）或 *ALL* 来修饰，而此时第二条 *SELECT* 命令将允许返回一个以上的值。*ANY/SOME* 的含义是：所有适当的值都必须被考虑，整个查询可以返回多个结果。语法形式 *col = ANY* 的含义与 *col IN ...* 的一样（参见下一种语法变体）。

ALL 的效果不怎么好理解：表达式 *operator ALL*（这里的 *operator* 可以是任何一种比较操作符）只在以下两种情况下会返回 *TRUE*：第二条 *SELECT* 命令的所有结果都能让这个表达式返回 *TRUE*；第二条 *SELECT* 命令没有返回任何结果；其他情况都将返回 *FALSE*。

□ **SELECT ... WHERE col [NOT] IN (SELECT ...)**。在这个语法变体里，第二条 *SELECT* 命令可以返回一个离散值列表（即 *(n1, n2, ...)*，其中 *n1, n2* 都是离散值）。这个列表将与第二条 *SELECT* 命令前面的内容构成 *SELECT ... WHERE col [NOT] IN (n1, n2, ...)* 形式的查询——这个查询的结果就是整个查询的结果。

□ **SELECT ROW(value1, value2, ...) = [ANY] (SELECT col1, col2, ...)**。这个查询将测试是否存在一条能满足给定条件的记录，测试结果是 1（真）或 *NULL*（假）。这个语法变体与前两个

的主要区别是：出现在比较操作符（比如“=”）两端的不是一个离散值而是一组离散值。

MySQL 将对记录 *ROW(value1, value2, ...)* 和第二条 *SELECT* 命令的结果进行比较，而第二条 *SELECT* 命令必须精确地返回一条记录。如果第二条 *SELECT* 命令的返回结果与记录 *ROW(value1, value2, ...)* 相匹配，整个查询就将返回 1；否则将返回 *NULL*。

如果使用了可选关键字 *ANY*（或它的同义词 *SOME*），第二条 *SELECT* 命令将允许返回一条以上的结果记录。只要这些结果记录里有一个与 *ROW* 记录相匹配，整个查询就将返回 1。

- ***SELECT ... WHERE col = [NOT] EXISTS (SELECT ...)***。在这个语法变体里，第一条 *SELECT* 命令查到了多少条记录，第二条 *SELECT* 命令就要执行多少次。只有当第二条 *SELECT* 命令有结果（至少找到一条记录）时，第一条 *SELECT* 命令找到那条记录才能进入最终的查询结果。在这个变体里，关键字 *EXIST* 的前面还可以加上取非操作符 *NOT* 做“不存在”比较。这种 *EXIST* 语法结构一般只在那两条 *SELECT* 命令所涉及的记录可以用 *WHERE* 条件相关联（就像 *JOIN* 操作一样）时才有实际意义。
- ***SELECT ... FROM (SELECT ...) AS name WHERE ...***。这个语法变体很少有人使用。MySQL 将先执行括号中的第二条 *SELECT* 命令，它将返回一个数据表作为外层 *SELECT* 命令的查询对象。换句话说，外层 *SELECT* 命令访问的是一个临时创建出来的数据表，这个数据表是另一个 *SELECT* 命令的查询结果；这种数据表被称为衍生数据表（derived table）。SQL 语法要求衍生数据表必须通过 *AS name* 子句获得一个名字。

子查询语法使得 *SELECT* 命令可以彼此嵌套，但这种命令既难以阅读又难以理解。*SELECT* 命令还可以用在 *UPDATE* 和 *DELETE* 命令的 *WHERE* 条件里，以确定需要对哪些记录进行修改或删除。

提示 可以在网址 <http://dev.mysql.com/tech-resources/articles/4.1/subqueries.html> 处找到一篇很有意思的关于子查询语法的文章。在 MySQL 文档里也可以找到许多关于子查询的讨论：<http://dev.mysql.com/doc/mysql/en/subqueries.html>。

1. 不足与缺陷

MySQL 目前还不支持在子查询命令里使用 *LIMIT* 子句；也就是说，不允许使用 *SELECT ... WHERE ... IN (SELECT ... LIMIT 10)* 这样的命令。如果执意这么做，将会看到这样一条出错消息：*ERROR 1234 (42000): This version of MySQL doesn't yet support 'LIMIT & IN/ALL ANY/SOME subquery* (MySQL 的这个版本还不支持“*LIMIT ...* 子查询”)。

如果使用子查询去修改数据记录 (*DELETE*、*UPDATE*)，打算修改的数据表和子查询所涉及的数据表必须不同。

根据笔者根据许多案例总结出来的经验，子查询的执行效率非常低下。这或许是因为子查询比普通的关联查询 (*SELECT ... JOIN*) 更难以让 MySQL 进行优化。在追求速度的应用项目里，请尽可能使用等效的 SQL 命令而不是使用子查询。

2. MySQL 的早期版本的子查询

如果使用的是 MySQL 的一个早期版本 (4.0 或更早)，通常可以把一个子查询命令改写为一个 *JOIN* 查询，但这种改写往往很困难。

下面两个等效的 *SELECT* 命令提供了一个例子。两条命令都是用来查找 *publID* 值不合法（也就是在 *publishers* 数据表里不存在）的图书的，但因为 *mylibrary* 数据库里的 *titles* 和 *publishers* 数据表之间有一条外键约束规则（一致性规则），所以这种事不可能发生。因此，这两条命令都不会返回任何结

果记录。下面是那个子查询：

```
SELECT * FROM titles
WHERE publID NOT IN
  (SELECT publID FROM publishers)
```

下面是一个 *JOIN* 查询：

```
SELECT titles.*
  FROM titles LEFT JOIN publishers
    ON titles.publID=publishers.publID
   WHERE ISNULL(publishers.publID) AND NOT(ISNULL(titles.publID))
```

如果想把一个比较复杂的子查询改写为一个 *JOIN* 查询，可能需要提前创建一些临时数据表来保存中间结果。

10.7.2 示例

下面这个查询将把出版公司名字以字母“O”开头的所有图书查出来。*IN* 关键字的右“操作数”（一个 *publID* 值的列表）是用另外一个 *SELECT* 查询（子查询）生成的：

```
USE mylibrary
SELECT title FROM titles WHERE publID IN
  (SELECT publID FROM publishers WHERE publName LIKE 'O%')
```

title
MySQL & MySQL
Practical UNIX & Internet security
MySQL Cookbook
Comédia Infantil
Hunderna i Riga

用“= *ANY*”语法去比较 *publID* 值的子查询也可以得到这个结果：

```
SELECT title FROM titles WHERE publID = ANY
  (SELECT publID FROM publishers WHERE publName LIKE 'O%')
```

最后，还可以用以下命令获得同样的结果，这条命令使用了比较难解的 *EXISTS* 语法变体。在这里，括号里的第二条 *SELECT* 命令将为每一本图书执行一遍。只有当这第二条命令返回一个非空的结果时，第一条 *SELECT* 所返回的一本图书才会被考虑。这两个 *SELECT* 查询之间的关系由条件表达式 *titles.publID = publishers.publID* 创建：

```
SELECT title FROM titles WHERE EXISTS
  (SELECT * FROM publishers WHERE
    titles.publID = publishers.publID AND
    publName LIKE 'O%')
```

以下命令将测试 *titles* 数据表是否收录有一本或多本符合 *title='Linux'*、*subtitle='Installation, Konfiguration, Anwendung'* 条件的图书。查询的结果是 1，也就是有这样的书。警告：如果不使用 *ANY*，这个查询将只在 *titles* 数据表里有且仅有这样一本图书的时候才返回 1。

```
SELECT ROW('Linux', 'Installation, Konfiguration, Anwendung') = ANY
  (SELECT title, subtitle FROM titles)
1
```

以下命令使用了一个衍生数据表。内层的 *SELECT* 命令将返回一个数据表，其内容是作者在一人以上的图书名称。外层的 *SELECT* 命令对这个结果做了进一步处理，并按照作者人数由多到少的顺序对它进行了排序：

```

SELECT * FROM
  (SELECT titles.titleID, title, COUNT(authID) AS authCount
   FROM titles, rel_title_author
   WHERE titles.titleID = rel_title_author.titleID
   GROUP BY rel_title_author.titleID
   HAVING authCount>1) AS titleAuthCount
ORDER BY authCount DESC



| titleID | title                             | authCount |
|---------|-----------------------------------|-----------|
| 23      | Maple                             | 3         |
| 9       | MySQL & mSQL                      | 3         |
| 3       | Client/Server Survival Guide      | 3         |
| 25      | Linux für Internet und Intranet   | 3         |
| 69      | Visual C#                         | 2         |
| 2       | The Definitive Guide to Excel VBA | 2         |
| 59      | PHP and MySQL Web Development     | 2         |
| 34      | MySQL                             | 2         |
| ...     | ...                               |           |


```

10.8 保证数据的一致性

在数据库项目的开发过程中（数据库设计、程序代码开发等），经常会出现因为测试数据被删改得乱七八糟而致使数据表之间的关联/引用关系一片混乱的情况。当然，在一个数据库系统实际运转起来之后，编程失误也会导致这种问题。无论发生在哪一阶段，只要数据表之间的关联/引用关系遭到破坏，就表明有关代码存在着错误。

不管什么原因，管理好数据的一致性并及时发现和纠正各种错误始终是数据库管理工作的重担之一。在下面的内容里，我们将向大家介绍一些查找和纠正这类错误的技巧。这一节里的所有示例都是在样板数据库 *mylibrary* 上进行的。

本节中的示例有不少是用子查询实现的，它们可以帮助大家对子查询都可以用来干些什么的问题有一个直观的认识。

10.8.1 找出没有作者的图书

mylibrary 数据库所收录的每一本书都应该至少有一位作者，*titles* 和 *authors* 数据表之间的关系体现在 *rel_title_author* 数据表里。

下面这个查询将为 *titles* 数据表里的每一本书测试 *rel_title_author* 数据表里是否存在一个有着同样的 *titleID* 编号的记录项。只有那些不存在对应记录的图书会出现在查询结果里（这是因为这里使用的是 *NOT EXISTS* 查询）：

```

SELECT title FROM titles WHERE NOT EXISTS
  (SELECT * FROM rel_title_author
   WHERE titles.titleID = rel_title_author.titleID)

```

10.8.2 找出无效的出版公司引用：1:n 关系中的无效记录

在理想的情况下，*titles* 数据表里的每一条记录都通过自己的 *publID* 字段关联着一家出版公司，如果某本图书的出版公司尚未被收录到数据库里，相应的 *publID* 字段将包含着 *NULL*。但万一某个已收录的出版公司被意外删除了会发生什么事情？这可能导致某些 *titles* 记录的 *publID* 字段关联的是一家已经不复存在的出版公司。（类似的问题还可能发生在其他数据表之间的关系上，比如 *titles* 数据表和 *categories* 数据表（涉及 *catID* 字段）之间、*titles* 数据表和 *languages* 数据表（涉及 *langID* 字段）之间。因此，下面的查询不是去寻找交叉引用关系有错误的 *titles* 记录，而是去寻找 *publID* 字段值无

效的 *titles* 记录。我们将通过一个示例来演示如何寻找这类无效的记录。)

注解 *mylibrary* 数据库肯定不包含无效的记录，因为数据表之间的所有关系都有外键规则来保证。

因此，为了制造出一个无效的记录，需要特意在 *titles* 数据表里插入一条无效的记录。(以下命令需要假设 *publID=9999* 的出版公司不存在。)为了让这条“非法”的 *INSERT* 命令能够执行，必须先暂时关闭 MySQL 的数据一致性规则检查机制 (*SET ...*):

```
SET foreign_key_checks=0
INSERT INTO titles (title, publID) VALUES ('deleteme', 9999)
SET foreign_key_checks=1
```

下面这个查询将找出 *publID* 值在 *publishers* 数据表里不存在的 *titles* 记录:

```
SELECT title, titles.publID FROM titles
WHERE publID NOT IN (SELECT publID FROM publishers)
```

找出无效的记录之后，有两个办法可以纠正这种错误：一是简单地删除那些无效的 *titles* 记录（本例将采用这一解决方案）；二是把无效的 *titles* 记录与某个现有的 *publishers* 记录关联起来（如果那家出版公司尚未被收录到 *publishers* 数据表里，还需要先把它创建出来才行）：

```
DELETE FROM titles WHERE publID NOT IN
(SELECT publID FROM publishers)
```

10.8.3 找出作者与图书之间的无效链接 (*n:m* 关系)

涉及 *n:m* 关系（比如图书/作者关系）的搜索就更加复杂了。这里有 3 种错误情况：

- 图书通过 *rel_title_author* 数据表引用着一位并不存在的作者。
- 作者通过 *rel_title_author* 数据表引用着一本并不存在的图书。
- *rel_title_author* 数据表里的某条记录的 *titleID* 和 *authID* 字段都引用着无效的记录。（只根据现有的 *authors* 或 *titles* 记录去查找非法记录是无法发现这第 3 种错误情况的。）

为了模拟这些错误，需要特意在 *rel_title_author* 数据表里插入 3 条无效的记录，它们分别对应着上述 3 种错误情况。以下命令需要假设 *authID=1* 的作者和 *titleID=1* 的图书都存在、但 ID 编号是 9999 的作者和图书都不存在：

```
SET foreign_key_checks=0
INSERT INTO rel_title_author (titleID, authID)
VALUES (1,9999), (9999,1), (9999, 9999)
SET foreign_key_checks=1
```

现在，只要一个简单的子查询就可以把这些无效记录都查找出来：

```
SELECT titleID, authID FROM rel_title_author
WHERE authID NOT IN (SELECT authID FROM authors) OR
      titleID NOT IN (SELECT titleID FROM titles)
```

authID	titleID
9999	1
1	9999
9999	9999

确认存在着无效记录后，用以下命令就可以把它们彻底删除：

```
DELETE FROM rel_title_author
WHERE authID NOT IN (SELECT authID FROM authors) OR
      titleID NOT IN (SELECT titleID FROM titles)
```

10.9 找出冗余的数据记录

在数据库管理工作中，经常会出现数据记录发生冗余重复的问题。不过，就像下面这个涉及 *authors* 数据表和 *mylibrary* 数据库的例子所演示的那样，这种冗余记录很容易查找。首先，需要按作者姓名对 *authors* 数据表进行一次分组查询，以便把重复出现一次以上的作者记录找出来：

```
USE mylibrary
SELECT authName, COUNT(*) AS cnt
FROM authors
GROUP BY authName
HAVING cnt>1
```

接下来的任务是把这些记录的 *authID* 编号全部查找出，并根据这些编号删除冗余的记录项。注意，应该先在 *rel_title_author* 数据表里把第二个 *authID* 编号替换为第一个，然后再从 *authors* 数据表里把第二条作者记录删掉。

比如说，假设因为输入错误而使得作者 *Gray Cornell* 出现了两次：

<i>authID</i>	<i>authName</i>
123	Cornell Gary
758	Cornell Gary

titles 数据表收录了 Gray 写的两本书：第一本是 *The Sex Life of Unix*，它对应着 *authID=123*；第二本书是 *More Sex Life of Unix*，它对应着 *authID=758*。这表明 Gray 有两个 *authID* 编号！

为了消除这个错误，我们决定保留 *authID=123*，所以要把 *rel_title_author* 数据表里 *authID=758* 的记录全部改为 *authID=123*：

```
UPDATE rel_title_authors SET authID=123 WHERE authID=758
```

现在，Gray 写的书都有 *authID=123*。最后，把 *authors* 数据表里的冗余记录删除干净：

```
DELETE FROM authors WHERE authID=758
```

在 *titles* 数据表里，允许好几本书有着同样书名的情况（不同的作者给书起的名字一样）。因此，必须同时检查多个数据列才能断定同名的图书是不是冗余的：

```
SELECT title, subtitle, edition, COUNT(*) AS cnt
FROM titles
GROUP BY title, subtitle, edition
HAVING cnt>1
```

因为打字错误而导致的冗余问题就更难以查找和纠正了。如果是英语单词，可以借助 *SOUNDEX()* 函数来寻找打字错误。这个函数能够根据给定文字的读音返回一个字符串，但这个结果只能依靠人工手段去检查。*SOUNDEX()* 函数不能保证把所有读音类似的记录都找出来，它认为重复的名字也不一定就是冗余的记录。在下面的查询里，使用了 *MIN(authName)* 函数来返回那些发音相似的名字当中按字母表排序的第一个：

```
SELECT SOUNDEX(authName) AS snd,
       COUNT(*) AS cnt,
       MIN(authName) AS firstname
  FROM authors
 GROUP BY snd
 HAVING cnt>1
```

10.10 数据表设计方案的改进

在第 8 章详细介绍了前三个范式，有许多人认为这是一个优秀数据库设计方案的出发点。如果把

数据库的设计者们划在一个圈子里，那么在圈外人看来，所有的问题都应该在数据库的设计阶段得到解决，他们看到和使用的数据库不应该有任何设计方面的缺陷；但作为圈内人，我们知道有许多数据库设计方案往往是在“太迟了”的时候才被发现存在着违反范式有关规则的问题。

可是，为什么要拖到“太迟了”才发现问题？为什么不利用下面将要介绍的技巧把问题在数据库的设计阶段就彻底解决呢？在下面的例子中，我们将向大家演示如何发现数据库设计细节方面的问题以及如何把冗余的信息转移到一个新的数据表里，经过改进的数据库设计方案将会是一个符合范式有关规则（尤其是第三范式的规则）的好方案。

这个例子从一个 *messages* 数据表开始：这个数据表的 *author* 数据列包含着某网上论坛所有网帖作者的名字。因为有许多作者在论坛上发表过多份帖子，所以他们的名字在 *author* 数据列里重复出现了许多次。这种冗余是违反范式有关规则的，而现在的任务就是解决这个问题。具体到这个例子，我们决定创建一个新数据表 *users* 来存放全体作者的名单，至于 *messages* 数据表里的 *author* 数据列，我们将把它替换为一个 *userID* 字段。

第一步是创建新数据表 *users*：

```
CREATE TABLE users (userID INT NOT NULL AUTO_INCREMENT,
                    username VARCHAR(60) NOT NULL,
                    PRIMARY KEY (userID))
```

接下来，利用本章前面介绍的 *INSERT INTO ... SELECT* 命令把所有的作者姓名插入新创建出来的 *users* 数据表里。这里的要点之一是必须给这条 *INSERT* 命令中的 *SELECT* 子命令加上一个关键字 *DISTINCT*，这将确保每个名字只会被插入一次。另一个要点是把 *NULL* 值传递给 *userID* 数据列，这将确保 MySQL 会自动地往这个数据列里放入一个 *AUTO_INCREMENT* 编号：

```
INSERT INTO users SELECT DISTINCT NULL, author FROM messages
```

现在，在原来的 *messages* 数据表里增加一个 *userID* 数据列：

```
ALTER TABLE messages ADD userID INT
```

接下来，要把一些编号值填充到 *userID* 数据列里，这些编号必须与 *users* 数据表里的 *userID* 字段值保持正确的对应关系。注意，下面的 *UPDATE* 命令仅适用于 MySQL 4.0 及更高的版本。虽然这条 *UPDATE* 命令用到了 *messages* 和 *users* 两个数据表，但只有 *messages* 数据表里的数据被修改了（*SET* 子命令只设置了 *messages.userID* 字段）：

```
UPDATE messages, users
SET messages.userID = users.userID
WHERE messages.author = users.username
```

在 MySQL 4.1 及更高的版本里，还可以使用如下所示的子查询来代替上面那条 *UPDATE* 命令。这两条命令的执行效果是一样的，只是下面的命令更容易理解一些。请注意，内层 *SELECT* 命令与外层 *SELECT* 是通过条件表达式 *users.username = messages.author* 相关联的：

```
UPDATE messages SET userID=
(SELECT userID FROM users WHERE users.username = messages.author)
```

简单查询表明一切都是按计划进行的：

```
SELECT messages.author, users.username, users.userID
FROM messages, users
WHERE messages.userID = users.userID LIMIT 5
```

<i>author</i>	<i>userName</i>	<i>userID</i>
boehnke	boehnke	9
Michael	Michael	10
cjander	cjander	11
Frauke	Frauke	12
Bernd	Bernd	13

现在，可以把 *messages.author* 数据列删掉了；保存在这个数据列里的信息以后可以通过 *messages.userID* 数据列和新的 *users* 数据表查出来：

```
ALTER TABLE messages DROP author
```

10.11 对前 *n* 条或后 *n* 条记录进行处理

本节示例都是在示例数据库 *myforum* 上实现的。

10.11.1 数据查询 (SELECT)

基于各种数据库的“十大排行榜”是一种相当常见的数据库应用，这里的“十大”还可以是“*n*大”。但不管 *n* 是多少，问题的核心无非是怎样从一个数据集合里把最早、最好、最坏、最老或者是其他最什么的前 *n* 条记录找出来而已。这类问题其实很容易解决：这正是 *SELECT ... ORDER BY ... LIMIT* 命令可以大显身手的地方。

需要在 *ORDER BY* 子句里给出一个排序条件，比如像 *ORDER BY date* 这样。如果想查的不是前 *n* 条记录而是后 *n* 条记录，再加上一个 *DESC* 关键字把排序方式倒过来就行了，比如像 *ORDER BY date DESC* 这样。关键字 *LIMIT* 的作用是把查询命令所返回的结果记录限制为一个特定的数量。

下面的查询将返回 *myforum* 讨论组里的最后 5 条记录。为了让查询结果保持整齐易读，这里还使用了 *LEFT* 函数把 *subject* 字段值（帖子的标题）的字符串长度减少到了 20 个字符：

```
USE myforum
SELECT LEFT(subject, 20) AS subj, forumID, ts
FROM messages ORDER BY ts DESC LIMIT 5
```

<i>subj</i>	<i>forumID</i>	<i>timest</i>
Re: Run-time error	1001	20050124154825
Re: Re: Suche Erfahr	1	20050124132214
Re: Full text search	1006	20050124092312
Re: MySQL on ASP wit	1006	20050124052830
Full text searches	1006	20050123211616

第二个例子更有针对性，它可不是什么帖子都搜，它只搜索各个讨论线程中的第一份帖子（也就是 *rootID=msgID* 的帖子），而且只在 1006 号论坛（用英语讨论 MySQL 问题的“MySQL, English”讨论组）里进行搜索：

```
SELECT LEFT(subject, 20) AS subj, forumID, ts FROM messages
WHERE rootID=msgID AND forumID=1006
ORDER BY ts DESC LIMIT 5
```

<i>subj</i>	<i>forumID</i>	<i>timest</i>
Full text searches	1006	20050123211616
MySQL and ASP	1006	20050121223216
where do I save this	1006	20050116013629
samples do not work	1006	20050114151020
ASP and MySQL	1006	20050730224220

下一个例子是找出最长的（即跟帖数量最多的）5 个讨论线程。这里使用了 *COUNT* 函数和 *GROUP BY*

子句来分组统计 *rootID* 编号相同的帖子数量。

```
SELECT COUNT(*) AS answerCount, rootID FROM messages
GROUP BY rootID ORDER BY answerCount DESC LIMIT 5
```

answerCount	rootID
11	767
9	1392
8	1134
7	495
7	748

很可惜，这个查询无法把 *subject* 字段值（帖子的标题）显示出来，这是因为 *GROUP BY* 只负责对讨论线程进行分组，*COUNT* 只负责统计符合筛选条件的帖子数量，整个查询命令里没有负责提取 *subject* 字段值（哪怕是它的前几个字符）的字符串函数。（如果 *subject* 字段存放的是数值，倒是可以用 *MIN* 或 *MAX* 函数把它们提取出来。）子查询帮不上这个忙，因为子查询里不允许使用 *LIMIT* 表达式。如果真想知道这些帖子的标题，就需要创建一个临时数据表或是以手动方式再进行一次如下所示的查询：

```
SELECT subject FROM messages WHERE msgID IN (767, 1392 ...)
```

10.11.2 修改现有记录 (*UPDATE* 和 *DELETE*)

既然把 *ORDER BY* 和 *LIMIT* 用在 *SELECT* 命令里可以查找出前 *n* 条或后 *n* 条记录，把它们用在 *UPDATE* 和 *DELETE* 命令里就可以对那些记录进行修改或删除了。

第一个例子很简单：为了节省空间，决定用 *UPDATE* 命令把最早的 100 份帖子的内容替换为两种人类语言的“过期删除”提示：*messages no longer available / Nachr icht steht nicht mehr zur Verfügung*。这里的要点是用表达式 *ts=ts* 确保 *ts* 字段值不会发生变化（否则最早的 100 份帖子会突然变成最新的）。

```
UPDATE messages
SET msgText="message no ... / Nachricht ...", ts=ts
ORDER BY ts LIMIT 100
```

把最早的 100 份帖子删掉就更简单了：

```
DELETE messages ORDER BY ts LIMIT 100
```

这两条命令都很容易理解，但执行它们之后会破坏一些讨论线程。不仅如此，在修改或删除的 100 条帖子记录里，说不定还有某个讨论线程的第一份帖子，而那个讨论线程到现在还有人在跟帖。这类后遗症当然不是我们想要的，怎么解决请看下节。

10.11.3 把全部讨论线程删除到只剩下最新的 500 个线程

以下命令将把 *myforum* 论坛上的全部讨论线程删除到只剩下最新的 500 个。整个过程相当复杂，我们将依次对有关命令进行解释。第一条命令将返回一个时间，这个时间是第 500 个最新讨论线程的起始时间。把这个时间保存在变量@*oldtime* 里：

```
SELECT @oldtime := ts FROM messages
WHERE rootID=msgID ORDER BY ts DESC LIMIT 499,1
```

```
@oldtime := ts
20050911093909
```

下一条命令负责查出 *myforum* 论坛上比第 500 个最新讨论线程还要“古老”的讨论线程有多少个。这项信息并不是后续操作所必需的，但希望大家知道应该如何进行这样的查询：

```
SELECT COUNT(*) FROM messages WHERE ts<@oldtime AND rootID=msgID
COUNT(*)
117
```

现在，需要把将要删除的各个讨论线程里的第一份帖子的 *msgID* 编号全部查出来。我们用了一条 *CREATE ... SELECT* 命令把它们写入了一个临时数据表 *rootIDs*，并且把这个数据表留在内存里以提高后续查询命令的执行效率 (*ENGINE=HEAP*)：

```
CREATE TEMPORARY TABLE rootIDs ENGINE=HEAP
SELECT rootID FROM messages WHERE ts<@oldtime AND rootID=msgID
```

接下来，利用临时数据表 *rootIDs* 把那些不再需要保留的帖子全部删除掉：

```
DELETE messages FROM messages, rootIDs
WHERE messages.rootID=rootIDs.rootID
```

最后，删除临时数据表 *rootIDs*：

```
DROP TABLE rootIDs
```

10.12 以随机方式选择数据记录

有时候，需要以随机方式选取数据记录，比如在网页上随机显示一幅图片或广告等。

10.12.1 通用方法：*RAND()*函数

以随机方式选取数据记录最简单的办法是用 *SELECT* 选出全部的有关记录，然后用 *ORDER BY RAND()* 子句对这些记录进行随机排序，最后用 *LIMIT n* 得到 *n* 条最终结果记录。

下面的例子是在 *titlescopy* 数据表上进行的，它是 *mylibrary* 数据库里的 *titles* 数据表的一份复制。我们用同一条命令对 *titlescopy* 数据表进行了两次查询，但它们返回的却是不同的图书（注意：因为是随机选取，所以前后两次的结果有可能相同）：

```
USE mylibrary
CREATE TABLE titlescopy SELECT * FROM titles
SELECT titleID, title FROM titlescopy
ORDER BY RAND() LIMIT 1

titleID  title
59  PHP and MySQL Web Development
SELECT titleID, title FROM titlescopy
ORDER BY RAND() random LIMIT 1

titleID  title
2  Definitive Guide to Excel VBA
```

如果想在同一次 *SELECT* 查询里得到两条或更多的随机记录，把这里的 *LIMIT 1* 子句替换为相应的 *LIMIT n* 子句即可。

10.12.2 自备随机数的数据表

刚才介绍的办法有一个明显的弊端：在比较大的数据表上效率很低下——MySQL 必须把所有的记录都读入内存并在那里按随机顺序对它们进行排序，而这一切只是为了选取一条记录。

因此，对于一个比较大的数据表，在创建它的时候专门定义一个用来存放随机数的数据列（不妨把它命名为 *random*）会是一个更好的办法。当然，也完全可以等到有必要的时候再去创建这个 *random*

数据列，就像下面这个例子那样：第一条 *ALTER TABLE* 命令给数据表 *titlescopy* 增加了一个 *random* 数据列，第二条 *ALTER TABLE* 命令给这个新数据列配上了索引，随后的 *UPDATE* 命令在这个新数据列里为每一条现有记录存入了一个随机数。（注意：在插入新记录的时候，千万不要忘记为新记录的 *random* 字段提供一个新的随机数。MySQL 目前还不支持使用诸如 *RAND()* 之类的函数为某个数据列定义默认值的做法。）

```
ALTER TABLE titlescopy ADD random DOUBLE
ALTER TABLE titlescopy ADD INDEX (random)
UPDATE titlescopy SET random = RAND()
```

现在，可以用下面这样的命令来选取一条随机记录：

```
SELECT titleID, title FROM titlescopy
WHERE random > RAND() ORDER BY random LIMIT 1
```

上面这条查询命令乍看上去比前面那个示例的效率高不到哪儿去，但实际情况却大不一样：MySQL 现在只需要从数据表里读出一条满足 *random>RAND()* 条件的记录即可；因为 *random* 数据列上有一个索引，所以这一次访问速度非常快。

千万不要忘记写出 *ORDER BY* 子句！否则 MySQL 将把它遇到的第一条满足 *random>RAND()* 条件的记录作为返回结果，而那条记录的 *random* 字段值不见得就是与 *RAND()* 最接近的随机数。换句话说，随机记录的选取将取决于数据记录在数据表里的存放顺序，而这就不是真正的随机选取了。

当然，增加一个 *random* 数据列的办法也不是没有缺点：

- 数据表需要占用更多的硬盘空间（因为增加了数据列和索引）。
- 数据表里的修改操作将变得比以前慢（因为多了一个 *random* 数据列上的索引）。
- 只适合选取一条随机记录。（可以用 *LIMIT 5* 得到 5 条“随机”记录，但后面 4 条其实并不随机——它们是排序排出来的。）
- 在插入新记录的时候，必须为新记录的 *random* 字段提供一个新的随机数。

10.12.3 利用 *id* 数据列选择随机记录

上面提供的两种解决方案都有点儿自找麻烦的味道：这其实是个如何从总共 *nmax* 条记录当中随机选出第 *n* 条的问题，而 *nmax* 是可以用 *SELECT COUNT(*) FROM table* 命令轻松查出的一个值。很可惜，SQL 语言没有用来读取第 *n* 条记录的命令。数据记录的存放顺序由 MySQL 内核（数据表驱动程序）决定，从外部无法对此加以控制。

有一种解决方案是利用一个 *AUTO_INCREMENT* 数据列：假设某个数据表里有 1000 条记录，那么它的 *id* 数据列就应该包含 1~1000 的整数值，于是就可以用 *WHERE id=CEILING(RAND()*1000)* 来选出一条随机记录。这个办法听起来很不错，但无法保证 *id* 数据列里的编号值没有“缝隙”，只要从这个数据表里删除过记录，就会留下这种“缝隙”，而这些“缝隙”是以后再怎么往这个数据表里插入新记录也填不上的。

10.13 全文索引

如果正在使用 SQL 搜索某个字符串里的一个单词，构造出来的查询命令就往往是下面这样：

```
SELECT * FROM tablename WHERE column LIKE '%word%'
```

这个查询绝对可以达到目的，但问题是等多长时间才能看到答案。这样的命令是 MySQL 数据库系统里最消耗时间的查询（比这更糟糕的情况大概只有一种，即正使用着一条这样的命令在多个数据列

里搜索着多个单词): MySQL 不仅需要把数据表里的所有记录全都读取过来, 还需要进行无数的字符串比较操作, 而这些操作还没有索引可以帮得上忙。

可是, 让计算机为难的事对用户来说却未必, 在因特网搜索驱动程序里随手输入几个搜索字符串早已成为众多网民的日常习惯, 而这么做并不需要与复杂的搜索条件打交道。

如果想在 MySQL 里对这类查询做出高效率的处理, 就需要使用全文索引 (full-text index)。这是一种特殊的索引, 它会把在某个数据表的某个数据列里出现过的所有单词生成为一份清单。MySQL 从 3.23.23 版本开始在 MyISAM 数据表上提供了对这种索引的支持。

10.13.1 基础知识

1. 创建一个全文索引

为了给一个已经存在的数据表创建一个全文索引, 应该执行以下命令。可以给出任意多个 *xxxText* 和 *xxx(VAR)CHAR* 数据列:

```
ALTER TABLE tablename ADD FULLTEXT(column1, column2)
```

当然, 全文索引也可以随新数据表一同创建:

```
CREATE TABLE tablename (id INT NOT NULL AUTO_INCREMENT,
                      column1 VARCHAR(50), column2 VARCHAR(100),
                      PRIMARY KEY (id), FULLTEXT (column1, column2))
```

注解 全文索引目前只能在 MyISAM 数据表里创建 (不适用于 InnoDB 数据表)。

少于 3 个字符的单词不会被包括在全文索引里, 因而也就无法利用全文索引去检索。(可以通过修改 MySQL 配置文件 my.cnf 或 my.ini 的办法来改变这个默认值: 修改选项 *ft_min_word_len=3*, 重新启动 MySQL 服务器, 然后用 *REPAIR TABLE tablename QUICK* 命令为有关的数据表重新生成全文索引。)

2. 全文检索

全文检索需要使用 SQL 表达式 *MATCH ... AGAINST ...*, 如下所示:

```
SELECT * FROM tablename
WHERE MATCH(column1, column2) AGAINST('word1 word2 word3') > 0.001
```

这个查询将把 *column1* 数据列或 *column2* 数据列至少包含着 *word1*、*word2*、*word3* 这 3 个单词之一的数据记录全部查找出来。在关键字 *MATCH* 后面列出的数据列必须与当初生成全文索引时列出的数据列相互对应。在关键字 *AGAINST* 后面列出的被检索单词不区分字母的大小写情况, 它们的先后顺序也无关紧要, 但少于 3 个字符的单词通常会被忽略, 不参加匹配。

MATCH ... AGAINST ... 表达式将返回一个浮点数作为它本身的求值结果, 这个数字反映了结果记录与被检索单词的匹配程度。如果没有匹配到任何记录, 或者是如果匹配到的结果记录太多反而被忽略, *MATCH ... AGAINST ...* 表达式将返回 0。

条件表达式 *MATCH ... > 0.001* 的作用是排除那些 *MATCH* 返回值太小 (也就是匹配程度不够高) 的结果记录, 这有助于减少浮点数误差对检索结果的影响。

注解 要想让上例中的 *MATCH(title, subtitle)* 命令能够执行, 必须提前创建一个包括且仅包括这两个数据列的全文索引。包括更多个数据列的全文索引、分别为这两个数据列建立的全文索引 (一个是 *title* 数据列上的, 另一个是 *subtitle* 数据列上的) 在上例中的查询里都不能用。

3. 对全文检索结果进行排序

MATCH 表达式还可以用来对全文检索的结果进行排序。下面这个查询将返回 5 条最匹配结果。请注意，这里的查询条件必须用关键字 *HAVING* 来给出，这是因为 MySQL 不允许在 *WHERE* 条件里使用假名（如本例中的 *mtch*）：

```
SELECT *,  
       MATCH(column1, column2) AGAINST('word1 word2 word3') AS mtch  
  FROM tablename  
 HAVING mtch > 0.001  
 ORDER BY mtch DESC  
 LIMIT 5
```

4. 布尔检索表达式

在默认的情况下，在关键字 *AGAINST* 后面列出的被检索单词按逻辑或（*OR*）关系相互组合。从 MySQL 4.0 开始，还可以用一个布尔表达式来组合那些被检索单词，但这需要在整个检索表达式的后面加上 *IN BOOLEAN MODE* 字样。在布尔检索模式里，可以使用表 10-3 里的特殊字符来构造检索表达式。

表 10-3 布尔检索表达式（*IN BOOLEAN MODE*）

符 号	含 义
+ <i>word</i> （加号）	单词 <i>word</i> 必须出现在数据记录里，即表达式 <i>AGAINST('+ word1 + word2' IN BOOLEAN MODE)</i> 相当于这两个单词的逻辑与（ <i>AND</i> ）关系
- <i>word</i> （减号）	单词 <i>word</i> 不允许出现在数据记录里
~ <i>word</i> （波浪符）	单词 <i>word</i> 不应该出现在数据记录里。与- <i>word</i> 的情况相比，这些记录并不是完全被排除，只是不优先考虑而已；所以~ <i>word</i> 很适合用来排除一些似是而非的“噪音”单词，这有助于提高检索速度
< <i>word</i> （小于号）	降低单词 <i>word</i> 的优先考虑程度（匹配程度）
> <i>word</i> （大于号）	提高单词 <i>word</i> 的优先考虑程度（匹配程度）
<i>word</i> *（星号）	搜索以 <i>word</i> 开头的单词（如 <i>word</i> 、 <i>words</i> 、 <i>wordless</i> 等）
" <i>word1 word2</i> "（双引号）	给定单词不仅必须出现在数据记录里，它们的先后顺序也必须得到匹配；此时区分字母的大小写情况
()（圆括号）	圆括号可以用来构造复杂的表达式。例如： <i>AGAINST('+mysql +(buck book) IN BOOLEAN MODE')</i> 将搜索包含着单词 <i>mysql</i> 和 <i>buck/book</i> 二者之一的数据记录。

下面是一个布尔模式的全文检索命令：

```
SELECT * FROM tablename  
WHERE MATCH(column1, column2)  
      AGAINST('+word1 +word2 -word3' IN BOOLEAN MODE)
```

布尔检索的一个最大缺点是 *MATCH* 表达式将只能返回 1（找到匹配记录）或者是 0（没找到匹配记录），它不再返回一个表示匹配程度的浮点数值。

注解 在进行全文索引时，不允许使用 *LIKE* 等操作符的匹配模板。唯一的例外是 *word**，但通配符 “*” 只允许出现在单词的末尾。

5. 同时对多个数据表进行全文检索

全文检索只能用于单个数据表。要想对多个关联数据表里的数据进行检索，必须为每一个数据表配上必要的全文索引。对多个数据表进行全文检索的 SQL 命令非常复杂，很难构造，也很难优化。（在

10.13.2 节里有一个这方面的例子。)

6. 不足与缺陷

为了进行全文检索，数据表里至少要有 3 条记录。在笔者进行的测试中，对只包含 1 条或 2 条记录的数据表进行全文检索都以失败告终。一般来说，数据表越大，全文检索的效果就越好；对比较小的数据表进行全文检索有时会返回一些难以理解的结果。

全文检索以整个单词作为匹配对象。单词稍有变形（比如复数形式、加上后缀等）就会被认为是另外一个单词，这种情况往往增加检索的难度。

若想把所有相关的单词全部查找出，就必须把它们都放到 *AGAINST* 表达式里（比如说，为了把与单词 *book* 有关的记录全部查找出，就需要使用 *AGAINST('book books')* 或 *AGAINST('book books bookcase, bookend')* 这样的表达式）。这种情况可以让通配符“*”来帮忙，但它只能在布尔模式下使用（比如 *AGAINST('book*' IN BOOLEAN MODE)*）。

在全文检索中，只有那些由字母、数字、“'”（单引号）和“_”（下划线）构成的字符串被认为是单词。还好，带有注音符号的字母（比如 ü à é）仍是字母。请注意，像“C++”这样的表达式不再被认为是单词，所以无法通过全文检索来查找。

单词的长度必须是 4 个或更多字符才能参加全文检索。这意味着像“SQL”这样的缩写形式无法通过全文检索来查找。（这一限制可以通过修改 MySQL 服务器的配置来调整；详见下一小节。）

全文检索不区分字母的大小写（这不是什么大问题，但有时会让人感到不方便）。

全文检索目前只能在 MyISAM 数据表上使用（不适用于 InnoDB 数据表）。

全文索引的创建速度相当慢。这不仅表现在新建一个全文索引的时候，还表现在有了全文索引之后的各种数据修改操作中。

7. 对少于 3 个字符的单词进行全文检索

正如前面提到的那样，全文检索在默认的情况下将只对 4 个或更多个字母的单词进行搜索。这么做是为了避免诸如“a”、“and”、“the”之类的单词把全文索引“撑”得太大，但这同时也使得像“ADO”和“PHP”这样的单词无法通过全文检索来查找。

可以通过 MySQL 服务器 (*mysqld*) 的配置参数 *ft_min_word_len* 为全文索引设置一个最小单词长度。在 MySQL 配置文件 (*/etc/my.cnf* 或 Windows\my.ini) 的 “[mysqld]” 段落里输入或修改这个参数即可。为了让修改生效，必须重新启动 MySQL 服务器并生成全文索引（最简单的办法是使用 *REPAIR TABLE tablename QUICK* 命令）。

10.13.2 图书检索

为了下面的示例，在完全由 InnoDB 数据表构成的 *mylibrary* 数据库里创建了一个新的 MyISAM 数据表 *fulltitles*。这里使用了 CREATE TABLE 命令的一种特殊语法：MySQL 将先执行 SELECT 命令，再把它的结果添加到新数据表里，同时还根据这个结果把新数据表的数据列名和数据列属性确定了下来：

```
CREATE TABLE fulltitles ENGINE = MyISAM
SELECT titleID, title, subtitle FROM titles
```

接下来，再为新数据表的 *titles* 和 *subtitles* 创建一个全文索引：

```
ALTER TABLE fulltitles ADD FULLTEXT(title, subtitle)
SELECT title,
       MATCH(title, subtitle) AGAINST('excel basic') AS fulltextmatch
FROM fulltitles
```

```
HAVING fulltextmatch > 0.001
ORDER BY fulltextmatch DESC

title          fulltextmatch
Visual Basic      2.7978503848587
Visual Basic Datenbankprogrammierung 2.7667480431589
VBA-Programmierung mit Excel 7       2.3731654830885
Definitive Guide to Excel VBA        2.3467841568515
Excel 2000 programmieren             2.3467841568515
```

在上面的查询里，条件表达式 *fulltextmatch*>0.001 必须用 HAVING 关键字给出，这是因为 MySQL 不允许在 WHERE 条件里使用假名（见本例中的 AS *fulltextmatch*）。下面这个变体的效率要略高一些，但需要多打不少字：

```
SELECT title, MATCH(title, subtitle) AGAINST('excel basic') AS fulltextmatch
FROM fulltitles
WHERE MATCH(title, subtitle) AGAINST('excel basic') > 0.001
ORDER BY fulltextmatch DESC
```

按“书名加作者”进行检索

全文索引可以只为一个数据列定义。如果想同时对“书名”和“作者”（比如 *excel* 和 *kofler*）进行检索，就必须给有关的数据表全部配上全文索引，然后在查询命令里为每一个索引给出 MATCH 表达式。不过，这么构造出来的查询命令往往效率很低（尤其是在对大数据表进行检索时）。

```
CREATE TABLE fullauthors ENGINE = MyISAM
SELECT authName, authID FROM authors
ALTER TABLE fullauthors ADD FULLTEXT(authName)
SELECT title, authname FROM fulltitles, fullauthors, rel_title_author
WHERE fulltitles.titleID = rel_title_author.titleID
AND fullauthors.authID = rel_title_author.authID
AND MATCH(title, subtitle) AGAINST('excel kofler')
AND MATCH(authName) AGAINST ('excel kofler')

title          authname
VBA-Programmierung mit Excel 7 Kofler Michael
Excel 2000 programmieren      Kofler Michael
Definitive Guide to Excel VBA Kofler Michael
```

假如 *fulltitles* 数据表包含 10 万条图书记录，*fullauthors* 数据表包含同样多的作者记录，那么上面这个查询即使使用了全文索引也会相当慢。更有效率的做法是把有关的数据全部保存到同一个数据表里。在下面的例子里，先在 *fulltitles* 数据表里增加了一个用来保存作者名字的 *authors* 数据列，然后为 *title*、*subtitle* 和 *author* 这 3 个数据列创建了一个新的全文索引：

```
ALTER TABLE fulltitles ADD COLUMN authors VARCHAR(255)
UPDATE fulltitles SET authors =
(SELECT GROUP_CONCAT(authname SEPARATOR ', ')
FROM authors, rel_title_author
WHERE authors.authID = rel_title_author.authID
AND fulltitles.titleID = rel_title_author.titleID
GROUP BY fulltitles.titleID)
ALTER TABLE fulltitles DROP INDEX title
ALTER TABLE fulltitles ADD FULLTEXT(title, subtitle, authors)
```

现在，“书名”和“作者”的全文检索命令变得简单多了：

```
SELECT title, authors FROM fulltitles
WHERE MATCH(title, subtitle, authors)
AGAINST('+excel +kofler' IN BOOLEAN MODE)
```

10.13.3 论坛文章检索

下面的例子与 *myforum* 数据库里的 *messages* 数据表有关。先用以下命令创建必要的全文索引：

```
USE myforum
ALTER TABLE messages ADD FULLTEXT(msgText, subject)
```

下面的查询找到了近 500 条与单词 MySQL 有关的记录：

```
SELECT COUNT(*) FROM messages
WHERE MATCH(msgText, subject) AGAINST('mysql') > 0.001
COUNT(*)
464
```

这些记录只是 MySQL 认为与单词 MySQL 最有关，是不是所需要的还必须由本人去判断：MySQL 的全文检索机制不具备人类的洞察力，在对有关记录做出取舍时不是根据文章的内容而是根据“MySQL”这个单词在文章里的出现频率来做出判断，所以怎么说也有点儿“撞大运”的味道：

```
SELECT subject, msgID,
       MATCH(msgText, subject) AGAINST('mysql') AS mtch
  FROM messages
 ORDER BY mtch DESC
 LIMIT 5
subject                         msgID  mtch
help needed with figuring out error message    332  3.368218421936
where is MySQL database?                  359  3.329582452774
MySQL access ...                      357  3.159619092941
An update to the first edition of MySQL?      775  3.151849031448
MySql ASP.NET & VB.NET sample code?     2585  3.133650779724
```

同时搜索 *mysql* 和 *odbc* 这两个字符串的结果是命中了 67 条记录：

```
SELECT COUNT(*) FROM messages
WHERE MATCH(msgText, subject)
      AGAINST('+mysql +odbc' IN BOOLEAN MODE)
COUNT(*)
67
```

为了查出前 5 个最匹配记录，对这个查询稍微做了些修改：在 *WHERE* 子句里，使用的是布尔检索表达式 '*+mysql +odbc*'，它只能返回 0 或者是 1；但在排序时，使用的是表达式 '*mysql odbc*'。此外，还为这个查询增加了一个限制条件 *forumID=1006*，让它只在用英语讨论 MySQL 问题的讨论组里进行搜索：

```
SELECT LEFT(subject, 40) AS subj,
       msgID,
       MATCH(msgText, subject) AGAINST('mysql odbc') AS mtch
  FROM messages
 WHERE MATCH(msgText, subject) AGAINST('+mysql +odbc' IN BOOLEAN MODE)
   AND forumID=1006
 ORDER BY mtch DESC
 LIMIT 5
```

subj	msgID	mtch
Connecting ASP with MySQL	1975	5.6947469711304
MySQL on Visual Basic with ADO	1655	5.2777800559998
Access & MySQL	238	4.8645100593567
RecordCount with MySQL	2948	3.7673554420471
MySQL on ASP with ADO	1464	3.3525133132935

10.14 锁定

MySQL 是一个采用了客户/服务器体系结构的数据库系统。这意味着多个程序可以同时去访问数据库、读取数据和进行修改。这么一来，万一有两个程序（客户）在同一时间去修改同一项数据，就会导致一系列问题。

来看一个飞机票预定系统的例子：有两家旅行社在同一时间去查看某个特定的航班是否还有剩余机票（执行 *SELECT ... WHERE flightnumber = ...* 命令）。因为还剩下一张机票，所以两家旅行社得到的回答都是 *OK*。于是，根据这一结果，两家旅行社都决定买下这张机票。如果那家航空公司的机票预定系统的代码足够完善，就应该只有一家旅行社能够收到确认信息，而另一家就只能怪自己运气不够好。可是，万一那个机票预定系统根据几微妙之前还准确无误的 *SELECT* 结果做出了错误的判断，就会导致一个相当糟糕的局面：两家旅行社都收到了确认信息，而这又意味着将会有两位旅客坐在同一航班的同一个座位上。

针对这类问题的解决方案有很多种，其中最巧妙同时也往往是最有效率的办法就是将在下一节讨论的事务处理机制。不过，MySQL 数据库系统的事务处理机制目前还只能在 InnoDB 数据表上使用。如果正在使用的是 MyISAM 数据表，就只能依靠所谓的锁定（locking）机制。简单地说，锁定就是让一个或多个数据表在一段时间内只能由某一个程序（客户）来使用的机制。在锁定解除之前，其他客户无法对被锁定的数据表做任何修改，甚至——根据具体的锁定类型——无法去读取它们。

10.14.1 语法

为了保留一个或多个数据表仅供一个客户使用，需要执行 *LOCK TABLE[S]* 命令：

```
LOCK TABLE table1 locktype, table2 locktype ...
```

上面这个语法中的 *locktype* 代表着锁定类型，可供选用的锁定类型有以下几种：

- **READ**。被锁定的数据表对全体 MySQL 用户可读，但不允许修改（发出 *LOCK* 命令的那位用户也包括在内）。*READ LOCK* 只在数据表上没有任何 *WRITE LOCK* 的时候才能生效。
- **READ LOCAL**。类似于 *READ* 锁定类型，但允许不影响任何现有数据记录的 *INSERT* 命令执行。
- **WRITE**。被锁定的数据表允许当前用户进行读和写，其他用户则完全被排除在外：他们既不能读取数据表中的数据，也不能对数据进行任何修改。*WRITE LOCK* 只在数据表上既没有任何 *READ LOCK*、也没有任何其他的 *WRITE LOCK* 时才能生效。
- **LOW PRIORITY WRITE**。类似于 *WRITE* 锁定类型，但在它等待其他 *READ LOCK* 和 *WRITE LOCK* 全部结束的期间，允许其他用户先施加一个新的 *READ LOCK*。如此安排是为了不影响其他用户对数据表进行的查询，但可能会让 *WRITE LOCK* 开始生效前的等待期变得更长。

解除锁定的命令是 *UNLOCK TABLE[S]*，它不需要任何参数。这条命令将解除当前客户设置的全部 *LOCK*。为了把对其他客户的影响降到最低，应该尽可能早地解除锁定。

提示 MySQL 总是以这样的方式来执行单条命令：不让它受到任何其他命令的影响。因此，在执行单条命令（比如 *UPDATE ... WHERE id=1234* 或者 *DELETE ... WHERE id=5678*）的时候，不必使用锁定机制。

只有当需要连续执行多条彼此独立的命令、而且在这些命令的执行过程中不允许其他客户修改正在使用的数据时，才需要使用锁定机制。比较典型的情况是：需要在执行完一条 *SELECT* 命令之后立刻使用一条 *UPDATE* 或 *DELETE* 命令对其查询结果进行修改。

不要对 InnoDB 数据表使用 *LOCK* 和 *UNLOCK* 命令；在 MySQL 4.0.22 及更早的版本里，那么做会让 MySQL 和 InnoDB 的锁定机制互相干扰。*LOCK* 命令会终止当前事务，而这通常是人们不希望看到的结果。对于 InnoDB 数据表，应该尽量使用事务来锁定各有关记录（而不是锁定整个数据表）。如果真的需要锁定整个 InnoDB 数据表，MySQL 从 5.0.3 版开始提供了一条新的命令 *LOCK TABLE[SJ] TRANSACTIONAL*。

10.14.2 GET_LOCK 和 RELEASE_LOCK 函数

如果只需要锁定两个进程之间的通信，最好不要使用 *LOCK/UNLOCK* 命令而是使用 *GET_LOCK* 和 *RELEASE_LOCK* 函数。*GET_LOCK(name, time)* 函数定义一把“锁”，其中 *name* 是锁的名字，*time* 是锁的持续时间（以秒为单位）。这两个函数通常用 *DO* 命令来执行，比如像 *DO GET_LOCK('abc', 10)* 这样。*RELEASE_LOCK(name)* 函数将释放名为 *name* 的锁。

这两个函数所定义/释放的锁其实与数据表无关；MySQL 服务器、数据库和数据表都不会因为它们而被锁定。事实是这样的：在某个进程用 *GET_LOCK* 函数和给定名字定义了一把锁之后，其他进程就不能再获得一把与之同名的锁——第二个进程会在调用 *GET_LOCK* 函数时被阻塞并一直持续到第一个进程释放那把锁为止。

10.15 事务

InnoDB 数据表与 MyISAM 数据表相比最让人感兴趣的地方是它们支持事务。这意味着我们可以把多条 SQL 命令封装为一个操作来执行。本节将解答以下几个问题：为什么要使用事务？如何实现它们？在使用它们时应该注意哪些问题？MySQL 的事务处理机制是如何工作的？

10.15.1 为什么要使用事务

事务有助于提高一个数据库系统的运行效率和安全性：首先，事务可以确保一组以关键字 *BEGIN* 开头、以关键字 *END* 结尾的 SQL 命令要么作为一个整体执行成功，要么作为一个整体根本不执行。换句话说，即使在事务过程中客户与服务器的连接掉了线、供电中断或是计算机硬件发生崩溃，也不会出现一部分命令执行了、另一部分命令没有执行的情况。

来看一个例子：打算把 100 美元从第 123 号账户转账到第 456 号账户去。为了完成这次转账，先执行以下命令：

```
UPDATE tablename SET value=value-100 WHERE accountno=123
```

然后立刻执行以下命令：

```
UPDATE tablename SET value=value+100 WHERE accountno=456
```

如果在进行这次转账时没有使用事务，那么万一在第一条命令执行之后、第二条命令执行之前系统出了问题，就可能出现第 123 号账户里的余额减少了 100 美元、但第 456 号账户里的余额却没有增加 100 美元的情况。如果使用了事务，这种问题就不可能发生：这两条命令要么全部执行成功、要么都不会真正执行（即使执行了也会被撤销）。

事务还可以保证同一批数据不会被两位用户同时修改。在 MyISAM 数据表上，可以用 *LOCK TABLE* 命令来获得这种效果，但那要以其他客户在进行操作期间都不能访问整个数据表为代价。在 InnoDB 数据表上，因为有了事务机制，所以其他客户只是不能访问正在处理的数据记录而已，其他客户对 InnoDB 数据表中其他数据记录的访问不受影响。

最后，事务可以使代码编程工作变得更加简明。因为事务随时都可以被中断或放弃，所以程序中出错处理部分的代码要比不使用事务的时候容易编写得多。

ACID原则

在回答“为什么要使用事务”这个问题的时候，数据库理论家往往把答案总结为一个单词：ACID。这个单词是英文 Atomicity（原子性）、Consistency（稳定性）、Isolation（隔离性）和 Durability（可靠性）的字头缩写，这四个概念涵盖了数据库理论对一个多用户数据库系统在信息安全方面的基本要求。MySQL 加 InnoDB 数据表驱动程序的组合已经通过了 ACID 测试，符合 ACID 这个单词所代表的原则和具体规定。下面，我们一起来看看这四个概念到底意味着什么。

- **Atomicity（原子性）**。这意味着事务就像原子那样是不可分割的，数据库系统必须保证同一个事务里所有命令要么全部执行成功，要么都不会真正执行（即使执行了也会被撤销）；即使在事务的执行过程中发生了诸如计算机硬件崩溃的极端事件也必须如此。
- **Consistency（稳定性）**。这意味着在事务执行完毕后数据库必须处于一个稳定可用的状态。如果数据库系统发现某个事务让数据库里的数据违反了有关的数据合法性规则（也就是在数据库里出现了非法数据），就会立刻中断这个事务并撤销已经执行过的命令，把数据库恢复到这个事务开始执行之前的状态（ROLLBACK）。这里所说的数据合法性规则也包括各种外键约束条件在内；对外键约束条件的详细讨论见第 8 章。
- **Isolation（隔离性）**。这意味着多个事务可以同时独立运行，在执行过程中不会彼此干扰。每一个事务看到的数据库在这个事务开始之前和结束之后的状态除了这个事务本身做出的修改以外不会发生任何变化。换句话说，即使某一个事务插入、修改或删除了数据记录，只要这个事务还没有被提交，与它同期进行的其他事务就不会受到影响；在这个事务提交后，所有受其结果影响的其他事务将自动撤销并返回一条出错消息给有关用户或客户程序以进行出错处理。¹不过，让事务之间达到百分之百的隔离必须付出巨大的代价，这种代价的外在表现集中在速度方面。因此，ANSI-92/SQL 标准定义了 4 种不同的隔离模式供程序员根据在安全性和速度两方面的具体需要加以选用。不同的数据库系统有着不同的默认隔离模式，InnoDB 数据表驱动程序的默认隔离模式是 *REPEATABLE READ*。我们在 10.15.5 节还将对隔离模式这个概念做进一步讨论。
- **Durability（可靠性）**。这意味着事务本身必须能够经受住软、硬件崩溃或其他意外故障，在故障消除后仍能继续执行。（InnoDB 数据表驱动程序采用的办法是把所有的修改先写入一个日志文件。如果系统在这些修改被实际写入数据库之前发生了崩溃，在 MySQL 服务器重新启动后，InnoDB 数据表驱动程序将利用那个日志文件重新构造出所有的修改再传输给数据库。高可靠性和高速度不可兼得，所以许多数据库系统都采用了某种折衷方案。InnoDB 数据表驱动程序在这方面的做法是提供了一个 `innodb_flush_log_at_trx_commit` 选项，这个选项决定在什么时候才可以把事务保存到日志文件里。）

10.15.2 事务的控制

在默认的情况下，MySQL 将以自动提交（auto commit）模式运行。这意味着每一条 SQL 命令都将被当做一个只包含一条命令的小事务来执行。这与我们正在使用的数据表能不能支持事务没有关系。

有两种办法可以把多条 SQL 命令集合为一个事务来执行：

1. 原书这里漏掉了几句非常关键的话！——译者注

- 可以用 *START TRANSACTION* 或 *BEGIN* 命令来开始一个事务，用 *COMMIT* 或 *ROLLBACK* 命令来结束这个事务。*START TRANSACTION* 或 *BEGIN* 命令将为本个事务暂时关闭“自动提交”模式。如果想开始另外一个事务，就必须用另外一条 *BEGIN* 命令作为开始。
- 可以关闭“自动提交”模式。在此之后执行的所有命令都将被认为是同一个事务，直到遇到 *COMMIT* 命令（提交事务）或 *ROLLBACK* 命令（放弃事务）为止。

再强调一次：只有 InnoDB 数据表支持事务，MyISAM 数据表不支持事务。

1. BEGIN、COMMIT 和 ROLLBACK

BEGIN 或 *START TRANSACTION* 命令开始一个事务。*COMMIT* 命令结束一个事务并保存它做出的修改；*ROLLBACK* 命令放弃一个事务，不保存它做出的修改。

对 InnoDB 数据表不允许以嵌套方式使用事务。如果在还没有为前一个事务发出过 *COMMIT* 或 *ROLLBACK* 命令之前又用 *BEGIN* 命令开始了一个新的事务，前一个事务将按 *COMMIT* 方式结束。

事务由客户负责管理。如果客户与服务器之间的连接在某个事务尚未结束时掉了线，所有未被提交的修改将被放弃（就好像发出了 *ROLLBACK* 命令那样）。

2. 折返点 (savepoint)

从 MySQL 4.0.14 版开始，InnoDB 数据表驱动程序开始支持折返点 (savepoint) 机制：用户可以在事务里用 *SAVEPOINT name* 命令设置一些折返点，以后当用户使用 *ROLLBACK TO SAVEPOINT name* 命令结束事务的时候，发生在折返点 *name* 之前的修改将被写入数据库，而发生在之后的则被放弃。

SAVEPOINT 命令只能在事务里使用，在事务结束时，所有的折返点都将被清除。

3. 事务的自动终止

事务还会因为以下命令而自动终止(视同执行了 *COMMIT* 命令): *ALTER TABLE*、*CREATE INDEX*、*CREATE TABLE*、*DROP DATABASE*、*DROP TABLES*、*LOCK TABLES*、*RENAME TABLE*、*SET AUTOCOMMIT=1*、*TRUNCATE*、*UNLOCK TABLES*。但在另一方面，事务不会因为事务内的某条 SQL 命令在执行时发生的简单错误而终止。

4. 设置“自动提交”模式

自动提交 (auto commit) 模式可以用 *SET AUTOCOMMIT=0* 命令关闭。在这之后，与支持事务的数据表有关的各条命令将自动地被认为是同一个事务里的不同命令，直到遇到 *COMMIT* 或 *ROLLBACK* 命令为止。随后又将自动开始一个新的事务。(也就是说，在发出一条 *SET AUTOCOMMIT=0* 命令之后，就用不着每开始一个事务就执行一次 *START TRANSACTION* 命令了。单是这种方便就足以让人们喜欢使用 *SET AUTOCOMMIT=0* 命令了。)

这里要特别提醒大家一句：在发出 *SET AUTOCOMMIT=0* 命令之后，万一客户与服务器的连接掉了线——无论有意还是无意，没有用 *COMMIT* 命令确认过的所有 SQL 命令都将被放弃，它们对数据的修改将不会被写入数据库。

此外，在发出 *SET AUTOCOMMIT=0* 命令之后，如果没有定期执行 *COMMIT* 或 *ROLLBACK* 命令，就可能导致一个非常长的事务。自动提交模式的当前状态可以用 *SET @@autocommit* 命令来查看。

5. 在客户程序里控制事务

绝大多数用于编写 MySQL 应用程序的函数库和 API (例如 JDBC、ODBC、ADO、ADO.NET) 都提供一些专用的函数用来开始和结束一个事务。万一它们不管用，可以通过明确地执行 *SET AUTOCOMMIT* 和 *START TRANSACTION/COMMIT/ROLLBACK* 等命令的办法来使用事务。

10.15.3 事务机制的工作流程

如果想亲身了解一下事务机制的工作流程，必须先对同一个数据库建立两条连接。做这件事最简

单的办法是在两个不同的窗口里执行 mysql 程序，然后轮流在这两个窗口里执行一些 SQL 命令。下面的例子需要假设在 *innorest* 数据库里存在着一个（内容是空白）的 *table1* 数据表。（当然，在实际工作中，数据表里肯定会有那么几条记录，但对这里的演示来说，有一条记录就已经足够了。）

```
CREATE DATABASE innorest
USE innorest
CREATE TABLE table1(colA INTEGER, colB INTEGER)
```

注解 这个例子可以让大家了解在有两位用户几乎同时去修改同一个数据表里的数据时会发生什么。

这个例子很简单，它模拟出来的有两位用户试图同时修改同一个数据表的操作节奏也非常慢，但它已足以让大家认识到这样一个事实：在现实世界里，迟早会发生来自两位用户或两个程序的数据库操作命令会彼此交织在一起的事情，就像这个例子所演示的那样。我们的建议是：只要存在发生这种“撞车”事件的概率，就应该利用事务或者（如果喜欢使用MyISAM数据表的话）LOCK TABLE命令来排除这种可能性。

请看表 10-4。在时刻 0（表 10-4 里最右边那一列），用户¹A 开始了一个事务（BEGIN），他打算通过这个事务对数据表 *table1* 里的一条记录进行修改。

表 10-4 两个并行的事务

用户 A	用户 B	时间
USE innorest	USE innorest	
INSERT INTO table1 VALUES (1, 10)		
SELECT * FROM table1 colA colB 1 10		
START TRANSACTION		0
UPDATE table1 SET colB=11 WHERE colA=1		
SELECT * FROM table1 colA colB 1 11	SELECT * FROM table1 colA colB 1 10	1
	BEGIN	2
	UPDATE table1 SET colB=colB+3 WHERE colA=1	
COMMIT		
SELECT * FROM table1 colA colB 1 11	SELECT * FROM table1 colA colB 1 14	3
	ROLLBACK	
SELECT * FROM table1 colA colB 1 11	SELECT * FROM table1 colA colB 1 11	4
		5

在时刻 1，两位用户在查看 *table1* 数据表时看到了不同的内容。在用户 A 看来，*colA=1* 的那条记录已经有了 *colB=11*，但因为用户 A 还没有提交他的事务，所以这个事务还没有真正执行。于是，用户 B 在时刻 1 去查看同一条记录时看到的仍是初始值 *colB=10*。

1. 直译是“连接”，但不如“用户”好。我们在表里也译做“用户”。——译者注

在时刻 2，用户 *B* 开始了一个事务，他打算通过这个事务给 *colA=1* 的那条记录的 *colB* 字段加上 3。InnoDB 数据表驱动程序觉察到它此时不能去执行这个命令，于是就阻断了用户 *B* 的事务，用户 *B* 不得不停在那里等待用户 *A* 完成他的事务。

在时刻 3，用户 *A* 终于用 *COMMIT* 命令结束了自己的事务，*colB* 字段值也实实在在地变成了 11。现在，用户 *B* 发出的那条 *UPDATE* 命令可以继续执行了。

在时刻 4，用户 *A* 看到的是 *colB=11*。用户 *B* 看到的是 *colB=14*，但他本应该看到 *colB=13* 的！

发现问题之后，用户 *B* 撤销了他的事务（如果用户 *B* 非要提交自己的事务，他将收到一条出错消息）¹。

在时刻 5，用户 *A* 和用户 *B* 看到的都是 *colB=11*。用户 *A* 当然很满意；用户 *B* 就不一定是，他或许会再开始一个事务去修改这条记录。

10.15.4 事务与锁定

在绝大多数时候，只要开始执行一个事务，InnoDB 数据表驱动器就会把必要的锁定操作安排好。但在某些场合，InnoDB 数据表驱动器做出的默认安排不见得是最优的。InnoDB 自己也明白这一点，所以它向人们提供了几种可以用来调控其锁定行为的办法。

1. *SELECT ... LOCK IN SHARE MODE*

InnoDB 数据表驱动器的默认安排之一是 *SELECT* 命令即使在已被锁定的数据记录上也能立刻执行，查询结果不受其他客户尚未提交或撤销的事务影响（参见表 10-4 里用户 *B* 在时刻 1 看到的情况）。这么做的好处是用户不会迟迟看不到查询结果，坏处是用户看到的查询结果说不定已经过时了。

如果在发出一条 *SELECT* 命令时加上了关键字 *LOCK IN SHARE MODE* 作为后缀，那么如果有已经开始的事务正在处理它将要查询的数据记录，它就会等到那些事务全部结束之后才开始进行查询。也就是说，如果表 10-4 中的用户 *B* 在时刻 1 发出的是下面这条命令：

```
SELECT * FROM table1 LOCK IN SHARE MODE
```

用户 *B* 就必须等到用户 *A* 的事务完成之后（时刻 3）才能看到查询结果。

如果 *SELECT ... LOCK IN SHARE MODE* 命令本身是某个事务的一部分，在它开始执行之后，在这个事务结束之前，数据表里与其结果记录有关的所有数据记录都将被锁定，其他客户将只能读取、不能修改或删除那些数据记录。因为这种查询使用了关键字 *SHARE*（“共享”），所以人们把它设置的锁称为共享锁（shared lock）。共享锁可以确保在事务过程中读取的数据记录不会是其他客户正在修改或删除的。

共享锁不禁止其他客户来读取这些数据记录，就算其他客户在读取这些数据记录时也使用了 *SELECT ... LOCK IN SHARE MODE* 命令也是如此。共享锁只阻断其他客户对被锁定数据记录的修改和删除操作，其他客户只能在事务完成之后才能对那些数据记录进行修改或删除。

2. *SELECT ... FOR UPDATE*

关键字 *FOR UPDATE* 代表着对普通 *SELECT* 命令的另一种功能扩展，这个关键字将给数据表里与这条 *SELECT* 命令的结果记录有关的所有数据记录加上一把排它锁（exclusive lock）。

排它锁不禁止其他客户使用普通的 *SELECT* 命令来读取被锁定数据记录，但其他客户对那些数据记录进行的修改和删除操作以及使用 *SELECT ... LOCK IN SHARE MODE* 命令进行的读取操作都将被

1. 原书这里漏掉了这句重要的话。——译者注

阻断。共享锁和排它锁的唯一区别在于是否阻断其他客户发出的 *SELECT ... LOCK IN SHARE MODE* 命令。

3. INSERT、UPDATE 和 DELETE

这 3 条命令都会在开始执行后给自己将要修改/删除的数据记录加上一把排它锁直到本个事务结束。如果在 *INSERT*、*UPDATE*、*DELETE* 命令执行期间需要检查数据表之间的关联/引用关系（外键约束条件），所有关联数据表里受其影响的数据记录也都将被加上一把排它锁。

4. 防插入锁

在默认的情况下，InnoDB 数据表驱动程序在遇到带有范围条件表达式（比如 *WHERE id>100* 或 *WHERE id BETWEEN 100 AND 200*）的 *SELECT ... LOCK IN SHARE MODE*、*SELECT ... FOR UPDATE*、*UPDATE* 或 *DELETE* 命令时还会多给它们加上一把防插入锁（gap and next key lock）。加上这把锁的效果是：不仅符合给定条件的现有数据记录会被锁定，连符合给定条件但当前并不存在的数据记录也会被锁定，而这意味着其他事务将无法在有关的数据表里插入符合给定条件的数据记录。（对哪些命令使用防插入锁还要取决于当时的事务隔离模式；参见 10.15.5 节。）

比如说，如果在某个事务里执行了 *SELECT ... WHERE id>100 FOR UPDATE* 命令，那么在事务完成之前，其他用户将无法插入 *id>100* 的新记录。

5. 死锁

InnoDB 数据表驱动程序能够自动识别死锁（deadlock，两个或多个进程互相阻塞，彼此都在等待对方结束，结果是谁都无法继续执行）条件并加以消除：触发死锁的那个进程（后来者）将收到一条出错消息，该进程尚未提交的所有 SQL 命令按 *ROLLBACK* 方式撤销；另一个进程（先来者）将继续执行。

不过，如果死锁是因为一些既访问 InnoDB 数据表又访问其他类型数据表的 SQL 命令而引起的，InnoDB 数据表驱动程序不一定能把它们识别出来。为了让那些陷入这种死锁条件的客户不至于无限期地僵持下去，我们可以通过配置参数 *innodb_lock_wait_timeout=n* 来设置一个最长等待时间（默认值是 50 秒）。

10.15.5 事务的隔离模式

在开始一个事务之前，必须先为它定义一个隔离模式：

```
SET [SESSION|GLOBAL] TRANSACTION ISOLATION LEVEL
  READ UNCOMMITTED | READ COMMITTED |
  REPEATABLE READ | SERIALIZABLE
```

这类用途的 *SET* 命令有 3 种用法：

- 不带关键字 SESSION 或 GLOBAL 的 SET 命令。*SET* 命令设置的隔离模式将只对下一个事务有效。这与下面两种更为常用的语法的不同之处在于：前者的效力仅限于下一个事务，后者的效力则会一直持续到下一条用来设置隔离模式的 *SET* 命令或自然失效。
- *SET SESSION*。为当前连接会话设置隔离模式，其效力将一直持续到下一条用来设置隔离模式的 *SET* 命令或当前连接会话结束，即客户与服务器断开连接。
- *SET GLOBAL*。为此后所有新建的 MySQL 连接设置隔离模式（当前连接不包括在内）。

隔离模式的影响力体现在事务命令的执行方式上。在下面的介绍里，隔离模式按从低到高的顺序排列。这意味着隔离模式 *READ UNCOMMITTED* 能让我们获得最快的访问速度（不会彼此阻塞），而隔离模式 *SERIALIZABLE* 能让用户在有多个客户同时进行修改时获得最大的安全保障。下面是对这四

级隔离模式的一个简单描述。

□ **READ UNCOMMITTED**。SELECT 命令在读取有关记录时会把尚未完成的其他事务做出的修改也考虑在内。也就是说，SELECT 命令与其他事务之间根本没有隔离。（*read uncommitted* 的意思是：即使是其他事务还没有用 COMMIT 命令提交的数据也允许进入 SELECT 命令的查询结果。）如果把表 10-4 里的例子放在 READ UNCOMMITTED 模式下执行，那么用户 B 在时刻 1 看到的就是 *colA=1* 和 *colB=11*。

请注意，虽然 READ UNCOMMITTED 模式不隔离 SELECT 命令，但它却隔离 UPDATE 命令。仍以表 10-4 为例，即便是在 READ UNCOMMITTED 模式下，用户 B 在事务里发出的 UPDATE 命令也会从时刻 2 开始一直被阻断到用户 A 的事务完成为止。这意味着 UPDATE 命令在 READ UNCOMMITTED 模式下完全可以正确执行。

□ **READ COMMITTED**。SELECT 命令在读取有关记录时会把已经用 COMMIT 命令提交过的其他事务做出的修改也考虑在内。这同样意味着同一个事务里完全一样的两条 SELECT 命令可以有不同的结果——如果另一个同期进行的事务稍早结束。与 READ UNCOMMITTED 模式相比，这种模式对 SELECT 命令的隔离效果虽然好了一些，但还谈不上完美。

□ **REPEATABLE READ**。SELECT 命令在读取有关记录时不把其他事务做出的修改考虑在内，不管那些事务是不是用 COMMIT 命令提交过。这种隔离模式完全符合 ACID 原则对 SELECT 命令的隔离要求。同一个事务里完全一样的 SELECT 命令将返回完全一样的结果（当然，这还需要以这个事务本身没有去修改有关数据为前提）。

□ **SERIALIZABLE**。这种隔离模式与 REPEATABLE READ 模式很相似，唯一的区别是这种模式会自动地把普通的 SELECT 命令也当做 SELECT ... LOCK IN SHARE MODE 形式的命令来执行并给受其影响的数据记录统统加上一把共享锁。

在默认的情况下，InnoDB 数据表驱动程序将使用 REPEATABLE READ 隔离模式，这种模式在绝大多数场合都能在不怎么影响事务速度的情况下为各种事务提供足够的隔离效果。人们对 InnoDB 数据表驱动程序进行的优化是按照它将工作在 REPEATABLE READ 模式下的假设进行的，它在这个模式下的锁定操作执行起来非常有效率。

可以通过 MySQL 配置文件里的 transaction-isolation 选项改变这一默认设置。在设置这个选项时，隔离模式名字里的多个单词必须用连字符隔开，如 *transaction-isolation = read-committed*。

可以用下面这条命令来查看当前连接和/或服务器的隔离模式：

```
SELECT @@tx_isolation, @@global.tx_isolation
```

<i>@@tx_isolation</i>	<i>@@global.tx_isolation</i>
READ-UNCOMMITTED	REPEATABLE-READ

隔离模式与防插入锁

正如前面提到的，InnoDB 数据表驱动程序支持“防插入锁”。这意味着范围条件表达式（比如 WHERE *id>100* 或 WHERE *id BETWEEN 100 AND 200*）不仅会导致符合给定条件的现有数据记录被锁定，还会导致符合给定条件但当前并不存在的数据记录也被锁定。

命令里有范围条件表达式是 InnoDB 数据表驱动程序使用防插入锁的大前提，是否真的使用还要具体取决于正在执行的命令和事务隔离模式。在 REPEATABLE READ 和 SERIALIZABLE 模式下，防插入锁将用于以下命令：SELECT ... LOCK IN SHARE MODE、SELECT ... FOR UPDATE、UPDATE、DELETE。

10.15.6 出错处理

在使用事务的时候，服务器返回某些特定出错消息的概率会比平时增加许多。因此，务必在编写的代码里对命令的执行情况进行检查，就算是不那么关键的命令（比如只读不写的 *SELECT* 命令）也不要放过，千万不要想当然地认为它们在执行时不会遇到问题。这里要特别注意因为以下几个原因而导致的执行出错：

首先，在某个事务还没有结束之前，该事务所涉及的数据记录将被锁定，其他客户无法执行任何将影响到那些记录的操作。比如说，在客户 A 执行了 *BEGIN* 命令并正在执行 *INSERT INTO table VALUE (1, 2)* 命令的时候，如果客户 B 试图执行 *SELECT * FROM table*，这条命令就会被阻塞，直到客户 A 用 *COMMIT* 或 *ROLLBACK* 命令结束他的事务为止。有不少 MySQL 客户端程序（比如 *mysql* 程序）会耐心地一直等到前一个事务结束（而不是等待一段倒计时时间后放弃）。但另外一些客户端程序却可能发生倒计时或其他问题。为了避免这类问题，除了要在程序代码里对这种情况进行检测和处理外，还应该尽快结束事务。

其次，每一个事务都有可能会因为陷入死锁状态而被终止。前面讲过，InnoDB 数据表驱动程序能自动识别死锁状态（两个事务互相阻塞），为了避免事务之间的死锁导致 MySQL 服务器被长期阻塞，InnoDB 数据表驱动程序会强行终止其中一个客户的事务。这意味着必须注意检查事务命令的执行状态并根据从服务器返回的出错消息做出妥善处理。

第 11 章

访问权限与信息安全

在生活中，或者说至少是在人类的社会组织当中，并不是所有的人都可以获得各方面的所有信息。对于 MySQL 数据库也是一样的，通常情况下，建立一个数据库以后，并不是所有的人都可以看到数据库里的所有数据（不管是改变或是删除它）。为了保护数据不被探查（或未经授权的篡改），MySQL 采用了一种双层的访问控制机制。第一层次用来检查各位用户是否有权与 MySQL 进行通信。第二层次用来检查各位用户都有权对哪些数据库、数据表和数据列进行哪些操作（如 *SELECT*、*INSERT*、*DROP* 等）。

本章将深入讨论 MySQL 的访问控制系统，包括它的内部管理模式以及一些可以用来改变访问权限的辅助工具。

11.1 简介

11.1.1 客户与 MySQL 服务器之间的连接

在 MySQL 的任何访问管理机制能够生效之前，客户程序（例如，Perl 脚本、命令编译器 mysql 或者 MySQL 查询浏览器）必须能够连接到 MySQL 服务器。本节概括出了连接的可能形式。

如果想试验各种连接协议，最好的方法就是使用本书第 4 章讨论过的程序 mysql，同时使用选项 --protocol。允许的设置是 *tcp*、*socket*、*pipe* 以及 *memory*。通过使用命令 *STATUS*，可以查看当前设置的连接协议：

```
> mysql -u root -p --protocol=tcp
Enter password: xxx
mysql> status
...
Connection: localhost via TCP/IP
```

1. 网络连接

当客户程序和服务器是在不同的计算机上运行的时候，情况非常简单：两个程序之间的连接必须要通过网络协议 TCP/IP。为了做到这一点，必须满足下面的条件。

- 计算机必须是由 TCP/IP 协议来链接。使用命令 ping computername 可以很容易地检查这一点。
- 由 MySQL 默认使用的 3306 端口不得被防火墙阻隔。

2. 本地计算机（localhost）上的连接

当客户程序和服务器是在同一台计算机上运行的时候，可能有以下几种方法。

- TCP/IP。网络协议也可以用于运行在单个计算机上的程序之间的连接。在 Windows 环境下，通常是这种情况。

- **套接字文件（仅用于 UNIX/Linux）。**套接字文件使 UNIX/Linux 环境下两个程序的连接效率更高。这不是一个普通文件（它不包含数据并只有 0 字节）。在 UNIX/Linux 环境下，本地连接默认使用套接字文件，因为这样要比使用 TCP/IP 协议速度更快。
- **命名管道（仅用于 Windows2000/XP）。**或多或少地与 UNIX/Linux 环境下使用套接字文件相似，在 Windows 环境下使用命名管道。不过，在 MySQL 服务器上，默认情况下命名的管道是禁止的，必须使用选项 enable-named-pipes 来激活它。而且，MySQL 的客户程序还必须支持这种连接方法。由于这些条件很少能够得到满足，因此，命名管道的方法在实际中很少用到。
- **共享内存（仅用于 Windows2000/XP）。**对应于 UNIX 使用的套接字文件，共享内存是 Windows 环境下的另外一种使用方法。在这里，客户程序和服务器连接使用的是同一个内存区域。但在实际使用中，共享内存与命名管道一样极少用到。这是因为，只有当 MySQL 服务器的配置文件包含有选项 shared_memory（这不是一个默认选项）时，才能使用这种形式的连接。即使满足这个条件，客户程序还必须支持这种连接方法，而通常情况都不是这样。从理论上来说，命令编译器 mysql 具有这个功能（选项 protocol=memory），但在实际测试中，它会返回一个出错信息：*Can't open shared memory; client could not create request event.*

在本书的下一步内容中，只讨论两种最普通的连接形式：TCP/IP 和套接字文件。

3. 定义MySQL服务器支持的连接形式

MySQL 服务器提供哪种连接方法取决于程序是如何编辑以及它是如何被配置的（参见第 14 章）。运行中的 MySQL 服务器的当前位置可以用一些 SHOW VARIABLES 命令来确定：

```
SHOW VARIABLES LIKE 'skip_networking'
skip_networking          OFF
SHOW VARIABLES LIKE 'port'
port                      3306
SHOW VARIABLES LIKE 'socket'
socket                   /var/lib/mysql/mysql.sock
SHOW VARIABLES LIKE 'named%'
named_pipe                OFF
SHOW VARIABLES LIKE 'shared%'
shared_memory              OFF
shared_memory_base_name    MYSQL
```

这些命令的效果是：这个 MySQL 服务器支持 TCP/IP，即网络功能没有由于安全原因通过选项 skip-networking 而被禁止。TCP/IP 连接使用 3306 号端口。

这个 MySQL 服务器支持通过套接字文件 /var/lib/mysql/mysql.sock 进行通信的连接。（变量 socket 只在 MySQL 服务器的 UNIX/Linux 版本中可以使用。）

这个 MySQL 服务器既不支持命名管道也不支持共享内存。（这三个变量 named_pipe、shared_memory 和 shared_memory_base_name 只能够使用于 MySQL 服务器的 Windows 版本下。）

11.1.2 访问管理

通常情况下，并不是每个人都可以执行所有的数据库操作。但可以肯定的是，肯定会有一个或几个管理员拥有更多的操作权限。但是，如果允许所有的用户都可以执行他们所请求的改变记录或删除整个数据库的操作，无疑在安全性方面会冒很大的风险。

可以有许多种不同程度的访问权限。以一个公司的雇员数据库为例，应该是允许所有的雇员读取部分数据库的内容（如查找某人的电话号码），而数据库的其他内容应该是隐藏的（如个人记录）。

在设置这种访问权限的时候，MySQL 提供了一个准确、相互衔接的系统。在 MySQL 文档中，这个系统被称为访问权限系统（access privilege system），这个系统里的各个列表被称为访问控制表（access control list, ACL）。ACL 表的管理是在 *mysql* 数据库中各有关数据表的内部进行，这部分内容将在以后详细讨论。

本节假定客户程序和服务器之间的连接在理论上都是可以建立起来的。

1. 设置访问权限

有几种方式可以用于设置访问权限：

- 最简单和最方便的方法就是以图形用户界面使用管理程序。两个这样的程序——MySQL Administrator 和 phpMyAdmin——已经在第 5 章和第 6 章讨论过。但是注意，如果根本不懂 MySQL 访问权限的概念，那么最方便的用户界面也不一定有什么帮助。
- 可以使用 *INSERT* 和 *UPDATE* 命令直接改变 *mysql*。
- 可以使用 SQL 的命令 *GRANT* 和 *REVOKE*，它们可以提供很大的方便。
- 可以使用 Perl 脚本程序 *mysql_setpermission.pl*。这个脚本程序甚至比使用 *GRANT* 和 *REVOKE* 命令还要简单，当然，它假设用户安装并运行着 Perl。

2. 用户名、密码和主机名

MySQL 数据库的访问过程分为两个阶段。在第一个阶段中，要检查是否允许用户来进行对 MySQL 的连接。（这种检查可不是说要检查用户是否具有读取或改变任何数据库内容的权限，而是要检查这样的访问是否具有将 SQL 命令传递到 MySQL 上的能力。然后，根据 MySQL 的安全设置，它会决定这个命令是否应该执行。）

当习惯于以提供一个用户名和密码的方法来进入到一个多用户计算机系统的操作系统中时，MySQL 要对第三个信息进行检查：用以访问 MySQL 的计算机名（主机名）。由于 MySQL 整个的安全系统都是基于这 3 个方面的信息，因此，我们有必要对它们进行一些讨论。

□ **用户名。**用户名就是借以存在于 MySQL 中的名字。MySQL 用户名的管理与操作系统所管理的登录名没有任何关系。当然，经常的情况是用户名与登录名使用同一个名字，但是应该知道，这两个名字的管理是相互独立的，不存在使两种操作同步的机制（例如，当操作系统生成一个新的用户名字的时候，一个新的 MySQL 用户名随之生成的机制是不存在的。）

用户名最多可以有 16 个字符，并且区分字母的大小写。从原则上来说，这样的用户名不一定必须要使用 ASCII 字符集当中的字符，但由于不同的操作系统处理特殊字符的方法不同，因此使用 ASCII 字符集以外的字符会导致问题的发生，因此应该尽可能避免。

□ **密码。**密码的性质与用户名一样。MySQL 使用的密码与用于操作系统的密码没有任何关系，即使它们有可能是一样的。在 MySQL 中，密码以 45 位加密格式储存，这样的格式允许检查密码，但不允许重新构造密码。即使一个攻击者进入到了 MySQL 的 user 数据表中，也不能修改密码。

注意 从安全因素考虑，对操作系统和 MySQL 数据库，不要使用同一个密码！

MySQL 的密码需要经常用于脚本文件的文本、配置文件和程序当中，因此，它们经常处于被拦截的危险当中。如果一个攻击者获取到了访问数据库的密码，同时还发现进入到操作系统的密码与访问数据库的密码一样，那么这个攻击者会欣喜若狂的。

这个警告特别要针对于那些允许客户程序除了访问 MySQL 数据库以外，还可以直接访问计算机（FTP、Telnet、SSH）的网络服务供应商。对于这两种访问形式，一定要给出不同的密码！

从4.1版本开始，MySQL使用了一个新的身份验证系统和强度更高的加密密码。较早的MySQL客户程序因此不再能够与MySQL服务器建立连接。针对这一问题的解决方法将在本章的后面部分来介绍。

与用户名一样，密码也要区分字母的大小写，但它可以是任意字长。尽管可以使用 ASCII 字符集以外的字符，但不推荐这样的做法。

□ **主机名**。在建立连接的过程中，通常必须要指定在 MySQL 服务器上运行的计算机。这个计算机名一般来讲是 *hostname*。主机名可以是以 IP 地址给出（如 192.168.23.45）。只有在 MySQL 服务器与客户程序是运行在同一台计算机上的时候，才可以省略主机名字的说明部分。

在决定访问权限的过程中，MySQL 要检查访问请求是来自哪里的信息，即来自于哪一个主机名。当客户程序和服务器是运行在不同计算机上的时候，对主机名的检查可能由于以下两个原因而发生问题：

- MySQL 服务器必须要能够识别这个计算机的名字。在建立连接的过程中，服务器首先是接到客户程序的 IP 地址，然后计算机试图通过一个 *name server*（域名服务器）寻找对应的计算机名。如果成功地寻找到了计算机名，计算机就会使用 IP 地址而不是计算机名来决定是否允许访问。
- 假如成功地识别了名字，域名服务器将以域名名字或不以域名名字返回这个主机名，这要取决于它是如何配置的（如，*mars* 或 *mars.sol*）。只有在主机名与 *mysql.host* 数据表中的条目完全一致的时候，才允许对 MySQL 的访问。

注解 当客户程序和服务器运行在UNIX/Linux环境下的同一台计算机上并使用套接字文件连接的时候，MySQL服务器不使用本地计算机名来进行验证，而将名字*localhost*作为主机名。这就意味着，根据连接是使用TCP/IP协议还是套接字文件，MySQL服务器在访问系统中区分两种不同的连接方式：

username1 / password1 / computername 适用于TCP/IP;
username2 / password2 / localhost under 适用于套接字文件。

有关建立连接过程中对这些特殊情况及各种其他有关主机名问题的讨论，放在本章的后面部分。

□ **默认值**。在建立一个与 MySQL 连接的过程中，如果没有给出其他的参数，那么在 UNIX/Linux 环境下，当前的登录名就用做用户名，而在 Windows 环境下，用户名为字符串 *ODBC*。作为密码使用的是空白字符串。主机名为 *localhost*。

可以很容易地测试这一点。启动 mysql 程序并执行命令 *status*，当前用户名就会显示在 *current user* 的数据行中。在 Linux 环境下，执行下面的命令：

```
kofler:~ > mysql -p
Enter password: xxx
Welcome to the MySQL monitor.
mysql> status
...
Current user: kofler@localhost
Connection: localhost via UNIX socket
```

如果在建立一个连接的时候，必须要提供一个用户名（而不是登录名）、一个主机名和一个密码，那么 mysql 的启动如下所示：

```
kofler:~ > mysql -u surveyadmin -h uranus.sol -p
Enter password: xxx
```

```
mysql> status
...
Current user:      surveyadmin@saturn.sol
Connection:        uranus.sol via TCP/IP
```

这里，有了一个客户（计算机为 *saturn.sol*）和服务器（计算机名为 *uranus.sol*）之间的 TCP/IP 连接。

MySQL 能够识别术语 *anonymous user*。在建立一个与 MySQL 的连接过程中，当所有的用户名都是允许的时候，可以使用这种表达方式。在这种场合里，*status* 命令显示 *i* 作为连接建立过程中使用的用户名，但是在内部，是以一个空白字符串作为用户名。这一点对于其他访问权限的检查尤为重要。在本章后面有关访问系统的内部工作部分，还要讨论这个问题。

当一个脚本建立了与数据库的连接，并且没有提供其他用户名的时候，那么，作为用户名使用着的就是账户名字，在这个账户名字下，PHP 编译器被执行。（作为一种规则，Apache 也是在同一个账户下执行。因此，从安全因素考虑，绝大多数 UNIX/Linux 操作系统所使用的账户几乎没有什么权限，如 *apache* 或者 *wwwrun*。）

3. 默认的安全设置

依据 MySQL 安装版本以及是在 Windows 还是 Linux 环境下安装的 MySQL，对于 MySQL 的访问管理有一些安全或不安全的默认设置。

- **当前的 Windows 版本。**从 4.1.5 和 5.0.2 版本开始，MySQL 服务器是和新的安装程序一起发行的，这已经在第 2 章进行了讨论。如果按照第 2 章给出的方法进行安装，那么 MySQL 安装是相对比较安全的。权限不受限制的唯一用户是系统管理员 *root*，而他也仅能够从本地计算机进行访问并且需要使用密码来登录。
- **较早的 Windows 版本。**与新的版本相比较，较早的 Windows 版本下的默认安装可是非常不安全。这时，*root* 可以从本地计算机上登录也可以从网络登录，而且这种登录可以不需要使用密码。更重要的是，所有本地系统的用户都可以不使用密码来进行访问，并且具有很大的权限。例如，他们可以读取所有的数据库，修改这些数据库，甚至可以完全删除这些数据库。最后，任何人（任意用户名、网络中的任何计算机）都可以进行访问，尽管这些用户并不具有权限而且在建立连接之后也没有什么要做的事情。
- **UNIX/Linux。**UNIX/Linux 环境下的默认设置虽然不像老版本下的 Windows 安装那样危险，但也绝对谈不到安全：*root* 用户可以在本地计算机上不使用密码来进行对数据库的访问。而且，本地系统上的所有用户（任意用户名）都允许不使用密码对 MySQL 进行访问；与 *root* 用户相比较，这些用户没有任何权限（甚至不能够执行 *SELECT* 命令）。在默认的情况下，从外部计算机进行的访问是不允许的。

注意 更简明地来说：只要 *root* 用户没有使用密码保护，任何人都可以使用这个名字对 MySQL 数据库进行访问，而且可以对数据库随心所欲。对于老版本的 Windows 安装，甚至不需要使用本地计算机：*root* 用户可以通过网络来连接。在 11.2 节中，将会介绍只要做一点工作就可以改变默认设置从而获得更高安全性的方法。

对于那些没有什么 UNIX/Linux 使用经验的人来说，UNIX/Linux 下的 *root* 用户所起的作用就类似于 Windows 下的 Administrator，即是一个权限不受限制的超级用户。但是，建议 MySQL 下的用户名不要与操作系统的登录名有任何联系。MySQL 下的 *root* 用户之所以有这么大的权限，是由于 MySQL 的默认设置就是这样，并不是 UNIX/Linux 下的 *root* 用户就应该具有如此大的权限。可以使用一个不是 *root* 的名字来作为 MySQL 具有管理访问权限的用户。事实上，从安全角度考虑，最好使用一个其他的用户名来代替 *root*。

读者大概会问自己这样一个问题：默认的安全设置是从哪里来的？这些设置就储存在 *mysql* 数据库里，在 Windows 下它是由 MySQL 的安装程序创建的。在 UNIX/Linux 环境下，在 RPM 安装过程中，脚本程序 *mysql_install_db* 被执行，从而创建并设置了 *mysql* 数据库。

在 UNIX/Linux 环境下，可以手动执行 *mysql_install_db*，重新创建 *mysql* 数据库。要做到这一点，必须首先停止 MySQL 服务器的运行。执行脚本程序所使用的账户，必须与用于执行 MySQL 服务器的账户相同（通常为 *mysql*）。可以作为 *root* 用户执行脚本程序 *mysql_install_db*，这时，必须拥有 *mysql* 数据库的目录，并且要改变包含在里面的下列文件：

```
root# mysql_install_db
root# chown mysql -R /var/lib/mysql/mysql
```

□ 测试数据库。无论 MySQL 数据库是在哪种环境下安装的，所有连接到 MySQL 的用户都可以

创建一个测试数据库。唯一的条件就是数据库的名字前面必须是以 *test* 开始。注意，不但所有的 MySQL 用户都可以创建这样的一个数据库，而且每一个用户都可以读取、改变以及删除这个数据库中的所有数据。事实上，任何人都可以将这个数据库整个删除掉。简而言之，*test* 数据库中的数据是没有任何保护的。

本章后面给出的表 11-2、表 11-3 和表 11-5 描述了这个默认设置。要想更清楚地了解这部分内容，就必须掌握 *mysql* 数据库里的 *user* 和 *db* 数据表的功能和用法，这部分内容也将在本章讨论。

11.2 急救

也许读者还没有完全准备好深层次地探讨 SQL 访问管理的内容。如果是这种情况，本节提供了一些用于急救的措施。

这些方法假定已经使用命令 *mysql* 和 *mysqladmin*（参见第 4 章）建立了与 MySQL 服务器的连接。当然，也可以使用图形化用户界面执行同样的命令。

11.2.1 保护 MySQL 安装

下面的命令假定能够以 *root* 身份登录 MySQL 服务器：

```
> mysql -u root
Welcome to the MySQL monitor. ...
```

如果 *root* 用户受到密码的保护，那么上面的命令就会失败。必须提供选项-p，然后将会被问到 *root* 的密码：

```
> mysql -u root -p
Enter password: xxx
Welcome to the MySQL monitor. ...
```

本节后面内容里的所有命令都与 *mysql* 数据库有关，在这个数据库中储存着用户的设置。（如果是在 *mysql* 程序里执行命令，不要忘记在每一个命令的末尾加一个分号。）下面这条命令将把 *mysql* 数据库设置为本次会话的默认数据库：

```
USE mysql
```

下面这条 *SELECT* 命令可以使读者对全体用户有一个快速的了解。当然结果有可能看上去和读者的计算机上的不一样。当 MySQL 服务器长时间运行并定义了许多 MySQL 用户的时候，这份名单可能会很长。

```
SELECT user, host, password FROM user
```

<u>user</u>	<u>host</u>	<u>password</u>
root	localhost	
root	uranus.sol	
	uranus.sol	
	localhost	

这里显示出当前有 4 个用户，而且他们都不需要密码的保护。*uranus.sol* 是本地系统的名字。下面列出的是为了取得更好的安全性而可能会采取的步骤。

设置 root 密码。一个简单的 *UPDATE* 命令就可以为所有的 *root* 用户建立一个密码，而不管他们是从哪里注册的：

```
UPDATE user SET password = PASSWORD('secret') WHERE user = 'root'
```

重新命名 root 用户。在默认的情况下，具有管理职责的 MySQL 用户名为 *root*。如果某人想要攻击该系统，首先要做的是尝试不用密码使用 *root* 登录能否成功，或者是能否猜到 *root* 的密码。因此，如果选择另外一个不是 *root* 的名字，就可以显著地增加系统的安全性：

```
UPDATE user SET user = 'myroot' WHERE user = 'root'
```

这么做的后果是以后为了执行系统管理任务而登录时必须一直使用这个新名字。从现在开始，不能再用 *mysql -u root -p* 登录 MySQL 服务器了，必须使用 *mysql -u myroot -p* 这样的命令。

删除匿名用户。为了增强安全性，应当删除掉所有的匿名用户，也就是那些在 *user* 数据列里空着的用户。只有在 MySQL 的 *user* 数据表里明确登记的用户才应该能够注册：

```
DELETE FROM user WHERE user = ''
```

为每一个用户设置密码。通常情况下，*user* 数据表包含着的用户都应当有密码。作为数据库管理员，可以使用命令 *UPDATE*，对每一个知道的用户给出一个密码。这样，其他用户不再能够注册到 MySQL 服务器，他们就得与管理员联系，而管理员可以对每一个这样的用户设置一个密码：

```
UPDATE user SET password = PASSWORD(secret) WHERE password = ''
```

对可以从任何地方注册的用户的处理。最后一个不安全的方面就是那些可以从任何地方注册的 MySQL 用户。可以通过 *host* 数据列里仅有的一百分号（%）来识别这些用户：

```
SELECT user, host, password FROM user WHERE host = '%'
```

对如何处理这样的用户没有一成不变的规则。在个别情况下，这样的用户允许具有这种权限。但是，将允许的网络地址限制到本地网络或是一个 IP 地址的范围就足够了。（注意，MySQL 只有在它不能够确定主机名的时候才检查 IP 地址。）下面的命令只允许从本地网络注册（在这个例子中使用的名字是 *sol*）：

```
UPDATE user SET host = '%.sol' WHERE host = '%'
```

为了使所有的这些命令生效，现在必须执行命令 *FLUSH PRIVILEGES*。MySQL 在 RAM 中保留一个 *mysql* 数据库的副本，为提高速度，它是通过 *FLUSH PRIVILEGES* 命令来刷新的：

```
FLUSH PRIVILEGES
```

11.2.2 创建新的数据库和用户

1. 创建新的数据库

数据库管理员的日常任务之一就是建立新的数据库，并使用户可以使用它（用户可以插入数据表

并用数据来填充它)。

这项工作很容易完成。只要使用mysql执行下面的两条命令就可以了。结果是创建了数据库*forum*, 用户*forumadmin*对这个数据库有不加限制的访问权限。

```
CREATE DATABASE forum
GRANT All ON forum.* TO forumadmin@localhost IDENTIFIED BY 'xxx'
```

创建第二个对数据库的访问经常是很有用的,这个访问具有较少的权限(因此在安全性方面的风险较小):

```
GRANT Select, Insert, Update, Delete ON forum.*
TO forumuser@localhost IDENTIFIED BY 'xxx'
```

根据实际情况,让普通用户获得锁定(*LOCK*)数据表、创建临时数据表和执行存储过程(从MySQL 5.0 版本开始)的能力或许是个好主意。*IDENTIFIED*部分被省略。即, *forumadmin*的密码保持不变:

```
GRANT Lock Tables, Create Temporary Tables, Execute ON forum.*
TO forumuser@localhost
```

当 MySQL 服务器是运行在 UNIX/Linux 环境下的时候, @*localhost* 仅允许一个通过套接字文件的本地注册。如果还希望允许通过 TCP/IP 协议进行本地注册(Java 程序所要求),就必须对 *username@localcomputername* 授予权限(*ALL*、*SELECT*、*INSERT*等):

```
GRANT ALL          ON forum.* TO forumadmin@rechnername IDENTIFIED BY 'xxx'
GRANT privileges ... ON forum.* TO forumuser@rechnername IDENTIFIED BY 'xxx'
```

2. 创建新的用户

在刚刚创建的数据库*forum*运行一段时间以后,如果另外一个用户从其他的计算机上要求对这个数据库具有不加限制的访问权限,那么下面的命令就授予计算机 *uranus.sol* 上的用户 *forumadmin2* 全部权限:

```
GRANT ALL ON forum.* TO forumadmin2@uranus.sol IDENTIFIED BY 'xxx'
```

如果想允许 *forumadmin2* 可以在任何计算机上登录,命令如下所示:

```
GRANT ALL ON forum.* TO forumadmin2@'%' IDENTIFIED BY 'xxx'
```

如果甚至想允许用户 *forumadmin2* 可以访问所有的数据库(不仅仅是数据库*forum*),命令如下所示:

```
GRANT ALL ON *.* TO forumadmin2@uranus.sol IDENTIFIED BY 'xxx'
```

与 *root* 用户相比较, *forumadmin2* 唯一没有的权限就是改变访问权限的权限。但是,只要做一点工作就可以解决这个问题:

```
GRANT ALL ON *.* TO forumadmin2@uranus.sol
IDENTIFIED BY 'xxx' WITH GRANT OPTION
```

现在, *forumadmin2* 具有和 *root* 一样的权限。

11.2.3 授予创建个人数据库的权限

在有许多用户、每位用户又都有他自己的数据库的时候(例如,网络服务供应商的系统上面),只能由数据库管理员为各位用户创建一个新数据库的做法就不妥当了,那会给管理员增加许多不必要的工作负担。在这种情况下,把创建数据库的权限授予某些用户是一个好方法。为了防止出现数据库的混乱,以至于不知道哪个数据库是属于谁,通常的做法是做出规定:每位用户创建的数据库必须以他本人的用户名作为数据库名的开头。例如,用户名是 *kofler* 的用户只能使用诸如 *kofler_test*、*kofler_forum*、*kofler1*、*koflerXy* 之类的名字来创建数据库,不能使用不以 *kofler* 开头的名字创建数据库。

在下面的部分中，继续这项工作使用的用户名为 *username*。如果 MySQL 数据库尚不知道该用户，那么首先要创建它：

```
GRANT USAGE ON *.* TO username@localhost IDENTIFIED BY 'xxx'
```

但是，必须要使用 *INSERT* 命令，虽然 *GRANT* 命令更加方便，但是它不允许在数据库名的说明部分出现通配符，因此，需要直接对数据库 *db* 做出改变。允许的数据库名是由 *username* 和通常带有符号 % 的任意其他字符所组成。下面命令的黑体部分给出了相关的内容：

```
INSERT INTO mysql.db (Host, Db, User, Select_priv, Insert_priv,  
Update_priv, Delete_priv, Create_priv, Drop_priv, Grant_priv,  
References_priv, Index_priv, Alter_priv, Create_tmp_table_priv,  
Lock_tables_priv, Create_view_priv, Show_view_priv)  
VALUES ('localhost', 'username%', 'username', 'Y', 'Y', 'Y', 'Y',  
'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y')
```

为了使这些改变生效，最后要执行命令 *FLUSH PRIVILEGES*：

```
FLUSH PRIVILEGES
```

11.2.4 忘记 *root* 密码情况的处理

如果忘记了 MySQL 的 *root* 密码，而且其他的 MySQL 用户由于没有足够的管理权限而不能恢复忘记掉的密码，该怎么办呢？

MySQL 的开发者们早就想到了出现这种情况的可能性。处理这个问题的步骤如下：首先终止 MySQL（即终止 MySQL 的服务器 mysqld），然后使用选项 *skip_grant_tables* 重新启动它。这样做的结果是没有加载具有访问权限的数据表。现在，可以删除经过加密的 *root* 密码，终止 MySQL，然后不使用指定选项而再一次启动它。这时，就可以对 *root* 用户给出一个新的密码了。

在处理这种情况的时候，必须要有 MySQL 所运行其上的操作系统的系统管理员权限。

第一步就是终止 MySQL 的运行。在 Windows 环境下，在 Service Manager 中结束 MySQL (Settings | Control Panel | Administrative Tools | Services)。而在 UNIX/Linux 环境下，执行下面的命令：

```
root# /etc/rc.d/mysql stop
```

现在，在 MySQL 配置文件 [mysqld] 部分中，输入选项 *skip_grant_tables*。配置文件在 UNIX/Linux 环境下放置于 /etc/my.cnf；在 Windows 环境下，放置于 C:\Programs\MySQL\MySQL Server n.n\my.ini：

```
# changes in my.cnf and my.ini  
...  
[mysqld]  
skip_grant_tables  
...
```

然后，重新启动 MySQL 服务器，在 Service Manager 中 Windows 环境下的对话前面已经讨论过，在 UNIX/Linux 环境下，使用下面的命令：

```
root# /etc/rc.d/mysql start
```

这时，可以使用 mysql 来为主机名 *localname* 和实际的计算机名（仅在 UNIX/Linux 下）重新设置密码：

```
root# mysql -u root  
Welcome to MySQL monitor.  
mysql> USE mysql;  
Database changed.  
mysql> UPDATE user SET password=PASSWORD('new password')  
> WHERE user='root' AND host='localhost';  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> UPDATE user SET password=PASSWORD('new password'wort')
   > WHERE user='root' AND host='computername';
Query OK, 1 row affected (0.00 sec)
```

在 *mysql.user* 数据表中做出这些改变以后,再一次如前所述的那样或是使用命令 mysqladmin shutdown 停止 MySQL 服务器。从 MySQL 配置数据库中删除选项 skip_grant_tables 并且再一次启动 MySQL 服务器。

注意 当然,这里所讨论的方法也一样可以被希望偷窥数据并试图操纵这些数据的攻击者所使用。唯一的条件(幸运的是攻击者可不容易做到这一点)是要拥有UNIX/Linux的*root*访问权限(或是Windows下的*administrator*权限)。从这里就可以看到,不仅是保护MySQL,而且要保护MySQL运行其上的计算机是多么重要。(有关UNIX/Linux和Windows操作系统的安全性的讨论以及一些其他相关问题的讨论,不在本书的论述范围之内。)

11.3 访问控制机制的内部工作原理

读者可能会想略过对这一基础部分的阅读,特别是由于访问权限的管理是相当复杂的。不过,还是要强烈建议读者做好准备工作,尽可能地阅读关于 *mysql* 数据表的讨论。无论是采用哪种方式来设置系统的安全性,首先了解系统的内部工作机制总是会有所帮助的。即使自己不是访问权限的管理者,它们也是要受到系统管理员的管理,对内部工作机制的了解也会发挥作用。只有在了解了它们都是如何工作的,才有可能告诉系统管理员你需要什么。

11.3.1 两级访问控制

对 MySQL 数据库的访问控制分为两个阶段:首先,检查用户是否具有建立与 MySQL 的连接的权力。这种检查的完成是要依靠对 3 个方面信息的检查:用户名、主机名和密码。

只有在建立连接的检查完成以后,第二阶段的访问控制才开始实施,这种控制将细化到每一条数据库命令。例如,要执行一个 *SELECT* 命令,MySQL 会检查该用户对数据库、数据表和数据列是否具有执行这个命令的权限。要执行一个 *INSERT* 命令,那么 MySQL 要检查是否允许该用户改变数据库、数据表直到数据列的内容。

11.3.2 权限

MySQL 是如何管理哪个命令是可以执行的这样的信息呢?MySQL 是使用储存着 *privileges* 的数据表进行管理的。如果一个用户,称为 *athena*,具有对数据库 *owls* 的 *Select* 权限,那么该用户就可以读取数据库 *owls* 中的所有数据(但是不能改变它)。如果 *athena* 具有全局的 *Select* 权限,那么该用户就可以读取所有储存于 MySQL 的数据库。

由 MySQL 识别的权限如表 11-1 所示。请注意, *mysql* 数据表的相应列中的名字总是以 *_priv* 结尾。因此, *Select* 权限是储存在 *Select_priv* 列中。部分列中的名字是缩写(如, *Create_temp_table* 表示 *Create Temporary Table* 权限)。

注意 过去,几乎每一个MySQL的版本都要引入一些新的权限。与 3.23 版本相比较, MySQL 5.0 版本提供了以下新的权限: *Alter Routine*, *Create Routine*, *Create Temporary Table*, *Excute*, *Lock Tables*, *Show Databases*, *Replicate Client*, *Replication Slave*, *Super*。(一份完整的权限清单以及对它们

的简要描述可以使用命令 *SHOW PRIVILEGES* 来得到。)

如果正在升级 MySQL 的一个老版本，必须同时升级 *mysql* 数据库里的各有关数据表，这样才能使用新的权限。在 UNIX/Linux 环境下，这个升级任务可以用 *mysql_fix_privilege_tables* 脚本来帮忙，而在 Windows 环境下，可以使用同名的 *.sql 文件（参见第 14 章）。

表 11-1 MySQL 权限

MySQL 权限	含 义
	用于数据表访问
<i>Select</i>	可以读取数据 (<i>SELECT</i> 命令)
<i>Insert</i>	可以插入新的记录 (<i>INSERT</i>)
<i>Update</i>	可以改变现有的数据记录 (<i>UPDATE</i>)
<i>Delete</i>	可以删除现有的记录 (<i>DELETE</i>)
<i>Lock Tables</i>	可以锁定数据表 (<i>LOCK</i>)
	用于改变数据库、数据表和视图
<i>Create</i>	可以创建一个新的数据库和数据表
<i>Create Temporary Table</i>	可以创建一个临时数据表
<i>Alter</i>	可以重命名数据表并改变它的结构
<i>Index</i>	可以添加或删除数据表索引
<i>References</i>	不属于文档内容，也许在将来会出现，用于建立两个数据表之间的关联
<i>Drop</i>	可以删除现有的数据库和数据表
<i>Create View</i>	可以定义视图
<i>Show View</i>	可以使用 <i>SHOW CREATE VIEW</i> 命令来检查视图定义（从 MySQL 5.0 版本开始）
	用于存储过程（从 MySQL 5.0 版本开始，参见第 13 章）
<i>Alter Routine</i>	可以改变现有的存储过程
<i>Create Routine</i>	可以定义新的存储过程
<i>Execute</i>	可以执行存储过程
	用于数据库访问
<i>File</i>	可以读取和改变本地文件系统的文件
<i>Create User</i>	可以创建新的 MySQL 用户（从 MySQL 5.0.3 版本开始）
	用于 MySQL 管理
<i>Grant Option</i>	可以赋予其他用户个人的权限
<i>Show Databases</i>	可以看到一份全体数据库的清单 (<i>SHOW DATABASES</i>)
<i>Process</i>	可以看到其他用户的 MySQL 进程 (<i>SHOW PROCESSLIST</i>)
<i>Super</i>	可以终止其他用户的 MySQL 进程 (<i>KILL</i>)，创建存储过程和触发器，并且可以改变和执行一些管理命令 (<i>CHANGE/PURGE MASTER, SET GLOBAL</i>)
<i>Reload</i>	可以执行各种命令 (<i>reload, refresh, flush-xxx</i>)
<i>Replication Client</i>	可以决定镜像系统中参与者的状态
<i>Replication Slave</i>	可以通过镜像机制读取 MySQL 服务器的数据
<i>Shutdown</i>	可以关闭 MySQL

在 MySQL 的在线文档中，会遇到权限 *All* 和 *Usage*（例如，在 *GRANT* 命令的描述中）。*All* 意味

着赋予 *GRANT* 以外的所有权限，而 *Usage* 的意思是拒绝所有的权限。

因此，*All* 和 *Usage* 自身不是独立的权限，只是在执行 *GRANT* 命令时，使用它们就可以不用列出权限。

绝大多数权限的含义不用进一步解释就应该非常清楚。对一些不是特别清楚的权限，下面的内容给出了一些解释。有关 *Replication Client* 和 *Replication Slave* 的讨论放在第 14 章镜像机制部分的内容里。

本书中的权限表示方法是首字母大写，其余字母小写，以区别于 SQL 命令 (*Select* 权限和 *SELECT* 命令)。MySQL 数据库并不区分权限名中的字母大小写情况。

1. *Grant Option* 权限

Grant Option 权限表示 MySQL 用户可以分配访问权限。(使用 SQL 命令 *GRANT* 可以非常简单地做到这一点，由此得出了权限的名字。)但是，分配权限的能力是限制在授予者拥有的权限范围之内的。也就是说，没有用户可以给另外一个用户他或她自身都没有的权限。

注意 很多人都没有意识到 *Grant Option* 权限存在着安全风险。比如说，在创建一个测试数据库的时候，为了让开发团队的所有成员都能不受限制地对它进行访问，数据库管理员也许会不假思索地把 *db* 数据表里各相关记录项的所有权限字段都设置为 *Y* (稍后介绍 *db* 数据表)。

这样，对这个测试数据库¹ 拥有最高权限的每位用户都可以把这些权限转授给其他数据库上的用户 (他本人或其他 MySQL 用户均可)。这无疑会带来非常高的安全风险，只可惜不少数据库管理员从未意识到这一点。

2. *File* 权限

具有 *File* 权限的 MySQL 用户可以使用 SQL 命令直接访问 MySQL 服务器运行其上的计算机的文件系统，例如，可以使用命令 *SELECT...INTO OUTFILE* 或是 *LOAD DATA*，也可以使用函数 *LOAD_FILE*。

当然，访问文件系统的时候，要注意文件系统的访问权限。(在 UNIX/Linux 环境下，MySQL 服务器通常是运行在 *mysql* 账户下。因此，只有那些可以被 UNIX/Linux 用户读取的文件，*mysql* 能够被读取。)然而，*File* 权限在安全性方面经常有较大的危险。

3. *Process* 和 *Super* 权限

Process 权限使得用户可以通过命令 *SHOW PROCESSLIST* 查看到一份全体进程 (连接) 的清单，其他用户的进程也包括在内。(普通用户即使没有这个权限也可以查看自己的进程。)

Super 权限允许用户使用 *KILL* 命令来结束他自己以及其他用户的进程。(如果没有授予 *Super* 权限，那就只能结束当前进程。)

Super 权限还允许执行一些管理命令：*CHANGE MASTER* 命令用来执行镜像系统的客户程序配置，命令 *PURGE MASTER* 用来删除二进制日志文件，命令 *SET GLOBAL* 用来改变全局的 MySQL 变量。

4. *Global* 权限与 *Object* 权限

在 MySQL 数据库中，权限可以是全局的，也可以是针对某一个对象的。*Global* 表示权限对所有的 MySQL 对象有效 (也就是说，对所有数据库、数据表以及数据表的数据列有效)。

有关对象的权限要稍微复杂一些，但它要安全许多。例如，可以允许一个特定的 MySQL 用户改变一个特定的数据表，而不是 MySQL 所管理着的所有的数据表。在使用对象权限的时候，要求相对

1. 原书这里把“数据库”误写为“数据表”，两者在这里的区别非常大。——译者注

应的全局权限没有设置。(允许全局权限以后，不能再限制在对象权限的层次上。)

这种体系的含义在对象权限里也是一样的。首先检查是否允许访问整个数据库；如果不允许，再检查是否允许访问 SQL 命令命名的整个数据表；如果仍不允许，再检查是否允许访问数据表的单个数据列。

5. *information_schema* 数据表

第 9 章中讨论过的虚拟数据表 *information_schema* 是权限管理当中的一个特例。任何允许登录到 MySQL 服务器的人都可以对这样的数据表执行查询命令 *SELECT*。因此，对于这样的数据表，不需要 *Select* 权限。

11.3.3 mysql 数据库

显然，由 MySQL 管理的访问权限是依靠数据库来实现的。这个数据库的名字是 *mysql*，它由几个数据表组成，对应着访问权限的各个方面。

注解 有时，区分“MySQL”的各种不同用途并不是一件简单的事情。在本书中，采用不同的字体在一定程度上区分它们：

MySQL：作为整体的数据库。

mysqld：MySQL 服务器（MySQL 后台程序）。

mysql：MySQL 监视程序（一种命令编译器）。

mysql：管理 MySQL 访问权限的数据库。

1. 数据库 *mysql* 的数据表

数据库 *mysql* 中包含了大量用于各种管理任务的数据表。其中，有 6 个数据表用于管理访问权限。这些数据表经常作为 *grant* 数据表。表 11-2 提供了这些 *grant* 数据表的任务情况。

表 11-2 用于访问权限管理的 *mysql* 数据表

名 字	含 义
<i>user</i>	控制谁（用户名）可以从哪一台计算机（主机名）访问 MySQL。这个数据表也包含着全局权限
<i>db</i>	规定哪个用户可以访问哪个数据库
<i>host</i>	依据有关允许的主机名的信息扩展 <i>db</i> 数据表（那些不在 <i>db</i> 中的主机名）
<i>tables_priv</i>	规定谁可以访问数据库的哪一个数据表
<i>columns_priv</i>	规定谁可以访问数据表的哪一个数据列
<i>func</i>	使之具备 UDFs 管理（user-defined-functions：用户指定函数），该项尚未生效
<i>procs_priv</i>	规定谁可以执行单个的存储过程

当安装了新的 MySQL 以后，*user* 和 *db* 的默认值取决于操作系统。这两个数据表的内容在后面的表 11-4、表 11-5 和表 11-7 中给出。这些设置的作用前面已经讨论过了。（*host*、*tables_priv* 和 *columns_priv* 数据表的初始内容全都是空白。）

在 MySQL 的两级访问系统中，对于第一级访问系统只是用 *user* 数据表（也就是说，用于对 MySQL 的连接访问）。*user* 数据表包含着所有的全局权限。

对于第二级（针对具体对象的访问：数据库、数据表和数据列）访问系统来说，除了要用到 *user*

数据表以外，还要使用数据表 *db*、*host*、*tables_priv* 和 *columns_priv*。

当相应等级的权限设置为 *N* (“NO”) 时，数据表 *db*、*host*、*tables_priv* 和 *columns_priv* 以这个顺序依次实施。换句话说，如果 *Select* 权限赋予了 *user* 数据表中的一个用户，那么其他的 4 个数据表就不再用于检查这个用户是否允许执行 *SELECT* 命令。

因此，要想对访问权限进行精确的区别，就必须把 *user* 数据表里的全局权限全部设置为 *N*。

2. 示例

在开始对每一个数据表进行详细讨论之前，先看一个例子。假定一个 MySQL 用户（姑且将称为 *zeus*）只能读取某个数据表中的某个数据列，除此之外再没有其他任何权限。那么，*zeus*（包括用户名、主机名、密码）必须被输入到 *user* 数据表里，*zeus* 的所有全局权限都必须设置为 *N*。

不但如此，*zeus*（用户名、主机名）还必须要在 *columns_priv* 数据表中注册。在 *columns_priv* 数据表中，还必须设定都允许 *zeus* 访问哪个数据列（数据库名、数据表名、数据列名）。最后，*Select* 权限（仅这个权限）必须要激活。至于数据表 *db*、*host* 和 *tables_priv*，这里没有必要为 *zeus* 用户创建一条记录。

11.3.4 *user* 数据表

user 数据表完成下面 3 个任务（见表 11-3）：

- 这个数据表决定着哪些人对 MySQL 根本就没有任何权限。
- 可以通过这个数据表赋予全局权限。注意这个数据表的数据列名与权限表中的数据列有些区别（如 *Create_temp_table* 数据列表示 *Create Temporary Table* 权限）。
- 从 MySQL 4.0 版本开始，这个数据表包含着几个新的数据列，用于通过 SSL (secure socket layer, 安全嵌套层) 的加密访问、与 X509 标准一致的身份控制和管理这种控制以及数据库连接的数值，即每小时允许执行的最大更新值和查询值。

表 11-3 *user* 数据表一览表

字 段	类 型	NULL	默 认 值
<i>Host</i>	char(60)	No	-
<i>User</i>	char(16)	No	-
<i>Password</i>	char(45)	No	-
<i>Select_priv</i>	enum('N', 'Y')	No	N
<i>Insert_priv</i>	enum('N', 'Y')	No	N
<i>Update_priv</i>	enum('N', 'Y')	No	N
<i>Delete_priv</i>	enum('N', 'Y')	No	N
<i>Create_priv</i>	enum('N', 'Y')	No	N
<i>Drop_priv</i>	enum('N', 'Y')	No	N
<i>Reload_priv</i>	enum('N', 'Y')	No	N
<i>Shutdown_priv</i>	enum('N', 'Y')	No	N
<i>Process_priv</i>	enum('N', 'Y')	No	N
<i>File_priv</i>	enum('N', 'Y')	No	N
<i>Grant_priv</i>	enum('N', 'Y')	No	N
<i>Create_view_priv</i>	enum('N', 'Y')	No	N
<i>Show_view_priv</i>	enum('N', 'Y')	No	N
<i>References_priv</i>	enum('N', 'Y')	No	N

(续)

字 段	类 型	NULL	默 认 值
<i>Index_priv</i>	enum('N', 'Y')	No	N
<i>Alter_priv</i>	enum('N', 'Y')	No	N
<i>Show_db_priv</i>	enum('N', 'Y')	No	N
<i>Super_priv</i>	enum('N', 'Y')	No	N
<i>Create_tmp_table_priv</i>	enum('N', 'Y')	No	N
<i>Lock_tables_priv</i>	enum('N', 'Y')	No	N
<i>Execute_priv</i>	enum('N', 'Y')	No	N
<i>Alter_routine_priv</i>	enum('N', 'Y')	No	N
<i>Create_routine_priv</i>	enum('N', 'Y')	No	N
<i>Create_user_priv</i>	enum('N', 'Y')	No	N
<i>Repl_slave_priv</i>	enum('N', 'Y')	No	N
<i>Repl_client_priv</i>	enum('N', 'Y')	No	N
<i>ssl_type</i>	enum('', 'ANY', 'X509', 'SPECIFIED')	No	0
<i>ssl_cipher</i>	blob	No	0
<i>x509_issuer</i>	blob	No	0
<i>x509_subject</i>	blob	No	0
<i>max_questions</i>	int	No	0
<i>max_updates</i>	int	No	0
<i>max_connections</i>	int	No	0
<i>max_user_connections</i>	int	No	0

1. 访问控制

为了控制谁能够连接到 MySQL 上，必须要有 3 个必要的标识符——正如已经几次谈到过的——用户名、主机名以及密码。这里将看一看这个信息是如何储存在字段 *User*、*Host* 和 *Password* 中的。

□ 用户名。访问控制区分字母的大小写。在 *User* 字段中，不允许使用通配符。如果 *User* 字段为空，那么允许任何用户名访问 MySQL。该用户被当做是匿名用户。这就是说，在第二级访问控制中，不使用实际的用户名，而使用一个空白字符串。

□ 密码。密码必须储存在数据列 *Password* 中，并且使用 SQL 函数 *PASSWORD* 进行加密。不能将密码储存在普通的文本中，而且，不允许使用通配符。如果密码字段为空，那么连接就不需要密码。（空白密码字段并不意味着可以使用任意密码。）

从 MySQL 4.1 版本开始，*Password* 数据列的加密方式进行了改变。密码的加密编码包含着 45 个字符（较早的 MySQL 版本仅有 16 个字符）。

再次提出警告：从安全性出发，绝对不要把用来登录操作系统的密码作为 MySQL 密码。

□ 主机名。主机名可以是一个名字，也可以是一个 IP 地址。这里允许使用通配符“_”（单个任意字符）或“%”（0 或多个字符），例如，192.168.37.%或是%.myfavoriteenterprise.com。IP 地址的格式既可以是 192.168.37.0/255.255.255.0，也可以是 192.168.37.0/24（含义与 192.168.37.% 相同）。如果主机名只是一个字符%，那么可以从任何计算机上进行连接。与用户名一样，主机名也区分字母的大小写。

要想在本地计算机上进行访问，那就以 *localname* 作为主机名。当 UNIX/Linux 用户想通过 TCP/IP 协议（对 Java 应用程序非常重要）来进行本地访问时，那就必须以实际的计算机名作

为主机名。如果允许两种访问格式，用户就必须要定义两次，一次是用主机名 *localhost*，另外一次是用计算机名 *computername*。

提示 通常情况下，主机名必须与域名一同给出。因此，如果计算机 *saturn.sol* 想从网络 *.*sol* 访问，*user* 数据表中的 *Host* 数据列必须包含着字符串 *saturn.sol* 而不仅仅是 *saturn*。不过，根据具体的网络配置（文件 /etc/hosts 或现有的域名服务器配置），有时必须省略主机名的域名部分。本章后面的内容里还会对此做进一步解释。

2. *user* 数据表中的检查顺序

经常会出现这样的情况，在建立一个与 MySQL 的连接过程中，几个记录都与注册数据相匹配。例如，在 Windows 的 *user* 数据表默认设置下（如表 11-5 所示），从本地计算机以 *root* 注册，那么所有 4 个条目都会匹配。MySQL 要决定哪个最接近准确描述（在这个例子中，是对 *root/localhost*，而不是对 "%")。这样做的原因在于，如果与注册相准确匹配，就可以期待拥有更多的访问权限。

为了加快正确选择的记录的速度，*user* 数据表是由 MySQL 在内部排序的。字段 *Host* 作为第一个标准，*User* 作为第二个分类标准。在排序中，没有通配符 “_” 和 “%” 的字符串优先给出。空字符串和字符串 “%” 在排序中最后给出。表 11-4 和表 11-5 给出了 *user* 数据表在其内部各自排序中的默认设置。

注意 *user* 数据表中的条目检查顺序经常导致无法预料的问题。假定 MySQL 的默认设置对用户权限有效，想通过 *Host = '%'* 和 *User = 'peter'* 在 *user* 数据表中增添一个新的用户。如果 *peter* 现在试图从本地计算机注册（即从 *localhost*），条目 *'%'peter'* 并不起作用，其作用的条目是来自默认设置的 *Host = 'localhost'* 和 *User = ''*。原因是 MySQL 使用唯一的主机名 (*Host = 'localhost'*) 作为条目的优先，优于使用通配符 (*Host = '%'*) 的条目。

解决这个问题最简单和最保险的方法是从 *user* 数据表中删除条目 *Host='localhost'* 和 *User=''*（这是两个单引号）。总的来说，从安全性考虑，应该尽量避免 *User* 数据列为空的条目。

3. 权限

User 数据表不但用于 MySQL 的访问控制，而且也用于全局权限。要实现这个目的，目前有难以计数的 *xxx_priv* 字段在发挥作用，它们可以设置成 *Y* 或 *N*。由于 *Y* 代表着 “yes”，这个设置就意味着相关的权限设置成 MySQL 用户的全局设置（对所有数据库、数据表和数据列）。反过来，*N* 表示 “no”（或是法语中的 *non*，德语中的 *nein*，俄语中的 *nyet*，读者肯定知道这个意思），因此，这样的设置说明讨论中的设置不是全局性的，所以数据表 *db*、*host*、*tables_priv* 和 *columns_priv* 将用于对特定对象权限的检查。

4. SSL 加密与 X509 身份标识

要实现客户程序和服务器之间的安全连接，X509 标准可以用于标识用户，并且数据传送可以通过 SSL（secure socket layer，安全嵌套层）加密实施。这要求一个特殊的 MySQL 服务器编译版本，如 MySQL Max。（SSL 功能目前还没有集成到 MySQL 的标准发行版本中。）可以使用 SQL 命令 *SHOW VARIABLES LIKE 'have_openssl'* 来查看 MySQL 版本是否支持 SSL。

注意，数据的加密会影响到 MySQL 服务器的速度，而且，只有在数据传送是通过网络实现，或者说，MySQL 服务器和客户程序是分别运行在不同的计算机上的时候，数据加密才有意义。

本地连接的加密（例如，PHP/Apache 访问一个运行在同一个计算机上的 MySQL 服务器）不会增加应用程序的安全性。在这种情况下，安全性更多地取决于 Apache 和 web 用户之间的连接是如何发生的（例如，通过 HTTPS 加密）。

提示 本书并不详细讨论SSL和X509的配置以及应用程序。更多的资料请访问站点：
<http://dev.mysql.com/doc/mysql/en/secure-connections.html>。

5. 限制MySQL的使用

使用数据列 *max_questions* 和 *max_updates*，可以规定在每个小时之内可以允许执行多少次数据查询（*SELECT* 命令）和数据修改（*INSERT* 和 *UPDATE*）。*max_connections* 规定了每小时可以建立多少个连接；从 MySQL 5.0 版本开始的 *max_user_connections*，对单个用户可以同时具有的连接数目进行限制。默认值为 0 就说明没有任何限制。

6. 默认设置

在本章的简介部分已经讨论过，安装 MySQL 以后，有各种 *user* 数据表的默认设置，它们取决于操作系统。表 11-4 和表 11-5 给出了这些设置（对于 MySQL 5.0.n 版本）。

表 11-4 *user* 数据表在 UNIX/Linux 环境下的默认设置

Host	User	Password	Select_priv to Alter_priv	Super_priv to Repl_client_priv	ssl_xxx, x509_xxx	max_xxx
<i>localhost</i>	<i>root</i>		Y	Y	NULL	0
<i>computername</i>	<i>root</i>		Y	Y	NULL	0
<i>localhost</i>			N	N	NULL	0
<i>computername</i>			N	N	NULL	0

表 11-5 *user* 数据表在 Windows 环境下的默认设置

Host	User	Password	Select_priv bis Alter_priv	Super_priv bis Repl_client_priv	ssl_xxx, x509_xxx	max_xxx
<i>localhost</i>	<i>root</i>	xxxxxx	Y	Y	NULL	0

注意，表 11-5 只对当前的 MySQL 安装版本有效，而且安装执行要像在第 2 章所叙述的那样。对于 Windows 下较早的 MySQL 安装版本，使用的默认设置比 UNIX/Linux 下的默认设置还要不安全。

11.3.5 *user.Host* 数据列

像在本节前面所简要介绍的那样，*user* 数据表的 *Host* 数据列规定了从哪一台计算机上一个特定的用户可以获得访问 MySQL 服务器的权限。计算机名可以使用通配符“%”或是由 IP 地址给出。

初看上去十分简单的别名在实际使用中会出现许多问题。因此，本节中介绍了许多的背景资料以及解决连接问题的提示。

注意，这里谈到的问题并不总是会出现。它主要取决于网络配置，即取决于 MySQL 服务器以外的一个因素。

1. 是否给主机名加上域名

当在 *user* 数据表中插入一个新的条目的时候，出现的问题是主机名应该和域名一同出现还是不需要域名，例如，应该是 *uranus* 还是 *uranus.sol*。哪种方法正确要取决于服务器是如何解析网络名字的。

笔者发现如果包含域名，成功的可能性很高。（网络名字的解析是依据文件/etc/hosts，以及运行在本地网络中的域名服务器配置，因此说是依据 MySQL 服务器之外的因素。）

依据网络设置，在安装 MySQL 服务器的过程中，可能会出现数据列 *user.Host* 没有域名而进入的情况。这是由于脚本程序 mysql_install_db 造成的。如果在建立连接过程中，MySQL 检查全部的网络名字，那么在 TCP/IP 协议下的本地连接是不可能的。问题在于，mysql -u root 在起作用，但 mysql -u root -h *computernname* 却不起作用。

在这种情况下，必须要明确地改变域名。必要的命令如下所示（当然，需要将 *uranus* 和 *uranus.sol* 改变成应用于实际情况中的名字）：

```
root# mysql -u root
mysql> USE mysql;
mysql> UPDATE user SET host="uranus.sol"
WHERE host="uranus";
mysql> FLUSH PRIVILEGES;
```

2. UNIX/Linux下的*computernname*与*localhost*

当 MySQL 服务器在 UNIX/Linux 下运行的时候，对于本地对服务器的访问，默认的安全设置提供了两个条目：*localhost* 和 *computernname*（如表 11-4 所示）。这样做的原因在于，客户程序可以用两种方式和运行在同一台计算机上的 MySQL 服务器连接：通过套接字文件或通过 TCP/IP 网络协议。

如果在 MySQL 服务器所运行的同一台计算机上加载程序 mysql，可以测试这两个连接选项：没有给出主机名或是以 *localhost* 作为主机名，通过套接字文件连接成功（下面的第一个和第二个命令）。如果使用选项--protocol=socket（用法 3），也能够得到同样的结果。

从另外一个方面来说，如果给出了主机名或是 IP 地址，连接就要通过 TCP/IP 协议进行（第四个到第八个命令）。可以使用命令 status 来决定实际的连接是如何完成的：

```
linux:~ $ mysql      -u root -p      via socket file (1)
linux:~ $ mysql -h localhost  -u root -p      (2)
linux:~ $ mysql -h localhost  -u root -p --protocol=socket (3)

linux:~ $ mysql -h uranus      -u root -p      via TCP/IP (4)
linux:~ $ mysql -h uranus.sol -u root -p      (5)
linux:~ $ mysql -h 127.0.0.1  -u root -p      (6)
linux:~ $ mysql -h 192.168.0.2 -u root -p      (7)
linux:~ $ mysql -h uranus      -u root -p --protocol=tcp      (8)
Welcome to the MySQL monitor. ...
mysql> status
...
Connection:          uranus via TCP/IP
TCP port:            3306
```

3. 使用Red Hat Linux和Fedora Core时的*localhost*问题

使用 Red Hat Linux 和 Fedora Core 时，在极少数情况下，必须把 *localhost.localdomain* 输入 *user* 数据表的 *Host* 数据列才能建立本地 TCP/IP 连接（而不是其他情况下的计算机名）。原因在于，使用 Red Hat 的时候，IP 地址 127.0.0.1 是链接到 *localhost.localdomain*（文件/etc/hosts）的。

4. 使用SUSE时的*localhost*问题

使用 SUSE 的时候，极少的情况下也有可能出现 *localhost* 的问题。原因在于，网络配置执行完以后，/etc/hosts 文件里将增加一行 127.0.0.2 *computernname* 语句。如果 *computername* 关联着一个不同的 IP 地址，这一行语句就有可能引起混淆。这个问题的出现通常只是在要进行本地 TCP/IP 连接的时候。有以下几个方法可以解决这个问题：

- 首先做出的努力是，应该尽可能地在 *Host* 数据表中给出完整的计算机名（包括域名）。至少是

在笔者的系统里，这么做已经足以建立一个本地 TCP/IP 连接——不管/etc/hosts 文件里有没有一行 127.0.0.2 ...语句。

- 如果这样做还不行，可以不使用计算机名，而在 Host 数据列中进入到 IP 地址 127.0.0.2。通常这样做就可以了，但它实际上只是减缓了问题的症状，并没有真正解决问题。
- 第三种方法是在 Host 数据表中给出完整的计算机名并且对/etc/hosts 中的 127.0.0.2 语句加以注释（加一个#符号作为前缀）。然后重新启动网络和 MySQL 服务器，使做出的改变生效，MySQL 服务器对名字解析将不再使用临时存储：

```
root# /etc/init.d/network restart
root# /etc/init.d/mysql stop
```

这样做的缺点是有可能与其他基于SUSE标准配置的插件产生兼容性问题。

5. 建立UNIX/Linux下的主机名和IP地址

可以使用命令 hostname 来查看当前的计算机名，使用 hostname -f 选项，可以获得完整的计算机名，包括域名。如果把这个计算机名用做 host 命令的参数，它将返回完整的计算机名（包括域名）和关联的 IP 地址。反之，它将返回给定 IP 地址处的计算机名：

```
linux:~ $ hostname
uranus
linux:~ $ hostname -f
uranus.sol
linux:~ $ host uranus.sol
uranus.sol. has address 192.168.0.2
linux:~ $ host 192.168.0.2
2.0.168.192.in-addr.arpa. domain name pointer uranus.sol.
```

不使用命令 host，也可以使用命令 resolveip，它同 MySQL 一同安装。在解析网络名字和 IP 地址的时候，MySQL 服务器使用与 resolveip 相同的算法。如果命令 resolveip 和 host 给出了不同的结果，说明配置有问题。最有可能出错的地方就是文件/etc/hosts 或是域名服务器（DNS）有错误配置，如果它们中的一个应用在了本地网络中：

```
linux:~ $ resolveip uranus
IP address of uranus is 192.168.0.2

linux:~ $ resolveip uranus.sol
IP address of uranus is 192.168.0.2

linux:~ $ resolveip 192.168.0.2
Host name of 192.168.0.2 is uranus.sol
```

提示 在默认的情况下，MySQL服务器会把最近用到的IP地址和主机名保存在一个缓存区里（目的是使重复的访问尽可能快）。如果改变了网络配置，就必须清理这个缓存区：root#mysqladmin flush-hosts。

6. 使用主机名还是IP地址

原则上说，可以在 Host 数据列里给出一个 IP 地址来代替计算机的主机名。IP 地址的优点在于在名字解析中的许多出错原因不再存在。但是，这个优点是以它的缺点作为代价的，在许多网络中，IP 地址经常变化（在有些情况下，如果 IP 地址是由 DHCP 服务器动态赋予，那么每一次重新启动，IP 地址都要变化）。因此，以 IP 地址为基础的访问控制只是在网络中的 IP 地址管理相对比较严格的时候，才有可能成功。

如果在 *user* 数据表中存在着两个意义相同的条目，区别仅在于一个是 IP 地址而另外一个是关联的计算机名，那么具有计算机名的条目将处于优先地位。

11.3.6 db 数据表和 host 数据表

1. db 数据表

db 数据表（见表 11-6）里包含着允许特定用户读取、编辑和删除一个数据库的信息。这个数据表的功能很容易理解：如果位于计算机 *h* 的用户 *u* 希望通过执行命令 *SELECT* 来访问数据库 *d*，并且该用户不具有全局的 *Select* 权限，那么，MySQL 会仔细检查 *db* 数据表，寻找第一个具有 3 个部分 *u/h/d* 的 *User/Host/Db* 条目。（如同我们在 *user* 数据表中看到的一样，唯一化标识条目优于使用通配符的条目。这些条目要区分字母的大小写。）如果找到了一个匹配条目，那么所要检查的就是数据列 *Select_priv* 是否包含着 Y 值。

表 11-6 *db* 数据表一览表

字 段	类 型	NULL	默认值
<i>Host</i>	char(60)	No	—
<i>Db</i>	char(64)	No	—
<i>User</i>	char(16)	No	—
<i>Select_priv</i>	enum('N', 'Y')	No	N
<i>Insert_priv</i>	enum('N', 'Y')	No	N
<i>Update_priv</i>	enum('N', 'Y')	No	N
<i>Delete_priv</i>	enum('N', 'Y')	No	N
<i>Create_priv</i>	enum('N', 'Y')	No	N
<i>Drop_priv</i>	enum('N', 'Y')	No	N
<i>Grant_priv</i>	enum('N', 'Y')	No	N
<i>References_priv</i>	enum('N', 'Y')	No	N
<i>Index_priv</i>	enum('N', 'Y')	No	N
<i>Alter_priv</i>	enum('N', 'Y')	No	N
<i>Create_tmp_table_priv</i>	enum('N', 'Y')	No	N
<i>Lock_tables_priv</i>	enum('N', 'Y')	No	N
<i>Create_view_priv</i>	enum('N', 'Y')	No	N
<i>Show_create_priv</i>	enum('N', 'Y')	No	N
<i>Create_routine_priv</i>	enum('N', 'Y')	No	N
<i>Alter_routine_priv</i>	enum('N', 'Y')	No	N
<i>Execute_priv</i>	enum('N', 'Y')	No	N

为了安全控制能够快速执行，MySQL 在 RAM 中预先对各种 *mysql* 数据表进行分类。但是这样做的结果是，只有在 MySQL 使用 *FLUSH PRIVILEGES* 或 *mysqladmin reload* 重新读取这些数据表时，对这些数据表的直接改变才成为动态。

实际上，与 *user* 数据表一样的规则也适用于 *db* 数据表的 *User* 和 *Host* 设置中。唯一的特例在 *Host* 数据列：如果它保持为空，那么 MySQL 还要检查 *host* 数据表（参见下文的“*host* 数据表”）。

2. 默认设置

安装了 MySQL 以后，所有用户（能够访问 MySQL 的用户）都允许建立、编辑和删除数据库 *test*，

以及数据库名是以 *test_* 开始的所有数据库。(在 Windows 环境下，数据库名不需要下划线。) 这样做的目的在于允许对 MySQL 进行实际测试，同时系统管理员又不受为必须要建立这样的数据库而感到的困扰。

但这样的设置有一个明显的问题，任何访问 MySQL 的用户都有权创建一个数据库，并且，只要计算机的硬盘驱动不满，就可以不断地往数据库里添加数据。

表 11-7 给出了 *db* 数据表的默认设置。特别注意，对于 *test* 数据库，除了 *Grant*、*Alter_routine* 以及 *Execute_priv* 权限以外，设置了所有的其他权限。*Grant='n'* 非常重要，使得 *test* 数据库的权限不能够转移到其他数据库。

表 11-7 *db* 数据表的默认设置

Host	Db	User	Select_priv, Insert_priv, etc.	Grant_priv, Alter_routine, Execute_priv
%	test		Y	N
%	test_%		Y	N

注解 已经理解了 MySQL 的权限系统了吗？那么看看能否回答下面的问题：如果一个 MySQL 用户的全局权限全都被设置为 N，该用户还能创建并编辑一个名为 *test* 的数据库吗？

正确的答案是可以。如果所有的全局权限都设置为 N，那么就要检查特定对象的权限（每一个名字以 *test* 开头的数据库都被认为是一个对象）。这是 MySQL 权限系统中的一个明确概念。

3. *host* 数据表

host 数据表（如表 11-8 所示）是对 *db* 数据表的一个扩展，此时 *db* 数据表的 *Host* 字段为空。在这种情况下，讨论中的数据库条目就要在 *host* 数据表里寻求。如果在这里发现了与计算机名相匹配的条目，*db* 数据表中的权限设置和 *host* 组和成逻辑 AND（即，权限要授予两个数据表）。

host 数据表极少用到。（一般来说，*db* 数据表里的设置已足以满足所有的要求）。而且事实上命令 *GRANT* 和 *REVOKE*（下面将要介绍）对 *host* 数据表不起作用。因此，*host* 数据表的默认设置为空。

表 11-8 *host* 数据表一览表

字 段	类 型	NULL	默 认 值
<i>Host</i>	char(60)	No	—
<i>Db</i>	char(64)	No	—
<i>Select_priv</i>	enum('N', 'Y')	No	N
<i>Insert_priv</i>	enum('N', 'Y')	No	N
...			
<i>Show_create_priv</i>	enum('N', 'Y')	No	N

11.3.7 *tables_priv* 和 *columns_priv* 数据表

使用数据表 *tables_priv*（如表 11-9 所示）和 *columns_priv*（见表 11-10），可以对单个数据表和数据列设置权限。*user* 数据表的规则也一样适用于 *Host* 和 *User*。从另外一个方面来看，在 *Db*、*Table_name* 以及 *Column_name* 字段，不允许使用通配符。除了 *Column_name* 字段以外，所有的字段都区分字母的大小写。

表 11-9 *tables_priv* 数据表一览表

字 段	类 型	NULL	默 认 值
<i>Host</i>	char(60)	No	
<i>Db</i>	char(64)	No	
<i>User</i>	char(16)	No	
<i>Table_name</i>	char(64)	No	
<i>Table_priv</i>	set1	No	
<i>Column_priv</i>	set2	No	
<i>Timestamp</i>	timestamp	Yes	
<i>Grantor</i>	char(77)		

表 11-10 *columns_priv* 数据表一览表

字 段	类 型	NULL	默 认 值
<i>Host</i>	char(60)	No	
<i>Db</i>	char(64)	No	
<i>User</i>	char(16)	No	
<i>Table_name</i>	char(64)	No	
<i>Column_name</i>	char(64)	No	
<i>Column_priv</i>	set2	No	
<i>Timestamp</i>	timestamp	Yes	
<i>Grantor</i>	char(77)		

MySQL 在线文档没有对 *Grantor* 数据列进行任何说明，这个数据列里存放着关于权限授予者的信息（例如，*root@localhost*）。

与前面介绍的数据表不同，这两个数据表使用了两个集合来管理它们的权限。（集合是 MySQL 的特色功能之一，参见第 8 章。集合里的字符串元素可以任意组合使用。）数据表 *tables_priv* 和 *columns_priv* 所使用的两个集合如下所示：

```
set1: SET('Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop',
          'Grant', 'References', 'Index', 'Alter')
set2: SET('Select', 'Insert', 'Update', 'References')
```

在这里，*set2* 只包含着那些对单个数据列有用的权限。（不存在 *Delete* 权限，因为只能删除整个的数据记录，而不能删除一个记录的某个字段。当然，可以把这个字段设置为 *NULL*、0 或是 “”（这是两个单引号），但这已经属于 *Update* 权限的管辖范围了。）

很遗憾地告诉大家，MySQL 在线文档没有说明何时该用数据表 *tables_priv* 中的 *Table_priv* 字段、何时该用数据表 *columns_priv* 中的 *Column_priv* 字段。有人在分析了 SQL 命令 *GRANT* 让这两个字段发生的变化之后做出了这样的推论：

- 与数据表相关的权限储存在字段 *Table_priv* 中。
- 与单个数据列相关的权限储存在字段 *Column_priv* 中，同时还储存在数据表 *tables_priv* 以及 *columns_priv* 中。数据表 *tables_priv* 在一定程度上说是所有权限的混合体，对于它的其他详细说明是在数据表 *columns_priv* 中（或许要分成几个数据记录）。这样做的原因是为了简化对两个数据表的检查或是为了提高 MySQL 的速度。

如果可能，要使用命令 *GRANT* 和 *REVOKE* 来设置数据列的权限。

11.3.8 procs_priv 数据表

从 MySQL 5.0.3 版本开始，可以使用 *procs_priv* 数据表（如表 11-11 所示）。它控制着谁可以执行哪一个存储过程。这个数据表只与那些在全局级别上（*user* 数据表）或是数据表级别上（*db* 数据表）没有权利执行和改变存储过程的用户有关。这时，*procs_priv* 数据表为用户提供了可以执行或改变特定存储过程的可能性。

表 11-11 *procs_priv* 数据表一览表

字 段	类 型	NULL	默 认 值
<i>Host</i>	char(60)	No	
<i>Db</i>	char(64)	No	
<i>User</i>	char(16)	No	
<i>Routine_name</i>	char(64)	No	
<i>Proc_priv</i>	set('Execute', 'Alter Routine', 'Grant')	No	
<i>Timestamp</i>	timestamp(14)	Yes	
<i>Grantor</i>	char(77)		

11.4 访问权限的设置工具

可以使用普通的 SQL 命令 *INSERT*、*UPDATE* 和 *DELETE* 来编辑一个数据库中的数据表（当然要假定已有合适的访问权限）。但是，这样做会很劳累，而且容易出错。使用命令 *GRANT* 和 *REVOKE* 会轻松许多，这是本节的中心话题。更方便的方法是使用第 5 章和第 6 章介绍过的图形化用户界面，如 MySQL Administrator 和 phpMyAdmin。

注意 出于速度优化的原因，MySQL 在 RAM 中保持一个 *mysql* 数据表的备份。因此，只有当 MySQL 通过 SQL 命令 *FLUSH PRIVILEGES* 或是外部程序 *mysqladmin reload* 明确重新读取数据表的时候，对数据表的直接改变才生效。（使用 *GRANT* 和 *REVOKE* 命令，这种重新读取自动发生。）

11.4.1 使用 GRANT 和 REVOKE 命令改变访问权限

GRANT 和 *REVOKE* 命令的简单句法格式如下：

```
GRANT privileges
ON [database.]table or database.spname
TO user@host [IDENTIFIED BY 'password']
[WITH GRANT OPTION]
REVOKE privileges
ON [database.]table or database.spname
FROM user@host
```

如果要对一个数据库中的所有数据表的权限做出改变，正确的使用格式是 *ON database.**。如果要改变的是全局权限，那就要定义 **.**。在数据库名中不允许使用通配符。

对于 *user*，可以用 “”（两个单引号）来表示一台特定计算机上的所有用户（例如，“@*computername*”）。而对于 *host*，必须要使用 “%”（例如，*username@%*）。

依据它们各自的功能，这些命令改变了 *mysql* 数据表 *user*、*db*、*tables_priv*、*columns_priv* 以及 *procs_priv*。*(host* 数据表保持不变。)

注意 GRANT 和 REVOKE 命令的完整句法在第 21 章给出。不过，可以从下面的例子中体会到这些命令的实质。

1. 注册新的用户

具有计算机名`*.myorganization.com`的所有用户，只要知道密码 `xxx` 都允许连接到 MySQL。权限 *Usage* 表明所有的全局权限都已经设置为 *N*。因此最初的时候，用户没有任何权限（从一定程度上说，到目前为止，没有一个数据库、数据表以及数据列可以被所有能够注册到 MySQL 的用户进行访问）：

```
GRANT Usage ON *.* TO ''@'%mycompany.com' IDENTIFIED BY 'xxx'
```

下面的命令给了用户 `admin` 在本地计算机上不受限制的权限。所有权限（包括 GRANT）都已得到设置：

```
GRANT All ON *.* TO admin@localhost IDENTIFIED BY 'xxx'  
WITH GRANT OPTION
```

2. 授予访问数据库的权利

下面的命令给了本地计算机上的用户 `peter` 对于数据库 `mylibrary` 的所有数据表进行读取和改变数据的权利。如果 `mysql` 数据库中的 `user` 数据表尚不知道 `peter@localhost`，那么这个名字就被加入而不使用密码。（如果已经有了 `peter@localhost`，则密码保持不变。）

```
GRANT Select, Insert, Update, Delete  
ON mylibrary.* TO peter@localhost
```

如果想给 `peter` 增加锁定数据表、创建临时数据表以及执行存储过程的权限（在许多应用程序中，这是非常有用的），则命令如下所示：

```
GRANT Select, Insert, Update, Delete, Create Temporary Tables,  
Lock Tables, Execute  
ON mylibrary.* TO peter@localhost
```

3. 禁止数据库的改变

下面的命令经用户 `peter` 对数据库 `mylibrary` 进行改变的权利取消掉，但是 `peter` 仍然保留着使用 *SELECT* 命令读取数据库的权利（假定上面例子中的命令刚刚执行过）。

```
REVOKE Insert, Update, Delete  
ON mylibrary.* FROM peter@localhost
```

4. 授予访问数据表的权利

使用下面的命令，本地计算机上的用户 `kahlila` 得到了读取数据库 `mylibrary` 中 `authors` 数据表里的数据的权利（但是不可以改变它）：

```
GRANT Select ON mylibrary.authors TO kahlila@localhost
```

5. 授予访问单个数据列的权利

与用户 `kahlila` 相比较，用户 `katherine` 的访问权限受到很大的限制：她仅获得允许读取数据库 `mylibrary` 中数据表 `book` 里面的 `title` 和 `subtitle` 数据列的权利。

```
GRANT Select(title, subtitle) ON mylibrary.books TO katherine@localhost
```

6. 对所有本地用户授予访问数据库的权利

本地计算机上的所有用户可以读取和编辑 `mp3` 数据库中的数据：

```
GRANT Select, Insert, Delete, Update ON mp3.* TO ''@localhost
```

11.4.2 使用 SHOW GRANT 命令查看访问权限

如果忘记了某个用户都有哪些权限, *SHOW GRANTS* 正是所需要用到的命令:

```
SHOW GRANTS FOR peter@localhost
Grants for peter@localhost:
GRANT SELECT ON mylibrary.* TO 'peter'@'localhost'

SHOW GRANTS FOR testuser@localhost
GRANT USAGE ON *.* TO 'testuser'@'localhost'
IDENTIFIED BY PASSWORD 'xxxxxxxxxxxxxxxxxxxx'
GRANT SELECT, INSERT, UPDATE, DELETE,
CREATE TEMPORARY TABLES, LOCK TABLES
ON 'myforum'.* TO 'ptestuser'@'localhost'
```

11.4.3 使用 mysqladmin 程序改变密码

程序 *mysqladmin* 完成各种各样的管理任务 (参见第 4 章和第 14 章)。尽管这个程序在管理访问权限中不提供任何直接的帮助, 但它可以用于改变密码。这就使得那些没有 *root* 权限的普通用户可以很简单地改变他们的密码。(当然, 必须要知道当前的密码。)

```
> mysqladmin -u peter -p password newPW
Enter password: oldPW
```

上面的命令改变了用户 *peter* 在计算机 *localhost* 上的密码。请注意, 新的密码是作为参数提交的, 而旧的密码是作为请求输入的。(先是新的, 然后是旧的, 这个顺序有些不同寻常。)

如果 MySQL 服务器是运行在 UNIX/Linux 环境下, 而试验用户 *peter* 具有通过 TCP/IP 协议的本地访问权限, 那么还必须要改变对这个计算机的密码。(同前面介绍过的一样, 在 UNIX/Linux 环境下, *mysql.user* 包含着来自本地用户的两个条目, 一个是用于套接字文件的 *hostname = 'localhost'*, 另外一个是为了 TCP/IP 协议的 *hostname = 'computernname'*。)

```
> mysqladmin -u peter -p -h uranus.sol password newPW
Enter password: oldPW
```

从 MySQL 4.1 版本开始, MySQL 服务器使用一个新的算法来对密码进行加密, 在下面的内容当中会介绍有关内容。如果从兼容性方面考虑, 仍然想使用过去的加密方法, 那就要使用 *mysqladmin old-password*:

```
> mysqladmin -u peter -p old-password newPW
Enter password: oldPW
```

11.5 MySQL 4.1 版本开始的安全密码验证

从 MySQL 4.1 版本开始, MySQL 服务器对密码的加密默认使用改进的方法。数据表 *mysql.user* 中的 *password* 数据列的文本长度增加到了 45 个字符 (原来是 16 个字符)。字符数代表着加密代码, 并不是密码本身。在 *password* 数据列中, 密码不是以普通文本储存, 而是以加密的形式储存。

同时, 验证协议也已经改变, 客户程序是通过验证协议将密码传递到 MySQL 服务器的。现在, 这种密码的传输以加密形式完成, 而在过去, 这种传输是以普通的文本进行的。

这两种措施改进了 MySQL 密码验证的安全性。但是, 它们也带来了一些问题: 在 MySQL 服务器升级以后, 众多的 MySQL 应用程序会突然失效。典型的错误信息是: *Client does not support authentication protocol requested by server* (客户程序不支持服务器请求的验证协议)。

本节中, 笔者将就这个问题阐述各种可能性。更多的相关信息可以访问下面的站点: <http://dev.mysql.com/doc/mysql/en/old-client.html>。

11.5.1 升级客户端函数库

对这个问题最好的解决方法是安装一个新的客户端数据库版本。但是推荐的这个方法说起来容易，但实施起来非常困难：对于 Linux 安装版本，当版本发行者没有提供合适的升级时，手动插入 PHP 或 Perl 是一件非常困难的事情，而且极少会出现这种情况。如果不想移植到一个新的发行版本，可以尝试从其他发行版本安装一个插件。但这样做不可避免地要导致 Linux 插件管理中的属地冲突，而要解决这个问题需要大量有关 Linux 的知识。另外一个解决问题的方法是编译这个程序，但这个办法仅适用于 Linux 高手。

下面所列出的内容告诉我们从哪一个版本开始，各种程序设计语言与新的 MySQL 验证兼容。

- C 语言。从版本 4.1 开始，需要 libmysqlclient。
- Perl。在版本 2.9004 或更高版本中，需要模块 perl-DBD-MySQL。
- 带 mysql 接口的 PHP。PHP 5.0.3。
- 带 mysqli 接口的 PHP。PHP 5.0.0。
- Java。必须使用 3.1 或更高版本的 Connector/J。
- ODBC。必须使用 3.51.10 或更高版本的 Connector/ODBC。

更进一步说，客户程序库具有与新的 MySQL 服务器连接的能力（即，它们管理着旧的和新的验证）。因此，升级与兼容性之间是矛盾着的。

11.5.2 *old-passwords* 模式

如果无法对客户库进行升级，可以以 *old-passwords* 模式运行 MySQL 服务器。为此，在 MySQL 配置文件 my.cnf 或 my.ini 中输入 [mysqld] 里的选项 old-passwords。重新启动以后，MySQL 服务器就像 4.0 版本那样运行。

- 函数 *PASSWORD* 使用老算法来加密密码。
- SQL 命令 *GRANT* 使用原算法进行加密。
- MySQL 服务器允许用户使用老验证协议进行登录。

注意：这个选项仅对新密码的定义有效。现有的密码保持不变而且必须要被重新创建。（如果不这样做，尽管是以 *old-passwords* 模式，MySQL 服务器会要求基于新协议的验证。）

11.5.3 同时使用旧的和新的密码的操作

old-passwords 模式有明显的缺陷，即 MySQL 4.1 版本所做出的安全性改进没有发挥作用。在读者的计算机系统上，假如只有一小部分用户依靠旧的验证协议，而其余的用户可以使用其他的程序设计语言和新的验证方法，那么推荐同时使用旧的和新的密码进行操作的方法。这时，不需要在 my.cnf 或 my.ini 中做出改变，而且不允许使用选项 old-passwords。

依据密码的字长，MySQL 服务器要决定是使用旧的还是新的密码验证协议。对于加密编码仅为 16 个字符长度的密码，使用旧的方法，而对更长的密码，使用新的方法。

同时操作的唯一问题是，作为数据库管理员必须要注意到是如何定义密码的。对于依靠老方法的用户，必须要使用函数 *OLD_PASSWORD('secret')* 或是 *mysqladmin old-password* 来定义密码。如果使用的是 SQL 命令 *GRANT*，那就要省去 *IDENTIFIED BY* 部分并手动插入密码。下面给出了如何完成这项工作：

```
GRANT ALL ON databasename.* TO newuser@localhost
UPDATE mysql.user SET password = OLD_PASSWORD('secret')
WHERE user='newuser' AND host='localhost'
FLUSH PRIVILEGES
```

11.6 建立连接的问题

如果遇到了连接问题，不管正在使用的是什么程序设计语言，首先应该使用程序 `mysql` 进行交互式测试，检查是否可以建立连接。如果使用 `mysql` 程序能够连接成功，再去检查问题是否出在正在使用或开发的程序里才有意义。

只有在执行了命令 `FLUSH PRIVILEGES` 以后（或是重新启动 MySQL 服务器以后），对访问权限做出的改变或是对 MySQL 数据库做出的改变才生效。许多用于用户管理的管理对话程序并不自动执行命令 `FLUSH PRIVILEGES`。

所以，如果已经改变了访问权限，但是没有看到改变结果，那就要以 `root` 身份在 `mysql` 中执行 `FLUSH PRIVILEGES` 命令（或者是意义相同的 `mysqladmin flush-privileges`）。

对于现有的连接，尽管执行了命令 `FLUSH PRIVILEGES`，但在新的登录以后，所做出的改变有可能只是部分生效。下面的规则在起作用：

- 只有在新的连接完成以后，改变的全局权限才会生效。
- 只有在执行了命令 `USE table` 以后，改变的数据库权限才会生效。
- 在下一个 SQL 命令中，改变的数据表和数据列权限才会生效。

11.6.1 连接困难的可能原因

下面的内容给出了建立连接的过程中出现问题的典型原因。请注意，一个特定的错误信息可能来自一个或几个不同的原因。

- **MySQL 服务器没有运行。**此时，如果尝试用 `mysql` 程序去建立连接，就会得到错误信息 2002 (*Can't connect to MySQL server on 'hostname'* (不能连接到 `hostname` 上的 MySQL 服务器)) 或 错误信息 2003 (*Can't connect to local MySQL server through socket /var/lib/mysql/mysql.sock* (不能通过套接字文件 `/var/lib/mysql/mysql.sock` 连接到本地 MySQL 服务器))。
在 Windows 环境下，可以利用任务管理器来辨别服务器是否在运行。在 UNIX 环境下，可以使用命令 `ps|grep -i mysql`，应该看到一份进程表（为了提高效率，服务器将它自己分成了许多的进程）。如果没有出现这种结果，就必须启动服务器（在 Linux 系统上执行命令 `/etc/init.d/mysql[d] start`）。
- **客户程序找不到套接字文件。**在 UNIX/Linux 环境下，如果服务器和客户程序是运行在同一台计算机上，连接几乎都是通过套接字文件。要做到这一点，服务器和客户程序使用的就必须是同一个套接字文件。如果遇到问题，应当先去检查配置文件 `/etc/my.cnf` 中的 [client] 选项组里是否有一个 `socket=filename` 设置项，其中 `filename` 必须是套接字文件的真实路径（不能是一个符号链接）。通常情况下，这个文件具有名字 `/var/lib/mysql/mysql.sock`。
- **客户程序不能访问套接字文件 (SELinux)。**许多 Linux 发行版本（如，RHEL4）是以这样的方式配置的，即 Apache 不能够访问 `htdocs` 目录以外的文件。因此，PHP 或 Perl CGI 脚本程序不能使用 MySQL 套接字文件。如果使用就会出现错误信息 2002。解决方法是：改用 TCP/IP 协议进行通信或禁用 Apache 的 SELinux 功能（在 RHEL 发行版本里，对应的配置选项是 `system-config-security`）。

- **客户程序和服务器之间的网络连接中断。**如果读者的程序和 MySQL 服务器分别运行在不同的计算机上，应该首先在客户计算机上执行命令 `ping serverhostname` 来测试是否存在与服务器计算机的连接。否则，必须先去修复网络配置。
- **MySQL 不接受通过网络 (TCP/IP 协议) 的连接。**这可能是因为 `my.cnf` 文件里使用了 `-skip-networking` 选项或是类似的选项。这个设置经常被用来提高 MySQL 的安全性。数据库的连接只能通过本地计算机并仅使用套接字文件来进行。
这个问题通常可以通过错误信息 2003 来识别 (*Can't connect to MySQL server* (不能连接到 MySQL 服务器))。解决办法之一是从 `my.cnf` 文件里删除 `-skip-networking` 选项¹。
- **MySQL 拒绝接受来自正在使用的那台计算机的任何连接请求。**这个问题通常出现在 MySQL 服务器是运行在网络服务供应商的计算机上的时候。此时服务器的配置通常是仅允许本地计算机（或本地网络）的连接。在这种情况下，如果连接 MySQL 服务器的目的是为了对自己的数据库进行管理，就必须创建一个 telnet/ssh 连接，或是使用一个在服务器本地执行、但可以通过网络去控制的工具程序（比如 phpMyAdmin）。
- **主机名的解析不正确。**在通过网络建立一个连接的过程中，出现错误信息 1130 (*Host n.n.n.n' is not allowed to connect to this MySQL server* (主机 n.n.n.n' 不允许连接到这个 MySQL 服务器))。这个错误最可能的原因要么是 `mysql.user` 数据表中的主机名不正确，要么是域名服务器的配置不正确。根据具体的网络配置情况，`user.Host` 数据列里的主机名往往必须带有域名部分（但极少数系统却要求必须没有域名部分）。
解决这个问题的方法是在 `mysql.user` 数据表中添加域名 (*uranus→uranus.sol*)，或者是删除它 (*uranus.sol→uranus*)，然后再试一次。（不要忘记执行 `FLUSH PRIVILEGES` 命令。）
如果问题仍然没有解决，可以使用 `hostname`、`host` 或 `resolveip` 命令来测试在名字解析中是否存在问题是。用 IP 地址来替代 `user.Host` 数据列中的主机名是一个几乎总能奏效的应急方法（但缺点是不够灵活）。我们在本章前面的内容里还介绍了许多可以用来解决主机名问题的其他办法。
- **用户名或密码错误。**警惕打字错误！不仅要检查用户名和密码是否正确，还应该检查主机名是否正确。（有些连接只能在特定的计算机上进行才能成功。）请参考上面“主机名的解析不正确”条目去检查问题是否出在主机名身上。
注意，`user.Password` 数据列里的密码是加密格式而不是普通文本格式。如果想通过 SQL 命令去修改密码，就必须使用函数 `PASSWORD("xxx")`。
- **客户程序使用了一个过时的 MySQL 函数库来验证密码。**产生错误信息 *Client does not support authentication protocol requested by server* (客户程序不支持服务器请求的验证协议) 的原因是试图创建一个与 4.1 或更高版本的 MySQL 服务器的连接时，客户程序只是 4.0 或更低版本。对于 MySQL 4.1 版本，使用的是经过改进的密码加密算法和新的验证协议，它们与旧的协议互相不兼容。解决方法是升级客户程序和它的 MySQL 库或是使用 MySQL 选项 `old-passwords`。前一节当中对这个问题有详细的讨论。
- **在 mysql.user 数据表中使用了错误的条目。**当用户 *x* 试图使用计算机 *y* 注册的时候，MySQL 服务器用一个特定的顺序比较 `user` 数据表中的条目。首先，检查 `Host` 字符串是唯一的条目，然后是使用通配符（% 和 _）的 `Host` 条目。在这两组当中，唯一的 `User` 字符串优于使用通配符的字符串。

1. 原书这里说的“…… 删除 my.cnf……”显然错误。——译者注

这种顺序优先的结果是，本地计算机上 (*localhost*) 的用户 *abc* 在某些场合不能注册，尽管在 *user* 数据表中有条目 *Host='%' / User='abc'*。原因是在访问权限的默认设置中，也有一个条目 *Host='localhost' / User=''*。由于主机名被明确给出，这个条目要优先排在第一位。

解决这个问题的办法有两种：一是在 *User* 数据表中添加一条记录 *Host='localhost' / User='abc'*；二是删除 *Host='localhost' / User=''* 记录。笔者推荐使用第二个办法，因为使用 *user=''* 会带来安全风险。

- **未给出 MySQL 用户名。**如果在使用某个程序去连接 MySQL 服务器的时候没有给出一个用户名，该程序将自动使用当前账户的用户名去连接 MySQL 服务器；对以交互方式启动的程序来说，这就是操作系统登录名。

对于运行在网络服务器上的程序（PHP 或 JSP 脚本程序，Perl CGI 文件等），使用的是网络服务器的账户名。从信息安全的角度考虑，网络服务器通常不以 *root* (Linux) 或是 *Administrator* (Windows) 权限来运行，而是以另一个低权限账户如 *wwwrun* 或 *apache* 来运行。现在的问题是，*mysql* 数据库里的各有关数据表并不知道用户 *wwwrun* 或 *apache*，所以会拒绝它们对数据库的访问。因此，千万不要忘记在脚本程序文件中明确指定连接的用户名。

- **连接成功，但是不能够访问数据库。**这种错误的现象是：在建立连接的过程中，只要指定了想访问的数据库，这个错误立刻就出现。这种错误还经常发生在选择目标数据库的时候 (*USE dbname*)。与此相伴的出错消息通常是 1045，例如，*Access denied for user ... to database ...* (拒绝某用户访问某个数据库)。

出现这个错误最有可能的原因是，该用户实际上不具备对这个数据库的访问权限。在 *GRANT* 命令中，大概只是给出了 *Usage* 权限（这个权限允许注册，但不允许实际使用数据库）。命令 *GRANT SELECT, INSERT ... ON dbname.* TO name@hostname* 将使 *name* 用户可以从 *hostname* 主机去访问 *dbname* 数据库。

如果 MySQL 运行在 ISP (网络服务供应商) 的计算机上，那么服务器通常是按照只是允许本地访问来配置的。换句话说，PHP 或 Perl 脚本程序可以运行（因为它们是在同一台计算机上执行），但是，不能够从另外一个点（比如家里）使用 MySQL Administrator 去访问数据库。

这种问题是因为正确的（安全的）访问权限设置而引起的。几乎找不到有哪一个 ISP 会允许从网络上的任何计算机都可以连接到 MySQL 上。因此，必须使用一种能够在 ISP 计算机上以本地方式运行的管理程序（如 phpMyAdmin）。

- **无法创建本地 TCP/IP 连接。**这个问题通常是在 UNIX/Linux 环境下。只有使用选项 -h 而不给出计算机名或 IP 地址时，可成功进行本地连接。对此，最可能的原因是主机名的解析出现了问题。按照规则，必须在数据列 *user.Host* 中添加域名，然后问题应该得到解决。另外一个可能原因是本地网络的配置问题（文件/etc/hosts）。本章中已经给出了一些方法，特别是对于 Red Hat Fedora 和 SUSE Linux。

- **本地连接对于 Java 程序失败。**这个问题通常与前面谈到的问题有关系，与其他大部分 MySQL 客户程序相比较，Java 程序经常使用 TCP/IP 协议（而不是套接字文件）。参见第 17 章，那里有一些诊断问题的方法和技巧。

另外一个错误原因可能是 Connector/J 的安装不正确（参见第 17 章），但是错误出现在试图使用 JDBC 的时候 (*java.lang.ClassNotFoundException: com.mysql.jdbc.Driver*)。

- **端口 3306 被阻断。**在 MySQL 服务器和程序之间，防火墙阻隔了端口 3306。只有在程序和 MySQL 是运行在不同的计算机上的时候，才会出现这个问题。如果是自己管理着防火墙，必

须清除端口 3306；否则，必须请求管理员这样做。

- **升级 MySQL 服务器引发的问题。**在过去升级 MySQL（从 3.23 版本到 4.0、4.1 再到 5.0）时，安全系统的 *mysql* 数据表要扩大许多。如果完成了一个 MySQL 服务器的升级，必须要确定 *mysql* 数据表要与新的安全系统一致。为了实现这一点，提供了脚本程序 *mysql_fix_privilege_tables*（参见第 14 章）。这个脚本程序在 *mysql* 数据表中创建全部新的数据列，但对这些数据列保持默认设置为 *N*。需要仔细查看新的访问权限，并明确授予某些新的权限。我们已经在前面介绍过从 4.0 版本升级到 4.1 或更高版本的时候可能引发的验证问题。

提示 其他的错误原因分析以及减少错误的方法可以参见 MySQL 在线文档：

<http://dev.mysql.com/doc/mysql/en/access-denied.html>.

11.6.2 错误检查的更多方法

如果确定 *mysql* 数据库配置正确，但对数据库的访问仍然失败，那么停止服务器的运行，在 *my.cnf* 或 *my.ini* 中的 [mysql] 部分里暂时添加 *skip-grant-tables*，并重新启动服务器。现在，每一个人都可以访问所有数据。如果访问成功，至少可以确定问题是出在 *mysql* 数据表中。不要忘记从配置文件中删除 *skip-grant-tables*。

MySQL 会把一份最近使用过的 IP 地址和相应主机名的名单放在一个临时内存块（缓存区）里。缓存区具有这样的功能：比较消耗时间的名字解析过程只在第一次访问时发生一次。但是，如果改变了网络配置而没有重新启动 MySQL 服务器，就可能出现这种情况：这个缓存区包含着错误的（不再有效的）信息。解决方法是在不启用这个缓存区的前提下启动 MySQL 服务器：先在 *my.cnf* 或 *my.ini* 文件中的 [mysql] 选项组里添加 *skip-host-cache* 选项，然后重新启动服务器。

很遗憾，没有一种方法可以让 MySQL 服务器把登录失败的准确原因保存下来。在出错日志里（这个日志的默认文件名是 *hostname.err*，参见第 14 章）只有客户机的 IP 地址，没有可以方便人们查出问题根源的信息（例如，关于域名解析过程的信息、本次登录涉及了 *mysql* 数据库里各有关数据表的哪些记录项等）。

11.7 系统安全性

本章从 MySQL 用户管理和访问权限管理的角度探讨了系统安全的问题。在实际工作中，如果想保护好存放在 MySQL 数据库里的数据，必须先把数据库服务器和它底层的操作系统保护好。

11.7.1 系统级安全措施

为了保证 MySQL 数据库系统安全，应该先问自己几个问题：

- 有没有让攻击者可以窃取到 *root* (UNIX/Linux) 或 *Administrator* (Windows 2000/XP) 权限的安全漏洞？为尽可能减少这种危险，应该尽量保证操作系统的安全、定期进行安防升级、安装防火墙等。（不过，系统安全问题不在本书的讨论范围内。）
- 日志文件安全吗？有时候，日志文件里会有明文形式的密码。
- 包含着明文密码的脚本文件够安全吗？（如果不能百分之百地保证脚本文件的安全，它里面的用户名/密码组合是不是只有最小的访问权限？那个密码是不是只能用来登录 MySQL 服务器而不是为了方便而使用了与 *root* 用户相同的密码？）

11.7.2 重要数据存储的安全保护

另外一个问题涉及数据库中的数据。应该做好最坏的打算，即攻击者盗取数据的可能性，为此，要对数据库中的关键资料加密。

- 如果在数据库中储存着注册信息（例如，Web 站点上的用户注册信息），那就不要以普通文本储存密码，而是要以 MD5 校验和的形式储存。这些校验和足以查证密码。（这样做的缺点是不能够从校验和中恢复忘记的密码。）
- 如果甚至还储存着更多的敏感信息（极端的例子是信用卡信息），必须要对这样的数据进行加密。应该考虑 MySQL 对于管理这样的数据是否是一个合适的数据库系统。如果储存了这样的关键数据，应该咨询安全性方面的专家，尽可能地保证安全。任何失误都会造成巨大的错误。

11.7.3 与 MySQL 服务器有关的安全风险

MySQL 服务器在安全性方面本身就存在着危险。由于 MySQL 服务器代码错误在不断地被发现和更正，因此绝对有必要定期升级到最新的 MySQL 版本。（对于许多 Linux 发行版本，可以通过升级系统自动进行。）

例如，在 2005 年年初，在用户定义函数（user-defined function，UDF）里发现了一些安全漏洞。由于大量基于 Windows 的 MySQL 服务器没有使用密码或使用了一个很容易猜到的密码，这些漏洞可以被蠕虫（一种病毒）所利用。从那时起，UDF 的使用发生了变化来防止这样的错误使用。更多资料请访问站点：http://dev.mysql.com/tech-resources/articles/security_alert.html 和 <http://dev.mysql.com/doc/mysql/en/udf-security.html>。

还要注意，具有 *File* 权限的 MySQL 用户可以读取和改变本地文件系统中的文件。这种能力的扩展依据文件系统本身是如何受到保护的（对目录和文件的访问权限）以及 MySQL 服务器（亦即 mysqld）是在什么账户下运行的。

11.7.4 不要使用 root 或 Administrator 权限来运行 MySQL 服务器

在任何情况下，MySQL 都不应该在 UNIX/Linux 下以 *root* 身份运行或是在 Windows 下以管理员权限运行。否则，任何一个具有 *File* 权限的 MySQL 用户都可以操纵整个文件系统。

Linux：对于所有的 Linux 当前版本，已经正确配置了 MySQL。可以使用下面的命令进行检查：

```
root# ps au | grep mysqld
root  ... /bin/sh /usr/bin/mysqld_safe ...
mysql  ... /usr/sbin/mysqld ...
mysql  ... /usr/sbin/mysqld ...
...
...
```

这说明脚本程序 mysqld_safe 由 *root* 执行，而 MySQL 服务器的各种线程由用户 *mysql* 执行。（脚本程序 mysqld_safe 必须由 *root* 执行。它只用于启动 mysqld，并且要运行到 MySQL 服务器被明确关闭。如果服务器崩溃，这个脚本程序会自动重新启动 mysqld。）

如果 mysqld 是作为 *root* 执行的，下面/etc/my.cnf 中的条目会有所帮助：

```
# file /etc/my.cnf
[mysqld]
user = mysql
```

Windows：非常不幸的是，在 Windows 环境下，MySQL 服务器默认是在 *System* 账户中运行，因此具有不加限制的权限。而且，取得安全配置十分困难，这需要创建一个新的用户，改变对全部 MySQL

目录的所有访问权限等。对如何进行这项工作的详细介绍请参阅前面提到过的站点里的文章中 http://dev.mysql.com/tech-resources/articles/security_alert.html (参见文章中的步骤 9)。

11.7.5 网络安全与防火墙

在 UNIX/Linux 环境下, MySQL 服务器经常是通过套接字文件专门服务于本地程序, 比如只允许本地网络服务器上的 PHP 脚本程序或本地管理程序去访问数据库服务器。在这类场合, 使用 /etc/my.cnf 文件 ([mysqld] 选项组) 里的 skip-networking 选项彻底禁止 MySQL 服务器的网络功能会更加安全。

注意, 即使安装 MySQL 的目的只是为了建立一个仅供本地网络使用的数据库服务器, 也不能使用 skip-networking 选项。不过, 至少应该使用 IP 数据包过滤器 (或防火墙) 阻断 3306 号 IP 端口。MySQL 服务器默认使用这个端口进行 TCP/IP 通信。关闭这个端口以后, 从本地网络以外的地方就不能访问 MySQL 服务器。

尽管采取了各种安全措施, 但通过动态 Web 站点的网络访问仍然会发生, 因为动态 Web 站点是从本地运行的 Web 服务器(更准确地说, 是由 Web 服务器调用一个脚本解释器)生成的。因此, MySQL 将把那些脚本视为本地程序, 外部通信不是通过 3306 号端口而是通过 HTTP 协议(一般是 80 号端口)来进行的。如果那里有一个安全漏洞, 那么即使关闭了 3306 号端口也不会有任何帮助。

第 12 章**GIS 函数**

人们把用来处理空间坐标数据的软件包统称为 GIS 系统 (Geographic Information System, 地理信息系统)。通常,此类程序包含在可以直接受理和处理地理数据的数据库系统中。MySQL 从 4.1 版本开始增加了这类函数。

这些函数不仅受到那些希望实现完整 GIS 功能的人的喜爱,而且那些需要处理地理数据的人们对此也很感兴趣。即使没有对地理数据处理的专业要求,也可以从中学到很多有关空间问题的各种数据的知识。例如,数码相机中的照片,每张照片都是在某个特定地点拍摄的。为什么不把这些照片按照地理坐标分类呢?这样,就可以进行这样的查询:把我在离家十公里范围内拍的照片显示出来。

与一些商业化产品(如 Oracle Spatial 和 DB2 Spatial 等)相比,MySQL 目前实现的几何函数还不够多,也不够好。而且,MySQL 中的 GIS 实现目前还只能对二维矢量数据进行处理。尽管如此,MySQL 的 GIS 函数还是能够胜任很多工作,它们开启了一道通往崭新领域的大门,意义重大。

注解 本章中的一部分为 Bernd Öggel 所著,并收入我们合作撰写的 *PHP 5 and MySQL 5* 一书。由于他在地理学方面的高深造诣,以及作为一名 GIS 系统与数据库的资深讲师,他在 GIS 领域的知识与经验远非我等所及。同时也非常感谢他允许笔者在此书中引用他的研究成果。

如果想要查询更多关于 MySQL GIS 函数的资料,登录网站 <http://dev.mysql.com/doc/mysql/en/spatial-extensions-in-mysql.html>。另外可以在网站 <http://dev.mysql.com/tech-resources/articles/4.1/gis-with-mysql.html> 中查询一些在非地理学应用方面使用地理数据的观点和理念。

12.1 GIS 数据格式

在开始详细描述 MySQL 的 GIS 函数之前,有必要了解一下现行的地理数据标准。本章总结了一些常见的概念及数据格式。这些信息在导入或导出地理信息时都是必不可少的。

12.1.1 地理坐标的表示方法

读者们很可能见到过“N47 16 06.6 E11 23 35.9”形式的坐标。这种坐标用度、分、秒来表示一个地理位置。在上例中,坐标可以读做“北纬 47 度 16 分 6.6 秒,东经 11 度 23 分 35.9 秒”。这种数据就是地理坐标。这种标准的缺陷在于,对于习惯了十进制规律的人们来说,计算起来有些不方便。而且由于地球是一个可以自转的椭圆体,所以计算距离时还必须考虑地球表面的曲率问题。

为了解决这些问题,在实际应用中使用投影坐标,即把整个椭圆体按照划分好的等距的直线距离

切割成很多长条状部分来计算。近年来，大量的投影坐标系统被应用到实际工作中，很多国家各自为政，根据自己国家的实际情况开发了针对本国的投影坐标系统。

UTM 格式。统一化格式，缩写为 UTM。如果 UTM 成为地理坐标的标准格式，将会规范投影坐标的格式，解决诸多投影坐标系统共存的混乱局面。它把每 6 度经度设为一个单位，把地球分成了 60 个这样的长条形单位。纬度上它选取了北纬 84° 至南纬 80° 的区间为有效区间。在每个单位中，坐标由向右和向上两个数值组成，单位为 m（米）。

所有的 UTM 格式坐标都对应于一个特殊的基准面（datum）。基准面记载着使用了哪种椭圆体的模型，以及与地心相关的原点应当设置在哪里。

可以从因特网免费下载相关程序及函数库来把传统坐标数据转化为 UTM 坐标数据。下面这个网站是个很好的出发点：<http://www.remotesensing.org/>。

示例。下面几行数据是奥地利 Tirol 城中几处地点的 UTM 坐标：

```
681547.32, 5237595.88, Innsbruck
680397.55, 5233845.59, Mutters
685271.40, 5239558.18, Rum
685387.69, 5235857.76, Aldrans
679141.11, 5233807.33, Natters
```

想看懂这些数字，就必须知道这里使用的是哪一种坐标系统。UTM 系统按照向右及向上的顺序记录数据，单位为 m（米）。因此从上例得知：Innsbruck 比 Mutters “高”（就是更向“北”）3750m，“右”（就是向“东”）约 1150m。Tirol 城在 UTM 坐标系统里属于第 32 区，以上数据可以从 UTM 32 区第 WGS84 号地图中查到。

12.1.2 Well-Known Text 和 Well-Known Binary (OpenGIS)

无论是描述一条线段还是一个多边形，都需要使用相应的语法规则。过去曾开发了大量的格式，目前具有开放标准的 Well-Known Text (WKT) 和 Well-Known Binary (WKB) 正在试图取代其他的格式。MySQL 直接支持这两种格式。

1. OGC组织

这两种格式都是 OGC 组织推出的。OGC 是“Open Geospatial Consortium”或“Open GIS Consortium”的缩写（“开放地球空间论坛”或“开放 GIS 论坛”），这是一个由商业机构、科研机构、院校以及公共机构组成的联合体，在它的 266 个成员当中有很多著名的公司、大学及非政府组织。

1997 年，OGC 发布了 *OpenGIS Simple Features Specifications for SQL*（用于 SQL 的 OpenGIS 简要特征说明），来介绍地理数据是如何在关系数据库系统中储存及管理的。这份说明包括了数据类型、文本格式及代表几何数据的二进制格式，除此之外，还有大量用于处理几何数据、完成运算、进行几何分析的函数。更多详细内容请浏览网站 <http://www.opengis.org/docs/99-049.pdf>。

2. WKT和WKB格式

WKT 是几何对象的一种文本表示格式。上边例子中的 Innsbruck 的 UTM 坐标用 WKT 格式表述如下：

```
POINT(681547.32 5237595.88)
```

注意，每对坐标是由空格分开。包含两个或更多点的线段由逗号分开：

```
LINESTRING(681547.32 5237595.88, 685387.69 5235857.76)
```

表 12-1 概述了 WKT 的要素。

表 12-1 各种几何对象的 Well-Known Text 表达方法

WKT 示例	含 义
<code>POINT(1 1)</code>	坐标 $x=1, y=1$ 的点
<code>LINESTRING(0 0, 1 1, 3 3)</code>	两个直线段
<code>POLYGON((0 0, 5 0, 5 5, 0 5, 0 0))</code>	长方形
<code>MULTIPOINT(1 1, 5 5)</code>	两个点的 <i>Multipoint</i>
<code>MULTILINESTRING((0 0, 5 5), (10 10, 20 20, 40 20), (10 10, 2 0))</code>	<i>Multilinestring</i> 元素中的 3 条线段
<code>MULTIPOLYGON(((0 0, 5 0, 5 5, 0 5, 0 0), ((10, 10, 30 30, 30 10, 10 10)))</code>	两个多边形
<code>GEOMETRYCOLLECTION(POINT(100 100), POINT(10 10), LINESTRING(1 1, 100 1, 100 100))</code>	由两个点和一个线段构成的一个几何对象集合

读者可能注意到了在定义多边形时的坐标数据使用了双层括号。这并不是印刷错误。将在后面的章节中看到，一个多边形的内部可以包含有空洞。为了定义这种形式，附加的闭合 *LINESTRING* 对象可以在外层的括号里加以说明。比如说下面的多边形在其中间就有一个空洞：

```
POLYGON((1 1, 9 1, 9 9, 1 9, 1 1), (3 3, 3 6, 6 6, 6 3, 3 3))
```

WKT 格式是将数据导入到 MySQL 数据库中的主要方式。

WKB 格式是 WKT 格式的二进制版本。所有的坐标都以双精度数值（64 位浮点数值）给出。MySQL 服务器内部使用 WKB 格式对几何数据进行储存。

12.2 MySQL 的 GIS 实现

从 MySQL 4.1 版本开始，MySQL 具有了 GIS 函数。地理数据目前只能存储在 MyISAM 数据表中，而不能储存到 InnoDB 或其他类型的数据表中。MySQL 的 GIS 应用基于前面提到过的 OpenGIS 标准，但 MySQL 目前尚未实现出这个标准所定义的全部函数。

12.2.1 数据类型

MySQL 中的几何数据类型以 OGC 标准为基础，它们之间的层次关系如图 12-1 所示。

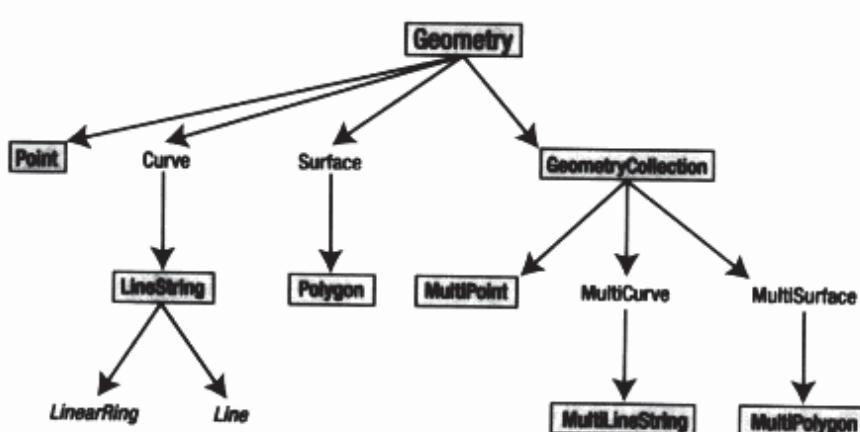


图 12-1 MySQL 几何对象之间的层次关系

图 12-1 中的灰色方框表示可以在 MySQL 中使用的数据格式。*LinearRing* 和 *Line* 虽然在 OGC 标准里得到了定义，但目前还不能在 MySQL 中使用。在实际工作中，因为可以使用 *LineString* 类型，

所以并不会给程序员带来任何限制。*Curve*、*MultiCurve*、*Surface*、*MultiSurface* 都是目前仅有一个衍生子类的抽象类。表 12-2 给出了图 12-1 中各个小方框里对象的类型。

表 12-2 MySQL 中只储存一个对象的几何类型

几何类型	含义
<i>GEOMETRY</i>	几何类型，没有进一步说明
<i>POINT</i>	坐标系统中的一个点；零维空间
<i>LINESTRING</i>	两点之间的一个或多个线段；一维空间
<i>POLYGON</i>	一个闭合的 <i>LINESTRING</i> ；二维空间

在 MySQL 数据表里，点、线和多边形都可以存储在 *Geometry* 类型的数据列当中。OGC 标准在定义这些类时恐怕没有想到它们会被这样使用，但这并不影响它们在 MySQL 里起作用。尽管如此，作为一个原则，还是应该把数据列定义为一种更加具体的几何类型。这也有助于在应用程序的开发过程中，构建一幅有关进行到何种程度的更为简明的画面。

POLYGON 对象可能需要做进一步的解释。由定义得知，一个 MySQL 多边形由一个外部闭合的环和在其内部可能出现的其他环（表示空洞）所组成。这些空洞可能会在某一点相接，但它们不会重叠。需要着重强调的是，多边形内的这些空洞并不是对象表面的一部分。12.3 节给出这方面的示例。

下面的例子显示的是一个数据表中的几个几何类型：

```
CREATE TABLE glacier (
    id      INT NOT NULL PRIMARY KEY,
    border  POLYGON NOT NULL,
    ela     LINESTRING,
    ref     POINT)
ENGINE=MyISAM DEFAULT CHARSET=latin1
```

几何类型可以像标准的 SQL 类型 (*INTEGER*, *VARCHAR* 等) 一样用在 MySQL 的 *CREATE* 命令中。

注解 目前，许多 MySQL 用户界面不支持 GIS。例如，对于 phpMyAdmin2.6，就不能够创建几何数据类型的数据表。当前唯一的特例是 Administrator。可以使用 mysql 命令编译器，该编译器会接受所有遵循 MySQL 语法的 SQL 命令。

MySQL 也提供在一个单元中储存类似对象集合的类型，如表 12-3 所示。

表 12-3 MySQL 中一个单元里储存几个对象的几何类型

几何类型	含义
<i>GEOMETRYCOLLECTION</i>	多个几何对象
<i>MULTIPOINT</i>	一个或多个 <i>POINT</i> 类型
<i>MULTILINESTRING</i>	一个或多个 <i>LINESTRING</i> 类型
<i>MULTIPOLYGON</i>	一个或多个 <i>POLYGON</i> 类型

这些类型符合文档 *OpenGIS Simple Features Specifications for SQL* (用于 SQL 的 OpenGIS 简要特征说明) 中所陈述的 OGC 格式。也有一个描述几何数据的树状结构。简单地说，就是有一个基础类 (*Geometry*) 以及衍生出的点、线和多边形的子类。每一种类型提供有关对象是何种类型的信息查询功能。在 12.2.2 节中将会讨论到这些内容。

12.2.2 简单的几何函数

1. 转换几何格式的函数

MySQL中有几种函数用于在内部几何格式和WKT以及WKB之间进行转换，如表 12-4 所示。

表 12-4 MySQL 几何数据的类型转换函数

函 数	含 义
<i>ASTEXT</i>	返回一个 WKT 几何类型
<i>ASBINARY</i>	返回一个 WKB 几何类型
<i>GEOMFROMTEXT</i>	从 WKT 字符串创建一个内部 MySQL 的 <i>GEOMETRY</i> 格式
<i>GEOMFROMWKB</i>	从 WKB 字符串创建一个内部 MySQL 的 <i>GEOMETRY</i> 格式

例如，要从数据表 *mountain* 中储存一个点，可以使用：

```
INSERT INTO mountain(pt) VALUES(GEOMFROMTEXT('POINT(681547 5237595)'))
```

对每一个数据类型，都还有一个从 WKT 或 WKB 格式的转换函数，所以，显然存在着 *POINTFROMTEXT*、*LINESTRINGFROMTEXT* 及 *POLYGONFROMTEXT* 函数。不过，可以使用 *GEOMFROMTEXT* 函数来得到同样的结果。

□ **SRID**。对于每一个 *xxxFROMTEXT* 和 *xxxFROMWKB* 函数，可以使用一个可选项 SRID 值(spatial reference system identifier，空间基准系统标示符)。该值用于表示几何对象的坐标系统。这使得从理论上说，各种几何对象——其各点是由不同坐标系统来定义的——储存于一个数据表中是可能的。

不过在目前的版本中，MySQL 尚不考虑 SRID 信息，所有的几何运算的完成都是假设使用平面笛卡尔坐标系统。

2. 几何类中的函数

MySQL 中的所有几何对象都支持这些函数。适用的几何对象都可以作为参数传递到这些 SQL 函数里，如表 12-5 所示。

表 12-5 MySQL 中所有几何对象的函数

函 数	含 义
<i>DIMENSION</i>	返回对象的尺寸。可能的结果为： -1 表示没有对象 0 表示点（长度=0，宽度=0） 1 表示直线（长度>0，宽度=0） 2 表示多边形等（长度>0，宽度>0）
<i>ENVELOPE</i>	返回几何对象的外接框。结果具有 <i>POLYGON</i> 数据类型
<i>GEOMETRYTYPE</i>	以字符串形式返回几何对象的类型（ <i>POINT</i> 、 <i>LINESTRING</i> 、 <i>POLYGON</i> 等）
<i>SRID</i>	返回坐标系统的标示符。（MySQL 对每一个几何对象都储存着坐标系统标示符；不过，它不对这个信息求值）

这里所说的“外接框”在几何学里被称为最小边界矩形 (minimal bounding rectangle, MBR)，它的定义为封闭几何外形的最小矩形。对于混合的几何类型，这个矩形包含各种组成对象。在下面的例子中，包含 3 个点坐标的变量 @mp 被赋值给了一个 *Multipoint* 对象。

```
SET @mp = GEOMFROMTEXT('MULTIPOINT(1 1, 10 99, 0 5)')
SELECT ASTEXT(ENVELOPE(@mp))
ASTEXT(ENVELOPE(@mp))
POLYGON((0 1,10 1,10 99,0 99,0 1))
```

ASTEXT 函数规定了 WKT 格式中的外接框：

```
SELECT GEOMETRYTYPE(@mp)
GEOMETRYTYPE(@mp)
MULTIPOINT
```

3. 点对象的函数

不仅是几何类的函数，还有两个其他方法可以应用于点对象：*X(pt)* 和 *Y(pt)*。它们以浮点数值返回点的坐标：

```
SET @pt = GEOMFROMTEXT('POINT(33.2 99.9)')
SELECT X(@pt), Y(@pt)
X(@pt)  Y(@pt)
33.2      99.9
```

4. 线对象的函数

LINESTRING 对象在泛函性方面比点对象要多一些。表 12-6 给出了这些几何函数。

表 12-6 线对象的几何函数

函 数	含 义
<i>GLENGTH</i>	线段长度（以浮点数值表示）
<i>ISCLOSED</i>	如果起始点和结束点相同，则为 1，否则为 0
<i>NUMPOINTS</i>	线段包含着的点的数目
<i>STARTPOINT</i>	第一个点
<i>ENDPOINT</i>	最后一个点
<i>POINTN(g, N)</i>	位置 <i>N</i> 处的点

最常见的应用程序是计算一条几何路线的长度：

```
SET @ls = GEOMFROMTEXT('LINESTRING(2 2, 9 0, 9 9)')
SELECT GLENGTH(@ls)
GLENGTH(@ls)
16.280109889281
```

要想以 WKT 格式规定开始和终止点，一个简单的 *SELECT* 查询命令就足够了：

```
SELECT ASTEXT(STARTPOINT(@ls)), ASTEXT(ENDPOINT(@ls))
ASTEXT(STARTPOINT(@ls))  ASTEXT(ENDPOINT(@ls))
POINT(2 2)              POINT(9 9)
```

相比较而言，只有两个函数可以用于 *MULTILINESTRING* 对象：*GLENGTH* 和 *ISCLOSED*。*GLENGTH* 的结果是所有线段长度之和，而如果所有线段组成一个闭环，则 *ISCLOSED* 返回 *TRUE(1)*。注意，*LINESTRING* 类型的闭合线段并不是多边形。如果使用多边形函数，如 *AREA*，那结果总是 *NULL*。

注解 OpenGIS 标准里的长度函数名为 *LENGTH*，而不是 *GLENGTH*。但 *LENGTH* 函数在 MySQL 中的返回值是某给定字符串的字节个数。

5. 多边形对象的函数

表 12-7 给出了可以用于多边形对象的函数。

表 12-7 多边形对象的几何函数

函 数	含 义
<i>AREA</i>	多边形的面积
<i>EXTERIORRING</i>	<i>LINESTRING</i> 数据类型的外环
<i>INTERIORRINGN(g, N)</i>	<i>LINESTRING</i> 数据类型的内环
<i>NUMINTERIORRINGS</i>	内环数目（空洞）

在计算面积的时候 (*AREA*)，所有多边形中的空洞要被减去。在下面的示例中，四边形@*po* 的空洞大小为 3×3 ：

```
SET @po = POLYFROMTEXT('POLYGON((1 1, 9 1, 9 9, 1 9, 1 1),
(3 3, 3 6, 6 6, 6 3, 3 3))')
SELECT AREA(@po), NUMINTERIORRINGS(@po)
AREA(@po)    NUMINTERIORRINGS(@po)
55           1
SELECT ASTEXT(INTERIORRINGN(@po, 1))
ASTEXT(INTERIORRINGN(@po, 1))
LINESTRING(3 3,3 6,6 6,6 3,3 3)
```

如果 *AREA* 函数应用于一个 *MULTIPOLYGON* 对象，它将返回各多边形的面积之和。

6. *GeometryCollection* 对象的函数

表 12-8 给出了可以用于 *GeometryCollection* 对象的函数。

表 12-8 *GeometryCollection* 对象的几何函数

函 数	含 义
<i>GEOMETRYN(g, N)</i>	返回位置 <i>N</i> 处的几何对象
<i>NUMGEOMETRIES(g)</i>	集合中的对象数目

同一个 *GeometryCollection* 对象可以容纳任意多个几何对象。在使用 *GEOMETRYN* 函数从一个集合里提取出想要的对象后，可以使用各种函数对它做进一步的处理。在下面的例子中，将计算一个 *LineString* 对象的长度：

```
SET @gc = GEOMFROMTEXT('GEOMETRYCOLLECTION(
POINT(10 10), POINT(0 100), POINT(40 40),
LINESTRING(10 0, 10 10, 20 10))')
SELECT GLENGTH(GEOMETRYN(@gc, 4))
GLENGTH(GEOMETRYN(@gc, 4))
20
```

12.2.3 空间分析函数

除了前面描述的几何函数以外，*OpenGIS Simple Features Specifications for SQL*（用于 SQL 的 OpenGIS 简要特征说明）还提供了一系列用于空间分析的函数。当前 MySQL 的 GIS 应用只支持基于几何对象外接框的计算。因此，其他 GIS 系统里的一些常用函数如 *Buffer*、*Distance*、*Intersection*、*Union*、*ConvexHull* 和 *SymDifference* 等都不能在 MySQL 里使用。

表 12-9 给出了常用的分析函数。每一个函数接受两个几何对象作为参数。结果要么是表示 *TRUE* 的 1，要么是表示 *FALSE* 的 0。

表 12-9 外接框函数

函 数	含 义
<i>MBRCONTAINS(G1, G2)</i>	如果 G_2 的外接框完全处于 G_1 的外接框里，则为 <i>TRUE</i>
<i>MBRWITHIN(G1, G2)</i>	如果 G_1 的外接框完全处于 G_2 的外接框里，则为 <i>TRUE</i>
<i>MBREQUAL(G1, G2)</i>	如果两个外接框完全相同，则为 <i>TRUE</i>
<i>MBRINTERSECTS(G1, G2)</i>	如果 G_1 和 G_2 的外接框相接或重叠，则为 <i>TRUE</i>
<i>MBROVERLAPS(G1, G2)</i>	如果 G_1 和 G_2 的外接框重叠，则为 <i>TRUE</i>
<i>MBRTOUCHES(G1, G2)</i>	如果 G_1 和 G_2 的外接框相接，则为 <i>TRUE</i>
<i>MBRDISJOINT(G1, G2)</i>	如果 G_1 和 G_2 的外接框既不相接也不重叠，则为 <i>TRUE</i>

下面的例子中，给出了 3 个长方形@r1、@r2 和 @r3。这 3 个对象的图解表示方法如图 12-2 所示。

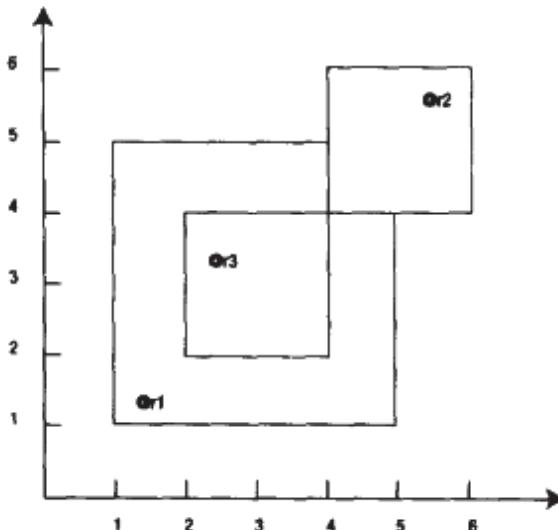


图 12-2 3 个长方形的外接框图解表示方法

以下 SQL 命令可以直接输入到 phpMyAdmin 工具的一个 SQL 文本输入字段里：

```

SET @r1 = GEOMFROMTEXT('POLYGON((1 1, 5 1, 5 5, 1 5, 1 1))')
SET @r2 = GEOMFROMTEXT('POLYGON((4 4, 6 4, 6 6, 4 6, 4 4))')
SET @r3 = GEOMFROMTEXT('POLYGON((2 2, 4 2, 4 4, 2 4, 2 2))')
SELECT MBROVERLAPS(@r1,@r2), MBRINTERSECTS(@r1,@r2)
MBROVERLAPS(@r1,@r2) MBRINTERSECTS(@r1,@r2)
      1           1
SELECT MBRINTERSECTS(@r1,@r3), MBRCONTAINS(@r1,@r3)
MBRINTERSECTS(@r1,@r3) MBRCONTAINS(@r1,@r3)
      1           1
MBRTOUCHES(@r2,@r3), MBROVERLAPS(@r2,@r3)
MBRTOUCHES(@r2,@r3) MBROVERLAPS(@r2,@r3)
      1           0
  
```

在各个对象的精确几何边界中查询关系的函数尚未在 MySQL 中得到应用。OGC 标准介绍了许多 MySQL 中的函数(如表 12-10 所示)，但它们只是用于外接框。在 MySQL 的当前版本中，使用 *CONTAINS* 和 *MBCONTAINS* 函数都是一样的。

表 12-10 几何函数(返回结果仅为 MySQL 中的外接框)

函 数	含 义
<i>CONTAINS(G1, G2)</i>	如果 G_2 的外接框完全处于 G_1 的外接框里，则为 <i>TRUE</i>
<i>WITHIN(G1, G2)</i>	如果 G_1 的外接框完全处于 G_2 的外接框里，则为 <i>TRUE</i>

(续)

函 数	含 义
<i>CROSSES(G1, G2)</i>	<i>G1</i> 和 <i>G2</i> 互不交叉 (参见在线文档)
<i>EQUALS(G1, G2)</i>	如果两个外接框完全相同，则为 <i>TRUE</i>
<i>INTERSECTS(G1, G2)</i>	如果 <i>G1</i> 和 <i>G2</i> 的外接框相接或重叠，则为 <i>TRUE</i>
<i>OVERLAPS(G1, G2)</i>	如果 <i>G1</i> 和 <i>G2</i> 的外接框重叠，则为 <i>TRUE</i>
<i>touches(G1, G2)</i>	如果 <i>G1</i> 和 <i>G2</i> 的外接框相接，则为 <i>TRUE</i>
<i>DISJOINT(G1, G2)</i>	如果 <i>G1</i> 和 <i>G2</i> 的外接框既不相接也不重叠，则为 <i>TRUE</i>

OpenGIS 标准中的 *DISTANCE* 和 *RELATED* 函数在 MySQL 的现有版本里没有实现，将在本章后面的一个例子里演示如何绕过这一问题。

12.2.4 为几何数据创建索引

对于任何的数据库查询，执行速度在很大程度上取决于是否存在索引，空间数据也不例外。MySQL 允许为几何数据创建索引，这种索引也很容易创建：

```
CREATE TABLE mountain (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    pt POINT NOT NULL,
    SPATIAL INDEX(pt) )
```

几何数据列 *pt* 被提供了一个空间索引。不过，实际中也可以在一个现有的数据表上添加索引。要做到这一点，使用具有 *ADD SPATIAL INDEX* 的 *ALTER TABLE* 命令或 *CREATE* 命令：

```
ALTER TABLE mountain ADD SPATIAL INDEX(pt)
CREATE SPATIAL INDEX pt ON mountain(pt)
```

在索引空间数据时，MySQL 使用着 *R-tree* 索引。这里的 *R* 表示区域 (region)。树状结构中描绘的区域包含索引几何对象的外接框或下面较低级别中 *R-tree* 长方形的基准。在 MySQL 中，这些 *R-tree* 是通过二次分割的方法管理的。

查询中经过加速的访问是以插入和存储要求的速度为代价的。例如，本章中介绍过的 *cities* 数据表就要求空间索引要比数据集合本身多出一半的空间。

如果想查看使用和不使用索引在速度方面有多大的差别，可以在查询中明确加入索引：*IGNORE INDEX(pt)*，关闭数据列 *pt* 的几何索引。

要查看 MySQL 是否已经在查询中建立了索引，可以使用 *EXPLAIN SELECT* 语句跟踪查询的执行：

```
EXPLAIN SELECT city FROM cities WHERE MBRCONTAINS ...
```

在这个 SQL 查询的输出里，将可以看到已经建立了哪些索引、它使用哪些索引，以及借助于索引可以排除多少个数据行。在接下来的例子里有许多比较复杂的查询命令，也许需要使用 *EXPLAIN SELECT* 命令去查看一下它们的执行状态信息。

12.3 SQL 示例（冰川数据库）

本节将给出一个使用 GIS 数据库的例子。在这个例子中，将会看到主要的 SQL 代码，可以自己对它们进行验证。图 12-3 给出了两个高度抽象化的冰川、一条冰川沉积带与消融带之间的界线（对应于 *ela* 字段，即那条虚线）和一个基准点（对应于 *ref* 字段）。其中一个冰川包含有一块岩礁，它没有被冰川覆盖，因此不属于那个冰川的多边形表面的一部分（图中的灰色部分）。

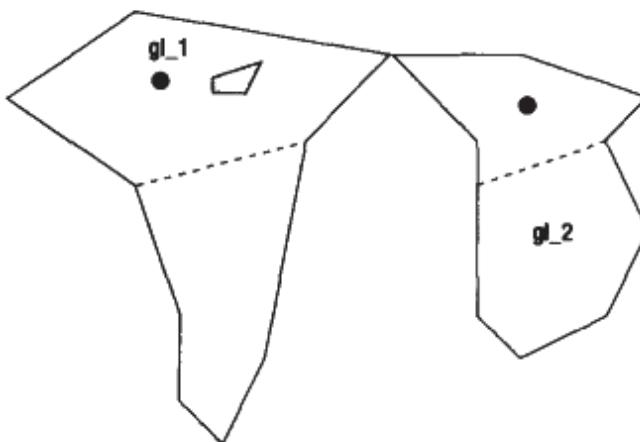


图 12-3 SQL 示例中的数据图解表示

12.3.1 创建数据表

下面的命令指出如何生成数据表 *glacier*, 以及怎样用数据来填充它。如果想避免在命令录入中出错, 可以先用 mysqladmin 程序创建一个新的数据库, 然后用 mysql 程序导入 *glacier.sql* 文件, 这个文件可以在本书示例文件中的 *databases* 子目录下找到。

```
CREATE TABLE glacier (
    id      INT NOT NULL PRIMARY KEY,
    border  POLYGON NOT NULL,
    ela     LINESTRING NOT NULL,
    ref     POINT NOT NULL,
    name    VARCHAR(20),
    SPATIAL KEY(border),
    SPATIAL KEY(ela),
    SPATIAL KEY(ref) )
```

12.3.2 插入数据

```
INSERT INTO glacier VALUES ( 1,
    GEOMFROMTEXT('POLYGON(
        (2500 1500,1000 2500,2500 3500,3000 5000,3000 6000,
        3500 6500,4000 5500,4500 3000,5500 2000,2500 1500),
        (3395 2827,3861 2840,4094 2374,3395 2594,3395 2827)
    )),
    GEOMFROMTEXT('LINESTRING(2500 3500, 4500 3000)'),
    GEOMFROMTEXT('POINT(2850 2350)'), 'gl_1')
INSERT INTO glacier VALUES ( 2,
    GEOMFROMTEXT('POLYGON((7000 2000,5500 2000,6500 3000,
        6500 5000,7000 5500,8000 5000,8500 4000,8000 3000,
        8500 2500,7000 2000))'),
    GEOMFROMTEXT('LINESTRING(6500 3500, 8000 3000)'),
    GEOMFROMTEXT('POINT(7250 2750)'), 'gl_2')
```

当使用 mysqldump 储存这个数据表的时候, 几何数据将以 WKB 而不是 WKT 的格式导出。

12.3.3 查询数据

录入数据后, 应该检查这些数据是否被正确导入。检查几何数据最简便的方法是使用 *asText* 函数:

name	<i>ASTEXT(ref)</i>	<i>ASTEXT(ela)</i>
gl_1	POINT(2850 2350)	LINESTRING(2500 3500,4500 3000)
gl_2	POINT(7250 2750)	LINESTRING(6500 3500,8000 3000)

也可以查询边界线的长度：

```
SELECT name, GLENGTH(ela), GLENGTH(border) FROM glacier
name      GLENGTH(ela)      GLENGTH(border)
gl_1      2061.5528128088 NULL
gl_2      1581.1388300842 NULL
```

结果多少有点令人意外，因为多边形的周长竟然是 *NULL* 值。事实上，多边形只有面积，没有长度。如果想知道多边形外围边界的长度，需要使用 *EXTERIORRING* 函数：

```
SELECT name, AREA(border), GLENGTH(EXTERIORRING(border)) FROM glacier
name      AREA(border)      GLENGTH(EXTERIORRING(border))
gl_1      8933474          15016.935459803
gl_2      5875000          11263.66792108
```

注意，第一个冰川的面积中自动减去了岩礁的面积。

岩礁的面积还可以通过下面的方法获得：以下命令将把几何对象转换为文本表示法，把 *LINestring* 替换为 *POLYGON*，再把结果转换为几何对象，然后返回它的面积：

```
SELECT AREA(GEOMFROMTEXT(
    CONCAT(REPLACE(
        ASTEXT(INTERIORRINGN(border, 1)),
        'LINestring(', 'POLYGON(', ',
        '''))))) AS innerArea
    FROM glacier WHERE id = 1
innerArea
191526
```

在这里，*INTERIORRINGN* 函数将返回一个 *LINestring* 类型。不过，计算面积需要用到一个 *POLYGON* 对象。但不幸的是没有办法在这两种类型之间相互变换，所以必须要使用 *REPLACE* 函数来进行文本替换。此外，这里还利用了 *CONCAT* 函数来插入多组外层括号。

两个几何对象（或是它们的外接框）是否相接可以用 *TOUCHES* 函数来测试：

```
SELECT TOUCHES(g1.border, g2.border)
    FROM glacier AS g1, glacier AS g2
    WHERE g1.id=1 AND g2.id=2
TOUCHES(g1.border, g2.border)
1
```

要比较这两个几何对象，该数据表必须在 *FROM* 中引用两次。在 *WHERE* 子句中，对于 *g1* 选取第一个记录，对于 *g2* 选取第二个记录。下一个感兴趣的值就是冰川的总面积。再没有比这更简单的事情了：

```
SELECT SUM(AREA(border)) FROM glacier
SUM(AREA(border))
14808474
```

或许还想知道冰川边界的总长度：

```
SELECT SUM(GLENGTH(EXTERIORRING(border))) FROM glacier
SUM(GLENGTH(EXTERIORRING(border)))
26280.603380883
```

冰川学家们一定想知道冰川沉积带的面积是多少（即虚线以上的部分）以及虚线以下的面积是多少。不过 MySQL 做不到这些，因为几何意义上的 *intersection*（交集）、*union*（并集）和 *difference*（异或）操作符还没有在 MySQL 里实现。

用户可以制定其他的查询，例如，考虑基准点的位置。*x* 值越小，冰川就位于越往西的位置。要想知道冰川最西面和最东面的点，使用下面的命令：

```
SELECT MIN(X(ref)), MAX(X(ref)) FROM glacier
MIN(X(ref))      MAX(X(ref))
2850            7250
```

还可以把同一个数据表里的多个几何数据列采集到同一个 *GEOMETRYCOLLECTION* 对象里：

```
SELECT ASTEXT(GEOMFROMTEXT(
    CONCAT('GEOMETRYCOLLECTION(',
           ASTEXT(border),',', ASTEXT(ref), ')'))
    AS collection FROM glacier
collection
GEOMETRYCOLLECTION(POLYGON((2500 1500,1000 2500,2500 3500,
    3000 5000,3000 6000,3500 6500,4000 5500,4500 3000,
    5500 2000,2500 1500),
    (3395 2827,3861 2840,4094 2374,3395 2594,3395 2827)),
    POINT(2850 2350))
GEOMETRYCOLLECTION(POLYGON((7000 2000,5500 2000,6500 3000,
    6500 5000,7000 5500,8000 5000,8500 4000,8000 3000,
    8500 2500,7000 2000)),
    POINT(7250 2750))
```

这个例子也用到了 *CONCAT* 函数。为了让 *GEOMFROMTEXT* 函数创建出一个正确的 *GEOMETRYCOLLECTION* 对象，必须把相应的字符串放在带括号的表达式的开头。而且，括号内的单个对象必须用逗号分开。这些工作都是由 *CONCAT* 函数负责完成的。

在 MySQL 里，几何对象的外接框永远是长方形；MySQL 用 *POLYGON* 类型来表示外接框。作为一个特例，*POINT* 对象的外接框是一个缩小为一个点的多边形：

```
SELECT ASTEXT(ENVELOPE(ela)) FROM glacier
ASTEXT(ENVELOPE(ela))
POLYGON((6500 3000,8000 3000,8000 3500,6500 3500,6500 3000))
POLYGON((2500 3000,4500 3000,4500 3500,2500 3500,2500 3000))
SELECT ASTEXT(ENVELOPE(ref)) FROM glacier
ASTEXT(ENVELOPE(ref))
POLYGON((7250 2750,7250 2750,7250 2750,7250 2750,7250 2750))
POLYGON((2850 2350,2850 2350,2850 2350,2850 2350,2850 2350))
SELECT ASTEXT(ENVELOPE(border)) FROM glacier
ASTEXT(ENVELOPE(border))
POLYGON((1000 1500,5500 1500,5500 6500,1000 6500,1000 1500))
POLYGON((5500 2000,8500 2000,8500 5500,5500 5500,5500 2000))
```

12.4 SQL 示例 (*opengeodb* 数据库)

示例数据库 *opengeodb* 只包含着一个名为 *cities* 的数据表。这个数据表的内容是奥地利、德国和瑞士等国家的一些城市、它们的地理坐标（UTM 坐标系统的第 32U 区）、邮政编码（*zip* 字段）和其他一些数据。下面的命令将随机选取 10 条记录显示出来：

```
SELECT id, ASTEXT(pt), zip, country AS cnt, state AS st, city, district
FROM cities ORDER BY RAND() LIMIT 10
id      ASTEXT(pt)      zip      cnt      st      city      district
3357   POINT(863667 5784840) 15848  DE      BB      Friedland
7249   POINT(320685 5500822) 54441  DE      RP      Mannebach
14857  POINT(605034 5213504) 6555   AT      7       Kappl
16458  POINT(588645 5122467) 7747   CH      GR      Viano
9104   POINT(702844 5728043) 6388   DE      ST      Piethen
2231   POINT(501094 5976100) 25709  DE      SH      Diekhusen...
13113  POINT(726772 5656658) 6712   DE      ST      Würchwitz
10491  POINT(474289 5403425) 75328  DE      BW      Schömberg
13518  POINT(421059 5942907) 26409  DE      NI      Wittmund      Berdum
12622  POINT(396497 5621193) 57635  DE      RP      Werkhausen
```

表 12-11 对这个数据表的数据列属性进行了汇总。注意，*pt* 数据列有一个二维坐标索引。

表 12-11 cities 数据表的数据列属性

字 段	数据类型	属 性	索 引
<i>id</i>	<i>INT</i>	<i>NOT NULL AUTO_INCREMENT</i>	<i>PRIMARY KEY</i>
<i>pt</i>	<i>POINT</i>	<i>NOT NULL</i>	<i>SPATIAL KEY</i>
<i>zip</i>	<i>INT</i>	<i>NULL</i>	<i>KEY</i>
<i>country</i>	<i>VARCHAR(10)</i>	<i>NULL</i>	<i>KEY</i>
<i>state</i>	<i>CHAR(2)</i>	<i>NULL</i>	
<i>city</i>	<i>VARCHAR(100)</i>	<i>NULL</i>	<i>KEY</i>
<i>district</i>	<i>VARCHAR(100)</i>	<i>NULL</i>	

cities 数据表比上面示例里的 *glacier* 数据表大多了，它有大约 18 000 条记录。*cities* 数据表里的数据大约有 1MB，这些数据的索引还需要另外 1.2MB 的空间。因此，这个数据表不仅可以用来测试各种 SQL 命令，还可以用来进行性能测试（比如说，对使用了索引和没有使用索引的坐标查找操作进行对比等）。

12.4.1 数据来源和导入

示例数据库 *opengeodb.sql* 来自同名的 OpenGeoDB 项目。这个项目由德语国家发起，目标是为这些国家的城镇建立一个地理信息数据库供人们免费使用。这个数据库可以从以下网址下载（它使用的是 LGPL 许可证，即把这些数据用于商业化目的也是允许的）：<http://opengeodb.de/download/>。

这个数据库里的信息质量很好，但与完美还有一段距离。OpenGeoDB 对此做出的解释是它们只对数据进行了适度的检查，错误在所难免。在下面给出的各有关示例里，将把重点放在以实际数据对 MySQL 提供的各种 GIS 函数进行测试方面，不探讨地理数据的具体应用问题。

12.4.2 导入

原始数据来自一个 CSV (comma-separated values, 用逗号分隔的数据值) 文件，这种文件与 MySQL 能够直接读入的 SQL 文件有着不同的格式。下面是这个文本文件里的部分内容：

```
30443; AT; 3; -; -; -; Zwingendorf; -; -; 48.7; 16.2333; -; 2063
30444; CH; AG; -; -; -; Aarau; -; -; 47.3833; 8.05; -; 5000,5004,5001
30445; CH; AG; -; -; -; Aarburg; -; -; 47.3167; 7.9; -; 4663
30446; CH; AG; -; -; -; Aettenschwil; -; -; 47.1833; 8.36667; -; 5645
```

以经度和纬度给出的地理坐标值出现在第 10 个和第 11 个字段。这种坐标系统不适用于 MySQL。还好，在因特网上找到了一个开源函数库 *Proj.4*，它可以把十进制的时、分、秒数值转换为 UTM 投影坐标系统。适用于 Windows 和 Linux 系统的二进制软件包都可以在 <http://www.remotesensing.org/proj/> 处找到。

下面将使用 *Proj.4* 软件包里的可执行程序 *cs2cs* 来完成坐标系统之间的转换工作。下面这条命令演示了如何把 *latlong* 坐标系统里的一组坐标转换为 UTM 坐标系统第 32U 区里的一组坐标：

```
user@host > echo "11.4 47.2667" | cs2cs +proj=latlong +to +proj=utm +zone=32U
681547.32      5237595.88 0.00
```

注解 本章中的一部分为 Bernd Öggel 所著（奥地利，Innsbruck 市）。由于他在地理学方面的高深造诣，以及作为一名 GIS 系统与数据库的资深讲师，他在 GIS 领域的知识与经验远非我等所及。同时我也非常感谢他允许我在此书中引用他的研究成果。

12.4.3 对圆形地理区域进行搜索

想像这样一个场景：为了开展某项业务，需要了解某个地区的现有状况以做出最佳的业务决策。比如说，需要了解在距公司半径 30km 的范围内所有旧洗衣机的分布情况。*cities* 数据表为这种搜索提供了必要的基础。

首先，需要把搜索范围的圆心（在下面的例子里，将以 Innsbruck 市作为这个圆心）坐标确定下来。可以用一条简单的 *SELECT* 命令把这个圆心的坐标提取到变量 @x0 和 @y0 里（对于比较大的城市，使用 “*zip*= ...” 作为条件表达式要比使用 “*city*= ...” 更好）：

```
SELECT X(pt), Y(pt) FROM cities
WHERE city='Innsbruck' LIMIT 1 INTO @x0, @y0
SELECT @x0, @y0
@x0      @y0
681547.32 5237595.88
```

现在，需要从 *cities* 数据表里把那些与圆心 (@x0, @y0) 之间的距离在给定范围内的地点查找出来。不过，因为 MySQL 中的 GIS 实现目前还不支持 *DISTANCE* 函数，所以这个距离还需要根据勾股定理 ($r^2 = (x1-x0)^2 + (y1-y0)^2$) 把它计算出来。按勾股定理计算出来的结果还要除以 1000 并进行必要的舍入才能得到一个以公里为单位的距离：

```
SELECT ROUND(SQRT(POW(@x0 - X(pt), 2) + POW(@y0 - Y(pt), 2)) / 1000) AS distance,
       city FROM cities
  HAVING distance<=30 ORDER BY distance
   distance   city
    0   Innsbruck
    4   Rum
    4   Natters
    4   Mutters
    4   Aldrans
    5   Sistrans
    5   Völs
    ...
    ...
   29   Wallgau
   29   Pertisau
   30   Tux
49 rows in set (0.15 sec)
```

速度优化

上面的计算过程和结果都是正确的，但查询命令的执行时间未免太长了点儿。这一是因为 MySQL 为 17 000 条记录计算了距离，二是因为在计算距离时没能用上 *pt* 数据列的空间索引。如果先用一个子查询命令把与圆心距离在 30km 内的所有记录查找出来再进行计算，就能大大加快总的查询速度。在下面的例子里，在子查询命令的条件表达式里使用了 *MBRCONTAINS* 函数，因为用了 *pt* 数据列的空间索引，MySQL 完成这个查询的速度快多了。

为了做到这一点，还需要做些准备工作。首先，必须创建一个几何对象（示例中的 @bbox 变量）来描述一个距起始点 30km 的正方形；这一点用基本的 SQL 代码是很难做到的。（两条用来立刻显示查询结果的 *SELECT* 命令只是为了让大家了解有关代码的执行情况，它们并不是必须的。在使用 PHP 或其他程序设计语言开发自己的应用程序时，直接拼装 *MULTIPOINT* 字符串的办法要简单得多。）

```
SET @d0 = 30000
SET @str = CONCAT('MULTIPOINT(', @x0-@d0, ', ', @y0-@d0, ', ',
                   @x0+@d0, ', ', @y0+@d0, ')')
SET @bbox = ENVELOPE(GEOMFROMTEXT(@str))
SELECT @str
@str
```

```

MULTIPOINT(651547.32 5207595.88, 711547.32 5267595.88)
SELECT ASTEXT(@bbox)
ASTEXT(@bbox)
POLYGON((651547.32 5207595.88, 711547.32 5207595.88,
711547.32 5267595.88, 651547.32 5267595.88,
651547.32 5207595.88))

```

利用@*box* 变量里的 *POLYGON* 对象，可以轻而易举地查出搜索范围内的所有地点。(注意，这次的处理时间是 0.00 s (秒)，即这个时间太短了，mysql 程序无法把它测量出来。)

```

SELECT city, X(pt), Y(pt) FROM cities
WHERE MBRCONTAINS(@bbox, pt)
city      X(pt)      Y(pt)
Farchant   659311.95 5266601.84
Krün       671959.24 5263256.57
...
Mittenwald 669666.07 5255771.14
Scharnitz   672340.12 5250287.98
54 rows in set (0.00 sec)

```

接下来，可以从上面这些结果里把那些距离超过 30km 的所有城市去掉（因为边长为 60km 的正方形要比半径为 30km 的圆形大一些）。在下面的子查询命令里，内层的 *SELECT* 命令将返回所有这样的城市，外层的 *SELECT* 命令则负责计算实际距离并排除那些太远的城市。注意，这条命令的总执行时间是 0.01s (秒)，比最早使用的那条 *SELECT* 查询命令快了很多。

```

SELECT ROUND(SQRT(POW(@x0 - sub.x, 2) + POW(@y0 - sub.y, 2))) / 1000 AS distance,
sub.city
FROM (SELECT city, X(pt) AS x, Y(pt) AS y FROM cities
      WHERE MBRCONTAINS(@bbox, pt)) AS sub
HAVING distance<=30 ORDER BY distance
distance sub.city
0 Innsbruck
4 Rum
...
30 Tux
49 rows in set (0.01 sec)

```

存储过程（stored procedure，SP）是 MySQL 5.0 版本中的最大创新。它们是一些由 MySQL 服务器直接存储和执行的定制过程或函数。SP 的加入把 SQL 语言扩展成了一种程序设计语言，可以利用 SP 把一个客户—服务器体系的数据库应用软件中的部分逻辑保存起来供日后使用。

本章将解释一下为什么说 SP 是个好东西（提高速度、提高数据的安全性、减少代码冗余等），然后再介绍 MySQL 里的 SP 实现细节并提供几个 SP 应用示例。

触发器（trigger）是在 *INSERT*、*UPDATE* 或 *DELETE* 命令之前或之后对 SQL 命令或 SP 的自动调用。比如说，可以为每一个 *UPDATE* 操作测试被修改的数据是否满足特定的条件。MySQL 5.0 里的触发器实现还很不完善。触发器大概要等到 MySQL 5.1 版本才会有实用价值，所以本章将只对触发器做简单的介绍。

提示 在 MySQL 文档里可以找到关于 SP 和触发器的信息：<http://dev.mysql.com/doc/mysql/en-stored-procedures.html>；<http://dev.mysql.com/doc/mysql/en/triggers.html>；<http://dev.mysql.com/tech-resources/articles/mysql-storedprocedures.pdf>。

13.1 为什么要使用存储过程和触发器

13.1.1 存储过程

存储过程是一些存储和执行在 MySQL 服务器里的 SQL 语句。根据具体的应用程序，它们有以下好处：

□ **更快的速度。**在进行数据库操作的时候，经常出现必须在 PHP 程序和数据库服务器之间来回传输大量数据的情况：PHP 程序执行一条 *SELECT* 命令，对查询结果进行某种处理，根据查询结果执行一条 *UPDATE* 命令、返回 *LAST_INSERT_ID* 等。如果把所有这些步骤都纳入一个 SP 在服务器上执行的话，数据传输方面的许多开销都可以节省下来。

SP 的另一个好处是数据库服务器可以对 SP 代码进行一些预处理（这类似于编译器把源代码编译成一个二进制文件）。不过，MySQL 可以对 SP 代码进行哪些优化以及优化后的代码在速度方面可以有多大幅度的提高等问题在 MySQL 文档里没有说明。

注意，使用 SP 并不能保证速度肯定会加快。只有高效率的 SP 代码才能赢得速度优势。SQL 比 PHP 原始得多，SQL 对 SP 代码的优化效果很难与 PHP 对等效 SQL 命令的优化效果相提并论。

使用 SP 可以大大提高 MySQL 服务器的集成程度，但使用 PHP 的 Web 服务器却不是这样。

最终效果的好坏要取决于系统的整体配置情况。如果 MySQL 服务器与 Web 服务器分别运行在不同的计算机上并且 MySQL 服务器已经是一个瓶颈，大量使用 SP 反而会使整体情况更加恶化——虽然这能减轻 Web 服务器的一部分负担。

- **避免代码冗余。**多个应用程序访问的是同一个数据库的事情很常见。这些应用程序在检查输入数据、插入新数据记录和修改数据表等方面所使用的代码往往大同小异，甚至有可能一模一样。把功能雷同的代码编写为一个 SP 不仅可以减少冗余代码，还可以使有关的应用程序变得更容易维护：在某个数据库的结构被改变时，程序员只须修改几个 SP 就可以了，用不着再去修改每一个会用到这个数据库的应用程序的代码。
- **提高数据库的安全性。**在许多对信息的安全性有着较高要求的行业（比如银行）里，不允许用户程序直接访问数据表。客户程序必须使用 SP 进行所有的数据库操作，查询数据(*SELECT*)、插入新数据、修改现有数据等操作都必须通过 SP 去完成。这看起来似乎相当复杂，而事实上创建和维护大量的 SP 也的确是一件工作量很大的任务。但对数据库管理员来说，这么做的好处是可以对每一次数据访问进行监控，并在必要时把操作情况记录到一个日志里去。总之，人们可以根据具体情况为不同的数据和数据访问操作设置不同严格程度的安全检查规则。

虽然，存储过程也并非全是优点。SP 的最大缺点之一是很难把它们从一个数据库移植到另一个数据库里去。这是因为（几乎）每一种数据库系统所使用的 SP 语法或语法扩展都是不同的，这也正是为什么需要花费很大的力气才能把 Microsoft SQL Server 或 Oracle 数据库系统的 SP 导入 MySQL 的原因。（当然，把 MySQL 的 SP 导入其他数据库系统也同样困难。）

13.1.2 触发器

触发器是在 *INSERT*、*UPDATE* 或 *DELETE* 命令之前或之后对 SQL 命令或 SP 的自动调用。比如说，可以为每一个 *UPDATE* 操作测试被修改的数据是否满足特定的条件。触发器的其他用途还包括把数据修改情况记载到日志里、对其他数据表里的变量或数据列进行同步修改等。

因为触发器代码是在数据表里的数据发生变化后自动执行的，所以比较复杂的触发器往往会对数据库的正常使用造成显著甚至严重的影响。这一问题在用户执行的是一条需要修改大量数据记录的命令（比如像 *UPDATE table SET columnA=columnA+1* 这样的命令）的时候表现得尤其明显：触发器代码必须为每一条数据记录执行一遍。

13.2 初识 SP

了解存储过程的最佳办法是通过一个简单的例子。本节将编写一个 *shorten* 函数，它可以把给定字符串截短到一个给定的最大长度。比如说，*shorten("abcdefghijklmnopqrstuvwxyz", 15)* 将返回字符串 "abcde...vwxyz"。

注解 存储过程可以分为两种类型：过程和函数。这两种类型在细节方面的差异将在本章后面的内容里陆续讨论。就目前而言，只要记住 SP 函数有返回值、SP 过程允许使用引用参数（reference parameter，相当于其他程序设计语言里的“可变参数”或“形式参数”）和更多种 SQL 命令（比如 *SELECT* 和 *INSERT*）就行了。此外，SP 过程必须用 *CALL* 命令来调用，SP 函数可以内嵌在普通的 SQL 命令里使用。本节示例演示的是一个 SP 函数的定义和使用情况。

在 2005 年初，只有两个程序可以用来输入和测试 SP 函数：MySQL 命令解释器 mysql 程序和 MySQL Query Browser。其他的 MySQL 数据库管理工具（比如 phpMyAdmin 2.6）目前还不支持 SP 的使用，这种局面很快就会好转，等读者读到这本书的时候说不定已经有许多工具可供选择了。但万一情况不是这样，不妨试试 SP Administrator，这是在第 15 章作为一个 PHP 编程示例介绍给大家的小工具。SP Administrator 工具可以帮助用户方便地完成 SP 的创建、编辑、测试和备份/恢复等任务。

13.2.1 MySQL 命令解释器：mysql 程序

启动这个命令解释器的办法是：在 UNIX/Linux 环境下，在一个控制台窗口里执行 mysql -u root -p 命令，并根据提示输入正确的密码；在 Windows 环境下，执行菜单命令 Programs（程序）| MySQL | MySQL Server | MySQL Command Line Client。

1. 定义SP函数

在 mysql 程序里，先把 *mylibrary* 数据库选为当前数据库，然后把命令结束符从 “;” 改为 “\$\$”。这两个字符将作为输入行的结束标志。SP 代码需要使用分号作为语句的结束标志，所以它不再适合充当 mysql 程序的命令结束符了。请记住，在执行完下面两条命令之后，所有的输入都必须以 “\$\$” 结束：

```
mysql> use mylibrary
mysql> delimiter $$
```

接下来的步骤是定义新函数 *shorten*。千万不要打错字！当然，等 mysql 报错后再去修改也不是不可以，但查错纠错的过程往往很麻烦。

shorten 函数应该很容易看懂。这个函数有两个参数：一个是字符串 *s*，另一个是负责给出结果字符串最大长度的整数 *n*。这个函数将返回一个数据类型为 *VARCHAR(255)* 的字符串。如果 *n* 小于 15，超长的字符串将由 SQL 字符串函数 *LEFT* 截短到 *n* 个字符；如果 *n* 大于或等于 15，超长的字符串将只保留前 (*n*-10) 个字符和后 5 个字符，它们中间用字符串 “...” 连起来。

```
mysql> CREATE FUNCTION shorten(s VARCHAR(255), n INT)
->   RETURNS VARCHAR(255)
-> BEGIN
->   IF ISNULL(s) THEN
->     RETURN '';
->   ELSEIF n<15 THEN
->     RETURN LEFT(s, n);
->   ELSE
->     IF CHAR_LENGTH(s) <= n THEN
->       RETURN s;
->     ELSE
->       RETURN CONCAT(LEFT(s, n-10), '...', RIGHT(s, 5));
->     END IF;
->   END IF;
-> END$$
Query OK, 0 rows affected (0.00 sec)
```

提示 直接在 mysql 程序里敲入 SP 代码只是办法之一，还可以用一个文本编辑器来做这件事，则文本文件必须包含以下内容：

```
- file sp.sql
DROP FUNCTION IF EXISTS shorten;
DELIMITER $$
```

```
CREATE FUNCTION shorten ... $$
```

现在，可以用下面这条 mysql 命令读入那个文本文件去定义 *shorten* 函数：

```
> mysql -u root -p mylibrary < sp.sql
```

如果遇到错误，重新编辑文本文件并再次读入它即可。请注意，以交互方式在mysql程序里使用`source filename`命令是无法读入那个文本文件的，`source`命令要求其输入文件里的SQL命令以分号作为分隔符，它不接受`delimiter`命令。

2. 使用SP函数

把`shorten`函数定义出来以后，就可以试用它了。SP 函数可以用在普通的 SQL 命令里，如下所示：

```
mysql> SELECT shorten("abcdefghijklmnopqrstuvwxyz", 15)$$
      abcde ... vwxyz
mysql> SELECT title, shorten(title, 20) FROM titles LIMIT 10$$
      A Guide to the SQL Standard          A Guide to ... ndard
      A Programmer's Introduction to PHP 4.0 A Programm ... P 4.0
      Alltid den där Annette            Alltid den ... nette
      Anklage Vatermord                Anklage Vatermord
      Apache Webserver 2.0              Apache Webserver 2.0
      Client/Server Survival Guide     Client/Ser ... Guide
      Comédia Infantil                 Comédia Infantil
      CSS-Praxis                      CSS-Praxis
      Dansläraren Återkomst           Danslärare ... komst
      Darwins Radio                   Darwins Radio
```

如果不喜欢单引号“`$$`”作为命令结束符，用下面这条命令把它改回 mysql 程序的默认分隔符“`;`”即可：

```
mysql> delimiter ;
```

13.2.2 MySQL Query Browser

除了 MySQL 自带的命令解释器 mysql 程序，还可以使用 MySQL Query Browser 来定义 SP。截止到 2005 年 3 月，这个工具对 SP 的支持还不够成熟，但怎么说它也比 mysql 程序用起来要更方便些。

1. 定义SP函数

先通过菜单命令 File（文件）| Select Schema（选择数据库）选中`mylibrary`数据库，然后执行菜单命令 Script（脚本）| Create Stored Procedure（创建存储过程）。在给新 SP 起好名字并选定它的类型（procedure（过程）或 function（函数））后，Query Browser 将提供一个模板供输入新 SP 的代码（如图 13-1 所示）。输入完成后，单击 Execute（执行）按钮以保存新 SP。如果遇到语法错误，需要单击 Stop（停止）按钮中断执行，等改正错误后再进行尝试。

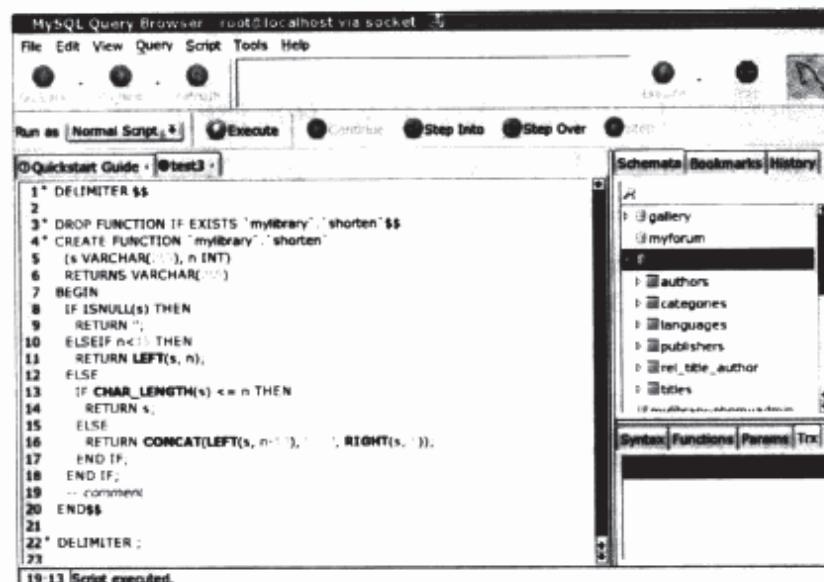


图 13-1 在 MySQL Query Browser 里定义一个新的 SP 函数

2. 使用SP函数

如果想试用一下新函数，用菜单命令 File（文件）| New query Tab（新查询）打开脚本对话框，在这个对话框中的命令输入栏里输入 `SELECT shorten(title, 20), title FROM titles` 命令，然后单击 Execute（执行）按钮执行这个命令（如图 13-2 所示）。

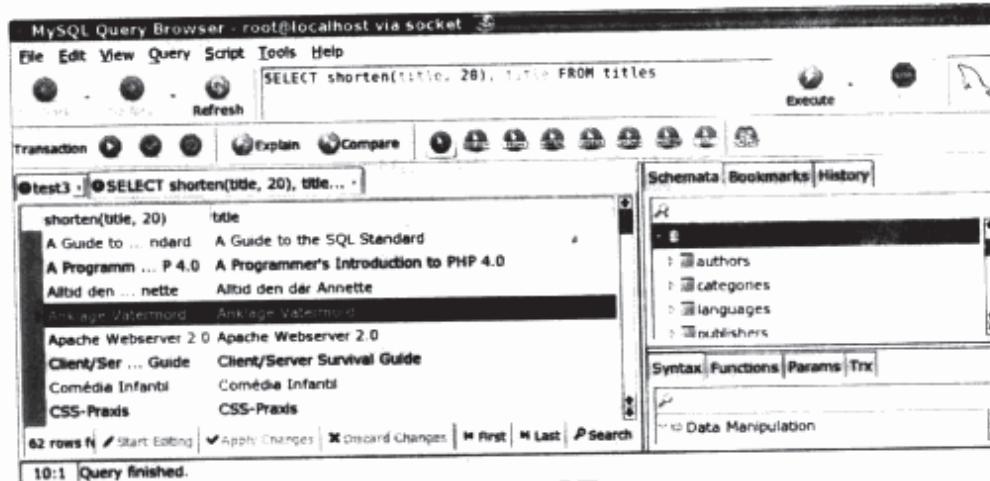


图 13-2 在 MySQL Query Browser 里试用一个 SP 函数

3. 编辑SP函数

只要还没有关闭刚才用来定义 `shorten` 函数的对话框，就可以随时对它进行编辑（千万不要忘记最后要单击 Execute（执行）按钮）。

如果已经关闭对话框，再想编辑 `shorten` 函数就不那么容易了。MySQL Query Browser 只提供了一个菜单命令 Script（脚本）| Edit All Stored Procedures（编辑全部的存储过程），它将打开一个新的对话框，其内容是当前数据库里所有 SP 的代码。（在笔者进行的测试里，MySQL Query Browser 1.15 版里的这个菜单命令不起作用，更早的版本倒是没问题。）现在，找到打算编辑的 SP（本例中是 `shorten` 函数），把它的代码（包括首、尾的 `DELIMITER` 命令以及它们中间 `DROP` 和 `CREATE` 命令）拷贝到一个新的脚本对话框里（菜单命令是 File（文件）| New Query Tab（新查询）进行编辑；完成后别忘了单击 Execute（执行）按钮保存它）。

13.3 SP 的实现

□ **SQL:2003 标准。**前面讲过，几乎每一种数据库系统所使用的 SP 语法或语法扩展都是不同的，有多少种数据库系统，就有多少种不同的 SP 语法。还好，MySQL 5.0 里的 SP 语法遵循 SQL:2003 标准，这使得 MySQL 里 SP 与 IBM DB/2 数据库系统里的 SP 有着很好的兼容性，但它们与来自 Oracle 和 Microsoft SQL Server 的 SP 就不兼容了。

在未来，MySQL 很可能会提供一个接口以允许人们使用 PHP 或 Perl 等外部程序设计语言来定义 SP。这将为那些 PHP 程序员提供一个特别有吸引力的编程机会，而 SP 也将能用上 PHP 的全部功能。这对 SP 的执行速度有何影响很值得期待。

□ **SP 的内部存储。**MySQL 把 SP 集中存放在 `mysql.proc` 数据表里。在这个数据表的各个数据列里存放着 SP 所属的数据库名、SP 的名字和类型（`PROCEDURE` 或 `FUNCTION`）、SP 的参数、SP 的实际代码以及各种各样的其他属性（如图 13-3 所示）。注意，每一个 SP 都属于一个特定的数据库。

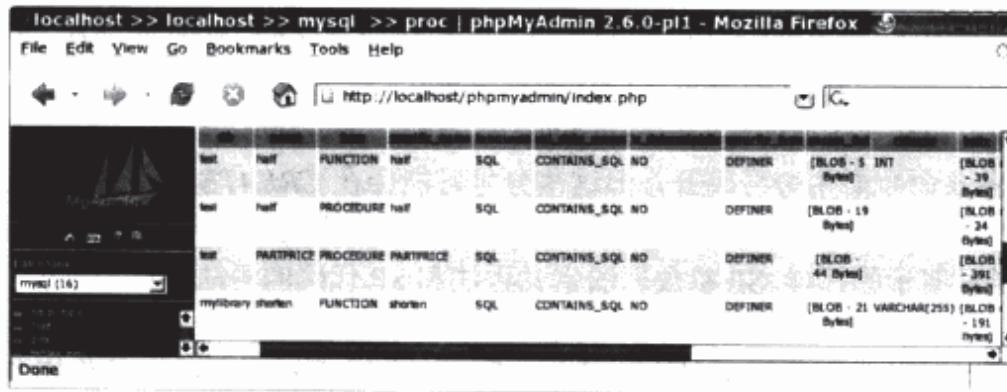


图 13-3 SP 在 MySQL 内部的存储情况

注意 *mysql.proc* 数据表必须存在才能用来存放 SP。如果正在使用的 MySQL 5.0 不是全新安装的而是从某个老版本升级而来的, *mysql* 数据库里很可能没有 *proc* 数据表。在升级到 MySQL 5.0 版之后创建 *proc* 数据表的具体步骤可以在下面这个网址处的文章里找到: <http://dev.mysql.com/doc/mysql/en/upgrading-grant-tables.html>。

13.4 SP 的管理

本节将介绍一些用来管理 SP 的 SQL 命令, 这些命令可以帮助我们创建、编辑和删除 SP。这些命令都比较冗长, 用起来很不方便。但截止到 2005 年 3 月为止, 还没有一种足够好的 MySQL 工具能简化 SP 管理工作中的各种操作。这样的工具应该会在不久的将来出现, 但就目前而言, 不妨使用将在第 15 章介绍的 SP Administrator 作为一种临时过渡。

13.4.1 创建、编辑和删除 SP

1. 创建 SP

正如 13.2 节中的例子演示的那样, 新 SP 要用 *CREATE FUNCTION* 或 *CREATE PROCEDURE* 命令来创建。必须具备 *Create Routine* 权限才能执行这两条命令。下面是这两条命令的语法:

```
CREATE FUNCTION name ([parameterlist]) RETURNS datatype
[options] sqlcode
CREATE PROCEDURE name ([parameterlist])
[options] sqlcode
```

这两条命令都将把新创建出来的 SP 与当前数据库关联起来。函数和过程允许有同样的名字。在定义里, 可以给出以下选项:

- **LANGUAGE SQL**。*SQL* 是 *LANGUAGE* 选项目前仅有的一个可取值, 也是这个选项的默认值。MySQL 的未来版本可能会给这个选项提供一些其他的可取值, 那将意味着我们可以使用其他程序设计语言去定义 SP。
- **[NOT] DETERMINISTIC**。如果某个 SP 对同样的参数总是返回同样的结果, 就说它是可确定的 (*DETERMINISTIC*)。如果某个 SP 的返回结果需要取决于数据表, 它就是不可确定的 (*NOT DETERMINISTIC*)。

在这个选项默认的情况下定义出来的 SP 都是不可确定的。不过, 可确定的 SP 能够以更高的效率执行, 因为把这种 SP 的返回结果保存在缓冲区里才有实际的意义。不过, 就目前而言, 即使在 SP 中给出了 *DETERMINISTIC* 选项, MySQL 也会忽略它。

SQL SECURITY [DEFINER | INVOKER]。SQL SECURITY 选项负责设置 SP 在执行时的访问权限。这方面的细节将在本章后面的内容里讨论。

COMMENT 'text'。注释内容'text'将随 SP 一同存储在 mysql.proc 数据表里。

SP 代码中的各条命令要用分号隔开，但这些分号可能会在 CREATE 命令执行时引起问题。因此，如果打算在 mysql 程序里定义 SP，就必须先用 DELIMITER 命令另行指定一个命令结束符；就像我们在本章前面的例子里演示的那样。SP 参数表的定义语法和 SP 代码的语法将在后面的章节里陆续介绍。

2. 删除SP

下面的命令可以用来删除一个现有的过程或函数。可选关键字 *IF EXIST* 的作用是：即使用户试图删除的 SP 不存在，SP 删除命令也不会触发一个错误。用户必须具备 *Alter Routine* 权限才能执行这条命令；SP 函数或过程的创建者（即当初执行 CREATE FUNCTION/PROCEDURE 命令的那位用户）将自动获得这一权限。

```
DROP FUNCTION [IF EXISTS] name
DROP PROCEDURE [IF EXISTS] name
```

3. 修改SP

可以用 ALTER 命令来修改一个 SP 的名字和它的某些选项，但必须具备相应的 *Alter Routine* 权限才能执行这条命令：

```
ALTER FUNCTION/PROCEDURE name
[NAME newname]
[SQL SECURITY DEFINER/INVOKER]
[COMMENT 'newcomment']
```

注解 就目前（5.0.3版）而言，MySQL还没有提供任何可以用来修改现有SP代码的手段。如果想修改某个现有SP的代码，就只能先用DROP命令删除这个SP，然后再用CREATE命令重新创建它。

4. 查看现有的SP

SHOW PROCEDURE STATUS 或 SHOW FUNCTION STATUS 命令将返回一份现有的过程和函数的完整清单，但可以利用一个 LIKE pattern 表达式让这两条命令只列出与给定搜索模板相匹配的过程或函数。不管使用的是这两条命令中的哪一个，看到的都将是全体数据库（不仅仅是当前数据库）里的 SP。为了节约篇幅，在下面这个例子里省略了返回结果里的 Modified 和 Created 列：

SHOW FUNCTION STATUS						
Db	Name	Type	Definer	Security_type	Comment	
mylibrary	faculty	FUNCTION	root@192.168.80.1	DEFINER		
mylibrary	shorten	FUNCTION	root@192.168.80.1	DEFINER		
mylibrary	shortentest	FUNCTION	root@localhost	DEFINER		
mylibrary	swap_name	FUNCTION	root@192.168.80.1	DEFINER		
test	test	FUNCTION	root@localhost	DEFINER		

information_schema.routines 数据表可以提供更详细的信息。它包含着 mysql.proc 数据表里的所有数据，只是有几个输出列的名字与上面的不一样：

```
SELECT routine_name, routine_type, created
FROM information_schema.routines
WHERE routine_schema='mylibrary'
routine_name          routine_type    created
categories_insert     PROCEDURE      2005-01-31 11:29:16
...
test                 FUNCTION      2005-02-06 10:43:11
titles_insert_all     PROCEDURE      2005-01-31 11:29:17
```

5. 查看SP的代码

必须知道某个 SP 的名字才可以去查看它的代码，具体做法是执行 `SHOW CREATE FUNCTION name` 或 `SHOW CREATE PROCEDURE name` 命令：

```
SHOW CREATE FUNCTION shorten
Function      sql_mode Create Function
shorten          CREATE FUNCTION mylibrary.shorten(s VARCHAR(255),
                                                n INT) RETURNS VARCHAR(255)
                    BEGIN
                        IF ISNULL(s) THEN
                            RETURN '';
                        ELSEIF n<15 THEN
                            RETURN LEFT(s, n);
                        ELSE
                            IF CHAR_LENGTH(s) <= n THEN
                                RETURN s;
                            ELSE
                                RETURN CONCAT(LEFT(s, n-10), ' ... ',
                                              RIGHT(s, 5));
                            END IF;
                        END IF;
                    END
```

13.4.2 信息安全问题

只有具备 *Create Routine* 和 *Alter Routine* 权限的 MySQL 用户才可以创建、修改或删除 SP，只有在某个 SP 上具备 *Excute* 权限的 MySQL 用户才允许执行这个 SP。这 3 项权限都可以通过 MySQL 提供的用户管理功能来设置（详见第 11 章）。

即使当前用户有权执行 SP，他能通过 SP 进行哪些数据库操作还是个问题。（比如说，他能否通过某个特定的 SP 去删除某个数据表里的记录？）MySQL 会根据各 SP 的具体定义做出两种选择：

- **SQL SECURITY INVOKER**。在定义时使用了这个选项的 SP 将以实际调用它们的那位 MySQL 用户的访问权限执行。（如果 MySQL 用户无权删除数据记录，他调用的 SP 就无法删除数据记录。）
- **SQL SECURITY DEFINER**。在定义时使用了这个选项的 SP 将以当初定义它们的 MySQL 用户的访问权限执行。这是 MySQL 采用的默认设置；也就是说，在定义时没有明确给出 *SQL SECURITY INVOKER* 选项的 SP 都将以 *DEFINER* 方式执行。（警告：*root* 用户定义的 SP 在所有的数据库里都有着不受限制的权限。）

13.4.3 SP 的备份和恢复

就目前（2005 年 3 月）而言，可以用来创建数据库备份的常用工具如 MySQL Administrator、`mysqldump` 命令和 phpMyAdmin 2.6 中的导出工具等都不能对 SP 进行备份。

如果想把所有数据库里的所有 SP 全都备份下来，最好的办法是对 `mysql.proc` 数据表进行备份，例如：

```
> mysqldump -u root -p mysql proc > backup.sql
```

但是这个办法有两个很大的不足：

- 必须是一位有权访问 `mysql.proc` 数据表的用户才能使用这个办法，但这个数据表一般只允许 *root* 用户访问，普通用户不具备相应的访问权限。

- 无法做到只备份某一个数据库里的 SP，而这又意味着无法在不覆盖其他数据库里的 SP 的情况下只读入某一个数据库里的 SP。

有个办法可以只备份某一个数据库里的 SP：以手动方式为这个数据库里的每一个 SP 执行一遍 *SHOW CREATE FUNCTION/PROCEDURE* 命令。还有一个更简便的办法是使用我们将在本书第 15 章介绍的 SP Administrator 来备份 SP。

13.5 SP 的语法和语言元素

SP 的代码部分主要由一些普通的 SQL 语言命令构成，这类命令在本书的第 9 章和第 10 章已经介绍过不少了。除了各种 SQL 语言命令，在编写 SP 代码时还可以使用一些其他的命令，它们不仅可以帮助编写出循环结构和条件分支，甚至还可以帮助用户遍历某个数据表里记录（见本章后面的 13.5.8 节）。这些命令只能在 SP 里使用，也是本节里的讨论重点。

正如在前面多次提到过的那样，有两种类型的 SP：一种是过程，另一种是函数。表 13-1 对这两种 SP 之间最主要的区别进行了汇总。

表 13-1 过程与函数之间的主要区别

	过 程	函 数
调用情况	只能使用 <i>CALL</i> 命令	可以嵌在所有的 SQL 命令里，比如在 <i>SELECT</i> 和 <i>UPDATE</i> 命令里
返回值	可以返回一个或多个 <i>SELECT</i> 结果	只能返回一个值（用 <i>RETURN</i> 命令）；返回值的数据类型必须在定义 SP 时用关键字 <i>RETURNS</i> 做出声明
参数	值参数和引用参数均可使用 (<i>IN</i> 、 <i>OUT</i> 、 <i>INOUT</i>)	只允许使用值参数，不允许使用 <i>IN</i> 等关键字
SP 代码中的可用命令	包括 <i>SELECT</i> 、 <i>INSERT</i> 、 <i>UPDATE</i> 、 <i>DELETE</i> 、 <i>CREATE TABLE</i> 等在内的所有的 SQL 命令	不允许使用那些会访问数据表的 SQL 命令
调用其他函数和过程	允许调用其他的过程和函数	只允许调用函数，不允许调用过程

13.5.1 基本语法规则

下面是 SP 最重要的语法规则：

- **分号 (;)。**同一个 SP 可以包含任意多条 SQL 命令。这些命令必须用分号隔开，就连分支和循环的控制结构也必须用分号结束。
- **BEGIN-END。**没有落在 SP 关键字之间（如 *THEN* 和 *END IF* 之间）的多条 SQL 命令必须放在关键字 *BEGIN* 和 *END* 之间。这意味着由多条 SQL 命令构成的 SP 的代码都必须以 *BEGIN* 开始、以 *END* 结束。
- **换行符。**换行符在 SP 代码中的语义效果与空格字符相同。这意味着把 *IF-THEN-ELSE- END-IF* 结构连续写在同一行上或分开写在多个行上都是可以的。
- **变量。**供 SP 内部使用的局部变量和局部参数不加“@”前缀。在 SP 内允许使用普通的 SQL 变量，但它们必须加“@”前缀。
- **字母大小写情况。**SP 在定义和调用时均不区分字母大小写情况，它写成（比如说）*shorten*、

SHORTEN 或 *Shorten* 的效果是一样的。

由于这个原因，不可能用同一个名字的不同大小写形式定义出多个函数或多个过程。

不过，一个函数可以和另一个过程的名字相同——MySQL 能够根据函数或过程的调用语法判断出应该调用哪一个。

□ **特殊字符。**在 SP 的名字里应避免使用 ä、à、ñ 等特殊字符。MySQL 允许这些字符出现在定义 SP 的 *CREATE* 命令里，但名字里有特殊字符的过程和函数在管理和使用时会带来很多不便。

□ **注释。**注释以两个连字符（--）开始并一直延续到这一行的末尾。

13.5.2 调用 SP (CALL)

函数和过程的调用办法是不同的。SP 与数据库相关联。如果想执行另一个数据库里的某个 SP，就必须在那个 SP 的名字前面加上数据库的名字作为前缀；如 *CALL database.spname()*。

1. 函数的调用

和 SQL 语言里的函数一样，函数也可以嵌在普通的 SQL 命令里。比如说，在 13.2 节里编写的 *shorten()* 函数可以像下面这样来调用：

```
SELECT shorten("a very long character string", 15)
SELECT shorten(title, 30), shorten(subtitle, 20) FROM titles
UPDATE titles SET title = shorten(title, 70)
WHERE CHAR_LENGTH(title)>70
```

可以用 *SET* 或 *SELECT INTO* 命令把函数的返回结果存入一个 SQL 变量，并在以后的 SQL 命令里使用这个变量：

```
SET @s = " a very long character string "
SET @a = shorten(@s, 15)
SELECT shorten(@s, 10) INTO @b
SELECT @s, @a, @b
@s          @a          @b
a very long character string a ver ... ring a very lon
```

2. 过程的调用

过程必须用 *CALL* 命令来调用。可以返回一个数据表作为过程的调用结果（就像一条 *SELECT* 命令那样），除此之外没有其他办法能把过程与 SQL 命令联系起来。

后面的示例使用了过程 *get_title()* 和 *half()*，下面是它们的定义：

```
PROCEDURE mylibrary.get_title(IN id INT)
BEGIN
    SELECT title, subtitle, publName
    FROM titles, publishers
    WHERE titleID=id
        AND titles.publID = publishers.publID;
END
PROCEDURE mylibrary.half(IN a INT, OUT b INT)
BEGIN
    SET b = a/2;
END
```

get_title() 的行为就像是一条 *SELECT* 命令，这并不令人感到奇怪，因为它就是这么定义出来的：

```
CALL get_title(1)
title      subtitle           publName
Linux Installation, Konfiguration, Anwendung Addison-Wesley
```

如果过程有引用参数 (*OUT* 或 *INOUT*)，它的返回结果将只能通过在调用时给它传递一个 SQL 变量的办法来使用。如下所示：

```
CALL half(10, @result)
SELECT @result
result
5
```

3. 递归

SP 可以调用其他 SP，甚至可以调用它们自己（递归）。不过，因为递归在 MySQL 5 系列的早期版本里引起了许多问题，所以 MySQL 5.0.9 版彻底禁用了递归功能，但 MySQL 5.0.17 版又部分地激活了它。目前，递归只允许在过程里使用，不允许在函数里使用。下面这个递归函数的例子可以在 MySQL 5.0.8 上运行，在最新的 MySQL 版本上却无法工作。它能不能在未来的 MySQL 版本里使用还是个未知数。

```
CREATE FUNCTION factorial(n BIGINT) RETURNS BIGINT
BEGIN
    IF n>=2 THEN
        RETURN n * factorial(n-1);
    ELSE
        RETURN n;
    END IF;
END
SELECT factorial(6)
factorial(6)
720
```

13.5.3 参数和返回值

函数和过程在声明参数和返回结果时使用的语法规则有点儿不同寻常。本节将对有关的语法变体进行讨论。

1. 过程的参数

过程必须使用 *CREATE PROCEDURE* 命令来创建，它们可以有、也可以没有参数表。请注意，即使定义的过程没有任何参数，也必须在它们的名字后面加上一对圆括号。

```
CREATE PROCEDURE name ([parameterlist])
[options] sqlcode
```

如果有一个以上参数，它们必须用逗号隔开。每个参数都要使用如下所示的语法来定义：

```
[IN or OUT or INOUT] parametername datatype
```

关键字 *IN*、*OUT* 和 *INOUT* 用来区分有关参数的用途是仅限于输入数据、仅限于输出数据、还是输入输出数据均可。（这几个关键字的默认设置是 *IN*。）

参数可以是 MySQL 所支持的任何一种数据类型（比如 *INT*、*VARCHAR(n)* 和 *DOUBLE*），但无法为它们提供有关数据类型的附加属性（比如 *NULL* 或 *NOT NULL*），这与为某个数据表定义一个数据列时的情况是不同的。就目前而言，MySQL 在传递参数时不会进行类型检查，但未来版本可能会改变这一做法。

注意 在给参数起名字的时候，一定要注意把它们与给数据表和数据列起的名字区分开。这有助于避免过程代码里的SQL命令出现二义性问题。

2. 过程的调用结果 (SELECT)

与函数不同，过程本身没有单个的返回值。不过，在过程里可以使用普通的 *SELECT* 命令。甚至可以连续使用多条 *SELECT* 命令，而这将使过程返回多个结果数据表。请注意，只有支持 *MULTI_RESULT* 的模式才能对多个结果进行处理——如果使用的是 PHP，就必须使用 *mysqli* 接口的 *multi_query()* 方法。

3. 函数的参数

用来创建新函数的 *CREATE* 命令与用来创建新过程的很相似：

```
CREATE FUNCTION name ([parameterlist]) RETURNS datatype  
[options] sqlcode
```

这里的最大区别是函数不支持引用参数，即关键字 *IN*、*OUT* 和 *INOUT* 不允许出现在函数的参数表里。

4. 函数的返回值

函数可以用 *RETURN* 命令返回一个值，这条命令同时也将结束函数的运行。*RETURN* 命令只能用在函数里，不能用在过程里。在定义一个函数的时候，必须在它的参数表里用关键字 *RETURNS* 对它的返回值的数据类型做出声明。

13.5.4 命令的封装 (BEGIN - END)

由一条以上的 SQL 命令构成的函数或过程必须以关键字 *BEGIN* 开头、以关键字 *END* 结束。此外，在代码内有时也需要使用 *BEGIN - END* 结构，比如在 *IF* 条件语句或循环语句里声明局部变量、条件、出错处理句柄或光标的时候。（出错处理句柄和光标将在后面的内容里讨论。）

在 *BEGIN - END* 语句块的内部，各有关语句或命令必须按照以下顺序写出：

```
BEGIN  
DECLARE variables;  
DECLARE cursors;  
DECLARE conditions;  
DECLARE handler;  
other SQL commands;  
END;
```

在关键字 *BEGIN* 的前面可以加上一个可选的标号，此时必须在与之对应的 *END* 关键字后面也写出这个标号。给一个 *BEGIN - END* 语句块加上标号的目的般是为了使用 *LEAVE* 命令提前退出这个语句块。下面是这种语句块结构的语法：

```
blockname: BEGIN  
  commands;  
  IF condition THEN LEAVE blockname; END IF;  
  further commands;  
END blockname;
```

1. 变量

SP 代码里的变量可以分为两大类：

□ **全局性的普通 SQL 变量**。这类变量的名字前面都有一个“@”字符作为前缀。这些变量在 SP 里的用法与它们在普通 SQL 命令里的用法一样（参见第 10 章）。它们的内容将一直保存到与 MySQL 服务器断开连接为止。

□ **SP 内部使用的局部变量和参数**。这些变量的名字前面没有“@”字符。必须在使用它们之前用 *DECLARE* 命令对它们做出声明。局部变量的内容在过程或函数退出执行时即告丢失。

局部变量只在对它们做出声明的那个 *BEGIN - END* 语句块里有效。这意味着可以在同一个过程中声明多个同名的局部变量——只要它们不在同一个 *BEGIN - END* 语句块里就行（请参见

下面的例子)。在过程的递归调用中, 过程的每一个实例都有它自己的变量, 不会与其他实例发生冲突(就像在 PHP 里那样)。

在定义的 SP 里使用什么类型的变量完全取决于自己, 但为了避免副作用, 还是尽量使用局部变量为好。全局性的普通 SQL 变量一般只用在递归过程或递归函数(请注意, 最新版本的 MySQL 只允许过程进行递归, 递归函数只能在一些早期的 MySQL 版本里使用, 参见前面的“递归”小节)里传递一些在递归时不允许或不需要发生变化的值。

2. 声明 (DECLARE)

变量的声明必须发生在 *BEGIN - END* 语句块里, 而且必须发生同一个 *BEGIN - END* 语句块里的其他命令之前。下面是变量声明的语法:

```
DECLARE varname1, varname2, ... datatype [DEFAULT value];
```

注意, 必须为所有的局部变量声明它们的数据类型。如果没有用关键字 *DEFAULT* 另行设定, 局部变量的初始值都将是 *NULL*。

注意 在给变量起名字的时候, 最好不要让它们与同一个 SP 里用到的数据列或数据表的名字相同。这在语法上是允许的, 但在实际工作中却往往会导致一些非常难以查找的错误。(查找这类错误的办法之一是检查 SP 里有没有无论怎样赋值都是 *NULL* 的变量。)

下面的例子演示了局部变量的定义层次。这里总共使用了 3 个变量, 它们的名字都是 *x*, 但因为它们是在不同的代码层次里声明的, 所以彼此不会发生冲突。当调用这个过程的时候, 它将依次返回 2、1、0 共 3 个结果。

```
CREATE PROCEDURE test()
BEGIN
    DECLARE x INT DEFAULT 0;
    BEGIN
        DECLARE x INT DEFAULT 1;
        IF TRUE THEN
            BEGIN
                DECLARE x INT DEFAULT 2;
                SELECT x;
            END;
        END IF;
        SELECT x;
    END;
    SELECT x;
END
```

3. 对变量赋值

SQL 语言不允许以 *x=x+1* 的形式对变量进行赋值, 必须使用 *SET* 或 *SELECT INTO* 命令来做这件事。后者是 *SELECT* 命令的一种变体, 它以 *INTO varname* 结束整条命令。这种语法变体要求 *SELECT* 命令返回且只能返回一条数据记录(不允许是多条记录)。请注意, 在函数里只能使用 *SET* 命令对变量进行赋值, 这是因为在函数里不允许使用 *SELECT* 或其他 SQL 命令。

```
SET var1=value1, var2=value2, var3=value3 ...
SELECT var:=value
SELECT 2*7 INTO var
SELECT COUNT(*) FROM table WHERE condition INTO var
SELECT title, subtitle FROM titles WHERE titleID=...
INTO mytitle, mysubtitle
```

13.5.5 分支

1. IF-THEN-ELSE 分支

IF 分支的 SQL 语法如下所示：

```
IF condition THEN
    commands;
[ELSE IF condition THEN
    commands;]
[ELSE
    commands;]
END IF;
```

正如前面提到过的那样，关键字 *BEGIN* 和 *END* 对一个控制结构来说不是必不可少的。*IF* 语句里的条件表达式可以像 *SELECT* 查询命令里的 *WHERE* 或 *HAVING* 条件表达式那样来构造。

除了这里给出的 *IF* 语句，还可以使用在第 10 章介绍的 *IF* 函数。

2. CASE 分支

CASE 语句是 *IF* 语句的一种语法变体，它们特别适合用在需要根据同一个表达式的不同取值来决定将执行哪一个分支的场合。下面是 *CASE* 分支的语法：

```
CASE expression
WHEN value1 THEN
    commands;
[WHEN value2 THEN
    commands;]
[ELSE
    commands;]
END CASE;
```

13.5.6 循环

MySQL 提供了好几种可以用来构造一个循环结构的语法变体，未来还有可能再增加一种可以用来构造 *FOR* 循环的语法变体。*FOR* 循环在 MySQL 5.0.3 里没有实现，它的语法也未见于有关文档。

1. REPEAT-UNTIL 循环

在这种循环结构里，关键字 *REPEAT* 和 *UNTIL* 之间的语句将一直循环执行到给定条件第一次得到满足为止。因为对条件表达式的求值发生在每次循环的末尾，所以整个循环至少会执行一次。

这种循环可以有一个可选的标号，此时必须在整个循环语句的末尾也写出同样的标号。给一个循环语句加上标号的一般是为了使用 *LEAVE* 命令提前退出整个循环，或者是为了使用 *ITERATE* 命令把循环体里的命令再执行一遍（这两个命令马上就会讲到）。

```
[loopname:] REPEAT
    commands;
UNTIL condition
END REPEAT [loopname];
```

下面是一个 *REPEAT-UNTIL* 循环的例子。这里的 *test(n)* 函数将返回一个包含着 *n* 个 “*” 字符的字符串。（想构造一个这样的字符串当然还有更简便的办法，这里只是为了演示这种语法。）

```
CREATE FUNCTION test(n INT) RETURNS TEXT
BEGIN
DECLARE i INT DEFAULT 0;
DECLARE s TEXT DEFAULT '';
myloop: REPEAT
    SET i = i+1;
    SET s = CONCAT(s, "*");
UNTIL i>=n END REPEAT;
```

```

    RETURN s;
END
SELECT test(5)
test(5)
*****

```

2. WHILE 循环

在这种循环结构里，关键字 *DO* 和 *END WHILE* 之间的语句将一直循环执行到给定条件第一次没有得到满足为止。因为对条件表达式的求值发生在每次循环的开始，所以如果给定条件在第一次求值的时候就没有得到满足的话，整个循环将一次也不执行。如果打算在一个 *WHILE* 循环里使用 *LEAVE* 和/或 *ITERATE* 命令，还必须给这个循环加上一个标号。

```
[loopname:] WHILE condition DO
    commands;
END WHILE [loopname];
```

3. LOOP 循环

在这种循环结构里，关键字 *LOOP* 和 *END LOOP* 之间的语句将一直循环执行到遇见一条 *LEAVE loopname* 命令并因此而退出整个循环为止。*LOOP* 循环的语法不要求必须给它们加上一个标号，但在实际运用中它们几乎总是有标号的（除非真的想创建一个无限循环）。

```
loopname: LOOP
    commands;
END LOOP loopname;
```

下面是一个 *LOOP* 循环例子。这里的 *test(n)* 函数也将返回一个包含着 *n* 个“*”字符的字符串。

```

CREATE FUNCTION test (n INT) RETURNS TEXT
BEGIN
    DECLARE i INT DEFAULT 0;
    DECLARE s TEXT DEFAULT '';
    myloop: LOOP
        SET i = i+1;
        SET s = CONCAT(s, "*");
        IF i>=n THEN LEAVE myloop; END IF;
    END LOOP myloop;
    RETURN s;
END

```

4. LEAVE 和 ITERATE 语句

LEAVE loopname 命令将使程序代码的执行流程跳出一个循环。*LEAVE* 命令还可以用来提前退出 *BEGIN-END* 语句块。

ITERATE loopname 命令的效果是把循环体里的命令再执行一遍。*ITERATE* 命令不能像 *LEAVE* 命令那样在 *BEGIN-END* 语句块里使用。

13.5.7 出错处理（出错处理句柄）

SP 里的 SQL 命令在执行过程中可能会出错，所以 SQL 也像其他一些程序设计语言那样向程序员提供了一种利用出错处理句柄（error handler，也有人称为“出错处理器”）来响应和处理这类错误的机制。

在一个 *BEGIN-END* 语句块里，对出错处理句柄的定义必须出现在变量、光标、出错条件的声明之后、其他 SQL 命令之前；具体语法如下所示：

```
DECLARE type HANDLER FOR condition1, condition2, condition3, ... command;
```

下面是对这个语法中的 *type*、*condition* 和 *command* 的解释。

- *type* (类型)。可供选用的类型目前只有 *CONTINUE* 和 *EXIT* 两种。(未来的 MySQL 版本可能会增加第 3 种选择: *UNDO*)。*CONTINUE* 的含义是: 如果当前命令在执行时发生错误, 从下一条命令继续执行。*EXIT* 的含义是: 如果有命令执行出错, 则退出当前 *BEGIN - END* 语句块并从该语句块之后的第一条命令继续执行。
- *condition* (条件)。这里可以列出一个或多个出错处理条件, 它们是出错处理句柄被调用的前提。出错处理条件可以用以下几种方式给出:
 - *SQLSTATE 'errorcode'*。单个 SQL 出错代码, 它的编号是 *errorcode*。
 - *SQLWARNING*。涵盖了 *SQLSTATE* 编号为 *0Innn* 的所有错误。
 - *NOT FOUND*。涵盖了所有其他的(即 *SQLSTATE* 编号不以 *01* 或 *02* 开头的)的错误。¹
 - *mysqlerrorcode*。这个数字是一个 MySQL 出错代码而不是一个 *SQLSTATE* 出错代码。
 - *conditionname*。一个用 *DECLARE CONDITION* 命令定义出来的出错处理条件, *conditionname* 是那个条件的名字(参见 13.5.8 节)。
- *command* (命令)。执行出错时将要执行的命令。因为这里只允许放上一条命令, 所以它通常是一个变量赋值命令, 该变量将在后续的出错处理代码中使用。这里必须给出一条命令(在 *DECLARE EXIT HANDLER* 的时候也不例外)。

提示 在 13.5.8 节里有一个出错处理的例子。MySQL 出错代码与 SQL 出错代码(*SQLSTATE* 编号)的区别是: 前者是 MySQL 对各种错误的编号, 后者是 SQL 语言中的标准出错编号。比如说, *No data to fetch*(没有可取回的数据)错误的 MySQL 出错代码是 1329, *SQLSTATE* 编号值是 '*02000*'。MySQL 出错代码的完整清单和对应的 *SQLSTATE* 值可以在 MySQL 在线文档里查到:
<http://dev.mysql.com/doc/mysql/en/error-handling.html>。

1. 条件

可以用 *DECLARE CONDITION* 命令给出错处理条件定义一个简明易记的名字。这些定义必须出现在出错处理句柄的声明之前, 定义出来的条件名可以用在出错处理句柄里。下面是出错处理条件的定义语法:

```
DECLARE name CONDITION FOR condition;
```

出错处理条件(语法中的 *condition*)可以用 *SQLSTATE 'errorcode'* 或 *mysqlerrorcode* 的形式写出(参见 13.5.6 节)。在下面的例子里, 为 *duplicate key*(键值重复)错误定义了一个变量、一个出错处理条件和一个出错处理句柄:

```
DECLARE dupkey VARCHAR(100);
DECLARE duplicate_key CONDITION FOR SQLSTATE '23000';
DECLARE CONTINUE HANDLER FOR duplicate_key SET myerror='dupkey';
```

2. 触发错误

有时候, 在某种特定情况出现时(比如说, 在检查出一个非法参数时), 有意识地在 SP 代码里触发一个错误可以简化编程工作。IBM DB/2 为此专门提供了一个 SQL 命令 *SIGNAL SQLSTATE xxx*, 但在 MySQL 里不存在这样的命令(至少目前如此)。

眼下的唯一办法是故意执行一条非法的命令, 比如说, 在过程里, 可以故意执行一条 *SELECT*

1. 这里与后面 13.5.8 节里的说法矛盾!! ——译者注

'errorport'或`SET 'errorport'=0`命令。MySQL会把反引号的这种用法解释为数据库、数据表或数据列的名字，所以这条命令将触发*Unknown column 'errorport' in 'field list'*错误。这种出错消息可能会让别人感到困惑，但执行者肯定知道它意味着什么和应该如何去处理。这个办法的好处是定制的出错消息('errorport')将成为MySQL出错消息的一部分，别人在看到这样的出错消息时或许会感到困惑，但执行者肯定知道它意味着什么和应该如何去处理。

因为在函数里不允许执行`SELECT`命令，所以需要把这个办法变通一下：故意执行一条`RETURN 'errorport'`命令，这条命令将触发*Unknown column 'errorport' in 'order clause'*错误。

3. 检查SP代码中的错误

出错处理句柄和出错处理条件可以帮助人们在SP代码执行出错时避免更大的损失，但这里还有一个大前提：SP代码本身没有从开发阶段遗留下来的隐患。因为MySQL没有调试器之类的查错工具，出错报告也帮不上多大的忙，所以想把SP代码里的编程错误查找出来还是有相当难度的。下面这个办法只适用于过程：如果想了解某个变量在SP执行期间的变化情况或是语句/代码段的执行顺序，可以在SP代码的适当位置插入一些`SELECT`命令去输出这个变量或一段文字（比如说，在某个循环结构里插入一条`SELECT varname`命令）。

13.5.8 光标

说起光标，读者肯定会想到鼠标和文本文档中的当前输入位置。但在数据库领域里，光标还有另一层含义：一个指向数据表里的数据记录的指针。这种光标向人们提供了遍历数据表里的全部数据记录的可能性。光标的典型应用包括：有选择地修改某些数据记录、有选择地把某些数据记录拷贝到另一个数据表里等。从更广泛的意义上讲，需要对数据表里的个别记录进行个别修改而总的修改量又比较大的场合都可以使用光标，而使用`UPDATE`命令来完成这类工作会相当吃力。

在是否使用光标的问题上有两种意见。支持者认为光标是一种实用的工具，它可以帮助人们非常轻松地解决一些用其他办法很难解决的问题。反对者则认为使用光标是一种不好的编程习惯，并强调几乎所有的问题都有更精巧、更高效的非光标解决方案。可话又说回来了，是否使用光标是个人的事，没有人能替你做出选择。笔者在这里只想提醒大家一句：如果需要处理的数据量很大，千万不要想当然地认为使用了光标的解决方案肯定会比不使用光标的解决方案更有效率，应该亲身对这两种解决方案进行过测试之后再做决定。

1. 语法

使用光标需要经历以下几个步骤：首先，用`DECLARE cursorname CURSOR FOR SELECT ...`对光标做出声明。这里允许使用任何一种`SELECT`命令，它的查询结果就是新光标将要访问的数据表。然后，必须用`OPEN cursorname`命令启用这个光标。

接下来，就可以使用`FETCH cursorname INTO var1, var2, ...`命令去遍历`SELECT`结果数据表里的数据记录了：数据记录的第一个字段将被读入变量`var1`，第二个字段将被读入变量`var2`，依此类推。（各有关变量必须提前做出声明，而且必须是正确的数据类型。请注意，变量和字段的名字必须不同，否则会导致难以预料的后果。）

现在，问题来了：如果`FETCH`命令把`SELECT`结果数据表里的记录都读完了会发生什么事？答案很简单：这将触发一个1329号MySQL错误*No data to fetch*（没有可取回的数据），相应的SQLSTATE编号是02000。这个错误无法回避，但可以用一个出错处理器来捕获（参见上一小节）。因此，每一个使用了光标的SP都必须要有相应的出错处理代码才行。这里的出错处理条件一般使用`NOT FOUND`

即可，它涵盖了 *SQLSTATE* 编号是 02nnn 的所有错误¹。

光标可以用 *CLOSE cursorname* 命令关闭。但这么做没有什么必要，因为光标会在对它们做出声明的 *BEGIN-END* 语句块执行终了时自动随之终结。

2. 不足与缺陷

MySQL 5.0 版对光标的使用有以下限制：

- 光标是只读的。也就是说，光标只能用来读取数据，不能用来修改数据。
- 光标只能前进。也就是说，光标只能按照有关数据被 MySQL 服务器发送过来的先后顺序依次对它们进行处理。
- 光标是敏感的。也就是说，在使用某个光标读取数据的同时，这个光标所涉及的数据表都不允许发生任何变化。如果非要那么做，MySQL 服务器可能出现难以预料的行为。

3. 示例

下面是一个简单的光标应用示例。示例中的过程先遍历 *titles* 数据表把图书名称（*title* 数据列）和图书副标题（*subtitle* 数据列）的字符总数统计出来，然后把这个总数除以 *titles* 数据表里的记录总数，得出图书名称和副标题的平均长度。代码中的光标命令和出错处理部分用黑体字突出显示：

```
CREATE PROCEDURE mylibrary.cursortest(OUT avg_len DOUBLE)
BEGIN
    DECLARE t, subt VARCHAR(100);
    DECLARE done INT DEFAULT 0;
    DECLARE n BIGINT DEFAULT 0;
    DECLARE cnt INT;
    DECLARE mycursor CURSOR FOR
        SELECT title, subtitle FROM titles;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1;
    SELECT COUNT(*) FROM titles INTO cnt;
    OPEN mycursor;
    myloop: LOOP
        FETCH mycursor INTO t, subt;
        IF done=1 THEN LEAVE myloop; END IF;
        SET n = n + CHAR_LENGTH(t);
        IF NOT ISNULL(subt) THEN
            SET n = n + CHAR_LENGTH(subt);
        END IF;
    END LOOP myloop;
    SET avg_len = n/cnt;
END
```

下面第一条命令调用这个过程，第二条命令查看调用结果；调用结果通过引用参数@*result* 变量返回：

```
CALL cursortest(@result)
SELECT @result
@result
29.47
```

这个结果不用 SP 和光标也可以获得，下面这条等效的 SQL 命令既简明又更高效：

```
SELECT (SUM(CHAR_LENGTH(title)) + SUM(CHAR_LENGTH(subtitle))) / COUNT(*) AS avg_len
FROM titles
avg_len
29.47
```

1. 这里与前面 13.5.7 节里的说法矛盾！！——译者注

13.6 SP 应用示例

本节将通过几个示例来演示 SP 的应用可能性, 它们都是在第 8 章介绍的 *mylibrary* 数据库上实现的。

13.6.1 增加新的图书门类

下面给出的过程将在 *categories* 数据表里增加一个新的图书门类。这个过程的两个输入参数分别是新门类的名字和 *parentCatID* 编号; 它的输出参数将返回新门类的 *CatID* 编号, 这样一来, 用户就不必在调用完这个过程之后再用其他办法去确定 *last_insert_id* 了。如果用户传递给这个过程的数据是非法的(空字符串、*NULL*、不正确的 *parentCatID* 编号), 这个过程的输出参数将返回-1。¹ 如果 *parentCatID* 参数值所对应的图书门类下已经有了一个与新门类名字相同的门类, 这个过程将返回那个现有门类的 *CatID* 而不是增加一个新门类。

这个例子还证明了这样一件事: 与使用一条简单的 *INSERT* 命令相比, 使用一个设计良好的 SP 来插入新记录可以一次完成更多的事情。如果是一名程序员, 可以根据应用项目的具体情况用一个 SP 完成所有必要的检查和测试, 例如检查各有关数据是否都落在它们各自的取值范围内等。如果是一名数据库管理员, 可以收回用户的 *Insert* 权限并限制他们只能使用一个 SP 来插入所有的数据, 这将使数据表里的错误记录大大减少。(当然, 还应该准备一个可以代替 *UPDATE* 命令的 SP, 以减少因为修改不当而导致的错误记录。)

下面的过程由一些普通的 SQL 命令和几个 *IF* 语句构成, 它应该很容易理解。我们给整个 *BEGIN-END* 语句块起了一个名字叫 *proc*, 这样我们就可以用 *LEAVE proc* 命令退出代码执行了。在笔者测试的 MySQL 版本里, 系统变量 *@@last_insert_id* 的取值是陈旧过时的, *LAST_INSERT_ID()* 函数的返回值完全正确。

```
CREATE PROCEDURE categories_insert
    (IN newcatname VARCHAR(60), IN parent INT, OUT newid INT)
proc: BEGIN
    DECLARE cnt INT;
    SET newid=-1;
    -- validation
    SELECT COUNT(*) FROM categories
    WHERE parentCatID=parent INTO cnt;
    IF ISNULL(newcatname) OR TRIM(newcatname)="" OR cnt=0 THEN
        LEAVE proc;
    END IF;
    -- Test whether the category already exists
    SELECT COUNT(*) FROM categories
    WHERE parentCatID=parent AND catName=newcatname
    INTO cnt;
    IF cnt>1 THEN
        -- determine the existing catID number
        SELECT catID FROM categories
        WHERE parentCatID=parent AND catName=newcatname
        INTO newid;
        LEAVE proc;
    END IF;
    -- insert new category
    INSERT INTO categories (catName, parentCatID)
    VALUES (newcatname, parent);
```

1. 原书刚在前面的 13.5.3 节提到过程没有返回值, 这里就用了一个 *return value*, 所以我们把它改译为现在这样。

——译者注

```

SET newid = LAST_INSERT_ID();
END proc

```

下面是调用这个过程的情况：

```

The procedure is used thus:
CALL categories_insert('Microsoft Access', 2, @catid)
SELECT @catid
@catid
90

```

13.6.2 增加一本新图书

本节给出的过程可以把一本新图书连同它的作者和出版公司一块儿插入 *mylibrary* 数据库。如果新书的作者不止一位，他们的名字必须用分号隔开。下面是过程 *titles_insert_all()* 的调用命令：

```

CALL titles_insert_all(
    'Programmieren mit der .NET-Klassenbibliothek',
    'Addison-Wesley', 'Eller Frank;Schwichtenberg Holger', @newID)

```

这个过程涉及 4 个数据表：把图书信息存入 *titles* 数据表，把作者信息存入 *authors* 数据表，把出版公司信息存入 *publishers* 数据表（如果数据库里还没有收录过这家出版公司的话），把新图书与其作者之间的交叉引用关系存入 *rel_title_author* 数据表。

过程 *titles_insert_all()* 能够识别姓氏和名字颠倒了的已收录作者，例如，它能判断出 *Stephen King* 和 *King Stephen* 其实是同一位作者。这一点是通过把现有作者与 *author* 和 *swap_name(author)* 进行比较做到的。（函数 *swap_name()* 的代码在程序清单的尾部。）

这个预防措施可以保证 *authors* 数据表不会出现明显重复的人名（例如一个是 *Stephen King*，另一个是 *King Stephen*）。人为地把作者姓名的正确格式规定为姓氏在前、名字在后，这在以人工方式插入新记录的时候根本不是什么问题，但要求这个 SP 也具备这种智能显然不现实。不过，虽然它不能保证作者姓名是按正确格式插入的，但以后修改起来并不困难。（如果有一部分图书的作者姓名顺序正确，另一部分图书的姓名顺序不正确，修改起来就比较麻烦了。）

当然，值得改进的地方还有很多，比如说，还可以增加一些参数来处理 *titles* 数据表里的其他数据列，还可以对参数进行更严格的检查等。但总的来说，这个过程的思路还是很清晰的。

SP 代码比较复杂和难以阅读的特点在这个例子里也表现得很明显。如果用 PHP 脚本来完成同样工作的话，程序清单的长度大概和这里的差不多，但在易于理解和便于长期维护等方面却要高出一筹。（这也正是为什么会有那么多的数据库管理员反对滥用 SP 的原因。对大量的 SP 进行维护绝不是件轻松的事。）

```

CREATE PROCEDURE titles_insert_all
    (IN newtitle VARCHAR(100), IN publ VARCHAR(60),
     IN authList VARCHAR(255), OUT newID INT)
proc: BEGIN
    DECLARE cnt, pos INT;
    DECLARE aID, pblID, ttlID INT;
    DECLARE author VARCHAR(60);
    SET newID=-1;
    ~ Search for/store publisher
    SELECT COUNT(*) FROM publishers WHERE publname=publ INTO cnt;
    IF cnt=1 THEN
        SELECT publID FROM publishers WHERE publname=publ INTO pblID;
    ELSE
        INSERT INTO publishers (publName) VALUES (publ);
        SET pblID = LAST_INSERT_ID();
    END IF;

```

```

- store the title
INSERT INTO titles (title, publID) VALUES (newtitle, pblID);
SET ttlID = LAST_INSERT_ID();
- loop over all authors in authList
authloop: WHILE NOT (authList="") DO
    SET pos = LOCATE(";", authList);
    IF pos=0 THEN
        SET author = TRIM(authList);
        SET authList ="";
    ELSE
        SET author = TRIM(LEFT(authList, pos-1));
        SET authList = SUBSTR(authList, pos+1);
    END IF;
    IF author = "" THEN ITERATE authloop; END IF;
    - search for/store author
    SELECT COUNT(*) FROM authors
    WHERE authName=author OR authName=swap_name(author)
    INTO cnt;
    IF cnt>=1 THEN
        SELECT authID FROM authors
        WHERE authName=author OR authName=swap_name(author)
        LIMIT 1 INTO aID;
    ELSE
        INSERT INTO authors (authName) VALUES (author);
        SET aID = LAST_INSERT_ID();
    END IF;
    - update rel_title_authors
    INSERT INTO rel_title_author (titleID, authID)
    VALUES (ttlID, aID);
END WHILE authloop;
- return value
SET newID=ttlID;
END proc
CREATE FUNCTION swap_name(s VARCHAR(100)) RETURNS VARCHAR(100)
BEGIN
    DECLARE pos, clen INT;
    SET s = TRIM(s);
    SET clen = CHAR_LENGTH(s);
    SET pos = LOCATE(" ", REVERSE(s));
    IF pos = 0 THEN RETURN s; END IF;
    SET pos = clen-pos;
    RETURN CONCAT(SUBSTR(s, pos+2), " ", LEFT(s, pos));
END

```

13.6.3 确定父门类

SP 过程 *get_parent_categories()* 的输入参数是一个图书门类的 *CatID* 编号, 返回结果是一个数据表, 其内容是这个图书门类的所有父门类。我们来看一个例子:

```

CALL get_parent_categories(57)
catID  catName
11  All books
1  Computer books
56  Operating Systems
57  Linux

```

这个 SP 过程将创建一个 *HEAP* 类型的临时数据表 *_parent_cats*(这种类型的数据表存在于计算机的内存中, 不占用硬盘空间) 来保存给定图书门类的所有父门类, 它的 *level* 数据列存放着各有关门类在层次结构中的级别。结果数据表按逆序排序 (*ORDER BY ... DESC*)。

这个 SP 函数将首先检查输入参数 *startid* 的值是否合法, 如果合法, 就把相应的图书门类数据读入 *id*、*pid* 和 *cname* 3 个变量, 然后再把计数器 *i* 以及 *id* 和 *cname* 插入刚才新建的临时数据表 *_parent_cats*。

接下来的循环语句将沿着父门类一直上溯到 *parentCatID* 字段的值是 *NULL* 为止。在上溯时查找到的数据将插入到 *_parent_cats* 数据表里去。

在循环结束之后，安排了一条 *SELECT* 命令来输出临时数据表 *_parent_cats* 的内容。这个输出也就是这个过程的调用结果。随后，已经完成其使命的临时数据表 *_parent_cats* 被删除了。

```
CREATE PROCEDURE get_parent_categories(startid INT)
BEGIN
    DECLARE i, id, pid, cnt INT DEFAULT 0;
    DECLARE cname VARCHAR(60);
    DROP TABLE IF EXISTS _parent_cats;
    CREATE TEMPORARY TABLE _parent_cats
        (level INT, catID INT, catname VARCHAR(60)) ENGINE = HEAP;
    main: BEGIN
        - check startid
        SELECT COUNT(*) FROM categories WHERE catID=startID INTO cnt;
        IF cnt=0 THEN LEAVE main; END IF;
        - insert the startid category into _parent_cats
        SELECT catID, parentCatID, catName
        FROM categories WHERE catID=startID
        INTO id, pid, cname;
        INSERT INTO _parent_cats VALUES(i, id, cname);
        - loop for searching for parent categories
        parentloop: WHILE NOT ISNULL(pid) DO
            SET i=i+1;
            SELECT catID, parentCatID, catName
            FROM categories WHERE catID=pid
            INTO id, pid, cname;
            INSERT INTO _parent_cats VALUES(i, id, cname);
        END WHILE parentloop;
    END main;
    SELECT catID, catname FROM _parent_cats ORDER BY level DESC;
    DROP TABLE _parent_cats;
END
```

请注意代码清单第 5 行上的 *DROP TABLE IF EXIST _parent_cats* 命令。如果没有 *IF EXIST* 关键字，这个过程在执行时就会触发一条警告消息，说 *DROP TABLE* 命令试图删除一个并不存在的数据表。¹

13.6.4 按层次结构生成图书门类清单

13.6.3 节的例子是为一个给定门类搜索所有的父门类。本节的例子则是为一个给定门类搜索所有的子门类。这个任务相对要复杂一些，因为子门类可以有分支。因此，过程 *get_subcategories()* 调用了一个递归的过程 *find_subcategories()*，后者负责具体完成为给定门类查找所有子门类的工作。

类似于刚才那个例子里的 *get_parent_subcategories()*, *get_subcategories()* 过程也使用了一个临时数据表 (*_subcats*) 来保存自己的结果。但与刚才不同的是，我们这一次不删除这个临时数据表，它将作为过程调用的结果保留下来。正如随后的几个例子所演示的那样，客户程序还可以用它来完成许多种查询。

给定的起始门类通过输入参数 *startid* 传递到 *get_subcategories()* 过程。这个过程将把找到的子门类的个数（包括起始门类在内）写入自己的第二个（输出）参数。

```
CALL get_subcategories(1, @result)
SELECT @result
@result
26
```

¹ 这里的译文与原文有较大出入，但原文显然与 SP 代码清单里的逻辑不匹配。——译者注

除了这个结果，*get_subcategories()*过程在调用结束后还留下了一个`_subcats`数据表，这个数据表的内容就是这个过程的查询结果。除了`catID`和`catName`数据列，`_subcats`数据表还有2个数据列：`rank`数据列存放着用来对数据进行排序的数据行编号；`level`数据列存放着各有关门类在层次结构里的级别（起始门类的`level=0`）。下面这条命令对各有关门类按照它们在层次结构中的位置进行了缩进：

```
SELECT rank, level, catID,
       CONCAT(SPACE(level*2), catname)
  FROM _subcats ORDER BY rank
  rank  level  catID  CONCAT(SPACE(level*2), catname)
    0      0      1  Computer books
    1      1      2  Databases
    2      2     69  IBM DB/2
    3      2     86  Microsoft Access
    4      2     67  Microsoft SQL Server
    5      2     34  MySQL
    6      2      5  Object-oriented databases
    7      2     68  Oracle
    8      2     77  PostgreSQL
    9      2      4  Relational Databases
   10      2      8  SQL
   11      1     36  LaTeX, TeX
   12      1     56  Operating Systems
   13      2     57  Linux
   14      2     58  Mac OS
   15      2     59  Windows
   16      1      3  Programming
   17      2     54  C
   18      2     53  C#
   19      2     55  C++
   20      2     50  Java
   21      2      7  Perl
   22      2      6  PHP
   23      2     52  VBA
   24      2     51  Visual Basic
   25      2     60  Visual Basic .NET
```

`_subcats`数据表还可以用来搜索这些门类里的图书：

```
SELECT title FROM titles, _subcats
 WHERE titles.catID = _subcats.catID
 ORDER BY title;
 title
A Guide to the SQL Standard
A Programmer's Introduction to PHP 4.0
Apache Webserver 2.0
Client/Server Survival Guide
...
```

如果不再需要使用`_subcats`数据表，用下面这条命令简单地把它删掉即可。（即使本人或是其他用户调用了`get_subcategories()`过程的用户忘了删除这个数据表，它也会在断开与MySQL服务器的连接时自动消失，所以不必担心服务器主机的内存被这些临时数据表无谓地占用。）

```
DROP TABLE _subcats
```

程序代码

`get_subcategories()`过程将先创建一个临时数据表`_subcats`，然后检查通过`startid`参数传递来的值是否合法，如果合法，就在`_subcats`数据表里插入第一条记录。接下来，它将调用`find_subcategories()`过程并在后者返回后用一条`SELECT`命令统计出`_subcats`数据表里的记录总数，这个总数将通过参数`n`返回。

```

CREATE PROCEDURE get_subcategories(IN startid INT, OUT n INT)
BEGIN
    DECLARE cnt INT;
    DECLARE cname VARCHAR(60);
    DROP TABLE IF EXISTS __subcats;
    CREATE TEMPORARY TABLE __subcats
        (rank INT, level INT, catID INT, catname VARCHAR(60))
        ENGINE = HEAP;
    SELECT COUNT(*) FROM categories WHERE catID=startID INTO cnt;
    IF cnt=1 THEN
        SELECT catname FROM categories WHERE catID=startID INTO cname;
        INSERT INTO __subcats VALUES(0, 0, startid, cname);
        CALL find_subcategories(startid, cname, 1, 0);
    END IF;
    SELECT COUNT(*) FROM __subcats INTO n;
END

```

*find_subcategories()*过程使用了一个光标来遍历所有的子门类。每找到一个门类，它就会在*__subcats*数据表里插入一条相应的记录并给 *INOUT* 参数 *catrank* 加上一个 1。接下来，它将递归地调用自己并在发出递归调用时给 *catlevel* 加上一个 1。

```

CREATE PROCEDURE find_subcategories
    (IN id INT, IN cname VARCHAR(60), IN catlevel INT,
     INOUT catrank INT)
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE subcats CURSOR FOR
        SELECT catID, catName FROM categories WHERE parentCatID=id
        ORDER BY catname;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1;

    OPEN subcats;
    subcatloop: LOOP
        FETCH subcats INTO id, cname;
        IF done=1 THEN LEAVE subcatloop; END IF;
        SET catrank = catrank+1;
        INSERT INTO __subcats VALUES (catrank, catlevel, id, cname);
        CALL find_subcategories(id, cname, catlevel+1, catrank);
    END LOOP subcatloop;
    CLOSE subcats;
END

```

13.7 触发器

触发器的用途是在 *INSERT*、*UPDATE* 和 *DELETE* 命令之前或之后自动调用 SQL 命令或 SP。比如说，可以为每一个 *UPDATE* 操作测试被修改的数据是否满足特定的条件。

本节只是对 MySQL 5.0 里的触发器实现的一个入门介绍。注意，MySQL 目前（MySQL 5.0.3）对触发器的实现还很不完善。与 SP 相比，触发器还远没有成熟到可以用于实际应用程序的地步。根据 MySQL 在线文档里的说法，MySQL 5.1 版本将提供更多触发器功能。在这个版本出现之前，只能用触发器完成一些很初级的任务，就像本节末尾给出的示例。

13.7.1 创建触发器

创建新触发器的命令是 *CREATE TRIGGER*。只有具备 *Super* 权限的 MySQL 用户才能执行这条命令。

```

CREATE TRIGGER name BEFORE|AFTER INSERT|UPDATE|DELETE
ON tablename FOR EACH ROW code

```

最多可以为同一个数据表定义 6 个触发器，即分别为 *INSERT*、*UPDATE* 或 *DELETE* 命令的前、后各定义一个。如果这些命令涉及到的数据记录不止一条，触发器将为每一条记录执行一遍。

从 MySQL 5.0.10 开始，触发器的名字在同一个数据库里必须独一无二。

触发器代码的语法规则与 SP 代码的语法规则很相似：比如说，如果触发器包含着的命令不止一条，它的代码也必须以关键字 *BEGIN* 开始、以关键字 *END* 结束。所有的 SP 语言元素都可以套用在触发器身上。不过，在允许人们在触发器代码里使用哪些 SQL 命令的问题上有许多限制（参见后面的“功能局限性”小节）。

在触发器代码里，可以通过以下方式去访问当前记录的各个字段：

- *OLD.columnname* 返回一条现有记录在被修改或删除之前的内容 (*UPDATE*、*DELETE*)。

- *NEW.columnname* 返回一条新记录或被修改记录的新内容 (*INSERT*、*UPDATE*)。

在 *BEFORE INSERT* 触发器和 *BEFORE UPDATE* 触发器里，可以改动 *NEW.columnname* 的内容数据。

13.7.2 删除触发器

删除触发器的命令是 *DROP TRIGGER*，这条命令不支持 *IF EXIST* 变体。

```
DROP TRIGGER [databasename.]triggername
```

与 SP 相比，MySQL 没有为触发器提供类似 *ALTER TRIGGER*、*SHOW CREATE TRIGGER* 或 *SHOW TRIGGER STATUS* 的管理性命令。MySQL 的未来版本是否会提供这些命令，以及在什么时候、会以什么样的语法来提供它们目前都是未知数。

13.7.3 实现细节和管理工具

MySQL 目前（MySQL 5.0.3）把触发器代码存放在数据库目录中的文本文件 *tablename.TRG* 里。这个位置很可能会在未来的 MySQL 版本里发生变化。触发器很可能会像 SP 那样被统一收录到 *mysql* 数据库中的一个数据表里去。

就目前而言，还没有一种管理工具可以用来管理触发器：MySQL Query Browser、MySQL Administrator、phpMyAdmin 等工具都无法对触发器进行管理。如果想执行 *CREATE TRIGGER* 命令，就必须使用 MySQL 自带的命令解释器 *mysql* 程序——与 SP 的情况一样，在创建触发器之前也必须另行指定一个命令结束符。

备份工具 *mysqldump* 程序也不支持触发器。如果想把数据库里现有的触发器备份下来，目前的唯一办法是对数据库目录里的*.TRG 文件进行备份。

13.7.4 功能局限性

MySQL 对触发器的使用有许多限制，这些限制使得触发器几乎没有什么实际用途：

- 在触发器代码里无法访问任何数据表，就连触发器为之定义的那个数据表也不能访问。类似于 SP 函数的情况，*SELECT*、*INSERT*、*UPDATE* 等命令都不允许在触发器代码里出现。
- MySQL 既没有提供可以用来在触发器里取消 *DELETE*、*UPDATE* 或 *INSERT* 命令的命令或语法元素，也没有提供可以用来在触发器里人为触发一个错误的命令或语法元素。
- 触发器代码不能用 *CALL* 命令调用一个 SP。
- 在触发器代码里不能调用事务命令。

13.7.5 触发器应用示例

下面给出的两个触发器示例可以确保用户只能把一个0~1之间的浮点数输入 *test* 数据表的 *percent* 数据列，超出这个范围的浮点数值将被替换为 *NULL*：

```
USE test
CREATE TABLE test (id SERIAL, percent DOUBLE)
DELIMITER $$ 
CREATE TRIGGER test_before_insert
BEFORE INSERT ON test FOR EACH ROW
BEGIN
IF NEW.percent < 0.0 OR NEW.percent > 1.0 THEN
SET NEW.percent = NULL;
END IF;
END$$
CREATE TRIGGER test_before_update
BEFORE UPDATE ON test FOR EACH ROW
BEGIN
IF NEW.percent < 0.0 OR NEW.percent > 1.0 THEN
SET NEW.percent = NULL;
END IF;
END$$
DELIMITER ;
```

下面的命令演示了这两个触发器的用法，它们在使用 *INSERT* 和 *UPDATE* 命令插入/修改有关数据的时候的确发挥了应有的作用：

```
INSERT INTO test (percent) VALUES (-1), (0.3), (1.5)
SELECT * FROM test
+-----+-----+
| id   | percent |
+-----+-----+
| 1    | NULL    |
| 2    | 0.3     |
| 3    | NULL    |
+-----+-----+
UPDATE test SET percent = 1.7 WHERE id = 2
SELECT * FROM test
+-----+-----+
| id   | percent |
+-----+-----+
| 1    | NULL    |
| 2    | NULL    |
| 3    | NULL    |
+-----+-----+
```

第 14 章

管理与服务器配置

本章的主要内容是如何管理和配置 MySQL 服务器。讨论重点包括备份、日志文件、把数据库从一台计算机移植到另一台计算机（或者从 MySQL 的老版本迁移到新版本），以及文本文件的导入/导出。本章还将探讨如何管理和维护 MyISAM 和 InnoDB 数据表。

这一章还涉及到了一些比较高级的论题，如果对如何建立镜像机制或如何获得更快的处理速度等问题感兴趣，本章提供了一些相关的介绍和技巧。

最后，本章还将探讨一个“痛苦”的话题：如何用好 ISP（Internet Service Provider，因特网服务提供商）提供的各种管理选项。（之所以说这是一个“痛苦”的话题，是因为此时需要假设用户不是一名数据库管理员，因而只有非常有限的访问权限。）

提示 本书其他章节也对 MySQL 数据库系统管理工作的一些方面进行了讨论，它们是：第 4 章到第 6 章，介绍如何使用各种客户工具程序（如 mysql、MySQL Administrator 等）去完成数据库管理任务；第 11 章，介绍 MySQL 数据库系统的信息安防和用户管理问题；第 22 章，介绍 mysqld、mysql 和 mysqladmin 等程序的使用指南。

14.1 基础知识

本节将对从事 MySQL 系统管理工作所必须掌握的基础知识进行介绍，并告诉大家还可以在本书的哪些地方找到更多的细节内容。

14.1.1 MySQL 数据库系统的管理工具

MySQL 提供了许多种管理工具（比如 mysqladmin 和 myisamchk），本章将对它们当中的几个做比较详细的介绍。这些工具的一个共同特点是它们都是基于命令行的程序，即这些程序都没有图形化的用户操作界面，人们只能通过各种选项和命令去使用它们。

在 UNIX/Linux 环境下，这些工具需要通过 shell 窗口去执行；在 Windows 环境下，它们需要通过命令窗口去执行。本书的第 4 章介绍了一些帮助大家更好地配置 shell 窗口/命令行窗口和设置 Windows 环境变量 PATH 的技巧。

这些管理工具中的绝大多数都需要以 MySQL 的 *root* 用户和密码登录才能使用它们的全部功能，必须在启动这些工具程序时用 -u 和 -p 选项给出 MySQL 的 *root* 用户名和密码才能做到这一点。¹

1. 原书“……用 -u 和 -p 选项启动 mysqld 程序”的说法有很强的误导性！mysqld 是 MySQL 服务器程序，用 *root* 用户来启动 MySQL 服务器是一种严重的安防漏洞！所以在翻译时稍微改了一下。——译者注

这些工具程序都提供有 help 命令（例如 mysql --help），这个命令可以让用户对这些工具都提供了哪些选项和命令以及它们的具体用法有一个大致的了解。本章将要介绍的部分选项和命令——尤其是那些与 shell 或 Perl 脚本有关的选项和命令——只能在 UNIX/Linux 系统上使用，不能用于 Windows 系统。

mysqladmin 程序

mysqladmin 程序可以用来完成多种数据库系统管理任务，其中包括：

- 创建和删除数据库；
- 修改用户的密码；
- 重新加载名为 *mysql* 的权限数据库；
- 刷新数据库里的数据和日志文件（清除各种缓冲区或临时缓存区）；
- 查看 MySQL 服务器的工作状态和变量设置情况；
- 列出和保存 MySQL 进程；
- 测试与 MySQL 服务器的连接是否可用（ping 命令）；
- 关闭 MySQL 服务器（shutdown 命令）。

下面的例子演示了这个工具的一部分用法：

```
> mysqladmin -u root -p create newDatabaseName
Enter password: xxx
> mysqladmin -i 5 ping
mysqld is alive
mysqld is alive
...
> mysqladmin status
Uptime: 435152 Threads: 2      Questions: 26464  Slow queries: 3
Opens: 140      Flush tables: 1  Open tables: 0
Queries per second avg: 0.061
> mysqladmin -u root -p processlist
Enter password: xxx
Id  User Host      db      Command  Time  State  Info
1   ODBC localhost books3  Sleep    7
196  root localhost        Query   0          show processlist
...
```

mysqladmin 程序每次只能完成一项操作任务。绝大多数 mysqladmin 操作都有与之对应的 SQL 命令（这些 SQL 命令可以用 MySQL 命令解释器 mysql 程序或其他一些客户端工具程序来执行）。mysqladmin 程序的优点是我们可以方便地利用这个程序来编写各种自动化的系统管理脚本。

mysqladmin 选项和命令的用法指南可以在本书的第 22 章查到。

14.1.2 设置 *root* 密码

在安装好 MySQL 服务器之后，最先也最重要的管理工作是给 MySQL 的 *root* 用户设置一个密码。如果在安装过程中（详见第 2 章）没有这么做过，现在请参照本书第 11 章的有关内容对 MySQL 的访问控制机制和 MySQL 服务器（不仅仅是 *root* 用户）进行安防配置。下面是对第 11 章有关内容的一个简单汇总。

- **UNIX/Linux。** 执行下面两条命令。必须给出 MySQL 服务器主机的完整网络名，例如 *uranus.sol*。

```
root# mysqladmin -u root -h localhost password xxx
root# mysqladmin -u root -h computername password xxx
```

- **Windows。** 执行菜单命令 Start(开始)| Programs(程序)| MySQL | MySQL Server Instance Config

Wizard (MySQL 服务器配置向导) 打开向导，这个向导可以带领我们完成各项基本配置。不要改变默认设置。现在要做的是通过 Please set the security options (设置安全选项) 对话框给 *root* 用户设置一个密码。别忘了选中 Root may only connect from localhost (root 用户只能从本地主机连接) 复选框和取消选中 Create an anonymous account (创建一个匿名账户) 复选框。

14.1.3 MySQL 服务器配置文件

如果已经按照本书第 2 章给出的步骤安装好了 MySQL，MySQL 服务器程序 mysqld 就应该能够顺利启动，并毫无问题地运行大多数 MySQL 应用程序。如果有特殊的要求（比如说，想改变日志功能或是想对 MySQL 进行性能优化），可以通过 MySQL 配置文件里的各种选项来达到目的；表 14-1 列出了这些文件的典型存放地点。

表 14-1 MySQL 配置文件的存放地点

选项的级别	Windows 系统	UNIX/Linux 系统
全局级选项（适用于 mysqld 程序和一部分客户端程序）	较新的 MySQL 版本：C:\Programs\MySQL\MySQLServer n.n\my.ini 较早的 MySQL 版本：C:\my.ini 或 C:\Windows\my.ini	/etc/my.cnf
用户级选项（仅适用于各种客户端程序）		/ .my.cnf

在配置文件里，与服务器有关的选项都集中在 [mysqld] 选项组里。如下所示的设置的效果是：新数据表将默认地被创建为 InnoDB 数据表（而不是 MyISAM 数据表）。

```
# Example of the server-specific part of
# /etc/my.conf (Linux) or my.ini (Windows)
[mysqld]
default-storage-engine=INNODB
```

本章将对一些重要的选项进行介绍，这些选项的完整清单可以在本书的第 22 章查到。

14.1.4 重新启动 MySQL 服务器

在配置文件里做出的修改必须重新启动受其影响的程序才能生效，这对 MySQL 服务器来说尤其如此。只要改变了与 MySQL 服务器有关的选项，就必须重新启动 MySQL 服务器。

在 Windows 环境下，重新启动 MySQL 服务器的最简单的办法是使用 MySQL System Tray Monitor。这是一个驻留在 Windows 系统托盘（通常位于 Windows 桌面的右下角）里的小图标，可以通过这个图标的鼠标右键菜单去执行 Shutdown Instance（关闭 MySQL 服务器）或 Start Instance（启动 MySQL 服务器）命令。如果在系统托盘上找不到这个图标，通过菜单命令 Start（开始）| Programs（程序）| MySQL 也可以执行这些命令。注意，如果能肯定自己已经运行了这个小工具、但它的图标没有出现在系统托盘里，那很有可能是因为 Windows 任务栏的设置有问题。请用鼠标右键单击 Windows 任务栏，在弹出的菜单里选中 Properties（属性），然后在弹出的对话框中取消选中 Hide inactive icons（隐藏不活跃的图标）复选框。

在 UNIX/Linux 环境下，事情就变得容易多了：只须发出一条简单的命令就可以完成在 Windows 环境下需要通过一系列比较复杂的图形化操作才能完成的任务。重新启动 MySQL 服务器的工作可以通过相应的 InitV 脚本来完成，只须先用 stop 参数、再用 start 数调用这个脚本即可，如下所示：

Red Hat:

```
root# /etc/init.d/mysqld stop  
root# /etc/init.d/mysqld start
```

SUSE:

```
root# /etc/init.d/mysql stop  
root# /etc/init.d/mysql start
```

14.1.5 MySQL 服务器的基本配置

本节将对 MySQL 配置文件 my.cnf 或 my.ini 里的一些最基本的选择进行介绍。这些选项必须重新启动 MySQL 服务器才能生效。在写出这些选项的时候，连字符（-）和下划线（_）有着同样的效果。比如说，skip_networking 和 skip-networking 是等价的。

14.1.6 目录

在启动之后，MySQL 服务器需要知道应该在哪些目录里寻找 MySQL 工具程序和数据库。Linux 系统上的常见做法是，在负责启动 MySQL 服务器的 InitV 脚本（如/etc/init.d/mysql 脚本）里给出这些目录；Windows 系统上的常见做法则是在 my.ini 文件里对有关选项做出设置：basedir 选项负责给出安装路径，datadir 选项负责给出数据库文件的路径，如下所示：

```
# in /etc/my.conf (Linux) or my.ini (Windows)  
[mysqld]  
basedir=C:/Programs/MySQL/MySQL Server 5.0/  
datadir=C:/Programs/MySQL/MySQL Server 5.0/Data/
```

在本章后面的内容里，还将看到一些为日志文件、InnoDB 表空间文件等设置存储路径的其他选项。如果这些选项没有被明确地设置，MySQL 将使用 basedir 和 datadir 选项的设置值作为它们的默认设置。

14.1.7 通信设置

MySQL 服务器能够以 4 种不同的方式与客户程序通信，实际情况还要取决于具体的操作系统。通信方式和通信参数都可以通过有关的选项来设置。

- **TCP/IP**。在默认的情况下，MySQL 服务器与客户程序之间的通信将使用 TCP/IP 协议和 3306 号端口来进行。可以用 port 选项来另行指定一个通信端口。
skip-networking 选项将禁止 MySQL 使用 TCP/IP 进行通信。这个选项经常因为信息安全的理由而被使用。客户程序将只能从本地计算机通过一个套接字文件（UNIX/Linux）或者通过命名管道或共享内存（Windows）与服务器进行通信。
- **套接字文件（仅适用于 UNIX/Linux 系统）**。socket 选项负责给出套接字文件的存放路径。因为只有客户与服务器都使用同一个套接字文件才能成功地进行通信，所以这个选项必须同时出现在[mysqld]和[client]两个选项组里。
- **命名管道（仅适用于 Windows 系统）**。enable-named-pipes 选项将激活命名管道通信。此时，可以用 socket 选项给出一个命名管道的名字（它的默认设置是“MySQL”）。
- **共享内存（仅适用于 Windows 系统）**。shared-memory 选项将使得通信通过一个共享内存块来进行。shared-memory-base-name 选项给这个内存块起一个名字（它的默认设置是“MySQL”）。

```
' # in /etc/my.conf (Linux) or my.ini (Windows)
[mysqld]
# TCP/IP
port      = 3306
# skip-networking
# Socket File (Unix/Linux)
socket    = /var/lib/mysql/mysql.sock
# Named Pipe (Windows)
enable-named-pipes
socket   = MySQL
# Shared memory (Windows)
shared_memory
shared_memory_base_name = MYSQL
[client]
socket   = /var/lib/mysql/mysql.sock
```

14.1.8 默认的数据表格式

正如在第 8 章描述的，MySQL 能够支持多种数据表类型。如果用 *CREATE TABLE* 命令创建一个新数据表，*ENGINE* 选项将决定着数据表的类型。如果这个选项没有被给出，MySQL 服务器将使用由 *default-storage-engine* 选项设定的默认数据表类型。

```
# in /etc/my.conf (Linux) or my.ini (Windows)
[mysqld]
default-storage-engine=INNODB
```

14.1.9 新数据表的默认字符集和排序方式

正如在第 8 章描述的，从 MySQL 4.1 版本开始，可以为每一个数据库、每一个数据表、甚至每一个数据列分别设定一个字符集和一个相应的排序方式。如果在 *CREATE TABLE* 命令里没有明确地指定任何字符集，新数据表的字符集就将由 MySQL 配置文件里的以下设置来决定。

注意，MySQL 的新版本用 *character-set-server* 选项取代了老版本里的 *default-character-set* 选项。出于向后兼容的考虑，新版本仍允许使用 *default-character-set* 选项，但这个选项已经有点儿过时了。

```
# in /etc/my.cnf (Linux) or my.ini (Windows)
[mysqld]
character-set-server = latin1
collation-server     = latin1_german1_ci
```

注意，MySQL 客户程序与服务器通信时使用的字符集与 *character-set-server* 选项的设置情况完全无关。在某些程序设计语言（如 C 语言、Perl、PHP 等）和客户程序（如 mysql 和 mysqldump 等）里，必须用 *default-character-set=xxx* 选项或者通过 SQL 命令 *SET NAMES 'xxx'* 来指定一个字符集。

14.1.10 地理时区

在默认的情况下，MySQL 服务器在启动时将自动沿用服务器主机所使用的地理时区设置，但完全可以在配置文件里用 *default-time-zone* 选项另行指定一个时区。在 Windows 系统上，时区只能以相对于 UTC 时间（Coordinated Universal Time，国际协调时间）的偏移值的形式来设置，例如柏林时间是+1:00、纽约时间是-5:00 等。在 Linux 系统上，还可以使用时区的国际标准代号或它们的简写形式来设置时区，但可以这么做的前提是 MySQL 服务器上的 *mysql.time_zone* 数据表已经提前进行过初始化（参见 14.1.11 节）。

```
# in /etc/my.cnf (Linux)
[mysqld]
default-time-zone = Europe/London
```

初始化“时区”数据表

MySQL 服务器把所有与访问权限有关的信息都集中存放在一个名为 *mysql* 的数据库里。从 MySQL 4.1.3 版本开始，*mysql* 数据库又增加了几个 *time_zone* 数据表。在默认的情况下，这些数据表都是空的。如果想用正确的数据来填充这些数据表，那么，在 UNIX/Linux 系统上，需要执行如下所示的命令——别忘了用自己的时区文件路径替换掉这个示例中的 /usr/share/zoneinfo。请注意，在执行这个命令的时候，MySQL 服务器必须处于运行状态：

```
root# mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql
Password: xxxxxxxx
```

执行完上面这条命令之后，*mysql.time_zone* 数据表将包含着一份长长的时区名单（在笔者的 Linux 系统上，这份名单有 1600 条记录项）。

```
SELECT name FROM mysql.time_zone_name LIMIT 5
name
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmara
```

MySQL 的 Windows 版本没有专门用来对“时区”数据表进行初始化的命令。因此，对那些运行在 Windows 环境下的 MySQL 服务器来说，时区只能使用相对于 UTC 标准时间的偏移值（例如：'+1:00'）来设置。

与如何对 MySQL 中的时区进行管理、如何为 MySQL 连接设置本地时区、以及如何使用 SQL 函数去进行日期/时间计算等问题有关的讨论，可以参见本书的第 10 章。

14.1.11 出错消息的显示语言

如果想让 MySQL 用德语、法语或是其他一些语言来显示出错消息，只须在任何一个 MySQL 配置文件里用 language 选项做出相应的设置即可；这项设置还将影响到出错日志文件 *hostname.err* 的内容。

MySQL 目前支持 20 种语言，它们的名单可以在 *mysql\share* 子目录（Windows 系统）或 */usr/share/mysql* 子目录（Linux 系统）里查到。如果想让出错消息显示为（比如说）德语文字，在 MySQL 配置文件里做出如下所示的修改即可：

```
# in /etc/my.cnf or my.ini (Windows)
[mysqld]
language = german
```

如果 MySQL 找不到相应的语言文件，请提供一个完整的路径，如 language = /usr/share/mysql/danish。

14.1.12 SQL 模式

从 4.1 版本开始，MySQL 服务器可以运行在几种不同的模式下。这些模式要用 *sql-mode* 选项来设置，在不引起冲突的前提下，允许同时激活多个模式（但必须用逗号把它们隔开）。还可以在 MySQL 服务器正在运行的同时用 SQL 命令 *SET [GLOBAL] sql_mode = 'mode1, mode2'* 去改变其 SQL 模式。

下面是对几种最重要的 SQL 模式的简单介绍，其中 ansi、db2、maxdb、mssql、mysql323、mysql40、oracle 和 postgresql 都是其他一些选项的简写形式。比如，ansi 相当于以下几种模式的组合：real_as_float、pipes_as_concat、ansi_quotes、ignore_space 和 only_full_group_by。

- ❑ ansi：MySQL 服务器的行为与 ANSI SQL 标准最为兼容。
- ❑ db2、maxdb、mssql、mysql323、mysql40、oracle、postgresql：MySQL 服务器的行为与给定数据库系统最为兼容。
- ❑ allow_invalid_dates：MySQL 服务器将接受显然不正确的日期（例如 30.2.2005）。这是自 MySQL 4.1 版本以来的默认行为，但 MySQL 5.0 版本加强了对日期/时间值的合法性检查。
- ❑ ansi_quotes：字符串必须用单引号括起来（例如：'I am a string'），对象名必须用双引号括起来（例如："table"）。
- ❑ error_for_division_by_zero：这项设置将使以 0 为除数的除法运算触发错误。（MySQL 服务器的默认行为是返回 NULL 值并发出警告。）
- ❑ ignore_space：在函数调用里，MySQL 服务器将允许函数名和括号之间有空格（例如：SQRT(3)）。在默认的情况下，这种情况是一个错误，因为它可能导致二义性。
- ❑ no_zero_date：MySQL 服务器将把 0000-00-00 视为非法日期并触发错误。（MySQL 服务器的默认行为是接受这样的日期值，这在需要表达和存储“日期不详”信息时很有用。）
- ❑ only_full_group_by：如果 GROUP BY 表达式里的数据列没有在 SELECT 命令行的前半部分被命名，MySQL 服务器将认为该表达式是非法的。
- ❑ pipes_as_concat：MySQL 服务器将把 “||” 符号视为字符串合并操作符（相当于 CONCAT() 函数）。在默认的情况下，“||” 相当于 OR 操作符。
- ❑ real_as_float：MySQL 服务器将把 REAL 视为 FLOAT（8 位精度）的同义词。在默认的情况下，REAL 是 DOUBLE（16 位精度）的同义词。
- ❑ strict_all_table：如果在使用 INSERT 命令插入新记录时没有为没有默认值的数据列提供值，MySQL 服务器将触发一个错误。（在默认的情况下，MySQL 服务器会自做主张地为没有默认值的数据列提供一个它认为最适当的值，比如为数值类型的数据列提供 0。）
- ❑ strict_trans_tables：这项设置类似于 strict_all_table，但在这个设置里，MySQL 服务器还区分不同的数据表类型。如果是支持事务的数据表（即 InnoDB），非法的 INSERT 命令将总是被放弃；如果上其他类型的数据表（MyISAM），则只在用 INSERT 命令插入的第一条新记录，存在这一错误时才放弃整个命令。换句话说，如果正在用一条 INSERT 命令插入多条记录，而错误发生在第 n 条记录上，这条 INSERT 命令将会得到执行而不是被放弃——MySQL 会自做主张地为第 n 条记录提供一个它认为最适当的值。

对这些模式的详细描述可以在 <http://dev.mysql.com/doc/mysql/en/server-sql-mode.html> 网址的文档里查到。本书只对 MySQL 服务器的默认 SQL 模式（即根本没有设置 sql-mode 选项时的模式）进行了描述。

注解 SQL 模式会对 MySQL 服务器的许多行为产生影响，其中之一是 MySQL 服务器会在定义存储过程的时候把当前 SQL 模式记载下来，然后在执行存储过程的时候会临时切换到相应的 SQL 模式下。MySQL 服务器在不同 SQL 模式下的行为变化可以网址 <http://sql-info.de/mysql/gotchas.html> 处的文档里查到。

14.2 备份

备份工作无疑是最重要的数据库管理任务之一。MySQL 提供了以下几种备份办法：

- **使用 mysql dump 命令。**这是最常用的备份办法。这个命令将返回一个文本文件，其内容是重新生成各有关数据库和数据表并把备份数据填入它们的 SQL 命令。用 mysql dump 命令进行备份的速度相对较慢，但如此生成的备份有着最好的兼容性。因此，mysql dump 命令最适合需要对数据库进行迁移的场合（比如把某给定数据库从这一个 MySQL 服务器复制到另一个 MySQL 服务器）。
- **直接复制整个数据库目录。**通过操作系统直接复制某给定数据库目录是最简单和最快的备份办法。从效率和信息安全的角度考虑，在采用这个办法进行备份时应该让 MySQL 服务器停止运行。如果实际情况不允许 MySQL 服务器停止运行，则必须保证被备份数据库在备份期间不会发生任何变化——如果是在 UNIX/Linux 环境下进行备份，用 mysqlhotcopy 脚本可以保证这一点。等以后需要重新创建那个数据库的时候，只须把当初备份下来的数据库目录重新复制到 MySQL 的数据库根目录下即可。注意，这种备份办法只适用于 MyISAM 数据表，不适用于 InnoDB 数据表。
- **把日志文件和定期制作的增量备份文件相结合。**这个办法的好处是可以把每次备份的数据量控制在比较小的规模，但不足是只适用于数据库在前、后两次备份之间的修改量不是很大的情况。基于日志文件的备份机制不在本节讨论，将在本章后面讨论到日志功能的时候再对这一机制做进一步介绍。
- **利用镜像机制进行备份。**镜像机制能够让同一个数据库同时存在于两台不同的计算机上并保持同步。对于变化比较频繁的数据库来说，这种备份手段对两台计算机之间的通信保障有着很高的要求。如果只是想在两块不同的硬盘上为数据库保存两份副本，利用 RAID 技术只对硬盘进行镜像的做法将更有效率。基于镜像机制的备份办法也不在本节讨论，我们将在本章后面讨论到镜像机制的时候再对这一问题做进一步介绍。

热备份。使用 mysql dump 备份工具和直接复制数据库目录这两种备份办法都可以在数据库仍在工作的情况下进行。但如果想保证备份数据的完整性，就必须在备份每一个数据表的过程中给它加上一个读操作锁（read lock）。

这么做的效果是：一方面，备份工作只能在数据表上没有写操作锁（write lock）时才能进行；另一方面，在备份期间，其他客户将不能访问正在被备份的数据表。因为备份一个比较大的数据库往往需要花费相当长的时间（尤其是在使用 mysql dump 程序进行备份的时候），所以这种“先锁定数据表、再对它进行备份”的办法往往会给其他用户带来不便。

为了解决这一问题，商业化的数据库系统几乎都提供有这样或那样的热备份机制，这种机制可以在不对整个数据表进行全面锁定的情况下进行备份。MySQL 目前还不具备对 MyISAM 数据表进行热备份的能力（mysqlhotcopy 脚本也做不到这一点），但如果打算备份的都是 InnoDB 数据表的话，有个名为 InnoDB Hot Backup 的工具可以用来进行热备份（详见 <http://www.innodb.com/order.php>）。

14.2.1 备份数据库 (mysql dump)

mysql dump 命令将生成一份 SQL 命令清单，这份清单可以把数据库按照原来的样子精确地重新创建出来。mysql dump 的基本工作原理是这样的：对于每一个数据表，先生成一条 CREATE TABLE 命令（为了重新创建这个数据表及其索引），再为这个数据表里的每一条记录分别生成一条 INSERT 命令（为

了把数据重新填充到数据表里去)。在下面的例子中,我们将使用 mysqldump 程序对 *mylibrary* 数据库里的 *authors* 数据表(请参见第 8 章)进行备份。

为了让大家看懂 mysqldump 程序的备份结果,这里先对其中的一些语法元素进行一下解释。以“*--*”开头的文本行是 SQL 语言里的注释语句。此外,类似于 C 语言,以“/*”开头的文本行也是注释。与 MySQL 有关的注释以“/*!n”开头,其中的 *n* 是 MySQL 版本号。如果 MySQL 服务器的版本号低于 *n*,则“/*”和“*/”之间的内容将被视为注释。如果 MySQL 服务器的版本号大于或等于 *n*,则“/*”和“*/”之间的内容将被视为 SQL 命令并得到执行。这可以防止比较老的 MySQL 版本触发各种语法错误。

在备份结果的开头有一些 *SET* 命令,它们的作用是保存当前字符集设置并把备份文件的字符集设置为 Unicode(也就是说,备份文件里的 SQL 命令都使用着 UTF8 字符集)。此外,*UNIQUE_CHECK=0* 和 *FOREIGN_KEY_CHECK=0* 的作用是暂时关闭唯一性检查和外键检查,这可以让 SQL 命令执行得更有效率。

真正的备份是从 *DROP TABLE* 命令开始的,它的作用是在恢复备份时先删除 *authors* 数据表——如果它已经存在。接下来是负责重新创建这个数据表的 *CREATE TABLE* 命令和负责填充数据的一系列 *INSERT* 命令。在整个备份结果的末尾还有一些用来恢复各种当前设置(字符集、外键检查等)的指令,但在这个例子里省略了这些指令。

```
> mysqldump -u root -p mylibrary authors
Enter password: xxx
-- MySQL dump 10.9
-- Host: localhost    Database: mylibrary
-- Server version      5.0.2-alpha-standard-log
/*!40101 SET @OLD_CHARACTER_SET_CLIENT =@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION =@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
    UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
    FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
    SQL_MODE="NO_AUTO_VALUE_ON_ZERO" */;
--
-- Table structure for table `authors`
--
DROP TABLE IF EXISTS `authors`;
CREATE TABLE `authors` (
    `authID` int(11) NOT NULL auto_increment,
    `authName` varchar(60) collate latin1_german1_ci NOT NULL default '',
    `ts` timestamp NOT NULL default CURRENT_TIMESTAMP
        on update CURRENT_TIMESTAMP,
    PRIMARY KEY (`authID`),
    KEY `authName` (`authName`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_german1_ci;
--
-- Dumping data for table `authors`
--
/*!40000 ALTER TABLE `authors` DISABLE KEYS */;
LOCK TABLES `authors` WRITE;
INSERT INTO `authors` VALUES
    (1,'Kofler Michael','2004-12-02 18:36:51'),
    (2,'Kramer David','2004-12-02 18:36:51'),
    (3,'Orfali Robert','2004-12-02 18:36:51'),
...

```

在实际工作中，把 `mysqldump` 程序的备份结果输出到屏幕上的情况非常少见，人们几乎总是用`> filename` 语法把它们输出到一个给定的文件里：

```
> mysqldump -u root -p mylibrary authors > backup.sql
```

1. 备份所有数据库

下面的命令将把某给定 MySQL 服务器上的全部数据库备份到一个文件里去。在备份期间，所有的写操作都将被阻断：

```
> mysqldump -u root --all-databases > backup.sql
```

2. mysqldump 命令语法

`mysqldump` 命令的基本语法如下所示：

```
mysqldump [options] dbname [tables] > backup.sql
```

如果没有给出任何数据表的名字，`mysqldump` 工具将把数据库 `dbname` 里的所有数据表备份到 `backup.sql` 文件里去。如果给出了数据表的名字，则备份工作将仅限于给定的数据表。如果想备份一个以上的数据库或是备份所有的数据库，就需要使用如下所示的语法变体：

```
mysqldump [options] --databases dbname1 dbname2 ...
mysqldump [options] --all-databases
```

备份工作的细节可以通过一系列选项来控制。对这些选项的描述可以在第 22 章找到。如果只是进行一次简单的备份，使用这些选项的默认设置就应该足够了。`mysqldump` 程序的默认备份行为是：

- 在备份过程中，在备份每一个数据表之前都先给它加上一个读操作锁。
- 让备份结果文件尽可能地小（让备份结果里的每条 `INSERT` 命令对应着尽可能多的数据记录）。
- 在备份结果文件里插入 `DROP TABLE` 命令，这条命令将在恢复备份时删除现有的同名数据表。
- 把数据库的所有属性（包括那些可能与其他数据库系统不兼容的 MySQL 独有属性）都保留下来。
- 使用 `UTF8` 字符集来创建备份，并在备份结果文件里增加必要的 SQL 命令以确保在将来恢复备份时还能够把字符集设置也恢复到原来的样子。

上述默认设置不适合用来进行下面这几种备份工作：一次备份多个 InnoDB 数据表——除非各有关数据表在备份期间都不会发生变化（这是为了避免备份文件里的数据出现彼此矛盾的现象）；备份结果文件需要向后兼容 MySQL 的老版本或是需要与其他的数据库系统保持兼容。在这些场合，必须对 `mysqldump` 程序的有关选项做出必要的调整。有时候，甚至有必要先用 `skip-opt` 选项把所有的默认设置全部禁用掉、然后再分别设置个有关选项。如果备份工作需要满足比较特殊的要求，请参考下面两项建议对有关选项进行调整。

最稳妥的 MyISAM 数据库备份办法。在默认的设置下，`mysqldump` 程序会在备份每一个数据表之前给它加上一个读操作锁以防止它在备份过程中发生变化，但这并不能防止数据表 `B` 在 `mysqldump` 程序备份数据表 `A` 的时候被修改。这就有可能导致这样一种后果：数据表 `A` 引用了数据表 `B` 里的某条记录，但这条记录却在 `mysqldump` 程序备份数据表 `B` 之前被删除了。解决这个问题的办法是用上 `--lock-all-tables` 选项：

```
> mysqldump -u root -p --lock-all-tables databasename > backup.sql
```

备份 InnoDB 数据库。在备份 InnoDB 数据表的时候，`LOCK` 命令解决不了任何问题。还好，`mysqldump` 程序的 `--single-transaction` 选项可以把备份工作放在一个事务里完成。但是，因为

--single-transaction 选项与默认设置不兼容，所以必须先用--skip-opt 选项把默认设置全部禁用掉，然后再以手动方式设置一系列的选项。如此得到的 mysqldump 命令行大大超出了本书的页面宽度——不得不把它写成 3 行：

```
> mysqldump -u root -p --skip-opt --single-transaction --add-drop-table \
  --create-options --quick --extended-insert \
  -set-charset -disable-keys databasename > backup.sql
```

3. 视图与存储过程的备份

mysqldump 程序可以对视图（view）的定义进行备份，但用它来备份存储过程（stored procedure, SP）却没那么容易。即使用 mysqldump 程序对某个数据库里的数据表全都进行了备份，属于这个数据库的存储过程也不会存在于备份结果文件里。事实上，目前还没有哪种现成的备份工具能够把某给定数据库里的存储过程简单地备份出来。当然，可以用下面这条命令去备份 mysql.proc 数据表：

```
> mysqldump -u root -p mysql proc > backup.sql
```

但如此得到的备份文件将包含着所有数据库的 SP，用这样的备份文件去恢复某给定数据库里的 SP 将会非常麻烦。

想获得一个比较实用的结果还是有办法的：只要在备份 proc 数据表时只把属于想备份的那个数据库的记录备份出来即可。要做到这一点，就必须用--where 选项给出一个条件表达式：在下面的例子里，db 是 proc 数据表用来存放数据库名字的数据列，而--no-create-info 选项的作用是让 MySQL 在为 proc 数据表制作备份时不生成 CREATE TABLE 命令，这样在恢复备份时就不会破坏 proc 数据表里属于其他数据库的存储过程了。

```
> mysqldump -u root -p "--where=db='dbname'" --no-create-info \
  mysql proc > backup.sql
```

这里还有一个细节问题需要注意：在使用 backup.sql 文件去恢复备份之前，必须先把 proc 数据表里属于数据表 dbname 的现有 SP 全部删掉——如果不这样做，当备份文件里的 SP 试图覆盖 proc 数据表里的现有 SP 时就将触发一个错误。

```
DELETE FROM mysql.proc WHERE db='dbname'
```

在存储过程的官方备份工具出现之前，还可以利用本书第 15 章介绍的 SP Administrator 工具去备份存储过程。

4. mybackup脚本

mybackup 脚本是一个用 Perl 语言编写的备份工具，它目前没有被收录在 MySQL 软件包里，但可以在因特网上找到它：<http://www.mswanson.com/mybackup/>。

mybackup 脚本其实是调用 mysqldump 程序去进行备份的，但它比 mysqldump 程序用起来更方便一些：mybackup 脚本会为每个数据库分别生成一个备份文件并把它们直接写入一个由用户指定的子目录，如有必要，mybackup 脚本还能在制作备份的同时调用 gzip 程序对备份文件进行切割和压缩（以 2GB 为单位）。

14.2.2 用备份恢复数据库（mysql）

mysqldump 程序本身没有提供专门用来恢复备份的命令或操作符，原因很简单：MySQL 自带的命令解释器 mysql 程序已经足以完成这项工作。

用 mysqldump 程序生成的备份文件去重新创建数据库的工作即使在使用了一个不同的（比较新的）MySQL 版本或者 MySQL 运行在另外一台计算机上（在不同的操作系统环境下）也可以进行。

从另一个方面讲，用比较老的 MySQL 版本去读取由 mysqldump 程序生成的备份文件可能会遇到一些麻烦：如果被备份的数据库使用了一些老版本的 MySQL 无法识别和支持的功能或数据类型，困难将不可避免。

1. 恢复单个数据库

如果 backup.sql 文件只包含着一个数据库，用以下命令就可以把它恢复出来：

```
> mysql -u root -p database < backup.sql  
Enter password: xxx
```

数据库 *databasename* 必须已经存在。否则，可以简单地用 mysqladmin create *databasename* 命令把它创建出来。

2. 恢复多个数据库

如果用 mysqldump 程序备份了多个数据库，backup.sql 文件将包含着必要的 *CREATE DATABASE* 命令，它们可以在恢复备份前把尚不存在的数据库创建出来。也就是说，不用提前创建那些数据库，也不用在执行 mysql 命令时指定一个数据库。

```
> mysql -u root -p < backup.sql  
Enter password: xxx
```

3. 以交互方式读入数据库和数据表

可以在 mysql 程序里以交互方式读入多个*.sql 文件，用不着每读入一个*.sql 文件都不得不执行一次 mysql 程序。具体做法是：在 mysql 程序里使用 *SOURCE* 命令（或者是它的简写形式“*./*”）并给出一个*.sql 文件的文件名。（在 Windows 系统上，文件路径中的子目录既可以用字符“/”分隔，也可以用字符“\”分隔；而且文件名不必用引号括起来。）

现在，mysql 将依次执行那个文件里的命令。不过，mysql 程序在以交互方式执行*.sql 文件里的 SQL 命令时会产生大量的控制台输出（即“Query OK, 1 row affected”消息），如果不想看到它们，就需要在启动 mysql 程序时使用--silent 选项，如下所示：

```
> mysql -u root -p --silent  
Enter password: xxx  
mysql> USE dbname  
mysql> \. /tmp/backup.sql
```

4. 字符集问题

在默认的情况下，版本比较新的 mysqldump 程序将使用 UTF8 字符集来生成备份文件。但因为备份文件里包含着使用这个字符集所必需的命令，所以在恢复这样的备份文件时一般不会有什么问题。

可是，如果备份文件是用 mysqldump 程序的老版本或是另外一种备份工具生成的，情况又不同了。在使用 mysql 程序恢复这样的备份时，应该用--default-character-set 选项明确地对字符集做出设定：

```
> mysql -u root -p --default-character-set=latin1 mylibrary < mylibrary.sql
```

14.2.3 快速备份 (mysqlhotcopy)

长期以来，MySQL 的 UNIX/Linux 版本一直提供有一个名为 mysqlhotcopy 的 Perl 脚本。用这个脚本制作和恢复一个数据库备份的速度要比 mysqldump 程序快很多。这个脚本的工作原理是：先给有关的数据库加上一个读操作锁，然后用 *FLUSH TABLES* 把缓存在内存里的修改写入硬盘上的数据库文件，最后把有关的数据库文件复制下来。

注意 因为mysqlhotcopy脚本是通过直接复制数据表文件的办法来创建备份文件的，所以它只适合用来备份MyISAM数据表，不适合用来备份InnoDB数据表——用它去备份InnoDB数据表不会产生任何出错消息，但得到的备份结果文件却不能用来恢复InnoDB数据表。

mysqlhotcopy脚本的文档可以用下面这条命令查看（注意，可以用另一个路径来代替 /usr/local/bin子目录）：user\$ `prerldoc /usr/local/bin/mysqlhotcopy`。

1. 制作一个备份

在最简单的情况下，mysqlhotcopy 脚本的用法如下所示：

```
root# mysqlhotcopy dbname1 dbname2 dbname3 backup/
```

上面这条命令将把数据库 *dbname1*、*dbname2* 和 *dbname3* 复制到给定的备份子目录里去（这个备份子目录必须已经存在）；每个数据库将被复制到一个下级子目录里，下级子目录的名字就是数据库的名字。为了完成备份工作，mysqlhotcopy 脚本必须有权读取有关的 MySQL 数据库文件（MySQL 数据库文件通常都存放在/var/lib/mysql/*dbname*/子目录里）。

2. 选项

表 14-2 只列出了 mysqlhotcopy 脚本最重要的选项。这些选项的完整清单可以在 mysqlhotcopy 脚本的文档里查到。

表 14-2 mysqlhotcopy 脚本的重要选项

选 项	含 义
--allowold	新生成的备份文件将覆盖现有的备份文件
--keepold	先把现有的备份文件转移到 <i>dbname_old</i> 子目录里。如果这个子目录已经存在，它的现有内容将被覆盖
--flushlog	在本次备份后，把对数据库的修改记入一个新的日志文件。这个选项只在打算使用“日志文件 + 备份文件”的增量备份方案时才有必要使用
--noindices	只复制(My) ISAM 数据表的数据文件，不复制它们的索引文件（更准确地说，是只复制索引文件的前 2KB 内容。有了这 2KB 的信息，我们就可以在恢复备份时利用 myisamchk -r 命令把索引文件重新“修复”出来，但这种修复往往需要花费较长的时间）。因为需要复制的数据量比较少，所以用 mysqlhotcopy 脚本去备份一个数据库的速度相当快

mysqlhotcopy 脚本的 -u、-p、-P 和 -S 选项以及 --user=、--password=、--port= 和 --socket= 与它们在其他 MySQL 工具程序里的用途完全一样，唯一需要注意的是 -p 选项的用法：密码必须紧跟在 -p 选项后给出（其他的 MySQL 工具程序都允许以交互方式输入密码，那么做相对要更安全一些。）

3. 恢复数据库

用 mysqlhotcopy 脚本制作的备份文件很容易恢复，只要把数据库的备份子目录复制到 MySQL 的数据根目录下即可。这里有一个细节问题需要大家注意：如果是以 *root* 用户的身份进行的复制操作，千万不要忘记在事后用 chown 命令对那些数据库文件的属主做出必要的修改（新属主必须是用来运行 MySQL 服务器程序 mysqld 的那个账户的用户名，它通常是 *mysql*）

```
root# chown -R mysql.mysql /var/lib/mysql/dbname
```

注意 如果打算恢复的数据库已经存在，MySQL 有可能不允许覆盖它的某些文件。为确保数据库恢复操作的成功，应该先执行一条 *DROP DATABASE* 命令。

此外，为了保证如此恢复出来的数据库能够正常使用，在恢复数据库时使用的 MySQL 版本必须与在制作备份时使用的 MySQL 版本相兼容。（MySQL 的新版本往往会对老版本里的数据库文件格式做一些改进，主版本号发生变化时尤其如此。）

14.3 数据库的迁移

这里所说的“迁移”是指把数据库从一个系统搬迁到另一个系统上。需要对数据库进行迁移的理由主要有以下几种：

- 安装了一个新的数据库服务器。
- 需要把数据库从一个开发系统（比如一台本地计算机）部署到日常使用的工作系统（比如一台属于 ISP 的计算机）上去。
- 对 MySQL 进行了升级（比如从 4.1 版本升级到了 5.0 版本）。
- 改用了另外一种数据库系统（比如从 Microsoft SQL Server 改用为 MySQL）。

1. 在 MySQL 系统之间迁移数据库

在 MySQL 系统之间迁移数据库的工作通常由 `mysqldump` 程序（备份）和 `mysql` 程序（恢复）来完成；具体做法已经在前面做过了介绍：先用 `mysqldump` 程序对源 MySQL 服务器进行一次完整的备份，再用 `mysql` 程序把备份出来的数据库安装到目标 MySQL 服务器上去。

如果准备迁移的数据表全都是 MyISAM 格式，并且在两台计算机上运行的 MySQL 版本也相互兼容（比如说，它们都是 5.0.n 版本），最简单也最有效率的办法是对数据库文件进行复制。这个办法对运行在不同操作系统环境下的 MySQL 系统也同样适用。

与 `mysqldump/mysql` 程序的办法相比，用直接复制 MyISAM 数据表的办法来迁移数据库的最大优点是速度非常快。

只有主版本号相同的 MySQL 版本（比如 MySQL 5.0.17 和 5.0.18 版本）才能保证数据库文件的格式完全兼容，各主要版本之间（比如 3.23 到 4.0 或者 4.0 到 4.1）不能保证这一点。对于后一种情况，还是采用 `mysqldump/mysql` 程序来迁移数据库更保险。至于 InnoDB 数据表，因为目前还没有能够直接复制主空间（`masterspace`）文件的工具，所以要想迁移 InnoDB 数据表，就一定会用到 `mysqldump` 程序。

用 `mysqldump/mysql` 程序来迁移数据库并不意味着非得创建一个或多个备份文件不可，完全可以把 `mysqldump` 程序的输出直接传递给 `mysql` 程序，就像下面这个例子里的命令所演示的那样。

这条命令将在源数据库 `dbname` 所在的计算机上执行，它将把有关数据传输到另一台计算机（它的主机名是 `destinationhost`）上去。这里需要假设 `dbname` 数据库已经在目标计算机里存在，而且执行这条命令的用户有权访问源计算机上的有关文件。因为页面宽度的限制，把这条命令写成了两行：

```
root# mysqldump -u root --password=xxx dbname | \
    mysql -u root --password=yyy -h destinationhost dbname
```

如果只想改变数据库文件的格式（例如把它们从 MyISAM 格式改为 InnoDB 格式），根本无需采用迁移数据库的办法，*ALTER TABLE tablename TYPE=newtype* 命令已足以满足需要。如果需要对一大批数据表进行转换，Perl 脚本 `mysql_convert_table_format`（仅适用于 UNIX/Linux 环境）可以帮助用户又快又好地完成这一任务（注意，使用这个脚本的大前提是必须有权与 MySQL 服务器建立连接）。

提示 作为数据库迁移工作中的一种特例，将在本章的末尾向大家介绍怎样把数据库安装到ISP计算机上去。这里最关键的问题是用户在ISP的计算机上有没有足够的权限。

如果想把数据库从Windows系统复制到UNIX/Linux系统，还必须特别注意文件扩展名的大小写问题。还好，这个问题可以用Perl脚本mysql_fix_extensions来解决。

2. 对MySQL服务器进行升级

对MySQL服务器进行升级（比如从5.0.7版本升级到5.0.8版本）的常见步骤是：先停止MySQL服务器的运行，然后卸载掉老版本，最后再把新版本安装到位。（这种升级一般不会引起任何问题，但为保险起见，在对服务器进行升级之前最好先用mysqldump程序把所有的数据库都备份下来。费事总比后悔强！）

从4.n版本升级到5.n版本也可以采用这一思路，但此时还需要解决以下几个细节问题：

□ **安装路径。**在Windows系统上，MySQL的默认安装路径已经发生了变化：它以前默认安装在C:\mysql子目录里，现在则默认安装在C:\Programs\MySQL\MySQL Server n.n子目录里。在升级之后，在启动服务器之前，必须把所有的数据库文件复制到新子目录下。

此外，MySQL配置文件的默认存放位置也发生了变化。在Windows系统上，老版本的MySQL服务器会去读取C:\Windows\my.ini文件，但新版本的MySQL服务器将去读取C:\Programs\MySQL\MySQL Server n.n\my.ini文件。

□ **mysql数据库。**如果想在升级后继续使用mysql数据库里的访问权限管理信息，必须在升级前把这个数据库备份下来、在升级后再把它重新读入。不过，因为MySQL的新版本经常会给mysql数据库扩充一些新的数据表和数据列，所以这件事做起来还是比较麻烦的。

在UNIX/Linux环境下，可以简单地用mysql_fix_privilege_tables脚本对用老版本的备份恢复出来的mysql数据表进行升级。要想执行这个脚本，必须知道root用户的密码。

Windows环境没有这样一个脚本可用。必须用MySQL自带的命令解释器mysql程序去执行script/mysql_privilege_tables.sql文件。（下面的命令必须写成一行；我们这里是因为页面宽度的限制才把它写成两行的。）

```
> mysql -u root -p mysql <
  "c:\programs\MySQL\MySQL Server 5.0\scripts\mysql_fix_privilege_tables.sql"
```

根据笔者个人的经验，在把老版本的mysql数据表导入新版本的MySQL服务器的时候往往会遇到各种问题。不仅如此，因为所有的密码还沿用着以前的格式，所以新的身份验证机制在信息安全方面的优点（详见第11章）也就无法得到体现。

如果必须迁移一个非常大的MySQL“用户”数据表，以手动方式把老mysql数据表里的访问控制数据迁移到新mysql数据表里的办法往往更简明易行。下面这个经验也许会对读者有所帮助：把老版本的mysql数据库换个名字（比如oldmysql）安装到新服务器上，让它与新的mysql数据库共存于新服务器中。

与MySQL服务器的升级工作有关的更多信息可以<http://dev.mysql.com/doc/mysql/en/update.html>处的文档里查到。

3. 改用另外一种数据库系统

从另外一种数据表系统迁移到MySQL或者从MySQL迁移到另外一种数据库系统没有普遍适用的解决方案。虽说几乎所有的数据库系统都有类似于mysqldump程序的工具可以把数据库的内容导出到一个SQL文件里去，但问题是如此得到的结果文件很难做到彼此完全兼容（出自不同厂商的数据

库系统在数据列类型方面的差异以及它们各自与 ANSI SQL/92 标准的兼容程度是导致这一局面的主要原因)。这类问题中的一部分可以用“查找加替换”的办法来解决。如果使用的是 UNIX/Linux 操作系统, 可以使用 awk 和 sed 工具。

笔者曾编写过一个专门用来把 Microsoft SQL Server 数据库迁移到 MySQL 系统里去的小脚本, 这个脚本在很多场合都能派上用场(请参见第 19 章)。至于其他的数据库系统, 应该可以在下面这两个网址找到一个适用的迁移/转换工具: <http://solutions.mysql.com/> 和 <http://solutions.mysql.com/software/>。

如果使用的是 Windows 系统, ODBC 往往能帮上忙。比如说, 可以先把一个来自某 ODBC 兼容系统的数据库导入 Access, 再从 Access 把它导出到 MySQL。不过, 千万不要想当然地认为数据列的所有定义细节在经过这么一番周折之后还能原封不动地保留下来。

14.4 导出和导入文本文件

有时候, 需要把某个数据表的内容以尽可能紧凑的格式写入一个文本文件或者是从一个这样的文件里把数据读入数据表。MySQL 在这方面提供了好几种办法:

- **LOAD DATA**。这条 SQL 命令可以读入一个文本文件的内容并把它们传输到一个数据表里去。
- **mysqlimport 程序**。这个工具相当于 *LOAD DATA* 命令的命令行版本, 特别适合用来编写各种自动化的数据导入脚本。
- **SELECT ... INTO FILE**。这条 SQL 命令可以把 *SELECT* 命令的查询结果写入一个文本文件。
- **mysqldump 程序**。如果打算编写一个自动化的数据导出脚本, 建议使用 *mysqldump* 程序。它的功能类似于 *SELECT ... INTO FILE* 命令。
- **mysql 程序**。在许多场合, 可以使用无所不能的 *mysql* 程序来实现文本格式、HTML 格式或 XML 格式的数据导出/导入操作。

如果以上这些命令都不适合需要, 就只能自行编写一些脚本来实现导入或导出操作了。Perl 语言很适合用来编写这类脚本。

14.4.1 文本文件里的特殊字符

LOAD DATA、*SELECT ... INTO FILE*、*mysqlimport* 和 *mysqldump* 的一个共同之处是都有一组专门用来处理文本文件里的特殊字符的选项。这组选项总共有 4 个, 下面是它们的 SQL 命令形式:

```
FIELDS TERMINATED BY 'fieldtermstring'
ENCLOSED BY 'enclosechar'
ESCAPED BY 'escchar'
LINES TERMINATED BY 'linetermstring'
```

字符串 *fieldtermstring* 代表同一记录各字段(数据列)之间的分隔符(比如一个制表符)。

字符 *enclosechar* 代表文本文件里各项数据的前缀和后缀字符(字符串数据的前、后缀字符通常是一个单引号或双引号)。如果某项数据本身由这个字符开头或结尾, 它们在这项数据被读入数据表时将被删除——字段(数据列)的结束标志是 *fieldtermstring* 字符串。

字符 *escchar* 代表用来标识特殊字符的转义前导字符(默认设置是反斜线字符)。如果在文本文件里用来分隔数据列或数据列的特殊字符本身有可能出现在字符串类型的实际数据里, 就必须指明一个转义前导字符。此外, MySQL 还要求零值字节必须是 “\0”的形式(别忘了把这个反斜线字符替换为 *escchar*)。

字符串 *linetermstring* 是各条记录之间的分隔符。在 DOS/Windows 文本文件里，这个字符串必须是'/'r/n'。

14.4.2 字符串、数值、日期/时间、BLOB 和 NULL 值

本节将要介绍的所有命令都要求有关的文本文件必须严格遵守一种数据格式。具体地说，在导入操作中，提供的文本文件必须严格遵循 MySQL 所要求的格式。导出操作对文本文件的格式要求不那么严格，可以在 *SELECT* 命令里使用 SQL 函数来排版有关的数据（例如：用 *DATA_FORMAT()* 函数来排版日期/时间值等）。

此外，还可以利用刚才介绍的 4 个选项来设定数据行和数据列之间的分隔符以及字符串和特殊字符串的识别标志。（对这些选项的详细介绍可以在第 21 章里的 *LOAD DATA* 和 *SELECT ... INTO FILE* 条目下查到。）

数值。对于非常大和非常小的 *FLOAT* 和 *SINGLE* 数值，可以用科学计数法来表示（例如：-2.3e-037）。

字符串。字符串在导入和导出操作中不发生变化。在默认的情况下，字符串里的特殊字符必须加上反斜线字符作为识别标志，以区别于各种分隔符（如制表符、回车符、换行符等）。

BLOB 数据。二进制对象按照单字节字符串来对待。导入和导出操作都不支持使用十六进制字符串（0x123412341234 ...）。

日期/时间。日期值按照 2005-12-31 格式的字符串来对待，时间值按照 23:59:59 格式的字符串来对待。时间戳值按照 20051231235959 格式的整数来对待。

NULL 值。*NULL* 值的处理比较麻烦。下面的讨论以反斜线 (\) 作为转义前导字符，以双引号 ("") 作为字符串的前、后缀标记；如果使用的是其他字符（选项 *FIELDS ESCAPED BY '?' ENCLOSED BY ?'*），请对以下内容里的相应字符进行替换。

在指定了转义前导字符的导出操作中，*NULL* 值将被表示为 \N；在没有指定转义前导字符的导出操作中，*NULL* 值将被表示为由 4 个字符构成的字符串“NULL”（不包括中文引号）。为了与正常的字符串数据相区别，代表 *NULL* 值的“NULL”和 \N 没有用双引号（"）括起来¹。当然，BLOB 数据里的 *NULL* 值字节不属于这种情况。

在指定了转义前导字符的导入操作中，MySQL 将把 *NULL*、\N 和 "N" 解释为 *NULL* 值，但 "NULL" 将被解释为一个由 4 个字符构成的字符串“NULL”。

注意 如果在导入和导出操作中既没有为特殊字符指定转义前导字符、也没有为字符串指定“括号”字符的话，MySQL 将无法把 *NULL* 值与字符串“NULL”区别开来。

此外，代表 *NULL* 值的 \N 必须使用大写字母 “N” —— \n 代表的是换行符。

14.4.3 用 *LOAD DATA INFILE* 命令导入

LOAD DATA 命令的语法如下所示：

```
LOAD DATA [loadoptions] INFILE 'filename.txt' [duplicateoptions]
    INTO TABLE tablename [importoptions] [IGNORE ignorenr LINES]
    [(columnlist)]
```

1. 原书这里出现严重错误！原文说的是“……要用双引号括起来”，但 MySQL 的导出操作没有这种行为。

这条命令将把 `filename.txt` 文件导入 `tablename` 数据表。这里有许多选项可供选用（详见第 21 章），还可以在 `columnlist` 部分有选择地列出一些 `tablename` 数据表里的数据列的名字，以表明这个文本文件只为这些数据列提供数据。执行这条命令必须具备 `File` 权限（这样才能读取文件）。

如果 MySQL 服务器与 `LOAD DATA` 命令不在同一台计算机上执行，`LOAD DATA` 命令的默认行为是到 MySQL 服务器主机的文件系统里寻找和读取文件。这往往是不正确的。如果想导入的是本地文件系统里的文件，就必须使用语法变体 `LOAD DATA ... LOCAL INFILE ...`。

如果打算导入的文本文件里包含有 ASCII 字符集无法表示的字符，就必须在执行 `LOAD DATA` 命令之前先用 `SET NAMES` 命令对其内容文本的字符集做出必要的设置。

1. 示例1

下面这个例子需要假设 `exceptions` 数据库里的 `importtable1` 数据表已经存在。这个数据表由 5 个数据列构成：一个 `AUTO_INCREMENT` 数据列 (`id`) 和 `a_double`、`a_datetime`、`a_time` 和 `a_text`。后 4 个数据列的数据类型可以从它们的名字上看出来。

将被导入这个数据表的文本文件是 `import-latin1.txt`。在下面给出这个文件的示例内容时，将用“►”符号来代表一个制表符。这个文件使用了 `Latin1` 字符集，各行之间用典型的 Windows 办法即“回车加换行”的办法来分隔。此外，这个文件还包含着一些十进制数值和以不同格式表示的日期值。

```
12.3    ► 12/31/1999 ► 17:30 ► text
-0.33e-2 ► 2000/12/31 ► 11:20 ► "text in quotes"
1,23     ► 31.12.2001 ► 0:13  ► "german text with äöüß"
```

在导入工作正式开始之前，`importtable1` 数据表里的现有内容将被删除，并把字符集设置为 `Latin1`：

```
USE exceptions
DELETE FROM importtable1
SET NAMES 'latin1'
```

导入命令如下所示：

```
LOAD DATA INFILE 'c:/import1-latin1.txt'
INTO TABLE importtable1
FIELDS TERMINATED BY '\t'
ENCLOSED BY '\"'
LINES TERMINATED BY '\r\n'
(a_number, a_date, a_time, a_string)
Query OK, 3 rows affected (0.00 sec)
Records: 3 Deleted: 0 Skipped: 0 Warnings: 3
```

下面是 `SHOW WARNINGS` 命令返回的由本次导入操作引起的警告消息：

SHOW WARNINGS		
Level	Code	Message
Warning	1264	Out of range value adjusted for column 'a_date' at row 1
Warning	1265	Data truncated for column 'a_number' at row 3
Warning	1264	Out of range value adjusted for column 'a_date' at row 3

从 `SELECT` 命令的查询结果可以看出，本次导入操作只取得了部分成功：第 1 行和第 3 行里的日期值显然不是我们想像的样子。此外，第 3 行里的十进制数值也出了问题：按德语习惯使用逗号作为小数点的做法导致小数点后面的数字全部丢失了。这里的教训是：在为导入操作准备文本文件的时候，一定要遵守 MySQL 对各有关数据格式的规定。

如果打算在一个输入窗口或控制台窗口里用 `mysql` 程序执行下面这条 `SELECT` 命令，必须在执行这条 `SELECT` 命令之前为那个窗口设置一个适当的字符集（比如说，在 Windows 环境下，把这个字符集设置为 `cp850`；在最新版本的 Linux 环境下，把它设置为 `utf8`）。

```
SET NAMES utf8
SELECT * FROM importtable1
id a_number a_datetime          a_time    a_string
1   12.3   0000-00-00 00:00:00  17:30:00  text
2   -0.0033 2000-12-31 00:00:00  11:20:00  text in quotes
3     1 0000-0000 00:00:00  00:13:00  german text with äöüß
```

2. 示例2 (BLOB数据、NULL值)

第2个例子将使用 *importtable2* 数据表来进行。这个数据表有 *id* 和 *a_blob* 两个数据列，第2个数据列允许包含 *NULL* 值。

将被导入这个数据表的是一个内容如下所示的 UNIX 文本文件 (“►” 符号仍代表着一个制表符):

```
1 ► NULL
2 ► "NULL"
3 ► \N
4 ► "\N"
5 ► \n
6 ► "\n"
7 ► 0x414243
8 ► "0x414243"
9 ► blob blob
10 ► "blob blob"
```

下面是用来导入这个文本文件的命令。(在默认的情况下，反斜线将被解释为转义前导字符，制表符将被解释为数据列分隔符。) *REPLACE* 关键字的意思是用新导入的记录替换掉数据表里的现有数据记录。

```
USE exceptions
LOAD DATA INFILE '/tmp/import2.txt' REPLACE
INTO TABLE importtable2
FIELDS ENCLOSED BY '\'
Query OK, 20 rows affected (0.01 sec)
Records: 10 Deleted: 10 Skipped: 0 Warnings: 0
```

为了更细致地对本次导入操作的结果进行分析，我们使用了一条 *SELECT *, LENGTH(a_blob), ISNULL(a_blob) ...* 命令来列出这个数据表的内容。查询结果中的后两个数据列可以帮助我们把 *NULL* 值(长度为空值 *NULL*) 和字符串 “*NULL*”(长度为4) 区分开来:

```
SELECT *, LENGTH(a_blob), ISNULL(a_blob) FROM importtable2
id a_blob      LENGTH(a_blob)  ISNULL(a_blob)
1  NULL        NULL          1
2  NULL        4             0
3  NULL        NULL          1
4  NULL        NULL          1
5    1           1             0
6    1           1             0
7  0x414243    8             0
8  0x414243    8             0
9  blob blob   9             0
10 blob blob   9             0
```

正如刚才讲过的那样，在前6条记录当中，第1条、第3条和第4条记录里的 *NULL* 值都得到了正确的处理；第2条记录实际存储的是字符串 “*NULL*”；第5条和第6条记录实际存储的都是一个换行符。

不过，以十六进制数值的格式读入二进制对象的尝试显然失败了：第7条和第8条记录里实际上存储的都是字符串 0x414243 (文本文件里的第8行数据被解释为字符串并不出乎意料，但文本文件里的第7行数据其实是字符串 ABC 的十六进制编码)。

3. 导入CSV格式的文本文件

有时候，人们可能需要把一些来自于 Excel 等电子表格软件的数据导入 MySQL。绝大多数电子表

格软件都可以把数据保存为 CSV (comma-separated value, 用逗号分隔的数据值) 格式。从理论上讲, 把 CSV 格式的文件导入 MySQL 并不困难: 如果想把在 Windows 环境下用 Excel 创建的 CSV 文件导入 MySQL, 只要使用如下所示的导入选项即可:

```
FIELDS TERMINATED BY ',' ENCLOSED BY '\"'
LINES TERMINATED BY '\r\n'
```

不过, 在实际工作中, 把 CSV 文件导入 MySQL 往往会由于日期值的格式而导致一些问题 (这主要是因为 MySQL 无法识别 12/31/2003 格式的日期值)。此外, Microsoft 软件按各国习惯对数值格式自动进行调整的功能——这种功能会让 Excel 把十进制数值中的小数点替换为逗号——也容易引起问题。对于这种情况, 如果找不到专用的 MySQL 导入工具来解决, 就只能自行编写一个 Excel 导出工具了。

14.4.4 用 mysqlimport 工具导入

在某些场合, 可能需要通过一个外部程序去执行 SQL 命令 *LOAD DATA*, MySQL 为此提供了一个名为 mysqlimport 的工具 (它依赖于 *LOAD DATA* 命令), 这个命令的基本语法如下所示:

```
mysqlimport database_name tablename.txt
```

这条命令将在 *database_name* 数据库创建一个名为 *tablename* 的数据表 (如果它尚不存在) 并把 *tablename.txt* 文件里的数据读入这个数据表。(目标数据表的名字与文本文件的名字相同, 只是没有扩展名而已。因此, 执行 mysqlimport db1 authors.txt 命令将把 authors.txt 文件里的数据读入 *db1* 数据库里的 *authors* 数据表。)

请注意, mysqlimport 程序的默认行为是到 MySQL 服务器主机的文件系统里寻找和读取文件。如果正在使用着另一台计算机并希望读入一个本地文件, 就必须用上 *--local* 选项。那个文本文件的字符集可以用传统的 *--default-character* 选项来设置。

--fields-terminated-by、*--fields-enclosed-by*、*--fields-escaped-by* 和 *--lines-terminated-by* 选项与我们在前面介绍过的 SQL 选项相对应。这些选项都必须用引号括起来, 就像 “*--fields-enclosed-by=+*” 这样。

如果想用 mysqlimport 工具来完成刚才在“示例 2”部分中演示的导入操作, 必须先把源文本文件和目标数据表的名字统一起来 (具体到“示例 2”, 它们的名字分别是 *importtable2.txt* 和 *importtable2*), 然后执行如下所示的命令 (因为页面宽度的限制, 我们在此用 “\” 字符串把这条命令写成了两行):

```
root# mysqlimport --local "--fields-enclosed-by="" "
exceptions /tmp/importtable2.txt
```

在 Windows 环境下 (也就是在导入一个 Windows 文本文件的时候), 需要使用如下所示的导入命令 (因为页面宽度的限制, 在此用 “\” 字符串把这条命令写成了两行):

```
> mysqlimport --local "--fields-enclosed-by="" "
"--lines-terminated-by=\r\n" exceptions C:\importtable2.txt
```

14.4.5 用 *SELECT ... INTO OUTFILE* 命令导出

SELECT ... INTO OUTFILE 命令与普通的 *SELECT* 命令基本一样, 只是在 *FROM* 子句前增加了一个用来指定输出文件名的 *INTO OUTFILE* 子句和几个与导出操作有关的选项而已。下面是它的语法:

```
SELECT [selectoptions] columnlist
INTO OUTFILE 'filename.txt' exportoptions
FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ... LIMIT ...
```

这条命令将把 *SELECT* 查询结果存入 *filename.txt* 文件。与特殊字符有关的各个选项可以在 *exportoptions* 部分给出（参见第 21 章）。

示例

下面这个例子需要假设当前数据库里存在着一个名为 *exporttable* 的数据表，它有如下所示的内容（请特别留意第二条记录的 *a_char* 字段）：

```
SELECT * FROM exporttable
id a_char    a_text      a_blob      a_date      a_time    ...
1  char char  text text  blob blob   2001-12-31  12:30:00
2  ' " \ ; +  adsf        NULL       2000-11-17  16:54:54
... a_timestamp    a_float     a_decimal   a_enum     a_set
20001117164643  3.14159    0.012      b          e,g
20001117165454  -2.3e-037   12.345     b          f,g
```

各数据列的类型可以从它们的名字上看出来。下面是使用默认设置进行 *OUTFILE* 导出时的 SQL 命令及其执行结果（“►”符号仍代表制表符）。结果文件只有两行数据，但因为页面宽度的原因，它们显示成了 4 行：

```
SELECT * INTO OUTFILE '/tmp/testfile.txt'
FROM exporttable
1 ► char char ► text text ► blob blob ► 2001-12-31 ► 12:30:00 ► 20001117164643 ► →
3.14159 ► 0.012 ► b ► e,g
2 ► ' " \ ; + ► adsf ► \N ► 2000-11-17 ► 16:54:54 ►
20001117165454 ► -2.3e-037 ► 12.345 ► b ► f,g
```

在下面的第二次尝试里，将使用分号作为字段（数据列）之间的分隔符，将把所有的字段值用双引号括起来，并使用反斜线作为转义前导字符（这是默认设置）。请注意 *a_char* 字段的变化：

```
' " \ ; + becomes " " \ " \ ; +"
```

在这个字符串里，分号没有任何变化，这是因为双引号字符已经清楚地表明了 *a_char* 字段的结束位置，所以不需要对它进行转义；双引号被转义为 “\"”，这将确保字符串里的双引号字符不会被错误地解释为字符串的结束标记；反斜线字符本身被转义为 “\\”。

此外，*NULL* 值被转义为 \N，但它并没有被双引号括起来。

```
SELECT * INTO OUTFILE '/tmp/testfile.txt'
FIELDS TERMINATED BY ';' ENCLOSED BY '\"'
FROM exporttable
1;"char char";"text text";"blob blob";"2001-12-31";"12:30:00";
20001117164643;3.14159;0.012;"b";"e,g"

2;"' " \ " \ ; + ";"adsf";\N;"2000-11-17";"16:54:54";
20001117165454;-2.3e-037;12.345;"b";"f,g"
```

在下面的第三次尝试里，将仍使用分号作为字段（数据列）之间的分隔符；使用加号 (+) 作为可选的字段首尾标记字符（这里所说的“可选”意思是只给字符串、日期、时间和 BLOB 数据加上首尾标记，数值类型的数据仍保持不变）；使用惊叹号 (!) 作为转义前导字符。这里还是需要注意 *a_char* 字段的变化：

```
' " \ ; + becomes +' " \ ; !+
SELECT * INTO OUTFILE '/tmp/testfile.txt'
FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '+' ESCAPED BY '!'
FROM exporttable
```

注意，*NULL* 值将被转义为 !N，因为这次是把惊叹号用做转义前导字符。

```

1;+char char+;+text text+;+blob blob+;+2001-12-31+;+12:30:00+;
20001117164643;3.14159;0.012;+b+;+e,g+
2;+' " \ ; !++;+adsf+;!N;+2000-11-17+;+16:54:54+;
20001117165454;-2.3e-037;12.345;+b+;+f,g+

```

一般来说，只要在导出和导入操作中使用的选项完全一致，用 *SELECT ... INTO OUTFILE* 命令导出的文本文件就可以不发生任何变化地用 *LOAD DATA* 命令导入到数据表里去。

14.4.6 用 mysqldump 程序导出

本章前面介绍过的备份工具 *mysqldump* 程序（另请参见第 22 章）还可以作为 *SELECT ... INTO OUTFILE* 命令的替代品来使用。

mysqldump 程序只能对整个数据表进行操作（不能用来备份或导出某个 *SELECT* 命令的查询结果）。此外，在默认的情况下，*mysqldump* 程序将生成一个包含着 *INSERT* 命令的 SQL 批处理文件而不是一个完全由数据构成的文本文件，前者可以由 *mysql* 程序读入并执行。如果想用 *mysqldump* 程序来生成一个只包含着数据的文本文件，就必须用上 *--tab* 选项，如下所示：

```
mysqldump --tab=verz [options] databasename tablename
```

具体地说，必须用 *--tab* 选项给出一个子目录。*mysqldump* 程序将在这个子目录里为每一个有关的数据表创建两个文件：*tablename.txt* 和 *tablename.sql*。其中，*.txt 文件容纳着数据表里的全部数据，它们与 *SELECT ... INTO OUTFILE* 命令生成的导出文件是一样的；*.sql 文件的内容是一条 *CREATE TABLE* 命令，这条命令可以把数据表重新创建出来。

类似于 *mysqlimport* 程序的情况，*mysqldump* 程序对特殊字符的处理行为也是由 *--fields-terminated-by*、*--fields-enclosed-by*、*--fields-escaped-by* 和 *--lines-terminated-by* 等 4 个选项控制的。这些选项与本节开头介绍的 SQL 选项有着同样的用途，但这些选项都必须用引号括起来，就像 "*--fields-enclosed-by=+*" 这样。因为页面宽度的限制，下面这个例子里的 *mysqldump* 命令写成了两行：

```
C:\> mysqldump -u root -p --tab=C:\tmp "--fields-enclosed-by="""
      exceptions exporttable
Enter password: *****
```

用 mysqldump 程序导出 XML 格式的文件

如果在执行 *mysqldump* 程序时使用了 *--xml* 选项，将获得一个 XML 格式的结果文件。在默认的情况下，*mysqldump* 程序将使用 Unicode 字符集（UTF8）来生成结果文件，但可以用 *--default-character-set* 选项另行指定一个字符集：

```
root# mysqldump -u root -p --xml mylibrary > /tmp/mylibrary.xml
Enter password: *****
```

14.4.7 用 mysql 程序的批处理模式导出

无所不能的 *mysql* 程序可以用来以批处理方式执行多条 SQL 命令，并把它们的返回结果存入一个文本文件。与 *mysqldump* 程序相比，用 *mysql* 程序导出的结果文件往往有着更好的可读性——当然，这是对人类用户而言。（但这种文件不太适合用于导入操作，因为 *mysql* 程序不会为此对导出结果文件做任何优化。）

在最简单的数据导出场合，执行如下所示的 *mysql* 命令即可（因为页面宽度的限制，我们在此用了 “\” 字符把整条命令分成了两行）：

```
> mysql -u root --password=xxx --batch --default-character-set=latin1 \
"--execute=SELECT * FROM authors;" mylibrary > output.txt
```

SQL 命令由--execute 选项负责传递，这个选项必须完整地用引号括起来。在结果文件 output.txt 里，各数据列之间将由制表符隔开；第一行包含着各有关数据列的标题。

1. 查询结果的纵向显示

如果某个数据表有很多的数据列，mysql 程序返回的数据行就会非常长，这样的显示效果是很难解读的。在这类场合，用--vertical 选项代替--batch 选项往往是个不错的主意：数据记录将按纵向排列显示，每个输出行只显示一个字段。实际效果如下所示（请注意，这条命令必须写成一行，但因为本书页面宽度的限制，这里只能把它分断为好几行）：

```
> mysql -u root -p =xxx --vertical
"--execute=SELECT * FROM titles;" \
mylibrary > test.txt
```

文件结果如下所示：

```
***** 1. ROW *****
titleID: 1
  title: Linux, 5th ed.
subtitle: NULL
edition: NULL
publID: 1
  catID: 1
langID: NULL
  year: 2000
  isbn: NULL
comment: NULL
***** 2. ROW *****
titleID: 2
  title: Definitive Guide to Excel VBA
...
...
```

2. 生成HTML表格形式的输出

如果把--batch 选项替换为--html 选项，mysql 程序就将生成一个带数据列标题的 HTML 表格（如图 14-1 所示），但导出的结果文件并不包含任何 HTML 标题。结果文件使用的字符集可以用--default-character-set 选项来设置。下面这条命令必须写在同一行上，但因为本书页面宽度的限制，这里只能把它分断为好几行：

```
> mysql -u root -p=xxx --html
"--execute=SELECT * FROM titles;" \
--default-character-set=latin1 mylibrary > test.html
```

The screenshot shows a MySQL Workbench interface with a results grid titled 'titles'. The grid displays the following data:

titleID	title	subtitle	edition	publID	catID	langID	year
1	Linux	Installation, Configuration, Programming	5	1	57	2	2000
2	The Definitive Guide to Excel VBA	NULL	NULL	2	3	NULL	2000
3	Chaos/Server Survival Guide	NULL	NULL	1	2	NULL	1997
4	Web Application Development with PHP 4.0	NULL	NULL	3	4	NULL	2000
5	MySQL	0	3	34	NULL	NULL	2000

图 14-1 由 mysql 程序创建的 HTML 表格

3. 用mysql程序生成XML格式的输出

类似于mysqldump程序，mysql程序也可以用来创建 XML 格式的文件。下面这条命令必须写在同一行上，但因为本书页面宽度的限制，这里只能把它分断为好几行：

```
> mysql -u root -p --xml --default-character-set=utf8  
    "--execute=SELECT * FROM titles;" mylibrary > C:\test.xml
```

14.5 日志

在数据库领域，日志（logging）的意思是把数据库里的每一个变化记载到一个专用的文件里，这种文件就叫做日志文件。（正如将在本节后面的内容里看到的那样，还有许多其他类型的信息也可以记载到日志文件里去。）下面来看一个例子：如果某个 MySQL 客户程序执行了以下命令：

```
INSERT INTO mydatabase.mytable (col1, col2) VALUES (1, 'xy')
```

MySQL 就将对 *mydatabase.mytable* 数据表做出相应的修改；如果已经启用了日志功能，这条命令本身也将被保存到日志文件里去。

提示 可以在 MySQL 在线文档里找到大量与日志功能有关的信息，下面这个网址就是一个很好的出发点：<http://dev.mysql.com/doc/mysql/en/log-files.html>。

14.5.1 为什么要使用日志

为什么要使用日志？因为日志可以帮助用户达到以下目的（以下讨论仅针对数据库系统，但日志的作用绝非只有这几点）：

- **提高系统的安全性。**数据库系统不可能每时每刻都在进行备份。因此，即使在每天夜里都进行一次备份，万一硬盘在某一天的白天发生了崩溃，在当天对数据库所做的修改就将全部化为乌有。可如果手里还有一个日志文件（当然，它应该在另一块硬盘上）的话，就可以把丢失的数据全部重新创建出来。
日志文件还可以用来查看哪些人在哪些时间使用了数据库、数据库在什么时候发生过不同寻常的事件（如大量的登录失败事件）等。
- **加强对系统的监控。**如果没有日志，就只能了解数据的现在，无法了解它们的过去。比如说，如果数据表里有一个 *TIMESTAMP* 数据列，就可以轻而易举地查出任何一条数据记录的最后一次修改时间，但如果想知道这次修改是谁干的，就无处可查了。如果这类信息对系统很重要，如果必须清楚地掌握谁在什么时间修改了哪些数据，就应该启用相应的日志功能去捕获这些信息。
- **便于对系统进行优化。**如果没有日志，想知道哪些查询命令会给数据库系统带来最大的工作负载将是一件相当困难的事情，而这又往往意味着对系统进行的优化不容易切中要害并达到最佳效果。MySQL 已经考虑到了这一点，它的日志功能可以把运行耗时超过一个特定时间量的查询命令（即所谓的“慢查询”）全部记载下来供人们进行分析。
- **建立镜像机制。**镜像机制说的是在多台计算机上以同步方式运行同一批数据库的过程。要想实现镜像机制，就必须在主控计算机上启用日志功能。（关于镜像机制的详细讨论见 14.6 节。）
- **让事务变得更加没有风险。**日志还可以应用于支持事务的数据表（比如 InnoDB 数据表）。在这种情况下，日志文件可以确保已经提交的事务即使在数据库发生崩溃后也可以正常地执行完毕。（关于 InnoDB 事务日志功能的详细讨论见本章稍后的有关内容。）

根据其具体用途，MySQL 支持多种类型的日志，它们正是本节的讨论重点。在默认的情况下，只要日志功能处于启用状态，日志信息就会源源不断地被写入 MySQL 数据根目录下的日志文件，数据根目录是 MySQL 为各有关数据库创建数据库目录的地方。不过，日志文件的存放地点可以通过各种选项加以改变。如果想获得最高的速度和最大的安全性，就应该把日志文件与数据库文件安排在不同的硬盘上。

14.5.2 缺点与不足

首先，在默认设置下，几乎所有的日志功能都处于禁用状态，唯一的例外是负责记载各种出错情况的出错日志（error log）。不启用这些日志功能的主要理由是为了提高速度：日志功能会对 MySQL 的操作处理速度产生相当大的影响。在速度方面会有多大的损失取决于具体启用了哪几种日志功能和频繁执行的是哪些 SQL 命令。比如说，在变更日志（update log）里，只有那些会导致数据库发生变化的 SQL 命令才会被记载下来（也就是说，变更日志不记载 SELECT 命令）。

第二个缺点是日志文件往往需要占用大量的硬盘空间。在数据修改活动发生得比较频繁的数据库系统里，日志文件占用的空间往往要比数据库本身大得多。

14.5.3 变更日志（update log）

变更日志（update log）负责记载数据库里的变化情况。在过去，MySQL 中的变更日志支持两种格式：文本格式（对应于 log-update 选项）和二进制格式（对应于 log-bin 选项）。从 5.0 版本开始，MySQL 只支持二进制格式的变更日志，不再允许使用 log-update 选项。做出这一决定的理由是二进制格式有许多优点：

- 二进制格式的变更日志文件包含着更多的信息，比如每次修改的时间、客户线程 ID 号等。（如果想知道客户线程 ID 号对应着哪位用户，必须用 log 选项启用另一个日志文件来记载 MySQL 服务器上的每一次用户登录事件。）
- 事务可以得到正确的处理。（事务命令只在 COMMIT 之后才会被记入日志。）
- 日志文件的长度大幅度减少。
- 日志文件上的写操作执行得更有效率。
- 日志文件可以用来实现镜像机制。
- 根据 MySQL 在线文档的说法，MySQL 服务器的操作速度在启用二进制格式的日志功能后只降低了 1%。（不过，MySQL 文档没有说明这个百分比数字是在什么样的测试环境下得到的。在数据修改操作发生得比较频繁的系统上，这个百分比数字很可能会直线上升。）

二进制日志文件的唯一缺点是文件内容无法用一个普通的文本编辑器来查看；不过，它们可以用辅助工具 mysqlbinlog 程序来查看。

MySQL 的每一个主版本（3.n、4.n、5.n）都对二进制日志文件的内部格式做了一些调整。版本比较新的 MySQL 服务器和 mysqlbinlog 程序可以向后兼容老版本的二进制日志文件格式¹，但反向结论不一定成立。

1. 启用二进制变更日志

二进制日志由配置文件里的 log-bin 选项负责启用，必须重新启动 MySQL 才能让这一修改生效：

1. 原书这里把新、老版本的顺序弄混了！——译者注

```
# in /etc/my.cnf (Unix/Linux) or \my.ini (Windows)
[mysqld]
log-bin [=name]
```

重新启动后，MySQL 服务器将在它的数据根目录里创建两个新文件 name-bin.001 和 name-bin.index。如果在 log-bin 选项里没有给出任何文件名，MySQL 将使用服务器主机的名字来命名这两个文件。name-bin.index 文件包含着一份全体日志文件的清单（MySQL 服务器会在每次启动时创建并使用一个新的日志文件）。

如果打算使用镜像机制（详见 14.6 节），就绝对应该在 log-bin 选项里给出一个文件名。如果没有这样做，一旦 MySQL 服务器主机的网络名发生了变化，那两个日志文件的名字也将随之发生变化，而这将导致此前建立的镜像机制无法正常工作。

MySQL 服务器会把用户对所有数据库的内容和结构的修改情况（比如生成了一个新的数据表、增加了一个新的数据列等）记入 name-bin.n 文件。MySQL 只把真实发生过的变化情况记入日志，只会读取数据的 *SELECT* 命令以及没有改变任何数据的 *UPDATE* 命令不会被记入日志。

二进制日志文件的内容可以用 mysqlbinlog 程序来查看：下面是一个例子（因为本书页面宽度的限制，我们把比较长的日志记录项分成了几行，还增加了一些必要的注释）：

```
> mysqlbinlog saturn-bin.002 | less
#050222 16:52:19 server id 1 end_log_pos 53492
#      Query  thread_id=20  exec_time=0    error_code=0
SET TIMESTAMP=1109087539;
SET ONE_SHOT CHARACTER_SET_CLIENT=33,COLLATION_CONNECTION=33,
    COLLATION_DATABASE=8,COLLATION_SERVER=8;
# at 53492
#050222 16:52:19 server id 1 end_log_pos 53601
#      Query  thread_id=20  exec_time=0    error_code=0
SET TIMESTAMP=1109087539;
/*!40000 ALTER TABLE "languages" ENABLE KEYS */;
# at 53601
#050222 16:52:19 server id 1 end_log_pos 53763
#      Query  thread_id=20  exec_time=0    error_code=0
SET TIMESTAMP=1109087539;
SET ONE_SHOT CHARACTER_SET_CLIENT=33,COLLATION_CONNECTION=33,
    COLLATION_DATABASE=8,COLLATION_SERVER=8;
# at 53763
#050222 16:52:19 server id 1 end_log_pos 54171
#      Query  thread_id=20  exec_time=0    error_code=0
SET TIMESTAMP=1109087539;
CREATE TABLE "publishers" (
    "publID" int(11) NOT NULL auto_increment,
    "publName" varchar(60) collate latin1_german1_ci NOT NULL default '',
    "ts" timestamp NOT NULL default CURRENT_TIMESTAMP
        on update CURRENT_TIMESTAMP,
    PRIMARY KEY  ("publID"),
    KEY "publName" ("publName")
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_german1_ci;
```

2. 启用一个新的日志文件

当停止并重新启动 MySQL 服务器的时候，服务器将自动开始把日志数据写入下一个日志文件（name-bin.000002、name-bin.000003 等）。不重新启动服务器也可以切换到下一个日志文件：执行 SQL 命令 *FLUSH LOGS* 或执行外部程序 mysqladmin flush-logs 均可。此外，如果当前日志文件的长度超过了 *max_binlog_size* 选项给出的上限（默认设置是 1GB），MySQL 将自动创建并使用一个新的日志文件。

切换到新日志文件的最佳时机是在对系统进行了一次完整的备份之后。（新的变更日志文件可以

帮助重新构造那些在备份之后发生的数据库变化。)

3. 删除日志文件

不再需要的日志文件应该用 *PURGE MASTER LOGS TO* 命令删除；将在后面介绍镜像机制的时候再对这条命令做进一步的说明。如果想删除所有的日志文件，执行 *RESET MASTER* 命令即可。

4. 利用日志文件去恢复一个数据库

如果不幸遇到灾难事件，应该先用最近一次制作的完整备份恢复数据库，然后再依次使用最近一次备份之后生成的日志文件把数据库恢复到最接近现在的可用状态。后一个步骤可以用 mysql 程序来进行。（一定要注意日志文件的恢复顺序，最早生成的日志文件应该最先恢复。）

```
root# mysqlbinlog name-bin.000031 | mysql -u root -p
root# mysqlbinlog name-bin.000032 | mysql -u root -p
root# mysqlbinlog name-bin.000033 | mysql -u root -p
```

5. 为日志文件指定一个存放路径

可以为日志文件指定一个存放路径（出于信息安全和速度方面的考虑，最好不要把日志文件与数据库存放在同一块硬盘上）：只须在 log-bin 选项里给出一个文件名即可，MySQL 会自动地给这个文件名加上一个.nnn 形式的扩展名（nnn 是一个顺序递增的三位整数）。

请注意，如果 MySQL 服务器运行在 UNIX/Linux 系统上，用来运行 MySQL 服务器的账户（这个账户的用户名通常是 *mysql*）必须有权对 MySQL 日志子目录（它通常是 /var/log/mysql）进行写操作才能让日志功能正常工作：

```
# in /etc/my.cnf or \my.ini (Windows)
[mysqld]
log-bin = /var/log/mysql/myupdatelog-bin
```

6. 数据表类型与日志文件

本节介绍的各种操作适用于各种 MySQL 数据表类型（与数据表类型无关）。不过，有几种数据表类型还有一些它们所独有的日志文件。我们将在本章稍后的内容里对 InnoDB 数据表的日志文件进行介绍。

7. 暂时停用日志功能

如果不想要 MySQL 把将要执行的 SQL 命令记入日志，提前执行一条 *SET SQL_LOG_BIN=0* 命令就可以暂时停用日志功能（必须具备 *Super* 权限才能执行这条命令）。重新启用日志功能的命令是 *SET SQL_LOG_BIN=1*。

8. 其他的日志选项

与日志有关的选项还有很多，它们的用途包括：只对给定数据库启用日志功能、把给定数据库排除在日志功能的覆盖范围之外、为日志文件设定一个最大长度、改变日志文件的同步频率（这里所说的“同步”是指把缓存在内存里的日志信息写入硬盘文件；同步频率越高，系统的安全性就越好，但对速度的负面影响也就越大）。我们在本书的第 22 章对这些选项进行了汇总，在 <http://dev.mysql.com/doc/mysql/en/binary-log.html> 网址处的文档里还可以查到更多的信息。

14.5.4 出错日志、登录日志和慢查询日志

1. 出错日志（error log）

MySQL 会自动地把 MySQL 服务器的每一次启动和关闭事件以及所有的服务器出错消息记入其数据库根目录下的 *hostname.err* 文件里（这里的 *hostname* 代表 MySQL 服务器的主机名）。*FLUSH LOGS* 命令将把这个文件重新命名为 *hostname.old* 文件并开始使用一个新的出错日志文件。

出错日志无法禁用，但可以通过 log-error 选项为出错日志文件另行指定一个文件名和存放地点：

```
# in /etc/my.cnf or my.ini (Windows)
[mysqld]
log-error=/var/log/mysql/mysqlerrorlog
```

2. 登录和操作日志（常规查询日志，general query log）

如果在配置文件里给出了 log 选项，MySQL 就会把来自用户的每一次连接请求以及他们执行的每一条 SQL 命令记入所谓的“登录和操作日志”（也叫“常规查询日志”或“基本查询日志”）。如果没有在 log 选项里给出一个文件名，MySQL 将在其数据库目录下创建一个名为 hostname.log 的文件（hostname 代表 MySQL 服务器的主机名）来存放这类日志信息。这个文件的用途不是为了恢复数据，而是为了监控用户的操作情况，比如哪些用户修改了哪些数据等。

```
# in /etc/my.cnf or my.ini (Windows)
[mysqld]
log
```

下面是这种日志文件里的一段示例内容：

```
/usr/sbin/mysqld, Version: 5.0.2-alpha-standard-log. started with:
Tcp port: 3306 Unix socket: /var/lib/mysql/mysql.sock
Time           Id Command   Argument
050222 11:18:58      1 Connect   root@localhost on
                     1 Query    SET SESSION interactive_timeout=1000000
                     1 Query    SET SESSION sql_mode=''
                     1 Query    SET NAMES utf8
                     1 Query    SHOW VARIABLES LIKE 'datadir'
                     1 Query    SHOW VARIABLES LIKE 'log_error'
050222 11:19:09      1 Query    show databases
050222 11:22:45      1 Query    /*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
                               sql_mode='' */
                     1 Query    SELECT count(*) FROM `mylibrary`.`authors`
                     1 Query    SHOW CREATE DATABASE
                     1 Query    WITH IF NOT EXISTS `mylibrary`
                     1 Query    SET SQL_QUOTE_SHOW_CREATE=1
                     1 Query    SHOW CREATE TABLE `mylibrary`.`authors`
                     1 Query    SELECT /*!40001 SQL_NO_CACHE */ *
                               FROM `mylibrary`.`authors`
                     1 Query    SELECT count(*)
                               FROM `mylibrary`.`categories`
```

与使用变更日志时的做法不同，MySQL 服务器会把自己接收到的每一条命令立刻记入常规查询日志。这里既不考虑那些命令是不是属于一个事务（也就是说，常规查询日志里的信息与事务是否用 COMMIT 命令提交过无关），也不考虑那些命令是不是真的修改了某些数据（也就是说，没有修改任何数据的命令也会被记入通用查询日志）。

3. 慢查询日志

比较复杂的查询操作会让 MySQL 的响应速度变慢。如果想知道哪些 SELECT 查询命令是导致 MySQL 响应速度变慢的“罪魁祸首”并分析其根源，就要使用 log-slow-queries 选项启用它的慢查询日志功能。MySQL 将创建一个名为 hostname-slow.log 的文件（hostname 代表着 MySQL 服务器的主机名）来记载关于慢查询的信息。

如果只设置了 log-slow-queries 选项，MySQL 将只把执行用时超过 long_query_time 选项设置值（默认设置是 10 秒）的查询命令记入慢查询日志。如果还设置了 log_long_format 选项，MySQL 将把如上所述的查询命令以及在执行时没有用到任何索引的查询命令（例如：SELECT * ... WHERE

txtcolumn LIKE '%abc%') 都记入慢查询日志。¹

```
# in /etc/my.cnf or my.ini (Windows)
[mysqld]
log-slow-queries
long_query_time=5
log-queries-not-using-indexes
```

14.5.5 日志文件的管理

日志文件很容易创建，但要想管理好它们并不那么容易。如果没有必要的预防措施，日志文件就会迅速耗尽硬盘的可用空间。因此，一定要本着最少和最需要的原则去选择应该记载哪些日志信息。

1. 变更日志和出错日志的管理

外部程序命令 `mysqladmin flush-logs` 或 SQL 命令 `FLUSH LOGS` 将关闭当前日志文件并开始一个新的变更日志文件（用一个新编号作为文件扩展名）。同时，出错日志文件 `name.err` 将被重新命名为 `name.old`，以后的出错日志信息将被写入一个新的出错日志文件（但文件名仍是 `name.err`）。

除 MySQL 正在使用的各种当前日志文件以外，可以对其他的日志文件进行任何操作。可以把它们移动到另一个子目录、可以对它们进行压缩、甚至可以删除它们（前提是它们已不再有保留价值）。在 UNIX/Linux 系统上，可以创建一个 cron 任务来自动完成这类事情。

另一种办法是先关闭 MySQL 服务器，然后重新命名有关的当前日志文件，再重新启动 MySQL 服务器。

2. 登录与慢查询日志的管理

常规查询日志（对应于 `log` 选项）和慢查询日志（对应于 `log-slow-queries` 选项）存在着这样一个问题：这两种日志文件都只有一个，它们没有像变更日志和出错日志那样的新旧交替机制，而这意味着它们只能越来越大。MySQL 在线文档推荐的办法是：在 MySQL 仍在运行的同时先对这两个文件进行重新命名、然后再执行外部程序命令 `mysqladmin flush-logs` 或 SQL 命令 `FLUSH LOGS`，如下所示：

```
root# cd logpath
root# mv hostname.log hostname.log.old
root# mv hostname-slow.log hostname-slow.old
root# mysqladmin flush-logs
```

现在，可以对*.old 文件进行查看、编辑、压缩、甚至删除等操作了。

另一种办法是：先关闭 MySQL 服务器，然后重新命名有关的当前日志文件，再重新启动 MySQL 服务器。

3. 出错日志的管理²

14.6 镜像机制

14.6.1 简介

镜像机制是一种能够让运行在不同计算机上的两个或更多个 MySQL 服务器保持同步变化的机制。

不同的数据库系统采用了不同的方法来建立镜像机制。如果对另外一种数据库系统的镜像机制很

-
1. 原书在这里有几个地方（见下划线部分）语焉不详，非常容易误导读者，译者进行了订正。——译者注
 2. 原书这一部分完全是误导！它与上面的“变更日志和出错日志的管理”部分以及再稍早一点儿的“出错日志”部分自相矛盾。译者认为它没有任何保留价值，所以把原书这一部分里有用的内容结合到了上面的“变更日志和出错日志的管理”和“慢查询日志的管理”部分里。——译者注

熟悉，不要想当然地认为 MySQL 的镜像机制有着与之完全一样的特点和属性。

MySQL 目前只支持“主-从”镜像关系。这种镜像关系的特点是：只有一台主控系统（可读/可写），所有的数据修改操作都必须在这台系统上进行；有一台或多台从属系统（只读），它们有着与主控系统完全一样的数据，主控系统上的数据变化在经过一个短暂的延时后也将发生在它们身上。

“主-从”镜像关系中的数据同步是通过主控系统的二进制日志文件实现的：主控系统把自己执行过的 SQL 命令记载到自己的二进制日志文件里，从属系统则通过从主控系统的二进制日志文件读出 SQL 命令并加以执行的办法来同步它们自己的数据库。

MySQL 镜像机制不要求主控和从属系统必须使用同一种操作系统（主控系统使用 Linux、从属系统使用 Windows 的情况是允许的）。

提示 除了这一节里的信息，与镜像机制有关的其他SQL命令还可以在第21章查到，与镜像机制有关的mysqld选项可以在第22章查到。

在MySQL文档里可以找到更多的信息：<http://dev.mysql.com/doc/mysql/en/replication.html>。

1. 为什么要建立镜像机制

为什么要建立镜像机制？支持者给出的理由有两点：其一是安全，其二是速度。

安全。镜像机制可以把内容相同的数据库分布在多台计算机上。即使某一台从属计算机出了故障，整个系统也可以继续提供数据库服务，而出故障的从属计算机在问题得到解决并重新投入使用后能够自动同步它自己。

即使主控计算机发生故障，以前的数据也不会丢失，因为从属计算机上都有与之同步的副本。数据库管理员只须重新配置整个系统并把其中一台从属计算机设置为新的主控计算机就可以恢复整个系统的运转。此时有一个细节问题需要注意：从属计算机上的默认设置只允许用户对数据库进行只读访问，如果想让用户还可以对数据库进行修改，就必须对新主控计算机的配置做出相应的调整。¹

如果只是出于安全方面的考虑才打算使用镜像机制，更好的解决方案是建立一个 RAID 系统，也就是对两块或更多块硬盘里的内容进行同步。不过，RAID 系统针对的问题是硬盘崩溃，不是操作系统崩溃、电源故障或其他类似意外。

另外，镜像机制还可以用来取代传统方式的数据库备份工作。（用镜像机制“制作”的备份永远是最新的。如果还需要以传统方式进行备份，完全可以把备份工作安排在一台从属计算机上进行而不去“打扰”主控计算机。）

速度。如果导致数据库系统速度变慢的主要原因是只读方式的查询操作太多（而不是因为数据修改操作太多），为之建立一个镜像系统将显著改善其响应等待时间：把开销昂贵的查询操作分散到多台从属计算机上去执行，让主控计算机只负责执行数据修改操作（主控计算机也可以继续负责执行一些开销不那么巨大的查询操作）。当然，理论上的速度提升幅度会因为通信开销的增加而被抵消一部分。

注意，如果想通过建立一个镜像系统来改善数据库系统的响应速度，客户程序能否支持和兼容镜像机制将是个决定性的因素：客户程序必须具备根据负载情况（或是简单地以随机方式）把查询命令分散到各台从属服务器去的能力；MySQL 本身没有提供这类机制。

如果只是出于速度和性能方面的考虑才打算使用镜像机制，应该首先考虑其他的性能改善措施，

1. 原书对这个细节说得不够到位。——译者注

尤其是硬件方面的升级办法（请按以下顺序考虑：扩充 RAM 内存、使用速度更快的硬盘、建立一个 RAID 系统、使用一个处理器系统）。

2 正 2. 镜像机制的局限性和不足

- MySQL 目前只支持“主-从”镜像关系（对应的术语是单向镜像关系（one-way replication），即所有的数据修改操作必须在那台唯一的主控系统上执行，从属系统只能用来完成数据库查询操作（只读）。目前，在主控系统发生故障的时候，MySQL 的镜像机制还不能把一台从属系统自动切换为新的主控系统（对应的术语是无缝镜像关系（fail-safe replication）。也就是说，MySQL 的镜像机制目前只能保证查询操作不会因为主控系统发生故障而无法执行，数据修改操作还得不到这样的保证。对无缝镜像关系的支持预计将在 MySQL 5.1 版本里得到实现。此外，MySQL 的镜像机制目前还不支持多台主控系统（对应的术语是多源镜像关系（multi-master replication），从属系统不能使用来自多台主控系统的信息进行数据同步。要想实现多源镜像关系，MySQL 还需要解决一些难题，比如怎样才能保证 *AUTO_INCREMENT* 编号值的唯一性等。因此，就目前而言，可以把主控计算机上的某个 MySQL 数据库下载到另一台计算机（比如笔记本电脑）里并对之进行修改，但无法把这些修改同步到主控计算机并进而同步到各台从属计算机去。（如果有足够的权限，通过笔记本电脑连接主控计算机并修改那里的数据库倒是可以。）
- 主控计算机和从属计算机上的 MySQL 版本最好相同。一般来说，为了确保镜像机制能够正常工作，从属系统所使用的数据库软件版本起码不能低于主控系统。
- MySQL 的老版本有许多命令在镜像环境里使用时会引起这样或那样的问题。这类问题中的绝大多数已经在 MySQL 5.0 里得到了解决。作为结论，请尽量使用最新的 MySQL 版本来建立镜像机制。

14.6.2 建立镜像机制的主控系统

本节将介绍如何配置镜像机制中的主控计算机。但这里介绍的步骤并不是唯一的办法，还有其他一些办法也能把初始数据传输给从属计算机。（我们在本章稍后的内容里再介绍另外一种办法。）

本节以及下一节“建立镜像机制：从属系统”中的讨论将假设 *mysql* 数据库里的访问控制信息也将被镜像。这种安排应该说是个好主意，这可以让那些有权在主控系统上读取数据库的用户还能使用同样的访问控制信息去读取从属系统里的数据。

不过，这种安排也有一些缺点，其中最值得关注的是，那些有权在主控系统上修改数据库的用户还能使用同样的访问控制信息去修改从属系统里的数据。我们刚刚讲过，在镜像机制里，所有的数据修改操作都必须在主控系统上进行，否则镜像关系就会遭到破坏。如果数据库管理员决定禁止用户对从属系统里的数据进行修改，就必须把 *mysql* 数据库排除在镜像关系之外（对应的选项是 *binlog-ignore-db=mysql*），并改用在主控系统和每一个从属系统上分别创建和管理一个 *mysql* 数据库的方案。但这一方案又存在着难以同步 *mysql* 数据库的问题——比如说，如果某位用户在主控系统的密码改变了，身为数据库管理员将不得不以手动方式把这位用户的新密码在每一台从属系统上分别设置一遍（这在强行修改了某位用户的密码时还不算是个大问题，麻烦的是用户本人也可以修改自己的密码而数据库管理员无法预知他们会在什么时候这样做）。

1. 为镜像机制创建一个 MySQL 用户账户

首先，需要在主控系统上创建一个 MySQL 用户账户供主控系统和从属系统来进行通信。对这个账户的用户名没有特殊要求，我们在本节讨论内容里将使用 *replicuser* 作为这个账户的用户名。

这个用户必须具备 *Replication Slave* 权限才能按照镜像机制的要求去访问主控系统的二进制日志

文件。在下面的命令里，*slavehostname* 代表着从属计算机的完整主机名或 IP 地址。出于安全的考虑，*replicuser* 用户的密码不应该与用来登录操作系统或数据库的密码相同：

```
GRANT REPLICATION SLAVE ON *.*  
TO replicuser@slavehostname IDENTIFIED BY 'xxx'
```

如果打算在后面的步骤里使用 *LOAD TABLE FROM MASTER* 和 *LOAD DATA FROM MASTER* 命令，还需要把 *Select*、*Reload* 和 *Super* 权限授予 *replicuser* 用户。这些命令是为 MySQL 程序员和专家级用户准备的，它们可以帮助完成镜像系统的创建和管理工作。

如果镜像系统里有多台从属计算机，必须为每一台从属计算机分别执行一次 *GRANT* 命令。当然，也可以把对主控系统的访问权限授予网络里的所有计算机 (*replicuser@'%.netname'*)，这种设置可以简化镜像系统的管理工作，但会带来一些不必要的信息安防风险。

现在，在从属系统上使用 `mysql -u repliuser -p -h masterhostname` 命令来测试一下连接能否成功。这个步骤与镜像机制没有任何关系，其目的只是为了发现可能存在的连接错误。

2. 关闭主控服务器

在配置从属计算机的时候（详见 14.7 节），需要用到一个日志文件名和这个日志文件里的一个起始位置，从属计算机将从这个日志文件里的这个位置开始读取数据。这两项信息可以用如下所示的命令来确定：

```
FLUSH TABLES WITH READ LOCK  
SHOW MASTER STATUS  


| File              | Position | Binlog_Do_DB | Binlog_Ignore_DB |
|-------------------|----------|--------------|------------------|
| saturn-bin.000005 | 375344   |              |                  |


```

把上面这份输出结果里前两个数据列的内容 (*File* 和 *Position*，即文件名和起始位置) 记下来。如果 *SHOW MASTER STATUS* 命令没有返回任何结果（输出结果是空白），就表明主控计算机还没有启用二进制日志功能。接下来，执行 `mysqladmin shutdown` 命令或调用 `/etc/init.d/mysql stop` 之类的 InitV 脚本把这台 MySQL 服务器暂时关停掉。

3. 为主控服务器创建一个快照

现在，需要为主控系统上的各有关数据库创建一个副本（术语称为快照）。在配置从属系统的时候，需要把这份快照安装到每一台从属系统上，作为那些数据库的初始状态。

Windows 用户可以使用 WinZip 或其他压缩工具来制作这份快照；UNIX/Linux 用户最好使用 tar 命令。如此生成的压缩文档应该只包含着 MySQL 数据根目录下的各有关数据库子目录和数据库文件，不要把 MySQL 数据根目录下的各种日志文件也包括进去。

```
root# cd mysql-data-dir  
root# tar cvzf snapshot.tgz
```

如果在主控系统上还有 InnoDB 数据表，这份快照还应该把 InnoDB 表空间文件 (ibdata 文件) 收录进去：

```
root# tar cvzf snapshot.tgz ibdata* /*
```

也可以用 `mysqldump --all-databases --master-data` 命令来创建这份快照（如果涉及到 InnoDB 数据表，请再加上一个`--single-transaction` 选项），但如此生成的结果快照文件往往相当大，因而在主控系统上创建快照时以及在从属系统上用 `mysql` 程序安装快照时需要花费的时间会比较长一些。

4. 修改配置文件

为了让一台 MySQL 服务器成为镜像机制中的主控服务器，还需要通过 *server-id* 选项为它分配一个唯一的 ID 编号。（事实上，镜像机制中的每一台计算机都必须有一个唯一的 ID 编号。）此外，还需要

通过 log-bin 选项启用主控系统上的二进制日志功能。请注意，绝对应该为日志文件明确地起一个名字（我们这里将使用 *mysqlmaster*），这是因为：如果没有这样做，MySQL 服务器将自动使用 *hostname-bin* 作为其二进制日志文件的名字（*hostname* 是这台计算机的主机名）；此后，万一这台计算机的主机名因为网络配置方面的调整而发生了变化，镜像机制就将无法正常工作。

```
# master configuration
# in /etc/my.cnf or my.ini (Windows)
[mysqld]
log-bin=mysqlmaster
server-id=1
```

完成上述准备工作之后，重新启动这台服务器。可以像平时那样使用这台服务器，但用户对数据库做出的所有修改从现在开始将被记入它的二进制日志文件，而从属系统在启动后将自动地根据这些日志文件去同步它们那里的各有关数据库。

提示 本书第22章在介绍mysqld程序的时候还将对其他一些与主控系统有关的选项进行简单的描述。

比如说，可以把镜像关系的覆盖范围限制在某个或某几个特定的数据库（对应的选项是 binlog-do-db）或是把某些特定的数据库排除在镜像关系之外（对应的选项是 binlog-ingnore-db）。

14.6.3 建立镜像机制的从属系统

1. 安装数据库（快照）

如果想让一台以前曾经加入过镜像机制的 MySQL 服务器改用新的配置，先使用 *RESET SLAVE* 命令清除它以前的镜像配置。

首先，关停从属服务器。如果它上面已经有了一些数据库，请把它们移动到一个备份子目录里去（麻烦总比后悔好）。

接下来，用 WinZip 工具或 tar xzf 命令把快照里的数据库文件复制到数据库根目录下。为确保那些文件能够被 MySQL 服务器进行读写，在 UNIX/Linux 系统上还应该使用 chown -R mysql.mysql（适用于 Red Hat 发行版本）或 chown -R mysql.daemon（适用于 SUSE 发行版本）命令改变它们的属主身份。

在 Linux 系统上安装数据库快照的命令如下所示。这里需要假设 MySQL 的数据根目录是 /var/lib/mysql：

```
root# cd /var/lib/
root# mv mysql/ mysql-bak/           creates a backup of /var/lib/mysql
root# mkdir mysql                     creates an empty mysql directory
root# cd mysql
root# tar xzvf snapshot.tgz          opens the snapshot there
root# cd ..
root# chown mysql.daemon -R mysql/  change owner of mysql/
```

2. 修改配置文件

为了建立镜像机制，从属系统上的配置文件也需要稍做修改：用 server-id 选项给从属系统分配一个独一无二的标识号；用 master-host、master-user 和 master-password 选项给出从属系统用来连接主控系统的登录信息。

注解 如果来自主控系统的快照里包含着InnoDB文件，从属系统的配置文件里的InnoDB选项就必须与主控系统保持精确的一致，这样才能确保从属服务器在启动时能够正确识别InnoDB表空间

文件。从属系统上的变更日志功能没有必要启用；从速度优化的角度考虑，还是把这类功能禁用掉更好。mysqld程序还有其他一些与镜像机制中的从属系统有关的选项，对这些选项的简单描述可以在第22章找到。

```
# slave configuration
# in /etc/my.cnf or my.ini (Windows)
[mysqld]
server-id=2
innodb_xxx = <as with the master>
```

完成上述准备工作之后，从属服务器就可以开机启动了。如果在启动时遇到问题，请检查出错日志（这个日志的默认文件名是 `hostname.err`）。

在从属系统的配置文件里还可以加上一个很有用的选项 `read-only`。这个选项的作用是让从属系统只能执行来自主控系统的SQL命令，而这可以确保在从属系统上不会发生主控系统不知道的数据修改操作。

3. 启动镜像机制

在正式启动镜像机制之前，还需要在各个从属系统上分别执行以下命令来确定它们与主控系统之间的镜像关系。请注意，别忘了把以下示例命令中的斜体字部分替换为主控系统的配置数据：

```
CHANGE MASTER TO
MASTER_HOST =      'master_hostname',
MASTER_USER =      'replication user name',
MASTER_PASSWORD =  'replication password',
MASTER_LOG_FILE =  'log file name',
MASTER_LOG_POS =   log_offset
```

假设主控系统的主机名是 `saturn.sol` 并且其他配置数据与在此前的讨论内容里给出的一致，实际执行的命令将如下所示：

```
CHANGE MASTER TO
MASTER_HOST =      'saturn.sol',
MASTER_USER =      'replicuser',
MASTER_PASSWORD =  'xxx',
MASTER_LOG_FILE =  'saturn-bin.000005,
MASTER_LOG_POS =   375344
```

如果在建立镜像机制之前主控系统没有启用过二进制日志功能、或者在建立主控系统的过程中对二进制日志文件进行了重新命名，就需要在以上命令里给出 `MASTER_LOG_FILE = " "`（那是两个单引号字符）和 `MASTER_LOG_POSITION = 0` 选项。

完成上述准备工作之后，只须执行下面这条命令就可以启动镜像机制中的从属服务器了：

```
START SLAVE
```

4. 检查镜像机制是否工作正常

首先，用 `mysql` 程序连接主控系统并在某个数据表里插入一条新记录；然后，用 `mysql` 程序连接从属系统并查看新记录是否已经出现在了那里；如果是，就表明镜像机制建立成功。（当然，也可以通过在主控系统上先创建、再删除几个新数据表或数据库的办法来进行这种测试。如果从属系统能够正确地反映出这些变化，就表明一切正常。）

另一种测试办法是查看从属系统上的出错日志文件（这个文件的默认文件名是 `hostname.err`）。如果一切顺利，应该看到如下所示的镜像状态信息：

```
050317 18:49:24 [Note] Slave SQL thread initialized, starting replication
in log 'FIRST' at position 0,
relay log './uranus-relay-bin.000001' position: 4
050317 18:49:24 [Note] Slave I/O thread: connected to master
```

```
'replicuser@merkuri.sol:3306',
replication started in log 'FIRST' at position 4
```

作为更进一步的测试，还可以先暂时关停从属系统，再在主控系统上执行一些数据修改操作，然后再重新启动从属系统。如果一切正常，从属系统应该在几秒钟之内自动完成对有关数据的同步。

14.6.4 用 *LOAD DATA* 命令建立镜像机制

LOAD DATA FROM MASTER 命令可以把镜像系统的建立工作变得更加简单一些。但这个办法必须满足以下条件才能使用：

- 主控系统上的数据表必须全部都是 MyISAM 数据表。
- *mysql* 数据库必须排除在镜像关系之外。（*LOAD DATA* 命令不对 *mysql* 数据库进行镜像。因此，这个数据库必须已经存在于从属系统上。）
- 主控系统的 MySQL 配置文件必须包含 *log-bin* 选项和一个独一无二的 *server-id* 设置。
- 从属系统的 MySQL 配置文件也必须包含一个独一无二的 *server-id* 设置。

下面是利用 *LOAD DATA* 命令建立镜像机制的基本步骤。

主控系统。 在上述条件全部得到满足的前提下，镜像机制的建立工作将非常简单。首先，在主控系统上为镜像机制创建一个 MySQL 用户账户，并把 *Select*、*Reload* 和 *Super* 权限属性授予它：

```
GRANT SELECT, RELOAD, SUPER, REPLICATION SLAVE ON *.*  
TO replicuser@slavehostname IDENTIFIED BY 'xxx'
```

如果主控系统中的数据库是完整的，那么变量 *net_read_timeout* 和 *net_write_timeout* 中的值必须增加。（其默认值是 30。镜像系统建立起来之后，变量必须重置为默认值。）

```
SET GLOBAL net_read_timeout=600  
SET GLOBAL net_write_timeout=600
```

从属系统。 在从属系统上，先用以下命令为镜像机制设置好主机名、用户名和密码：

```
CHANGE MASTER TO  
MASTER_HOST = 'saturn.sol',  
MASTER_USER = 'replicuser',  
MASTER_PASSWORD = 'xxx'
```

然后，还是在从属系统上，适当加大系统变量 *net_read_timeout* 和 *net_write_timeout* 的设置值：

```
SET net_read_timeout=600  
SET net_write_timeout=600
```

最后，用下面这条命令把所有的数据库和数据表从主控系统传输到从属系统并启动镜像机制：

```
LOAD DATA FROM MASTER
```

如果在执行这条命令的时候发生错误，事情就比较复杂了：必须关停从属服务器、删除所有已经被传输过来的数据库子目录，然后再重复以上几个步骤。比较常见的问题有：主控系统和/或从属系统上的 *net_read_timeout* 和 *net_write_timeout* 变量设置得不够大；镜像机制还涉及到了 InnoDB 数据表；使用了设置不当的 *replicate-ignore-xxx* 选项（这种选项的作用是把给定的数据表或整个数据库排除在镜像关系之外，它们会对 *LOAD DATA FROM MASTER* 命令产生影响，参见第 22 章）。

14.6.5 内部镜像机制

1. *master.info* 文件（从属系统）

在首次启动镜像机制的时候，从属系统会在它的数据根目录下创建一个 *master.info* 文件。

MySQL 会把以下信息记入这个文件：当前使用的二进制日志文件是哪一个、从这个文件的哪个位置开始进行数据同步、如何与主控系统建立连接（主机名、用户名、密码）等。这个文件是镜像机制维持正常运转所不可缺少的，所以 MySQL 的在线文档特别强调了不要随意更改这个文件。下面是这个文件的样板内容：

```
14
masterbinlogname.000007
265
saturn.sol
replicuser
saturn
3306
60
0
```

注解 master.info 文件的内容可以用 *SHOW SLAVE STATUS* 命令来查看，用 *CHANGE MASTER TO* 命令来修改。对这两个命令的详细介绍见第 21 章。

2. 中继文件（从属系统）

进入镜像工作状态之后，在从属计算机的数据根目录下将出现 relay-log.info、hostname-relay-bin.index 和 hostname-relay-bin.nnn 等几个文件。这几个文件是由从属服务器上的另外一个 I/O 线程（子进程）根据主控服务器二进制日志文件的一个副本创建出来的，而该 I/O 线程的唯一任务就是把有关数据从主控服务器复制到从属服务器上的这几个文件里去。包含在这几个日志文件里的 SQL 命令将由另一个 SQL 线程负责执行。

在默认的情况下，从属服务器每执行完一条命令，就会把这条命令从中继文件里删除掉。如果不希望让它这样做，用 relay-log-purge=0 选项禁用这种默认行为即可。

3. 把数据库和数据表排除在镜像关系之外

如果不希望主控系统上的某些数据库或数据表被镜像到从属系统，有两个办法可以把它们排除在镜像关系之外：其一是在主控系统的配置文件里禁止主控系统把某个或某几个数据库里的数据变化情况记入二进制日志文件（通过 binlog-ignore-db 选项）；其二是在从属系统的配置文件里禁止从属系统对某个或某几个数据库或数据表里的数据变化情况进行镜像同步（通过 replicate-ignore-table、replicate-wild-ignore-table 和 replicate-ignore-db 选项）。

4. 暂时禁用镜像机制（主控系统）

如果想在主控服务器上执行一些 SQL 命令但不希望它们被镜像到从属服务器上去，就需要事先执行一条 *SET SQL_LOG_BIN = 0* 命令，等事后再执行一条 *SET SQL_LOG_BIN = 1* 命令。（这两条命令只有具备 *Super* 权限的用户才能使用。）

5. 结束主控服务器和从属服务器的运行

镜像机制中的主控服务器和从属服务器在运行时是彼此独立的，可以按任意顺序来启动和停止它们，而不必担心那会导致数据丢失。如果从属服务器无法与主控服务器建立连接（或是在掉线后再次连接），它将每隔 60 秒重复尝试一次。一旦连接建立成功，从属服务器将从主控系统的二进制日志文件读回自己尚未对之进行同步的数据修改操作并加以执行。即使从属服务器“休息”了很长一段时间，落后了一大段距离，它在重新加入镜像机制后也会像这里描述的那样迅速追赶上米。

6. 多台从属服务器与镜像链

MySQL 的镜像机制允许任意多台从属系统去访问同一台主控系统。站在主控系统的立场上看，

除了访问控制方面的开销会有所增减以外，多一台或多一台从属系统并没有什么不同。

MySQL 的镜像机制还可以用来创建 $A \rightarrow B \rightarrow C$ 形式的镜像链： B 是 A 的从属系统，但它同时又是 C 的主控系统。镜像链通常只会增加开销，没有太大的实用价值。但在某些特殊场合——比如在 A 到 B 之间的网络连接速度很慢（例如，它们一个在欧洲，另一个在美国）、 B 到 C 之间的网络连接却很快的时候——还是有用的。作为镜像链的中间环节，在计算机 B 上必须使用 `log-slave-updates` 选项。

7. 镜像机制与事务

只有在主控系统上用 `COMMIT` 命令提交过的事务才会在从属系统上得到执行。如果主控系统上的某个事务最终被 `ROLLBACK` 命令撤销了，这个事务里的 SQL 命令就根本不会被记入日志文件，当然也就不会在从属系统上得到执行了。

甚至可以在主控系统上使用一种支持事务的数据表格式（InnoDB），在从属系统上使用一个普通的 MyISAM 数据表来与之对应——因为事务的执行和管理行为全部发生在主控系统上，所以在从属系统上就没有必要再启用对事务的支持了。不过，如果想做出这种安排，就必须在启动镜像机制之前对从属系统做出必要的设置，而且还需要特别注意以下两个问题：首先，如果从属系统上的各有关数据表的初始状态是来自一个文件快照，它们的格式就将与主控系统上的对应数据表保持一致，此时必须以手动方式明确地把从属系统上的各有关数据表转换为 MyISAM 格式；其次，在镜像机制启动之后，一定要尽量避免在主控系统上对数据表的格式进行修改，因为那些变化会被镜像到从属系统上，而这意味着为实现这种安排而做出的努力全都白费了。

14.6.6 客户端编程

如果建立镜像机制的目的是为了把查询命令分散到多个系统以预防系统故障或提高查询效率，还需要对客户程序做出必要的修改。

这里的关键是要让客户程序在准备连接 MySQL 服务器时能够进行这样一个判断：这次操作是为了对数据进行查询、还是为了对数据进行修改。如果将要执行的 SQL 命令是 `INSERT`、`UPDATE` 或 `DELETE`，那么本次连接的目标就应该是主控系统。

反之，如果只是为了进行查询，本次连接的目标就应该是当前最不忙碌的从属系统。不过，在实际工作中，因为客户程序无法预知哪个从属系统最不忙碌，所以比较实用的办法是让 `Connect()` 函数从一份预先定义好的主机名或 IP 地址清单里随机选择一台计算机去尝试与之建立连接，如果与那台计算机的连接尝试没有获得成功，`Connect()` 函数应该随机选择另一个服务器去尝试与之建立连接。

1. 随机选择一个服务器并尝试与之建立连接

在下面这段 PHP 示例代码里，我们将假设后续操作是为了读取数据（而不是为了修改数据）；为了提高效率，这里将以随机方式选择一个服务器去与之建立连接。这段示例代码先定义了一个 `mysqlhosts[]` 数组，这个数组的内容是一份由各有关服务器的主机名（或 IP 地址）构成的清单，然后利用 `rand(min, max)` 函数从这个数组里随机选取了一个元素。

与随机选取的服务器成功地建立起连接之后，这段示例代码将执行由 `mysql_list dbs` 变量提供的 SQL 命令 `SHOW DATABASES`：

```
<html><head><title>test</title></head><body>
<?php
$mysqlhosts[0] = "venus.sol";           // list of all servers
$mysqlhosts[1] = "mars.sol";            // of the replication system
$mysqlhost = $mysqlhosts[rand(0,1)];    // select the server randomly
$mysqluser = "user";                   // user name
$mysqlpasswd = "xxx";                  // password
```

```

$connID=mysql_connect($mysqlhost, $mysqluser, $mysqlpasswd);
...
?>
</body></html>

```

2. 出错处理

第 2 个例子与上面那个例子的思路很相似，但这次增加了一些出错处理代码，以避免因连接失败而导致整个程序发生崩溃。下面这段代码将尝试与随机选取的服务器进行 10 次连接，如果这些尝试都没有获得成功，则显示一条出错消息后退出：

```

$tries=0;
while($tries<10 && !$connID) {
    $mysqlhost=$mysqlhosts[rand(0,1)];
    $connID = @mysql_connect($mysqlhost, $mysqluser, $mysqlpasswd);
    $tries++;
}
if(!$connID) {
    echo "<p>Sorry, no database server found.\n";
    echo "</body></html>";
    exit();
}

```

14.7 管理 MyISAM 数据表

在第 8 章曾经讲过，每个 MyISAM 数据表对应着两个硬盘文件，即 dbname/tablename.MYD（用来存放数据）和 dbname/tablename.MYI（用来存放索引）。这使得我们可以通过简单的复制操作来复制/移动各有关数据表和数据库。注意，这类操作只允许在服务器没在使用那个数据库时才能进行（最好能先停止服务器的运行、再进行这类操作）。

必须直接对 MyISAM 数据表文件进行操作的场合相当少见，但并非不可能发生；比如说，在需要重建索引或是需要修复受损文件的时候，我们往往只能采取这个办法。本节将向大家介绍几个可以帮助管理好 MyISAM 数据表的工具。

14.7.1 myisamchk 程序

说到管理 MyISAM 数据表，myisamchk 程序大概是用途最广泛的工具了。这个程序可以帮助我们完成下面这些管理工作：

- 检查 MyISAM 数据表里的数据是否完好无损。
- 修复受损的 MyISAM 数据表文件（比如说，在发生供电中断故障之后）。
- 释放 MyISAM 文件内的未使用存储空间。
- 为 MyISAM 数据表重新创建索引。

除了 myisamchk 程序，还可以使用以下 SQL 命令：

- ANALYZE TABLE*。这个命令可以提供关于索引内部管理状况的信息。
- CHECK TABLE*。检查数据表文件是否完好无损。
- OPTIMIZE TABLE*。对数据表的存储空间使用情况进行优化。
- REPAIR TABLE*。尝试修复受损的数据表。

OPTIMIZE TABLE 和 *REPAIR TABLE* 命令只能用于 MyISAM 数据表。使用 SQL 命令的最大好处是根本用不着担心它们会像 myisamchk 程序那样与 MySQL 服务器发生这样或那样的冲突，但这么做的缺点似乎要更多一些：MySQL 服务器必须处于运行状态（这一要求在系统发生崩溃后往往无法得

到满足)才能使用它们、有时无法查出和修复所有的错误、可以用来控制操作过程的选项比较少、执行速度比较慢等。

经常与 myisamchk 程序配合使用的另一个工具是 myisampack 程序, 它可以对 MyISAM 数据表进行压缩。这样可以节约大量的空间, 但经过压缩的 MyISAM 数据表将只允许用户进行读操作。我们将在本节的末尾对 myisampack 程序做进一步的介绍。

提示 对 myisamchk 和 myisampack 程序的所有选项的介绍可以在第 22 章查到。myisamchk 程序的使用方法(尤其是在修复受损的数据表文件方面)可以在 MySQL 在线文档里查到: <http://dev.mysql.com/doc/mysql/en/table-maintenance.html>。

14.7.2 myisamchk 程序的使用方法

myisamchk 程序的命令语法如下所示:

```
myisamchk [options] tablename1 tablename2 ...
```

myisamchk 命令中的文件名参数既可以是 MyISAM 数据表的名字, 也可以是 MyISAM 索引文件的名字(*.MYI), 但不能使用 MyISAM 数据文件的名字(*.MYD)。但根据具体使用的命令行选项, myisamchk 程序也许会对*.MYI(索引)和*.MYD(数据)文件都进行分析或修改。

下面这条 myisamchk 命令将对 *mydatabase* 数据库里的所有数据表进行检查(在使用 myisamchk 程序时请不要忘记把 /var/lib/mysql 子目录替换为自己的 MySQL 数据根目录):

```
root# myisamchk /var/lib/mysql/mydatabase/*.MYI
```

myisamchk 程序不依赖于 MySQL 服务器, 不管 MySQL 服务器是否正在运行都可以使用。不过, 如果 MySQL 服务器正在运行, 就必须在执行 myisamchk 命令之前先执行一次外部程序命令 mysqladmin flush-tables 或 SQL 命令 FLUSH TABLES。

注意 如果打算使用 myisamchk 程序去改变 MyISAM 文件而不仅仅是对它们进行分析, 就一定要保证 MySQL 服务器在此期间不会对相应的 MyISAM 数据表做出任何修改。

也就是说, 如有必要, 必须先使用 mysql 程序执行一次 SQL 命令 LOCK TABLE, 然后再去执行 myisamchk 命令, 最后再执行一次 SQL 命令 UNLOCK TABLE。在此期间, 千万不要退出 mysql 程序——退出 mysql 程序将使 LOCK TABLES 命令失去效力。

14.7.3 速度优化与内存使用情况

对体积比较庞大的数据表进行分析是一项开销非常巨大的操作, 对它们进行修复的开销就更大了。可用 RAM 内存量的大小对 mysqisamchk 程序的执行速度有着非常大的影响。

mysqisamchk 程序的内存使用情况由 4 个变量控制。在默认的情况下, mysqisamchk 程序需要大约 3MB 的 RAM 内存才能正常运行。如果计算机的内存比较大, 就应该适当加大那 4 个变量的设置值, 这可以让 mysqisamchk 程序更快地完成对大数据表的分析和修复工作。下面是 MySQL 在线文档给出的推荐设置:

```
root# myisamchk -sort_buffer=64M -key_buffer=64M \
    -read_buffer=1M -write_buffer=1M ...
```

此外，在对数据库文件进行修复的时候，myisamchk 程序还需要占用大量的硬盘空间，这是因为，别的先不说，myisamchk 程序会首先为数据库文件制作一份复制副本。myisamchk 程序会把这份副本放在环境变量 TMPDIR 指定的临时子目录里；可以通过--tmpdir 选项另行指定一个临时子目录。

14.7.4 对 MyISAM 数据表进行压缩和优化

MyISAM 数据表驱动程序会尽量把 MyISAM 数据表文件的长度控制在最小，但如果在数据表里删除过大量的数据记录、或者如果经常修改的数据记录包含着一些可变长度的数据列（*VARCHAR*、*xxxTEXT*、*xxxBLOB*）的话，MyISAM 数据表驱动程序的内存优化算法就无能为力了。在最坏的情况下，数据库文件的实际长度会远远超出它的实际需要。这不仅浪费内存，还会导致在数据表里连续排列的数据在数据表文件里散布得到处都是，这会使数据库的存取操作速度变慢。

下面给出的命令提供了一些帮助。它将重新生成数据库文件并按照“使数据库存取速度最快”的要求对索引文件进行优化。这里还可以使用--set-character-set 选项来设置字符集和排序方式。--check-only-changed 选项的含义是，只对此前最后一次用 myisamchk 程序处理这个数据库后又发生了变化的数据表进行处理。（下面这条命令应该写在一行上，但因为本书页面宽度的限制，这里只能把它写成两行。）

```
root# myisamchk --recover --check-only-changed --sort-index \
--analyze databasepath/*.MYI
```

提示 如果有兴趣测试一下 myisamchk 程序的性能，可以这样来创建一个测试数据库：先把这个测试数据库填满数据，然后用如下所示的命令随机地删除它里面的一半数据记录：DELETE FROM testtable WHERE RAND()>0.5。

14.7.5 修复受损的 MyISAM 数据表

因为笔者未曾遇到过 MyISAM 数据表发生损坏的问题，所以本节内容完全是理论性探讨。可能导致文件受损的常见原因包括数据库因供电突然中断而停止、MySQL 或操作系统发生崩溃、MyISAM 数据表驱动程序内部执行出错等。

如果 MyISAM 文件受损，用户在访问相应的数据表时就会看到 MySQL 报告的出错消息 *Index-file / Record-file / Table is crashed*（“索引文件/数据文件/数据表遭到破坏”）或 *No more room in index/record file*（“索引文件/数据文件空间不足”）。受损文件可以用 myisamchk 程序来修复，但它并不能创造奇迹——如果打算修复的数据已经不复存在或是已被其他数据覆盖，再好的工具也无能为力。myisamchk 程序只能把它能够修复的所有数据记录修复到可以被正常读取的程度，修复后的数据记录的内容是否正确要靠自己去检查核对：

```
root# myisamchk --recover databasepath/*.MYI
```

如果怀疑只有索引文件遭到了破坏，可以在执行 myisamchk 程序时加上一个--quick 选项（这个选项可以大大加快修复速度）；此时 myisamchk 程序将只重新生成索引文件。

如果遇到 myisamchk --recover 命令无法完成修复工作的困难局面，可以再用 myisamchk --safe-recover 命令去进行修复，但使用这个选项进行修复工作所花费的时间要比--recover 选项长很多。

14.7.6 压缩 MyISAM 数据表 (myisampack 程序)

如果有一个体积庞大但仅用于查询（不需要再做修改）的数据表，最好对相应的数据表文件进行压缩。这不仅可以节约空间，在许多场合还可以提高访问速度（因为这个数据表现在可以有更多的内容驻留在操作系统的文件缓冲区里）：

```
root# myisampack databaspath/*.MYI
```

请注意，myisampack 程序要求文件名参数必须是索引文件 (*.MYI)，但它实际上只对数据文件 (*.MYD) 进行压缩。如果想对经过压缩的数据表文件进行解压缩，应该使用--unpack 选项来运行 myisamchk 程序。

14.8 InnoDB 数据表的管理

与 MyISAM 数据表相比，InnoDB 数据表能够支持更多的功能（事务、数据一致性规则等），但它们的管理工作相对要复杂一些。在这一节里，将对以下几个问题进行讨论：如何管理 InnoDB 数据表的表空间（tablespace）文件和日志文件，如何复制和移动 InnoDB 数据表，如何为 InnoDB 数据表优化 MySQL 服务器的配置。

提示 SQL命令`SHOW VARIABLES LIKE 'innodb%'`可以把最重要的InnoDB参数列成清单供用户查看。

与InnoDB数据表有关的MySQL服务器配置参数参见第22章。

14.8.1 表空间的管理

MyISAM 数据表的数据和索引都有它们自己的文件，这些文件存放在 MySQL 数据根目录下的一个子目录里，子目录的名字就是 MyISAM 数据表所在的数据库的名字（例如：data/dbname/tablename.myd）。InnoDB 数据表采用的是另一种做法：所有 InnoDB 数据表及其索引都存储在一个虚拟的文件系统里，InnoDB 文档把这个虚拟文件系统称为表空间（tablespace）；表空间本身又有可能由多个文件构成。

如果没有特殊的要求，完全可以随时创建和使用 InnoDB 数据表，而无须对 MySQL 服务器的配置做出任何调整。在默认的情况下，MySQL 服务器会在首次启动时创建一个长度为 10MB 的 ibdata1 文件作为集中存放 InnoDB 数据表的表空间，这个表空间在必要时能够以每次 8MB 的幅度扩张。

表空间文件的特点之一是它们只能增大，不能缩小。换句话说，在用户删除数据表或数据记录的时候，虽然表空间里的空间会被释放并可以用来存储新的数据表，但表空间文件的长度却不会发生变化。

从 MySQL 4.1 版本开始，我们还可以通过一些配置选项让 MySQL 为每个 InnoDB 数据表分别创建一个表空间文件（而不是把所有的 InnoDB 数据表都集中存放在同一个表空间文件里）。关于这个问题的详细讨论见本节稍后的内容。

1. 确定InnoDB数据表的空间需求量

表空间或多或少地有点儿像一个无法看透的黑箱：MySQL 没有提供任何可以用来查看表空间内部的子目录层次结构的命令。不过，`SHOW TABLE STATUS` 命令可以让我们了解到每个 InnoDB 数据表及其索引在表空间文件里占用了多少空间，以及整个表空间在它下次扩张之前还有多少可用空间。在下面的例子里，只保留了 `SHOW` 命令的部分输出结果，与这里的讨论没有太大关系的数据列都省略了。因为 InnoDB 数据表及其索引在存储和管理时都以 16KB 为单位，所以它们的长度在 `SHOW` 命令

的返回结果里总是 16 384 的整数倍数：

```
SHOW TABLE STATUS FROM mylibrary
Name      Engine   Data_Length  Index_Length Comment
authors   InnoDB    16384        16384  InnoDB free: 3072 kB ...
categories InnoDB    16384        49152  InnoDB free: 3072 kB ...
```

在 MySQL 4.n 版本里，表空间文件通常都要比包含着同样内容的 MyISAM 数据表文件长很多。从 MySQL 5.0.3 版本开始，表空间文件使用了一种更为紧凑的默认存储格式，所以这种情况已经有了明显的改善。新格式让表空间文件的空间占用量减少了 20%，但它们的长度还是要比 MyISAM 数据表大不少。还好，新格式在速度方面没有带来任何负面影响。

根据其开发计划，InnoDB 数据表驱动程序将从 MySQL 5.1 版本开始具备对表空间里的各有关页面进行透明压缩的功能。这将大大减少表空间文件的硬盘和内存空间占用量——尽管这需要以增加 CPU 负担为代价。

2. 单个InnoDB数据表的表空间文件

从 MySQL 4.1 版本开始，用户可以选择为每一个 InnoDB 数据表分别创建一个表空间文件。与以前只有一个表空间文件的做法相比，这一改进带来了许多的好处；这也是我们为什么最先介绍这种变化的原因。

为每个 InnoDB 数据表分别创建一个表空间文件的做法目前还不是 InnoDB 数据表驱动程序的默认行为，用户必须通过 `innodb_file_per_table` 选项明确地启用这项功能。不过，在设置了这个选项并重新启动 MySQL 服务器之后，这个选项只对新创建的 InnoDB 数据表起作用，此前创建的 InnoDB 数据表仍将集中存放在原来的表空间里——它现在被称为主空间或主表空间（`masterspace`）。

注意，除了与各有关 InnoDB 数据表分别对应的表空间文件以外，MySQL 服务器上始终会有一个主空间文件，这个主空间文件是 MySQL 服务器在首次启动时在没有使用任何特殊配置的情况下自动创建的。

```
# in /etc/my.cnf or my.ini (Windows)
[mysqld]
innodb_file_per_table
```

重新启动 MySQL 服务器之后，MySQL 服务器将为每一个新创建的 InnoDB 数据表分别创建一个表空间文件，这个表空间文件的文件路径名是 `databasename/tablename.ibd`，路径中的 `databasename` 和 `tablename` 分别是数据库和数据表的名字。这种文件的初始长度只有 64KB，但可以在必要时自动扩张。同一个 InnoDB 数据表的所有数据和所有索引都包含在同一个表空间文件里。

如果想把以前创建的 InnoDB 数据表从主空间转移到它们自己的表空间文件里，请按以下步骤进行：先为这些数据表制作一个备份（用 `mysqldump` 程序），然后删除这些数据表，最后通过恢复备份的方法把它们重新创建出来。

如果后来又从配置文件里删除了 `innodb_file_per_table` 选项，此前创建的那些单个表空间文件将不会受到任何影响，但此后创建的 InnoDB 数据表将被集中存放在主空间里。把单个表空间文件迁移到主空间里的办法和我们上面介绍的步骤完全一样：制作数据表备份、删除数据表、恢复备份。

如果想知道哪些 InnoDB 数据表有自己的表空间文件、哪些 InnoDB 数据表集中存放在主空间里，目前的唯一办法是去直接查看 MySQL 的数据根目录里都存在着哪些 `databasename/tablename.ibd` 文件。`SHOW TABLE STATUS` 命令无法提供这些信息；事实上，MySQL 服务器目前没有提供任何可以用来查看这些信息的其他办法。

单个表空间文件的优点和缺点。根据 InnoDB 开发团队成员 Heikki Tuuri 在一封发给 MySQL 邮件

表的电子邮件里给出的说法，与 MySQL 服务器上只有一个表空间文件时的情况相比，单个表空间文件虽然较慢，但差距非常小。使用单个表空间文件的最大好处是：在用户使用 *DROP TABLE* 命令删除 InnoDB 数据表时，相应的表空间文件也将被删除并立刻释放它们所占用的硬盘空间；这与 MySQL 服务器上只有一个表空间文件时的情况形成了鲜明的对照。¹

注意，与 MyISAM 数据表文件不同，把表空间文件从一个子目录复制或移动到另一个子目录或是把表空间文件从一个 MySQL 服务器复制或移动到另一个 MySQL 服务器都是不允许的。这类操作必须按照本章后面内容里的“*InnoDB* 数据表的复制、删除和移动”小节所给出的步骤进行。

3. 主空间的配置

在没有使用任何配置选项的默认情况下，MySQL 服务器会在首次启动时创建一个长度为 10MB 的 *ibdata1* 文件作为集中存放 InnoDB 数据表的表空间，这个表空间在必要时能够以每次 8MB 的幅度扩张。（为了与那些只对应于单个 InnoDB 数据表的表空间相区别，MySQL 在线文档把这个默认创建的表空间称为主空间或主表空间（*masterspace*）。以下讨论仅适用于那些有兴趣以手动方式对主空间文件的存放路径和大小进行管理的读者（如果没有特殊的必要，最好还是把这些事情交给 MySQL 服务器去负责）。主空间文件的存放路径和大小由 *innodb_data_home* 和 *innodb_data_file_path* 两个选项控制：前一个选项负责为所有的 InnoDB 文件指定一个用来存放它们的子目录（默认设置是 MySQL 的数据根目录）；后一个选项负责设定主空间文件的名字和长度。注意，如果打算使用这两个选项，必须在 MySQL 服务器首次启动之前把它们设置好。下面是这两个选项在 MySQL 配置文件里的用法示例：

```
# in /etc/my.cnf or my.ini (Windows)
[mysqld]
innodb_data_home = D:/data
innodb_data_file_path = ibdata1:1G;ibdata2:1G:autoextend:max:2G
```

上述设置的含义是：主空间将由 D:\data\ibdata1 和 D:\data\ibdata2 两个文件构成，如果这两个文件不存在，则创建它们；这两个文件的初始长度都是 1GB，如果 InnoDB 数据表需要更多的空间来存储，MySQL 服务器将自动地以每次 8MB 的幅度加大 *ibdata2* 文件，但 *ibdata2* 文件的最大长度不得超过 2GB。

如果打算以手动方式去管理 InnoDB 主空间文件，务必注意以下几个问题：

- MySQL 服务器必须具备 *innodb_data_home* 子目录的 *write*（写）权限才能在其中创建和修改 InnoDB 表空间文件。
- 如果在 *innodb_data_file_path* 选项里没有使用 *autoextend* 属性，作为数据库管理员，就必须在适当时刻（在 InnoDB 数据表的总长度超出表空间文件的当前最大长度之前）以手动方式加大表空间里的可用空间长度。要知道，如果 InnoDB 数据表驱动程序在执行一个事务的时候发现表空间已满且无法再扩张，它就会使用 *ROLLBACK* 命令去撤销这个事务。

4. 直接使用硬盘分区作为表空间

InnoDB 数据表驱动程序还可以直接使用硬盘分区作为表空间，也就是说，InnoDB 数据表驱动程序有能力绕过操作系统的文件系统直接对硬盘进行读写。要想做到这一点，必须先对那个硬盘分区进行初始化，这需要在 *innodb_data_file_path* 选项里给出那个硬盘分区的设备名（而不是文件路径）、硬盘分区的长度（这个长度值必须是 1MB 的整数倍数）和一个 *newraw* 后缀。（下面这个示例使用的是 Linux 操作系统的设备名。）

1. 原书在这里出现了明显的笔误，将“……好处……”写成了“……坏处……”！——译者注

```
innodb_data_home_dir=
innodb_data_file_path=/dev/hdb1:61440Mnewraw
```

重新启动 MySQL 服务器以完成对硬盘分区的初始化。接下来，必须关停 MySQL 服务器并把 newraw 后缀替换为 raw（只在添加一个新硬盘分区的时候才需要使用 newraw 后缀），如下所示：

```
innodb_data_home_dir=
innodb_data_file_path=/dev/hdb1:61440Mraw
```

很遗憾，InnoDB 在线文档没有对直接使用硬盘分区作为表空间的做法是否可以提高性能（笔者猜测应该可以）以及能够提高多少等问题给出任何答案，MySQL 邮件表上关于这方面的准确信息也很少。

5. 加大主空间

除了带有 autoextend 属性的最后一个表空间文件（这个属性也只能用于最后一个表空间文件），在配置文件里通过 innodb_data_file_path 选项为创建 InnoDB 主空间而定义的其他表空间文件都不能再进行扩张。如果想加大主空间里的可用空间长度，最简单的办法是在 innodb_data_file_path 选项中再增加一个文件，具体操作步骤是：

- 关停 MySQL 服务器。
- 如果最后一个表空间文件带有 autoextend 属性，必须查出它的实际长度（以 MB 为单位；这个数字可以通过把 DIR 或 ls 命令查到的文件字节长度除以 1 048 576 的办法计算出来）。这个长度值是这次设置 innodb_data_file_path 选项所必需的。如果配置文件里根本没有 innodb_data_file_path 选项，就说明此前使用的是默认的表空间文件 ibdata1。对于这种情况，现在必须查出 ibdata1 文件的实际长度。
- 给 innodb_data_file_path 选项增加一个或多个新文件。在默认的情况下，所有的表空间文件都存放在同一个子目录（或它的下级子目录）里。如果想把表空间文件存放到不同的硬盘分区、不同的硬盘等，就必须在 innodb_data_home 选项处给出一个空白字符串、在 innodb_data_file_path 选项处以绝对路径的方式给出表空间文件的文件名。请注意，innodb_data_file_path 选项的现有文件设置顺序不允许改变（当然，这些文件也一个都不能少）。
- 重新启动 MySQL 服务器。如果服务器对新配置进行检测之后没有发现任何错误，它就会生成新的表空间文件。如果遇到错误，启动过程将把出错信息记入 MySQL 出错日志 hostname.err 文件。

下面来看一个例子：下面是加大主空间之前的设置，ibdata2 文件的当前实际长度是 1904MB：

```
innodb_data_file_path = ibdata1:1G;ibdata2:1G:autoextend
```

决定把表空间的长度加大到 4GB，下面是相应的新设置：

```
innodb_data_file_path = ibdata1:1G;ibdata2:1904M;ibdata3:1100MB:autoextend
```

如果采用了为各个 InnoDB 数据表分别创建一个表空间文件的办法，上面这些麻烦事就都没有了。此时，主空间将只用来存放各种临时数据，数据量相对很小，所以它的实际尺寸也很有限。至于那些分别对应着各个 InnoDB 数据表的单个表空间文件，它们会在必要时自动扩张。

6. 缩小主空间

MySQL 没有提供任何可以用来缩小主空间文件的办法。在用户把一个存放在主空间里的 InnoDB 数据表删掉或是把它转换为另一种数据表格式的时候，它在主空间里占用的空间只是重新成为表空间里的可用空间而已，相应的硬盘空间并没有得到释放，所以表空间文件并不会因此而变小。要想缩小主空间，就只能采用下面这个办法：

- 用 mysqldump 程序为所有的 InnoDB 数据表制作一个备份。

- 删除所有的 InnoDB 数据表 (*DROP TABLE ...*)。
- 关停 MySQL 服务器 (`mysqladmin shutdown`)。
- 删除当前表空间文件 (`ibdata...`)。当然，如果有足够的空间，把这些文件移动到另一个目录的办法更保险。
- 对 MySQL 配置文件里的 `innodb_data_file_path` 选项进行必要的修改。
- 重新启动 MySQL 服务器，按照 `innodb_data_file_path` 选项的新设置把新的表空间文件创建出来。
- 用刚才制作的备份把所有的 InnoDB 数据表重新创建出来。

7. InnoDB数据表的复制、删除和移动

与 MyISAM 数据表打过交道的读者肯定知道这样一个事实：复制或移动 MyISAM 数据表的工作可以通过复制或移动(最好先关停 MySQL 服务器)相应的数据表文件*.MYI 和 *.MYD 来完成，而 MySQL 服务器在重新启动时就能知道哪些数据表可以在哪些地方找到。这一事实为 MyISAM 数据表和 MyISAM 数据库的备份和复制工作提供了极大的方便。

很遗憾，这一切都不能用在 InnoDB 数据表身上——即使在每个 InnoDB 数据表都有它自己的表空间文件的情况下也是如此。在需要移动、复制或删除一个 InnoDB 数据表的时候，必须按照以下方法来进行：

- **移动 InnoDB 数据表。**要想把 InnoDB 数据表从一个数据库移动到另一个数据库，必须使用 SQL 命令 `RENAME dbname.tablename TO dbnamenew.tablenamenew`。
- **复制 InnoDB 数据表。**复制 InnoDB 数据表的办法有两种：其一，先创建一个新的数据表，再用 SQL 命令 `INSERT ... SELECT` 把数据复制过去；其二，先为那个数据表制作一个备份，再用一个新的名字把它重新创建出来。
- **删除 InnoDB 数据表。**删除 InnoDB 数据表必须使用 SQL 命令 `DROP TABLE` 来进行。如果这个 InnoDB 数据表有自己的表空间文件，它们将同时被删除。如果这个 InnoDB 数据表存放在主空间里，它在主空间里占用的空间将重新成为表空间里的可用空间，但主空间文件并不会因此而缩小。
- ***.frm 文件。**注意*.frm 文件。这些文件的内容是数据表的结构定义，它们存放在相关的数据库子目录里(即使对 InnoDB 数据表也不例外)。*.frm 文件与存放在表空间里的相关数据表永远保持着同步。不能简单地删除*.frm 文件。如果想删除一个数据表，执行 `DROP TABLE` 命令即可，相应的*.frm 文件将随之一起被删除。

8. 为InnoDB数据表制作备份

为 InnoDB 数据表制作备份的办法有好几种：

- 最专业的办法是使用 InnoDB Hot Backup 工具。这个辅助程序能够在 MySQL 服务器仍在运行的同时对 InnoDB 数据表进行备份，并且不需要锁定数据表。这个工具是一个需要花钱购买的商业化软件，详细情况请参见 <http://www.innodb.com/order.php>。
- 当然，还可以使用 `mysqldump` 程序。在备份 InnoDB 数据表时，这个程序必须与 `single-transaction` 选项配合使用。
- 如果有足够的权限，先关闭 MySQL 服务器、再直接复制表空间文件也是一个办法。这个办法需要注意以下几个问题：备份时不要遗漏*.frm 文件；在恢复备份时一定要使用与进行备份时完全一样的 `innodb_data_file_path` 设置；不必备份 InnoDB 事务日志(只要 MySQL 服务器是按正常流程关停的，这些日志文件里就不会有什么值得备份的数据)。

9. 迁移表空间

根据 MySQL 在线文档的说法，表空间文件不依赖于具体的操作系统——只要两台计算机里的 CPU 使用的是同样的浮点数表示法，就可以毫无问题地在它们之间迁移 InnoDB 表空间文件。也就是说，只要上述条件得到满足，我们就可以对表空间文件在 Windows 和 Linux 系统之间进行迁移。当然，在具体操作的时候，还需要保证那两台计算机使用着同样的 `innodb_data_file_path` 设置，还需要把所有的*.frm 文件也一起迁移过去。（在实际工作中，分两次对同一台 MySQL 服务器里的 MyISAM 数据表和 InnoDB 数据表分别进行迁移的事情很少见，它们几乎总是一块儿被迁移的。这反而会让迁移工作变得更容易，只要把所有的数据库子目录和所有的表空间文件一起复制下来就行了。）

InnoDB 文档没有对表空间文件的格式是否与 MySQL 服务器/InnoDB 驱动程序的具体版本有关这一问题做任何说明。根据 InnoDB 数据表驱动程序的开发者之一 Heikki Tuuri 的说法，无论是现在还是将来，新版本肯定能够向后兼容老版本。（MySQL 的向后兼容性一直很好，老版本的数据库文件总是能够毫无问题地迁移到新版本环境下。）

无论如何，用 `mysqldump` 程序来制作备份是最保险的办法：由它生成的备份文件是纯文本格式，所以根本不存在兼容性问题。

14.8.2 日志文件

1. InnoDB 日志（事务日志）

InnoDB 数据表驱动程序会把所有的数据修改情况记入所谓的“事务日志”，这份日志由多个顺序编号的文件（`ib_logfile0`、`ib_logfile1` 等）构成。这些文件的用途主要有两个：一是确保大型事务能够顺利完成；二是在发生崩溃后对 InnoDB 数据表进行恢复。

只要 MySQL 服务器的配置得当并且有足够的内存，用户最近访问过的绝大部分数据就应该驻留在系统的 RAM 内存里。出于加快操作速度的考虑，MySQL 总是先对内存里的有关数据进行修改，等积攒到一定程度之后才把它们写入硬盘上的数据文件（对 InnoDB 而言就是表空间）。

只有当一个事务以 `COMMIT` 命令完成之后，它对数据做出的修改才会被写入硬盘，在此之前，那些修改会随着事务的进行先行保存在 `ib_logfile0`、`ib_logfile1` 等 InnoDB 日志文件里。表空间被修改的部分是分次分批地被写入硬盘的，这一切都是为了提高效率。如果系统在这一系列过程中发生了崩溃，表空间可以在有关日志文件的帮助下得到恢复。

`ib_logfile0` 等日志文件有两个用途：一是确保 ACID 原则中的“D”（Durability，可靠性），即确保在系统发生崩溃前提交的事务在系统恢复正常后不受任何影响；二是确保那些临时数据总量超出了可用 RAM 内存总量的超大型事务也能够顺利完成。

InnoDB 日志文件是按顺序填写的。每当最后一个文件被填满时，InnoDB 数据表驱动程序就会再一次从第一个日志文件开始写入数据。因此，这些 InnoDB 日志文件的总长度就相当于，为同一个事务里在 `COMMIT` 命令之前执行的其他命令对 InnoDB 表空间里的数据进行的修改总量设置了一个上限。就目前而言，InnoDB 日志文件的总长度不得超过 4GB。

InnoDB 数据表的事务日志文件只在 MySQL 服务器正在运行时才是必不可少的。MySQL 服务器在正常关机的过程中会把这些文件的内容写入表空间，所以在 MySQL 服务器正常关机之后，这些文件就不再有用了。因此，在对 `ibdata` 文件进行备份的时候，完全用不着把 InnoDB 日志文件也复制下来。

2. InnoDB 日志文件的长度和存放路径

InnoDB 日志文件的总长度对 MySQL/InnoDB 的运行速度有非常大的影响，它们的存放路径、单个文件的长度和文件总个数由配置选项 `innodb_log_group_home`、`innodb_log_files_size` 和

`innodb_log_files_in_group` 控制。在默认的情况下，MySQL 服务器会在它的数据根目录下创建两个长度都是 5MB 的 InnoDB 日志文件。

InnoDB 在线文档推荐的做法是把 InnoDB 日志文件的总长度设置为 InnoDB 事务缓冲区的长度或稍微再大一点儿（后者由配置选项 `innodb_buffer_pool_size` 控制，默认设置是 8MB）。如果这些日志文件的总长度小于这个缓冲区的长度，InnoDB 数据表驱动程序在处理大事务的时候，就有可能不得不从某个位置开始把一些尚未得到确认的中间结果从这个缓冲区里写入硬盘上的临时文件；这里的“某个位置”术语称为检查点（checkpoint）。

如果想改变 InnoDB 日志文件的存放路径、单个文件的长度和文件总个数，必须先让 MySQL 服务器停止运行。然后，在关机过程没有发生任何错误的前提下，删除现有的 InnoDB 日志文件（`ib_logfile0`、`ib_logfile1`、...、`ib_logfileN`）并修改 `/etc/my.cnf` 或 `my.ini` 文件。MySQL 服务器将在自己下次启动时把新的 InnoDB 日志文件创建出来。

注解 如果 InnoDB 数据表的响应速度对读者来说至关重要，就应该把 InnoDB 日志文件与表空间文件安排在不同的硬盘上。请注意，如果 MySQL 服务器在启动时发现 InnoDB 日志文件与 `innodb_log_xxx` 选项的设置不相符，启动过程将半途而废。可以在 MySQL 的出错日志（这个日志的默认文件名是 `hostname.err`）里找到有关的出错消息。创建新 InnoDB 日志文件的操作（即使没有发生任何错误）也将被记入 `hostname.err` 文件。

3. InnoDB 日志文件的同步

配置选项 `innodb_flush_log_at_trx_commit` 和 `innodb_flush_method` 分别控制着何时以及如何对 InnoDB 日志文件进行同步。`innodb_flush_log_at_trx_commit` 选项有 3 种可取值，可以根据自己对速度和安全的不同要求加以选择。

- 把这个选项设置为 0 的意思是把日志数据每隔一秒写入一次当前日志文件并对该文件进行同步。（这里的“写入”意思是把数据传递给操作系统的 I/O 函数。“同步”的意思是把数据物理地址写到硬盘上。）不过，一秒钟对计算机来说是一个相当长的时间，万一在某个事务执行了 `COMMIT` 命令之后、但在 InnoDB 日志文件得到同步之前系统发生了崩溃，那个事务就再也找不回来了——修复故障后重新启动系统也无法把它重新构造出来。换句话说，`innodb_flush_log_at_trx_commit=0` 不能确保 ACID 原则中的“D”。
- 把这个选项设置为 1（默认设置）的意思是在每执行完一条 `COMMIT` 命令就写一次日志并进行同步。这是最保险的办法，但如果执行的净是些小事务的话，每秒可以执行多少个事务还要受到来自硬盘的限制。（每分钟 7200 转的硬盘每秒转 120 圈，这意味着每秒最多能够执行 120 笔事务。但不要忘记这只是一个理论数字，实际情况根本做不到这一点。）
- 把这个选项设置为 2 是一个折衷办法，它的含义是每执行完一条 `COMMIT` 命令写一次日志，每隔一秒进行一次同步。如果 MySQL 服务器发生崩溃，刚刚完成的事务不会丢失（因为由操作系统负责的同步操作可以发生在崩溃之后）。不过，万一操作系统发生崩溃（供电中断等），刚刚完成的事务就会丢失，就像把这个选项设置为 0 时的情况那样。

`innodb_flush_method` 选项负责在由操作系统提供的 `fsync()` 函数（默认设置）和 `O_SYNC()` 函数（设置为 `O_DSYNC`）当中选择一个来同步 InnoDB 日志文件。对大部分 UNIX 版本而言，`O_SYNC()` 函数比较快一些。

注解 即使在 MySQL 里使用了最安全的设置，数据的安全性也要取决于操作系统是如何具体完成对 InnoDB 日志文件的同步操作的。如果 InnoDB 日志文件因为电源故障而受损或同步操作只成功了一半，MySQL 服务器在下次启动时就有可能无法重新构造出受损或丢失的数据。这种事情曾在 2005 年 2 月在 [wikimedia.com](http://meta.wikimedia.org/wiki/February_2005_server_crash) 公司发生过。以下网址处的文章列举了很多在启用了所有的安全措施、日志功能等之后仍未能避免数据库文件遭到破坏的案例（也有成功地避免了损失的案例）：http://meta.wikimedia.org/wiki/February_2005_server_crash 和 <http://lists.mysql.com/mysql/180326>。

4. 归档日志

InnoDB 事务日志文件只适用于对数据进行内部管理，不适合用做备份。如果需要一份从某个特定时间（比如最近一次全面备份）开始的数据修改情况清单，就必须启用 MySQL 的二进制日志功能，该功能与 Innodb 数据表驱动程序完全无关。（请参见本章前面内容里的有关讨论。）

InnoDB 数据表也支持类似的日志功能。InnoDB 在线文档把这种日志功能称为归档日志（archive logging）。不过，归档日志功能只在 InnoDB 数据表驱动程序没有与 MySQL 服务器搭配使用时才有意义。如果因为某种理由必须使用归档日志，可以通过 `innodb_log_archive` 选项来启用它。

5. 速度优化技巧

以下是一些进行速度优化的技巧。这里提供的信息仅适用于使用 InnoDB 数据表处理大量数据的场合。

提示 请参阅 InnoDB 在线文档中的“Performance Tuning tips”（性能优化技巧）一节：http://dev.mysql.com/doc/mysql/en/innodb_tuning.html。在进行速度优化的时候，InnoDB 状态信息可以提供很多有用的信息，这些信息可以通过 `SHOW INNODB STATUS` 命令查看。

6. 与 InnoDB 日志有关的缓冲区配置选项

`innodb_buffer_pool_size` 选项大概是对 InnoDB 数据表驱动程序的速度最有影响的选项了。这个选项负责设定应该使用多少 RAM 内存来临时存放 InnoDB 数据表及其索引。RAM 内存里的可用数据越多，`SELECT` 命令需要访问硬盘的次数就越少。在默认的情况下，InnoDB 数据表驱动程序只为这个缓冲区分配了 8MB 内存。加大这个数字（InnoDB 在线文档推荐的做法是把一台专用的数据库服务器高达 80% 的主内存用做这个缓冲区）可以显著提高 `SELECT` 查询命令的执行速度。（InnoDB 事务日志文件的总长度应该和这个缓冲区一样大。）

根据具体的应用情况，或许还需要用到另外两个选项来设定需要把哪些数据临时存放在 RAM 内存里：`innodb_log_buffer_size` 选项将为 InnoDB 事务日志的缓冲区设置一个长度；`innodb_additional_mem_pool_size` 选项负责设置需要在 RAM 内存里为其他信息（比如，关于数据表的元数据）保留多少空间，如果正在与大量的 InnoDB 数据表打交道，就应该增加这个缓冲区的长度（默认设置是 1MB）。

7. 与 InnoDB 数据表有关的锁定操作

如果正在进行大量的锁定操作（例如，正在导入一个有 100 万条数据记录的数据表、正在把 MyISAM 数据表转换为 InnoDB 格式等），还可以利用下面几个技巧加快操作过程：

- 使用 `SET unique_checks = 0` 命令，意思是不检查 `UNIQUE` 数据列或主索引数据列里的数据是不是真的独一无二。注意，只有在能够绝对肯定有关数据不会出现重复的前提下才可以使用这一设置。

- 使用 `SET foreign_key_checks = 0` 命令，意思是不检查一致性条件。注意，只有在能够绝对肯定有关数据的一致性完好无损的前提下才允许使用能使应该种设置（比如，正在恢复一个备份）。
- 把同一个数据表上的所有 `INSERT` 命令放在同一个事务里来执行。一般来说，导入操作往往由大量的 `INSERT` 命令构成。在默认的情况下（自动提交），每条 `INSERT` 命令都是一个小事务。这时候，只须执行 `SET AUTOCOMMIT=0` 命令，就可以把所有的 `INSERT` 命令纳入同一个事务去执行。千万不要忘记用 `COMMIT` 命令确认这个事务。这个技巧只在事务日志文件足够大的时候才能奏效。注意，对于大型的事务，`ROLLBACK` 操作往往需要耗用相当长的时间，甚至可能需要花费好几个小时。还要注意的是，`CREATE TABLE` 命令有着 `COMMIT` 命令的效果，所以不可能在同一个事务里导入多个数据表。

8. InnoDB 日志选项

在前面的讨论中，提到了几个 InnoDB 日志选项的用法。下面是一个简单的汇总：

- `innodb_flush_log_at_trx_commit=2`。如果希望在每一秒钟执行尽可能多的（小）事务，同时又对万一发生崩溃时可能蒙受的数据损失有心理准备，这个设置就很合适。
- 只要条件允许，就应该把 InnoDB 日志文件与表空间文件分别存放在不同的硬盘上。
- 因不同的操作系统而异，`innodb_flush_method=O_DSYNC` 可以加快 InnoDB 日志功能。

14.9 MySQL 服务器的优化

这里所说的优化是指通过调整 MySQL 服务器的配置参数得到一个最佳配置方案的过程，这种优化可以让 MySQL 服务器最大限度地发挥出服务器主机硬件的潜力和以最高的效率来执行 SQL 命令。

MySQL 服务器的默认配置已经足以满足绝大多数应用的需要，但如果 MySQL 数据库体积庞大（达到了 GB 的规模）、负荷沉重（每秒需要处理许多查询）或者使用了一台专用的计算机来充当数据库服务器的话，就应该根据实际应用情况对它做进一步的优化。

本节内容只是对这个话题的一个入门级介绍，主要围绕如何正确地对各有关缓冲区进行配置以及如何正确地使用 MySQL 的查询缓存区等问题展开讨论。对一个完整的数据库应用解决方案而言，调整服务器的配置参数仅仅是系统优化工作的一小部分内容。要想把与系统优化这个主题有关的方方面面说清楚，恐怕还得有类似这么厚的一本书才行，而书中至少要回答以下几个问题：

- 如何优化数据库的设计方案才能保证那些使用最频繁的命令能够以最高的效率得到执行？如何创建一个效果最佳的索引？毫无疑问，良好的数据库设计方案是速度优化工作最重要的基础之一，但它同时也是人们最容易忽略的事情。如果数据库的设计方案很差劲，再怎么调整服务器的配置参数也不会有好的效果。（请参见第 8 章。）
- 在给定的预算范围内，什么样的硬件最适合这个任务。
- 什么样的操作系统最适合这个任务（如果能有选择的话）。
- 哪种数据表格式最适合这个任务。
- 如果 `SELECT` 查询命令的密度和/或规模很大，能不能把它们分散到多台计算机上去？怎样才能以最佳方式做到这一点。

提示 以上问题只是大型数据库应用项目的优化工作需要解决的一小部分问题，但遗憾的是本书没有足够的篇幅来讨论它们。更详细的信息可以在以下 MySQL 文档处找到：<http://dev.mysql.com/doc/mysql/en/mysql-optimization.html>。

在MySQL领域的权威人士Jeremy Zawodny的著作、网页和新闻中有大量的相关信息。这些信息可以通过Google搜索到，或查看以下网址：<http://jeremy.zawodny.com/mysql/>。

下面的内容是相当有价值的，它是以OpenOffice/Star office格式列出的：peter zaitsev关于MySQL/InnoDB相关特性的一篇演说。网址如下：<http://dev.mysql.com/tech-resources/presentations>和<http://dev.mysql.com/Downloads/Presentations/OSCON-2004.sxi>。

14.9.1 优化内存管理

MySQL 在启动时会保留一部分主内存作为各种用途的缓冲区，比如数据记录的缓存区、用来对数据进行排序的缓存区等。这些缓冲区的长度由配置文件里的各有关选项控制，这些长度值在服务器运行时一般不允许改变。因为这个缘故，有时虽然计算机还有许多内存可供 MySQL 使用，但 MySQL 却不能把它们利用起来。

这些参数都需要通过 MySQL 配置文件中的[mysqld]选项组进行设置。内存长度的计量单位可以是 KB（千字节）、MB（兆字节）和 GB（千兆字节）。下面是设置这些选项的基本语法：

```
# in /etc/my.cnf or my.ini (Windows)
[mysqld]
key_buffer_size = 32M
```

下面是一些比较重要的配置参数（这类参数还有很多）。很遗憾，我们无法告诉大家应该把哪些参数修改成什么样的值，这在很大程度上要取决于具体的应用情况。不过，如果需要调整，key_buffer_size 和 table_cache 参数几乎总是首当其冲。

- **key_buffer_size**（默认设置是 8MB）。索引缓冲区的长度。这个值越大，对数据表里有索引的数据列进行访问时的速度就越快。在专用的数据库服务器上，可以根据具体情况把这个选项设置为可用 RAM 内存的四分之一。
- **table_cache**（默认设置是 64B）。可以同时打开的数据表的个数。打开和关闭数据表需要花费时间，所以这个值越大，能够同时访问的数据表就越多。可是，打开的数据表需要占用内存，而且这个数字还会受到操作系统的限制。MySQL 服务器当前实际打开的数据表的个数可以用 SHOW STATUS 命令来查看（对应的 MySQL 系统变量是 open_tables）。
- **sort_buffer**（默认设置是 2MB）。排序缓冲区的长度。如果没有索引可用，带 ORDER BY 或 GROUP BY 子句的 SELECT 命令将使用这个缓冲区来对数据进行排序。如果这个缓冲区太小，就需要用到一个临时文件，那就慢多了。这个选项 2MB 的默认设置足以满足大部分应用项目的需要。
- **read_buffer_size**（以前叫做 record_buffer；默认设置是 128KB）。需要为每个线程保留多少内存供它从数据表里读取连续排列的数据时使用。这个参数不必很大，因为每一条新的 MySQL 连接都需要有一个这样的缓冲区（也就是说，每个 MySQL 线程都会有一个这样的缓冲区，而不是整个服务器共用一个）。如果需要在某次会话里加大这个参数，用 SET SESSION read_buffer_size = n 命令临时修改一下即可。
- **read_rnd_buffer_size**（默认设置是 256KB）。与 read_buffer_size 选项作用类似，只是这次针对的是按某种特定顺序（比如按 ORDER BY 子句）读取数据记录的情况。把这个参数设置得大一些有助于避免在硬盘上进行搜索操作：如果数据表比较大，搜索硬盘将使响应时间明显变长。与 read_buffer_size 选项的情况类似，这个选项也可以在必要时用 SET SESSION 命令临时改变。

- **bulk_insert_buffer_size** (默认设置是 8MB)。一次插入多条记录的 *INSERT* 命令 (比如 *INSERT ... SELECT ...*) 使用的缓冲区的长度。这个选项也可以在必要时用 *SET SESSION* 命令临时改变。
- **join_buffer_size** (默认设置是 128KB)。为那些没有索引可用的 *JOIN* 操作分配的缓冲区的长度。(经常参与关联查询的数据表绝对应该为被关联的字段创建一个索引——这要比单纯加大这个选项的设置值更有助于提高速度。)
- **tmp_table_size** (默认设置是 32MB)。*HEAP* 类型的临时数据表的最大长度。实际长度超过了这个上限值的 *HEAP* 临时数据表将被转换为 *MyISAM* 类型并被存入一个临时文件。
- **max_connections** (默认设置是 100B)。同时打开的数据库连接的最大个数。这个参数不必很大, 因为每个连接都需要占用一些内存和一个文件描述符。不过, 这个数字越大, 就可以同时打开更多的永久性连接, 而一条永久性连接的好处是可以减少刚关闭一条连接就不得不再次打开一条同样的连接的现象。*(SHOW STATUS* 命令的返回结果里有一个 *max_used_connections* 变量, 这个变量的值是 MySQL 在此前某一时刻同时打开的数据库连接的最大个数。)

提示 可以用 *SHOW VARIABLES LIKE '%size%'* 命令查看到许多重要选项的当前设置值。

如果使用了 InnoDB 数据表, 千万不要忘记对有关的 InnoDB 选项做出必要的修改; 参见本章前面的有关内容。

用 *--help* 选项启动 *mysqld* 程序可以获得一份 MySQL 服务器的可用参数清单。MySQL 在线文档对此做了更详细的描述: <http://dev.mysql.com/doc/mysql/en/show-variables.html>。

14.9.2 查询缓存区

查询缓存区 (query cache) 是 MySQL 4.0 版本引入的一项新功能, 其基本思路是: 把 SQL 查询命令的返回结果存放在内存里, 当某位用户又执行了一条完全一样的查询命令时, MySQL 将把缓存在内存里的查询结果发送给那位用户, 而不是再对各有关数据表进行一次查询。

查询缓存区的出发点非常好, 但可惜的是它并不能解决所有的速度优化问题, 有时反而会增加系统的负担——只要底层的数据表被改变, 受其影响的所有查询都必须立刻从查询缓存区里删除掉。因此:

- 查询缓存区只适用于数据修改操作相对非常少 (也就是 *SELECT* 命令远远多于 *UPDATE*、*INSERT* 和 *DELETE* 命令) 并且同样的查询操作经常重复出现的场合 (基于 Web 的数据库应用往往属于这种情况)。
- *SELECT* 命令必须精确地相同 (包括空格和字母大小写) 才能让查询缓存区知道它们是重复的。
- *SELECT* 命令不得包含用户定义变量, 也不能使用某些特定的函数如 *RAND()*、*NOW()*、*CURTIME()*、*CURDATE()*、*LAST_INSERT_ID()*、*HOST()* 等。

如果这些条件得不到满足, 查询缓存区里的 *SELECT* 查询命令及其查询结果反而会给系统带来负面影响——它们的管理开销会让系统变得更慢。

1. 启用查询缓存区

在默认的情况下, 查询缓存区是被禁用的 (因为默认设置 *query_cache_size=0* 的缘故)。如果想启用查询缓存区, 必须在 MySQL 配置文件里做出如下所示的修改:

```
# in /etc/my.cnf or my.ini (Windows)
[mysqld]
query_cache_size = 32M
query_cache_type = 1      # 0=Off, 1=On, 2=Demand
query_cache_limit = 50K
```

上面这些指令的效果是：为查询缓存区保留 32MB 的 RAM 空间并启用它；允许进入查询缓存区的 *SELECT* 查询结果必须小于 50KB（这是为了避免少数大数据量的查询结果把其他查询“排挤”出查询缓存区）。

有了如上所示的配置，重新启动 MySQL 服务器之后，查询缓存区将自动被激活。MySQL 应用程序不需要做任何调整（但那些重复雷同的查询命令现在的响应时间应该比原先要短一些）。

2. 查询缓存区的demand模式

查询缓存区还可以工作在 *demand* 模式下。此时，只有包含着 *SQL_CACHE* 关键字的 *SELECT* 查询命令（比如：*SELECT SQL_CACHE * FROM authors*）才会进入查询缓存区。如果想控制哪些命令可以进入查询缓存区，这个模式将很有用。

3. 不把SQL查询结果放入查询缓存区

在使用了 *query_cache_type=1* 选项的前提下，如果不想让某条 *SELECT* 命令使用已被启用的查询缓存区，在这条命令里加上一个 *SQL_NO_CACHE* 关键字即可。这个关键字可以避免那些只执行一次或短期内不会重复执行的命令毫无必要地消耗查询缓存区的空间。

4. 启用/禁用查询缓冲区

可以为某个特定的连接随时改变其查询缓存区的当前工作模式：只须执行相应的 *SET query_cache_type = 0/1/2/OFF/ON/DEMAND* 命令即可。

5. 查看查询缓冲区的工作状态

如果想知道查询缓存区是否已被启用、它的长度是否适当等，可以通过 *SHOW STATUS LIKE '%qcache%'* 命令查看其工作状态。将看到一系列状态变量，它们的含义如表 14-3 所示。

表 14-3 查询缓冲区的状态变量

状态变量	含 义
<i>Qcache_queries_in_cache</i>	查询缓存区里目前缓存着多少条查询命令的结果
<i>Qcache_inserts</i>	查询缓存区此前总共缓存过多少条查询命令的结果。这个值几乎总是大于 <i>Qcache_queries_in_cache</i> ，这是因为缓存在这个缓存区里的查询结果会因为缓存区空间不足或底层的数据表发生变化等原因而发生新旧交替的缘故
<i>Qcache_hits</i>	查询缓存区的命中次数，即直接从这个缓存区做出响应的查询命令的个数
<i>Qcache_lowmem_prunes</i>	因为查询缓存区已满而从其中删除的查询结果的数量（如果这个值很大，就说明查询缓存区的长度被设置得小了）
<i>Qcache_not_cached</i>	被拒绝纳入查询缓存区的查询命令（因为它们包含着 <i>RAND()</i> 、 <i>NOW()</i> 等函数或 <i>SQL_NO_CACHE</i> 关键字等）的数量
<i>Qcache_free_memory</i>	查询缓存区现在还有多少可用内存

可以用 *FLUSH QUERY CACHE* 命令来整理查询缓存区里的碎片（根据 MySQL 在线文档的说法，这可以改善内存使用情况，但不会清除查询缓存区里的现有内容）。用来清除查询缓存区的命令是 *RESET QUERY CACHE*，它将删除查询缓存区里的所有内容。

14.10 ISP 数据库管理

到目前为止，本章里的讨论一直是在这样一个假设的前提下进行的：把 MySQL 安装在了属于自己的计算机上并且对 MySQL 服务器有着不受限制的访问权限；换句话说，你本人就是一名数据库管理员。

不过，在 MySQL 的领域里，事情并非总是如此。很多时候，数据库会安装在一台属于某个 ISP 的计算机上，而在那台计算机上几乎没有什么系统管理员级的权限——ISP 自行管理着它的 MySQL 安装。（如果 ISP 有足够的责任心，它就会自动地为数据库定期制作一个备份，但千万不要过分指望这一点。）但不管怎么说，有些管理性任务还是要由自己去完成：

- 创建新数据库（当然，这需要 ISP 允许这样做）。
- 进行备份。（即使 ISP 会定期进行这项工作，能够对自己的数据进行备份也是件好事。）
- 上传数据库。（比如说，在自己的计算机上开发出了一个新的数据库应用程序，现在需要把一个填满了数据的数据库传输到 ISP 的计算机上去。）

14.10.1 ssh 工具

最理想的情况是 ISP 允许使用 ssh 工具去访问它那里的机器。这样一来，可以直接登录到 ISP 的 MySQL 服务器并执行本章介绍的所有命令。当然，发出的 mysqldump 等命令只能访问自己的数据库，但这已经足够了。如果需要在本地计算机和 ISP 之间移动文件，可以使用 ftp 工具。

不过，并非所有的 ISP 都允许它们的客户使用 ssh 来访问自己，那会增加 ISP 的工作量和信息安防风险。

14.10.2 phpMyAdmin 工具

针对这类管理问题最流行的解决方案是 phpMyAdmin 工具，它是安装在 ISP 计算机里的某个 Web 目录下的一组 PHP 脚本（请参见第 6 章）。从理论上讲，本章介绍的所有管理任务都可以用 phpMyAdmin 工具来完成，但这一点在实际上工作中往往不容易做到。

Web 服务器对 PHP 脚本——phpMyAdmin 工具也不例外——有一个最长执行时间的限制，如果某个脚本的实际执行时间超过了预定的上限，Web 服务器就会自动终止这个脚本。这项限制措施主要是为了防止严重的编程错误（比如死循环等）对整个系统造成影响，但这同时难免会让大型数据库的备份工作很难顺利完成。

14.10.3 实现自定义的 PHP 脚本

phpMyAdmin 工具并不是唯一的选择，完全可以自行编写一些 PHP 脚本去完成各项数据库管理任务；这类脚本需要存放在网站的某个子目录里。比如说，可以用一个内容如下所示的 PHP 文件去备份数据库：

```
<?php system("/usr/bin/mysqldump --host=hostname --user=username " .
"--password=xxx dbname > backup.sql"); ?>
```

别忘了把上面这段示例代码中的 /usr/bin 替换为自己的命令路径（例如：/usr/local/bin）。在某些场合，不给出一个路径名也可以执行有关的命令。此外，如果 Web 服务器与 MySQL 运行在同一台计算机上，还可以把 localhost 用做 hostname 参数的值。

完成准备工作之后，只须用 Web 浏览器去访问相应的页面（也就是执行这个 PHP 脚本）就可以生成 backup.sql 文件，它与 PHP 脚本位于同一个子目录。接下来，可以通过 Web 浏览器或 FTP 把它从 Web 服务器下载到本地计算机上。

注解 ISP那里的Web服务器（以及PHP脚本）通常运行在*nobody*或*apache*账户下，这个账户通常是有权向Web页面/脚本目录写文件的。

这里需要注意的是个人账户是否有权向PHP脚本目录写文件：具体到上面这个例子，如果个人账户没有相应的*write*权限，mysqldump程序将无法生成backup.sql文件。可能需要为此执行一条chmod a+w directory命令——如果无法通过ssh去执行这条命令（比如说，因为ISP没有提供ssh服务）的话，还可以用一个FTP客户程序去设置那个子目录的访问权限。

解决了对数据库进行备份并下载备份结果文件的问题之后，再来看看怎样才能把数据上传并加载到数据库里：先通过 FTP 把在本地计算机上生成的 upload.sql 文件传输到在 Web 服务器上的 PHP 脚本目录，然后用 Web 浏览器执行一个内容如下所示的 PHP 脚本（需要提前把这个脚本上传到在 Web 服务器上的脚本目录）：

```
<?php system("/usr/bin/mysql --host=hostname --user=username " .  
    "--password=xxx dbname < upload.sql"); ?>
```

注意，把数据加载到数据库里去的工作只有在那个给定的数据库已经存在的情况下才能成功。如果不是这样，必须先用一条 mysqladmin 命令（这条命令也需要通过在 PHP 脚本里调用 *system()*函数的办法来执行）把那个数据库创建出来。

如果 upload.sql 文件是用 mysqldump 程序生成的，还需要注意这样一个细节问题：包含在 upload.sql 文件里的数据表在 ISP 那里的数据库中是否已经存在——如有必要，请在 upload.sql 文件里插入一些 *DROP TABLE IF EXISTS* 命令。

与 phpMyAdmin 工具相比，定制脚本的好处是通过因特网传输数据库文件的时间不计算在脚本的执行时间内——需要加载到数据库里去的数据已经在脚本开始执行前上传到了 Web 服务器，把数据库文件下载到本地计算机的工作是在脚本执行结束后才进行的。虽然这不能改变 Web 服务器为 PHP 脚本设定的执行时间上限，但留给 PHP 脚本的实际执行时间显然要长不少，而这意味着定制脚本能够处理和传输一些更大（与使用 phpMyAdmin 工具的办法相比）的数据库。当然，PHP 脚本的执行时间上限始终是必须认真对待的一个问题。

顺便说一句，不要忘记用一个 PHP 脚本在 Web 页面/脚本目录里创建一个 .htaccess 文件，这个文件可以保护 Web 页面/脚本目录。对 .htaccess 文件的详细介绍见第 2 章。

14.10.4 自定义：Perl 脚本

用 PHP 脚本可以完成的工作用 Perl 脚本也可以完成，而且往往能完成得更好。Perl 脚本的好处是它们通常属于 CGI 脚本的范畴，而 CGI 脚本在执行时间方面一般没有上限（具体情况取决于 ISP）。

Perl 语言也提供了一个 *system()*函数来执行外部程序命令。在下面的 Perl 脚本里，增加了一些用来处理和显示 *system()*函数返回结果的代码，mysqldump 和 mysql 命令则与 14.10.3 节里的 PHP 脚本完全一样。前期准备工作也完全一样：

```
#!/usr/bin/perl -w
use CGI qw(:standard);
use CGI::Carp qw(fatalsToBrowser);
print header(), start_html("Backup");
if(system("/usr/bin/mysqldump --host=hostname --user=username " .
          "--password=xxx dbname > backup.sql")) {
    print p(), "failed", end_html();
} else {
    print p(), "done", end_html(); }
```

Part 4

第四部分

程序设计

本部分内容

- 第 15 章 PHP
- 第 16 章 Perl
- 第 17 章 Java (JDBC 和 Connector/J)
- 第 18 章 C 语言
- 第 19 章 Visual Basic 6/VBA
- 第 20 章 Visual Basic .NET 和 C#

第 15 章

PHP

PHP 是英文 PHP: Hypertext Preprocessor 的字头缩写 (给软件起这种回文形式的名字是 UNIX 程序员的习惯)。PHP 是一种用来编写 HTML 页面的脚本程序设计语言, 嵌在 HTML 文件里的 PHP 代码是在 Web 服务器上执行的。(微软公司的 ASP 技术采用的也是这一思路。)

在需要选用一种程序设计语言来编写 MySQL 数据库系统的 Web 应用程序时, 人们几乎总是会选择 PHP。“PHP + MySQL”组合可以说是建设动态网站的绝佳搭档; 它们的共同特点是: 安装简便、速度快、性价比高(都是自由软件)。这也正是本章篇幅为什么会比后面介绍另外几种程序设计语言的几章要长很多的原因之一。

从 PHP 5 开始, PHP 向程序员提供了两种 MySQL 应用程序编程接口: 一种是从 PHP 早期版本一直就有过的 *mysql* 功能模块, 这个接口是 PHP 程序员都很熟悉的; 另一种是从 PHP 5 才开始有的 *mysqli* 接口, 这个接口提供了更丰富细致的编程功能, 程序员可以通过它们去访问 MySQL 的各种新增功能(比如说, *mysqli* 接口提供的 SQL 命令预处理功能可以让带参数的 SQL 命令获得更高的执行效率)。不过, 必须安装 PHP 5 和 MySQL 4.1 或更高版本才能使用 *mysqli* 接口。

本章将先简要地介绍一下 *mysqli* 接口最重要的一些函数、类、方法和属性, 然后再通过一些示例介绍一些基本的编程技巧, 如查询结果的分页显示、层次化数据的处理、Unicode 字符串的处理等。

假设: 本章需要假设 PHP 和 MySQL 都已经正确安装和配置(参见第 2 章)。本章中的绝大多数示例都是在 *mysqli* 接口和本书第 8 章介绍的示例数据库 *mylibrary* 的基础上完成的。为了让这些示例正常工作, 必须让 *mysql-intro.php*、*mysqli-intro.php* 和 *password.php* 这 3 个文件里的用户名和密码字符串保持一致。

15.1 *mysql* 功能模块

mysql 功能模块不是 PHP 的一个集成组件。要想使用这个功能扩展模块, PHP 的 Linux 版本必须在编译时加上一个 `--with-mysql` 选项。PHP 的 Windows 版本通过一个 DLL 文件提供了相应的扩展。不管使用的是哪一种操作系统, 都必须在 *php.ini* 文件里启用这个扩展以确保 PHP 能够找到所有必要的 DLL(参见第 2 章)。

可以用一个最简单的 PHP 脚本来检查一下正在使用的 PHP 版本是否支持 *mysql* 功能模块, 这个 PHP 脚本只包含一条语句`<?php phpinfo() ?>`。这个脚本的执行结果是一份非常长的表格, 里面列出了可供使用的所有 PHP 功能。如果使用的 PHP 版本支持 *mysql* 功能模块, 输出的 *mysql* 部分应该如图 15-1 所示。

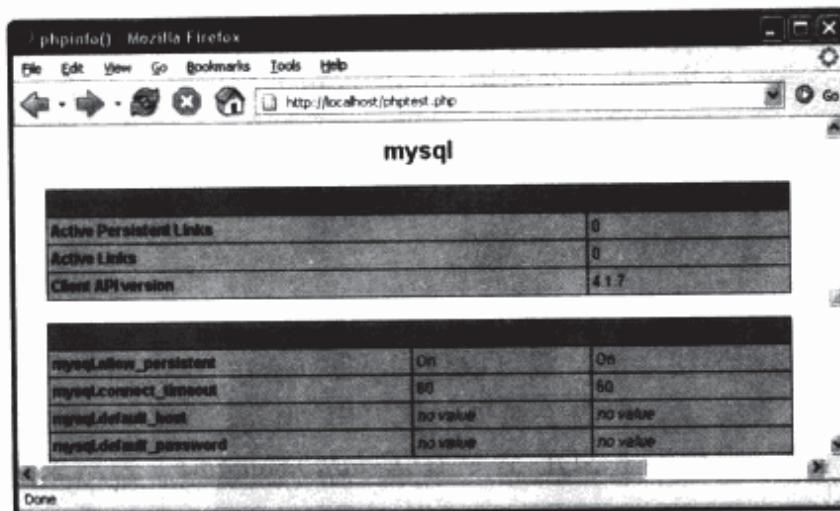


图 15-1 检查 PHP 是否支持 mysql 功能模块

15.1.1 连接 MySQL 服务器

通过 *mysql* 功能模块去连接 MySQL 服务器的办法是调用 *mysql_connect()* 函数，它需要提供 3 项信息：MySQL 服务器的主机名、MySQL 用户名和密码。如果 MySQL 服务器与 PHP 脚本运行在同一台计算机上，可以使用 *localhost* 作为它的主机名。

```
$conn = mysql_connect("localhost", "username", "xxx");
```

在基于 RHEL 和 Fedora 的系统上，SELinux 往往会被配置成不允许 Apache 和 PHP 访问 MySQL 套接字文件。这样，在调用 *mysql_connect()* 函数的时候就不能把 *localhost* 用作 MySQL 服务器的主机名，而是必须给出它的真实名字，这将使得 PHP 脚本与 MySQL 服务器之间的通信使用 TCP/IP 协议，而不是通过套接字文件来进行。

如果连接成功，这个函数将返回一个标识号码。如果与某个 MySQL 服务器建立了一条以上的连接，在以后的操作中就必须使用它们的标识号码来区分它们。（如果只与某个 MySQL 服务器建立了一条连接，这条连接就会成为与 MySQL 服务器之间的默认连接，也就无需在调用各种 *mysql_xxx()* 函数时给出这个标识号码了。）

mysql_connect() 函数的第 4 个参数是可选的。在使用同样的连接数据多次调用 *mysql_connect()* 函数去连接同一个 MySQL 服务器时，如果把这个参数设置为 *false*（这也是它的默认值），*mysql_connect()* 函数将返回现有连接的一个链接；如果把这个参数设置为 *true*，该函数将建立一个新连接。换句话说，如果需要与同一 MySQL 服务器建立多个不同的连接，在调用这个函数的时候就必须使用 *mysql_connect(\$host, \$name, \$pw, true)* 格式。

可以将客户标志传递给第 5 个可选参数。如果数据转换被压缩，则参数内容可能是 *MySQL_CLIENT_COMPRESS*。

如果连接没有成功，*mysql_connect()* 函数将返回 *FALSE*（给变量 *\$conn*），并向 Web 服务器发送一条出错消息，这将使 PHP 脚本所生成的结果 HTML 文档里也出现一条出错消息。如果不想让来自网络的最终用户在他们的 Web 浏览器里看到这样的出错消息，就必须在调用 *mysql_connect()* 函数的时候在它的前面加上一个@字符。（出错消息往往包含着一些不应该让最终用户看到的技术细节，所以很多程序员会在调用 PHP 函数时给它们加上一个@字符作为前缀，来防止由系统返回的出错信息直接显示在最终用户的 Web 浏览器画面里。）

不过,为了让最终用户知道发生了问题,PHP 代码应该向最终用户返回一条简明易懂的出错消息。具体到连接 MySQL 服务器这件事,应该在调用 *mysql_connect()* 函数之后加上一些如下所示的出错检查和处理代码:

```
$conn = @mysql_connect("localhost", "username", "xxx");
if($conn == FALSE) {
    echo "<p>error message ...</p>\n";
    exit();
}
```

在建立起与 MySQL 服务器的连接之后,就可以使用各种 *mysql_xxx()* 函数和 *mysql* 功能模块提供的其他一些手段去执行 SQL 命令了。但为了避免每次调用 *mysql_xxx()* 函数都指定目标数据库,最好先用 *mysql_select_db()* 函数(它相当于 SQL 命令 *USE databasename*)为后续操作选定一个默认数据库,例如:

```
mysql_select_db("mylibrary");
```

与当前连接有关的更详细的信息可以通过以下函数查知:*mysql_get_client_info()*,客户端 API 函数库的版本信息;*mysql_get_host_info()*,与 MySQL 服务器的连接类型;*mysql_get_proto_info()*,通信协议的版本信息;*mysql_get_server_info()*,MySQL 服务器的版本信息;*mysql_client_encoding()*,客户端使用的默认字符集;*mysql_stat()*,MySQL 服务器的当前工作状态;*mysql_thread_id()*,MySQL 服务器进程的线程编号。

完成数据库访问工作之后,可以用 *mysql_close()* 函数断开与 MySQL 服务器的连接。

15.1.2 执行 SQL 命令

为了执行 SQL 命令,需要把它们作为一个字符串传递给 *mysql_query()* 函数。如果想访问的不是当前数据库,就需要调用 *mysql_db_query()* 函数来添加 SQL 命令并明确地给出那个数据库的名字。这两个函数的最后一个参数(连接的 ID 号码,即 *mysql_connect()* 函数的返回值)都是可选的,只有在与同一个 MySQL 服务器建立了多条连接的时候才必须给出这个参数。

```
$result = mysql_query("SELECT COUNT(*) FROM titles");
$result = mysql_db_query("mylibrary", "SELECT COUNT(*) FROM titles");
```

mysql_query() 函数可以用来执行任何一种 SQL 命令,比如 *SELECT* 命令(查询)、*INSERT* 命令(插入新记录)、*UPDATE* 命令(修改现有记录)、*DELETE* 命令(删除现有记录)、*CREATE TABLE* 命令(创建新数据表)、*ALTER TABLE* 命令(改变数据表结构)等。

注意 通过调用 *mysql_query()* 函数而执行 SQL 命令不得以分号(;)结束;否则会导致产生 SQL 语法错误。同一次 *mysql_query()* 函数调用无法执行多条 SQL 命令。如果想执行一条以上的命令,就必须为它们分别调用一次 *mysql_query()* 函数。

在调用 *mysql_query()* 函数执行完 *INSERT*、*UPDATE* 和 *DELETE* 命令(以及会修改数据记录的所有其他命令)之后,可以调用 *mysql_affected_rows()* 函数去查知它们到底修改了多少条数据记录。如果在执行完 *CREATE TABLE ... SELECT ...* 命令后调用这个函数,就可以查知新创建的数据表到底插入了多少条记录。

此外,在执行完 *INSERT* 命令之后,还可以调用 *mysql_insert_id()* 函数去查知刚才插入的最后一条

新记录的 *AUTO_INCREMENT* 值到底是多少。

```
$n = mysql_affected_rows(); // number of changed records
$new_id = mysql_insert_id(); // ID number of the last AUTO_INCREMENT record
```

从 PHP 4.3 开始, *mysql_info()* 函数还可以提供关于最后一条 *ALTER TABLE*、*CREATE TABLE*、*INSERT INTO*、*LOAD DATA* 和 *UPDATE* 命令更多细节的信息, 如关于重复记录和出错警告的信息等。比如说, 如果刚执行了一条 *CREATE TABLE backup SELECT * FROM table* 命令, *mysql_info()* 函数将返回一个 *Records: nnn Duplicates: nnn Warnings: nnn* 形式的字符串, 告诉用户该命令总共处理了多少条记录、其中有多少是内容重复的、它引发了多少次警告。

如果 SQL 命令执行成功, *mysql_query()* 函数将返回一个非零值。如果一条查询命令, *mysql_query()* 函数的返回值将是某个 PHP 资源的引用指针 (一个 *Resource id #2* 格式的字符串)。这个返回值可以用在各种其他 *mysql_xxx()* 函数 (比如 *mysql_fetch_row()*) 里, 对结果数据表的各个字段进行处理。在本章的示例中, 通常会把这个返回值保存在 *\$result* 变量里。(将在本章后续各节详细讨论如何处理和使用 *SELECT* 查询结果。)

另一方面, 如果 SQL 命令没有执行成功, *mysql_query()* 函数将返回 *FALSE* (即数值 0), 并会在 HTML 结果文档里生成一条出错消息; 如果想抑制这样的出错消息, 就需要在调用 *mysql_query()* 函数时在它的前面加上一个@字符。(出错原因可以利用 *mysql_errno()* 和 *mysql_error()* 函数来确定。马上就会讨论到出错处理问题。)

15.1.3 处理 *SELECT* 查询结果

在调用 *mysql_query()* 函数执行完一条 *SELECT* 查询命令之后, 可以用下面的命令获得结果数据表的数据行个数 (*\$row*) 和数据列个数 (*\$col*) :

```
$result = mysql_query("SELECT * FROM titles");
$rows = mysql_num_rows($result);
$cols = mysql_num_fields($result);
```

这里需要考虑两种特殊情况:

- 如果查询命令返回且仅返回了一个值 (比如像 *SELECT COUNT(*) FROM table* 这样的命令), 结果数据表将只有一行、一列。
- 如果查询命令根本没有返回任何结果 (比如因为原始数据表里没有满足 *WHERE* 条件的记录), 结果数据表的数据行个数将是 0。这种特殊情况只能通过检查 *mysql_num_rows()* 函数返回值是否为零的办法来识别。

只有当 SQL 查询命令本身有语法错误或是因为与 MySQL 的通信出现问题的时候, 才得不到任何结果, 此时 *mysql_query()* 函数的返回值将是 *FALSE*。

如果想访问结果数据表里各个字段, 最简单、但同时也是最慢的办法是使用 *mysql_result()* 函数。这个函数需要给它传递一个行号和一个列号 (行号和列号从 0 开始计数, 所有的 PHP/MySQL 函数都如此) 作为参数, 列号也可以替换为数据列的名字 (或数据列的假名——如果在那条 SQL 查询命令里使用了 *AS alias* 语法的话)。如下所示:

```
$item = mysql_result($result, $row, $col);
```

一个数据行一个数据行地对查询结果进行处理会更有效率, 这需要用到以下 4 个函数:

```
$row = mysql_fetch_row($result);
$row = mysql_fetch_array($result);
$row = mysql_fetch_assoc($result);
$row = mysql_fetch_object($result);
```

- *mysql_fetch_row()* 函数将以一个普通数组的形式返回一条结果记录，它的各个字段需要以 *\$row[\$n]* 的方式进行访问。
- *mysql_fetch_array()* 函数将以一个关联数组的形式返回一条结果记录，它的各个字段需要以 *\$row[\$n]* 或 *\$row[\$colName]* 的方式（如 *\$row[3]* 或 *\$row["publName"]*）进行访问。数据列的名字区分字母的大小写情况。
- *mysql_fetch_assoc()* 函数（仅适用于 PHP 4.0.3 及更高版本）也将以一个关联数组的形式返回一条结果记录，但它的各个字段只能以 *\$row[\$colName]* 的方式进行访问，不允许像 *mysql_fetch_array()* 函数一样使用列号作为数组变量 *\$row* 的下标。
- *mysql_fetch_object()* 函数将以一个对象的形式返回一条结果记录，它的各个字段需要以 *\$row->colName* 的方式进行访问。

这 4 个函数的共同特点是：每次调用将自动返回下一条结果记录（但如果已经到达结果数据表的末尾，则返回 *FALSE*）。如果想改变这个顺序，就必须用 *mysql_data_seek()* 函数明确地改变当前结果记录：

```
mysql_data_seek($result, $rownr);
```

PHP 会把查询结果一直保存到 PHP 脚本执行结束。如果必须提前释放某次查询的结果（例如当在某个脚本里已经进行了大量的查询、不想再浪费内存的时候），可以用 *mysql_free_result()* 函数提前释放它。这个函数在使用了循环来进行 SQL 查询的 PHP 脚本里经常可以看到：

```
mysql_free_result($result);
```

注解¹ PHP 脚本里的数组多为关联数组，这种数据类型不要求数组的下标必须是顺序递增的整数，它们可以是任何一些具有唯一性和可关联性的值。在后面的讨论里，如果没有特别说明，在提到数组的时候都指的是关联数组。

1. 示例：处理 SELECT 查询结果

下面这个例子应该可以让大家看清楚前面介绍那些函数的交错顺序。在成功地与 MySQL 服务器建立连接之后（这里用了一个 *if* 语句来进行检查），对 *mylibrary* 数据库进行了 *SELECT * FROM titles* 查询并利用 *\$row* 变量把查到的图书名称（以及副标题——如果有）显示出来。*htmlspecialchars()* 函数的作用是确保图书名称里的特殊字符（如 <、>、" 等）都按 HTML 标准进行编码。在输出完查询结果之后，删除了查询结果对象并断开了与 MySQL 服务器的连接。

```
// example mysql-intro.php
if($conn = @mysql_connect("localhost", "root", "xxx")) {
    mysql_select_db("mylibrary");
    if($result=mysql_query("SELECT * FROM titles ORDER BY title")) {
        printf("<p>Number of records: %d</p>\n", mysql_num_rows($result));
        printf("<p>Number of columns: %d</p>\n",
            mysql_num_fields($result));
        while($row = mysql_fetch_object($result)) {
            if($row->subtitle)
```

1. 这条“注解”原文里没有，但译者认为有必要在这里说明一下，以免不熟悉 PHP 的读者对此感到困惑。

```

printf("<br />%s -- %s\n",
      htmlspecialchars($row->title), htmlspecialchars($row->subtitle));
else
    printf("<br />%s\n", htmlspecialchars($row->title));
}
mysql_free_result($result);
}
} else {
    printf("<p>Sorry, could not connect to MySQL server! %s</p>\n",
           mysql_error());
}

```

上面这段代码的输出效果如图 15-2 所示。

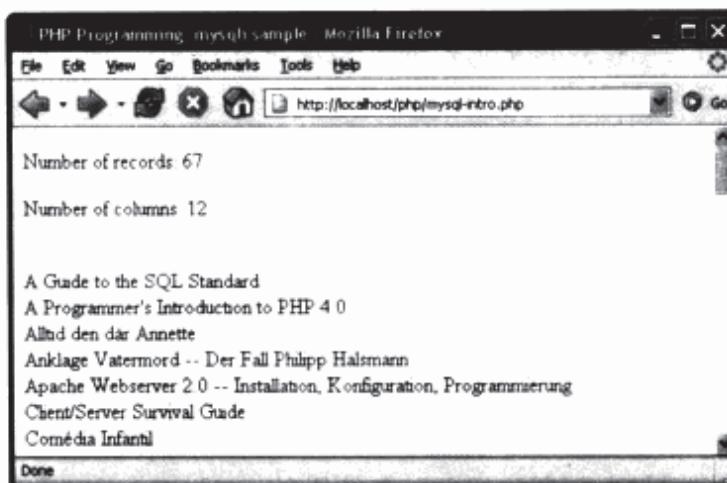


图 15-2 从 *mylibrary* 数据库查询出来的图书清单

2. 数据列的名字和元信息

如果对查询进行处理的目的是为了把它们的查询结果显示出来，那不仅需要获得结果数据本身，还需要获得关于结果数据的元信息，如各有关数据列的名字、数据类型等。

mysql_field_name() 函数可以返回一个给定数据列的名字。*mysql_field_table()* 函数可以告诉用户一个给定数据列来自哪一个数据表(这在处理来自多个数据表的查询结果时非常有用)。*mysql_field_type()* 函数把给定数据列的数据类型返回为一个字符串(比如，“*BLOB*”)。*mysql_field_len()* 函数返回一个给定数据列的最大长度(这在处理 *CHAR* 和 *VARCHAR* 类型的数据时非常有用)。

```

$colname = mysql_field_name($result, $n);
$tblname = mysql_field_table($result, $n);
$typename = mysql_field_type($result, $n);
$collength = mysql_field_len($result, $n);

```

mysql_field_flags() 和 *mysql_fetch_field()* 函数可以提供更详细的信息。*mysql_field_flag()* 函数把一个给定数据列最重要的属性返回为一个字符串(比如，“*not_null primary_key*”)，各个属性之间以空格隔开，所以很容易用 *explode()* 函数把它们分别提取出来。

mysql_fetch_field() 函数把一个给定数据列最重要的属性返回为一个对象，各个属性可以直接以 *cinfo->name*、*cinfo->blob* 的形式使用。

```

$colflags = mysql_field_flags($result, $n);
$colinfo = mysql_fetch_field($result, $n);

```

3. 使用 *mysql_unbuffered_query()* 函数进行数据库查询

还可以用 *mysql_unbuffered_query()* 函数代替 *mysql_query()* 函数去执行 *SELECT* 查询命令。这么做

的效果是：MySQL 服务器将不会把查询结果一次全部传输到 PHP 解释器的内存里，而是会根据具体情况（在 PHP 脚本需要读取下一条结果记录时）分多次传输。这将产生以下后果：

- 数据会留在服务器上直到 PHP 脚本把它们取走，而这意味着 MySQL 服务器上的有关资源将在这段时间内一直处于阻塞状态。（从另一个方面讲，PHP 解释器只须分配一小块内存作为中间存储就够了。）
- 只能用 `mysql_fetch_row()`、`mysql_fetch_array()`、`mysql_fetch_assoc()` 或 `mysql_fetch_array()` 函数一个接一个地读取（只能前进）那些结果记录，无法使用 `mysql_fetch_result()` 和 `mysql_data_seek()` 函数。因为 PHP 解释器的内存里每次只有一条记录，所以已经读取过的记录无法再读第二次。
- 在把这次的 `SELECT` 结果全部读完之前，无法在给出的这条 MySQL 连接上执行一条新的 `SELECT` 命令。如果必须那么做，只能与 MySQL 服务器另外建立一条连接或是明确地用 `mysql_free_result()` 函数释放这条 `SELECT` 命令占用的资源。
- 因为无法使用 `mysql_num_rows()` 函数，只有把这次查找到的记录全部读取完毕之后才能确定它们总共有多少条。
- `mysql_unbuffered_query()` 函数只能用来执行 `SELECT` 查询，不能用来执行 `INSERT` 或 `DELETE` 等没有返回结果的命令。

`mysql_unbuffered_query()` 函数特别适合这样的场合：查找出的结果记录比较多、必须逐条地对结果记录进行处理、可是客户端的内存又不是很充足。不过，因为 PHP 页面的数据量一般都不会很大，PHP 代码的执行时间又受到了限制（见 `php.ini` 文件里的 `max_execution_time` 选项），所以这个函数的适用场合远远少于 `mysql_query()` 函数。

15.1.4 事务

`mysql` 功能模块没有专门用于管理事务的函数。如果想把多条 SQL 命令当做一个事务来执行，就只能采用下面的办法：先执行一个 `mysql_query("BEGIN")` 调用、然后执行一个 `mysql_query ("COMMIT")` 调用用来确认事务或是执行一个 `mysql_query("ROLLBACK")` 调用用来撤销这个事务的全部命令。这里还要再次提醒：MySQL 中目前只有 InnoDB 数据表才支持事务。

15.1.5 出错处理与查找

如果在执行时出现错误，所有的 `mysql_xxx()` 函数都将返回 `FALSE`。此时 `mysql_errno()` 函数（出错代码）和 `mysql_error()` 函数（出错信息）可以返回关于出错原因的信息。

有不少 `mysql_xxx()` 函数会在执行出错时把它们自己的出错消息写入 HTML 结果文档。如果想制止这种行为，就必须在调用有关函数时给它们加上一个@字符作为前缀或是在 `php.ini` 文件里给出 `display_errors = Off` 选项。

```
$result = @mysql_query($sql);
if(!$result) {
    printf("<p>error: %d -- %s</p>\n", mysql_errno(),
        htmlspecialchars(mysql_error()));
}
```

原则上，必须假设肯定会有错误出现，尤其是在执行那些会修改数据记录的 SQL 命令时。最容易引发错误的原因之一是 PHP 脚本与 MySQL 服务器之间的连接出了问题。

在查找错误原因的时候，*mysql_query_test()* 函数往往很有用。有不少程序员在为 MySQL 数据库开发 PHP 脚本的过程中干脆就把它当做 *mysql_query()* 函数的替代品来使用。这个函数会把执行过的 SQL 命令以蓝色文字显示，还会在发生错误时立刻把出错消息显示出来，而不是写入 HTML 结果文档。

```
function mysql_query_test($sql) {
    echo '<p><font color="#0000ff"> SQL:', htmlspecialchars($sql),
        '</font></p>\n';
    $result = mysql_query($sql);
    if($result) return $result;
    echo '<p><font color="#ff0000">Error: ', 
        htmlspecialchars(mysql_error()), "</font></p>\n";
    die();
}
```

提示 如果在试用这本书里的某个示例时遇到了问题该怎么办？有时候，就算是有出错消息，想迅速确定错误根源也不那么容易。

根据笔者的经验，几乎所有的问题都与数据库访问权限有关：PHP 脚本没有从某个数据库里读取数据的权限，所以返回了不正确的数据或是根本没有返回任何数据。导致这类问题的根源有很多，但几乎都与访问权限有关：不正确的密码、忘了提供密码等。要知道，只有从未进行过安全配置的 MySQL 服务器才会允许用户在本地计算机上使用 *name="root"* 和 *password=""* 的组合去连接它。

15.2 *mysqli* 的类、方法和属性

mysqli 接口被认为是 PHP 5 最重大的创新之一，至少在 MySQL 应用程序的开发团队中间是如此。这个接口使程序员能够使用面向对象的编程思想为 MySQL 数据库系统开发应用程序，并帮助程序员编写出更容易阅读和理解的代码。

与 *mysql* 功能模块相比，*mysqli* 接口的第二个优点是它能以非常高的效率执行各种带参数的 SQL 命令（预处理语句）。这种命令只能在 MySQL 4.1 及以后的版本里使用，在需要连续执行多条除参数以外在其他方面完全一样的 SQL 命令的时候它们的执行效率很高。

如果正在使用存储过程，*mysqli* 接口还有一个优点：可以用 *multi_query()* 函数去执行那些会返回不止一个结果（多条 *SELECT* 命令的结果）的存储过程。通过 *mysql* 功能模块去完成这种任务会非常复杂。

mysqli 接口既能以面向对象的方式使用，也能以过程化方式使用，本书将重点介绍它面向对象的用法。如果更喜欢以过程化方式编写程序，那还是使用前面介绍的 *mysql* 功能模块好了。

15.2.1 选择编程接口：*mysql* 还是 *mysqli*

既然有两种接口，就不可避免地会出现这样的问题：哪一种接口更好？我应该使用哪一种接口？

赞成使用 *mysqli* 接口的人会说它是一种更现代的接口，它提供了更多的函数，可以使代码变得更简明易读，在许多场合的运行速度更快。也正是因为这个理由，本章后面的内容将全部用来讨论 *mysqli* 接口。

反对使用 *mysqli* 接口的人会说这个接口只能在 PHP 5 和 MySQL 4.1（或是更新的版本）上使用。如果正在编写的代码必须与 PHP 4.n 或 MySQL 4.0 保持兼容，就只能使用 *mysql* 功能模块。

是否在自己的开发计算机上安装最新版本的软件是个人自由，但许多 ISP 在是否使用最新版软件的方面是相当保守的，对他们而言，软件的稳定性和成熟性要比新功能更重要。因此，*mysql* 功能模

块肯定还会继续存在一段时间。

15.2.2 有效性测试

与 *mysql* 功能模块的情况一样, *mysqli* 接口也不是 PHP 的一个集成组件。要想使用这个功能扩展模块, PHP 的 Linux 版本必须在编译时加上一个--with-mysqli 选项。PHP 的 Windows 版本通过一个 DLL 文件提供了相应的扩展。不管使用的是哪一种操作系统, 都必须在 *php.ini* 文件里启用这个扩展, 以确保 PHP 能够找到所有必要的 DLL (参见第 2 章)。

可以用一个最简单的 PHP 脚本来检查一下正在使用的 PHP 版本是否支持 *mysqli* 接口, 这个 PHP 脚本只包含一条语句<*?php phpinfo(); ?>*。这个脚本的执行结果是一份非常长的表格, 里面列出了可供使用的所有 PHP 功能。如果使用的 PHP 版本支持 *mysqli* 接口, 应该在这份表格里找到如图 15-3 所示的 *mysqli* 输出内容。

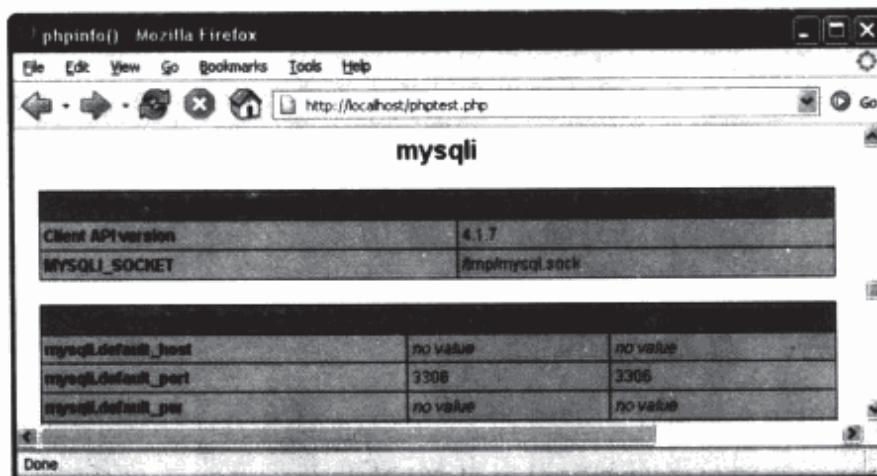


图 15-3 检查 PHP 是否支持 *mysqli* 接口

15.2.3 构成 *mysqli* 接口的类

mysqli 接口包括以下 3 个类:

- *mysqli*。这个类的对象控制着与 MySQL 服务器的连接。为了建立一个连接, 需要使用构造函数。这个类最重要的方法是 *query*, 它的用途是执行 SQL 查询。执行 *SELECT* 查询将获得一个 *mysqli_result* 类的对象作为结果。
- *mysqli_result*。这个类的对象包含 *SELECT* 查询的结果。
- *mysqli_stmt*。可以用这个类的对象去定义和执行参数化的 SQL 命令。*mysqli_stmt* 对象需要使用 *\$mysqli->prepare()* 方法来创建。

15.2.4 连接 MySQL 服务器

通过 *mysqli* 接口连接 MySQL 服务器的办法是调用 *mysqli* 类的构造函数, 需要向它提供 3 项信息: MySQL 服务器的主机名、MySQL 用户名和密码。如果 MySQL 服务器与 PHP 脚本运行在同一台计算机上, 可以使用 *localhost* 作为它的主机名。这个构造函数还有 3 个可选的参数: 要使用的数据库的名字、MySQL 服务器的端口号 (默认设置是 3306 号端口)、一个套接字文件或命名管道。在连接成功后, 可以通过 *select_db()* 方法为当前连接改变它的默认数据库。

在基于 RHEL 和 Fedora 的系统上, SELinux 往往会被配置成不允许 Apache 和 PHP 访问 MySQL

套接字文件。如果真是那样，在调用 *mysqli* 类的构造函数时就不能把 *localhost* 用作 MySQL 服务器的主机名，而是必须给出它的真实名字，这将使得 PHP 脚本与 MySQL 服务器之间的通信使用 TCP/IP 协议而不是通过套接字文件来进行。

如果连接成功，构造函数将返回一个 *mysqli* 对象。注意，必须使用 *mysqli_connect_errno()* 函数测试在建立连接的过程中是否发生过错误。相关的出错消息由 *mysqli_connect_error()* 函数负责返回。

当前数据库连接的更详细的信息可以通过以下方法和属性查知：*client_info*、*character_set_name*、*get_client_info()*、*get_host_info()*、*get_server_info()* 等。

完成数据库访问工作，不再需要连接到数据库之后，应该用 *close()* 方法明确地释放有关的 *mysqli* 对象。

```
// create connection to the database
$mysqli = new mysqli("localhost", "user", "password", "mylibrary");
// test whether connection OK
if(mysqli_connect_errno()) {
    echo "<p>Sorry, no connection! ", mysqli_connect_error(), "</p>\n";
    exit();
}
// use connection
...
// close connection
$mysqli->close();
```

本节下面的例子都假设 \$mysqli 是有 MySQL 连接的对象。

1. 用 *read_connect()* 方法来建立连接

上面介绍的使用 *mysqli* 构造函数来建立连接的办法有一个缺点：无法设置任何 MySQL 特有的连接选项。如有必要（但这种情况很少见），还可以像下面这样去创建一个连接：

```
$mysqli = mysqli_init(); // create mysqli object
$mysqli->options(...); // set additonal options (optional)
$mysqli->ssl_set(...); // set SSL options (optional) .
// create connection to MySQL server
$mysqli->real_connect("localhost", "user", "password", "mylibrary");
// test whether OK
if(mysqli_connect_errno()) {
    echo "<p>Sorry, no connection! ", mysqli_connect_error(), "</p>\n";
    exit();
}
```

2. 可选的连接选项

options() 和 *ssl_set()* 方法都是可选的。*options()* 方法用来设置各种各样的 MySQL 选项，例如设置连接倒计时时间，在连接成功之后立刻执行一条 SQL 命令等：

```
$mysqli->options(MYSQLI_OPT_CONNECT_TIMEOUT, 10);
$mysqli->options(MYSQLI_INIT_COMMAND, "SQL command ...");
```

ssl_set() 方法用来设置 SSL 加密连接的密钥和其他参数：

```
$mysqli->ssl_set("key", "cert", "ca", "capath", "cipher");
```

read_connect() 方法第 5 个~7 个参数是可选的，可以通过它们向这个方法传递以下信息：MySQL 服务器的端口号（一般是 3306 号端口）、供 UNIX/Linux 内部通信用的服务器套接字文件、由几个连接属性构成的操作标志。比如说，如果需要对被传输的数据进行压缩或 SSL 加密，就需要给出 *MYSQL_CLIENT_COMPRESS* 或 *MYSQL_CLIENT_SSL* 操作标志。

```
$mysqli->real_connect("localhost", "user", "password", "mylibrary",
3306, "/var/lib/mysql/mysql.sock", MYSQLI_CLIENT_COMPRESS);
```

15.2.5 执行 SQL 命令

mysqli 接口提供了几种执行 SQL 命令的办法，其中最常用的是 *query()* 方法。对于不会返回数据的 SQL 命令（比如 *INSERT*、*UPDATE*、*DELETE*），*query()* 方法在 SQL 命令执行成功时的返回值将是 *TRUE*。在此基础上，*\$mysqli->affected_rows()* 方法将返回有多少条数据记录发生了变化，*\$mysqli->insert_id()* 方法将返回最后一条 *INSERT* 命令生成的 *AUTO_INCREMENT* 编号值。

```
if($mysqli->query("INSERT ...")) {
    echo "<p>changed records: ", $mysqli->affected_rows, "</p>\n";
    echo "<p>new ID value: ", $mysqli->insert_id, "</p>\n";
}
```

对于 *SELECT* 命令，*query()* 方法将返回一个 *mysqli_result* 对象，*\$result->num_rows* 属性将给出结果数据表里的记录个数，*\$result->field_count* 属性将给出结果数据表里的数据列个数。

```
if($result = $mysqli->query("SELECT ...")) {
    echo "<p>found records: ", $result->num_rows, "</p>\n";
    echo "<p>number of columns: ", $result->field_count, "</p>\n";
    ...
    $result->close();
}
```

如果在执行 SQL 命令时发生错误，*query()* 方法将返回 *FALSE*。此时可以通过 *\$mysqli->errno* 和 *\$mysqli->error* 属性获得关于出错原因的更多信息。

提示 *query()* 方法每次调用只能执行一条 SQL 命令。如果想一次执行多条命令，就必须使用稍后将要介绍的 *multi_query()* 方法。

如果想以不同的参数多次执行同一条 SQL 命令，最有效率的办法是先对那条命令做一些预处理再执行。这种预处理语句的使用方法是本章 15.3 节的讨论主题。

query() 方法的第 3 种替代品是 *real_query()* 方法，后者在执行完一条 SQL 命令后不从 MySQL 取回结果。根据 SQL 命令在执行时是否出现错误，*read_query()* 方法的返回值将是 *TRUE* 或是 *FALSE*。如果通过 *read_query()* 方法执行的 SQL 命令会返回一些记录，必须用 *store_result()* 或 *use_result()* 方法来处理。（警告：如果不使用 *store_result()* 或 *use_result()* 方法来读取结果记录，后续的 SQL 命令将发生 *out-of-sync*（数据不同步）错误。）

15.2.6 处理 SELECT 查询结果 (*mysqli_result()* 方法)

在默认的情况下，*query()* 方法将把所有的结果数据从 MySQL 服务器取回到客户端，并把它们保存为一个 *mysqli_result* 对象。如果希望把结果留在服务器上、在有需要时才一条记录一条记录地读取它们，就需要在调用 *query()* 方法时加上一个 *MYSQL_USE_RESULT* 参数。这个参数在处理的数据集合尺寸比较大、不适合一次全部取回到客户端的时候很有用。换句话说，在客户端有可能不堪重负的时候，把 *SELECT* 结果在 MySQL 服务器的内存里多保留一些时间是有好处的。不过，使用 *MYSQL_USE_RESULT* 参数的缺点之一是只有把所有的结果记录全部读取完毕之后才能知道本次查询到底找到了多少条记录。

与 *mysql* 功能模块类似，*mysqli* 接口也提供了 4 个彼此很相似的方法来依次读取结果数据行：

```
$row = $result->fetch_row();
$row = $result->fetch_array();
$row = $result->fetch_assoc();
$row = $result->fetch_object();
```

- *fetch_row()*方法将以一个普通数组的形式返回一条结果记录，它的各个字段需要以 \$row[\$n] 的方式访问。
- *fetch_array()*方法将以关联数组的形式返回一条结果记录，它的各个字段需要以 \$row[\$n] 或 \$row[\$colName] 的方式（比如说，\$row[3] 或 \$row["publName"]）访问。数据列的名字区分字母的大小写。
- *fetch_assoc()*方法也将以一个关联数组的形式返回一条结果记录，但它的各个字段只能以 \$row[\$colName] 的方式访问，不允许像 *fetch_array()* 方法那样使用列号作为数组变量 \$row 的下标。
- *fetch_object()*方法将以一个对象的形式返回一条结果记录，它的各个字段需要以 \$row->colName 的方式进行访问，数据列的名字区分字母的大小写情况。

这 4 个方法的共同特点是：每次调用将自动返回下一条结果记录（但如果已经到达结果数据表的末尾，则返回 *FALSE*）。如果想改变这个顺序，就必须用 *\$result->data_seek()* 方法明确地改变当前结果记录：

```
$result->data_seek($rownr);
```

这 4 个 *fetch_xxx()* 方法都可以自动地把 SQL 语言中的 *NULL* 值转换为 PHP 语言中的 *NULL* 常数。下面这个例子使用 *fetch_assoc()* 方法来输出一条简单的 *SELECT* 命令的查询结果。这里还使用了 *htmlspecialchars()* 函数以确保来自数据库的特殊字符（如 <、>、"、& 等）都按 HTML 标准进行了编码。

```
// example mysqli-intro.php
if($result = $mysqli->query("SELECT title, subtitle FROM titles")) {
    while($row = $result->fetch_assoc()){
        if($row["subtitle"]==NULL)
            printf("<br />%s\n", htmlspecialchars($row["title"]));
        else
            printf("<br />%s -- %s\n", htmlspecialchars($row["title"]),
                htmlspecialchars($row["subtitle"]));
    }
    // release content of result
    $result->close();
}
```

SELECT 查询结果的元数据

刚才介绍的 *\$result->num_rows* 和 *\$result->field_count* 属性将分别给出结果数据表里的记录个数和数据列个数。*\$result->lengths* 属性返回的是一个数组，该数组的各个元素是 *fetch_xxx()* 方法最后读取的结果记录各字段里的字符个数。

有关查询结果更详细的数据信息可以通过对 *\$meta = \$result->fetch_fields()* 调用结果进行分析获得。这个函数将返回一个对象数组，其中的每个对象都包含着（但不限于）以下信息：

<i>\$meta[\$n]->name</i>	数据列的名字
<i>\$meta[\$n]->table</i>	数据列来自哪一个数据表
<i>\$meta[\$n]->max_length</i>	本次 <i>SELECT</i> 查询结果中最长字符串的长度
<i>\$meta[\$n]->type</i>	数据类型的 ID 编号（这是一些整数值）

下面这段示例代码利用了一个循环来提取这个对象数组里的各个元素：

```

foreach($result->fetch_fields() as $meta)
printf("<br />Name=%s Table=%s Len=%d Type=%s\n",
      $meta->name, $meta->table, $meta->max_length, $meta->type);

```

15.2.7 一次执行多条 SQL 命令

*query()*方法每次调用只能执行一条 SQL 命令。如果需要一次执行多条 SQL 命令，就必须使用*multi_query()*方法，具体做法是把多条命令写在同一个字符串里作为参数传递给*multi_query()*方法，各命令之间用分号(;)隔开。如果第一条命令在执行时没有出错，这个方法将返回 *TRUE*，否则将返回 *FALSE*。*(multi_query())*方法的返回值是 *TRUE* 并不意味着后续命令在执行时没有出错，参见下面的“警告”。)

因为每条 SQL 命令都可能返回一个结果，所以对*multi_query()*返回结果的处理也有了一些变化：第一条命令的结果要用*use_result()*或*store_result()*方法来读取。这两个方法的主要区别是：*use_result()*方法将把结果留在服务器上，它们是逐条记录传输到客户端的；*store_result()*方法则是把全部结果立刻取回到客户端，这种做法往往效率更高。

可以用*more_results()*方法来检查是否还有其他的结果。如果想对下一个结果进行处理，应该先调用*next_result()*方法检查一下，这个方法的返回值是 *TRUE*（有下一个结果）或 *FALSE*（没有下一个结果）。如果有下一个结果，它们必须用*use_result()*或*store_result()*方法来读取。

下面这个例子演示了这几种方法的典型用法：第一条 SQL 命令创建了一个变量@a，第二条命令把这个变量传递给了一个存储过程（一个由用户定义的 SQL 函数），第三条命令（*SELECT*）输出了变量@a 和@b 的值。这段代码使用了*show_result()*函数来显示每一个结果对象，它将把结果输出为一个表格（*show_result()*函数的代码清单在本章后面的内容里介绍）。具体到下面这个例子，第 3 条 SQL 命令只返回了一个结果。

```

$sql = "SET @a=12; CALL mysp(@a, @b); SELECT @a, @b";
$ok = $mysqli->multi_query($sql);
if($ok)
do {
    $result = $mysqli->store_result();
    if($result) {
        show_table($result);
        $result->close();
    }
} while($mysqli->next_result());

```

注意 如果在命令的处理过程中发生了错误，*multi_query()*和*next_result()*方法就会出问题。

*multi_query()*方法的返回值以及*mysqli*的属性*errno*、*error*、*info*和*warning_count*都只与第一条 SQL 命令有关，无法判断第二条及以后的命令是否在执行时发生错误。

这个问题的根源在于*next_result()*方法的具体实现情况：虽然底层的 C-API 函数*mysql_next_result()*区分了 3 种情况（有下一个结果、没有下一个结果、出错），但 PHP 5.0.3 版本中的这个方法只能返回 *TRUE* 或 *FALSE*。

15.2.8 带参数的 SQL 命令（预处理语句）

在生成网页的时候，许多 PHP 脚本都需要执行一系列除参数以外其他方面完全一样的查询。针对

这种情况，MySQL 从 4.1 版本开始提供了一种名为预处理语句（prepared statement）的机制：整个命令只需向 MySQL 服务器发送一次；以后只有参数发生变化。这不仅大大减少了需要传输的数据量，还提高了命令的处理效率——虽然需要执行好几条命令，但 MySQL 服务器只须对命令的结构做一次分析就够了。

从 PHP 程序设计的角度看，预处理语句的最大好处是有关代码可以编写得更精巧、更易于理解。程序员不必再为各组参数（包括字符串和 BLOB 的引用）分别构造一条合法的 SQL 命令，只要把那些参数放入几个简单的 PHP 变量再调用 *execute()* 方法即可。

在调用 *execute()* 方法之前，需要先做一些准备工作。首先，必须用 *\$mysqli->prepare()* 方法对打算执行的 SQL 命令进行处理。需要把命令中的各有关参数替换为问号（？）。作为结果，将获得一个 *mysqli_stmt* 对象，这个对象是后面所有操作的基础。下面的讨论将假设这个对象已被存入变量 *\$stmt*。

接下来，需要使用 *\$stmt->bind_param()* 方法把各有关参数绑定到一些 PHP 变量上（一定要注意它们的先后顺序是否正确），每一个参数的数据类型必须用相应的字符（如表 15-1 所示）明确给出。

表 15-1 数据类型在 *bind_param()* 方法里的表达方式

字 符	含 义
<i>i</i>	整数
<i>d</i>	浮点数（双精度）
<i>s</i>	字符串
<i>b</i>	二进制数据（BLOB、二进制字符串）

为了实际执行那条 SQL 命令，把参数值存入 PHP 变量并执行 *\$stmt->execute()* 即可。可以随时随地重复这个过程。等不再需要 *mysqli_stmt* 对象时，应该立刻明确地释放它。这么做不仅从本地内存释放了这个对象，还通知了 MySQL 服务器：后面不会再有这样的命令了，请删除它的预处理语句。

下面这个示例演示了整个过程：

```
// example mysqli-prepared.php
// prepare SQL command with parameters
$stmt = $mysqli->prepare(
    "INSERT INTO titles (title, subtitle, langID) VALUES (?, ?, ?)");
$stmt->bind_param('ssi', $title, $subtitle, $langID);
// execute command multiple times
$title = "new Linux title 1";
$subtitle = "new subtitle 1";
$langID = 1;
$stmt->execute();
$title = "new MySQL title 2";
$subtitle = "new subtitle 2";
$langID = 2;
$stmt->execute();
// release command
$stmt->close();
```

采用这个办法执行完 *INSERT*、*UPDATE* 和 *DELETE* 命令之后，*\$stmt->affected_rows* 属性将返回被修改的记录的个数，*\$mysqli->insert_id* 属性（不是 *\$stmt->insert_id* 属性）将返回最后生成的 *AUTO_INCREMENT* 值。

处理 *SELECT* 查询结果（预处理语句）

与 *\$mysqli_query()* 方法不同，*\$stmt->execute()* 方法的返回值不是一个 *mysqli_result* 对象。*mysqli_stmt* 对象提供了一种更精巧的办法来处理 *SELECT* 查询结果：在完成查询之后，用 *\$stmt->bind_result()* 方法

把查询结果的各个数据列绑定到一些 PHP 变量上；然后用 `$stmt->fetch()` 方法把下一条结果记录读取到这些变量里。`fetch()` 方法的返回值是 `TRUE`（成功地读入了下一条记录）或 `FALSE`（已经读完所有的结果记录）。

默认情况下，`SELECT` 查询结果将留在 MySQL 服务器上等待 `fetch()` 方法把记录逐条取回到客户端。如果想把所有的结果一次全部传输到客户端，需要执行一个 `$stmt->store_result()` 调用。如果需要对所有的结果记录而不只是其中的一小部分进行处理，这么做不仅更有效率，而且能减轻服务器的负担。除了读取数据，`store_result()` 方法不改变任何东西，即这个方法是可选的。

```
// example mysqli-prepared.php
// execute query
$stmt = $mysqli->prepare("SELECT titleID, title FROM titles");
$stmt->execute();
// transfer all results to the client (optional!)
$stmt->store_result();
// bind SELECT result to variables
$stmt->bind_result($titleID, $title);
// loop through all results
while($stmt->fetch())
    printf("<br />%d %s\n",
        htmlspecialchars($titleID), htmlspecialchars($title));
$stmt->close();
```

如果想知道 `SELECT` 命令查找到了多少条记录，可以从 `$stmt->num_rows` 属性检索到这个数字。不过，这个属性只有在提前执行过 `$stmt->store_result()` 调用的情况下才可以使用。

当然，在需要多次执行一条（比如说）`SELECT ... FROM ... WHERE column = ?` 形式的命令时，完全可以把本小节介绍的 `bind_result()` 方法和 15.2.7 节介绍的 `bind_param()` 方法结合起来使用。下面就是一个这样的例子：

```
// example mysqli-prepared.php
// preparation
$stmt = $mysqli->prepare(
    "SELECT titleID, title FROM titles WHERE title LIKE ?");
$stmt->bind_param('s', $pattern);
// first SELECT command
$pattern = "%Linux%";
$stmt->execute();
$stmt->store_result();
$stmt->bind_result($titleID, $title);
echo "<p></p>\n";
while($stmt->fetch())
    printf("<br />%d %s\n", $titleID, htmlspecialchars($title));
// second SELECT command
$pattern = "%MySQL%";
$stmt->execute();
$stmt->store_result();
$stmt->bind_result($titleID, $title); // possibly unnecessary
echo "<p></p>\n";
while($stmt->fetch())
    printf("<br />%d %s\n", $titleID, htmlspecialchars($title));
// clean up
$stmt->close();
```

这里有一个细节问题：只在第一次调用 `execute()` 方法后执行一次 `bind_result()` 调用是否足够？或者说是否在每次查询后都必须执行一次 `bind_result()` 调用？很遗憾，笔者未能在 PHP 文档里找到这个问题的答案。在笔者进行的测试中，上面这个例子没有第二条 `bind_result()` 调用语句也工作得很好。但这种

行为能否依赖还不清楚。为保险起见，最好成对使用 `execute()` 和 `bind_result()` 方法（反正也花不了多少时间）。

15.2.9 事务

`mysqli` 接口目前没有提供与 SQL 命令 `BEGIN TRANSACTION` 和 `START TRANSACTION` 相对应的方法。如果想使用事务，必须先执行一个 `$mysqli->autocommit(0)` 调用暂时关闭 MySQL 事务机制的自动提交模式。（`mysqli` 接口目前没有提供任何可以用来查看当前提交模式的方法或属性，只能通过执行一条 `SELECT @@autocommit` 命令查看这一信息。这里还要再次提醒：MySQL 目前只有 InnoDB 数据表才支持事务。）

暂时关闭自动提交模式后，后续执行的所有 SQL 命令将构成一个事务，直到用 `mysqli` 类对象的 `commit()` 方法提交它们或者是用 `rollback()` 方法撤销它们为止。接下来执行的 SQL 命令又构成了另一个事务，直到再次遇到 `commit()` 或 `rollback()` 调用。

注意 在执行 `commit()` 调用之前（或者是忘了应该执行一个 `commit()` 调用），一旦有 SQL 命令执行出错或是与 MySQL 服务器的连接掉线，当前事务里的所有 SQL 命令都将被撤销。

除了 `autocommit()`、`commit()` 和 `rollback()` 方法，还可以简单地调用 `real_query()` 方法去执行一条相应的 SQL 命令，比如像 `$mysqli->real_query("START TRANSACTION")` 这样。

15.3 把数据库功能打包为一个类

绝大多数 PHP 项目都包括大量的 PHP 文件，其中又有相当一部分文件包含同样的代码片段：与 MySQL 数据库建立一个连接、执行简单的查询、处理各种常见错误等。这种做法并不好，原因有以下几条：

- **冗余。** 冗余代码总是越少越好。否则，每发现一个错误，就必须对几个甚至几十个地方进行修改。这种情况在数据库系统的登录信息发生变化时也会发生，比如说，如果 ISP 要求改用另一个用户名和密码，将不得不在几个甚至几十个文件里修改这些信息。
- **信息安全。** 出于信息安全方面的考虑，MySQL 用户名和密码不应该以明文的形式出现在任何可以从网络直接访问的 PHP 文件里。
- **出错处理。** 为了维护方便，也为了最大限度地提高应用代码的紧凑度和可读性，把出错处理例程集中存放在一起很有必要。不应该让应用代码和出错处理代码混杂在一起。

这 3 个问题的解决方案其实是一个，就是把所有的 MySQL 登录信息、常用函数、出错处理例程等分门别类地集中存放到几个 PHP 文件里，再在各有关项目文件里把它们声明为头文件。

15.3.1 使用单独的密码文件提高安全性

从理论上讲，网站的访问者不应该看到任何 PHP 源代码——这些代码会在 Web 服务器上执行，访问者只能看到它们生成的 HTML 结果文档。但事情没有那么绝对，有很多意外情况会让访问者看到明文形式的 PHP 文件内容。

- Web 服务器未经配置或是配置不当（这种情况多发生在 Web 服务器升级之后），进而导致它不是去执行 PHP 文件的内容（代码）而是把它们直接发送给了访问者的浏览器。

- 通过匿名 FTP 服务去访问 PHP 页面。（这种情况属于严重的配置失误！只要 Web 服务器上有提供匿名 FTP 服务的目录，就一定要把它们与用来存放 HTML 和 PHP 文件的目录区分开！）
- PHP 文件在修改之后会留下一个备份副本（文件名多为*.php~或*.php.bak 形式）。因为它们的文件标识符已不再是*.php 形式，所以 PHP 解释器也就不会再对它们进行处理。有经验的攻击者是不会放过这些文件的。

为了不让陌生人轻易获得 MySQL 密码，应该把 MySQL 登录信息单独保存到另外一个文件里去。这些信息如下所示：

```
<?php
// file password.php
$mysqluser="user";           // user name for MySQL access
$mysqlpasswd="xxx";          // password
$mysqlhost="localhost";       // name of the computer on which MySQL is running
$mysqldb="mylibrary";         // name of the database
?>
```

把这个文件存放在哪儿才安全要取决于 Web 服务器的具体配置情况。笔者在自己的个人网站上创建了一个 htdocs/_private/ 目录并用了一个 .htaccess 文件来防止别人访问该目录。这样一来，如果不知道笔者存放在那个 .htaccess 文件里的 HTTP 用户名和存放在相关身份验证文件里的加密密码，就根本不能访问 http://www.kofler.cc/_private/password.php 文件——就算知道了，从 Web 浏览器里看到的也会是一片空白。

提示 一定要把密码文件和其他头文件命名为*.php 的形式（不要命名为*.inc），这将确保 PHP 解释器会在有人通过 HTTP 直接访问那些文件时是运行它们而不是把它们发送出去。

加载密码文件

在某个 PHP 文件试图使用 `require_once()` 函数去连接 MySQL 服务器的时候，那个文件必须能加载密码文件才可以建立起连接。`require_once()` 函数的作用是限定密码文件只加载一次，即使在几个相互调用的文件里都出现了这个函数也是如此。如果没有找到密码文件，`require_once()` 函数将返回一个错误并停止执行自己所在的脚本。

不妨假设 `intro.php` 和 `password.php` 文件分别位于目录 `/www/user1234/htdocs/php-example/general/` 和目录 `/www/user1234/htdocs/_private/password.php` 里，那么 `intro.php` 文件里的 `require_once()` 函数就必须像下面那样去寻找 `password.php` 文件。路径名里的连续两个句点(..) 意味着必须使用一个目录。在自己的系统上，请根据 PHP 脚本与头文件各自所在的目录的相对位置来写出 `require_once()` 函数里的路径名。

```
// file intro.php
require_once("../_private/password.php");
$mysqli = new mysqli($mysqlhost, $mysqluser, $mysqlpasswd, $mysqldb);
... 
```

15.3.2 使用 MyDb 类实现安全和方便

把 MySQL 登录数据隐藏起来之后，还应该更进一步地把访问有关数据库所必需的代码集中存放到了一个单独的文件。也就是说，应该把 PHP 项目最经常用到的 MySQL 函数全部封装在那个文件里。在这么做的过程中，还可以利用 PHP 5 为支持面向对象编程技术而提供的各种新功能把相关代码封装为一个它自己的类。

下面这些示例代码可以让大家对一个这样的类应该是什么样子有一个直观的认识。在这个示例里使用的是 *mysqli* 接口：用户完全可以根据自己的具体情况把它们改写为使用 *mysql* 功能模块：

```
// example file mydb.php
class MyDb {
    protected $mysqli;
    // constructor (create an object of this class)
    function __construct() {
        require_once('password.php');
        $this->mysqli = @new mysqli($mysqlhost, $mysqluser,
            $mysqlpasswd, $mysqldb);
        // test whether connection OK
        if(mysqli_connect_errno()) {
            printf("<p>Sorry, no connection! %s</p>",
                mysqli_connect_error());
            // add any needed HTML code to conclude the code
            // (</body></html> etc.)
            $this->mysqli = FALSE;
            exit();
        }
    }
    // destructor (delete object)
    function __destruct() {
        $this->close();
    }
    // explicitly terminate object/connection
    function close() {
        if($this->mysqli)
            $this->mysqli->close();
        $this->mysqli = FALSE;
    }
    // execute SELECT return object field
    function queryObjectArray($sql) {
        if($result = $this->mysqli->query($sql)) {
            if($result->num_rows) {
                while($row = $result->fetch_object())
                    $result_array[] = $row;
                return $result_array; }
            else
                return FALSE;
        } else {
            printf("<p>Error: %s</p>\n", $this->mysqli->error);
            return FALSE;
        }
    }
    // execute SELECT return individual value
    // Note: return value for error is -1 (not 0)!
    function querySingleItem($sql) {
        if ($result = $this->mysqli->query($sql)) {
            if ($row=$result->fetch_array()) {
                $result->close();
                return $row[0];
            } else {
                return -1;
            }
        } else {
            printf("<p>Error: %s</p>\n", $this->mysqli->error);
            return -1;
        }
    }
    // execute SQL command without result (INSERT, DELETE, etc.)
    function execute($sql) {
```

```

if ($this->mysqli->real_query($sql)) {
    return TRUE;
} else {
    print $this->mysqli->error;
    return FALSE;
}
}
// return insert_id
function insertId() {
    return $this->mysqli->insert_id;
}
}

```

1. 对MyDb类进行调试

变量在这个类里扮演着非常重要和实用的角色。它们控制着已经执行过的所有 SQL 命令和将被输出到 HTML 结果文档里的所有出错消息。这些变量在代码开发阶段非常实用，它们可以大大简化查找出错根源的工作（如图 15-4 所示）。下面这段代码给出了应该如何对 *execute()* 函数进行修改。

```

class MyDb {
    protected $mysqli;
    protected $showerror = TRUE; // display error message
    protected $showsql = FALSE; // display SQL code
    function execute($sql) {
        $this->printsq($sql);
        if($this->mysqli->real_query($sql))
            return TRUE;
        else {
            $this->printerror($this->mysqli->error);
            return FALSE;
        }
    }
    ...
    private function printsq($sql) {
        if($this->showsql)
            printf("<p><font color=\"#0000ff\">%s</font></p>\n",
                htmlspecialchars($sql));
    }
    private function printerror($txt) {
        if($this->showerror)
            printf("<p><font color=\"#ff0000\">%s</font></p>\n",
                htmlspecialchars($txt));
    }
}

```

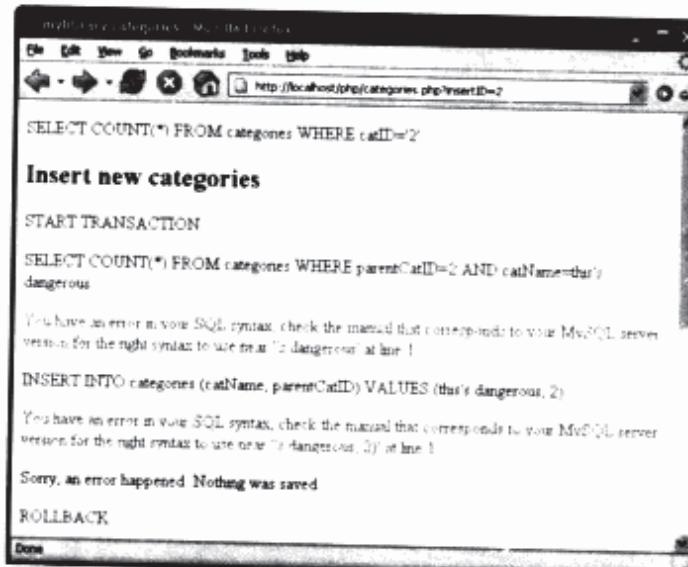


图 15-4 出错根源很容易查找

提示 在本章稍后的内容里，笔者还将提出一些对 *MyDb* 类的改进办法，它们可以帮助用户更有效地对 PHP 页面（脚本）进行优化。那里的 *showStatistics()* 方法将把以下信息显示出来：已经执行了多少条 SQL 命令、已经返回了多少条记录、已经过去了多长的时间等。

2. *MyDb*类应用示例

基于 *MyDb* 类的 PHP 代码非常紧凑，如下所示：

```
// example file test-mydb.php
require_once("mydb.php");
$db = new MyDb();
// query for single value
if(($n = $db->querySingleItem("SELECT COUNT(*) FROM titles"))!=-1)
    printf("<p>number per title: %d</p>\n", $n);
// SELECT
if($result = $db->queryObjectArray("SELECT * FROM titles")) {
    foreach($result as $row)
        printf("<br />TitleID=%d Title=%s Subtitle=%s\n",
            $row->titleID, $row->title, $row->subtitle);
}
```

本章后面还有许多示例都使用了 *MyDb* 类。

15.4 把 *SELECT* 查询结果显示为一个表格

下面这个例子演示了如何把一个简单的查询命令 (*SELECT * FROM titles*) 的返回结果显示在一个 HTML 表格里（如图 15-5 所示）。用来输出 HTML 表格的代码包含在图背后的 *show_table()* 函数里。所有的字符串都用 PHP 函数 *htmlspecialchars()* 转换为正确的 HTML 编码，这个函数可以确保包含 HTML 特殊字符（比如<和>）的字符串也可以正确显示。示例中的 HTML 表格采用的是最简单的结构，但只需稍加努力就可以用 CSS 文件 *table.css* 把这个表格的外观改造成喜欢的样子。

titleID	title	subtitle	editor	pubID	catID	langID	year

图 15-5 把查询结果显示为一个 HTML 表格

```
// example file mysqli-table.php
// displays the result of a query as an HTML table
function show_table($result) {
    if(!$result) {
        echo "<p>No valid query result.</p>\n";
        return;
    }
    // ... (rest of the code for generating the HTML table)
```

```

    }
    if($result->num_rows>0 && $result->field_count>0) {
        echo "<table>";
        // column labels
        echo "<tr>";
        foreach($result->fetch_fields() as $meta)
            printf("<th>%s</th>", htmlspecialchars($meta->name));
        echo "</tr>\n";
        // table content
        while($row = $result->fetch_row()) {
            echo "<tr>";
            foreach($row as $col)
                printf("<td>%s</td>", htmlspecialchars($col));
            echo "</tr>\n";
        }
        echo "</table>\n";
    }
}

```

下面这个示例演示了 *show_table()* 函数的用法：

```

if($result = $mysqli->query("SELECT * FROM titles")) {
    show_table($result);
    $result->close();
}

```

15.5 字符串、日期、时间、BLOB 和 NULL

在读取和输出来自 MySQL 数据表的数据时，必需遵循 HTML 语法规则。类似地，在构造 *INSERT* 和 *UPDATE* 命令的时候，又必需遵循 MySQL 语法。本节将介绍一些处理这类问题时的技巧。

提示 如果正在使用 *mysql* 功能模块，可以利用绑定参数和绑定结果变量来构造语句（参见本章前面的有关讨论）。这么做的好处是 PHP 会自动地设置好字符串、BLOB 和日期/时间的输出格式。

1. 存储数据

如果想改变数据库里的记录，就必须把必要的 SQL 命令作为一个字符串传递给 *mysql* 功能模块中的 *mysql_query()* 函数或 *mysqli* 接口中的 *mysqli->query()* 方法。下面是构造一条 *INSERT* 命令的例子：

```

$sql = "INSERT INTO tablename (column, column2, ...)"
    . "VALUES($data1, $data2, ...)";

```

2. 读取数据

从数据库读取数据的基本步骤是：先用 *mysql_query()* 函数或 *mysqli->query()* 方法执行一条查询命令，再用 *mysql_fetch_xxx()* 函数或 *mysqli->fetch_xxx()* 方法对查询结果 *\$result* 进行处理：

```

$result = mysql_query("SELECT ...");
$row = mysql_fetch_row($result);
$data = $row[0];

```

15.5.1 字符串和 BLOB

1. 把字符串存入MySQL数据库

在 SQL 命令里，字符串必须用引号括起来，单引号 (') 和双引号 (") 均可。当字符串本身包含引号或其他 SQL 特殊字符（反斜线字符 \ ）的时候，问题就来了，这很容易导致语法错误：

```
$data = "O'Reilly";
$sql = "INSERT INTO publishers (publName) VALUES('$data');
```

如此构造出来的 SQL 命令是非法的，如下所示：

```
INSERT INTO publishers (publName) VALUES('O'Reilly')
```

这时候需要使用 *mysql_real_escape_string()* 函数或 *\$mysqli->escape_string()* 方法，它们会把单引号(')、双引号(")、反斜线(\)和 0 字节分别替换为\\'、\\"、\\\\和\\0。 (这种转换叫做字符的转义。)

```
$sql = "INSERT INTO publishers (publName) VALUES('"
    . $mysqli->escape_string($data) . "')";
```

现在生成的 SQL 命令就是正确无误的了：

```
INSERT INTO publishers (publName) VALUES('O\\'Reilly')
```

注解 如果将被存储的数据来自一个HTML表单并且PHP配置变量 *magic_quotes_gpc* 已被设置为 *On*，数据就将是已经转义好了的，当然也就用不着再对它们进行一次转义了。对字符转义问题的详细讨论见第 5 章。

2. 把字符串输出到HTML结果文档

在把字符串输出到一份 HTML 文档里去的时候，应该用 *htmlspecialchars()* 或 *htmlentities()* 函数：前者会把字符<、>、"和&分别替换为<、>、"和&；后者会把更多的特殊字符替换为相应的 HTML 编码，比如把 ä 替换为ä等。这在 HTML 文档的字符集无法提供特殊字符的直接表示时尤其必要。

htmlspecialchars() 和 *htmlentities()* 函数还有两个可选的参数。第一个参数控制着是否需要对字符'和"进行替换，它有 3 个可取值：*ENT_QUOTES*（替换'和")、*ENT_NOQUOTES*（不替换'和") 和 *ENT_COMPAT*（替换"，但不替换')；默认设置是 *ENT_COMPAT*。

第二个参数给出了输出字符串所使用的字符集。在默认的情况下，这两个函数都将假设输出字符串使用的是 *Latin1* 字符集（ISO-8859-1），其他的可取值包括 *iso-8859-15*、*utf8* 和 *cpi252*。

比如，如果数据库里存放的是 Unicode 字符串，正确的 *htmlentities()* 函数调用就应该如下所示：

```
printf("<p>%s</p>\n",
    htmlentities($data, ENT_COMPAT, "UTF-8"));
```

针对 Unicode 字符串的更多处理技巧见本章后面的有关内容。

3. BLOB（二进制数据）

PHP 把二进制数据（BLOB 数据）当做字符串对待。PHP 不像 C 语言那样存在零字节(\0)问题。

15.5.2 日期和时间

1. 把日期/时间数据存入MySQL数据库

为了让日期/时间数据的格式符合 MySQL 语法规则，可以使用 *date()* 和 *strftime()* 等 PHP 函数。如果数据来自用户输入，就必须对它们进行常规的合法性检查。（MySQL 的数据合法性检查功能很弱，很多不可能存在的日期/时间都可以通过检查。）

在另一方面，如果想把一个 PHP 时间戳存入一个 MySQL 的 *TIMESTAMP* 数据列，就应该在 *INSERT* 或 *UPDATE* 命令里使用 MySQL 函数 *FROM_UNIXTIME()*，如下所示：

```
$data = time(); // data contains the current time as a Unix timestamp
$sql = "INSERT INTO table (column) VALUES (FROM_UNIXTIME($data));
```

2. 从MySQL数据库读出日期/时间数据

当从一个 *DATETIME* 数据列读出日期/时间的时候，得到的是一个 *2005-12-31 23:59:59* 格式的字符串。从 *DATE* 数据列读出的数据也是一个这样的字符串，只是没有时间部分。如果需要在 PHP 脚本里对来自 MySQL 的日期/时间数据进行处理，最简便实用的办法是用 MySQL 函数 *UNIX_TIMESTAMP()* 读取它们，该函数将返回一个 32 位整数，整数的值是从 1970 年 1 月 1 日开始算起的秒数。

```
$sql = "SELECT UNIX_TIMESTAMP(column) FROM table WHERE ...";
$result = $mysqli->query($sql);
$row = $result->fetch_row();
$ts = $row[0];
```

可以用 MySQL 函数 *DATE_FORMAT()* 来获得需要的格式。下面这 SQL 命令将返回一个 *December 31 2005* 格式的字符串：

```
SELECT DATE_FORMAT(column, '%M %d %Y') FROM table
```

从 *TIME* 数据列读出的时间是 *23:59:59* 格式的字符串。如果不从这个字符串里提取小时、分钟和秒，可以用 MySQL 函数 *TIME_TO_SECOND()* 获得一个从 *00:00:00* 开始算起的秒数。除了这里提到的几个函数，MySQL 还提供了很多用来处理日期/时间的其他函数。

注意 *UNIX_TIMESTAMP()* 函数不能在 *TIME* 数据列上使用！

从 *TIMESTAMP* 数据列读出的日期/时间是一个 *yyyy-mm-dd hh:mm:ss* 格式的字符串。如果需要的是一个与 PHP 兼容的时间戳值，在 SQL 查询命令里使用 *UNIX_TIMESTAMP()* 函数，如下所示：

```
SELECT UNIX_TIMESTAMP(column) FROM table
```

15.5.3 NULL 值

存储NULL值

如果想把 *NULL* 值存入一个 MySQL 数据列，把不带引号的字符串 *NULL* 传递给 SQL 命令即可：

```
INSERT INTO table (column) VALUES (NULL)
```

如果需要用 *NULL* 值来代替空字符串，最好使用如下所示的代码：

```
$sql = "INSERT INTO table (column) VALUES (";
if(isset($data))
    $sql .= "'" . $mysqli->escape_string($data) . "'}";
else
    $sql .= "NULL)"
NULL-Test
```

在从一个允许出现 *NULL* 值的 MySQL 数据列读出数据的时候，必须在对读出的数据做进一步处理之前先检查它是不是 *NULL* 值。如果使用的是 *mysql* 功能模块，*isset(\$data)* 可以完成这一检查。如果被检查的数据是 *NULL* 值，这个函数将返回 *FALSE*。类似地，还可以用 *is_numeric()*、*is_string()* 等函数对其他的数据类型进行测试。注意：用 *empty()* 函数去检查字符串时可能会得到不正确的结果；这个函数在遇到 *NULL* 值和空字符串（“”）时的返回值都是 *TRUE*。

mysqli 接口向程序员提供了更多的方便：4 种 *fetch_xxx()* 方法都可以把 MySQL 中的 *NULL* 值字段

转换为 PHP 中的 `NULL` 值。此外，还可以直接执行一个 `if($data==NULL)` 形式的查询。

15.6 向关联数据表插入新数据记录

如果只是把一条特定的记录插入一个特定的数据表，简单地执行一条 `INSERT` 命令即可。但如果事情涉及到多个彼此关联的数据表，问题就会变得困难许多，因为必须保证在执行 `INSERT` 命令的时候使用的是正确的关键字。

作为一个普遍原则，`AUTO_INCREMENT` 数据列几乎总是两个关联数据表之间的联结纽带。可以利用 `$mysqli->insert_id()` 方法（`mysqli` 接口）或 `mysql_insert_id()` 函数（`mysql` 功能模块）来确定最新插入的数据记录的 ID 值。

下面这段代码将把一本由两位作者合著的新图书存入 `mylibrary` 数据库。注意，为了能在最后把一条新记录插入 `rel_title_author` 数据表，在处理过程中必须查出并保存 3 个 ID 值。这段示例代码的前提假设是新图书的出版公司（Addison-Wesley 出版公司）已经收录在了这个数据库里，它的 `publID=1`。

```
$mysqli->query("INSERT INTO titles (title, publID, year)
    VALUES ('A Guide to the SQL Standard', 1, 1997)");
$titleID = $mysqli->insert_id;
$mysqli->query("INSERT INTO authors (authName)
    VALUES ('Date Chris')");
$author1ID = $mysqli->insert_id;
$mysqli->query("INSERT INTO authors (authName)
    VALUES ('Darween Hugh')");
$author2ID = $mysqli->insert_id;
$mysqli->query("INSERT INTO rel_title_author (titleID, authID)
    VALUES ($titleID, $author1ID),
    ($titleID, $author2ID)");
```

在实际应用中，数据往往是以交互方式输入的，而这又往往会把新记录插入数据库的工作变得更加复杂。比如说，必须检查新图书的作者是否早已收录在了数据库里。如果是，把这位“老”作者的 ID 存入 `rel_title_author` 数据表即可；如果不是，就必须先往 `author` 数据表插入一条新记录，然后才能把这位新作者的 ID 存入 `rel_title_author` 数据表。（万一新作者的名字是因为打字错误而导致的怎么办？为了把这种偶然性带来的麻烦降到最低，或许应该把发音相似或拼写相似的名字也列出来供人们核对和选择。）

15.7 处理来自 HTML 表单的输入数据

本节将为 `mylibrary` 数据库实现一个供人们输入和修改图书信息的 HTML 表单（如图 15-6 所示）并对有关代码进行解释和分析。这里给出的示例代码相当详尽。这个示例很值得大家认真研究，原因很简单：几乎每一个数据库应用都会涉及到对来自一个 HTML 表单的输入数据进行检查、处理和存储，以及把现有数据传输到一个 HTML 表单供人们修改或删除的问题。如果说这个例子有什么特别，那就是表单数据不是来自同一个数据表而是来自多个彼此关联的数据表。

这个 HTML 表单本身的用法没有什么需要多说的：大多数输入字段都是可选的，只有图书名称（*Title*）和作者姓名（*Author*）这两个字段必须填写。如果一本图书有多位作者，在输入时必须用分号把他们的姓名隔开。在出版公司（*Publisher*）和语言（*Language*）这两个输入字段里，既可以从下拉列表里选择一个现有的，也可以输入一家新的出版公司和一种新的语言。`Delete title`（删除图书）按钮只在这个表单是用来修改一本现有图书时才是可用的。

The screenshot shows a Mozilla Firefox browser window with the title 'mylibrary title form - Mozilla Firefox'. The address bar displays 'Http://localhost/php/titleform.php?editID=11'. The main content area is titled 'Edit title'. A note at the top states: 'You must specify at least the title and one author.' Below this are instructions for authors and publishers. The form fields include:

- Title:** A Guide to the SQL Standard
- Subtitle:** (empty)
- Authors:** Darween Hugh;Date Chris
- Publisher:** Addison-Wesley (dropdown menu)
- New publisher:** (text input field)
- Category:** SQL (dropdown menu)
- Edit categories:** (link)
- Publishing year:** 1997
- Edition:** (dropdown menu)
- Language:** english (dropdown menu)
- New language:** (text input field)
- Save** button
- Delete title** button
- Input new title (clear form)** button

At the bottom of the form, there is a search bar labeled 'Search for titles/authors' and a 'Done' button.

图 15-6 mylibrary 数据库的输入表单

15.7.1 代码结构

有关代码分别存放在表 15-2 所示的几个文件里。

表 15-2 有关代码存放的文件

文件名	内 容
formtitle.php	主控代码
mydb.php	MyDb 类，用来完成各种数据库操作的（函数）方法
mylibraryfunctions.php	辅助函数
formfunctions.php	用来生成 HTML 表单的辅助函数
form.css	对 HTML 表单进行排版的 CSS 文件

表 15-3 是 HTML 表单里的数据发生变化时用来传输有关数据的两种机制。

表 15-3 两种机制

机 制	数 �据
POST	数组形式的表单数据（如 <code>form["title"]</code> 、 <code>form["author"]</code> 等）
REQUEST	URL 参数（比如说，参数 <code>editID</code> 里对应着需要修改的图书 ID 号）

注解 formtitle.php 文件与稍后将要介绍的 find.php 和 categories.php 两个文件构成了一个完整的示例：输入新图书、检索现有图书和管理图书门类。这 3 个脚本在执行时都需要访问在 mydb.php、mylibraryfunctions.php 和 formfunctions.php 3 个文件里定义的类和函数。

在 mylibraryfunctions.php 文件里定义了一个 `array_item()` 函数，这个简短但不可缺少的函数在表单数据的处理工作中扮演着重要的角色。`array_item($x, "abc")` 调用将返回 `$x["abc"]`（如果这个元素

存在的话) 或 *FALSE* (如果这个元素不存在的话)。要知道, 并不总是能提前知道在一个给定的数组里都存在着哪些元素, 而 *array_item()* 函数可以让用户少看到很多烦人的“*xxx* 元素在数组中不存在”的警告消息或“*xxx* 字段不存在”的出错消息。

```
function array_item($ar, $key) {
    if(is_array($ar) && array_key_exists($key, $ar))
        return $ar[$key];
    else
        return FALSE; }
```

代码的管理

相对而言, *title.php* 文件里的主控代码并没有多少行, 所以把它们列在了下边, 而所有与之相关的代码都可以在 *MyDb* 类的各有关函数里找到。根据将要传递给 HTML 页面的数据, 这个示例可以分为两大部分。

□ 对一本现有图书进行处理(*editID*参数值已知)。此时, *titleID* 编号值将被传递到 *read_title_data()* 函数。这个函数将检索出所有的数据(图书、作者、出版公司等)并把它们返回为一个数组。这个数组与表单数据有着同样的结构。

□ 对表单数据进行处理。此时, 根据用户在提交表单时单击的是哪一个按钮, 将需要保存或删除一条图书记录或是清空整个表单(以便用户输入一条新的图书记录)。表单数据存放在数组 *formdata[]* 里, 该数组里的各个元素已被滤掉了转义前导字符。

如果表单是用户单击 *Save* (保存) 按钮提交的, 新的图书记录将只有在 *validate_data()* 函数对它进行的数据合法性检查全部成功的情况下才会被存入数据库。接下来, 将删除 *formdata[]* 数组并向用户显示一个空白的表单, 以便他继续输入下一条图书记录。

如果表单是用户单击 *Delete* (删除) 按钮提交的, 有关的图书记录将在经过一个查询/确认的步骤后才会被删除。查询/确认表单由 *build_delete_form()* 函数负责生成和输出, 该表单上的 *Delete* (删除) 按钮的名字是 *btnReallyDelete*。

图书记录的存储和删除操作将使事务在内存里完成, 只有在整个过程没有出现任何错误的情况下我们才会用 *COMMIT* 命令确认本个事务。

最后, 还需要对表单进行刷新并把它显示给用户。这里分为 3 种情况: 第一种情况是为了让用户改正打字错误而用现有数据去刷新表单; 第二种情况是为了让用户继续输入下一条图书记录而显示一个空白表单; 第三种情况是显示一条关于图书记录删除操作的确认消息。是否需要对表单进行刷新显示由变量 *\$showform* 控制; 如果变量 *\$showform== FALSE* (仅对应于第三种情况), 则不刷新表单(改为显示关于删除操作的确认消息)。

注解 虽然这本书不是一本专门讨论 PHP 的著作, 但趁这个机会介绍一下 PHP 的自动化字符转义功能(有关文档称之为 *Magic Quotes* 功能)应该没有坏处。*Magic Quotes* 是 PHP 开发团队为了减轻 PHP 程序员的编程负担而提供的一项功能, 该功能可以自动地给来自外部来源(比如说, 用 *GET* 或 *POST* 方法提交的表单、服务器发送给客户的 *cookie* 等)的字符串里的特殊字符加上反斜线(\)。这有助于避免特殊字符在(比如说)复杂的 SQL 命令里引起问题。*Magic Quotes* 功能总的来说非常有用, 只可惜事情并非总是如此。

问题的根源在于 *Magic Quotes* 功能是在 PHP 配置文件 *php.ini* (*magic_quotes_gpc* 变量) 里启用或禁用的。如果 PHP 脚本正运行在一台属于某个 ISP 的计算机上, 对那台计算机上的 PHP 配置往往没有任何控制, 而这意味着在每一次开始处理表单数据之前都必须先去检查 *Magic Quotes* 功能是

否处于启用状态。在本章的示例里，我们将先去掉由*Magic Quotes*功能自动添加的字符转义序列（删除多余的反斜线），再开始正式的处理（详见下面的程序清单）。

```
$db = new MyDb(); // establish connection to MySQL
$showform=TRUE; // should the form be displayed at the end?

if(($editID = array_item($_REQUEST, 'editID'))
&& is_numeric($editID))
// case 1: process existing title
$formdata = read_title_data($editID);

else {
    // case 2: process form data
    $formdata = array_item($_POST, "form");
    if(is_array($formdata)) {
        if(get_magic_quotes_gpc()) // eliminate Magic Quotes
            while($i = each($formdata))
                $formdata[$i[0]] = stripslashes($i[1]);

        // respond depending on button
        if(array_item($formdata, "btnClear"))
            // clear form
            $formdata = FALSE;

        elseif(array_item($formdata, "btnDelete")) {
            // query whether title really should be deleted
            if(build_delete_form($formdata))
                $showform = FALSE; }

        elseif(array_item($formdata, "btnReallyDelete")) {
            // delete title
            $db->execute("START TRANSACTION");
            if(delete_title($formdata))
                $db->execute("COMMIT");
            else
                $db->execute("ROLLBACK");
            $formdata = FALSE; }

        elseif(validate_data($formdata)) {
            // check data and store if ok
            $db->execute("START TRANSACTION");
            if(save_data($formdata))
                $db->execute("COMMIT");
            else
                $db->execute("ROLLBACK");
            $formdata = FALSE; }
    }
}

if($showform) {
    // alter data or input new?
    if(array_item($formdata, "titleID"))
        echo "<h1>Edit title</h1>";
    else
        echo "<h1>Input new title</h1>";

    // display form
    build_form($formdata);
}
```

15.7.2 创建 HTML 表单

只有了解了 HTML 表单的内部结构，才能更好地把握它的处理流程。正是出于这一考虑，在下面

给出了一些 HTML 代码，这个示例中的 HTML 表单就是由它们生成的。请特别注意其中的 *form[title]*、*form[publisher]* 等数组变量名（黑体字部分）。为了对这个 HTML 表单进行排版，这里使用了一个由 4 个列构成的 HTML 表格。

```

<table class="myformtable">
<form method="post" action="titleform.php">
<tr><td class="myformtd" colspan="4">explanation ...</td></tr>
<tr><td align="right" class="myformtd">
    <span class="red">Title:</span></td>
    <td class="myformtd" colspan="3">
        <input class="mycontrol" name="form[title]" size="60"
            maxlength="100" value="Client/Server Survival Guide"/></td>
    </tr>
    ...
<tr><td align="right" class="myformtd">Publisher:</td>
    <td class="myformtd">
        <select class="mycontrol" name="form[publisher]" >
            <option value="none">(choose)</option>
            <option value="1" selected="selected" >Addison-Wesley</option>
            ...
        </select></td>
        <td align="right" class="myformtd">New publisher:</td>
        <td class="myformtd">
            <input class="mycontrol" name="form[newpubl]"
                size="20" maxlength="40" /></td>
    </tr>
    ...
<tr><td align="right" class="myformtd"></td>
    <td class="myformtd">
        <input class="mybutton" type="submit" value="Save"
            name="form[btnSave]" /></td>
    <td class="myformtd">
        <input class="mybutton" type="submit" value="Delete title"
            name="form[btnDelete]" /></td>
        <input type="hidden" name="form[titleID]" value="3" />
    <td class="myformtd">
        <input class="mybutton" type="submit"
            value="Input new title (clear form)"
            name="form[btnClear]" /></td>
    </tr>
</form></table>
```

这个表单的整体布局及其各个格子和控件的细节外观由 *form.css* 文件里的几条 CSS 定义语句负责确定：

```

/* example file form.css */
.myformtable {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    border: 5px solid #9090e0;
    border-spacing: 0px;
    border-collapse: collapse; }
.myformtd {
    background-color: #b0b0ff;
    border-width: 4px 0px;
    border-style: solid;
    border-color: #9090e0;
    padding: 3px 5px; }
.mycontrol { ... }
.mybutton { ... }
.red { color: red; }
```

1. 显示HTML输入表单

表单数据将通过一个数组传递给 *build_form()* 函数（如果数据需要改正或改变的话）。这些数据将

被插入它们各自对应的表单元素 (*value*="...")。

用 PHP 代码来创建 HTML 表单是一项出现频繁并且容易出错的工作。在本章以及本书的许多示例里，我们把这项工作交给了一些辅助函数去完成，它们的代码见 *formfunctions.php* 文件（请参见下一节）。

```
// example file titleform.php
function build_form($formdata) {
    global $db; // database access
    form_start("titleform.php");
    // input field for book title
    form_new_line();
    form_label("Title:", TRUE);
    form_text("title", array_item($formdata, "title"), 60, 100, 3);
    form_end_line();
    // input field for subtitle and author names
    ... and so on
```

从数据库的角度看，如何把出版公司的名单显示在一个下拉列表框里是一个很让人感兴趣的问题。出版公司的名单是用一个 SQL 查询命令生成的，这份名单将以一个二维数组的形式被传递给 *form_list()* 函数。该函数的第 3 个参数负责指定应该选取这个二维数组中的哪一个元素。

```
// selection list for publisher
form_new_line();
form_label("Publisher:");
$sql = "SELECT publName, publID FROM publishers ORDER BY publName";
form_list("publisher", $db->queryArray($sql),
    array_item($formdata, "publisher"));
form_label("New publisher:");
form_text("newpubl", array_item($formdata, "newpubl"), 20, 40);
form_end_line();
```

把图书门类清单显示在一个下拉列表框里的工作要更复杂一些。因为这次处理的数据构成了一个层次化的结构，必须对清单里的元素进行必要的缩进。这个层次化的图书门类清单（下拉列表框）由 *build_category_array()* 函数（代码见 *formfunctions.php* 文件）负责生成。在本章稍后将对层次化数据的处理问题进行讨论时再对这个递归函数做详细的分析。

```
// selection list for category
form_new_line();
form_label("Category:");
// determine all categories
$sql = "SELECT catName, catID, parentCatID FROM categories ".
    "ORDER BY catName";
$rows = $db->queryObjectArray($sql);
// form two associative fields
// subcats[catID] contains a field with sub-catIDs
// catNames[catID] contains the name
foreach($rows as $row) {
    $subcats[$row->parentCatID][] = $row->catID;
    $catNames[$row->catID] = $row->catName; }
// form hierarchical category list
$rows = build_category_array($subcats[NULL], $subcats, $catNames);
form_list("category", $rows, array_item($formdata, "category"));
form_url("categories.php", "Edit categories", 2);
form_end_line();
```

这个 HTML 表单里的其他文本输入字段和文本标记字段不需要特殊的处理，但显示在那些字段下方的按钮需要在这里解释一下。Delete（删除）按钮只在 HTML 表单里显示一个现有图书记录的时候才会出现，那条图书记录的 *titleID* 编号值此时将被传递为一个不可见的 HTML 元素，这是为了确保

这个 *titleID* 编号值肯定会被包含在表单数据里。

```
// buttons
form_new_line();
form_label("");
form_button("btnSave", "Save");
// delete button for existing title
if(array_item($formdata, "titleID")) {
    form_button("btnDelete", "Delete title");
    form_hidden("titleID", $formdata["titleID"]); }
else
    form_empty_cell(1);
form_button("btnClear", "Input new title (clear form)");
form_end_line();
// end of the form
form_end();
```

2. 与HTML表单创建工作有关的辅助函数

build_form() 函数需要用到的辅助函数都集中存放在 *formfunctions.php* 文件里，下面将对其中几个比较重要的函数做一个简单的解释。在这本书里，还有许多其他的示例也要用到这些函数。

```
// example file formfunctions.php
// start and end form
function form_start($action) {
    echo '<table class="myformtable">', "\n";
    echo '<form method="post" ';
    html_attribute("action", $action), ">\n"; }
function form_end() {
    echo "</form></table>\n\n"; }
// start and end one line of the form
function form_new_line() { echo "<tr>"; }
function form_end_line() { echo "</tr>\n\n"; }
// auxiliary function, returns $name="value"
function html_attribute($name, $value) {
    return $name . '=' . htmlspecialchars($value) . ' ';
```

为了对 HTML 表单进行排版，使用了一个 HTML 表格。这个表格里的列数并不是给定的，所以必须自己注意在这个表格的每一行里给出个数相等的列（字段）。*formfunctions.php* 文件里的大部分函数都以一个 HTML 表格字段 (*<td> ... </td>*) 作为它们的输出，有几个函数可以通过一个可选的参数 (*colspan* 属性) 让自己的输出横跨好几个 HTML 表格字段。

form_label() 函数将以右对齐方式把文本写入一个 HTML 表格字段，所以很适合用来把 HTML 表单里的输入框和控件的标题/名字写出来。这个函数的可选参数 *emphasize* 决定着是否需要以彩色方式来显示给定的标题或名字。

```
// label cell
function form_label($caption, $emphasize =FALSE) {
    echo '<td align="right" class="myformtd">';
    if($emphasize )
        echo '<span class="red">', htmlspecialchars($caption), '</span>';
    else
        echo htmlspecialchars($caption);
    echo '</td>', "\n"; }
```

form_text() 函数将在 HTML 表单里创建一个文本框，它的 *\$name* 参数是文本框控件的名字（结果将是 *form[\$name]*），*\$value* 参数是默认值，*\$size* 和 *\$maxlength* 参数是文本输入框的显示长度和最大长度。

```
// create a one-cell text input field
function form_text($name, $value,
    $size=40, $maxlength=40, $colspan=1) {
    if($colspan>1)
        echo '<td class="myformtd" ' .
            html_attribute("colspan", $colspan), '>';
    else
        echo '<td class="myformtd"> ' .
            echo '<input class="mycontrol" ' .
                html_attribute("name", "form[$name]"),
                html_attribute("size", $size),
                html_attribute("maxlength", $maxlength);
    if($value)
        echo html_attribute("value", $value);
    echo '></td>', "\n";
}
```

*form_list()*函数将在 HTML 表单里创建一个下拉列表框及其选项清单，它的\$name 参数是下拉列表框控件的名字；\$rows 参数是一个二维数组，\$row[n][0]存放着下拉列表框里第 n 个选项的屏显文字，\$row[n][1]存放着第 n 个选项的编号值。变量\$selected 用来存放选中选项的编号值。

```
// create selection list (<option>-Tag)
function form_list($name, $rows, $selected=-1) {
    echo '<td class="myformtd">';
    echo '<select class="mycontrol" ' .
        html_attribute("name", "form[$name]"), '>', "\n";
    echo '<option value="none">(choose)</option>';
    foreach($rows as $row) {
        echo '<option ', html_attribute("value", $row[1]);
        if($selected==$row[1])
            echo 'selected="selected" ';
        $listentry = str_replace(" ", " ", htmlspecialchars($row[0]));
        echo ">$listentry</option>\n";
    }
    echo '</select></td>', "\n";
}
```

15.7.3 对表单数据进行合法性检查

在表单数据被传递到 titleform.php 脚本的时候，*validate_data()*函数将检查各有关字段（数组变量）里的数据是否正确。根据检查的结果，这个函数将返回 TRUE 或 FALSE，这个结果决定了是把数据存入数据库、还是把数据重新显示在表单里让用户改正。

出错消息将被传递到 *show_error_msg()*函数。这个函数来自 formfunctions.php 文件，它将返回一段红色的文本。

```
// example file titleform.php
function validate_data($formdata) {
    $result = TRUE;
    if(trim($formdata["title"])=="") {
        show_error_msg("You must specify a title!");
        $result = FALSE;
    }
    if(trim($formdata["authors"])=="") {
        show_error_msg("You must specify at least one title!");
        $result = FALSE;
    }
    $year = $formdata["publyear"];
    if(!empty($year) && (!is_numeric($year)) || $year<1000
        || $year>2100) {
        show_error_msg("Publishing year must be a four-digit number " .
            "(or empty).");
        $result = FALSE;
    }
}
```

```

if(!empty($edition) && (!is_numeric($edition)) || $edition<1
    || $edition>100) {
    show_error_msg("Edition must be a number <= 100 (or empty).");
    $result = FALSE;
}
return $result;
}

```

15.7.4 把表单数据存入数据库

从数据库的角度看，这个示例中最有趣的函数是 *save_data()*。它会把通过 *formdata[]* 数组传递来的数据分门别类地存入 *mylibrary* 数据库中的各有关数据表。根据那条图书记录是刚输入的新数据还是刚经过修改的现有数据，这个函数将分别执行 *INSERT* 或 *UPDATE* 命令来存储它。（具体到这个例子，*save_data()* 函数将根据 *\$formdata[titleID]* 是否已经存在来判断那是不是一条新的图书记录。）

注意，这里必须把各种特殊情况都考虑进去；比如说，在存储新作者、新出版公司、新图书语言之前也必须检查它们是否已经存在。除了这些事情，*save_data()* 函数还将在 *rel_title_author* 数据表里插入或修改一些记录，这个数据表里的记录对应着图书及其作者之间的关联关系。

在调用 *save_data()* 函数之前，将开始一个事务。如果在存储那些数据的过程中没有发生任何错误，*save_data()* 函数将返回 *TRUE*，否则将返回 *FALSE*。*save_data()* 函数的返回值决定着本个事务是得到确认还是被撤销。

```

// example file titleform.php
function save_data($formdata) {
    global $db;

    // search for and store authors
    $authors = explode(";", $formdata["authors"]);
    foreach($authors as $author) {
        $author = trim($author);
        $sql = "SELECT authID FROM authors WHERE authName = " .
            $db->sql_string($author);
        $rows = $db->queryObjectArray($sql);
        if($rows)
            // note IDs of existing authors
            $authIDs[] = $rows[0]->authID;
        else {
            // also note new author and ID
            $sql = "INSERT INTO authors (authName) " .
                "VALUES (" . $db->sql_string($author) . ")";
            if(!$db->execute($sql))
                return FALSE;
            $authIDs[] = $db->insertId();
        }
    }

    // store new publisher or note ID of selected publisher
    if($formdata["newpubl"]) {
        $sql = "SELECT publID FROM publishers WHERE publName = " .
            $db->sql_string($formdata["newpubl"]);
        $rows = $db->queryObjectArray($sql);
        if($rows)
            // publisher already exists
            $publID = $rows[0]->publID;
        else {
            // store new publisher
            $sql = "INSERT INTO publishers (publName) " .
                "VALUES (" . $db->sql_string($formdata["newpubl"]) . ")";
            if(!$db->execute($sql))
                return FALSE;
        }
    }
}

```

```

        return FALSE;
        $publID = $db->insertId();
    }
}

else
    // read publID from selection list
    $publID = $formdata["publisher"];

// analogous procedure for languages (language) --> $langID
...

// update existing title (UPDATE)
if(array_item($formdata, "titleID")) {
    $titleID = array_item($formdata, "titleID");
    $sql = "UPDATE titles SET ".
        "title=" . $db->sql_string($formdata["title"]) . ", ".
        "subtitle=" . $db->sql_string($formdata["subtitle"]) . ", ".
        "langID=" . ID_or_NULL($langID) . ", ".
        "publID=" . ID_or_NULL($publID) . ", ".
        "catID=" . ID_or_NULL($formdata["category"]) . ", ".
        "year=" . num_or_NULL($formdata["publyear"]) . ", ".
        "edition=" . num_or_NULL($formdata["edition"]) . " ".
        "WHERE titleID=$titleID";
    if(!$db->execute($sql))
        return FALSE;
    // delete existing rel_title_author entry
    $sql = "DELETE FROM rel_title_author WHERE titleID=$titleID";
    if(!$db->execute($sql))
        return FALSE;
}
else {
    // store new title (INSERT)
    $sql = "INSERT INTO titles (title, subtitle, ".
        "langID, publID, catID, year, edition) VALUES (" .
        $db->sql_string($formdata["title"]) . ", ".
        $db->sql_string($formdata["subtitle"]) . ", ".
        ID_or_NULL($langID) . ", ".
        ID_or_NULL($publID) . ", ".
        ID_or_NULL($formdata["category"]) . ", ".
        num_or_NULL($formdata["publyear"]). ", ".
        num_or_NULL($formdata["edition"]) . ")";
    if(!$db->execute($sql))
        return FALSE;
    $titleID = $db->insertId();
}

// create link between title and authors
foreach($authIDs as $authID) {
    $sql = "INSERT INTO rel_title_author (titleID, authID) " .
        "VALUES ($titleID, $authID)";
    if(!$db->execute($sql))
        return FALSE;
}

// output confirmation of success as link
echo "<p>Title ",
    build_href("titleform.php", "editID=$titleID",
        $formdata['title']),
    " has been saved.</p>\n";
return TRUE;
}

```

为了让用户知道数据已经成功地存入了数据库，*save_data()*函数还将输出一条确认消息，并在这条消

息里嵌入一个链接 (*titleform.php?editID=nnn*)。用户只须单击一下这个链接就可以再次看到自己刚输入的图书信息；如果发现其中有错，可以立刻加以修改。

与表单数据存储工作有关的辅助函数

在构造 SQL 命令的时候，还用到了辅助函数 *ID_or_NULL()* 和 *num_or_NULL()* 以及 *sql_string()* 方法：

```
// example file titleform.php
function num_or_NULL($n) {
    if(is_numeric($n)) return $n;
    else return 'NULL';
}
function ID_or_NULL($id) {
    if($id=="none")
        return 'NULL';
    else
        return $id;
}
// example file mydb.php
function escape($txt) {
    return trim($this->mysqli->escape_string($txt));
}
function sql_string($txt) {
    if(!$txt || trim($txt)==="")
        return 'NULL';
    else
        return "'" . $this->escape(trim($txt)) . "'";
}
```

在创建一个链接的时候，还用到了辅助函数 *build_href()*：

```
// example file mylibraryfunctions.php
function s($url, $query, $txt) {
    if($query)
        return "<a href=\"$url?\" . $query . "\">" .
            htmlspecialchars($txt) . "</a>";
    else
        return "<a href=\"$url\">" . htmlspecialchars($txt) . "</a>"; }
```

15.7.5 删 除一本图书

为慎重起见，在删除一条现有的图书记录之前，有关代码将要求用户对这一操作进行确认（如图 15-7 所示）。这个表单是由 *build_delete_form()* 函数生成和输出的。

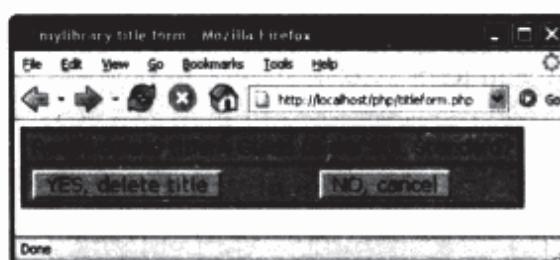


图 15-7 在删除一本图书之前要求用户进行确认

图书记录的删除操作是由 *delete_title()* 函数具体完成的。它不仅会把图书记录删掉，还会把 *rel_title_author* 数据表里已经没有任何用处的有关记录也一起删掉。

```
// example file titleform.php
function delete_title($formdata) {
    global $db;
    if((!$titleID = array_item($formdata, "titleID"))
        || !is_numeric($titleID))
        return FALSE;
    $sql = "DELETE FROM rel_title_author WHERE titleID=$titleID";
```

```

if (!$db->execute($sql))
    return FALSE;
$sql = "DELETE FROM titles WHERE titleID=$titleID LIMIT 1";
if (!$db->execute($sql))
    return FALSE;
echo "<p>One title has been deleted.</p>\n";
return TRUE;
}

```

15.7.6 值得改进的地方

如果有兴趣对这个示例做进一步改进，笔者建议从以下几个方面入手：

- **图书名称。**在存储一条新的图书记录时，示例代码没有对数据库是否已经收录了那本书的情况进行检查。这种检查看似简单，但实现起来相当复杂。只比较图书的名字是不够的，不同的图书完全有可能有着同样的书名，比如说 *Linux*。ISBN 国际书号可以作为唯一的标识符，但遗憾的是 HTML 示例表单没有提供这样的字段。不过，对大多数情况而言，用一条简单的查询命令检查两本图书的名字和作者是否都相同应该可以解决这个问题。
- **图书作者。***authors* 数据表里的作者姓名是按姓氏在前的方式存储的，因为只有这样才能使对作者姓名进行排序变得有意义。因此，*titleform.php* 脚本默认由用户输入的作者姓名也是这个顺序；也就是说，用户在输入作者姓名时应该输入（比如 *Kofler Michael* 而不是 *Michael Kofler*）。万一用户没有按照这条规定去做，就会导致一系列问题。
解决这个问题的办法之一是在 HTML 表单里为作者的姓氏和名字分别提供一个输入字段，但这会让为同一本书输入多位作者姓名的操作变得相当复杂。
另一个办法是在存储一个新的作者姓名之前检查一下 *authors* 数据表里是否已经存在姓氏和名字刚好与它相反的姓名。这个办法实现起来比较简单，效果也应该不错。
- **图书门类。**在图 15-6 所示的图书信息输入表单里没有办法定义一个新的图书门类，新门类目前只能通过 *categories.php* 脚本生成的 HTML 页面去管理。注意，如果在图书信息输入表单里增加这项功能，前面给出的许多示例代码都需要改写和增加，因为它们根本没有考虑如何对 *categories* 数据表进行刷新的问题。
从最终用户的角度考虑，如果能在图书信息输入表单里直接增加一个新的图书门类——就像在这个表单里增加一个新的出版公司或一种新的图书语言那样——当然很方便。从程序员的角度考虑，在 HTML 表单里提供一个下拉列表框让用户可以从现有图书门类当中挑选一个作为新门类的父门类也不算什么难题。但是，从信息安全的角度考虑，这么做的风险比较大：万一用户在定义一个新的图书门类时把事情弄乱了，后果可能会很严重。
- **删除图书。**示例中的 *delete_title()* 函数只删除那本图书在 *titles* 和 *rel_title_author* 数据表里的有关记录，这一点很值得改进：*authors* 和 *publishers* 数据表里的有关记录也应该删除——如果它们已经不再与任何图书记录相关联的话。

15.8 分页显示查询结果

几乎每一个动态网站都有一个搜索选项。如果有 n 个以上的搜索结果，比较常见的做法是只显示前 n 个，然后提供一些指向下一个页面的按钮或链接。本节将通过分析脚本文件 *find.php* 里代码向大家介绍一些分页显示查询结果的程序设计技巧。

find.php 程序可以用来查询 *mylibrary* 数据库所收录的图书和图书作者。这个程序很容易使用：如

果想搜索一本图书，只须给出图书名称和/或图书门类的前几个字符即可（如图 15-8 所示），这个程序将把同时满足这两个搜索条件的图书查找出来；如果想搜索一位作者，给出作者姓氏的前几个字符即可。

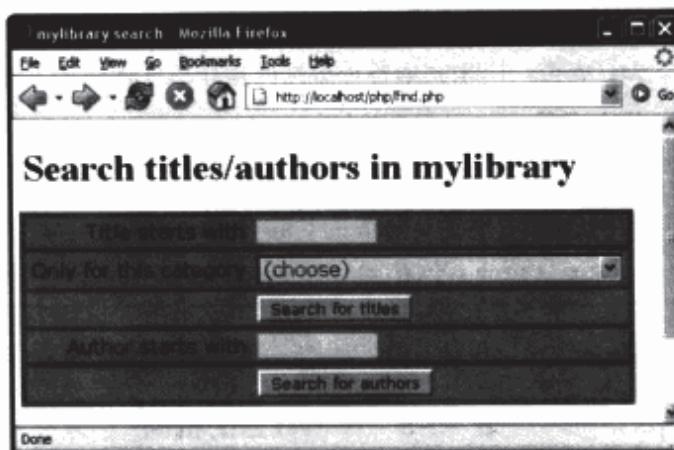


图 15-8 mylibrary 数据库的搜索表单

提示 这个示例没有提供全文检索功能。尽管如此，还是可以在这个例子里对搜索字符串出现在书名任意位置的图书进行搜索（即搜索字符串不必出现在书名的开头）：在搜索字符串的开头简单地加上一个百分号（%）作为前缀即可。比如说，搜索字符串%mysql不仅可以找到MySQL Cookbook，还可以找到PHP 5 and MySQL 5。

搜索结果（如图 15-9 所示）按字母表顺序排序并包含着指向 find.php 文件的链接，可以利用这些链接去查找一位给定作者写的全部图书、同属某个门类的全部图书以及关于一本特定图书的详细信息。如果搜索结果的个数大于\$pagesize（这个变量是在 find.php 文件的开头声明的），在本页搜索结果的下面就会出现指向下一页和上一页的链接。

图 15-9 图书搜索（左）和作者搜索（右）的搜索结果

15.8.1 代码结构

有关代码分别存放在表 15-4 所示的几个文件里。

表 15-4 代码所存放的文件

文件名	内 容
find.php	主控代码
mydb.php	<i>MyDb</i> 类, 用来完成各种数据库操作的(函数)方法
mylibraryfunctions.php	辅助函数
formfunctions.php	用来生成 HTML 表单的辅助函数
form.css	对 HTML 表单进行排版的 CSS 文件
resulttable.css	对搜索结果的 HTML 输出文档进行排版的 CSS 文件

表 15-5 是 HTML 表单里的数据发生变化时用来传输有关数据的两种机制。

表 15-5 传输数据的机制

机 制	数 �据
<i>POST</i>	数组形式的表单数据: <i>form["title"]</i> 、 <i>form["author"]</i> 、 <i>form["btnAuthor"]</i> 等
<i>REQUEST</i>	URL 参数, 它们的含义如下: <i>titleID</i> : 显示这本书; <i>titlePattern</i> : 显示与这个搜索模板匹配的所有图书; <i>authorID</i> : 显示这位作者; <i>authPattern</i> : 显示与这个搜索模板匹配的所有作者; <i>catID</i> : 显示属于这个门类的所有图书; <i>page</i> : 显示这一页

代码流

有关代码的主要部分都在各有关函数里, 接下来的几个小节将对其中一些比较重要的函数进行分析和介绍。下面是这些代码的执行流程控制部分。

首先, 对传递到这个页面的参数进行分析和提取。这里使用了来自 *mylibraryfunctions.php* 文件的 *array_item()* 函数, 已经在前面的有关内容里对它做过了介绍。*titlePattern* 和 *authPattern* 参数必须被编码在 URL 里, 这样其中的特殊字符(如果有)才不会引起任何问题。这个编码是用 *urldecode()* 函数解码的:

```
// example file find.php
$db = new MyDb(); // connection to MySQL
$pagesize = 5; // search results per page
// evaluate URL parameters
$titleID = array_item($_REQUEST, 'titleID');
$authID = array_item($_REQUEST, 'authID');
$catID = array_item($_REQUEST, 'catID');
$authPattern = urldecode(array_item($_REQUEST, 'authPattern'));
$titlePattern = urldecode(array_item($_REQUEST, 'titlePattern'));
$page = array_item($_REQUEST, 'page');
// validation control
if(!$page || $page<1 || !is_numeric($page)) $page=1;
elseif($page>100) $page=100;
if(!is_numeric($catID)) $catID = FALSE;
if(!is_numeric($authID)) $authID = FALSE;
if(!is_numeric($titleID)) $titleID = FALSE;
```

接下来, 需要把 PHP 的 *Magic Quotes* 功能自动进行的字符转义全部再转回来; 还需要给图书和作者的搜索模板加上一个百分号(%)作为后缀, 这个字符在稍后构造的 *SELECT* 命令里是必不可少的。

(*WHERE titles LIKE 'abc%* 将搜索以字符串 *abc* 开头的图书。)

```
// form data
$formdata = array_item($POST, "form");
if(is_array($formdata)) {
    // remove Magic Quotes
    if(get_magic_quotes_gpc())
        while($i = each($formdata))
            $formdata[$i[0]] = stripslashes($i[1]);
    $authPattern = array_item($formdata, "author") . "%";
    if(!array_item($formdata, "btnAuthor")) {
        $catID      = array_item($formdata, "category");
        $titlePattern = array_item($formdata, "title") . "%"; }
}
```

完成这些准备工作之后，就到了最关键的部分。如果参数 *titlePattern*、*titleID* 或 *catID* 里有参数值，*find.php* 脚本将依次调用以下 3 个函数完成对图书的搜索：*build_title_query()*，构造一条 *SELECT* 查询命令；*show_title()*，显示查询结果；*show_page_links()*，显示指向下一页和上一页的链接。作者搜索代码与图书搜索代码在执行流程方面完全一样。如果没有可以用来进行搜索的数据，*find.php* 脚本将调用 *build_form()* 函数显示一个空白表单：

```
if($titlePattern || $titleID || $catID) {
    // title search
    $sql = build_title_query($titlePattern, $titleID, $catID,
        $page, $pagesize);
    $rows = $db->queryObjectArray($sql);
    show_titles($rows, $pagesize);
    $query = "catID=$catID&titleID=$titleID&titlePattern=" .
        urlencode($titlePattern);
    show_page_links($page, $pagesize, sizeof($rows), $query);
    echo '<p><a href="find.php">Back to search form</a></p>', "\n";
}
elseif($authPattern || $authID) {
    // author search, analogous code
    ...
}
else {
    // display search form
    echo "<h1>Search titles/authors in mylibrary</h1>\n";
    build_form(); }
```

15.8.2 对图书作者进行搜索

用来搜索图书作者的 SQL 命令是在 *build-author_query()* 函数里生成的。如果 *authorID* 参数包含着一位作者的 ID 编号，将得到一条简单的 SQL 命令，如下所示：

```
SELECT authID, authName FROM authors WHERE authID=...
```

从另一个方面讲，如果搜索是为了查找与某个模板匹配的作者，最终生成的 SQL 代码就会比较复杂：

```
SELECT authID, authName FROM authors
WHERE authName LIKE 'abc%'
ORDER BY authName
LIMIT 20, 11
```

这里的 *LIMIT* 子句需要特别解释一下，它的作用是把结果记录的显示个数限制在一个给定的数字。具体到上面这条 *SELECT* 命令，*LIMIT 20, 11* 的意思是这条 *SELECT* 命令应该返回从第 20 条到第 30 条结果记录（如果 *\$size=10*，这就是第 3 页）。

用户可能会感到奇怪：既然 *\$size=10*，为什么要查找 11 位作者？这么做的理由是为了便于判断是

否还有更多的查询结果。（既然只需要显示 10 位作者，那么如果这条查询命令返回了 11 位作者，就说明后面还有更多的查询结果，而这意味着我们应该调用 *show_page_links()* 函数去创建一个指向下一页的链接。）

```
// example file find.php
function build_author_query($pattern, $authID, $page, $size) {
    global $db;
    $sql = "SELECT authID, authName FROM authors ";
    if($authID)
        return $sql . "WHERE authID = $authID";
    else
        return $sql . "WHERE authName LIKE " . $db->sql_string($pattern) .
            " ORDER BY authNAME ".
            "LIMIT " . ((($page-1) * $size) . "," . ($size + 1));
}
```

15.8.3 对图书进行搜索

build_title_query() 函数的工作原理与 *build-author_query()* 函数几乎完全一样。它们之间的唯一区别是后者在某些情况下需要同时考虑两个搜索条件，也就是图书门类和图书搜索模板。因此，如果用户同时给出了这两个条件，它们在最终生成的 *SELECT* 命令里将以 *WHERE cond1 AND cond2* 的形式组合在一起。如果用户只给出了图书门类或图书搜索模板之一，另一个搜索条件将被简单地设置为 *TRUE*。

在涉及图书门类的搜索中，只使用条件表达式 *catID=n* 是不够的：给定门类可能会有下级门类。因此，需要调用 *subcategory_list()* 函数为给定门类生成一份它的全体下级门类的清单。（这一点其实很好理解：如果正在搜索的是 *Computer books*（计算机图书）门类，那肯定希望还能看到对 *Database*（数据库）、*Programming*（程序设计）等门类的搜索结果。）*subcategory_list()* 函数来自 *mylibraryfunctions.php* 文件，我们将在 15.9 节里对它进行分析。

```
// example file find.php
function build_title_query($pattern, $titleID, $catID,
                           $page, $size) {
    ... analogous to build_author_query
    if($catID && $catID!="none") {
        $catsql = "SELECT catID, parentCatID FROM categories";
        $rows = $db->queryObjectArray($catsql);
        foreach($rows as $row)
            $subcats[$row->parentCatID][] = $row->catID;
        $cond1 = "catID IN (" .
            subcategory_list($subcats, $catID) . ")";
    }
    else
        $cond1 = "TRUE";
    if($pattern)
        $cond2 = "title LIKE " . $db->sql_string($pattern) . " ";
    else
        $cond2 = "TRUE";
    $sql .= "WHERE " . $cond1 . " AND " . $cond2 .
        " ORDER BY title ";
    ... and so on, as in build_author_query
}
```

15.8.4 显示搜索结果

把满足查询条件的作者全部查找出来之后，这些作者的 *authID* 和 *authName* 将以一个数组的形式被传递给 *show_authors()* 函数。类似地，把满足查询条件的图书全部查找出来之后，这些图书的 *titleID* 和 *titleName* 也将以一个数组的形式被传递给 *show_titles()* 函数。*show_authors()* 或 *show_titles()*

函数会在它们的内部用一两条 SQL 查询命令把这些作者或图书的所有其他信息查询出来。比如说，为了把同一位作者写的其他图书查找出来，*show_authors()* 函数会在它的内部构造出一条如下所示的查询命令：

```
SELECT title, rel_title_author.titleID, authID
  FROM titles, rel_title_author
 WHERE titles.titleID = rel_title_author.titleID
   AND rel_title_author.authID IN (1, 2, 3, ...)
 ORDER BY title
```

把作者或图书的所有信息全都查询出来之后，剩下来的事情就是如何把这些信息以一种整齐易读的方式输出到 HTML 结果文档。这个示例使用了一个 HTML 表格来排版输出内容，这个 HTML 表格的结构和格式由 CSS 文件 *resulttable.css* 控制。下面是 *show_authors()* 函数的代码：

```
// example file find.php
function show_authors($authors, $pagesize) {
    global $db;
    echo "<h1>Search results</h1>\n";
    if(!$authors) {
        echo "<p>Sorry, no authors found.</p>\n";
        return;
    }
    // create string with authIDs
    $items = min($pagesize, sizeof($authors));
    for($i=0; $items; $i++)
        if($i==0)
            $authIDs = $authors[$i]->authID;
        else
            $authIDs .= "," . $authors[$i]->authID;
    // return titles that these authors have written ermitteln
    $sql = "SELECT title, rel_title_author.titleID, authID "
        . "FROM titles, rel_title_author "
        . "WHERE titles.titleID = rel_title_author.titleID "
        . "AND rel_title_author.authID IN ($authIDs) "
        . "ORDER BY title";
    $rows = $db->queryObjectArray($sql);

    // display all authors
    echo '<table class="resulttable">', "\n";
    for($i=0; $items; $i++) {
        echo td1("Author:", "td1head"),
            td2($authors[$i]->authName, "td2head");
        // show all titles for each author
        $titles=0;
        foreach($rows as $row)
            if($authors[$i]->authID == $row->authID) {
                if($titles==0)
                    echo td1("Titles[s]:");
                else
                    echo td1("");
                echo td2url($row->title, "find.php?titleID=$row->titleID");
                $titles++;
            }
        // blank line before next title
        echo td1("", "tdinvisible"), td2asis("&nbsp;", "tdinvisible");
    }
    echo "</table>\n";
}
```

HTML 表格的各个字段是用 *td1()* 和 *td2xxx()* 等几个辅助函数输出的：

```
function td1($txt, $class="td1") {
    echo "<tr><td class=\"$class\">",
```

```

htmlspecialchars($txt), "</td>\n"; }
function td2($txt, $class="td2") {
    echo "<td class=\"$class\">",
        htmlspecialchars($txt), "</td></tr>\n"; }
function td2asis($txt, $class="td2") {
    echo "<td class=\"$class\">$txt</td></tr>\n"; }
...

```

*show_titles()*函数的代码要比*show_authors()*函数长很多，但那只不过是因为图书需要输出的信息比较多而已。既然这两个函数在工作原理方面完全一样，就不在这里再对*show_titles()*函数进行讨论了。

15.8.5 指向其他结果页面的链接

在分页显示查询结果时候，每一个结果页面都应该有一些指向其他结果页面的链接。这种链接有两种比较常见的显示方式：

- 先用一条 *SELECT COUNT(*) ... WHERE ...* 形式的命令统计出查询结果里的记录总数，再计算出总共需要多少个 HTML 结果页面来显示它们，然后在每一个结果页面上把指向其他结果页面的链接全部创建并显示出来。
- 在每一个结果页面上只显示两个分别指向下一页和上一页的链接——当然，第一页没有上一页、最后一页没有下一页。这么一来，只须在创建一个结果页面的时候顺便检查一下后面是否还有结果记录（比如像我们刚才提到的那样使用 *LIMIT pagesize+1* 而不是使用 *LIMIT pagesize*）就可以知道是否需要创建一个指向下一页的链接。这个办法显然要简单一些，如果数据库不是很大，它还是很值得考虑的。

*show_page_links()*函数采用了第 2 种思路。这个函数有 4 个参数：*\$page*，当前页面的编号；*\$pagesize*，每页的结果记录个数；*\$result*，最近一次查询返回的结果记录的个数；*\$query*，需要通过指向下一页或上一页的链接来传递的查询参数表（比如 *authPattern=abc*）。

*show_page_links()*函数将先创建一个指向 上一页 的链接（如果 *\$page > 1*），再创建一个指向 下一页 的链接（如果 *\$result > \$pagesize* 的话）。这些链接的具体创建工作由来自 *mylibraryfunctions.php* 文件的辅助函数 *build_href()*负责完成。

```

// example file find.php
function show_page_links($page, $pagesize, $results, $query) {
    if(($page==1 && $results<=$pagesize) || $results==0)
        return;
    echo "<p>Goto page: ";
    if($page>1) {
        for($i=1; $i<$page; $i++)
            echo build_href("find.php", $query . "&page=$i", $i), " ";
        echo "$page ";
    }
    if($results>$pagesize) {
        $nextpage = $page + 1;
        echo build_href("find.php", $query . "&page=$nextpage",
            $nextpage);
    }
    echo "</p>\n";
}

```

15.9 处理层次化数据

用数据表来表示层次关系是一件相当简单的事情：多创建一个数据列来存放父记录的 ID 编号值（或是其他可以起到 ID 作用的字段值）就足够了。比如说，*mylibrary* 数据库里的 *categories* 数据表就采用了这个办法来管理图书门类之间的层次关系（请参见第 8 章）。类似的应用示例还包括公司员工

之间的层次关系（最顶端是老板、然后是部门经理、再往下是小组长等）和家族谱系（最顶端是亚当和夏娃等）。

不过，虽然这种层次关系很容易构造，但保存层次关系的数据表却相当难以处理。如果想把层次关系里某个结点的上级或下级元素全部查找出来，就必须学会使用递归函数（自己调用自己的函数）。

本节将以 *mylibrary* 数据库里的 *categories* 数据表为例介绍一些编程技巧，只要能够活学活用这些编程技巧，就一定可以管理好自己的层次化数据。

本节将要讨论的函数大都来自 *categories.php* 文件。这个脚本生成的 HTML 页面（如图 15-10 所示）可以帮助用户管理图书门类。它很容易使用：只须单击就可以删除一个现有的图书门类或是创建出一个新的下级图书门类（参见图 15-12）。

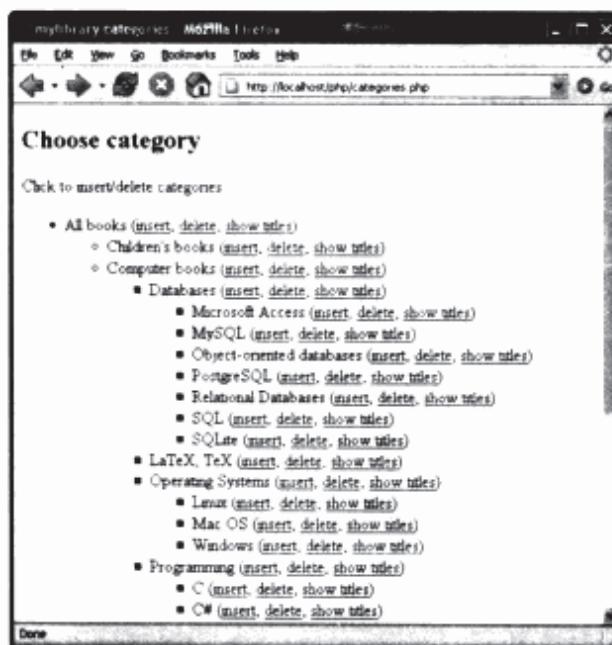


图 15-10 管理图书门类

在图 15-10 里，当删除一个图书门类时，它所有的下级门类将同时被删除（当然，前提是这些门类已经不再有人使用了）。图中的 *show titles* 链接分别指向着一个 *find.php?catID= ...* 页面（参见 15.8 节里的有关内容），那些页面将把与给定门类及其所有下级门类相关联的图书全都查找出来。

从 MySQL 5.0 开始，还可以使用 SP (stored procedure，存储过程) 来处理那些其内容数据有着层次化关系的数据表，这方面的示例可以在本书第 13 章找到。

15.9.1 代码结构

有关代码分别存放在如表 15-6 所示的几个文件里。

表 15-6 代码及其存放的文件

文件名	内 容
categories.php	主控代码
mydb.php	<i>MyDb</i> 类，用来完成各种数据库操作的（函数）方法
mylibraryfunctions.php	辅助函数

表 15-7 是 HTML 表单里的数据发生变化时用来传输有关数据的两种机制。

表 15-7 传输数据的机制

机 制	数 �据
<i>POST</i>	来自输入表单 (<i>submitbutton</i> 、 <i>subcategories</i>) 的新图书门类
<i>REQUEST</i>	URL 参数，它们的含义如下： <i>insertID</i> : 把新门类插入到这个门类之下作为子门类； <i>deleteID</i> : 删除这个图书门类

代码流

有关代码的主要部分都在各有关函数里，接下来的几个小节将对其中一些比较重要的函数进行分析和介绍。下面是这些代码的执行流程控制部分。

```
// example file categories.php
$db = new MyDb(); // connection to MySQL
$insertID = array_item($_REQUEST, 'insertID');
$deleteID = array_item($_REQUEST, 'deleteID');
$submitbutton = array_item($_POST, 'submitbutton');
$subcategories = array_item($_POST, 'subcategories');
if(get_magic_quotes_gpc()) // no magic quotes
    $subcategories = stripslashes($subcategories);
```

delete_categories() 和 *insert_new_categories()* 函数都必须用一个事务来调用执行，它们的返回值决定事务是得到确认还是必须撤销。这么做是为了确保最终结果只存在以下两种可能：一是有关的删除或插入操作在没有任何错误的情况下全部执行成功；二是它们全部都不执行，图书门类清单保持不变。

```
// delete category
if($deleteID) {
    $sql = "SELECT COUNT(*) FROM categories WHERE catID='$deleteID'";
    if($db->querySingleItem($sql)==1) {
        $db->execute("START TRANSACTION");
        if(delete_category($deleteID)==-1)
            $db->execute("ROLLBACK");
        else
            $db->execute("COMMIT"); }
// store or insert new categories
if($insertID) {
    $sql = "SELECT COUNT(*) FROM categories WHERE catID='$insertID'";
    $n = $db->querySingleItem($sql); }
if($insertID && $n==1) {
    echo "<h2>Insert new categories</h2>\n";
    // store new categories
    if($subcategories) {
        $db->execute("START TRANSACTION");
        if(insert_new_categories($insertID, $subcategories))
            $db->execute("COMMIT");
        else
            $db->execute("ROLLBACK"); }
    // input form
    print_category_entry_form($insertID); }
// display complete list of all categories
else {
    echo "<h2>Choose category</h2>\n";
    echo "<p>Click to insert/delete categories.</p>\n";
    ... details in the next section
    print_categories($rows, NULL); }
```

15.9.2 显示图书门类树

要想得到如图 15-10 所示的图书门类树，必须先进行一次 SQL 查询，把所有的图书门类查找出来。接下来，利用这些数据创建两个数组。创建这两个数组的目的是为了避免过一会儿在 *print_categories()* 函数里还得再次到数据库里去搜索这两个数组中的元素。有了这两个数组，在 *print_categories()* 函数里就不必用一个循环去查找所有满足条件 *parentID==n* 的图书门类了，因为它们可以直接从数组元素 *subcats[catID]* 里提取出来。（更准确地说，这其实是把对数据库进行搜索转化为 PHP 内部对某个特定的数组元素进行搜索而已：当访问数组元素 *\$array[key]* 的时候，PHP 将使用关键字（下标）*key* 去查找 *\$array[]* 数组里的对应元素。PHP 进行这种搜索的效率非常高。）

\$catNames[] 数组的用途从它的名字上就能看出来：数组元素 *\$catNames[catID]* 里存放着 ID 编号等于 *catID* 的图书门类的名字。

相对而言，*\$subcats[]* 数组的用途就不那么容易看出来了：数组元素 *\$subcats[catID]* 的值还是一个数组，那个数组保存着 ID 编号等于 *catID* 的图书门类的所有下级门类的 ID 编号。如果把利用 *catID* 值去查找 *parentCatID* 值看做是正向逻辑，*\$subcats[]* 数组里的元素就是按照与此刚好相反的逆向逻辑生成的：在一个 *foreach* 循环里，各个图书门类的 *parentCatID* 将被用做键字（下标），它们的 *catID* 值将成为数组元素的取值。幸运的是 *parentCatID==NULL* 的特殊情况不需要特殊处理——PHP 允许使用 *NULL* 值作为数组的键字（下标）。顺便说一句，数组元素 *\$subcats[n]* 所包含的下级门类按字母表顺序排序，这是因为变量 *\$row* 里的原始数据是我们用 *ORDER BY catName* 子句从数据库里查出来的。

PHP 语法 *\$array[] = abc* 的含义是：把 *abc* 作为一个新的数组元素插入 *\$array[]* 数组；如果 *\$array[]* 数组此前尚不存在，则创建一个新数组并把 *abc* 作为新数组的第一个元素。

```
// preparations for calling print_categories
// determine all categories
$sql = "SELECT catName, catID, parentCatID "
      . "FROM categories ORDER BY catName";
$rows = $db->queryObjectArray($sql);
// create fields subcats and catNames
foreach($rows as $row) {
    $subcats[$row->parentCatID][] = $row->catID;
    $catNames[$row->catID] = $row->catName;
}
// display categories beginning with the root element
print_categories($subcats[NULL], $subcats, $catNames);
```

下面是 *print_categories()* 函数的代码。这个函数的第一个参数是某个数组的根元素（具体到这个例子，就是 *\$subcats[NULL]*），它将依次遍历这个数组里的所有元素去查找和输出有关信息（具体到这个例子，就是 *\$catNames[]* 数组里的图书门类名称）。如果某个图书门类还有下级门类（这项检查由代码中的 *array_key_exists(\$catID, \$subcats)* 函数调用负责完成），*print_categories()* 函数就将以 *\$subcats[\$catID]* 作为参数去递归调用它自己。

```
function print_categories($catIDs, $subcats, $catNames) {
    echo "<ul>";
    foreach($catIDs as $catID) {
        printf("<li>%s (%s, %s, %s)</li>\n",
               htmlspecialchars($catNames[$catID]),
               build_href("categories.php", "insertID=$catID", "insert"),
               build_href("categories.php", "deleteID=$catID", "delete"),
               build_href("find.php", "catID=$catID", "show titles"));
        if(array_key_exists($catID, $subcats))
            print_categories($subcats[$catID], $subcats, $catNames);
    }
}
```

```

    }
    echo "</ul>\n";
}

```

HTML结果页面里的图书门类下拉列表

由 titleform.php 和 find.php 脚本生成的 HTML 结果页面里都有一个供用户选择图书门类的下拉列表框，里面的图书门类条目都按照它们在层次结构中的级别进行了缩进（如图 15-11 所示）。

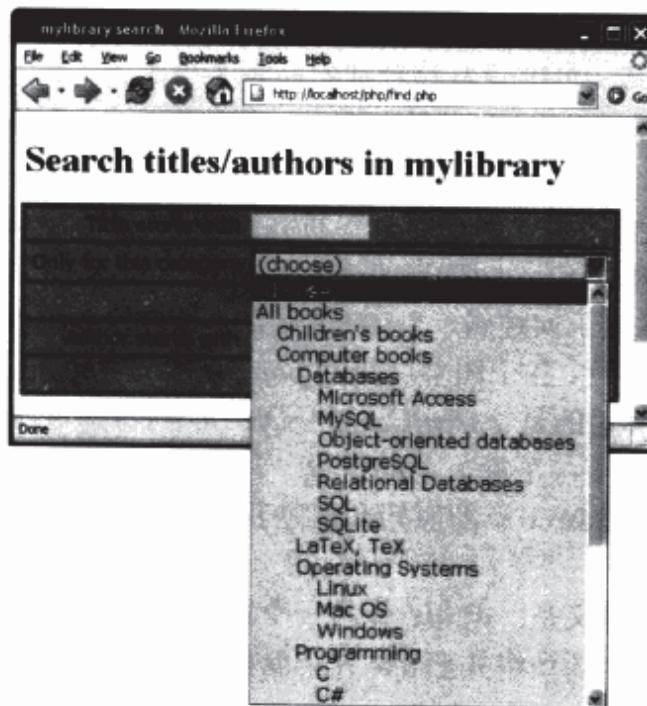


图 15-11 下拉列表框里的图书门类

这样的下拉选择框是 titleform.php 和 find.php 脚本调用 *build_category_array()*（来自 mylibraryfunctions.php 文件）和 *form_list()* 函数（来自 formfunctions.php 文件）创建的。*build_category_array()* 函数将返回一个二维数组，每个数组元素包含两项数据：图书门类的名称和它的 *catID* 编号值；*form_list()* 函数将使用这些数据创建下拉列表框。

build_category_array() 函数的结构与刚才介绍的 *print_categories()* 函数很相似，而且也需要同样的参数（尤其是 \$catNames[] 和 \$subcats[] 数组）。这里的新东西是可选的第 4 个参数 \$indent，它负责给出当前的缩进级别。

build_category_array() 函数将先把第一个参数里的图书门类插入 \$tmp[] 数组，然后检查这个门类是否还有下级门类，如果有，则递归调用自己去处理那些下级门类（见黑体字部分）。

\$tmp[] 是一个静态数组，所以它的值在 *build_category_array()* 函数的递归调用里不会丢失。（一般来说，函数里的变量都应该声明为局部变量，即每一次函数调用都有它自己独享的变量。）在下面的代码里，在 *build_category_array()* 函数以 \$indent=0 为参数的首次调用结束之后，\$tmp[] 数组将被清零。注意，这里不能使用普通的 *unset()* 函数来清零 \$tmp[] 数组，因为它不能用于静态变量。（这是 PHP 语言里的一条规定。）

为了对清单里的条目进行正确的缩进，*build_category_array()* 函数需要通过它的第 4 个参数 *indent* 来控制当前缩进级别，清单里的每一个条目将向右缩进 *indent*3* 个空格。在 *form_list()* 函数里，这些前缀空格将被替换为 HTML 特殊字符 （即所谓的“硬空格”；硬空格与普通空格的区别是可以在

Web 浏览器窗口里得到显示而不是被忽略不计)。

```
// example file mylibraryfunctions.php
function build_category_array($catIDs, $subcats, $catNames, $indent=0) {
    static $tmp;
    if($indent==0)
        $tmp = FALSE; // clear tmp
    foreach($catIDs as $catID) {
        $pair[0] = str_repeat(" ", $indent*3) . $catNames[$catID];
        $pair[1] = $catID;
        $tmp[] = $pair;
        if(array_key_exists($catID, $subcats))
            build_category_array($subcats[$catID], $subcats, $catNames, $indent+1);
    }
    if($indent==0)
        return $tmp;
}
```

build_category_array() 函数还将在后面的“速度优化”一节里再次出现。

15.9.3 插入一个或多个新图书门类

当在图 15-10 所示的 HTML 页面上单击了某个 *insert* (插入) 链接时, *build_category_entry_form()* 函数将最先被调用, 它将返回一个“缩水”了的图书门类清单, 里面只有选中的图书门类以及它的直接上级和下级门类 (如图 15-12 所示)。定义一个新图书门类的具体操作是通过一个小 HTML 表单完成的, 这个表单本身由一个文本框和一个按钮构成。因为篇幅的关系, 就不在这里对这个函数做详细的介绍了, 因为它在编程技巧方面没有用到什么新东西。

把新图书门类插入数据库的工作由 *insert_new_categories()* 函数负责具体完成, 这个函数有两个参数: 第一个是某现有图书门类的 *catID* 编号值, 新门类将成为这个门类的下级门类; 第二个是一个字符串, 其内容是新门类的名字 (如果一次定义了多个新门类, 它们的名字要用分号隔开)。

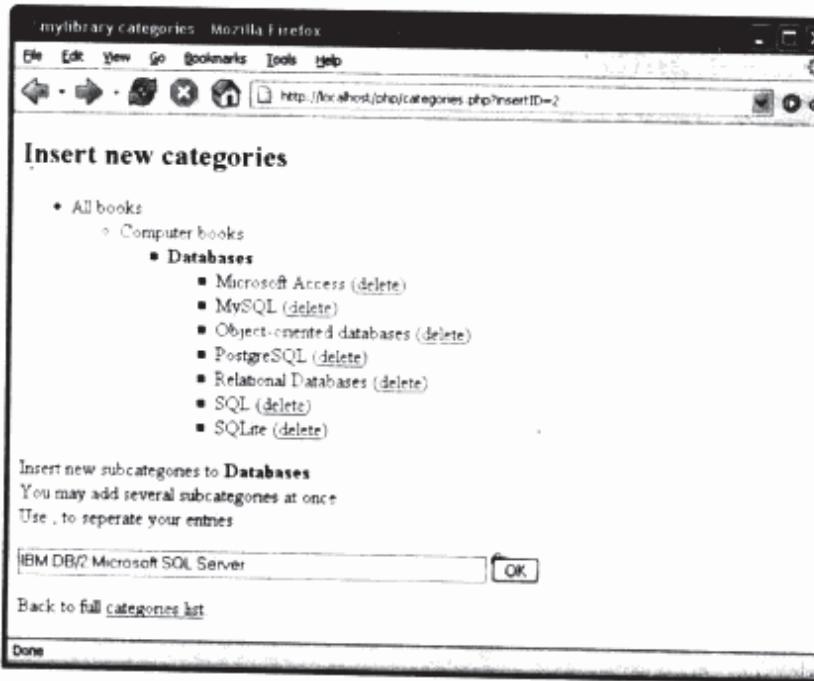


图 15-12 定义新的图书门类

insert_new_categories() 函数将先调用 *explode()* 函数把字符串分拆为一些数组元素 (每个数组元素存放着一个新门类的名字), 然后再为每一个数组元素调用一次 *insert_new_category()* 函数。只要在存储某个新门类时发生错误 (某次 *insert_new_category()* 函数调用的返回值是 -1), 整个

*insert_new_categories()*函数将立刻退出执行并返回 *FALSE*（这个返回值决定着这个插入新图书门类的事务是被确认还是被撤销）。

```
// example file categories.php
function insert_new_categories($insertID, $subcategories) {
    global $db;
    $subcatarray = explode(";", $subcategories);
    $count = 0;
    foreach($subcatarray as $newcatname) {
        $result = insert_new_category($insertID, trim($newcatname));
        if($result == -1) {
            echo "<p>Sorry, an error happened. Nothing was saved.</p>\n";
            return FALSE;
        }
        elseif($result)
            $count++;
    }
    if($count)
        if($count==1)
            echo "<p>One new category has been inserted.</p>\n";
        else
            echo "<p>$count new categories have been inserted.</p>\n";
    return TRUE;
}
```

*insert_new_category()*函数负责把一个新的图书门类存入数据库。在此之前，它会先去检查一下数据库里是否已经收录了这个“新”门类。这个函数有3种返回值：1（新门类存储成功）、0（新门类已经存在）、-1（SQL命令或数据库有错误）

```
function insert_new_category($insertID, $newcatName) {
    global $db;
    // newcatName is empty
    if(!$newcatName) return 0;
    $newcatName = $db->sql_string($newcatName);
    // newcatName already exists
    $sql = "SELECT COUNT(*) FROM categories " .
        "WHERE parentCatID=$insertID " .
        "AND catName=$newcatName";
    if($db->querySingleItem($sql)>0) {
        return 0;
    }
    $sql = "INSERT INTO categories (catName, parentCatID) " .
        "VALUES ($newcatName, $insertID)";
    if($db->execute($sql))
        return 1;
    else
        return -1;
}
```

15.9.4 删除一个图书门类及其下级门类

一个给定的图书门类只有在它的所有下级门类都被删除之后才允许删除，这意味着 *delete_category()* 函数需要进行递归调用（见代码清单中的黑体字部分）才能完成任务。在删除动作发生之前还必须再满足一个条件：将被删除的图书门类里已经不再有任何图书。

```
// example file categories.php
function delete_category($catID) {
    global $db;
    // search out and delete subcategories
    $sql = "SELECT catID FROM categories " .
        "WHERE parentCatID='$catID'";
    if($rows = $db->queryObjectArray($sql)) {
        $deletedRows = 0;
```

```

foreach($rows as $row) {
    $result = delete_category($row->catID);
    if($result == -1)
        return -1;
    else
        $deletedRows++;
}
// if not all subcategories can be deleted,
// then do not delete this category
if($deletedRows != count($rows))
    return 0;
}
// if category is being used, do not delete
$sql = "SELECT COUNT(*) FROM titles WHERE catID='$catID'";
if($n = $db->querySingleItem($sql)>0) {
    $sql = "SELECT catName FROM categories WHERE catID='$catID'";
    $catname = $db->querySingleItem($sql);
    printf("<br />Category %s is used in %d titles. " .
        "You cannot delete it.\n", $catname, $n);
    return 0;
}
// delete category
$sql = "DELETE FROM categories WHERE catID='$catID' LIMIT 1";
if($db->execute($sql))
    return 1;
else
    return -1;
}

```

15.9.5 搜索上级图书门类

在 `find.php` 脚本生成的 HTML 结果页面里，不仅有指向当前图书门类的链接，还有指向当前图书门类所有上级门类¹的链接（如图 15-13 所示）。这为相关图书的搜索操作提供了极大的便利。

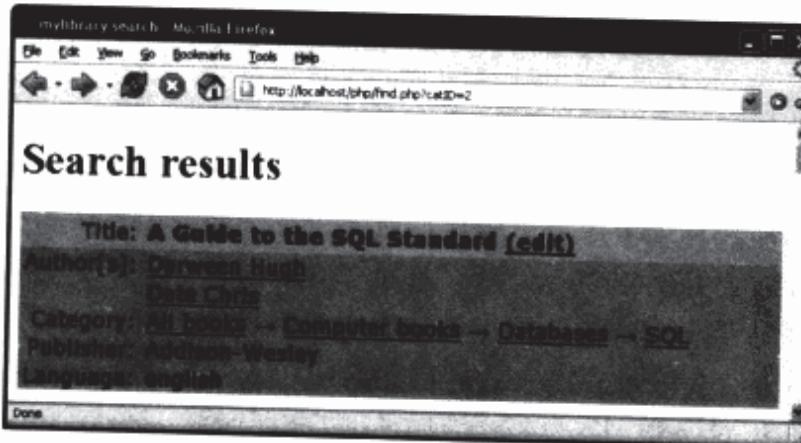


图 15-13 指向当前图书门类及其所有上级门类的链接

这里最让人感兴趣的当然是如何对上级图书门类进行搜索。具体到这个例子，`show_titles()`函数是调用 `build_cat_string()` 函数来完成这一任务的，后者将返回一个字符串，它的内容是所有应该有的链接。`build_cat_string()` 函数有 3 个参数：与当前图书门类相关联的 `$catNames[]` 和 `$catParents[]` 数组以及当前图书门类的 `CatID` 编号值。

在调用 `build_cat_string()` 函数之前，`show_titles()` 函数将先对 `$catNames[]` 和 `$catParents[]` 数组进行

1. 原文这里以及图 15-13 的标题出现了印刷错误，把“上级”写成了“下级”。——译者注

初始化：找出当前图书门类的所有上级门类，把它们的名字存入`$catNames[]`数组，把它们的`parentID`编号值存入`$catParents[]`数组。

```
// example file find.php
function show_titles($titles, $pagesize) {
    ...
    // determine all categories
    $sql = "SELECT catName, catID, parentCatID FROM categories";
    $rows = $db->queryObjectArray($sql);
    // form associative fields for category names and parentCatID
    foreach($rows as $cat) {
        $catNames[$cat->catID] = $cat->catName;
        $catParents[$cat->catID] = $cat->parentCatID; }

    ...
    // loop over all titles
    foreach($titles as $title) {
        ... display various title information
        // display category and subcategories
        if($title->catID)
            echo td1("Category:"), td2asis(
                build_cat_string($catNames, $catParents, $title->catID));
    }
}
```

`build_cat_string()`函数将首先为当前`catID`编号值所对应的图书门类(即当前图书门类)创建一个链接，然后通过一个循环去搜索当前门类的所有上级门类，并为它们也创建一个链接。链接之间的右箭头符号由特殊字符`→`提供。¹与本书许多其他示例中的做法一样，对链接进行必要的排版(排版结果的格式是` txt `)以及把它们输出到HTML结果页面的工作是由来自`mylibraryfunctions.php`文件的`build_href()`函数完成的。

```
function build_cat_string($catNames, $catParents, $catID) {
    $tmp = build_href("find.php", "catID=$catID", $catNames[$catID]);
    while($catParents[$catID] != NULL) {
        $catID = $catParents[$catID];
        $tmp = build_href("find.php", "catID=$catID", $catNames[$catID]) .
            " &rarr; ". $tmp; }
    return $tmp;
}
```

15.9.6 搜索下级图书门类

在`find.php`脚本生成的搜索表单里，用户可以选定一个图书门类作为搜索条件。如果某个用户选择了`Database`(数据库)门类作为搜索条件，那他肯定希望自己还可以在搜索结果里看到`MySQL`、`SQL`等下级门类里的图书。因此，在`SELECT`查询命令里只使用`catID=nnn`作为搜索条件是不够的。必须先把它所有下级门类的`ID`编号查出来，再把它们用在`SELECT`查询条件里，即最终的搜索条件表达式应该是`catID IN (n1, n2, n3, ...)`。

这样的图书门类列表可以用`subcategory_list()`函数返回，这个函数有两个参数：起始图书门类和关联数组`$subcats[]`数组，数组元素`$subcats[$catID]`本身又是一个数组，该数组里存放着起始图书门类的所有下级门类的`catID`编号值。下面是`build_title_query()`函数里负责构造`$subcats[]`数组和调用`subcategory_list()`函数的有关代码：

```
// example file find.php
function build_title_query(...) {
    ...
    $catsql = "SELECT catID, parentCatID FROM categories";
    $rows = $db->queryObjectArray($catsql);
    foreach($rows as $row)
```

```

$subcats[$row->parentCatID][] = $row->catID;
$cond1 = "catID IN (" .
subcategory_list($subcats, $catID) . ")";

```

subcategory_list() 函数将先把通过 *\$catID* 参数传递来的 *catID* 编号值保存在一个临时变量里，然后检查这个图书门类是否还有下级门类。如果有（即 *\$subcats[]* 数组里存在着一个以当前图书门类的 *catID* 编号值为下标的元素），*subcategory_list()* 函数将进行递归调用去搜索所有的下级门类，并把找到的 *catID* 编号值依次追加到变量 *\$lst*。

```

function subcategory_list($subcats, $catID) {
    $lst = $catID;
    if(array_key_exists($catID, $subcats))
        foreach($subcats[$catID] as $subCatID)
            $lst .= ", " . s($subcats, $subCatID);
    return $lst;
}

```

15.10 速度优化

当在自己的本地系统上测试 PHP 脚本时，它们的执行速度往往让人感到非常满意。可是，当把整个项目部署到因特网上以后，那些脚本的执行速度又往往会突然变得相当慢：大多数 ISP 都会在同一台服务器主机上运行多个网站，对这些网站的同时访问量越大，它们当中的每一个网站就都会变得越慢。

此外，有很多数据库应用程序还会随着数据库体积的增加而变得更慢。如果代码没有针对比较大的数据集合（内存、CPU 周期、数据库读写操作、网络数据传输量等）进行过优化，情况就更会如此。

基于上面提到的这些理由，即便是一些很简单的项目，也需要程序员尽最大的努力去写出经过深思熟虑和高效率的代码。本节将介绍一些与 PHP 编程有关的速度优化技巧和例子。对 MySQL 服务器的配置（如查询缓存区等）进行优化以获得最高效率的一些思路和办法将在第 14 章讨论。

15.10.1 提高代码执行效率的基本原则

- **数据库设计方案。**要想建立一个高效率的动态网站，良好的数据库设计方案是最重要的因素。如果数据库的框架结构达不到要求，对程序代码进行的所有优化都将劳而无功。
从追求效率的角度看，只有按照以下几条原则设计出来的数据库才能有更高的效率。（注意，按照以下原则优化出来的数据库不一定符合在本书第 8 章介绍的数据库设计规则。比如说，为了追求效率，数据库可能需要包含冗余的数据。数据库领域的理论家们不会喜欢这样的数据库，但在实际工作中，以数据冗余为代价往往可以在速度方面获得明显的回报。）
数据库设计工作中的一个重要方面是正确地创建和使用各种索引。需要频繁对其内容进行搜索的数据列（*WHERE column = ...*）都应该配有一个适当的索引。但索引并不是越多越好：它们的存在会增加数据表的内存占用量并减慢数据修改操作的速度。
- **用来生成每个 HTML 页面的 SQL 查询命令越少越好。**从 PHP 解释器发往 MySQL 服务器的每一条 SQL 命令都会消耗相当客观的资源：传输一条命令、在服务器上执行一条命令、传输一条命令的执行结果。因此，作为一项基本原则，生成一个 HTML 页面——或者从更广泛的意义上讲，完成一项任务——所需要执行的 SQL 命令越少越好。
这项原则不仅适用于数据库的设计工作，也适用于 PHP+MySQL 应用程序的开发和实施工作，但千万注意不要矫枉过正：如果 MySQL 服务器配置得当，常用的数据表应该完全驻留在内存里；如果 Apache 和 MySQL 还运行在同一台计算机上（实际情况也经常如此），网络就不会

成为提高速度的障碍；这些因素可以把执行一条 SQL 命令的开销降到非常低的水平。这时，如果可以用一条额外的 SQL 命令避免一个复杂的 PHP 循环，就可以使项目总体效率得到提高。总之，要尽量避免不必要的 SQL 命令，但千万不要不计代价地避免它们。

- **SELECT 查询结果越精准越好。**要尽最大可能避免使用那些会返回 1000 条记录的查询命令。这类查询命令的巨大开销几乎都是因为需要把查询结果传输给 PHP 解释器、PHP 解释器需要对它们进行处理而产生的，因为需要执行查询命令而产生的开销并不大。要给 *SELECT* 命令提供最适当的查询条件，在 MySQL 服务器上就会把不必要的数据过滤掉。
- **选用最高效的 PHP 编程算法（减少循环次数等）。**许多问题都有不止一个的解决方案，根据具体的应用情况，它们在效率方面肯定有高低之分。尽量避免使用 PHP 代码进行搜索和排序，这些事应该在 *SELECT* 命令里用适当的查询条件和排序规则来完成。（MySQL 最擅长的就是搜索和排序，PHP 代码很难在这两个方面比 MySQL 做得更有效率。）

用户也许已经注意到上面这些规则有几条是彼此矛盾的。在拿不准怎么做最好的时候，下面的方法往往有助于做出正确的选择：先用一条 *SELECT* 命令返回 1000 条记录（开销集中在数据传输方面），再用 20 条 *SELECT* 命令每次返回 100 条记录（开销集中在 PHP 解释器需要与 MySQL 服务器联系 20 次并等待它做出响应），最后比较这两种做法哪一种更有效率。

提示 在开始优化工作之前，先想想用户对哪些 HTML 页面的访问会发生得比较频繁。一般来说，用来阅读和查找信息页面要比用来修改数据的页面受到的访问更频繁。这里的原则是：对页面的优化工作应该从最常用的开始，对数据库设计方案的优化工作要让那些最频繁执行的任务能以最有效率的方式完成。

这里不妨假设信息查询操作和数据修改操作的发生频率是 100 比 1。如果能把信息查询操作的速度提高一倍，即使数据修改操作的速度因设计方案发生变化而需要花费原来三倍的时间才能完成，整个项目的总体效率也将得到极大的改善。

如果正在建设的网站像 Yahoo 或 GMX 那样每天的访问量高达好几百万次，程序代码的执行效率就将是最重要的是。不过，最有效的代码往往很复杂、很难以维护和很容易发生错误，所以如果网站的访问量不是非常大的话，对程序代码的优化工作还是应该把清晰、简明和易于维护放在首位。

15.10.2 统计信息和性能指标

把数据库上的各种访问功能封装起来（比如像在本章前面介绍的 *MyDb* 类那样）以后，下面这些问题的答案就很容易得到了：已经执行了多少条 SQL 命令？已经查找到了多少条记录？执行刚才那条 SQL 命令花了多少时间？创建数据库对象花了多少时间？系统自上次重启起来持续运转了多长时间。这些基本的统计信息和性能指标可以用一个 `$db->showStatistics()` 调用显示在每一个 HTML 页面的末尾。

这些信息至少可以让用户从数据库的角度对创建这个页面的开销有一个直观的印象（如图 15-14 所示）。但这里要提醒大家一句，从 HTML 页面上看到的开销时间与计算机在处理这个页面的同时还执行着哪些操作、这个页面上的数据当时是不是在 MySQL 服务器的缓冲区里等许多因素有着很大的关系。因此，不要过分相信那些统计数字，要对它们持适当的怀疑态度。

为了计算出精确的时间，可以使用 PHP 函数 `microtime_float()`，这个函数会把它的结果转换为一个浮点数字。下面是我们对 *MyDb* 类定义文件 *mydb.php* 所做的扩展：

```

// example file mydb.php
class MyDb {
    ...
    protected $sqlcounter = 0;      // counter for SQL commands
    protected $rowcounter = 0;      // counter for SELECT records
    protected $dbtime      = 0;      // time elapsed for SQL commands
    protected $starttime;          // total time
    // constructor
    function __construct() {
        ...
        $this->starttime = $this->microtime_float();
    }
    function queryObjectArray($sql) {
        $this->sqlcounter++;
        $this->printsq($sql);
        $time1 = $this->microtime_float();
        $result = $this->mysqli->query($sql);
        $this->dbtime += ($time2 - $this->microtime_float());
        if($result) {
            if($result->num_rows) {
                while($row = $result->fetch_object())
                    $result_array[] = $row;
                $this->rowcounter += sizeof($result_array);
                return $result_array;
            } else
                return FALSE;
        } else {
            $this->printerror($this->mysqli->error);
            return FALSE;
        }
    }
    ...
    function showStatistics() {
        $totalTime = $this->microtime_float() - $this->starttime;
        printf("<p><font color=\"#0000ff\">SQL commands: %d<br/>%d</font></p>\n",
               $this->sqlcounter);
        printf("<br />Sum of returned rows: %d<br/>\n", $this->rowcounter);
        printf("<br />Sum of query time (MySQL): %f<br/>\n", $this->dbtime);
        printf("<br />Processing time (PHP): %f<br/>\n",
               $totalTime - $this->dbtime);
        printf("<br />Total time since MyDb creation / " .
               "last reset: %f</font></p>\n", $totalTime);
    }
    function resetStatistics() {
        $this->sqlcounter = 0;
        $this->rowcounter = 0;
        $this->dbtime     = 0;
        $this->starttime = $this->microtime_float(); }
    private function microtime_float() {
        list($usec, $sec) = explode(" ", microtime());
        return ((float)$usec + (float)$sec); }
}

```

15.10.3 示例：高效地生成图书门类下拉列表

在前面的小节里，曾介绍过一个来自mylibraryfunctions.php文件的*build_category_array()*函数，这个函数将返回一个数组，图书门类的下拉列表就是以这个数组为基础创建的。

本节将对这个函数的 3 种变体进行对比，并演示如何一步一步地对一个函数进行优化。最慢和最快的变体之间的速度差距在图书门类很少的情况下几乎无法察觉，但随着图书门类的增加，这个速度差距会迅速加大（如果图书门类超过 1000 个，会相差 50% 左右）。

1. 创建测试数据

要想开展后面的测试工作，首先需要有一个大约 1000 条记录的图书门类清单。如果想在自己的测试系统上创建一个这样的 *categories* 数据表，调用一次本书配套的 *optimize/create_categories_test.php* 文件即可，该文件里的代码将解析 *titles.catID* 数据列上的外键约束条件、把现有的 *categories* 数据表重新命名为 *oldcategories*、然后用随机数据创建一个新的 *categories* 数据表。测试结束后，只须调用一次 *optimize/remove_categories_test.php* 文件就可以把数据库恢复成原来的样子。

2. 变体1：单次SQL查询，最低效的PHP代码

build_category_array1() 函数需要提供两个参数：一个存放着所有图书门类的数组和一个起始图书门类的 *parentCatID* 编号值（*\$parentCatID=NULL* 意味着从根元素开始显示整个图书门类树）。图书门类清单是在调用 *build_category_array1()* 函数之前只用一条 *SELECT* 命令查询出来的。

这个函数将遍历整个 *\$row* 数组去搜索有着匹配 *parentCatID* 编号值的数组元素，并把它们依次追加到静态数组 *\$tmp* 里去。每找到一个图书门类，*build_category_array1()* 函数¹就会采用递归调用搜索这个门类的下级门类。

```
// example file optimize/show_categories_list.php
$sql = "SELECT catName, catID, parentCatID FROM categories ORDER BY catName";
$categories = $db->queryObjectArray($sql);
$rows = build_category_array1($categories);

...
function build_category_array1($rows, $parentCatID=NULL, $indent=0) {
    static $tmp;
    if($indent==0)
        $tmp=FALSE; // unset does not work with static variables
    foreach($rows as $row)
        if($row->parentCatID==$parentCatID) {
            $pair[0] = str_repeat(" ", $indent*3) . $row->catName;
            $pair[1] = $row->catID;
            $tmp[] = $pair;
            build_category_array1($rows, $row->catID, $indent+1);
        }
    if($indent==0)
        return $tmp;
}
```

结果：*build_category_array1()* 函数总共需要调用 n^2 次，这也正是这第一种变体慢得让人难以忍受的原因。但从另一个方面讲，它的代码非常容易阅读和理解。

3. 变体2：许多小规模的SQL查询

build_category_array1() 函数之所以会那么慢，是因为它在搜索每一个图书门类的时候都要遍历一次整个图书门类清单。为了避免这一点，*build_category_array2()* 函数只遍历给定图书门类的直接下级门类。

换句话说，这次不是只进行一次查询，返回全部结果记录；而是进行多次查询，每次返回一小部分结果记录。（如果有 1000 个图书门类，就需要进行 1000 次查询才能返回全部的图书门类。与第一种变体相比，从 MySQL 服务器传输来的数据量是一样的，但第二种变体为获得这些数据而需要做的事情要多得多。）

与 *build_category_array1()* 函数相比，*build_category_array2()* 函数的递归调用部分没有多少变化。它们之间的最大差异体现在构造 SQL 查询命令的部分，这次只需要区分一种情况：参数 *\$parentCatID* 包含的是不是 *NULL*。注意，在 SQL 语言里，像 *catID = NULL* 这样的表达式是非法的，想知道 *catID*

1. 原文这里显然把函数名写错了。——译者注

数据列是否包含着 *NULL* 值只能用 SQL 函数 *ISNULL(catID)* 进行测试。

```
function b($parentCatID=NULL, $indent=0) {
    global $db;
    static $tmp;
    if($parentCatID==NULL) {
        $tmp = FALSE; // delete tmp
        $sql = "SELECT catName, catID FROM categories ".
            "WHERE ISNULL(parentCatID) ORDER BY catName"; }
    else
        $sql = "SELECT catName, catID FROM categories ".
            "WHERE parentCatID=$parentCatID ORDER BY catName";
    if($rows = $db->queryObjectArray($sql))
        foreach($rows as $row) {
            $pair[0] = str_repeat(" ", $indent*3) . $row->catName;
            $pair[1] = $row->catID;
            $tmp[] = $pair;
            build_category_array2($row->catID, $indent+1);
        }
    if($parentCatID==NULL)
        return $tmp;
}
```

结果：图书门类越多，变体 2 的执行速度就比变体 1 快得更多。通过这个结果可以得出一个结论：在许多场合，用增加 SQL 查询次数的办法去避免使用效率不高的 PHP 代码对大量 *SELECT* 查询结果进行处理往往更划算。（如果图书门类很少，变体 2 的执行速度反而会比变体 1 还要慢。）

如图 15-14 所示，虽然创建图书门类下拉列表的总体时间缩短了，但 MySQL 服务器的负担却要比第一种变体高出许多（约 0.38s 比约 0.04s）¹。

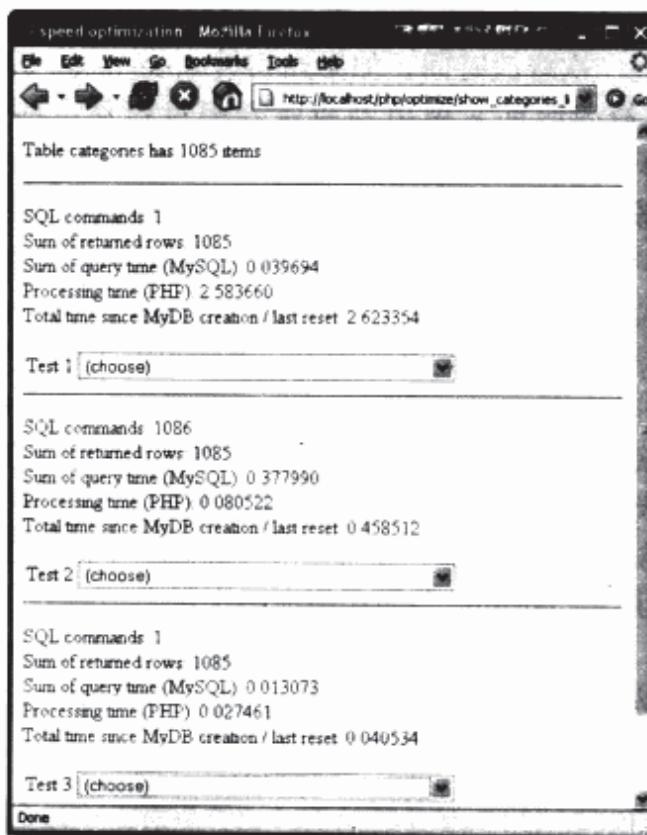


图 15-14 创建图书门类下拉列表的速度对比

1. 原文在这里给出的数字在图 15-14 里根本找不到。这里是按图 15-14 的数据译的。——译者注

注解 用户或许会奇怪这里为什么没有使用任何预处理语句。答案很简单：因为 `build_category_array2()` 是一个递归函数，所以在这里使用预处理语句比较困难，就算用了也不会对提高速度有明显的帮助。预处理语句更适合在一个循环里用不同的参数反复执行同一条查询命令。

4. 变体3：单次SQL查询，经过优化的PHP代码

变体3又回到了变体1的思路上：先用一条查询命令把所有的图书门类全部查找出来。但这次我们不再盲目地遍历整个图书门类数组去搜索一个特定的元素，而是提前创建了 `$catNames[]` 和 `$subcats[]` 两个数组，它们分别使用各个图书门类的 `catID` 和 `parentCatID` 编号值作为下标（关键字）。这样一来，不必经过漫长的搜索和循环就可以直接找到需要的数据。已经在 15.9.2 节里分析过这个变体的代码，这里就不再重复了。

结果：在图书门类比较多的时候，变体3的执行速度比前两种变体都要快得多。但即便是在图书门类比较少的时候，创建 `$catNames[]` 和 `$subcats[]` 两个数组所花费的时间也没有让变体3落在前两种变体的后面。这个变体的唯一不足是需要人们多动些脑子才能弄明白这段代码，这让它在以后修改起来会比较困难（尤其是修改工作由其他程序员来进行的时候）。

5. 其他值得改进的地方

第3种变体的效果已经非常好了，甚至可以说已经没有必要再对它进行什么优化。不过，如果假设图书门类清单相对来说很少需要修改、但经常会被读取，下面两项改进就很值得考虑：

□ **使用一个辅助数据表。**每次需要显示一个图书门类下拉列表框的时候，图书门类列表都需要按照层次结构进行排序和缩进。这个步骤的结果可以保存到一个辅助数据表里去。

在输出这个辅助数据表之前，必须检查是否有图书门类发生过变化。这里的判断标准可以是图书门类的总数和最近一次修改时间。如果当初给 `categories` 数据表配上一个 `TIMESTAMP` 数据列，用最后一次修改时间来做出判断就会相对容易一些（`SELECT MAX(ts) FROM categories`）。总之，只要 `categories` 数据表发生了变化，辅助数据表就需要重新创建。

□ **采用另外一种数据表设计方案。**比增加一个经常需要刷新的辅助数据表更精巧的办法是给 `categories` 数据表增加两个数据列：`indent` 数据列给出各图书门类的缩进级别，`position` 数据列给出各图书门类在层次结构中的位置。这么一来，由 `SELECT catName, catID, indent FROM categories ORDER BY position` 命令返回的图书门类清单就可以直接用于创建 HTML 结果页面上的下拉列表框，连图书门类在层次结构中的先后顺序问题都用不着再费心了。

当然，这个办法也有缺点：新增加的两个数据列与数据库设计理论相违背，它们包含冗余数据，并且增加了数据表的存储空间占用量（虽然不是很大）。最后，插入新图书门类的操作会变得更加复杂，在插入新门类之前，必须先为它确定一个正确的 `position` 值并对层次结构中位于新门类之下所有门类的 `position` 值做出相应的调整。

从追求效率的角度看，这些缺点不算什么大问题。它们只在 `categories` 数据表发生变化时才会发生，而刚才的假设是图书门类清单很少需要修改。尽管如此，用来插入新图书门类记录的 PHP 代码将不可避免地会变得更复杂和更容易出问题。

15.11 Unicode

在默认的情况下，PHP 使用 `Latin1` 字符集（SISO-8859-1）来生成 HTML 文档。直接在 PHP 代码里设置另外一个字符集目前还不行，PHP 的未来版本也许会提供这一功能。不过，从理论上讲，PHP

差不多可以使用任何一种字符集来创建 HTML 文档。*echo()* 和 *print()* 函数在输出字符串的时候不会对它们做任何改变，所以在使用这两个函数时根本不必考虑字符集的问题。

真正的问题在于所有的 PHP 字符串函数都假设它们正在处理的字符串使用的是 *Latin1* 字符集（每个字符占用 1 个字节）。万一字符串是按 UTF-8 字符集编码的，其中的每个字符最多将占用 3 个字节，而这会导致一系列的问题：

- *strlen()* 函数的返回值将是字节个数而不是字符个数。
- 字母大小写转换操作符的结果可能会不正确。
- 对一组字符串的排序结果可能会不正确。
- *stripslashes()*、*substr()* 和 *split()* 等常用的字符串函数会引起问题。

这些问题并不是完全无法解决：PHP 提供的 *utf8_encode()* 和 *utf8_decode()* 函数可以完成字符串在 UTF-8 和 ISO-8859-1 编码方案之间的转换。如果 PHP 在编译时带有 *iconv* 扩展模块（UNIX/Linux）或者如果这个扩展模块已被启用（Windows：在 *php.ini* 文件里加上 *extension = php_iconv.dll* 语句），一些标准的 PHP 字符串函数就会有 *iconv_xxx()* 形式的变体可供选用。比如说，*iconv_strlen(\$s, "UTF-8")* 可以正确地返回一个 UTF-8 字符串的实际字符个数。*mbstring* 扩展模块提供了更多的辅助函数。利用这些函数，我们就可以在规则表达式里使用多字节字符串、就可以收发内容是多字节字符串的电子邮件、等等。

总之，如果 *Latin1* 字符集已经足以满足应用需要，那就不要再考虑其他的字符集了；这可以把可能遇到的问题减少到最少。反之，如果程序要求必须使用 Unicode 字符集，就应该认真学习和体会本节后续内容里将要介绍的技巧，并根据具体情况在编程工作中加以应用。

设置 HTML 和 HTTP 编码

- **HTML 标记<meta>**。根据 HTML 标准，凡是沒有另行指定一个字符集的 HTML 文档都将使用 *Latin1*（也就是 ISO-8859-1）字符集。可以在 HTML 文档的开头用一个 *<meta>* 标记来另行指定一个字符集。下面是有关的 HTML 代码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html><head>
<meta http-equiv="Content-Type"
  content="text/html; charset=utf-8" />
...
```

charset 参数的可取值包括 *utf-8*、*iso-8859-1*、*iso-8859-15* 等。不过，这里还有一个问题：有不少 Web 浏览器会忽略 *<meta>* 标记里的字符集设置信息而完全依赖 HTTP 首部里给出的设置。

- **HTTP 首部信息**。HTTP 协议对 Web 浏览器与 Web 服务器之间的通信如何进行做出了规定。除了 HTML *<meta>* 标记，这个协议还提供了设置字符集的第 2 种办法。大多数 Web 浏览器会依赖 HTTP 协议里的字符集设置信息而忽略 *<meta>* 标记做出的字符集设置。

HTTP 协议是在文档类型首部信息里给出字符集设置的。HTTP 首部在 HTML 文档正文之前发送，所以它不是 HTML 代码的一部分。有些 Web 浏览器提供了一个查看页面信息和属性的命令（比如 FireFox 浏览器的菜单命令 Tools（工具）| Page Info（页面信息），如图 15-15 所示），这类命令可以把 HTTP 首部里的一部分信息也显示出来。

那么，HTTP 首部里关于文档类型和字符集的信息又是从哪儿来的呢？这有几种可能性。（如果有多项相互冲突的设置，在下面的清单里列在前面的设置将优先考虑。）

- 负责生成 HTML 页面的 PHP 代码：

```
<?php header("Content-Type: text/html; charset=utf-8"); ?>
```

- PHP 的默认设置 (php.ini 文件里的 *default_charset="utf-8"* 语句)。

- Apache 服务器上的 Web 子目录配置参数 (.htaccess 文件里的 *AddDefaultCharset* 选项)。

- Apache 服务器的默认配置 (http.conf 文件里的 *AddDefaultCharset utf-8* 选项)。



图 15-15 查看 HTML 文档的字符集设置

如果网站完全由 Unicode 文档构成，修改 Apache 配置文件将是最简便有效的办法。如果不能修改 Apache 和 PHP 的配置(比如当网站设立在某个 ISP 的计算机上时)，在 PHP 脚本里插入一个 *header()* 函数是最保险的办法。这里要提醒大家一句，如果想使用 *header()* 函数，就必须在输出其他 HTML 内容之前调用它。

1. MySQL与Unicode字符集

mysql 和 *mysqli* 接口都没有提供可以用来设置字符集的函数或方法。不过，下面这条简单的 SQL 命令可以通知 MySQL 服务器：随后到达的 SQL 命令采用的是 UTF-8 编码、*SELECT* 查询结果也要使用这种编码：

```
SET NAMES 'utf8'
```

用 *mysql_query()* 函数或 *\$mysqli->query()* 方法执行这条命令即可。

请注意，*SET NAMES* 命令只对 MySQL 服务器与 PHP 脚本之间的通信有影响，它对 MySQL 数据表里实际存储的数据所使用的字符集没有任何影响 (MySQL 数据表里的字符集设置是在 *CREATE TABLE* 命令创建它们的时候确定的)。收到 *SET NAMES* 命令之后，MySQL 将自动地把所有的字符串从有关数据表的字符集转换为新设置的字符集。这里要特别提醒大家注意这样一个问题：如果字符串里有新字符集无法表示的字符，那些字符将被替换为一个问号，这意味着比较特殊的字符容易导致信息缺失和不完整。

2. PHP字符集函数

几乎所有的 PHP 脚本都会用到一些字符串函数。在默认的情况下，PHP 字符串函数都将认为它们正在处理的字符串是由一些单字节字符构成的。这种假设显然不适用于 UTF-8 字符串，因为每个

UTF-8 字符可以占用 1~3 个字节。下面是一些需要在编程和移植代码时注意的问题：

- *printf()* 和 *echo()* 等输出型函数一般不会有什么问题。
- 用“.”操作符合并字符串不会引起任何问题。
- 对字符串进行排序的工作最好交给 MySQL 去完成。
- 在生成一个与 UTF-8 字符集兼容的表单时，允许调用 *htmlentities()* 和 *htmlspecialchars()* 函数：

```
echo htmlspecialchars($txt, ENT_COMPAT, "UTF-8");
echo htmlentities($txt, ENT_COMPAT, "UTF-8");
```

- 前面提到的 PHP 扩展模块 *iconv* 和 *mbstring* 为一些基本的字符串函数提供了同样功能的替代品。（但令人遗憾的是，这些扩展模块没能为所有的 PHP 字符串函数都提供一个与 UTF-8 字符集兼容的替代品，至少在目前是如此。）

3. 示例

在本章前面的内容里以 *mylibrary* 数据库为例向大家介绍了很多 PHP 示例脚本，在本书配套的 PHP 示例文件部分有一个 *unicode* 目录，可以在那里找到那些示例脚本与 UTF-8 字符集兼容的变体。这个示例项目在 Unicode 转换方面基本上没有问题，下面是对字符集转换工作的总结：

- 在 *find.php*、*titleform.php* 和 *categories.php* 文件的开头插入了一个 *header()* 函数，把 HTTP 编码转换为 Unicode 编码：

```
<?php header("Content-Type: text/html; charset=utf-8"); ?>
```

- 在这些文件的 HTML 首部插入了一个 *<meta>* 标记，把 HTTP 编码转换为 Unicode 编码。（正如刚才指出的，有很多 Web 浏览器会忽略这项信息。但从遵循 HTML 语法的角度考虑，它们还是应该给出的。）

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

- 在与 MySQL 服务器建立起连接之后，用 *SET NAMES 'utf-8'* 命令把通信改为使用 Unicode 进行：

```
$db = new MySql();
$db->execute("SET NAMES 'utf8'");
```

- 把所有的 *htmlspecialchars(\$txt)* 函数调用替换为 *htmlspecialchars(\$txt, ENT_COMPAT, "utf-8")*。为了减少打字量，在 *mylibraryfunction.php* 文件里定义了一个新的 *htmlspecialchars_utf8()* 函数。

```
function htmlspecialchars_utf8($txt) {
    return htmlspecialchars($txt, ENT_COMPAT, "UTF-8"); }
```

已知问题。在示例代码里，有一些字符串函数与 UTF-8 字符集不兼容。其中特别容易出问题是 *title.php* 文件里的 *explode()* 函数，其用途是把多位作者的姓名从一个“*name1;name2*”格式的字符串里提取出来。可是，万一用户输入的某一个作者姓名里包含着一个 UTF-8 编码是 59 的字节（这个编码在 ASCII 字符集里是分号），*explode()* 函数就会把那位作者的姓名从那个字节处分开为两截。这种非法 Unicode 字符串很有可能存在。（并不是每一个随机的字节代码序列都可以转换为一个合法的 UTF-8 字符串。）

这样的错误在笔者进行的测试中没有出现。但如果在测试 *titleform.php* 脚本时输入的是多位作者的日文或中文图书，就迟早会发生这样的事情。如果真的如此，来自 *mbstring* 扩展模块的 *mb_split()* 函数可以帮助解决这个问题。

另一个容易出问题的函数是 *stripslashes()*——虽然它在这个示例里不应该出任何问题。在示例代码里多次使用了这个函数来去掉表单数据里的反斜线转义字符。（根据其具体的配置，PHP 可以自动

插入反斜线对特殊字符进行转义。PHP 文档把这一功能称为 *Magic Quotes*, 有关这一功能的详细信息可以在任何一本关于 PHP 的好书里查到。)

stripslashes() 函数与 UTF-8 字符集是兼容的¹, 但 *Magic Quotes* 功能与 Unicode 编码一起使用——这意味着同一个特殊字符将被转义两次——却会让这个函数返回一个错误的结果。我们应该把可能引起这种错误的情况提前排除掉。最保险的办法是通过 *php.ini* 文件里的 *magic_quotes_gpc = Off* 选项禁用 PHP 的 *Magic Quotes* 功能。

mylibrary 数据库的字符集。 *mylibrary* 数据库里的所有数据表都使用的是 *Latin1* 字符集（这个数据库的设计细节见第 8 章）。读者或许会对没有把这些数据表转换到 UTF-8 字符集上的做法感到奇怪：难道不想让 *mylibrary* 数据库收录以 Unicode 字符串输入的图书、作者和图书门类吗？是这样的，把这些数据表转换到 UTF-8 字符集并没有绝对的必要，因为 MySQL 服务器能自动完成字符串在 UTF-8 字符集（PHP 代码）与 *Latin1* 字符集（*mylibrary* 数据库）之间的必要转换。总的来说，这个示例项目在笔者进行的测试中表现得还不错。可是，万一用户输入的 Unicode 字符是 *Latin1* 字符集无法表示的，就可能发生信息缺失或不完整的问题。

作为一个结论，如果想把一个 PHP 项目转换到 UTF-8 字符集，首先应该把这个项目所涉及的数据库转换到 UTF-8 字符集；否则就没有实际的意义。如果真想这么做，可以用如下所示的 SQL 命令把数据表里的现有数据转换到 UTF-8 字符集：

```
ALTER TABLE tblname CONVERT TO CHARACTER SET 'utf8' [COLLATE 'utf8...']
```

15.12 二进制数据 (BLOB) 和图像

在与 PHP 和 MySQL 有关的新闻组或网上论坛里，经常可以看到这样的问题：应该怎么做才能让访问者可以把图像上传到我的网站？又应该怎么做才能把那些图像再显示给访问者看？本节将通过一个示例项目（如图 15-16 所示）来回答这些问题，并演示一些必要的程序设计技巧。

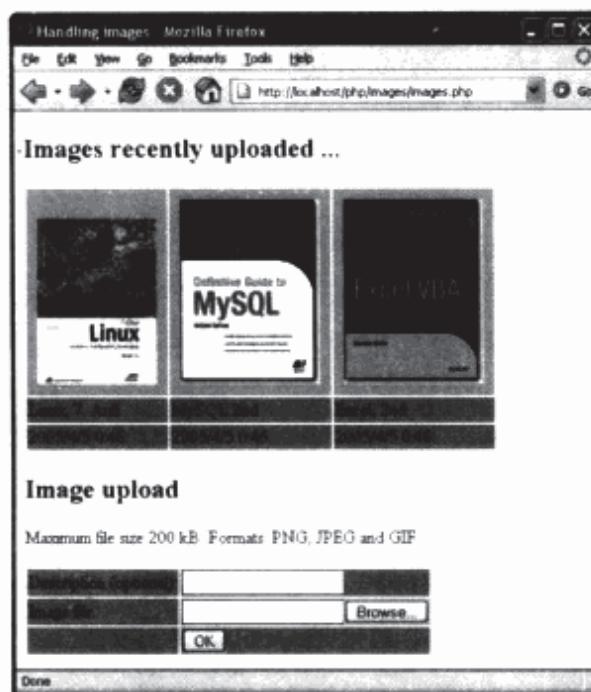


图 15-16 图像的存储与显示

1. 原书这里出现重大错误，它说 *stripslashes()* 函数与 UTF-8 字符集不兼容。——译者注

15.12.1 在数据库里存储图像的基础知识与编程技巧

1. 数据库设计方案

本示例从 *testimages* 数据库中的 *images* 数据表开始。下面是创建这个数据表的 *CREATE TABLE* 命令：

```
CREATE TABLE images (
    id      BIGINT      NOT NULL AUTO_INCREMENT,
    name    VARCHAR(100) NOT NULL,
    type    VARCHAR(100) NOT NULL,
    image   LONGBLOB    NOT NULL,
    ts      TIMESTAMP   NOT NULL,
    PRIMARY KEY (id))
```

images 数据表由以下几个数据列构成：*id*, 主键; *name*, 图像的名字; *type*, 图像的格式（如 *JPEG IMAGE*）; *image*, 图像数据; *ts*, 图像的插入时间。

注意 从追求效率的角度讲, 把图像或其他类型的大块二进制数据集存储在一个数据库里并不是个好主意。对于这类应用, 传统的文件系统会更适合。如果数据库里有很多BLOB数据, 就会在速度方面蒙受很大的损失, 这主要是因为数据库系统很难把这类数据及其索引缓存在计算机的内存里。更有效率的办法是把图像保存为普通的文件, 在数据库里只保存它们的文件名。

可是, 在实际工作中, 把二进制数据保存到BLOB数据列里的做法还是比较常见的。这主要是由于两方面的原因: 其一, 把所有数据集中保存在同一个地方可以为数据管理工作(备份、更换服务器等)提供许多方便; 其二, 很多ISP不允许PHP脚本或其他程序代码创建和读取本地文件。

2. 用来上传文件的HTML表单

为了让一个HTML页面具备向Web服务器上传文件的能力, 就必须在这个页面上创建一个特殊的表单。这里的关键有两点: 一是必须把 *method* 和 *enctype* 属性设置正确; 二是文件名输入字段(Web浏览器会为这种字段自动提供一个 *Browser* (浏览) 按钮) 必须是一个带有 *type="file"* 属性的 *<input>* 字段。

如果想对上传文件的长度做出限制, 可以用一个带有 *name= "MAX_FILE_SIZE"* 属性的隐藏字段来做到这一点。下面是本示例所使用的文件上传表单(如图 15-16 所示)的部分代码, 它把上传文件的最大长度限制在 200KB。为了获得整齐美观的显示效果, 这个表单将排版为一个HTML表格, 表格元素的布局由 *table.css* 文件控制。

```
<form method="post" action="filename.php"
      enctype="multipart/form-data">
<input type="hidden" value="204800" name="MAX_FILE_SIZE" />
filename: <input name="imgfile" type="file" />
<input type="submit" value="OK" name="submitbtn" />
</form>
```

3. 用PHP代码处理\$_FILES数组

与其他用途的HTML表单一样, 在文件上传表单返回到PHP脚本的表单数据里也有一个 *\$_POST* 数组, 这个数组包含着文件上传表单上的普通字段的返回内容(比如说, *\$_POST['descr']* 包含着对上传图像的描述, *\$_POST['submitbtn']* 包含着字符串'OK'、等等)。与上传文件有关的信息可以在 *\$_FILES* 数组里找到。从 *\$_FILES* 数组提取上传文件信息时必须使用当初在文件上传表单里给出的关键字

(HTML 标记`<input ... type="file">`的 `name` 属性值) —— 具体到这个例子, 这个关键字是 `imgfile` (见上面的代码)。文件信息提取操作返回的还是一个数组, 这个数组的元素就是上传文件的各项属性信息了:

```
$imgfile = $_FILES['imgfile'];
$name   = $imgfile['name'];      // file name (without volume/path)
$type   = $imgfile['type'];      // file type, e.g., "image/gif"
$size   = $imgfile['size'];      // file size in Bytes
$uperr  = $imgfile['error'];     // error number (0 = no error)
$tmpfile = $imgfile['tmp_name']; // name of the local temporary file
```

Web 服务器将把上传文件保存为一个本地临时文件, 并把这个临时文件的文件名存入 `$tmpfile` 变量。可以在文件上传表单里给定的 PHP 脚本(本例中是 `filename.php` 脚本, 见上面的代码)里使用 `fopen()` 和 `fread()` 等函数去读取这个临时文件。注意, 这个临时文件将在给定 PHP 脚本执行结束时被自动删除; 换句话说, 如果想保留上传文件, 就需要在 PHP 脚本里把临时文件转存为一个正式文件(本例只是把上传文件里的数据存入数据库, 但不保留上传文件)。

从信息安全的角度讲, 文件传输是一种相当危险的操作。因此, 在对上传文件做进一步处理之前, 一定要对它的合法性做严格的检查。下面这段代码进行了 3 项检查: `$tmpfile` 变量所代表的文件是否为空、它是否发生过传输错误、它会不会不是上传来的那个文件而是一个本地正式文件 (`is_uploaded_file()`)。最后一项检查非常有必要, 它可以防止黑客通过篡改 `$tmpfile` 变量而盗取或破坏某个本地系统文件(如 UNIX/Linux 系统的密码文件 `/etc/passwd`) —— 这种事并不是没有发生过!

```
if (!$tmpfile or $uperr or !is_uploaded_file($tmpfile))
    echo "<p>error ...</p>\n";
else {
    ... // read and process the temporary file
}
```

如有必要, 还应该对 `$tmpfile` 变量所代表的那个文件的类型、长度等进行检查。

4. 把上传文件存入 MySQL 数据库

把上传文件存入 MySQL 数据库的办法是: 把整个文件读入一个变量, 然后执行一条 `INSERT` 命令。对最简单的情况, 有关代码如下所示:

```
$file = fopen($tmpfile, "rb"); // open file (read-only, binary)
$imgdata = fread($file, $size); // read file
fclose($file);
$mysqli->query(
    "INSERT INTO images (image) " .
    "VALUES ('" . $mysqli->escape_string($imgdata) . "')");
```

在后面给出的示例代码里, `INSERT` 命令都比这要复杂一些, 那是因为需要把图像的 MIME 类型(如 `GIF image`)、文件名及关于图像的说明等信息都存入数据库。

5. 从 MySQL 数据库读出并显示图像

从 MySQL 数据库读出一幅图像并把它显示在一个 HTML 文档里相对要更复杂一些。简单地说, 因为这里涉及到查询数据库的问题, 所以同一个 PHP 脚本不可能一次完成创建一个 HTML 文档 (PHP 脚本的正常任务) 和生成一些用来显示图像的数据这两项任务。这两项任务必须用两个 PHP 脚本来完成。

第一个(正常任务)脚本负责创建一个 HTML 页面, 并在这个页面的适当位置为那幅图像生成一个``标记。不过, 这次不是在这个标记里写入一个本地图像文件的名字, 而是要写入第二个(从数据库提取图像) PHP 脚本的名字和那幅图像的 ID 编号值。下面是第一个脚本的代码:

```
// PHP-Script pictures.php
echo "<img src=\"showpic.php?id=$id\" />";
```

比如说，当 Web 浏览器遇到``标记时，就会去调用第二个脚本`showpic.php?id=3`。下面是第 2 个脚本的代码（经过一些简化）：

```
// PHP-Script showpic.php
$result = $mysqli->query("SELECT image FROM images WHERE id = $id");
$row = $result->fetch_object();
header("GIF image"); // file type
echo $row->image;
```

在实际应用中，第二个脚本还增加一些代码去完成以下任务：连接数据库、确定参数`$id` 的值、检查脚本执行是否出错并进行必要处理等。如果在图像数据库里不仅存放着 GIF 图像，还存放着其他格式的位图，那就需要从数据库里查出图像文件的类型并使用 `header()` 函数调用完成必要的设置。这里的关键之处是不能让第二个脚本再创建出一个 HTML 文件，也不能在调用 `header()` 函数之前用 `echo` 命令创建任何 HTML 输出——只有这样，PHP 解释器才会知道这次不是要创建一个普通的 HTML 文本文件，而是要创建一个图像文件。

6. 如何处理大文件

如果想让用户能够上传非常大的文件，就必须注意以下几个要点：

- 一定要把上传文件的最大长度限制设置得足够大。这个限制由 PHP 配置文件 `php.ini` 里的 `upload_max_filesize` 变量控制（默认设置是 2MB）。
- 一定要把 `POST` 传输操作的数据最大长度限制设置得足够大。这个限制由 PHP 配置文件 `php.ini` 里的 `post_max_size` 变量控制（默认设置是 8MB）。
- PHP 解释器必须保留足够的内存空间来处理最大长度的上传文件。这个限制由 PHP 配置文件 `php.ini` 里的 `memory_limit` 变量控制（默认设置是 8MB）。
- PHP 解释器必须给脚本以足够的时间去处理有关数据。这个限制由 PHP 配置文件 `php.ini` 里的 `max_execution_time` 变量控制，30 秒的默认设置往往不足以让脚本完成对大文件的处理。
- 可能需要加大变量 `max_input_time` 的值（默认设置是 60s）。这个变量控制着 PHP 将为通过因特网进行的数据传输操作分配多少时间。
- 为了让 PHP 脚本与 MySQL 服务器之间的图像数据传输工作取得成功，数据包的最大长度（这个数字对 `INSERT` 等命令有影响）必须足够大。MySQL 变量 `max_allowed_packet` 既可以在 MySQL 配置文件（`/etc/my.cnf` 或 `my.ini`）里修改，也可以用 SQL 命令 `SET max_allowed_packet = n` 来设置。这个变量的默认设置只有区区 1MB！

15.12.2 在数据库里存储图像的程序代码

这个示例项目的全部代码由表 15-8 文件构成。

表 15-8 示例项目代码的文件

文件名	内 容
connect.php	提供 <code>connect_to_picdb()</code> 和 <code>array_item()</code> 函数
images.php	显示一个 HTML 表格（里面是最近上传来的 10 幅图像）和一个表单（供用户上传新的图像）。这个脚本还包含着对表单返回数据进行处理的代码。
showpic.php	显示单个图像
table.css	用来对 HTML 表格进行排版的 CSS 文件

1. connect.php脚本：连接MySQL服务器

这个文件包含着两个函数：一个是 `connect_to_picdb()`，负责创建与 MySQL 数据库 `testimages` 的连接；另一个是 `array_item()`，负责从一个 PHP 数组里读出一个元素（如果该元素存在的话）。

```
// example file images/connect.php
function connect_to_picdb() {
    $mysqluser = "root";           // user name
    $mysqlpw = "xxxx";             // password
    $mysqlhost = "localhost";       // name of the computer on which MySQL is running
    $mysqldb = "testimages";        // name of the database
    $mysqli = new mysqli($mysqlhost, $mysqluser, $mysqlpw, $mysqldb);
    if(mysqli_connect_errno()) {
        echo "<p>Sorry, no connection to database ...",
              "</p></body></html>\n";
        exit();
    }
    return $mysqli;
}
```

2. images.php脚本：图像的存储和读取

`images.php` 脚本文件里的代码分为三大部分。第一部分负责处理表单数据。上传文件的内容数据只有在传输过程没有发生任何错误的情况下才会被存入 `images` 数据表。

代码中的 `switch` 结构负责检查上传文件的类型并返回一个新的 MIME 字符串。（MIME 是 multipurpose internet mail extensions 的缩写，意思是多用途因特网电子邮件的（文件）扩展名。MIME 最早只用于标识电子邮件附件的类型，但现在 MIME 还可以用来控制 Web 浏览器在遇到不同类型的数据时应该进行哪些操作和处理。）要想正确地显示不同格式的图像，就必须正确地给出它们的 MIME 类型。因此，把每一幅图像的 MIME 类型与图像本身的数据一起保存在了 `images` 数据表里。但令人遗憾的是，`images` 数据表 `type` 数据列里的信息在经过文件传输操作之后，有可能不再是 MIME 类型的标准名字，而是会变成其他的字符串（会变成什么样要取决于用户使用的是哪一种 Web 浏览器）。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html><head> ... </head><body>
<?php // images/images.php
include("connect.php");
$mysqli = connect_to_picdb();
// Part I: store the image
$submitbtn = array_item($_POST, 'submitbtn');
$descr = array_item($_POST, 'descr');
$imgfile = array_item($_FILES, 'imgfile');
// are there form data to be processed?
if($submitbtn == 'OK' and is_array($imgfile)) {
    $name = $imgfile['name'];
    $type = $imgfile['type'];
    $size = $imgfile['size'];
    $uperr = array_item($imgfile, 'error');
    $tmpfile = $imgfile['tmp_name'];
    if(!$descr) $descr = $name;
    switch ($type) {
        case "image/gif":
            $mime = "GIF Image"; break;
        case "image/jpeg":
        case "image/pjpeg":
            $mime = "JPEG Image"; break;
        case "image/png":
        case "image/x-png":
            $mime = "PNG Image"; break;
        default:
            $mime = "unknown";
    }
    $query = "INSERT INTO images (name, type, size, mime, tmp_name, error, descr) VALUES ('$name', '$type', '$size', '$mime', '$tmpfile', '$uperr', '$descr')";
    $result = $mysqli->query($query);
    if($result) {
        echo "File uploaded successfully!";
    } else {
        echo "Error uploading file: " . $mysqli->error;
    }
}
$mysqli->close();
?>
```

```

if(!$tmpfile or $uperr or $mime == "unknown" or
   !is_uploaded_file($tmpfile))
  echo "<p>error message ...</p>\n";
else {
  // read file and store it in the database
  $file = fopen($tmpfile, "rb");
  $imgdata = fread($file, $size);
  fclose($file);
  if(!$mysqli->query(
    "INSERT INTO images (name, type, image) " .
    "VALUES ('" . $mysqli->escape_string($descr) . "', '" .
    "'$mime', '" .
    "'". $mysqli->escape_string($imgdata) . "')"))
    printf("<p>error message ...: %s</p>\n", $mysqli->error);
}
}

```

images.php 脚本的第二部分代码负责从数据库里读出最后上传来的 10 幅图像，并把它们显示在一个 HTML 表格里。正如刚才描述的，图像显示工作由 showpic.php 脚本负责具体完成。

```

// Part II: Display the most recently inserted images
echo "<h2>last inserted images ...</h2>\n";
$sql =
  "SELECT id, name, " .
  "DATE_FORMAT(ts, '%Y/%c/%e %k:%i') AS dt " .
  "FROM images ORDER BY ts DESC LIMIT 10";
$result = $mysqli->query($sql);
if($result->num_rows==0)
  echo "<p>There are no more images ...</p>\n";
else {
  // display the images in a table
  while($row = $result->fetch_object())
    $rows[] = $row;
  echo "<table>\n<tr>";
  for($i=0; $i<sizeof($rows); $i++) // first line: images
    echo "<th>",
      "<img src='showpic.php?id=" .
      $rows[$i]->id . "' /></th>";
  echo "</tr>\n<tr>";
  for($i=0; $i<sizeof($rows); $i++) // second line: text
    echo "<td>", htmlspecialchars($rows[$i]->name), "</td>";
  echo "</tr>\n<tr>";
  for($i=0; $i<sizeof($rows); $i++) // third line: date
    echo "<td>", $rows[$i]->dt, "</td>";
  echo "</tr>\n</table>\n";
}

```

images.php 脚本的第三部分代码负责生成供用户上传新图像的 HTML 表单的 HTML 代码。为了获得整齐美观的显示效果，用了一个 HTML 表格来排版这个表单。

```

// Part III: represent the form
?>
<h2>image upload</h2>
<p>maximal file size: 200 kByte.
  allowed formats: PNG, JPEG, and GIF.</p>
<table>
<form method="post" action="images.php"
      enctype="multipart/form-data">
  <input type="hidden" value="204800" name="MAX_FILE_SIZE" />
  <tr><td>description (optional):</td>
    <td><input name="descr" type="text" /></td></tr>
  <tr><td>file name:</td>
    <td><input name="imgfile" type="file" /></td></tr>
  <tr><td></td>

```

```

<td><input type="submit" value="OK"
           name="submitbtn" /></td></tr>
</form>
</table>
</body></html>

```

15.13 存储过程

关于如何在 PHP 脚本里使用 MySQL 存储过程 (stored procedure, SP) 没什么可多说的：调用或修改存储过程 (*CREATE FUNCTION / PROCEDURE*) 都可以像平常一样通过 *mysqli* 接口所提供的有关方法来完成。下面是大家在实际工作中可能会遇到的一些情况：

- 调用一个不会返回任何记录的存储过程。可以使用 *\$mysqli->real_query()* 或 *\$mysqli->query()* 方法来完成这种操作：

```

$sql = "CALL categories_insert('abc', 123, @newcatID)";
$ok = $mysqli->real_query($sql);

```

- 调用一个会返回一些记录（单条 *SELECT* 命令）的存储过程。这种操作应该尽量使用 *\$mysqli->query()* 方法来完成，而这是因为这个方法可以立刻返回查询结果：

```

$sql = "CALL categories_select(123)";
$result = $mysqli->query($sql);

```

- 调用一个会返回多组结果（多条 *SELECT* 命令）或者需要执行多条 SQL 命令的存储过程。如果在同一个存储过程里执行了多条 *SELECT* 命令，该过程将返回多组结果记录。为了对多组结果记录进行处理，就必须用 *\$mysqli->multi_query()* 方法来执行这种存储过程。各组结果记录需要用 *use_result()* 或 *store_result()* 方法来依次读取，前进到下一组结果记录需要使用 *next_result()* 方法：

```

$sql = "CALL many_selects(123)";
$ok = $mysqli->multi_query($sql);

```

还可以使用 *multi_query()* 方法来一次执行多条 SQL 命令（这么做的好处是不必顾虑将会返回多少组结果记录的问题）。此时，SQL 命令之间必须用分号 (;) 隔开。

```

$sql = "SET @myvar=1;
        CALL complicated_calculation(123, @myvar);
        SELECT * FROM table WHERE id=@myvar";
$ok = $mysqli->multi_query($sql);

```

- 定义一个新的存储过程 (*CREATE FUNCTION / PROCEDURE* 命令)。在绝大多数的 PHP 项目里，很少有必要去定义一个新的存储过程。但万一需要这样做，就必须使用 *real_query()* 或 *query()* 方法去执行 *CREATE FUNCTION* 或 *CREATE PROCEDURE* 命令；千万不要使用 *multi_query()* 方法。这是因为 *multi_query()* 方法会把分号解释为 SQL 命令之间的分隔符，可是分号又会出现在存储过程的代码里。如果想在执行 *CREATE* 命令之前先执行一条 *DELETE* 命令，就必须分别执行这两条命令。

```

$sql = "DROP PROCEDURE IF EXISTS mysp";
$ok = $mysqli->real_query($sql);
$sql = "CREATE PROCEDURE mysp (OUT abc DOUBLE)
BEGIN
    DECLARE t, subt VARCHAR(100);
    ...
END";
$ok = $mysqli->real_query($sql);

```

注解 即使在编写 PHP 脚本时使用的是比较老的 *mysql* 功能模块，也可以使用存储过程。唯一的限制是 *mysqli* 接口中的 *multi_query()* 方法在 *mysql* 功能模块里没有等效替代品。换句话说，在使用 *mysql* 功能模块的 PHP 脚本里，必须先把一个包含着多条 SQL 命令的字符串拆分为多个包含着一条 SQL 命令的字符串，然后再依次执行那些命令。

15.14 SP Administrator

截止到 2005 年 3 月，现有的 MySQL 用户操作界面都不适合用来处理、管理和测试 MySQL 存储过程（stored procedure，SP）。在这方面的专用软件工具出现之前，下面将要介绍的 SP Administrator 程序会是一个很好的过渡。与此同时，它也是研究 PHP 编程技巧的好例子。

与可能会在近期正式推出的 PhpMyAdmin 新版本相比，SP Administrator 的人机界面谈不上精致，但它的确可以帮助我们高效便捷地完成以下几项 SP 日常管理任务：

- 创建新 SP（提供了代码模板）；
- 编辑 SP；
- 执行和测试 SP，条理清晰地显示任意多组 *SELECT* 查询结果；
- 删除 SP；
- 以数据库为单位对所有的 SP 进行备份和恢复。

这个程序展示了一些很有意思的编程技巧和通过 PHP 脚本使用 SP 时需要注意的各种问题。

15.14.1 安装 SP Administrator

SP Administrator 程序的代码收录在本书配套的 *spadmin* 目录里。在试用这个程序之前，请大家根据自己的具体情况对 *password.php* 文件做出必要的修改，如果在这个文件里设定的 MySQL 用户名不是 *root*，将需要具备 *Execute*（执行）权限才能定义和执行 SP。

15.14.2 使用 SP Administrator

这个程序的用户界面全部在 *spadmin.php* 脚本生成的 HTML 页面上。在那里，必须首先选择一个数据库，然后在第二个表单里选择一个打算执行的 SP 管理任务（见上面）。有些操作（*ALTER*、*CALL*、*DROP*）还需要选择一个 SP 作为操作对象。（警告：SP 删除操作会立刻执行而不是要求用户加以确认。）图 15-17 是这个程序正在修改一个 SP 时的画面。

1. 对 SP 进行测试/试用

如果选择的操作动作是 *CALL A PROCEDURE*（调用一个 SP），可以在输入区给出多条命令，随后它们将都被执行，因此而产生的所有 *SELECT* 查询结果将按照有关命令的执行先后顺序依次显示（如图 15-18 所示）。

2. 备份和恢复 SP

当执行 *Backup All SPs*（备份所有的 SP）操作的时候，大多数 Web 浏览器会显示一个对话框以确认保存结果文件 *sp.sql*。包含在这个文件里的 SQL 命令可以把当前数据库里的所有 SP 重新创建出来。SP 备份文件的格式和部分内容如下所示：

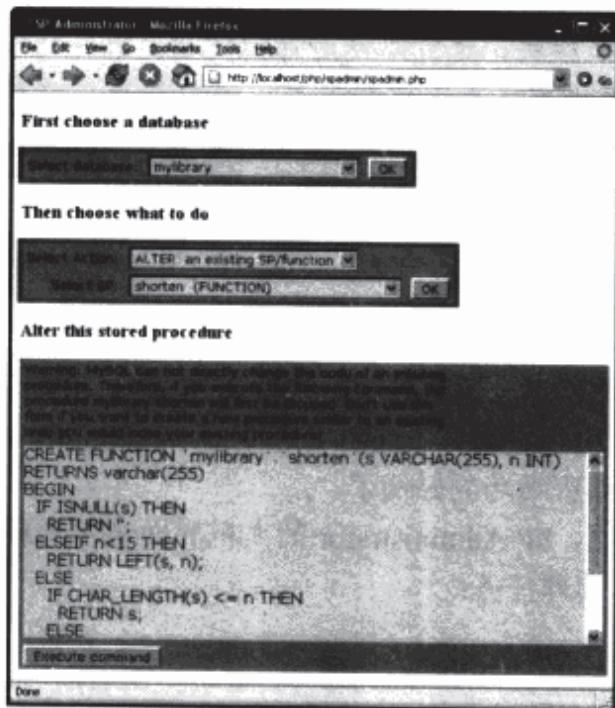


图 15-17 修改一个 SP

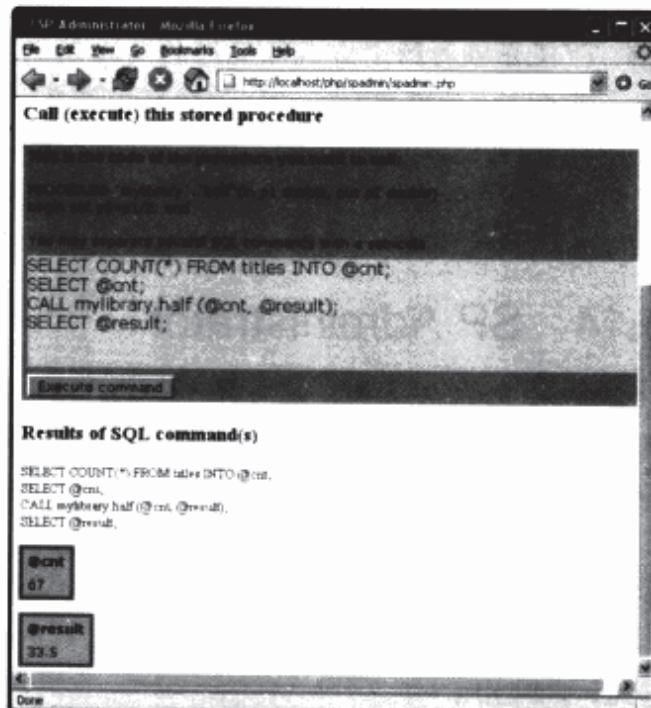


图 15-18 试用一个 SP

```

DELIMITER $$

DROP PROCEDURE IF EXISTS mylibrary.cursortest$$
CREATE PROCEDURE 'mylibrary'.'cursortest'(OUT avg_len DOUBLE)
BEGIN
    DECLARE t, subt VARCHAR(100);
    ...
    SET avg_len = n/cnt;
END$$

DROP FUNCTION IF EXISTS mylibrary.faculty$$
CREATE FUNCTION 'mylibrary'.'faculty'(n BIGINT) RETURNS BIGINT
BEGIN
    ...
END$$
DELIMITER ;

```

选择 Restore SPs (恢复 SP) 操作打开一个表单，可以在那里读入备份文件。选择的文件必须有如上所示的格式。

注意 SP Administrator程序的备份恢复功能将忽略备份文件里的`DELIMITER`指令，它总是使用`$$`作为SQL命令之间的分隔符。

SP Administrator程序无法为SP的备份和恢复操作另行指定一个字符集。它总是使用`Latin1`字符集。

用 SP Administrator 程序制作的 SP 备份文件还可以用 `mysql` 命令来恢复：

```
user$ mysql -u root -p --default-characterset=latin1 databasename < sp.sql
Password: *****
```

15.14.3 SP Administrator 代码

SP Administrator 的代码分别存放在如表 15-9 所示的文件里。

表 15-9 SP Administrator 代码的存放位置

文件名	内 容
spadmin.php	起始页面
spfunctions.php	辅助函数
backup.php	把当前数据库里的所有 SP 备份为文本文件 sp.sql 并传输到 Web 浏览器（文件下载）
password.php	MySQL 登录数据
formfunctions.php	用来输出 HTML 表单的函数
sp.css	用来对 HTML 表单和表格进行排版的 CSS 文件

这个程序使用了一些会话变量来保存全部 3 个表单的输入，这样它们在页面切换时就不会丢失。表单数据的处理工作和会话变量的管理工作由 spadmin.php 脚本里的代码负责，实际工作由 spfunctions.php 文件里的函数具体完成。下面将对它们当中最有意思的几个进行介绍。

1. SP下拉列表

*show_sp_form()*函数负责显示一个列表供用户选择一个 SP 和一个操作动作（如 *ALTER* 等），这个函数需要调用来自 formfunctions.php 文件的许多 *form_xxx()* 函数。当前数据库里的现有 SP 清单是通过对数据表 *information_schema.routines* 进行一次 *SELECT* 查询而得到的，在 *SELECT* 查询命令里用 SQL 函数 *CONCAT()* 把各 SP 的名字和类型合并为一个 *name_type* 格式的字符串。这个字符串将被输入为 SP 下拉列表里各有关元素的一个 *value* 属性，并在用户选中一个 SP 时被拆分为它的各组成部分。

```
// example file spfunctions.php
function show_sp_form($mysqli, $formdata, $dbname) {
    form_start("spadmin.php");
    form_new_line();
    form_label("Select Action:");
    $rows = array(array('ALTER an existing SP/function', 'alter'),
                  array('CALL a procedure/function', 'call'),
                  ...
                  array('Restore SPs', 'restore'));
    form_list("action", $rows, $formdata["action"],
              "[choose an action]");
    form_empty_cell();
    form_end_line();
    form_new_line();
    form_label("Select SP:");
    $sql = "SELECT CONCAT(routine_name, ' (' , routine_type, ')'
           AS displayname,
           CONCAT(routine_type, '_', routine_name) AS internalname
      FROM information_schema.routines
     WHERE routine_schema = '$dbname'
       ORDER BY displayname";
    $rows = queryArray($mysqli, $sql);
    form_list("spname", $rows, array_item($formdata, "spname"),
              "[choose a SP]");
    form_button("btnAction", "OK");
    form_end_line();
    form_end();
}
```

2. 执行和测试一个SP

为了测试一个 SP，用户可以在专门为为此提供的表单里输入任意数量的 SQL 命令（如图 15-18 所示）。这些 SQL 命令通过 *\$cmd* 变量被传递到 *test_sp()* 函数，这个函数先依次显示各条 SQL 命令的代码（通过

调用 `spfunctions.php` 文件里的 `printsql()` 函数），然后调用 `$mysqli->multi_query()` 方法执行它们。

如果 PHP 在这些 SQL 命令的执行期间检测到一个错误（就目前而言，PHP 只能检测到第一条 SQL 命令是否执行出错），`printerror()` 函数就会把相应的出错信息回显给用户。接下来，此前已经得到的所有查询结果将被 `store_result()` 和 `next_result()` 方法依次读入并被 `show_table()` 函数输出到一个 HTML 表格里供用户查看。`printerror()` 和 `show_table()` 函数是在 `spfunctions.php` 文件里定义的辅助函数。）

```
// example file spfunctions.php
function test_sp($mysqli, $dbname, $cmd) {
    echo "<h3>Results of SQL command(s)</h3>\n";
    $mysqli->select_db($dbname);
    printsql($cmd);
    $ok = $mysqli->multi_query($cmd);
    if($mysqli->info)
        printerror("Info: " . $mysqli->info);
    if($mysqli->warning_count)
        printerror("Warnings: " . $mysqli->warning_count);
    if($mysqli->errno)
        printerror("Error: " . $mysqli->error);
    if($ok) {
        do {
            $result = $mysqli->store_result();
            if($result) {
                show_table($result);
                $result->close();
            }
        } while($mysqli->next_result());
    }
}
```

3. 备份SP（文件下载）

`backup_sps()` 函数将返回一个字符串，其内容是本次 SP 备份操作所备份下来的所有 SQL 命令。这个函数会先查询出一份当前数据库里所有 SP（SP 过程和 SP 函数）的清单，然后为清单里的每一个 SP 执行一次 `SHOW CREATE FUNCTION / PROCEDURE` 命令，这些 `SHOW CREATE` 命令的结果将被收集到变量 `$result` 里。在备份期间，每一条 `CREATE FUNCTION / PROCEDURE` 的前面将插入一条 `DROP IF EXISTS` 命令，这意味着在以后恢复 SP 备份时将先删除现有的 SP（没有出错信息）再重新定义它。

```
// example file spfunctions.sp
$separator = "##";
function backup_sps($mysqli, $dbname) {
    global $separator;
    $sep = $separator . "\n";
    $result = "DELIMITER $sep";
    $sql = "SELECT routine_name, routine_type
           FROM information_schema.routines
          WHERE routine_schema = '$dbname'";
    $rows = queryArray($mysqli, $sql);
    foreach($rows as $row) {
        $spname = $row[0];
        $type = $row[1];
        if($type=="FUNCTION") {
            $sql = "SHOW CREATE FUNCTION $dbname.$spname";
            $create = queryArray($mysqli, $sql);
            $result .= "DROP FUNCTION IF EXISTS $dbname.$spname" . $sep .
                       $create[0]["Create Function"] . $sep;
        }
        else {
            $sql = "SHOW CREATE PROCEDURE $dbname.$spname";
```

```

    $create = queryArray($mysqli, $sql);
    $result .= "DROP PROCEDURE IF EXISTS $dbname.$pname" . $sep .
        $create[0]["Create Procedure"] . $sep;
    }
}
$result .= "DELIMITER ;\n";
return $result;
}

```

backup_sps() 函数是通过 HTML 页面 *backup.php?dbname=name* 调用的。*backup.php* 脚本负责实现 SP 备份文件的下载操作，这个脚本创建的不是一个 HTML 页面，而是利用了 3 个 *header()* 函数通知位于客户端的 Web 浏览器（以及沿途的 Web 代理）：我发给你的数据是一个文本文件（*Content-Type: text/plain* 类型）；应该把这个文本文件当做一个电子邮件附件（*attachment*）来处理；这个文件不适合存放在临时缓存区里（*Expires* 指令）。在完成这些准备工作之后，这个脚本用一条 *echo* 命令输出了实际备份数据。

```

// example file backup.php
require_once 'spfunctions.php';
require_once 'password.php';
$dbname = array_item($_REQUEST, 'dbname');
if (!$dbname)
    exit;
else
    $dbname = urldecode($dbname);
$mysqli = @new mysqli($mysqlhost, $mysqluser, $mysqlpasswd);
if ($mysqli_connect_errno()) {
    html_start('MySQL connection error');
    printerror("Sorry, no connection!");
    html_end();
    exit();
}
header('Content-Type: text/plain');
header('Content-Disposition: attachment; filename="sp.sql"');
header('Expires: ' . gmdate('D, d M Y H:i:s') . ' GMT');
echo backup_sps($mysqli, $dbname);

```

下面讨论 *backup.php* 页面如何被调用。评估成组数据过程中，在 *spadmin.php* 中处理这个调用。存储的数据库名被 *urlencode* 编码，所以任何可能出现的特殊字符都不会引起问题。

```

// example file spadmin.php
if($action=='backup') {
    header("Location: " . baseurl() . "/backup.php?dbname=" .
        urlencode($dbname));
    exit;
}

```

恢复SP（文件上传）

备份是把数据从服务器传输到客户端。恢复备份则是把数据从客户端传输到服务器。这里使用了一个带有 *<input>* 标记的 HTML 表单，这个 *<input>* 标记的 *type="file"* 属性会让客户端的 Web 浏览器在该标记生成的输入框的旁边显示一个 *Browse*（浏览）或 *Search*（搜索）按钮，用户可以通过这个按钮去选择本地文件系统里的一个文件作为上传文件——单击这个按钮将打开一个文件选择对话框。这个表单是用 *show_restore_form()* 函数创建的，这里没有给出它的代码。下面是这个 HTML 表单的 HTML 代码（为简明起见，这里省略了对这个表单进行排版的有关代码）：

```

<form method="post" action="spadmin.php"
      enctype="multipart/form-data">
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
<input type="file" size="40" name="spfile" />
<input type="submit" value="Restore" name="form[btnRestore]" />
</form>

```

对表单数据的处理发生在 spadmin.php 脚本里。如果数据经检查没有发现错误，下面这些代码就会把备份文件的内容读入 *sql* 变量并把这个变量传递给 *restore_sps()* 函数。

```
// example file spadmin.php
$formdata = array_item($_POST, "form");
if(array_item($formdata, 'btnRestore')) {
    $spfile = array_item($_FILES, 'spfile');
    if($spfile && is_array($spfile)){
        $uperr = $spfile['error']; // error number (0 = no error)
        $tmpfile = $spfile['tmp_name']; // local, temporary file
        $size = $spfile['size'];
        if(!$uperr && $size>0 && $tmpfile && is_uploaded_file($tmpfile)) {
            $file = fopen($tmpfile, "rb"); // open file
            $sql = fread($file, $size); // read file
            fclose($file);
            restore_sps($mysqli, $sql);
        }
    }
}
```

restore_sps() 函数将调用 *explode()* 函数把字符串分断为一条一条的 SQL 命令，在分断字符串时使用的分隔符是变量 *\$separator* 的值。随后，除空行和包含着 *DELIMITER* 字样的命令以外，其余的命令将依次得到执行。

restore_sps() 函数还用到了两个辅助函数（它们的代码也在 spfunctions.php 文件里）：*printsql()*，把 SQL 命令的代码输出到网站（作为反馈）；*execute()*，调用 *mysqli->real_query()* 方法执行 SQL 命令，并在必要时显示一条出错信息。

```
// example file spfunctions.php
function restore_sps($mysqli, $sql) {
    global $separator;
    echo "<h3>Restore SPs</h3>\n";
    $cmds = explode($separator, $sql);
    foreach($cmds as $cmd)
        if(trim($cmd)!="" && strpos($cmd, "DELIMITER")!==FALSE) {
            printsql($cmd);
            $ok = execute($mysqli, $cmd);
            if($ok) echo "<font color=\"#00cc00\">OK</font>\n";
        }
}
```

第 16 章

Perl



多 年以来，Perl 一直是 UNIX/Linux 程序员最喜爱的脚本语言之一。同时，作为一种 CGI 脚本程序设计语言，Perl 在实现动态网站方面也一直扮演着重要的角色。本章将简要地介绍一下如何使用 Perl 去访问 MySQL 数据库，并通过几个示例来展示 Perl 的广阔应用前景。

16.1 编程技巧

本章内容需要假设 *DBI* 和 *DBD::mysql* 模块已经安装在了计算机里并能够正常运行。这两个模块的安装步骤可以在本书第 1 章里查到。

对这两个 Perl 模块的详细介绍可以在 MySQL 官方文档里查到（见“MySQL APIs”章节）。此外，通过 *perldoc* 命令也可以获得大量的在线帮助：

```
perldoc DBI
perldoc DBI::FAQ
perldoc DBD::mysql
```

这 3 份帮助文档的 HTML 版本可以在许多网站找到，如 <http://search.cpan.org/>。

最后，对 *DBI* 函数和属性的简要说明可以在本书第 23 章找到。

16.1.1 *DBI* 和 *DBD::mysql* 模块

Perl 脚本对 MySQL 数据库的访问需要通过 *DBI* 和 *DBD::mysql* 模块来实现。（*DBI* 是 *database interface* 的缩写，意思是数据库接口；*DBD* 是 *database driver* 的缩写，意思是数据库驱动程序。）*DBI* 是一种通用的数据库编程接口，与具体的数据库系统无关，所以 *DBI* 可以用来为 Oracle、DB2 等不同数据库系统编写应用软件。一般来说，为不同的数据库系统实现同样功能的 *DBI* 代码是一样的；这意味着 Perl 代码无须修改（除了用来连接各个数据库的 *datasource* 字符串）就可以拿到另一种数据库系统上使用。

DBD::mysql 是 *DBI* 与 MySQL 进行通信的驱动模块。（*DBI* 与其他的数据库系统的通信也需要通过类似的驱动模块来进行。）*DBI* 具体使用的是哪一种驱动模块要由建立数据库连接（*connect()*方法）时给出的数据源字符串来决定。在 Perl 脚本的开头只须加上一条 *use DBI* 指令就可以使用 *DBI* 进行编程，细节问题可以等到遇见时再解决。

DBI 与具体的数据库无关，*DBD::mysql* 模块里的大多数函数却是 MySQL 专用的。这是因为 *DBI* 接口只是各种数据库 API 接口的最大交集，所以它覆盖的功能相对很少。使用 *DBD::mysql* 模块提供的 MySQL 专用函数可以简化用 Perl 语言编写 MySQL 应用程序的工作，但这么做的另一个后果是编写出来的代码不太容易移植到另一种数据库系统上去。

DBI 和 *DBD* 都是面向对象的模块，所以它们提供的函数大都以某种数据对象（Perl 文档称为“句柄”）的方法的形式出现。比如，*DBI->connect()*方法将返回一个指向给定 *database*（数据库）对象的句柄（Perl 程序员的习惯做法是把这种句柄保存在变量 *\$dbh* 里），对给定数据库进行各种操作的其他函数将作为这个 *database* 对象的方法来执行；比如，*\$dbh->do("INSERT ...")*方法将在句柄 *\$dbh* 所代表的那个数据库里执行一条 *INSERT* 命令。

16.1.2 与数据库建立连接

数据库连接由 *DBI* 方法 *connect()*负责建立。这个方法的第一个参数是一个字符串，它给出了数据库的类型和数据库服务器主机的计算机名（或 *localhost*）。这个字符串的语法可以从下面这个例子里看出来。这个方法的第 2 个和第 3 个参数必须是用户名和密码。

第 4 个参数是可选的，这个参数可以用来设置各种属性。比如，属性 '*RaiseError*'=>*1* 的效果是：如果未能与数据库建立连接，Perl 脚本将退出执行并返回一条出错信息。可以用做第 4 个参数的各种连接属性可以在第 23 章找到。

```
use DBI;
$datasource = "DBI:mysql:database=mylibrary;host=localhost";
$user = "root";
$passw = "xxx";
$dbh = DBI->connect($datasource, $user, $passw,
    {'RaiseError' => 1});
```

在基于 RHEL 和 Fedora 的系统上，SELinux 往往会被配置成不允许 Apache 和 Perl 去访问 MySQL 套接字文件的样子。如果真是那样，在调用 *connect()*方法的时候就不能把 *localhost* 用做 MySQL 服务器的主机名，而是必须给出它的真实名字，这将使得 Perl 脚本与 MySQL 服务器之间的通信使用 TCP/IP 协议，而不是通过套接字文件来进行。

提示 *DBD::mysql* 模块的早期版本不支持 MySQL 从 4.1 版开始使用的新密码系统（相应的出错消息是 *Client does not support authentication protocol requested by server*，意思是“客户不支持服务器要求使用的身份验证协议”）。如果对是否安装 *DBD::mysql* 模块的新版本没有决定权，就只能在启动 MySQL 服务器的时候使用 *--old-passwords* 选项；或者使用 SQL 函数 *OLD_PASSWORD()* 来插入新记录。与这个问题有关的更多细节见第 11 章。

1. 指定一个配置文件（仅适用于 UNIX/Linux）

如果使用的是 UNIX/Linux 系统，还可以把数据源字符串（即上面代码中的 *\$datasoure* 变量）里的用户名、密码以及必要的连接选项放在一个配置文件里。这是一种很实用的做法——只要把不同内容的数据源配置文件放在不同用户的登录子目录里，就可以让他们使用不同的用户名、密码和连接选项去连接数据库服务器。（创建数据源配置文件的具体步骤可以在第 22 章找到。）

在下面的例子中，选项 *mysql_read_default_file* 给出了一个数据源配置文件的相对文件路径（相对于环境变量 *HOME* 给出的登录子目录），选项 *mysql_read_default_group* 的意思是在连接数据库时必须使用给定数据源配置文件里的 *[mygroup]* 段落所给出的参数——如果不给出这个选项，*connect()*方法将自动使用 *[client]* 段落所给出的参数去连接数据库。请注意，在下面这段代码里，*connect()*方法的第 2 个和第 3 个参数都是 *undef*（Perl 语言里的空值），它们的含义可不是“无须给出用户名和密码”，而是“用户名和密码在数据源配置文件里”：

```
$datasource = "DBI:mysql:database=mylibrary;" .
    "mysql_read_default_file=$ENV{HOME}/.my.cnf;" .
    "mysql_read_default_group=mygroup";
$dbh = DBI->connect($datasource, undef, undef, {'RaiseError' => 1});
```

与上面这段代码相对应的数据源配置文件~/.my.cnf 的内容如下所示：

```
[mygroup]
user=root
password=xxx
host=uranus.sol
```

请注意，每位用户的数据源配置文件的访问权限应该设置成只允许这位用户读取。

在笔者在 Windows 环境下进行的测试中，*DBI->connect()*方法无法读取数据源配置文件。看来这个功能只能在 UNIX/Linux 环境下使用（Perl 文档没有对此做出明确的说明）。

2. 断开与数据库的连接

对数据库的所有操作都必须通过变量*\$dbh*（或是从它派生出来的其他变量）进行。在完成数据库操作之后，应该尽早使用 *disconnect()*方法关闭不再需要的数据库连接：

```
$dbh->disconnect();
```

3. 与数据库建立永久性连接

在需要连续进行多次数据库操作的场合，反复多次地建立与数据库的连接未免有些麻烦；这时候，要是能建立一条长久的连接、等所有的数据库操作全部完成后再关闭它无疑更有效率。Perl DBI 模块本身没有提供任何可以用来建立这种永久性连接的机制，但我们可以利用其他办法来获得一条这样的连接。

如果是通过 Apache 模块 *mod_perl* 来执行 CGI 脚本的（实际情况也往往如此，因为这可以提高 CGI 脚本的执行效率），就可以利用 Perl 模块 *Apache::DBI*，让某个脚本建立的 MySQL 连接在该脚本结束后仍保持在打开状态，供下一个需要使用一条同样类型的 MySQL 连接的脚本使用。具体做法很简单：只须在脚本里的所有 *use DBI* 命令之前插入一条 *use Apache::DBI* 命令即可，其他地方都用不着修改。（*Apache::DBI* 将把 DBI 的 *connect()*方法替换为它自己的版本。）有关 *Apache::DBI* 模块的更多消息可以在下面这个网址处找到：<http://search.cpan.org/~abh/Apache-DBI/DBI.pm>。

凡事有利就有弊，永久性连接也不例外：永久性连接在脚本结束后的工作状态无法预知；最近执行的脚本可能会改变某些（系统）变量；事务的自动提交（*autocommit*）模式有可能被设置为不是下一个脚本所需要的状态；等等。

16.1.3 执行 SQL 命令

1. 不会返回结果记录的SQL命令

不会返回结果记录的 SQL 命令应该用 *do()*方法来执行：

```
$n = $dbh->do("INSERT INTO authors (authName) " .
    "VALUES ('New author')");
```

如果执行成功，*do()*将返回该命令改变的记录个数，它的其他返回值还有：

- "0E0": 没有改变任何记录。这个字符串可以用一个算术操作（如：*\$n+=0*）转换为一个数值。
- -1: 被改变的记录个数无法确定。
- *undef*: 执行出错。

2. 确定AUTO_INCREMENT值

在执行完 *INSERT*命令后，经常需要确定新插入的那条记录的 *AUTO_INCREMENT* 值。这个任务

可以用 *DBD::mysql* 模块提供的 *mysql_insertid* 属性来完成:

```
$id = $dbh->{ 'mysql_insertid' };
```

注解 *mysql_insertid* 属性没有可移植性; 换句话说, 它只适用于 MySQL 数据库。如果想让自己的代码也能用于另一种数据库系统, 就必须采用其他办法来完成这项任务。

3. SELECT查询

会返回一些结果记录的 SQL 命令(通常是 *SELECT* 命令)不能用 *do()* 方法来执行。必须先用 *prepare()* 方法对它们进行一下预处理。

从 4.1 版开始, MySQL 已经能够以更高效的方式来执行那些经过预处理的命令, 但目前只有 *DBD::mysql* 模块的最新版本才能使用 MySQL 的这项新功能。虽然老版本的 *DBD::mysql* 模块也提供了 *prepare()* 方法, 但它还是要慢一些。

prepare() 方法的返回值是一个 *statement* (语句) 类型的句柄, 与当前查询有关的所有其他操作都必须通过这个句柄来进行, 就连实际执行当前查询命令的 *execute()* 方法调用也不能例外。(如果在执行 SQL 查询命令的过程中出现错误, *execute()* 方法的返回值将是 *undef*。)

```
$sth = $dbh->prepare("SELECT * FROM titles LIMIT 5");
$sth->execute();
```

对结果记录的处理和输出也必须通过 *\$sth* 句柄来进行, 我们将在稍后对此做专题讨论。在结果记录的处理工作全部完成后, 绑定在 *\$sth* 句柄上的资源应该尽早用 *finish()* 方法释放:

```
$sth->finish();      # delete query object
```

4. 参数里有通配符的SQL查询

在实际工作中, 经常会遇到需要用不同的参数 (*WHERE id=1*、*WHERE id=3* 等) 反复执行同一条 SQL 查询命令的情况。对于这类情况, DBI 提供了这样一个办法: 把 *prepare()* 方法里的各有关参数分别替换为一个问号 (?), 这个问号的作用相当于一个通配符。每个问号对应着一个参数, 这些参数必须在随后的 *execute()* 调用命令里给出。请注意, 在 *execute()* 调用命令里, 参数值的给出顺序必须与 *prepare()* 方法里的问号通配符保持一致, 例如:

```
$sth = $dbh->prepare("SELECT * FROM titles ".
    "WHERE catID=? AND publID=?");
$sth->execute(1, 1);  # titles with catID=1 und publID=2
...
# evaluate results
$sth->execute(1, 2);  # titles with catID=1 und publID=2
...
# evaluate results
$sth->finish();       # delete query object
```

在使用问号通配符的时候, 必须注意以下几点: 首先, 不要在 *prepare()* 方法里给 SQL 命令中的问号通配符加上引号(即使它们出现在字符串里也必须如此); 其次, 不要给 *execute()* 方法里的参数值加上引号——DBI 模块会自动调用 *quote()* 方法来处理这些参数值, 该方法会给参数值里的字符串加上单引号并在特殊字符(如'、\等)的前面加上一个反斜线字符 (\) 作为前缀。如果需要把 *NULL* 值传递给一条 SQL 命令, 在调用 *execute()* 方法时必须在相应位置给出 *undef*(Perl 语言里的空值)——*quote()* 方法会把它转换为 *NULL*(SQL 语言里的空值)。

比如说, 如果在一个 Perl 脚本里执行了下面的命令:

```
$sth = $dbh->prepare("INSERT INTO publishers (publName) ".
    "VALUES (?)");
$sth->execute("O'Reilly");
```

实际传输给 MySQL 服务器去执行的 SQL 命令如下所示：

```
INSERT INTO publishers (publName) VALUES ('O'Reilly')
```

从这个例子可以看出，*prepare()*和*execute()*方法不仅可以用来预处理和执行 *SELECT* 查询命令，还适用于其他任何一种需要反复多次执行的 SQL 查询命令。

提示 通配符也可以用在由*do()*方法执行的SQL命令里。这么做不一定能提高那些SQL命令的执行速度，但可以改善代码的可读性——因为字符串中的单引号和其他特殊字符将由DBI自动调用*quote()*方法来处理，程序员就用不着操心这些事情了：

```
$dbh->do("INSERT INTO table (cola, colb)
    VALUES (?, ?, undef, ($data1, $data2));")
```

16.1.4 处理 *SELECT* 查询结果

如果一条 *SELECT* 查询命令在执行时没有出现任何错误（这可以用 *if(defined(\$sth))* 来测试），它的查询结果就可以通过它的语句（*statement* 类型）句柄（即 *\$sth*）来读取。有好几种不同的 DBI 方法可以用来完成这一任务。

1. 用 *fetch()* 和 *fetchrow_array()* 方法来读取查询结果里的数据记录

fetch() 和 *fetchrow_array()* 方法是等价的，它们将返回一个数组，其内容是下一条结果记录各个字段的值。在这个数组里，SQL 语言里的空值 *NULL* 将被表示为 Perl 语言里的空值 *undef*。如果已经到达最后一条结果记录或者在读取下一条结果记录时出现错误（这两种情况只能根据 *\$sth->err()* 方法的返回值来区分），这个数组将是空的。下面这个例子在执行完一条 *SELECT* 查询命令后，把它的查询结果逐行地显示了出来，查询结果里的 *NULL* 值将被显示为字符串<*NULL*>：

```
$sth = $dbh->prepare("SELECT * FROM titles LIMIT 5");
$sth->execute();
while(@row = $sth->fetchrow_array()) { # process all records
    foreach $field (@row) { # each field
        if(defined($field)) { # test whether NULL
            print "$field\t";
        } else {
            print "<NULL>\t";
        }
    }
    print "\n";
}
$sth->finish();
```

用一个 *foreach* 循环来提取结果记录里的各个字段（数据列）并不是唯一的办法。还可以用读取数组元素 *\$row[n]* 的方式来提取某个特定的字段（必须指定 *where=0* 为第 1 个字段）。另一种做法是把结果记录的所有字段一次提取到一个变量数组里——当然，各有关变量的顺序必须与各有关字段逐一对应：

```
($titleID, $title, ...) = @row;
```

2. 如果查询结果是一个离散值

如果从一开始就确切地知道将要执行的查询命令只会返回一个离散值（如 *SELECT COUNT(*) FROM ...*），用一个循环把结果记录逐条提取到一个数组里显然没有必要。这时候，可以简单地直接把 *fetchrow_array()* 方法的返回值赋值给一个离散变量：

```
$sth = $dbh->prepare("SELECT COUNT(*) FROM titles");
$sth->execute();
$result = $sth->fetchrow_array();
print "$result\n";
$sth->finish();
```

注意 只有在 *SELECT* 查询结果肯定只包含一个字段（数据列）的时候才能把 *fetchrow_array()* 方法用在一个离散上下文里。如果查询结果有一个以上的字段，*fetchrow_array()* 方法的离散结果将无法预知，它可能是第一个字段，也可能是最后一个字段。

除此之外，把 *fetchrow_array()* 用在一个离散上下文里（比如直接赋值给一个 Perl 离散变量 *\$field*）这种做法本身还有一些需要注意的地方：*fetchrow_array()* 方法会因为三种原因——SQL 查询结果本身是 NULL 值、已经到达最后一条结果记录、发生错误——而返回 *undef* 值，而这意味着必须在离散变量 *\$field* 包含 *undef* 值时对这三种原因进行测试和处理。

如果觉得用上面那么多代码来提取一个离散值有些小题大做，可以使用如下所示的简化版本：

```
$result = $dbh->selectrow_array("SELECT COUNT(*) ... ");
```

3. 把结果记录里的各个字段绑定到Perl变量

在处理 *SELECT* 查询结果的时候，以人工方式反复地把每一条结果记录的各有关字段（数据列）提取到一些变量里未免有些麻烦。其实这个步骤是可以自动完成的，这不仅可以提高效率，还可以改善程序代码的可读性。其具体做法是：用 *bind_col()* 方法把各有关字段绑定在一些 Perl 变量上。绑定之后，每调用一次 *fetchrow_array()*，下一条结果记录的各有关字段的值就会被自动提取到与之关联的变量里。这里要提醒大家注意以下几点：*bind_col()* 方法必须在 *execute()* 方法之前得到调用才能有上述效果；如果绑定操作执行出错，*bind_col()* 方法的返回值将是 *false*；有关字段（数据列）的计数值从 1 开始（这与 Perl 语言从 0 开始计数的常见做法不一样）：

```
$sth = $dbh->prepare("SELECT titleID, title FROM titles");
$sth->execute();
$sth->bind_col(1, \$titleID);
$sth->bind_col(2, \$title);
while($sth->fetchrow_array()) {
    print "$title $titleID\n";
}
$sth->finish();
$dbh->disconnect();
```

把各有关字段（数据列）分别绑定到各个变量上还是有些麻烦。可以用 *bind_columns()* 方法一次完成对所有字段（数据列）的绑定——但那些变量的顺序和个数一定要正确：

```
$sth->bind_columns(\$titleID, \$title);
```

4. 统计结果记录的总个数

DBI 模块没有提供可以用来统计 *SELECT* 查询命令所返回的结果记录总共有多少条的方法。（这是因为许多数据库服务器只在必要时才会把下一批结果记录传输给 *DBI* 模块的缘故。）如果确实需要知道结果记录总共有多少条，有以下几个办法：

- 在对结果记录进行处理的同时对它们的总数进行统计。（这个办法显然不适用于必须提前知道这一统计结果的场合。）
- 多进行一次 *SELECT COUNT(*) ...* 查询。根据具体的应用情况，这么做的代价可能比较高。
- 在 *execute()* 方法成功返回之后，用 *DBI* 方法 *fetchall_arrayref()*（我们马上就要介绍到它）一次性

地把所有的查询结果取回到一个本地数组里来，这样就可以轻而易举地统计出它们的总数了。

- 在 `execute()`方法成功返回之后，用 `$sth->rows()` 方法来统计结果记录的总数。这个办法虽然看起来是个通用的解决方案，但存在着以下几点不足。

首先，`rows()`方法没有可移植性。（根据 `DBI` 文档，`rows()`方法只适合用来统计 `UPDATE` 或 `DELETE` 命令总共改变了多少条记录，不适合用来统计 `SELECT` 查询结果里的记录总数。）

其次，`rows()`方法只在 `DBD::mysql` 驱动模块没有使用 `mysql_use_result` 属性去执行 SQL 查询命令时才能返回正确的结果。虽说 `DBD::mysql` 驱动模块的默认设置就是不使用这个属性，但利用 `rows()`方法去统计结果记录总数的做法并不总是那么有效率。

其次，`rows()`方法只在 `DBD::mysql` 驱动模块没有使用 `mysql_use_result` 属性去执行 SQL 查询时才能返回正确的结果。

5. 确定各结果数据列的名字和其他元信息

如果想把 SQL 查询结果输出为一个表格，那不仅需要获得有关数据本身，还需要获得有关数据的元信息（数据列的名字、数据类型等）。这些信息可以通过 `DBI` 模块为 `$sth` 句柄准备的以下几种属性来获得。

- `$sth->{'NUM_OF_FIELDS'}`：返回 SQL 查询结果中的数据列个数。
- `$sth->{'NAME'}`：返回一个数组指针，该数组的各个元素分别存放着各结果数据列的名字。
- `$sth->{'NAME_lc'}`或`$sth->{'NAME_uc'}`：同上，只是各结果数据列的名字全部是小写或大写字母。
- `$sth->{'NULLABLE'}`：返回一个数组指针，该数组的各个元素分别表明各结果数据列是否允许包含 `NULL` 值。
- `$sth->{'PRECISION'}`或`$sth->{'SCALE'}`：各返回一个数组指针，这两个数组的各个元素分别给出了各结果数据列的最大字符长度或小数点位置。
- `$sth->{'TYPE'}`：返回一个数组指针，该数组的各个元素以数字形式分别表明了各结果数据列的数据类型。

下面是一个利用以上属性查出结果数据列元信息的例子：

```
$sth = $dbh->prepare("SELECT * FROM testall");
$sth->execute();
for($i=0; $i < $sth->{'NUM_OF_FIELDS'}; $i++) {
    print @{$sth->{'NAME'}}[$i] . " ";
    @{$sth->{'TYPE'}}[$i] . "\n";
}
```

提示 `DBD::mysql` 驱动模块还可以提供一些仅适用于 MySQL 数据库系统的元信息。我们将在本章稍后讨论到 `DBI` 模块中的 MySQL 独有（因而不具备可移植性的）扩展功能时再对这些属性进行介绍。

6. 用 `fetchrow_arrayref()` 方法读取结果记录

`fetchrow_arrayref()` 方法与 `fetchrow_array()` 方法很相似，唯一的区别是这次返回的是一个数组指针而不是数组本身。如果已经到达最后一条结果记录或发生错误，这个方法将返回 `undef`。

```
while(my $arrayref = $sth->fetchrow_arrayref()) {
    foreach $field (@{$arrayref}) {
        ... as before
    }
}
```

7. 用`fetchrow_hashref()`方法读取结果记录

`fetchrow_hashref()`方法将返回一个关联数组（散列表），其内容是下一条结果记录各个字段的值。如果已经到达最后一条结果记录或发生错误，这个方法将返回 `undef`。

此时，对结果记录各字段的访问需要以`$row->{'columnname'}`的形式来进行；注意，`columnname`区分字母的大小写情况。下面是一个例子：

```
$sth = $dbh->prepare("SELECT title, titleID FROM titles LIMIT 5");
$sth->execute();
while($row = $sth->fetchrow_hashref()) {
    print "$row->{'title'}, $row->{'titleID'}\n";
}
$sth->finish();
$dbh->disconnect();
```

如果在调用`fetchrow_hashref()`方法时给出了可选参数`"NAME_lc` 或 `NAME_uc`，它返回的关联数组里的全部散列关键字（数组下标）都将被转换为小写或大写字母。

8. 用`fetchall_arrayref()`方法读取全部结果记录

前面介绍的几个`fetchrow_xxx()`方法都存在着这样一个不足：只能依次读取下一条结果记录，每条结果记录只能被读取一次。换句话说，不能在结果集里随意前后移动。在其他的数据库系统上，这至少还有减少资源占用的好处。但在 MySQL 数据库系统上，前面介绍的那几个`fetchrow_xxx()`方法¹连这点好处都体现不出来——在默认的情况下，MySQL 服务器会把所有的结果记录立刻发送到客户端；而这种情况只有使用了`mysql_use_result`属性才能避免（对`mysql_use_result`属性的讨论见本章稍后的有关内容）。

如果需要按任意顺序去访问所有的结果记录或者需要对它们进行任意多次的访问，就应该使用这里介绍的`fetchall_arrayref()`方法一次性地把所有的结果记录全部取回到客户端。这个方法将返回一个指针数组（即数组元素是一些指针），其中的每个指针分别指向一个数组，那些数组中的每一个分别对应着一条结果记录。这意味着如果想访问一条特定结果记录的一个特定字段（数据列），就需要通过`$result->[$row][$col]`的方式来进行（请注意，`$row` 和 `$col` 这两个下标变量都是从 0 开始计数的）：

```
$sth = $dbh->prepare("SELECT titleID, title FROM titles");
$sth->execute();
$result = $sth->fetchall_arrayref();
print "$result->[2][5]\n"; # third record, sixth column
$sth->finish();
$dbh->disconnect();
```

查询结果中的记录总数和数据列总数可以像下面这样来确定：

```
$rows = @{$result};
$cols = @{$result->[0]};
```

`fetchall_arrayref()`方法还可以有一个可选的参数，这个参数控制着`fetchall_arrayref()`方法将读取查询结果的哪几个数据列以及如何组织这些数据。下面这条语句将读取查询结果的第一个、第四个和最后一个数据列：

```
$result = $sth->fetchall_arrayref([0,3,-1]);
```

下面这条语句将读取查询结果的全部数据列，它仍将返回一个指针数组，但那些指针指向的是一些散列表，每个散列表分别对应着一条结果记录。这意味着必须通过`$result->[$row]->{'columnname'}`的方式去访问一条特定结果记录的一个特定字段（数据列），比如`$result->[3]->{'titleID'}`这样：

1. 原书这里有误，从前后文看，这里不应该像原文那样是“`fetchall_arrayref()`方法”。——译者注

```
$result = $sth->fetchall_arrayref({});
```

最后一个例子返回的也是一个由一些指向散列表的指针构成的数组，只不过这次只返回了名为 *titleID* 和 *title* 的两个数据列：

```
$result = $sth->fetchall_arrayref({titleID=>1, title=>1});
```

因为用 *fetchall_arrayref()* 方法一次取回全部结果记录的等待时间可能会比较长，所以在某些场合，在使用一条 *SELECT* 命令把需要的数据从数据库里全部查询出来之后，利用 *fetchall_arrayref()* 方法和它的可选参数有选择地只取回一部分数据列还是有一定的实用价值的。注意，在每次 SQL 查询之后只能发出一次 *fetchall_arrayref()* 调用，如果想反复多次地调用这个方法，就必须在每次调用它之前进行一次 *execute()* 调用。

如果 SQL 查询命令没有返回任何结果，*fetchall_arrayref()* 方法将返回一个空数组。如果在对数据库进行查询时发生错误，\$result 将只包含着在发生错误前已经查询出来的所有结果记录。如果在连接数据库时没有使用 '*RaiseError*'=>1 属性，就应该在 *fetchall_arrayref()* 调用语句的后面安排一次 *\$sth->err()* 调用，并对可能发生的错误进行处理。

因为 *prepare()*、*execute()* 和 *fetchall_arrayref()* 这 3 个方法经常一起使用，所以 DBI 还提供了一个功能相当于它们 3 个加在一起的 *\$dbh->selectall_arrayref(\$sql)* 方法。

16.1.5 字符串、BLOB、日期值、SET、ENUM 和 NULL

1. 修改数据库里的数据

为了对数据库里的数据进行修改，必须把有关的 SQL 命令作为字符串传递给 *do()* 方法。这个字符串的结构和格式必须符合 MySQL 的语法（参见第 21 章）。

在下面关于各种数据类型的讨论中，使用变量 \$data 来代表将被存入数据库的数据。也就是说，这个变量的内容 ('*data*' 或 "data") 将被放入 *INSERT* 或 *UPDATE* 命令，而这些 SQL 命令本身又临时存放在变量 \$sql 里。下面是一种最简单的情况：

```
$sql = "INSERT INTO tablename VALUES('$data1', '$data2', ...);
```

□ 日期/时间。为了让日期/时间值的格式符合 MySQL 的有关规定，必须使用相应的 Perl 函数或模块（比如 *gmtime()* 函数或 *TIME::Local* 模块）。

□ 时间戳。Perl 和 MySQL 的时间戳有着同样的含义，但使用的是不同的格式。Perl 时间戳 (*time()* 函数) 是一个 32 位整数，它的值是从 1970 年 1 月 1 日算起经过的秒数。MySQL 则要求时间戳的格式是 *yyyy-dd-mm hh:mm:ss*。

一般来说，时间戳的主要用途是给出最近一次修改时间。对于这种用法，只须把 *NULL* 值传递给 *INSERT* 或 *UPDATE* 命令即可，MySQL 会把正确的时间戳值自动存入数据库，如下所示：

```
$sql .= "NULL";
```

从另一个方面讲，如果真的想把一个 Perl 时间戳保存为一个 MySQL 时间戳，就应该在 *INSERT* 或 *UPDATE* 命令里用上 MySQL 函数 *FROM_UNIXTIME()*，如下所示：

```
$data = time(); // data contains the current time as a Unix timestamp
$sql .= "FROM_UNIXTIME(" . $data . ")";
```

□ 字符串和 BLOB。如果字符串或 BLOB 里有特殊字符，对它们进行转义往往是件很麻烦的事情。SQL 语言要求单引号、双引号、0 字节和反斜线字符的前面必须有一个反斜线字符作为前缀。

如果想让自己编写出来的 Perl 代码有良好的可移植性，就应该用`$dbh->quote()`方法对在代码里使用的字符串进行一下预处理。这个方法不仅会在字符串里的适当位置加上“\”或“\0”符号，还会把整个字符串用单引号括起来。比如说，`$dbh->quote("ab'c")`将返回`'ab\'c'`。此外，如果在 SQL 命令里使用了参数，`quote()`方法将自动作用于它们。

```
$sql .= $dbh->quote($data);
```

在拼装 SQL 命令的时候，Perl 语言中的`qq//`语法结构会非常有用。`qq//`可以把给定字符串变成一个标准化的 SQL 字符串，还会把字符串里的参数变量替换为它们的内容（并正确地去掉参数变量名两端的引号）。与使用字符串合并操作符“.”直接拼凑出一条（比如说）`$sql = "INSERT ..."`命令相比，使用`qq//`结构的好处是可以在`INSERT`命令直接使用单引号(') 和双引号(“) 字符而无须对之进行转义：

```
$data = $dbh->quote($data);
$sql = qq{INSERT INTO table (col1, col2, col3)
VALUES ($data, 'abc', PASSWORD("abc"))};
```

□ **NULL 值**。这里有一个好消息：`$dbh->quote()`方法还可以正确地完成对 Perl 空值`undef`的必要转换——具体到 MySQL 数据库系统，它将正确地返回 MySQL 空值`NULL`（注意，不带单引号）：

```
$sql .= $dbh->quote($data);
```

如果不想使用`quote()`方法，就应该像下面这样做：

```
$sql .= defined($data) ? "'$data'" : "NULL";
```

注意，在 SQL 命令字符串里，不要给`NULL`值加上引号，那样得到的将是一个字符串`NULL`。

2. 读取数据库里的数据

在下面的讨论中，将使用变量`$data` 来代表将被修改的数据库数据，即某个数据字段。比如，像下面这样来对变量`$data` 进行初始化：

```
$sth = $dbh->prepare("SELECT * FROM titles");
$sth->execute();
@row = $sth->fetchrow_array();
$data = $row[0];
```

□ **时间戳**。如果把 MySQL 时间戳存入`$data` 变量，它将是一个`2005-12-31 23:59:59` 格式的字符串。Perl 不会识别这种时间戳。如果想在 Perl 脚本里对 MySQL 时间戳进行处理，就必须在构造那条`SELECT`命令的时候用上 MySQL 函数`UNIX_TIMESTAMP()`，这个函数可以把一个格式如上所示的 MySQL 日期/时间字符串转换为一个 UNIX 时间戳值¹：

```
SELECT ... , UNIX_TIMESTAMP(a_timestamp) FROM ...
```

□ **日期**。如果把 MySQL`DATETIME` 数据列的值存入`$data` 变量，它将是一个`2005-12-31 23:59:59` 格式的字符串；如果是`DATE` 数据列的值，则不包括时间部分。如果想在 Perl 脚本里对 MySQL 日期进行处理，最简便实用的办法仍是用 MySQL 函数`UNIX_TIMESTAMP()`。可以用 MySQL 函数`DATE_FORMAT()`把日期排版为需要的任意格式。下面的命令会让变量`$data` 包含着一个

1. 原文这里出现严重错误。“20051231235959”不是“2005-12-31 23:59:59”的 UNIX 时间戳值！`UNIX_TIMESTAMP('2005-12-31 23:59:59')`的返回值不会这么大！而且，按照原文隐含的逻辑，“1970-01-01 00:00:00:00”将被转换为“19700101000000”，但这个时间的 UNIX 时间戳值是 0。——译者注

December 31 2005 格式的字符串：

```
SELECT ... , DATE_FORMAT(a_date, '%M %d %Y')
```

- **时间。**如果把 MySQL *TIME* 数据列的值存入 \$data 变量，它将是一个 23:59:59 格式的字符串。如果不从这个字符串里提取出小时、分钟和秒，最好用 MySQL 函数 *TIME_TO_SEC()* 把它转换为一个从 00:00:00:00 开始计数的秒数。当然，用其他的 MySQL 日期/时间函数来处理这个字符串也完全可以。警告：*UNIX_TIMESTAMP()* 函数不能用于 *TIME* 数据列。

提示 把用来处理和转换日期/时间的 MySQL 函数汇总在了本书的第 21 章。

- **NULL 值。**只要还没有对 \$data 变量进行过定义（赋值），它就包含着 Perl 空值 *undef*，它相当于 MySQL 空值 *NULL*。如果想知道从 MySQL 数据库读入 \$data 变量的值是不是 *NULL*，可以用 Perl 函数 *defined (\$data)* 加以测试。（不要使用把 \$data 变量与空字符串（""）或数值 0 进行比较的办法来测试它的内容是不是 *NULL*——虽然会先看到一条 Perl 警告消息，但这两种比较都将返回 *TRUE*，而脚本在看到那条警告消息的时候很可能已经按照错误的路线执行完了！）
- **字符串和 BLOB。**与 C 语言不同，用 Perl 处理二进制数据不存在任何问题，即便是内部包含着 0 字节的字符串也可以得到正确的处理。这意味着 \$data 变量的内容与提取自 MySQL 数据库的数据将完全一致。

作为一个原则，如果想把从 MySQL 数据表里读取出来的字符串输出到一份 HTML 文档里，就必须用 *escapeHTML()* 函数。

如果因为 Perl 对单个变量（或数组元素）在最大数据容量方面的限制、而在读取比较巨大的 BLOB 时发生错误，可以（在调用 *execute()* 方法之前）用 *\$dbh->['LongReadLen']=n* 来改变这一限制，这里的 *n* 是最大字节长度。

注意，*\$dbh->['LongReadLen']=0* 意味着诸如 BLOB 之类的超长 MySQL 字段将根本不会被读取。此时，\$data 的取值将是 *undef*，这种情况与 MySQL 字段的值就是空值 *NULL* 的情况无法区分。

3. 确定 *ENUM* 或 *SET* 集合里的元素

用 Perl 脚本处理 *ENUM* 和 *SET* 数据列不会遇到任何问题：这两种字段的值在 Perl 和 MySQL 之间传递时都是普通的字符串。这里需要注意的是：在 *ENUM* 集合里，用逗号隔开的各项数据之间不允许出现空格。

如果需要在一个 Perl 程序里把某个 *ENUM* 或 *SET* 数据列的值显示为字符串形式（比如说，把它们显示为一个 HTML 下拉选择框里的选项），就必须使用 SQL 命令 *DESCRIBE tablename columnname* 来查出这个数据列的定义，如下所示：

```
USE exceptions
DESCRIBE test_enum a_enum
Field  Type          Null  Key ...
a_enum enum('a','b','c','d','e') YES  ...
```

结果数据表里的 *Type* 列包含着需要的信息。下面这段代码将先对结果数据表中 *Type* 列里的字符串值 *enum('a','b','c')* 一步一步地进行拆分，然后再把它们一行一行地显示出来。（下面这段代码采用的算法需要假设 *SET* 或 *ENUM* 字符串本身不包含逗号。）

```

$sth = $dbh->prepare("DESCRIBE test_enum a_set");
$sth->execute();
$row = $sth->fetchrow_hashref();
$tmp = $row->{'Type'};           # enum(...) or set(...)
($tmp) = $tmp =~ m/^(.*)\)\)/;   # xyz('a','b','c') --> 'a','b','c'
$tmp =~ tr///d;                 # 'a','b','c' --> a,b,c
@enums = split(//, $tmp);        # @enums[0]=a, @enums[1]=b ...
foreach $enum (@enums) {         # output all values
    print "$enum\n";
}

```

16.1.6 DBD::mysql 模块特有的方法和属性

DBI 接口只是各种数据库 API 接口的最大交集, *DBD::mysql* 模块里的大多数函数却是 MySQL 专用的。使用 *DBD::mysql* 模块提供的 MySQL 专用函数编写出来的 Perl 代码不太容易移植到另一种数据库系统上去, 但这样的代码通常有着更高的执行效率。

本节将 *DBD::mysql* 模块里最重要的 MySQL 专用函数和属性。第 23 章收录了对 *DBD::mysql* 模块的完整指南。

1. 用 *rows()* 方法来确定 *SELECT* 查询结果中的记录总数

前面介绍过的 *rows()* 方法其实并不是一个 MySQL 独有的函数, 但这个函数只有在与 *DBD::mysql* 模块一起使用时才会返回查询结果中的记录总数——当然, 前提是 *execute()* 方法成功返回。不过, *rows()* 方法只在 *DBD::mysql* 驱动模块没有使用 *mysql_use_result* 属性去执行 SQL 查询命令时才能返回正确的结果。

2. 确定 *AUTO_INCREMENT* 值

在执行完 *INSERT* 命令后, 经常需要确定新插入的那条记录的 *AUTO_INCREMENT* 值。这个任务可以用 *DBD::mysql* 模块为 *\$dbh* 句柄提供的 *mysql_insertid* 属性来完成:

```
$id = $dbh->{ 'mysql_insertid' };
```

3. 确定关于数据列的更多信息

DBI 提供了一系列 *\$sth* 属性 (如 *\$sth->{NAME}'*) 供程序员用来确定关于 *SELECT* 查询结果里各数据列的信息。在此基础上, *DBD::mysql* 模块又提供了一些很有用的属性, 比如说, '*MYSQL_IS_BLOB*' 属性可以用来判断某个结果数据列是否包含着 BLOB 数据、'*MYSQL_TYPE_NAME*' 属性可以用来确定给定数据列的数据类型的名字、等等。

4. *mysql_store_result* 模式与 *mysql_use_result* 模式

在默认的情况下, 当用 *prepare()* 和 *execute()* 方法来执行一条 *SELECT* 命令的时候, *DBD::mysql* 模块将会调用 C 语言函数 *mysql_store_result()* 来传输本次查询结果, 这意味着所有的结果记录将一次性地被发送到客户端并将驻留在内存里直到 *\$sth->finish()* 方法被调用为止。

如果在调用 *prepare()* 方法后、调用 *execute()* 方法前把 *mysql_use_result* 属性设置为 1, *DBD::mysql* 模块将会调用 C 语言函数 *mysql_use_result()* 来传输结果记录。这意味着结果记录只在脚本需要它们的时候才会从 MySQL 服务器被传递到客户端。这么做的好处是: 如果需要处理的数据量比较大, 数据在客户端的内存占用量会大为降低。(不过, 如果在取回查询结果的时候使用的是 *fetchall_arrayref()* 方法, 就享受不到这个好处了。)

```

$sth = $dbh->prepare("SELECT * FROM table");
$sth->{ 'mysql_use_result' }=1;
$sth->execute();

```

mysql_store_result 模式与 *mysql_use_result* 模式之间的区别如表 16-1 所示。

表 16-1 *mysql_store_result* 模式与 *mysql_use_result* 模式之间的区别

	<i>mysql_store_result</i> 模式（默认设置）	<i>mysql_use_result</i> 模式
<code>\$sth->rows()</code>	这个函数将返回结果记录的总数	这个函数将返回 0 或此前已经传输来的结果记录个数
锁定	<i>READ-LOCK</i> 时间最短	如果使用了锁定，数据表将被锁定到最后一条结果记录被读走为止
客户端内存占用情况	所有的结果记录都存放在客户端	客户端每次只存放一条结果记录
速度	读出第一条结果记录之前的等待时间会比较长（因为此前需要把所有的结果记录都传输到客户端），以后就快多了	读出第一条结果记录之前的等待时间非常短，但后续结果记录的读取速度要比 <i>mysql_store_result</i> 模式慢不少

16.1.7 Unicode

如果打算用一个 Perl 脚本来处理 Unicode 数据，就必须注意以下几件事情。

- 首先，必须用以下命令让 MySQL 服务器知道本次通信将使用 UTF-8 字符集进行：

```
$dbh->do("SET NAMES 'utf8'");
```

除了 '*utf-8*'，还可以在这条命令里给出 MySQL 服务器所能支持的其他字符集的名字，比如 '*latin*' 等。MySQL 服务器上的默认字符集是由它的具体配置情况决定的。因此，千万不要想当然地认为每一个 MySQL 服务器都会有需要的字符集，即便是 *Latin1* 字符集也不一定能在所有的 MySQL 服务器上使用。

注意，*SET NAMES* 命令只对 MySQL 服务器与 Perl 脚本之间的通信有影响，它对 MySQL 数据表里实际存储的数据所使用的字符集没有任何影响（MySQL 数据表里的字符集设置是在 *CREATE TABLE* 命令创建它们的时候确定的）。收到 *SET NAMES* 命令之后，MySQL 将自动地把所有的字符串从有关数据表的字符集转换为新设定的字符集。这里要特别提醒大家注意这样一个问题：如果字符串里有新字符集无法表示的字符，那些字符将被替换为一个问号；这意味着比较特殊的字符容易导致信息缺失和不完整。

- 完成以上设置之后，来自 MySQL 服务器的字符串数据就都是 UTF-8 编码的了。为了让 Perl 对这些字符串做出正确的解释，还需要把这件事告知 Perl，这需要来自 *Encode* 模块的 *decode("utf8", ...)* 方法来完成。

```
use Encode; # makes Encode and Decode functions available
...
$utf8data = decode("utf8", $data_from_mysql);
```

- 如果 Perl 脚本会产生输出，还需要再告诉 Perl 必须对输出结果进行 UTF-8 编码：

```
binmode(STDOUT, ":utf8"); # set output to be in UTF8
```

- 如果 Perl 脚本自身包含着 UTF-8 字符串，必须在脚本的开头写明 *use utf8*。

```
use utf8;
$utf8data = "äöüß";
```

- 如果 Perl 脚本是一个 CGI 脚本，还必须在 HTTP 首部里设定对输出结果进行 UTF-8 编码：

```
print header(-type => "text/html", -charset => "utf-8");
```

根据脚本的具体运行环境，上面说的这些操作不一定都能完成。在 UNIX/Linux 环境下，Perl 解释器

往往可以根据环境变量`$LANG`识别出自己正运行于一个UTF-8系统并沿用这一默认设置，从而正确地对Unicode做出处理。但根据笔者个人的经验，事实并非总是如此。

提示 在本章稍后的内容里给出了两个可以处理Unicode的CGI Perl脚本，它们体现了几种Unicode编程技巧。这里没有足够的篇幅来进一步讨论Perl和Unicode的问题，对此感兴趣的读者可以找一本关于Perl的最新教科书或者到下面两个网站上去看看，那里有许多关于Perl和Unicode（还有MySQL）的信息：

<http://www.perldoc.com/perl5.8.0/pod/perluniintro.html>
<http://perlwelt.horus.at/Beispiele/Magic/PerlUnicodeMysql/>
<http://lists.mysql.com/perl/3312>

16.1.8 事务

在默认的情况下，即便是在支持事务的数据表（MySQL目前只有InnoDB数据表支持事务）里，数据库命令也是逐条执行的。为了把多条命令当做一个事务来执行，必须先用`$dbh->{'AutoCommit'} = 0`指令暂时关闭MySQL的事务自动提交（autocommit）模式。如果在与数据库建立连接时没有使用`'RaiseError'=>1`属性（见下面的讨论），还应该在发出这条关闭自动提交模式的指令后去检查一下它是否真的关闭了。

在关闭自动提交模式后，所有的SQL命令将构成一个事务，直到使用`$dbh->commit()`方法加以确认或者是使用`$dbh->commit()`方法加以撤销为止——此时，下一个事务将自动开始。下面是上述过程的Perl代码：

```
$dbh->{'AutoCommit'} = 0;      # turn off autocommit mode
if($dbh->{'AutoCommit'}) {    # test whether it worked
    print "transaction error\n";
    exit();
}

$dbh->do("SQL-Kommando");    # all SQL commands form a transaction
$dbh->do("SQL-Kommando");
...
$dbh->commit();              # confirm transaction
$dbh->rollback();            # or abort transaction
```

16.1.9 出错处理

Perl脚本有好几种出错处理方式。最简便易行的办法（尤其是对程序开发阶段而言）是在与数据库建立连接时使用`'RaiseError'=>1`属性，这么做的效果是每发生一个错误都可以看到一条相应的出错消息，而且Perl脚本也会立刻停止执行。（这一属性不仅在与数据库建立连接时有这种效果，在其他的DBI方法执行出错时也将发挥作用。）

在大多数场合，出错处理做到这一步已经足够了。可如果Perl脚本是用来生成动态Web网页的，那么脚本一出错就退出执行（并让用户看到一条摸不着头脑的出错消息）显然不是一种对用户友好的态度。

如果想建立一种更精细的出错处理机制，就应该关闭DBI的自动化错误响应功能并在调用`connect()`方法时给出`'PrintError'=>0`属性。从现在起，必须时刻注意检查`err()`和`errstr()`这两个DBI方法的返回值。如果没有发生错误，`err()`方法将返回0；如果发生错误，它将返回出错代码，此时`errstr()`方法将返回一条关于最近一次错误的出错消息。

注解 *DBI*方法在发生错误时将统一返回*undef*, 但有那么几个*DBI*方法在正常执行时才会返回*undef* (比如在数据字段包含NULL值的时候)。因此, 只根据各个方法的返回值去判断它们在执行时是否发生错误往往不足以得出正确的结论。

*err()*和*errstr()*方法还可以与*DBI*句柄\$*dbh*和\$*ssth*搭配使用 (此时它们将返回与给定连接或给定SQL命令有关的出错信息)。注意, 如果想检查与数据库的连接是否建立成功, 就应该检查*DBI->err()*或*DBI->errstr()*的返回值 (这是因为如果与数据库的连接没有建立成功的话, \$*dbh*将不可用。)

1. 出错处理示例

下面这个例子给出了Perl *DBI*代码中应该进行出错处理的地方。在这个例子里, 脚本会在发生任何一种错误后退出执行, 读者在自己的脚本里可以根据具体情况执行其他的指令。这个示例的演示重点是*err()*和*errstr()*方法应该在什么时候分别与*DBI*、\$*dbh*和\$*ssth*搭配使用。

这里还有一个仅适用于MySQL的细节需要提醒大家注意: 对于MySQL数据库上的SELECT命令, 出错处理代码只能放在*execute()*方法调用语句之后。如果是其他的数据库系统, 把出错处理代码放在*prepare()*方法调用之后就能捕获到错误。

```
$datasource = "DBI:mysql:database=exceptions;host=localhost";
$user = "root";
$passw = "xxx";
$dbh = DBI->connect($datasource, $user, $passw,
    {'PrintError' => 0});
if(DBI->err()) {
    print "connection error: " . DBI->errstr() . "\n";
    exit();
}
$dbh->do("INSERT INTO testall (a_float) VALUES (10.0)");
if($dbh->err()) {
    print "error wth INSERT command: " . $dbh->errstr() . "\n";
    exit();
}
$sth = $dbh->prepare("SELECT * FROM test_blob");
$sth->execute();
if($sth->err()) {
    print "error in SELECT execute: " . $sth->errstr() . "\n";
    exit();
}
while(my $hashref = $sth->fetchrow_hashref()) {
    if($sth->err()) {
        print "error in SELECT fetch: " . $sth->errstr() . "\n";
        exit();
    }
    print length($hashref->{'a_blob'}) . "\n";
}
$sth->finish();
$dbh->disconnect();
```

2. *DBI*出错日志 (*trace()*方法)

如果怀疑引起错误的不是代码, 而是MySQL服务器、*DBI*模块、*DBD::mysql*模块或其他方面, 把*DBI*出错日志消息显示出来或写入一个文件可以帮助追踪和调试有关的错误。这个功能需要使用*trace()*方法来明确地启用。类似于*err()*和*errstr()*方法的情况, *trace()*方法也可以分别与*DBI*、\$*dbh*和\$*ssth*搭配使用。

调用*trace()*方法需要给出一个日志级别数字 (0=不记录出错日志; 9=最详细的出错日志) 和一个可选的文件名。如果没有给出文件名, 出错日志数据将被发送到*STDERR*设备 (如果是Windows环境, 则发送到标准输出设备*STDOUT*去):

```
DBI->trace(2); # activate logging globally for the DBI module
$sth->trace(3, 'c:/dbi-trace.txt'); # logging for $sth methods only
```

16.2 示例：删除无效的数据记录 (*mylibrary*)

在经过了几个月的试用之后，*mylibrary* 数据库里肯定会积累出各种各样无效记录。这种事几乎每一位数据库开发人员都遇到过，但令人遗憾的是他们当中有不少人在测试工作结束后忘记了应该把测试数据库好好清理一下。

回到 *mylibrary* 数据库这个例子上来，*delete-invalid-entries.pl* 脚本可以用来完成许多清理工作，其中包括：

- 删除那些名字以 *test* 开头的图书、作者和出版公司。（为避免违反外键约束条件，在删除这样的图书和作者之前，必须先去删除 *rel_title_author* 数据表里的相关记录，然后才能去删除 *titles* 和 *authors* 数据表里的有关记录。）
- 按照同样的顺序，无效的图书、作者和出版公司记录也将被删除。这里所说的“无效记录”指的是那些不能构成关于一本图书完整信息的记录，例如没有图书的作者、没有图书的出版公司或没有作者的图书等。

当然，这个脚本的清理范围还可以再扩大一些，比如删除无效的图书门类和/或图书语言记录等。不过，这里没有必要让它去检查 *mylibrary* 数据库里各数据表之间的关系是否无效——外键约束条件可以保证不会发生那样的情况；比如说，不可能出现 *titles* 数据表引用的出版公司在 *publishers* 数据表里不存在的问题。

程序代码

在这个脚本里，负责与 *mylibrary* 数据库建立连接的代码部分没有什么特别。因为在调用 *connect()* 方法的时候使用了 *{'RaiseError'=>1}* 属性，所以如果在建立连接时（以及以后）发生错误，这个脚本将退出执行并返回一条出错消息：

```
#!/usr/bin/perl -w
# delete-invalid-entries.pl
use strict;
use DBI;
# declare variables
my($datasource, $user, $passw, $dbh, $sth, $n, $row);
# create connection to database
$datasource = "DBI:mysql:database=mylibrary;host=localhost;";
$user = "root";
$passw = "xxx";
$dbh = DBI->connect($datasource, $user, $passw,
    {'RaiseError' => 1});
```

接下来的 *SELECT* 命令负责搜索那些书名或副标题以 *test* 开头的图书。在随后的循环里，搜索出来的图书将先从 *rel_title_author* 数据表、再从 *titles* 数据表里被删除。

```
# delete test titles
$sth = $dbh->prepare(
    "SELECT DISTINCT title, titles.titleID ".
    "FROM titles, rel_title_author ".
    "WHERE titles.titleID = rel_title_author.titleID ".
    "AND (title LIKE 'test%' OR subtitle LIKE 'test%')");
$sth->execute();
while($row = $sth->fetchrow_hashref()) {
    print "Delete title: $row->{'title'}\n";
    $dbh->do("DELETE FROM rel_title_author ".
        "WHERE titleID=$row->{'titleID'}");
    $dbh->do("DELETE FROM titles ".
        "WHERE titleID=$row->{'titleID'}");
}
$sth->finish();
```

```
# delete test authors
# code similar to the above ...
# delete test publishers
# code similar to the above...
```

接下来的 *DELETE* 命令用上了从 MySQL 4.1 版本才开始引入的子查询功能。它将把 *titleID* 没有在 *rel_title_author* 数据表里出现过的所有图书删除干净。无效的作者记录也是用类似办法清理的。

```
# delete orphaned titles
$n = $dbh->do(
    "DELETE FROM titles WHERE titleID NOT IN ".
    " (SELECT titleID FROM rel_title_author)");
if($n>0) {
    print "Deleted $n orphaned titles\n";
}
# delete orphaned authors
# code similar to the above ...
```

在删除无效的出版公司记录时，这个脚本用了一个子查询来排除那些 *publID* 字段包含 *NULL* 值的记录。这一是因为 *publID* 字段允许包含 *NULL* 值，二是因为 *NULL* 值也不适合用在 *NOT IN ...* 比较操作里。

```
# delete orphaned publishers
$n = $dbh->do(
    "DELETE FROM publishers ".
    "WHERE publID NOT IN ".
    " (SELECT DISTINCT publID FROM titles ".
    " WHERE NOT publID IS NULL)");
if($n>0) {
    print "Deleted $n orphaned publishers\n";
}
# end of program
$dbh->disconnect();
```

16.3 CGI 示例：图书管理 (*mylibrary*)

Perl CGI 脚本里的数据库访问操作与普通的 Perl 脚本程序没有什么两样，只是前者出于信息安全方面的考虑往往以一个独立程序的方式运行（而不是被其他程序调用）。这里必须注意的要点是：在输出来自数据库的数据时，字符串里的特殊字符必须用 *escapeHTML()* 函数进行编码，以符合 HTML 的有关语法规定。

本节将介绍两个为 *mylibrary* 数据库编写的 Perl CGI 小程序：

- *mylibrary-find.pl* 脚本，负责实现对现有图书的检索功能。
- *mylibrary-simpleinput.pl* 脚本，负责实现新图书的录入功能。

16.3.1 图书检索 (*mylibrary-find.pl* 脚本)

mylibrary-find.pl 脚本可以从 *mylibrary* 数据库里把满足搜索条件的图书检索出来。把图书名称的前几个字符输入一个图书检索表单，再单击 **OK** 按钮就可以把查找到的图书以及它们的作者、出版公司和出版年月按字母表顺序全部开列出来（如图 16-1 所示）。

1. 程序结构

mylibrary-find.pl 脚本将完成显示一个简单的图书检索表单、对表单数据进行处理和显示检索结果等三项任务。这里使用了 CGI 模块的 *param()* 方法来分析 HTML 表单变量 *formSearch*。

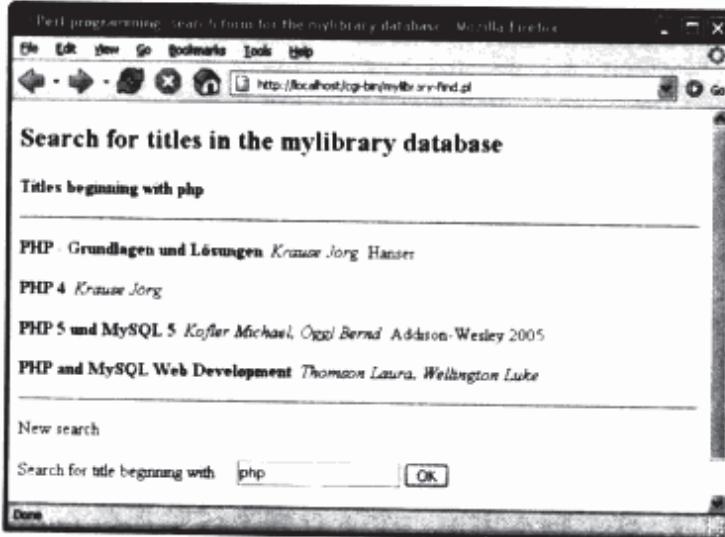


图 16-1 对图书进行检索

2. 程序代码

这个脚本的开头部分是一些用来声明各有关模块和变量的代码。同时使用 *strict* 模块和 *my* 指令有助于发现变量名里的打字错误；*DBI* 模块负责访问数据库；*CGI* 模块负责输出 HTML 结构；*CGI::Carp* 模块负责把出错消息显示在 HTML 结果文档里，这特别有助于追查各种错误的根源。

```
#!/usr/bin/perl -w
# mylibrary-find.pl
use strict;
use DBI;
use CGI qw(:standard);
use CGI::Carp qw(fatalsToBrowser);
# declaration of variables
my($datasource, $user, $passw, $dbh, $search, $sql, $sth, $result,
   $rows, $i, $row);
```

与本章前面给出的示例相比，这个例子里负责连接数据库的代码没有用到什么新东西。如果在试图与数据库建立连接时发生错误，这个脚本将退出执行并在其 HTML 结果文档里显示一条相应的出错消息。如果连接成功，这个脚本将调用几个来自 *CGI* 模块的方法去打开它的 HTML 结果文档：

```
# create connection to database
$datasource = "DBI:mysql:database=mylibrary;host=localhost;";
$user = "root";
$passw = "xxx";
$dbh = DBI->connect($datasource, $user, $passw,
    {'PrintError' => 0});
# display error message and end script if necessary
if(DBI->err()) {
    print header(),
        start_html("Sorry, no database connection"),
        p("Sorry, no database connection"), end_html();
    exit();
}
# inform MySQL that communication will be in the Latin-1
# character set
$dbh->do("SET NAMES 'latin1'");
# introduce HTML document
print header(-type => "text/html", -charset => "latin-1"),
    start_html("Perl programming, search form for the ",
               "mylibrary database"), "\n",
    h2("Search for titles in the mylibrary database"), "\n";
```

3. 处理表单数据，显示检索结果

如果有表单数据返回，这个脚本将先调用 `param()` 方法提取出 HTML 表单变量 `formSearch`，然后去掉这个字符串里的特殊字符 “_” 和 “%”。接下来，如果 `$search` 变量不为空，利用 `$sql` 变量构造出来的那一长串 `SELECT` 命令就会被执行，它的查询结果将由 `fetchall_arrayref()` 方法传输到一个二维数组里。

这里使用的 `SELECT` 查询命令需要一些解释：`GROUP_CONCAT()` 函数和 `GROUP BY title.titleID` 子句将联手为每一本被查找出来的图书创建一个由它的全体作者的姓名构成的排序字符串；`LEFT JOIN` 负责关联 `titles` 和 `publishers` 数据表，其效果是没有出版公司的图书也会被查找出。

```
# process form data
$search = param('formSearch');
# delete characters _ and %
$search =~ tr/%/_//d;
if($search) {
    print p(), b("Titles beginning with ", encode_entitiesescapeHTML($search));
    # Titelsuche
    $sql = "SELECT titles.titleID, title, year, publName, ".
        " GROUP_CONCAT(authname ORDER BY authname SEPARATOR ', ') ".
        " AS authors ".
        "FROM titles, authors, rel_title_author ".
        " LEFT JOIN publishers ON titles.publID = publishers.publID ".
        " WHERE titles.titleID = rel_title_author.titleID ".
        " AND authors.authID = rel_title_author.authID ".
        " AND title LIKE '$search%' ".
        " GROUP BY titles.titleID ".
        " ORDER BY title ".
        " LIMIT 100";
    $sth = $dbh->prepare($sql);
    $sth->execute();
    $result = $sth->fetchall_arrayref({});
    $sth->finish();
```

接下来是对二维数组进行处理。首先检查本次检索是否真的找到了一些图书，接下来用一个循环来输出各本图书的信息，其中可能包含的 HTML 特殊字符都用 `escapeHTML()` 函数转换成了正确的 HTML 编码。

```
# were titles found?
$rows = @{$result};
if($rows==0) {
    print p(), "Sorry, no titles found.";
}
# display titles
else {
    # loop over all records
    for($i=0; $i<$rows; $i++) {
        $row = $result->[$i];
        print p(),
            b(escapeHTML($row->{'title'})), ": ",
            i(escapeHTML($row->{'authors'})), ".",
            escapeHTML($row->{'publName'}), " ",
            $row->{'year'}, ".";
    }
    print p(), hr(), p(), "New search:", p();
}
```

4. 显示HTML表单

以下代码负责显示用来输入图书检索信息的 HTML 表单和关闭 HTML 文档：

```

# display the form
print start_form(),
p(), "Search for title beginning with ... ",
textfield({-name => 'formSearch', -size => 20,
           -maxlength => 20}), " ",
submit({-name => 'formSubmit', -value => 'OK'}),
end_form();
print end_html();
# program ends
$dbh->disconnect();

```

5. 值得改进的地方

这里给出的示例代码只能对图书名称的前几个字符进行搜索。要是它还能以全文检索的方式对作者、图书门类等进行搜索当然会更理想。(在本书第 15 章里有一个从 *mylibrary* 数据库检索图书的改进示例，那个示例使用的编程语言是 PHP。那个程序不仅具备分页显示查询结果的能力，还会显示一些交叉链接供用户做进一步搜索，比如把某位特定作者写的书全部查找出来等。)

16.3.2 新图书的简单输入 (*mylibrary-simpleinput.pl* 脚本)

本节将介绍一个简单的图书信息输入表单。这个表单有两个文本框，一个用来输入新图书的书名，另一个用来输入新图书的作者姓名。在表单数据返回后，这个示例脚本将首先检查在新图书的作者当中有没有已经被收录在了数据库里的，然后用 *INSERT* 命令把所有此前尚未收录过的作者姓名添加到数据库、把新图书存入数据库、最后对 *rel_title_author* 数据表里的作者和图书关系进行刷新。

1. 程序结构

mylibrary-simpleinput.pl 脚本将显示一个简单的表单，并对用户通过单击这个表单里的 OK 按钮而提交来的表单数据进行处理。这里使用了 *CGI* 模块的 *param()* 方法来分析两个 HTML 表单变量：一个是负责传递新图书名字的 *formTitle*，另一个是负责传递新图书作者名单的 *formAuthors*。

2. 程序代码

与 16.3.1 节中的 *mylibrary-find.pl* 脚本一样，这个脚本的开头部分也是一些用来声明各有关模块和变量的代码：

```

#!/usr/bin/perl -w
# mylibrary-simpleinput.pl
use strict;
use DBI;
use CGI qw(:standard);
use CGI::Carp qw(fatalToBrowser);
# declaration of variables
my($datasource, $user, $passw, $dbh, @row,
   $formTitle, $formAuthors, $titleID, $authID, $author);

```

接下来的代码也类似于 *mylibrary-find.pl* 脚本里的同用途代码，就不在这里重复解释了：

```

# create connection to database
... as with mylibrary-find.pl
# if an error occurs, display error message, end script
... as with mylibrary-find.pl
# introduce HTML document
... as with mylibrary-find.pl

```

3. 把新图书的书名及作者存入数据库

如果有表单数据返回（用 *if(param())* 来判断），这个脚本将先提取出 HTML 表单变量 *formTitle* 和 *formAuthors*，然后检查它们是否有内容（即判断用户有没有输入一个书名和一些作者姓名）。如果它们没有内容，则显示一条出错消息并把图书信息输入表单重新显示给用户：

```
# evaluate form data
if(param()) {
    $formTitle = param('formTitle');
    $formAuthors = param('formAuthors');
    # were both title and authors given?
    if($formTitle eq "" || $formAuthors eq "") {
        print p(), b("Please specify title and at least one author!");
    }
}
```

为了把新图书存入 *titles* 数据表，这里采用的办法是把一条带有一个参数的 *INSERT* 命令传递给 *do()* 方法去执行，参数值由 *\$formTitle* 变量负责传递。这么做的效果是：在实际传递到 MySQL 服务器的 *INSERT* 命令字符串里，*\$formTitle* 变量将被自动替换为它的内容（也就是新图书的名字）并加上单引号，书名里的特殊字符也将得到正确的处理：

```
# form data are correct; store
else {
    # store title
    $dbh->do("INSERT INTO titles (title) VALUES (?)",
              undef, ($formTitle));
    $titleID = $dbh->{mysql_insertid};
```

作者名单是在一个 *foreach* 循环里处理的。在这个循环里，首先检查每位作者是不是已经存在于 *authors* 数据表里。如果是，则从 *authors* 数据表里读出这位现有作者的 *authID* 编号值；如果不是，则存入这位新作者并利用 *mysql_insertid* 属性确定新作者的 *authID* 编号值。最后，把新书的 *titleID* 编号值和各位新作者的 *authID* 编号值的组合正确地存入 *rel_title_author* 数据表：

```
# store authors
foreach $author (split(/;/, $formAuthors)) {
    # does the author already exist?
    @row = $dbh->selectrow_array("SELECT authID FROM authors " .
                                  "WHERE authName = " .
                                  "$dbh->quote($author)");
    # yes: determine existing authID
    if(@row) {
        $authID = $row[0];
    }
    # no: store new author, determine new authID
    else {
        $dbh->do("INSERT INTO authors (authName) VALUES (?)",
                  undef, ($author));
        $authID = $dbh->{mysql_insertid};
    }
    # store entry in in rel_title_author table
    $dbh->do("INSERT INTO rel_title_author (titleID, authID) " .
              "VALUES ($titleID, $authID)");
}
```

数据存储操作全部顺利完成之后，还要在 HTML 文档里显示一条简短的提示信息，告诉用户新书已被成功地存入了数据库。接下来，清除 HTML 表单变量，以便用户可以在表单里继续输入下一本新书的信息：

```
# feedback
print p(), "Your last input has been saved.";
print br(), "You may now continue with the next title.";
# delete form variables (for the next input)
param(-name=>'formTitle', -value=> '');
param(-name=>'formAuthors', -value=> '');
}
```

4. 显示HTML表单

以下代码负责显示用来输入图书信息的 HTML 表单和关闭 HTML 文档：

```

print start_form(),
p(), "Title:",
br(), textfield({-name => 'formTitle', -size => 60,
                 -maxlength => 80}),
p(), "Authors:",
br(), textfield({-name => 'formAuthors', -size => 60,
                 -maxlength => 100}),
br(), "(Last name first! If you want to specify more ",
      "than one author, use ; to separate them!)",
p(), submit({-name => 'formSubmit', -value => 'OK'}),
end_form();
print end_html();
# program end
$dbh->disconnect();

```

5. 值得改进的地方

`mylibrary-simpleinput.pl` 脚本不能用来输入新图书的副标题、出版公司、图书门类等信息，没有对来自 HTML 表单的用户输入数据做任何合法性检查，也不能用来修改数据库里的现有图书记录。总之，这个脚本有很多值得改进的地方！（在第 15 章给出了一个比较令人满意的图书信息输入表单。那个示例使用的编程语言是 PHP，但其中包含的许多技巧都很容易移植到 Perl 脚本里来。）

16.4 CGI Unicode 示例

这一节里的 `mylibrary-find-utf8.pl` 和 `mylibrary-simpleinput-utf8.pl` 脚本是在前两节给出的两个示例脚本的基础上增加了 Unicode 处理功能而分别得到的变体。这两组变体脚本都有着基本相同的代码，所以在这一节里的讨论将集中在它们之间的区别上，新增加或需要修改的代码将以黑体字突出显示。

注解 某些早期的 Web 浏览器可能无法处理 Unicode HTML 文档，这尤其体现在对表单数据进行处理的时候。本节的两个示例脚本在两种比较新的 Web 浏览器（一种是 Windows XP SP2 环境下的 Internet Explorer 6.0，另一种是 Windows 和 Linux 环境下的 Firefox 1.0）上都成功地通过了测试，Unicode 与早期 Web 浏览器无法兼容的问题应该不会再困扰人们了。

16.4.1 图书检索 (`mylibrary-find-utf8.pl` 脚本)

在这个脚本的开头部分出现了两条新的令：`use Encode qw(decode)` 指令使用户可以在脚本里使用 `Encode::decode()` 函数；`binmode()` 函数告诉 Perl 必须用 Unicode 字符集 (UTF-8) 对输出结果进行编码。

SQL 命令 `SET NAMES 'utf8'` 负责通知 MySQL 服务器本次通信将使用 Unicode 字符集进行。接下来，我们把选项 `-charset=>"utf-8"` 和 `-encoding=>'utf-8'` 传递给了 `header()` 和 `start_html()` 函数，这是让 Web 浏览器知道由这个脚本生成的 HTML 文档使用了 UTF-8 编码所必须的步骤。注意，这里不需要使用 `use utf8` 指令，因为这个脚本自身没有包含任何 Unicode 字符。

```

#!/usr/bin/perl -w
# mylibrary-find-utf8.pl
...
use Encode qw(decode);
binmode(STDOUT, ":utf8");
# definition of variables as in mylibrary-find.pl ...
# connection to the database as in mylibrary-find.pl ...
# inform MySQL that communication
# is in the Latin-1 character set
$dbh->do("SET NAMES 'utf8'");
# introduce HTML document

```

```

print header(-type => "text/html", -charset => "utf-8"),
    start_html(-encoding => 'utf-8',
        "Perl programming, search form for the mylibrary database"), "\n",
    h2("Search for titles in the mylibrary database"), "\n";

```

接下来，必须把从 HTML 表单里传递到这个脚本的搜索字符串从 UTF-8 格式转换为 Perl 内部使用的字符集格式。(如果省略了这个步骤，Perl 会认为来自 HTML 表单的输入数据是一个 Latin-1 字符串。)

图书检索部分的代码与前两个小节里的示例没有任何差异。只是在输出检索结果的时候，必须用 *decode()* 函数告诉 Perl 它正在使用 UTF-8 字符串。

```

$search = decode('utf8', param('formSearch'));
# title search as in mylibrary-find.pl ...
# output results
...
print p(),
    b(escapeHTML( decode("utf8", $row->{'title'})    )), ": ",
    i(escapeHTML( decode("utf8", $authors)           )), ". ";
    escapeHTML( decode("utf8", $row->{'publName'})) , " ",
    $row->{'year'}, "\n";

```

作为 Perl (以及其他脚本语言解释器) 的一项自动功能，在显示搜索结果的同时，Perl 还会把相应的 HTML 搜索表单以及这个表单里的原始输入一起输出到 HTML 结果页面里；这也正是人们在使用 Yahoo 等搜索引擎时在看到搜索结果的同时还能看到自己刚才输入的搜索条件的原因。可是，Perl 本身是无法知道搜索表单里的原始输入 (具体到这个例子，就是表单变量 *formSearch* 的内容) 使用的是什么字符集 (具体到这个例子，就是 UTF-8 字符集)，所以还需要把经过 UTF-8 解码而得到的 *\$search* 变量值重新赋值给 HTML 表单变量 *formSearch*；如果不这样做，万一 Perl 使用的默认字符集不是 UTF-8 的话，它就可能会把所有的非 ASCII 字符弄得一团糟。

HTML 表单选项 *accept-charset="utf-8"* 的作用是让 Web 浏览器知道它应该为这个脚本返回 UTF-8 编码的输入。(不过，在笔者进行的测试中，Web 浏览器在没有这个选项的情况下也工作得很好。)

```

# display search form
param('formSearch', $search);
print start_form('-accept-charset' => 'utf-8'),
    ... and so on as in mylibrary-find.pl

```

16.4.2 新图书的输入 (*mylibrary-simpleinput-utf8.pl* 脚本)

把 16.4.1 节里的 Latin-1 变体修改为 *mylibrary-simpleinput-utf8.pl* 脚本的情况与在 *mylibrary-find-utf8.pl* 脚本上发生的事情差不多。这里没有使用 *binmode(STDOUT, ":utf8")* 函数，因为这个脚本只需要对输入进行处理，不涉及 Unicode 输出的问题。

```

#!/usr/bin/perl -w
# mylibrary-simpleinput-utf8.pl
...
use Encode qw(decode);
# definition of variables as in mylibrary-simpleinput.pl ...
# connection to database as in mylibrary- simpleinput.pl ...
# inform MySQL that communication is
# to be in the Latin-1 character set
$dbh->do("SET NAMES 'utf8'");
# HTML/HTTP header as in mylibrary-find.pl ...
# form data are in the UTF-8 character set
if(param()) {
    $formTitle  = decode('utf8', param('formTitle'));
    $formAuthors = decode('utf8', param('formAuthors'));
    ...
    # and so on, as in mylibrary-find.pl ...
}

```

第 17 章

Java(JDBC 和 Connector/J)

本章将讨论用 Java 程序设计语言开发 MySQL 应用程序的问题。将使用 JDBC 和 Connector/J 模块作为开发 MySQL 应用程序的 API 接口。

与本章内容有关的测试全部是在 Windows XP (配备 Sun JDK 1.5.0) 和 SUSE Linux 9.2 (配备 Sun JDK 1.4.2, 软件包的名字是 `java-1_4_2-sun-devel`) 环境下进行的。在这两种测试环境里, 在连接 MySQL 服务器时分别使用了 Connector/J 3.1.7 和 3.2.0 alpha 版本。

17.1 基础知识

17.1.1 Java 的安装

要想编写 Java 程序, 必须先安装 Java Software Development Kit (Java SDK, 或 JDK; Java 软件开发工具)。目前应用最广的 JDK 来自 Sun 公司 (该公司也是 Java 的诞生地); 但包括 IBM 在内的其他一些软件开发公司也推出了它们自己的 Java 实现。本章内容将主要围绕 Sun 公司的 Java 版本展开。

JDK 已被收录在了许多种 Linux 发行版本里, 用它们自带的软件包管理器可以迅速完成 Java 的安装。如果使用的 Linux 发行版本没有提供 Java, 或者使用的是 Windows 系统, 可以从 java.sun.com/downloads 网站免费下载 JDK (大约 40MB)。JDK 的官方正式名称叫做 Java 2 Platform Standard Edition *n* SDK (简称 J2SE *n* SDK), 其中的 *n* 是版本号。

把 JDK 安装好以后, 还需要把 Java 安装目录下的 bin 子目录添加到系统环境变量 *PATH* 里去。如果使用的是 Windows 9x/Me 系统, 需要到 autoexec.bat 文件里去修改这个变量; 如果使用的是 Windows NT / 2000 / XP 系统, 可以通过菜单命令 Control Panel | System | Advanced | Environment Variables (环境变量) 打开的对话框 (如图 17-1 所示) 去修改这个变量。注意, Windows 系统的 *PATH* 变量里的路径必须用分号 (;) 隔开。下面是某个 Windows 系统上的 *PATH* 变量的内容:

```
C:\> PATH
PATH=C:\Programs\Perl\bin\;C:\WINDOWS\system32;C:\WINDOWS;
C:\WINDOWS\System32\Wbem;C:\programs\mysql\mysql server 5.0\bin;
C:\Programs\Java\jdk1.5.0_01\bin
```

在 SUSE 9.2 系统上, Java 的默认安装路径是 /usr/lib/jvm/java-1.4.2-sun 子目录。还应该在那里看到两个符号链接: 一个是从 /usr/lib/jvm/jre 链接到 /etc/alternatives/jre, 另一个是从 /etc/alternatives/jre 链接到 /usr/lib/jvm/java-1.4.2-sun。(利用这些符号链接, 就可以在同一台计算机上同时安装多种 Java 版本并在它们之间进行切换。) 下面是某个 SUSE 9.2 系统上的 *PATH* 变量的内容 (注意, 这里使用的分隔符是冒号):

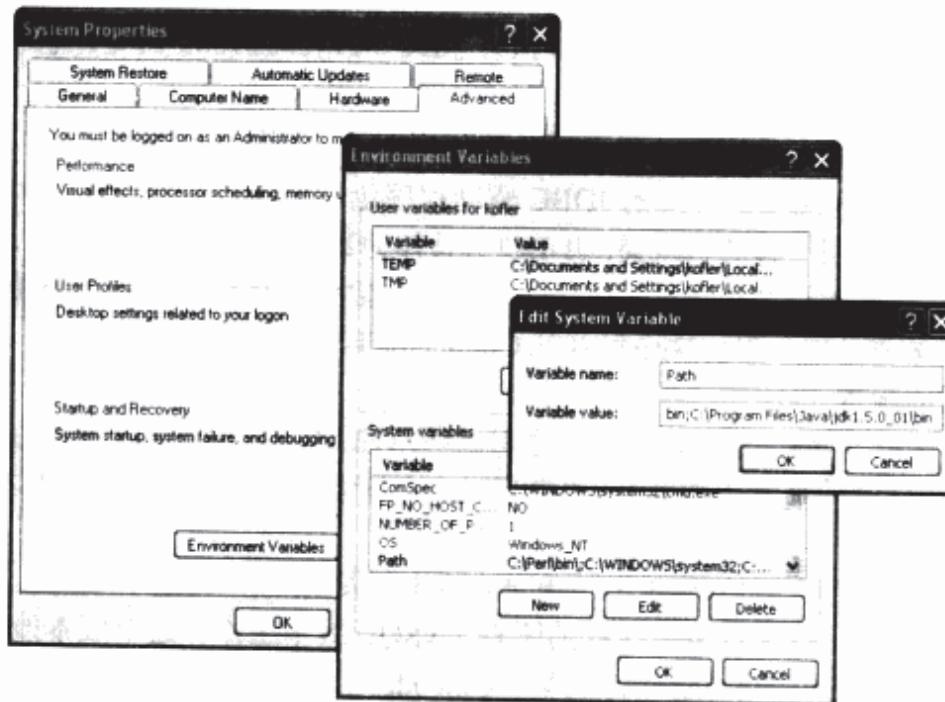


图 17-1 在 Windows XP 系统上设置 PATH 变量

```
linux:~ $ echo $PATH
/home/suse/bin:/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:
/usr/games:/opt/gnome/bin:/opt/kde3/bin:/usr/lib/jvm/jre/bin/
```

安装工作全部结束后，可以打开一个命令窗口（Windows 系统）或控制台窗口（Linux 系统），并执行以下命令来检查这次安装是否成功：

```
C:\> java -version
java version "1.5.0_01"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_01-b08)
Java HotSpot(TM) Client VM (build 1.5.0_01-b08, mixed mode, sharing)

linux:~ # java -version
java version "1.4.2_06"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_06-b03)
Java HotSpot(TM) Client VM (build 1.4.2_06-b03, mixed mode)
```

Hello, World!

如果想检查一下 Java 编译器能否正常工作，可以创建一个内容如下所示的 *Hellow.java* 文件。注意，这个文件的文件名必须与在代码里定义的 *Hellow* 类完全一致，连字母的大小写也必须一致。

```
/* example file Hellow.java */
public class Hellow
{
    public static void main(String[] args)
    {
        System.out.println("Hello, world!");
    }
}
```

接下来，切换到 *Hellow.java* 文件所在的目录，然后执行以下命令对它进行编译：

```
> javac Hellow.java
```

作为编译的结果，将得到一个 *Hellow.class* 文件。这个文件可以用以下命令来执行：

```
> java Hellow
Hello World!
```

注意，这里不能给出.class 后缀名。如果非要执行 java Hellow.class 命令，它将不工作。

17.1.2 Connector/J 的安装

Connector/J 是专为 MySQL 而开发的 JDBC 驱动程序包。JDBC 是人们为了使用 Java 语言开发各种数据库应用软件而打包在一起的一些类。JDBC 与具体的数据库系统无关。因此，如果想通过 JDBC 去连接某个特定的数据库系统，就必须使用一个专为这种数据库系统而开发的 JDBC 驱动程序包。对 MySQL 数据库系统来说，这个驱动程序包就是 Connector/J。

1. Connector/J 的版本

Connector/J 是所谓的第四类驱动程序，这句话的意思是说它的全部功能都是用 Java 实现的。Connector/J 要求系统必须安装有 JDBC 2.0 或更高版本（即至少是 Java 2 的 1.2 版本）。就目前而言，比较常见的 Connector/J 版本有 3 种：

- Connector/J 3.0，这是为 MySQL 3.23 和 4.0 版本开发的 Connector/J 版本。
- Connector/J 3.1，可以配合目前流行的所有 MySQL 版本工作。这个版本能够支持 MySQL 4.1 版的新增功能如新的密码身份验证机制、Unicode 字符集、预处理语句等。
- Connector/J 3.2，这是 2005 年 3 月推出的 alpha 测试版本。这个版本能够支持 MySQL 5.0 版的新增功能如光标函数等。在 Connector/J 3.2 版正式发布之前，应该尽量选用 Connector/J 3.1 版（即使需要配合 MySQL 5.0 工作也应该如此）。

注解 Connector/J 早期版本的名字叫做 MM.MySQL。这个驱动程序包是在其作者 Mark Matthew 加盟 MySQL 开发团队之后才正式更名为 Connector/J 的，它目前的技术支持工作由 MySQL 公司提供。Connector/J 采用 GPL 许可证发行。如果想把 Connector/J 用于商业目的，MySQL 服务器必须有相应的许可证。

2. Connector/J 的安装

第 7 章曾简单地提到过 Connector/J 的安装问题。本节是第 7 章有关内容的延续，但这次将会涉及更多的技术细节和安装选项。

从 www.mysql.com 网站可以下载到 Connector/J 软件包的*.zip(适用于 Windows 系统)和*.tar.gz(适用于 UNIX/Linux 系统)压缩文档。这两个压缩文档的内容其实是一些同样的文件。(Java 是与平台无关的。)作为安装 Connector/J 的第一步，必须先把下载回来的压缩文档释放到一个选定的子目录里。在 Windows 系统上，可以使用 Windows Explorer(即 Windows 的资源管理器)或 WinZip 软件来完成这一任务。在 Linux 系统上，将需要执行以下命令：

```
linux:~ # tar -xzf mysql-connector-java-n.tar.gz
```

表 17-1 对 Connector/J 压缩文档里最重要的文件和子目录进行了汇总。

表 17-1 Connector/J 压缩文档里的目录和文件

目 录/文 件	内 容
mysql-connector-java-n.n.n/	实际存放各种库文件的子目录；这份清单里的其他文件路径都是相对这个目录而言的；n.n.n 是版本号
/mysql-connector-java-n.jar	一个 Java 档案文件，其内容是这个驱动程序包的所有 Java 类。这个文件就是真正的驱动程序；n 是版本号

(续)

目录/文件	内 容
/com/*	这个目录存放着从压缩文档释放出来各个文件，它们分别对应着这个驱动程序包里各个 Java 类
/docs/*	PDF 和 HTML 格式的帮助文档，它们的数量非常多
/org/*	这个驱动程序包在它自己的老名字 (MM.MySQL) 下的 Java 起始类 (org.gjt.mm.mysql.Driver)

这里的关键之处是必须让 Java 引擎能够在执行 Java 程序时找到新安装的驱动程序库。为了确保这一点，可以有以下几种选择：

- 最简单的办法是把 mysql-connector-java-n.jar 文件复制到 Java 安装目录 jre\lib\ext 里去，Java 程序在执行时会自动到这个地方来寻找驱动程序。这里需要解决的一个小问题是 Java 到底安装在计算机上的什么地方。这个文件的典型安装路径是：在 Windows 系统上，C:\Programs\Java\jre1.5.0_01；在 SUSE 系统上，/usr/lib/jvm/java-1.4.2-sun/lib/ext。
 - 第二种办法是设置或修改 *CLASSPATH* 环境变量。Java 程序在执行时会到这个环境变量所列出的各个目录里去寻找驱动程序。也就是说，如果想让 Java 程序使用 Connector/J，就必须把 mysql-connector-java-n.jar 文件所在的目录添加到这个环境变量里。这里有一点需要大家特别注意：如果想让自己能够执行当前目录里的 Java 程序，就必须让这个环境变量包含着路径“.”（这是一个英文句号字符，其含义是“当前目录”）。
- 在 Windows 系统上，可以用 DOS 命令 *SET var = xxx* 临时改变 *CLASSPATH* 变量的设置值。如果想永久地改变它，在 Windows 2000/XP 系统上可以通过菜单命令 Control Panel (控制面板) | System (系统) | Advanced (高级) | Environment Variables (环境变量) 打开的对话框进行，在 Windows 9x/Me 系统需要通过 autoexec.bat 文件去进行——*CLASSPATH* 变量里的路径必须用分号 (;) 隔开。
- 在 Linux 系统上，可以用命令 *export var = xxx* 临时改变 *CLASSPATH* 变量的设置值，对它进行永久性修改需要通过/etc/profile 文件来进行——*CLASSPATH* 变量里的路径必须用冒号 (:) 隔开。
- 如果只是出于测试目的，还可以采用这样一种临时办法：把 Connector/J 软件包里的 com 和 org 目录复制到本地子目录（也就是开发的 Java 程序所在的那个目录）。

最后，可以用下面这个小程序来测试 Connector/J 是否安装成功。它应该不返回任何出错消息：

```
/* example file HelloMySQL.java */
import java.sql.*;
public class HelloMySQL {
    public static void main(String[] args) {
        try {
            Driver d = (Driver)
                Class.forName("com.mysql.jdbc.Driver").newInstance();
            System.out.println("OK");
        } catch(Exception e) {
            System.out.println("Error: " + e.toString());
        }
    }
}
```

3. Connector/J 安装失败的常见原因

如果有什么地方不对劲，首先应该检查是不是因为 Java 引擎找不到 Connector/J 驱动程序的缘故；与此有关的典型出错消息是：*java.lang.ClassNotFoundException: com.mysql.jdbc.Driver*（未能找到

JDBC 驱动程序)。这类问题的根源主要有以下几种:

- 如果当初采用的是把 mysql-connector-java-n.n.n.jar 文件复制到 jre\lib\ext 目录里的办法, 问题的根源很可能是计算机里安装了一个以上的 Java 解释器(以前安装过一个正式版, 而这次又安装了一个开发版)。比如说, 很可能把最新的 mysql-connector-java-3.0.n.jar 文件复制到了以前的 Java 安装目录里。检查一下计算机, 然后再重新复制一次试试。

在 Linux 系统上, 可以用 which java 命令轻而易举地查出正确的 Java 目录应该是哪一个。但要注意的是, 有些 Linux 发行版本里的这个命令将返回一个指向文件实际存放路径的符号链接而不是那个路径本身。比如说, 在 SUSE Linux 系统上, 必须连续追踪两个符号链接才能查出 Java 解释器的实际存放地点; 如下所示:

```
linux$ which java
/usr/bin/java
linux$ ls -l /usr/bin/java
... /usr/bin/java -> /etc/alternatives/java
linux$ ls -l /etc/alternatives/java
... /etc/alternatives/java -> /usr/lib/jvm/jre-1.4.2-sun/bin/java
```

在 Windows 环境里没有这么方便的命令可用, 与此有关的设置信息都保存在 Windows 的注册表里。但这里有一个相对简单得多的办法: 通过菜单命令 Start(开始)|Settings(设置)|Control Panel(控制面板)|Add/Remove Software(添加/删除程序) 打开的对话框去查看已经安装了哪几种 Java 版本以及它们的安装都发生在什么时候。

- 如果当初采用的是修改 CLASSPATH 环境变量的办法, 问题的根源往往不会是 Java 引擎找不到 Connector/J 驱动程序, 而是因为它找不到所编写的 Java 程序的类。这种情况的典型出错消息是 java.lang.NoClassDefFoundError(类没有定义)。检查 CLASSPATH 环境变量的定义, 尤其要注意那个代表着“当前目录”的“.”(英文句号) 符号有没有在里面。

在 Windows 系统上, 可以在一个命令窗口里用 echo %CLASSPATH% 命令来查看 CLASSPATH 环境变量的设置情况; 在 Linux 系统上, 可以在一个控制台窗口里用 echo \$CLASSPATH 命令来进行这一检查。Windows 系统上的 PATH 变量使用分号(;)作为分隔符, Linux 系统上的 PATH 变量使用冒号(:)作为分隔符。下面的命令给出了一些可能的设置(当然, 应该根据自己的系统配置情况来检查 PATH 变量):

```
> ECHO %CLASSPATH%
.;C:\Programs\mysql-connector-java-3.0.3-beta

linux: $ echo $CLASSPATH
.:/usr/local/mysql-connector-java-3.0.3-beta/
```

17.2 程序设计技巧

以下示例都需要假设 Java SDK 和 Connector/J 都已经正确地安装好了。这里给出的所有示例都只能运行在文本模式下。采用 AWT 或 Swing 接口编写出来的 Java 程序看起来当然会比较赏心悦目, 但这里只是为了演示程序设计技巧, 文本模式已经足够用的了, 更何况这么做还有让程序代码更为简明易懂的好处。

提示 关于 Connector/J 的更多信息可以在与之有关的在线文档里查到, 它们有 PDF 和 HTML 等多种格式, 在因特网上也能查到这方面的信息: <http://dev.mysql.com/doc/connector/j/en/index.html>。

每一个使用了 Connector/J 的 Java 程序同时也是一个 JDBC 程序。(JDBC 是 Java 的数据库 API 接口。MySQL JDBC 程序只在很少的细节方面是仅适用于 MySQL 的。

因为本书的主题是 MySQL 而不是 Java 和 JDBC，所以在此只能对 JDBC 程序设计问题做一个非常简单的介绍。关于 Java 和 JDBC 更多和更详细的信息可以在专门讨论它们的书刊里查到。

17.2.1 第一个示例

下面这个例子可以让大家对 JDBC 程序的构造过程有一个直观的印象。`import java.sql` 指令的作用是让用户可以在后面的代码里使用 JDBC 提供的各种接口和类。`Class.forName("...").newInstance()` 方法的作用是加载 Connect/J 驱动程序。(这个调用必须发生在与数据库服务器建立连接之前。)

与数据库服务器建立连接的工作由 `DriverManager.getConnection()` 方法负责具体完成，它最重要的几个参数（驱动程序的名字、主机名、数据库名）是以一个 URL 地址字符串的形式传递给它的；关于这个方法以及这种 URL 字符串的详细说明见下一小节。

要想执行一条 SQL 命令，必须先创建一个 `Statement`（意思是“语句”）对象，这类对象的 `executeQuery()` 方法将返回一个 `ResultSet`（意思是“结果集”）对象，这个对象包含着本次查询的结果。在下面的例子里，有关代码将依次读入每一条结果记录 (`next()` 方法)、从中有选择地提取出一些数据 (`getInt()`、`getString()` 等方法)、然后把它们输出到了一个控制台窗口里：

```
/* example file SampleIntro.java */
import java.sql.*;
public class SampleIntro
{
    public static void main(String[] args)
    {
        try {
            Connection conn;
            Statement stmt;
            ResultSet res;
            // load the Connector/J driver
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            // establish connection to MySQL
            conn = DriverManager.getConnection(
                "jdbc:mysql://uranus/mylibrary", "username", "xxx");
            // execute SELECT query
            stmt = conn.createStatement();
            res = stmt.executeQuery(
                "SELECT publID, publName FROM publishers " +
                "ORDER BY publName");
            // process results
            while (res.next()) {
                int id = res.getInt("publID");
                String name = res.getString("publName");
                System.out.println("ID: " + id + " Name: " + name);
            }
            res.close();
        }
        catch(Exception e) {
            System.out.println("Error: " + e.toString());
        }
    }
}
```

下面是这个示例程序的运行结果：

```
ID: 1 Name: Addison-Wesley
ID: 2 Name: Apress
ID: 9 Name: Bonnier Pocket
ID: 5 Name: Hanser
```

17.2.2 与 MySQL 服务器建立连接

1. 用 *DriverManager.getConnection()* 方法建立连接

正如刚才提到的，加载 Connect/J 驱动程序的操作动作必须发生在与数据库服务器建立连接之前。这个操作动作是用 JDBC 驱动程序的名字作为参数去调用 *Class.forName()* 方法而完成的。这个方法将返回一个 *Driver* 对象，但需要用到这个对象本身的时候并不多，重要的是 Connect/J 将作为这个函数调用的结果而被注册为一个 JDBC 驱动程序的事实，因为它必须经过注册才能使用。如下所示：

```
// SampleConnection1.java
// load Connector/J driver
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

在接下来的 Java 代码里，最引人注目的东西应该是 *DriverManager.getConnection()* 方法创建的 *Connection*（连接）对象。在调用这个方法的时候，通常需要向它传递 3 个字符串参数，它们的内容分别是：URL（uniform resource locator，统一资源定位器）格式的数据库基本信息、打算使用的 MySQL 用户名和相应的密码。如下所示：

```
// connection to MySQL
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://uranus/mylibrary", "username", "password");
```

getConnection() 方法的 URL 参数字符串的语法如下所示（这里把它写成了两行）：

```
jdbc:mysql://[host1][,host2...][:port]/
[dbname][?para1=val1][&para2=val2][&para3=val3]...
```

表 17-2 和表 17-3 对这种 URL 参数字符串的各个组成部分和几个最重要的可选连接选项进行了描述。允许用在这种 URL 参数字符串里的参数清单可以在 Connect/J 的 *Readme* 文件里查到。

表 17-2 *getConnection()* 方法的 URL 参数字符串的组成部分

组成部分	用 途
<i>jdbc:mysql://</i>	选定一个 JDBC 驱动程序（对 MySQL 数据库而言，就是 Connect/J）
<i>host1</i>	选定一个主机名（如果 Java 与 MySQL 服务器运行在同一台计算机上，也可以用 <i>localhost</i> 来给出这个主机名）
<i>host2, host3</i>	给出其他（可选）的主机名；它们的含义是：如果主机 <i>host1</i> 无法连接，则尝试连接 <i>host2</i> ，以此类推
<i>:port</i>	设定与数据库进行通信的端口（对 MySQL 数据库而言，默认端口是 3306）
<i>/dbname</i>	指定一个默认使用的数据库
<i>?para1=value1</i>	可选参数当中的第 1 个（详见表 17-3）
<i>&para2=value2</i>	第 1 个可选参数之后的其他参数（第 2 个、第 3 个等）

表 17-3 *getConnection()* 方法的 URL 参数字符串中的可选参数（以及它们的默认设置值）

参 数	作 用
<i>allowMultiQueries=false</i>	是否允许一次执行多条 SQL 命令。如果使用了这个选项，就必须用分号把多条 SQL 命令分隔开。这个选项始见于 Connect/J 3.1.1 版本
<i>autoreconnect=false</i>	是否允许 JDBC 驱动程序在与数据库的连接掉线时自动重新建立连接

(续)

参数	作用
<i>connectTimeout=0</i>	等待连接建立成功的倒计时时间。如果把这个选项设置为 0，则表示不进行倒计时。这个选项只能与 Java 2 的 1.4 及更高版本配合使用
<i>initialTimeout=2</i>	重建一条掉了线的连接时，前、后两次尝试之间的等待时间（秒）。这个选项不影响首次连接数据库的尝试
<i>maxReconnects=3</i>	重建连接时的最大尝试次数
<i>useCompression=false</i>	Java 程序与 MySQL 服务器在通信时是否需要对数据进行压缩
<i>relaxAutoCommit=false</i>	如果 Java 程序访问的是一个不支持事务的 MySQL 数据表（比如因为那个数据表是 MyISAM 格式），Connector/J 是否默默地接受 COMMIT 和 ROLLBACK 命令而不报告出错
<i>useTimezone=false</i>	是否需要调整客户端与服务器之间的地理时差

后续的数据库操作都需要通过 *Connection* 对象来进行。如果想在 Java 程序结束时关闭当前连接，简单地调用一次当前 *Connection* 对象的 *close()* 方法即可。

2. 用 *DataSource.getConnection()* 方法建立连接

从 Java 2 的 1.4 版开始，除了上面刚介绍的 *DriverManager.getConnection()* 方法，Java 程序在与数据库建立连接的时候又多了一种选择：*DataSource* 类 (*javax.sql* 库)。JDBC 文档推荐使用这种办法，因为它能支持更多的高级功能，比如数据库连接的缓冲排队功能 (connection pooling)、分布式事务功能 (用同一个数据库操作对来自多个数据库的数据进行处理) 等。

下面的示例不涉及那些高级功能，它只是简单地演示了如何使用 *DataSource.getConnection()* 方法去建立与数据库的连接。首先，需要创建一个 *com.mysql.jdbc.jdbc2.optional.MysqlDataSource* 类的对象，然后用 *setServerName()* 和 *setDatabaseName()* 方法分别设置一个主机名和一个数据库名，最后用 *getConnnection()* 方法去建立与数据库的连接。在调用 *getConnnection()* 方法的时候只须给出用户名和密码就行了。这个方法将返回一个 *Connection* 对象，后续的数据库操作都需要通过这个 *Connection* 对象来进行。

如果需要设置一些可选的连接参数，可以使用 *setURL()* 方法 (URL 参数字符串的格式见上一小节的讨论) 或者向 *getConnection()* 方法传递一个名为 *Properties* 的参数表：

```
// SampleConnection2.java
import java.sql.*;
import javax.sql.*; // for DataSource
public class SampleConnection2
{
    public static void main(String[] args)
    {
        try {
            com.mysql.jdbc.jdbc2.optional.MysqlDataSource ds;
            Connection conn2;
            // create connection with a DataSource object
            ds = new com.mysql.jdbc.jdbc2.optional.MysqlDataSource();
            ds.setServerName("uranus");
            ds.setDatabaseName("mylibrary");
            conn2 = ds.getConnection("root", "xxxx");
            // and so on, as before ...
        }
        catch(Exception e) {
            System.out.println("Error: " + e.toString());
        }
    }
}
```

提示 如果想使用 JDBC 的高级功能，应该先到 Connector/J 安装目录下的 com\mysql\jdbc\jdbc2\optional 目录里检查一下有没有安装相应的 Connector/J 对象类。在那里将找到（比如说） MysqlPooledConnection 类和 MySQLConnectionPoolDataSource 类的 Java 源代码。

17.2.3 连接 MySQL 服务器时可能遇到的问题

一般来说，如果客户程序与 MySQL 服务器没有运行在同一台计算机上，它们之间的连接将通过 TCP/IP 协议来建立。反过来讲，如果客户程序与 MySQL 服务器运行在同一台计算机上而那台计算机又是一台 UNIX/Linux 系统，它们之间的连接将通过一个套接字文件来建立；套接字文件比 TCP/IP 协议的效率高。

可是，Java 不支持套接字，所以 Java 程序在连接各种服务器的时候总是使用 TCP/IP 协议。这意味着在连接 MySQL 数据库的时候必须使用主机的真实名字而不是 *localhost* 作为主机名。在这个地方犯错误往往会引起很多麻烦，与此有关的典型出错消息如下所示：

```
linux:~/ java SampleConnection
Error: java.sql.SQLException: Server configuration denies access
to data source
```

这种失误很不容易查找，因为使用同样的连接参数发出 mysql -h localhost -u name -p 命令可以成功地建立起一条工作正常的本地连接。导致这种差异的根源是 mysql 命令在尝试建立一条本地连接的时候将默认地使用套接字文件来进行，但 Java 却不是这样。

下面几个步骤可以帮助大家解决这类问题。（有许多原因会导致 MySQL 服务器无法访问，有助于解决这类问题的更多技巧可以在本书的第 11 章找到，但那些技巧大都与 Java 无关。）为了便于讨论，我们将使用一个主机名为 *uranus.sol*、IP 地址为“192.168.0.2”的 MySQL 服务器作为例子；Java 程序就运行在这同一台计算机上。

- 出于信息安全方面的考虑，有不少 MySQL 服务器不接受来自网络的连接请求（这需要在启动 MySQL 服务器的时候用上--skip-networking 选项或者是在 MySQL 配置文件 my.cnf 里对这个选项做出相应的修改）。遇到的问题是否属于这种情况可以用如下所示的 mysql 命令来判断：

```
linux:~/ $ mysql -h uranus.sol -u name -p database
Enter password: xxxxx
ERROR 2003: Can't connect to MySQL server
```

对于这种情况，唯一的解决办法是删除这个选项。

- 另一种情况是 MySQL 服务器接受来自网络的连接请求，但负责访问控制的 *mysql.user* 数据表在它的 *hostname* 数据列里有一项无效或非法的设置。遇到的问题是否属于这种情况可以用如下所示的 mysql 命令来判断（请注意，它这次返回的出错消息与刚才不一样）：

```
linux:~/ $ mysql -h uranus.sol -u name -p database
Enter password: xxxxx
ERROR 1130: Host '192.168.0.2' is not allowed to connect
to this MySQL server
```

解决这类问题的办法是：先用 resolveip 192.168.0.2 命令查出与这个 IP 地址相对应的主机名（在笔者的计算机上，这条命令的返回结果是 *uranus.sol*），然后去检查 *mysql.user* 数据表里是否有一条与此对应的记录项（应该看到一个完整的主机名而不是它的一部分，如 *uranus*）。

- 还有一种情况是 *mysql.user* 数据表也没有问题，但 Java 程序给出的用户名和密码不匹配。遇到的问题是否属于这种情况可以用如下所示的 mysql 命令来判断（请注意，它这次返回的出

错消息与前两次不一样):

```
linux~/mysql -h uranus.sol -u name -p databaseName
Enter password: xxxxx
ERROR 1045: Access denied for user: 'root@192.168.0.2'
(Using password: YES)
```

经过以上步骤之后, 如果 mysql 命令能用给出的主机名、用户名和密码成功地连接上 MySQL 服务器, 在 Java 程序里使用同样的主机名、用户名和密码去连接 MySQL 服务器就应该没有问题了。

17.2.4 执行 SQL 命令

要想在 Java 程序里执行 SQL 命令, 必须先用 *conn.createStatement()* 方法获得一个 *Statement*(语句) 对象:

```
Statement stmt = conn.createStatement();
```

如果打算执行的是 *SELECT* 查询命令, 还可以通过两个可选的参数来选择一种光标类型和设置 *SELECT* 命令所返回的结果数据是否允许修改:

```
stmt = conn.createStatement(resultcursortype, resultconcurrency);
```

在默认的情况下, 光标类型将是 *forward only* (只能前进), 结果数据则是 *read only* (只允许读取)。对这两个参数的详细介绍见本章后面的有关内容。

Statement 类目前提供了以下几种 *execute* 方法。

- *executeUpdate*: 用来执行 *INSERT*、*UPDATE* 和 *DELETE* 命令。
- *executeQuery*: 用来执行 *SELECT* 查询命令。
- *executeBatch*: 用来执行此前用 *addBatch()* 方法定义的一个由多条 SQL 命令构成的语句块 (这有助于提高效率)。
- *execute*: 这个方法可以用来执行任何一种 SQL 命令 (这在无法事先确定将会执行哪一种 SQL 命令的时候非常有用)。

接下来, 将对 *executeUpdate*、*executeQuery* 和 *executeBatch* 方法做详细的介绍。

1. 执行 *INSERT*、*UPDATE* 和 *DELETE* 命令

执行 *INSERT*、*UPDATE* 和 *DELETE* 命令的办法很简单: 把 SQL 命令字符串直接传递给 *executeUpdate* 方法即可:

```
stmt.executeUpdate(
    "INSERT INTO publishers (publName) VALUES ('new publisher')");
```

executeUpdate 方法将返回一个 *int* 值告诉用户刚才执行的那条命令修改了多少条记录。

注意, 如果在 SQL 命令里使用了包含着单引号 ('), 双引号 ("), 反斜线 (\) 等特殊字符的字符串, 就必须使用预处理语句来执行它们 (我们马上就会讨论到预处理语句的问题)。

2. 确定新记录的 ID 编号值 (*AUTO_INCREMENT*)

有好几种办法可以用来确定一条新记录的 *AUTO_INCREMENT* 编号值。哪一种办法更好需要取决于正在使用的 Java 版本。

变体 1: Java 1.4 及更高版本里的 *getGeneratedKeys* 方法。如果使用的是 Java 的最新版本, 最好、最有效率和最有可移植性的办法就是使用 *getGeneratedKeys* 方法。这个方法将返回一个 *ResultSet* 对象 (*ResultSet* 的意思是“结果集”; 将在 17.2.5 节里对它做详细介绍)。

如果刚才只插入了一条新记录, *ResultSet* 对象将只包含一条结果记录, 其内容就是我们想知道

的 ID 编号值。为了提取出这个编号值，必须先调用一次 *next* 方法（这是为了找到 *ResultSet* 对象里的第一条记录），然后再调用 *getInt(1)* 方法读取这个 ID 编号（参数 1 表示要读取结果记录的第一个数据列）：

```
// SampleGetID1
stmt.executeUpdate("INSERT ...");
ResultSet newid = stmt.getGeneratedKeys();
newid.next();
int id = newid.getInt(1);
```

如果 *INSERT* 命令一次插入了许多条新记录，*getGeneratedKeys* 方法还可以返回所有的新 ID 编号值。但在此之前，为了让 JDBC 知道需要那些新 ID 编号值，必须在调用 *executeUpdate* 方法的时候把可选参数 *Statement.RETURN_GENERATED_KEYS* 传递给它。

```
stmt.executeUpdate(
    "INSERT INTO publishers (publName) VALUES ('publisher1'), ('publisher2')",
    Statement.RETURN_GENERATED_KEYS);
ResultSet newids = stmt.getGeneratedKeys();
while(newids.next()) { // returns the IDs of publisher1 and publisher2
    System.out.println("ID: " + newids.getInt(1));
}
```

变体 2：如果使用的 Java 版本比较老，可以使用 *Statement* 对象的 *getLastInsertID* 方法来获得最新的 *AUTO_INCREMENT* 值。不过，这个方法只在 Connector/J 的 *Statement* 类里有定义，*java.sql* 模块里的 *Statement* 类没有这个方法。因此，在使用这个方法之前，必须先用一个投射操作把 *Statement* 对象明确地转换为一个 *com.mysql.jdbc.Statement* 类的对象才行。（这个办法也适用于稍后将要介绍的 *PreparedStatement* 对象。）

getLastInsertID 方法的最大缺点是用它编写出来的代码没有可移植性，只能在“MySQL 服务器 + Connector/J 驱动程序”的应用环境里使用：

```
// SampleGetID2.java
Statement stmt = conn.createStatement();
stmt.executeUpdate(
    "INSERT INTO publishers (publName) VALUES ('new publisher')");
long id = ((com.mysql.jdbc.Statement)stmt).getLastInsertID();
```

变体 3：第 3 种变体（它最慢的变体）是在执行完 *INSERT* 命令之后再多执行一条 SQL 命令 *SELECT LAST_INSERT_ID*。注意，如果打算采用这个办法，就必须保证这两条命令将在同一条 MySQL 连接上的同一个事务里执行，这意味着如果正在与 *Innodb* 数据表打交道，就必须在开始执行这两条命令之前做出 *conn.setAutoCommit(false)* 设置，并在最后以 *conn.commit* 方法来结束事务：

```
// SampleGetID3.java
conn.setAutoCommit(false);
stmt.executeUpdate(
    "INSERT INTO publishers (publName) VALUES ('new publisher')");
ResultSet newid = stmt.executeQuery("SELECT LAST_INSERT_ID()");
if(newid.next()) {
    id = newid.getInt(1);
    System.out.println("new ID = " + id);
}
conn.commit();
```

17.2.5 处理 *SELECT* 查询结果

要想对 *SELECT* 命令的查询结果进行处理，就必须使用 *executeQuery* 方法来执行 SQL 命令。这个方法将返回一个 *ResultSet* 对象：

```

Statement stmt = conn.createStatement();
ResultSet res = stmt.executeQuery(
    "SELECT publID, publName FROM publishers " +
    "WHERE publID < 10 ORDER BY publName");

```

注意 如果有一些不是使用Connector/J驱动包开发出来的JDBC程序，就应该对ResultSet对象的默认属性给予高度注意：

首先，在默认的情况下，Connector/J的ResultSet对象是不允许编辑的（只读）。

其次，在默认的情况下，Connector/J的ResultSet对象支持自由遍历。这意味着MySQL服务器会把所有的结果记录一次性地传输到客户端，而且是查询结果再怎么多也会如此。这种行为与许多其他的JDBC驱动程序的做法——即ResultSet对象只允许向前遍历、结果记录在默认的情况下只在需要时才会被传输到客户端——是不同的。

如果想让ResultSet对象改变它们的默认行为，就必须在为有关的SQL查询命令创建Statement对象时明确地表明这一点。对这个问题的进一步讨论见本小节稍后的“只允许向前遍历ResultSet对象”和“把ResultSet对象当做变量来使用”部分。

*executeQuery*方法只负责执行SQL命令，不负责返回结果记录。为了读取第一条结果记录，必须执行一次对*res.next*方法的调用。后续的结果记录也需要用*next*方法依次取回，直到这个方法返回的结果是*false*为止。

对于当前结果记录，可以用*getInt*、*getString*等方法提取出它的各个字段（数据列），这些方法的参数既可以是数据列的编号（比如说，*res.getInt(1)*将返回当前结果记录的第一个字段），也可以是数据列的名字（比如像*res.getInt("publID")*这样）。后一种变体显然会让程序代码有更好的可读性，但在速度方面要稍微慢一些。

下面这个循环将依次读取前一条SELECT查询命令所查找出来的结果记录，并把它们显示在一个控制台窗口里：

```

while (res.next()) {
    int id = res.getInt("publID");
    String name = res.getString("publName");
    System.out.println("ID: " + id + " Name: " + name);
}

```

1. 无效查询和空查询

如果调用*executeQuery*方法去执行的SQL命令违反了有关的语法规则，这个方法在执行时将抛出一个*SQLException*异常。

如果SQL查询命令本身没有返回任何结果，*executeQuery*方法将返回一个空ResultSet对象（注意，不是空值NULL）。判断SELECT查询命令是否返回了结果的唯一办法是执行一次*res.next*方法，如果这个方法的返回值是*false*，就说明ResultSet对象是空的。

2. 对NULL值进行检查

返回值是Java基本数值类型的方法如*getInt*、*getFloat*等不能区分0和NULL，如果想知道这些方法读取的字段是不是包含着NULL，就必须使用*wasNull*方法，如下所示：

```

int n = res.getInt(1);
if(res.wasNull)
    System.out.println("n = [NULL]");
else
    System.out.println("n = " + n);

```

对于字符串方法的返回值，不需要使用 `wasNull` 方法去判断它们是不是 `NULL`。这是因为 `String` 类型的变量能够保存 `null` 值。换句话说，可以先执行 `String s = res.getString(...)`，再测试 `s==null` 是否成立。

3. 字符串/Unicode

在内部，Java 把所有的字符串都保存为 Unicode 格式。Connector/J 和 MySQL 服务器可以把来自 Java 程序的字符串自动转换为 MySQL 数据表的字符集。不过，这种自动转换功能只有 MySQL 4.1 和 Connector/J 3.1 以上的版本里才有。如果使用的是老版本，可以通过在 JDBC URL 参数字符串里设置有关参数的办法来选择需要的字符集。

4. 对结果集进行遍历

可以使用以下方法读取 SQL 查询结果里的记录，并把下次读取指针定位到当前结果记录之后：`next` 方法，前进到下一条结果记录；`previous` 方法，后退到上一条结果记录；`first` 方法，直接到达第一条结果记录；`last` 方法，直接到达最后一条结果记录。`beforeFirst` 或 `afterLast` 方法将把下次读取指针定位到第一条结果记录之前（这样就可以用 `next` 读取第一条结果记录了）或最后一条结果记录之后（这样就可以用 `previous` 方法读取最后一条结果记录了）。¹此外，还可以用 `isFirst` 和 `isLast` 方法来测试是否已经到达查询结果的开头或末尾。当前结果记录的编号（第一条结果记录的编号是 1）可以用 `getRow` 方法获得；`absolute(n)` 将直接到达编号为 `n` 的结果记录。

5. 确定结果记录的总数

因为 Connector/J 默认返回的都是允许自由遍历的 `ResultSet` 对象，所以确定结果记录的总数是一个非常简单的任务：先利用 `last` 方法直接到达最后一条结果记录，再利用 `getRow` 方法查出这条记录的编号即可：

```
res.last();
int n = res.getRow();
```

如果想从头开始遍历所有的结果记录，必须先把下次读取指针定位到第一条结果记录之前：

```
res.beforeFirst();
while(res.next()) ...
```

如果还想知道结果数据列的个数，将需要一个 `ResultSetMetaData` 对象（见下一小节里的讨论）。

6. 确定关于结果集的元数据

即使无法事先知道 `SELECT` 查询命令会返回什么样的数据（比如说，正在编写的是一个可以对任意数据表进行查询的通用性程序），也可以用一个 `ResultSetMetaData` 对象来获得关于 `SELECT` 查询结果的所有相关信息：数据列的个数（`getColumnName` 方法）、它们的名字（`getColumnLabel` 方法）、它们的数据类型（`getColumnType` 和 `getColumnTypeName` 方法）、数值精度和小数点位数（`getPrecision` 和 `getScale` 方法）。还可以用 `isNullable`、`isAutoincrement` 和 `isSigned` 方法查出各结果数据列是不是允许包含 `NULL` 值、是不是一个 `AUTO_INCREMENT` 数据列、是不是存放着带符号（+、-）数值等信息。所有这些方法都需要一个数据列编号作为参数，只须把某个结果数据列的编号传递给它们就可以获得想要的信息。

`ResultSetMetaData` 对象是由 `getMetaData` 方法创建的，这个方法需要一个 `ResultSet` 对象作为参数。下面这个简单的例子演示了 `ResultSetMetaData` 对象的使用办法：

```
int i, n;
ResultSetMetaData meta = res.getMetaData();
n = meta.getColumnCount();
```

1. 原文这里有严重错误。——译者注

```

System.out.println("number of columns: " + meta.getColumnCount());
for(i=1; i<=n; i++) {
    System.out.println("column " + i + ": " +
        " Name: " + meta.getColumnName(i) +
        " datatype: " + meta.getColumnTypeName(i));
}

```

7. 只允许向前进遍历ResultSet对象

在默认的情况下，*ResultSet* 对象允许自由遍历，即，可以随意使用 *previous*、*next* 等方法去读取 *SELECT* 查询结果里的记录。这的确很方便，但存在着这样一个不足：在 Java 程序执行 *executeQuery* 方法的时候，MySQL 服务器会把它找到的所有数据一次性地返回给 Java 程序。这不仅意味着需要传输大量的数据，还意味着 Java 程序必须保留大量的内存来容纳那些数据。如果只需要对结果记录依次进行处理，一次性地接收大量的数据显然不是最佳的办法。

如果想让结果记录陆续到达 Java 程序，就必须在为 *SELCT* 查询命令定义 *Statement* 对象的时候按如下所示的方式进行。（*Integer.MIN_VALUE* 是一个代表着最小可表示整数的 Java 常数。如果想表明只允许向前进遍历 *ResultSet* 对象，Connector/J 要求必须使用这个值作为 *setFetchSize* 方法的参数。）

```

Statement stmt = conn.createStatement(
    java.sql.ResultSet.TYPE_FORWARD_ONLY,
    java.sql.ResultSet.CONCUR_READ_ONLY);
stmt.setFetchSize(Integer.MIN_VALUE);
ResultSet res = stmt.executeQuery("SELECT ...");

```

注意 只允许向前进遍历*ResultSet*对象将产生以下影响：首先，将只能使用*next*方法来遍历*ResultSet*对象（这正是所谓“只能向前”的含义）；其次，如果需要在当前MySQL连接（*Connection*对象）上执行另一条SQL命令，就必须把前一次的查询结果全部读取完毕，或者先用*close*方法关闭当前连接、然后再使用相同的参数重新打开一条连接。

8. 把*ResultSet*对象当做变量来使用

如果想编辑数据表里的数据，最普通的方法是使用 *INSERT*、*UPDATE* 和 *DELETE* 等 SQL 命令。JDBC 提供了另一种解决方案：直接在 *ResultSet* 对象里插入、编辑或删除数据记录，由 JDBC 把做的这些修改写入相关的数据表里。这个功能很方便，但要想使用它必须满足以下几个条件：

- 在创建 *Statement* 对象时必须使用 *ResultSet.CONCUR_UPDATABLE* 属性。
- *SELECT* 查询命令只能涉及一个数据表（不允许使用 *JOIN* 关联操作）、不允许使用 *GROUP* 函数、必须包括主键索引。

如果这些条件都能得到满足，就可以用 *deleteRow* 方法从 *ResultSet* 对象和底层的数据库里删除当前结果记录了。

如果想编辑当前记录，先执行 *updateInt(n, 123)*、*updateString(n, "new text")* 等方法，最后用 *updateRow* 方法确认这些修改。

如果想插入一条新记录，先执行 *moveToInsertRow* 方法，然后用各种 *update* 方法存入有关数据，最后用 *insertRow* 方法确认这次插入操作。

如果想知道新记录的 ID 编号值 (*AUTO_INCREMENT*)，先执行 *last* 方法把新记录选为当前记录、再用 *getInt(n)* 方法提取出它的 *AUTO_INCREMENT* 字段值即可。

在下面这段程序里，将先从 *mylibrary* 数据库把 *publishers* 数据表里的全部记录选取出来，然后插入一条新记录，再对新记录进行一些编辑，最后把这条记录再删除掉。因为新记录在插入 *ResultSet* 对象之后就不允许再做修改了，所以还需要在插入新记录之后、编辑新记录之前对 *ResultSet* 对象进行

一次刷新。

```
// SampleChangeResultSet.java
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                           ResultSet.CONCUR_UPDATABLE);

// insert new publisher; then display all publishers
res = stmt.executeQuery(
    "SELECT publID, publName FROM publishers ORDER BY publID");
res.moveToInsertRow();
res.updateString(2, "New publisher");
res.insertRow();
res.last();
int newid = res.getInt(1);
res.beforeFirst();
while (res.next())
    System.out.println(res.getString(1) + " " + res.getString(2));
res.close();

// change previously inserted new publisher
res = stmt.executeQuery(
    "SELECT publID, publName FROM publishers WHERE publID = " + newid);
res.next();
res.updateString(2, "new with another name");
res.updateRow();

// ... and delete
res.last();
res.deleteRow();
res.close();
```

9. 关闭ResultSet对象

如果不再需要某个 *ResultSet* 对象了，应该尽早使用 *close* 方法明确地关闭它。之所以要尽早关闭 *ResultSet* 对象，是因为这样可以尽早释放被它占用的客户端内存。不仅如此，在只允许向前遍历 *ResultSet* 对象的情况下，如果在遍历全部结果记录之前调用了 *close* 方法，这个对象在 MySQL 服务器上占用和阻塞的资源也将得到释放。

17.2.6 预处理语句

预处理语句（prepared statement）使用户可以利用参数（或者叫占位符）来构造 SQL 命令，那些占位符将在 SQL 命令执行之前被替换为它们的实际值。这么做的好处是 JDBC 将替我们把字符串和二进制数据自动转换为符合 MySQL 语法要求的格式。这些转换包括在字符串的两端加上引号、把单引号 ('') 和反斜线 (\) 等特殊字符转义为 “\” 和 “\\” 等。

预处理语句还可以提高 SQL 命令的执行速度，因为 SQL 代码只须向数据库服务器传递一次，以后只要向数据库服务器传递一些必要的参数值就够了。

要想使用预处理语句，必须先用 *prepareStatement* 方法创建一个 *PreparedStatement* 对象。注意，预处理语句的字符串参数不能放在引号里（即只能是 *VALUE(?)*，不能是 *VALUE('?)*）。

向预处理语句传递参数值的工作可以用 *setString*、*setInt*、*setNull*、*setDate* 和 *setBinaryStream* 等许多方法（它们的完整清单可以在 JDBC 文档介绍 *PreparedStatement* 对象的章节里查到）来进行。

在执行经过预处理的 SQL 命令时，必须根据 SQL 命令的具体类型选择 *executeUpdate*、*executeQuery*、*addBatch* 和 *executeBatch* 等方法来执行它们；这些方法与 *Statement* 对象上的同名方法用途相同。如果说有什么新东西的话，那就是在执行预处理语句的时候不再需要向这些方法传递参数，

这是因为：SQL 命令本身已经在 *PreparedStatement* 对象里得到了定义，SQL 命令的参数也已经由各种 *setXxx* 方法设置好了。

在下面的例子里，将把两家新出版公司插入到 *mylibrary* 数据库中的 *publishers* 数据表里去：

```
// SamplePreparedStatement.java
PreparedStatement pstmt = conn.prepareStatement(
    "INSERT INTO publishers (publName) VALUES (?)");
pstmt.setString(1, "O'Reilly"); // inserts O'Reilly
pstmt.executeUpdate();
pstmt.setString(1, "\\\abc\\\"efg"); // inserts \abc"efg
pstmt.executeUpdate();
```

17.2.7 事务

JDBC 通过 *Connection* 对象提供了对事务的支持。当然，MySQL 数据表必须是支持事务的格式（目前只有 InnoDB 数据表支持事务）才能使用事务。

在默认的情况下，JDBC 将处于 AutoCommit 模式：每条 SQL 命令都是一个独立的事务并立刻得到执行。AutoCommit 模式的当前设置情况可以用 *getAutoCommit* 方法来确定，用 *setAutoCommit* 方法来改变。在开始一个事务之前，应该先用 *setAutocommit(false)* 明确地关闭 AutoCommit 模式，在一个事务的最后，必须调用 *commit* 或 *rollback* 方法对这个事务进行确认或撤销；这两条指令在结束上一个事务的同时也将开始下一个事务。

下面是一个简单的应用示例：把几条 *INSERT* 命令放在一个 *try* 结构里加以执行，只要发生错误，整个事务就将被撤销：

```
// no AutoCommit after every command
conn.setAutoCommit(false);

// execute several commands
try {
    stmt.executeUpdate("INSERT INTO table1 ...");
    stmt.executeUpdate("INSERT INTO table2 ...");
    conn.commit();
}
catch(Exception e) {
    conn.rollback();
}
```

17.2.8 批处理命令

JDBC 提供了把多条 SQL 命令当做一个语句块来执行的可能性。这种语句块的数据库术语是“批处理命令”。与一条一条地执行 SQL 命令相比，批处理命令的最大好处是效率更高（当然，这还要取决于具体的应用）。

使用批处理命令需要经过两个步骤：首先，用 *addBatch* 方法把所有将要执行的 SQL 命令传递到一个 *Statement* 或 *PreparedStatement* 对象里去；然后，用 *executeBatch* 方法把那些命令一次性执行完毕。

executeBatch 方法将返回一个 *int* 数组，该数组的各个元素分别是每条 SQL 命令所修改的数据记录个数（一个大于或等于 0 的整数）。如果这个数字无法确定，但与之相应的 SQL 命令确实执行成功了，这个数组里相应的返回值将是 *SUCCESS_NO_INFO*；如果 SQL 命令在执行时发生了错误，相应的返回值将是 *EXECUTE_FAILED*。（注意，如果发生的是语法错误，这组批处理命令的执行将半途而废并抛出一个异常。）

下面是一个简单的批处理命令应用示例：

```

int i;
stmt = conn.createStatement();
stmt.addBatch("INSERT INTO publishers (publName) VALUES ('publ1')");
stmt.addBatch("INSERT INTO publishers (publName) VALUES ('publ2')");
stmt.addBatch("INSERT INTO publishers (publName) VALUES " +
    "('publ3'), ('publ4')");
int[] n = stmt.executeBatch();
for(i=0; i < n.length; i++)
    System.out.println(
        "Recordsets changed by batch command no " + (i+1) + ": " + n[i]);

```

如果在执行期间没有发生任何错误，这组批处理命令将把以下文本显示在一个控制台窗口里：

```

Recordsets changed by batch command no 1: 1
Recordsets changed by batch command no 2: 1
Recordsets changed by batch command no 3: 2

```

注解 不要把批处理命令和事务混为一谈。批处理命令只适用于不需要在本组批处理命令的整个执行过程中把修改过的数据再读出来的场合。换句话说，批处理命令最好全部由`INSERT`或`UPDATE`命令组成，而且最适合用在既不需要确定`AUTO_INCREMENT`编号值、也不需要穿插进行`SELECT`查询的场合。此外，批处理命令本身没有提供任何出错处理机制，如果在批处理命令的执行过程中发生错误，此前已经执行完的所有SQL命令做出的修改都将生效，尚未执行的SQL命令将不再执行。

批处理命令是一种能够更高效地执行大量`INSERT`或`UPDATE`命令的手段，人们使用批处理命令的目的是追求效率。与此形成对照的是，事务是一种能够更安全地执行关键命令的手段，人们使用事务的目的是为了保证数据的完整和稳定。事务机制可以保证一个事务里的所有命令要么全部执行成功，要么就像这些命令从来都没执行过一样。

当然，我们完全可以把一组批处理命令安排在一个事务里执行。

17.2.9 二进制数据 (BLOB) 的处理

JDBC 提供了以下几种处理二进制数据的办法：

- 如果编程任务是从文件读出数据或是把数据写入文件，`ResultSet.getBinaryStream` 和 `PreparedStatement.setBinaryStream` 方法应该是最值得推荐的选择。如果是前一种情况，可以利用一个 `InputStream` 对象把二进制数据从数据库里读出来；如果是后一种情况，只须把一个现有的 `InputStream` 对象（它对应着一个打开的文件）作为参数传递给 SQL 命令，就可以把那个文件里的数据传输到数据库里。
- 如果编程任务是对包含着 Unicode 文本的 BLOB 字段进行处理，`getCharacterStream` 和 `setCharacterStream` 方法可以帮上大忙。如果把这两个方法和 `java.io.Reader` 对象结合起来，就能逐个字符地对大段文本进行选取。
- 如果需要对一些字节序列（也就是 Java 中的 `byte` 数据类型）进行处理，`getBytes` 和 `setBytes` 方法将是最明智的选择。
- 最后，JDBC 还提供了 `Blob` 和 `Clob` (character large object，字符串大对象) 接口：`Blob` 对象上的 `getBytes` 和 `setBytes` 方法可以让我们逐个字节地对有关数据进行读取和编辑；而 `Clob` 对象上的 `getAsciiStream`、`getCharacterStream` 等方法可以让我们方便地以字符串为单位对有关数据进行读取和编辑。在需要以 `Blob/Clob` 对象为单位来传递数据的场合，`getBlob`、`setBlob`、

getBlob 和 *setBlob* 等方法可以让编程工作大为简化。

1. 示例

下面这个例子演示了 *getBinaryStream* 和 *setBinaryStream* 方法的使用办法。示例程序将完成以下操作：先读取 *test.jpg* 文件的内容并把它存为 *exceptions* 数据库 *a_blob* 数据表里的一条新记录，然后从那里再把这条新记录读出来存入一个新创建的 *test-copy.jpg* 文件，最后删除 *a_blob* 数据表里这条新记录。

为了执行有关的 SQL 命令，我们先在这个示例程序里创建了几个 *PreparedStatement* 对象。为了把 *test.jpg* 文件的内容传递到 *a_blob* 数据表里，我们用 *setBinaryStream* 方法把对应于这个文件的 *FileInputStream* 对象作为参数传递给了第一个 *PreparedStatement* 对象 (*pstmt1*)。

从 *a_blob* 数据表读出有关数据并把它们存入 *test-copy.jpg* 文件的工作相对要复杂一些：先把有关数据从 *a_blob* 数据表查询到一个 *ResultSet* 对象 (*res*) 里，再用 *getBinaryStream* 方法从这个 *ResultSet* 对象里把数据提取到一个 *InputStream* 对象 (*is*) 里，最后利用一个循环和一个读写缓冲区把这个 *InputStream* 对象所包含的数据写入了一个 *FileOutputStream* (*fos*) 对象：

```
// example SampleBlob.java
import java.sql.*;
import java.io.*;
...

// make connection
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://uranus/exceptions", "root", "uranus");

// create three PreparedStatement objects
PreparedStatement pstmt1, pstmt2, pstmt3;
pstmt1 = conn.prepareStatement(
    "INSERT INTO test_blob (a_blob) VALUES(?)");
pstmt2 = conn.prepareStatement(
    "SELECT a_blob FROM test_blob WHERE id=?");
pstmt3 = conn.prepareStatement(
    "DELETE FROM test_blob WHERE id=?");

// read file test.jpg and store in a BLOB field
File readfile = new File("test.jpg");
FileInputStream fis = new FileInputStream(readfile);
pstmt1.setBinaryStream(1, fis, (int)readfile.length());
pstmt1.executeUpdate();
fis.close();

// determine id of the new records
long id = ((com.mysql.jdbc.Statement)pstmt1).getLastInsertID();
// create new, empty file copy-test.jpg
File writefile = new File("copy-test.jpg");
if(writefile.exists()) {
    writefile.delete();
    writefile.createNewFile();
}
FileOutputStream fos = new FileOutputStream(writefile);

// read BLOB field from the database
pstmt2.setLong(1, id);
ResultSet res = pstmt2.executeQuery();
res.next();
InputStream is = res.getBinaryStream(1);

// store binary data in the new file
final int BSIZE = 2^15;
```

```
int n;
byte[] buffer = new byte[BSIZE];
while((n=is.read(buffer, 0, BSIZE))>0)
    fos.write(buffer, 0, n);

// close open objects
is.close();
fos.close();
res.close();

// delete the new record
pstmt3.setLong(1, id);
pstmt3.executeUpdate();
```

2. 处理BLOB数据时的注意事项

MySQL 和 Java 都是把每一项 BLOB 数据当做一个整体传输给对方的。Connector/J 目前还不具备把 BLOB 数据分割为几块分次按需传输的能力。因此，在配置 Java 虚拟机（Java virtual machine, JVM）的时候，必须让它能够保留足够的空间，以便把一项 BLOB 数据完整地容纳在本地内存里。

此外，还必须把 MySQL 服务器配置成允许传输大量数据包的样子（与此相关的 MySQL 配置变量是 *max_allowed_packet*；详见第 22 章）。如有必要，可以用如下所示的命令在客户端加大这个变量的值：

```
SET max_allowed_packet=160000000
```

第18章**C 语 言**

本章将对如何使用 C 语言编写 MySQL 应用程序的问题做一个入门级介绍。具体地说，将介绍 MySQL 客户端软件开发库 `libmysqlclient`，这个函数库是 MySQL 的组成部分之一。本章介绍的编程技巧仅适用于 UNIX/Linux 系统，不适用于 Windows 系统。

在为这一章里的示例程序搭建测试系统的时候，笔者使用了自带 C 语言编译器 `gcc 3.3` 的 SUSE Linux Professional 9.2 以及从 `mysql.com` 网站下载的 MySQL 软件包（版本是 `5.0.n`）。如果用户使用的是另外一种 Linux 发行版本或其他操作系统，可能需要对有关的文件安装路径和系统配置参数做必要的调整。

18.1 MySQL C API (`libmysqlclient`)

MySQL C API 是开发 MySQL 应用程序最基础的 API (application programming interface，应用编程接口)，PHP、Perl 和 C++ 等其他一些程序设计语言的 MySQL API 都以 C API 为基础。因此，掌握这个 C API 有助于学习和使用其他的 API。这个 API 里的函数同时也是 MySQL 客户端软件开发库的组成部分，每一个需要访问 MySQL 数据库的 C 语言程序都不能不用到这个函数库。

目前仍在广泛使用的 `libmysqlclient` 函数库有好几个版本：版本 10，适用于 MySQL 3.23.*n* 系列；版本 12，适用于 MySQL 4.0.*n* 系列；版本 14，适用于 MySQL 4.1.*n*/5.*n.n* 系列。（本章内容将围绕版本 14 展开。）

在 `libmysqlclient` 函数库的这些不同版本里，基础性函数大部分是兼容的，但用来帮助实现各种新增功能的辅助性函数在兼容性方面就要差一些。这里有一个需要大家特别注意的问题：从 MySQL 4.1 版本开始，MySQL 内部使用的密码加密算法发生了变化，而 MySQL C API 是从版本 14 才开始支持这一变化的。换句话说，如果想让 C 语言程序与 4.1.*n* 或更高版本的 MySQL 服务器进行通信，就必须使用版本 14 或更高的 `libmysqlclient` 函数库来开发它们。（解决这一问题的根本办法是把 `libmysqlclient` 函数库升级到最新版本，但作为一种临时救急措施，可以在启动 MySQL 服务器时用上 `--old-password` 选项把它降级到使用老版本的 `libmysqlclient` 函数库。）

提示 对出现在本章内容里的 `libmysqlclient` 函数库的数据结构和函数的详细介绍可以在第 23 章查到。

18.2 Hello, World

18.2.1 对系统的基本要求

既然是在 Linux/UNIX 环境下编写 C 语言程序，C 语言编译器（通常是 `gcc`）和 `make` 程序当然不

不可缺少。除了它们，还必须把编译和链接 C 语言程序时需要用到的各种头文件和库文件在系统上安装好。有些 Linux 发行版本(比如 SUSE、Red Hat)还需要安装其他一些软件包，它们的名字大都以 -devel 结尾(例如 glibc-devel)。上述工作告一段落后，应该用一个简单的 C 语言程序来测试一下安装出来的系统是否完备。

既然是要开发 MySQL 应用程序，当然还需要把一些 MySQL 软件开发工具以及必要的头文件和库文件全部安装好。这些文件通常都可以在 MySQL-Devel-n 软件包里找到。在这一章的讨论过程中，我们将假设 MySQL 软件开发辅助文件都安装在以下两个文件路径里：头文件都安装在 /usr/include/mysql 目录下；库文件都安装在 /usr/lib/mysql 目录下。

如果这些文件在系统上被安装到了其他的位置，请对有关的编译和链接选项做出相应的修改。

18.2.2 入门级示例

为了让大家对使用 C 语言开发 MySQL 应用程序的过程有一个直观的认识，在这里准备了一个简短的入门级示例程序。如果想试用一下这个示例程序，必须先把 *mylibrary* 数据库安装到本地计算机上(具体安装步骤见附录 B)。当然，还需要对这段代码里 *mysql_real_connect()* 函数调用语句中的用户名("root")和密码("XXX")进行必要的修改。这段代码里的注释应该可以让用户了解在这个程序里发生的事情，而我们将在本章后面的内容里对这里用到的各种 MySQL 函数做出详细的解释。

```
// example file hellow/main.c
#include <stdio.h>
#include <mysql.h> // functions from libmysqlclient
int main(int argc, char *argv[])
{
    int i;
    MYSQL *conn;          // connection
    MYSQL_RES *result;   // result of the SELECT query
    MYSQL_ROW row;        // a record of the SELECT query

    // create connection to MySQL
    conn = mysql_init(NULL);
    if(mysql_real_connect(
        conn, "localhost", "root", "XXX",
        "mylibrary", 0, NULL, 0) == NULL) {
        fprintf(stderr, "sorry, no database connection ...\\n");
        return 1;
    }

    // only if Unicode output (utf8) is desired
    mysql_query(conn, "SET NAMES 'utf8'");

    // create list of all publishers and number of titles published for each publisher
    const char *sql="SELECT COUNT(titleID), publName \
                    FROM publishers, titles \
                    WHERE publishers.publID = titles.publID \
                    GROUP BY publishers.publID \
                    ORDER BY publName";
    if(mysql_query(conn, sql)) {
        fprintf(stderr, "%s\\n", mysql_error(conn));
        fprintf(stderr, "%s\\n", sql);
        return 1;
    }
    // process result
    result = mysql_store_result(conn);
    if(result==NULL) {
        if(mysql_error(conn))
            return 1;
    }
    while((row = mysql_fetch_row(result))) {
        printf("%s %d\\n", row[1], atoi(row[0]));
    }
    mysql_free_result(result);
}
```

```

if(mysql_error(conn))
    fprintf(stderr, "%s\n", mysql_error(conn));
else
    fprintf(stderr, "%s\n", "unknown error\n");
return 1;
}
printf("%i records found \n", (int)mysql_num_rows(result));

// loop over all records
while((row = mysql_fetch_row(result)) != NULL) {
    for(i=0; i < mysql_num_fields(result); i++) {
        if(row[i] == NULL)
            printf("[NULL]\t");
        else
            printf("%s\t", row[i]);
    }
    printf("\n");
}

// release memory, sever connection
mysql_free_result(result);
mysql_close(conn);
return 0;
}

```

如果执行了这个程序，应该看到如下所示的输出结果：

```

uranus:~/hellow-c $ ./hellow
13 records found
23 Addison-Wesley
4 Apress
1 Bonnier Pocket
1 Diogenes Verlag
1 dpunkt
1 Galileo
1 Hanser
2 Markt und Technik
2 New Riders
3 O'Reilly & Associates
2 Ordfront förlag AB
1 Sybex
1 Zsolnay

```

18.2.3 编译与链接

在 Linux/UNIX 系统上，上面给出的示例程序可以用以下命令来完成编译和链接：

```
$ gcc -o hellow -I/usr/include/mysql -lmysqlclient main.c
```

这条编译命令各组成部分的具体含义如表 18-1 所示。

表 18-1 编译命令的组成部分

组成部分	含 义
gcc	GNU C/C++ 编译器
-o hellow	作为编译/链接结果的可执行文件的名字
-I/usr/include/mysql	MySQL 头文件 (mysql.h 文件) 的存放位置
-lmysqlclient	把可执行文件与 MySQL 客户库链接起来
main.c	C 语言源代码文件

根据系统的具体配置, MySQL 头文件在其他系统上的存放位置可能会与这里给出的不一样(比如说,这个位置可能是/usr/local/include/mysql 目录)。

18.2.4 Makefile

用来编译和链接一个程序的命令往往比较长(尤其是在编译一个由多个文件构成的项目时),构造起来也相当麻烦,而程序往往需要几经修改才能最终完成。这里有一个好主意:按照 make 命令的语法把编译和链接某个程序所必须的指令写入一个名为 Makefile 的文件,以后只须简单地输入 make 就行了,make 命令可以正确地判断出那个程序的哪些个组成部分需要重新编译。

可以参考下面这个例子去创建自己的 Makefile 文件。请注意,make 语法要求 Makefile 文件里的语句行缩进必须使用制表符(键盘上 Tab 键)来设置,不允许使用空格:

```
# example file hellow/Makefile
CC = gcc
INCLUDES = -I/usr/include/mysql
LIBS = -lmysqlclient

all:hellow

main.o: main.c
    $(CC) -c $(INCLUDES) main.c

hellow: main.o
    $(CC) -o hellow main.o $(LIBS)

clean:
    rm -f hellow main.o
```

18.2.5 以静态方式绑定 MySQL API 函数

如果能在预定位置(在大部分 Linux 系统上,这个位置是/lib 或/usr/lib 目录)找到 libmysqlclient 函数库的共享版本(libmysqlclient.so 文件),在上面给出的 Makefile 文件就将以动态方式链接 libmysqlclient 函数库与被编译的 C 程序。

如果它没能找到那个库文件或者如果想把所有的 MySQL API 函数直接集成到由 make 命令生成的可执行程序里(即以静态而不是动态方式去链接 C 程序),需要明确地告诉 make 命令必须使用 libmysqlclient 函数库的静态版本 libmysqlclient.a 文件来进行编译和链接。这个文件通常存放在 /usr/lib/mysql 目录里。

此外,现在还必须增加一个-lz 选项,这个选项的作用是让 make 命令在进行链接时把 libz 函数库也包括进去。libmysqlclient 需要调用这个函数库里的函数(在必要时)去压缩 MySQL 客户与 MySQL 服务器之间交换的数据:

```
$ gcc main.c -I/usr/include/mysql -L/usr/lib/mysql -lmysqlclient \
-lz -o hellow
```

如果正在使用 make 命令,只需要修改 Makefile 文件里的 LIBS 变量:

```
LIBS = -L/usr/lib/mysql -lmysqlclient -lz
```

根据系统的具体配置情况和 C 程序使用了哪些 libmysqlclient 函数,除了 libz 函数库,可能还需要再增加一些其他的函数库。表 18-2 汇总了以静态方式链接 MySQL API 函数时可能需要用到的其他函数库:

表 18-2 以静态方式链接 MySQL API 函数时可能用到的其他函数库

函数库选项	含 义
-L/usr/lib/mysql	静态 MySQL 函数库的文件路径 (libmysqlclient.a 文件)
-lc	容纳着各种常用函数的基础函数库 (libc)
-lcrypt	容纳着各种加密函数的函数库 (libcrypt)
-lm	容纳着各种算术函数的函数库 (libm)
-lnsl	容纳着各种域名解析函数的函数库 (libs1)
-lnss_files 和 -lnss_dns	容纳着各种域名解析服务切换函数的函数库；详细信息可以用 man nsswitch.conf 命令查看
-lz	容纳着各种数据压缩函数的函数库 (libz)

以静态方式链接 MySQL API 函数的好处是 C 程序里用到的所有 libmysqlclient 函数都被直接编译到了最终的可执行文件里。如此编译出来的程序即使在没有安装 libmysqlclient 函数库的计算机上也能正常工作。

不过，获得这个好处的代价是可执行文件的静态链接版本要比它自己的动态链接版本大很多（具体相差多大要取决于 C 语言程序使用了多少 libmysqlclient 函数）。以 18.2.2 节里给出的示例程序为例，以动态方式链接出来的可执行文件长度只有 11KB，而以静态方式链接出来的可执行文件长度是 1MB 左右（这还只是仅以静态方式链接 MySQL API 函数的结果，那些来自其他函数库的函数都没有算上）。

如果想知道某个程序在运行时都需要访问哪些个动态函数库，可以使用 ldd 命令。以 18.2.2 节里给出的示例程序为例，在笔者的测试系统上，用 ldd 命令去检查它的静态链接版本的结果如下所示：

```
$ ldd hellow
linux-gate.so.1 => (0xfffffe000)
libz.so.1 => /lib/libz.so.1 (0x4002b000)
libc.so.6 => /lib/tls/libc.so.6 (0x4003c000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

下面是用 ldd 命令去检查那个程序的动态链接版本的结果。这份清单显然要长得多，它里面不仅出现了 libmysqlclient 函数库，还出现了 libmysqlclient 函数库进一步调用的其他一些函数库：

```
$ ldd hellow
(0xfffffe000)
libmysqlclient.so.14 => /usr/lib/libmysqlclient.so.14 (0x4002b000)
libc.so.6 => /lib/tls/libc.so.6 (0x4012d000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x40243000)
libs1.so.1 => /lib/libs1.so.1 (0x40275000)
libm.so.6 => /lib/tls/libm.so.6 (0x4028b000)
libz.so.1 => /lib/libz.so.1 (0x402ae000)
libnss_files.so.2 => /lib/libnss_files.so.2 (0x402bf000)
libnss_dns.so.2 => /lib/libnss_dns.so.2 (0x402c9000)
libresolv.so.2 => /lib/libresolv.so.2 (0x402d0000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

18.3 与 MySQL 服务器建立连接

要想在 C 语言程序里与 MySQL 服务器建立连接，必须先用 `mysql_init()` 函数创建一个 `MYSQL` 类型的数据结构。与 MySQL 服务器建立连接的工作由 `mysql_real_connect()` 函数具体完成，这个函数的第一个参数是一个指向刚才创建的那个 `MYSQL` 数据结构的指针，接下来是一些必要的连接参数，它们依次是：MySQL 服务器的主机名（或 "localhost"）或 IP 地址、用户名、密码、默认数据库的名字、

套接字的序号（0 表示使用默认的套接字）、套接字文件的名字（NULL 表示使用默认的套接字文件）以及用来为新连接设置各种属性的操作标志。

CLIENT_COMPRESS 标志控制着是否需要以压缩格式来传输数据；*CLIENT_MULTI_STATEMENTS* 标志控制着是否允许一次执行多条 SQL 命令（这些 SQL 命令必须用分号隔开）；*CLIENT_MULTI_RESULTS* 标志控制着是否能对多条 *SELECT* 命令的查询结果进行处理（如果 C 程序将调用一个 MySQL 存储过程，就必须给出这个标志）。在本书的第 23 章可以找到更多可以用在这里的操作标志。

```
MYSQL *conn;
unsigned long flags;
flags = CLIENT_MULTI_STATEMENTS | ...;
conn = mysql_init(NULL);
if(mysql_real_connect(conn, "hostname", "username", "password",
                      "databasename", 0, NULL, flags) == NULL) {
    fprintf(stderr, "error messages ...\n");
```

一旦不再需要这条连接了，必须用 *mysql_close* 关闭它：

```
mysql_close(conn);
```

18.3.1 处理 MySQL 配置文件 my.cnf

在与 MySQL 服务器建立连接的时候，如果想对配置文件 *my.cnf* 里的设置进行处理（比如说，为了使用一个非标准化的端口或套接字文件进行连接），就需要在 *mysql_init()* 和 *mysql_real_connect()* 函数之间增加一条对 *mysql_options()* 函数的调用语句。如下所示的代码将从 *my.cnf* 文件读入 [client] 段落里的各项设置，并把它们作为参数传递给随后的 *mysql_real_connect()* 函数，没有明确给出的参数将根据它们的具体情况以 *NULL* 或 0 作为默认值（用户名和密码参数也是如此）。

```
conn = mysql_init(NULL);
mysql_options(conn, MYSQL_READ_DEFAULT_GROUP, "");
mysql_real_connect(...);
```

在 [client] 段落所给出的各个选项的基础上，如果还想把更多的选项读入 *mysql_options()* 函数，把那些选项归纳在 *my.cnf* 文件中的另一个段落里并把那个段落的名字作为这个函数的最后一个参数即可：

```
mysql_options(conn, MYSQL_READ_DEFAULT_GROUP, "mygroup");
```

18.3.2 处理命令行选项

在使用标准化 MySQL 程序（如 *mysql*、*mysqladmin*、*mysqldump* 等）时，可以在命令行里以一种统一的格式（如 *--hname*、*--host=name* 等）发送连接参数。如果打算用 C 语言开发一个适用于各种 MySQL 数据库的数据库管理工具，就应该让那个程序像标准化的 MySQL 程序那样也能对来自命令行的连接参数进行处理。令人遗憾的是，MySQL 的 C API 没有为此专门提供一种像 *mysql_options()* 函数那样简便易用的手段，只能凭借一些技巧以手动方式去做这件事。下面是与此有关的一些步骤。

(1) 用 MySQL 函数 *load_default()*（MySQL 文档没有对这个函数进行说明）代替 *mysql_options()* 读出 *my.cnf* 文件里的设置并把它们复制到命令行参数（即 *main.c* 程序中的 *argc* 和 *argv* 参数）的有关字段里。这么做的好处是能够以一种统一的方式对来自 *my.cnf* 文件的设置和来自命令行的参数进行处理。在下面的代码里，声明了一个字符串数组 *groups[]* 来保存 *my.cnf* 文件里的某个选项组的名字，该选项组里的选项将被读出并复制到 *argv* 参数里去。

```
const char *groups[] = {"mygroup", "client", NULL};
load_defaults("my", groups, &argc, &argv);
```

(2) 接下来，创建一个循环，在这个循环里用标准函数 `getopt_long()`（来自头文件 `getopt.h`）对 `argv` 里的所有参数进行处理并对主机名、用户名等变量进行初始化。出于信息安全方面的考虑，如果密码是通过命令行参数传递来的，还应该用空格把它覆盖掉。如果密码是以交互方式输入的，可以用 `get_tty_password()` 函数来提取和处理它。

(3) 把处理好的参数值传递给 `mysql_real_connect()` 函数。

提示 Paul DuBois在他的MySQL一书中的“MySQL C API”章节给出了上述过程的详细步骤。那一章的内容也可以在该作者的个人网站上找到，最新的网址是 <http://kitebird.com/mysql-book>。

18.4 执行 SQL 命令

18.4.1 简单的 SQL 命令

与 MySQL 服务器连接成功之后，就可以执行 SQL 命令了。这需要把 SQL 命令当做一个字符串传递给 `mysql_query()` 函数。这种字符串一般只能包含一条 SQL 命令，而且不允许以分号结束。本章稍后的内容里将介绍如何一次执行多条 SQL 命令。

如果 SQL 命令被 MySQL 服务器接受，`mysql_query()` 函数将返回 0。（这个返回值只能用来判断 SQL 命令是否存在语法错误，它不能告诉我们 SQL 命令有没有修改数据或者 `SELECT` 命令有没有返回结果。）

在执行 `INSERT`、`UPDATE` 或 `DELETE` 命令之后，可以用 `mysql_affected_rows()` 函数来确定这些命令插入、修改或删除了多少条数据记录。这个返回值的数据类型是 `my_ulonglong`（这是由 MySQL C API 定义的一种数据类型，它相当于一个 64 位无符号整数。）

在插入一条新记录之后，可以用 `mysql_insert_id()` 函数来确定新记录的 `AUTO_INCREMENT` 编号值。这个函数的返回值也是一个 `my_ulonglong` 类型的整数：

```
mysql_query(conn, "INSERT INTO publishers (publName) \
    VALUES ('publisher')");
printf("publID = %i\n", (int)mysql_insert_id(conn));
```

处理 `SELECT` 查询结果

在调用 `mysql_query()` 函数执行了一条 `SELECT` 命令之后，可以用 `mysql_store_result()` 函数或 `mysql_use_result()` 函数来获取该命令的查询结果。这两个函数都将返回一个 `MYSQL_RES` 结构，可以用 `mysql_fetch_row()` 函数来逐条读取其中的结果记录。下面是这一流程的基本框架：

```
MYSQL_RES *result; // result of a SELECT query
MYSQL_ROW row; // a record of the result
...
result = mysql_store_result(conn);
if(result==NULL)
    fprintf(stderr, "%error ...\n");
else {
    while((row = mysql_fetch_row(result)) != NULL) {
        // process each record
    }
    mysql_free_result(result); // release result
}
```

在处理完所有的结果记录之后，应该尽早使用 `mysql_free_result()` 函数释放 `MYSQL_RES` 结构。这有助于把程序在客户端的内存占用量（用 `mysql_store_result()` 函数取回查询结果时）和在 MySQL 服务器上的资源占用量（用 `mysql_use_result()` 函数取回查询结果时）降到最低。

如表 18-3 所示，`mysql_store_result()` 和 `mysql_use_result()` 函数有着本质的区别。如果能预见到 `SELECT` 查询命令所返回的数据量比较小，`mysql_store_result()` 函数会是更好的选择：占用的服务器资源比较少，查询结果在客户端处理起来比较方便。只有在确切地知道 `SELECT` 查询命令所返回的数据量非常大（需要以 MB 来计算）的时候，才应该考虑使用 `mysql_use_result()` 函数。

表 18-3 `SELECT` 查询结果的内部管理

<code>mysql_store_result()</code> 函数	<code>mysql_use_result()</code> 函数
所有的结果记录将一次性地立刻传输到客户端。如果数据量很大，就会在客户端占用大量的内存	不传输任何结果记录，数据都保留在服务器上。服务器将负责结果记录的管理工作。在客户端依次取回所有的结果记录之前，服务器上的有关资源将一直处于被占用状态
可以按任意顺序对任何一条结果记录进行任意多次访问（用 <code>mysql_data_seek()</code> 函数）	结果记录只能依次读取，而且只能读取一次
可以立刻使用 <code>mysql_num_rows()</code> 函数来确定 <code>SELECT</code> 查询命令找到了多少条记录	只有在使用 <code>mysql_fetch_rows()</code> 函数依次取回所有的结果记录之后才能知道本次查询找到了多少条记录

用 `mysql_fetch_rows()` 函数把一条结果记录读入一个 `MYSQL_ROW` 结构之后，就可以通过 `row[n]` 形式的变量对它的各个字段进行存取和处理了。结果记录里的所有字段值（数值、日期/时间值、`TIMESTAMP` 值等全部包括在内）都是以零值字节（8 个二进制位全都是 0 的字节）结束的字符串。如果某个结果字段在数据库里的值是 `NULL`, `row[n]` 的值也将是 `NULL`。这种安排会给二进制数据的处理工作带来许多难题（详见本章后面的有关内容）。

在下面这个循环里，将依次读取所有的结果记录，并把它们的各个字段输出到一个控制台窗口里，相邻字段之间用制表符隔开：

```
int i;
while((row = mysql_fetch_row(result)) != NULL) {
    for(i=0; i < mysql_num_fields(result); i++) {
        if(row[i] == NULL)
            printf("[NULL]\t");
        else
            printf("%s\t", row[i]);
    }
    printf("\n");
}
```

如果正在开发一个适用于各种 MySQL 数据表的通用性程序，即如果无法在事先知道结果记录会有多少个字段（数据列）以及它们在 MySQL 数据库里的原始数据类型，还可以用 `mysql_num_fields()` 函数来确定结果数据列的个数、用 `mysql_fetch_fields()` 函数来获得关于各结果数据列的元信息（数据类型、最大字符和数字长度等）。对这些函数的详细介绍见本书的第 23 章。

18.4.2 一次执行多条 SQL 命令

一般来说，用 `mysql_query()` 函数每次只能执行一条 SQL 命令，而且除非 SQL 语法允许，SQL 命令里不允许出现分号（;）字符。这里要特别提醒的是，SQL 语法不允许 SQL 命令以分号结束。（如果非要这么做，MySQL C API 将抛出一个错误。）

不过，这项规定不包括 *CREATE PROCEDURE*、*CREATE FUNCTION* 和 *CREATE TRIGGER* 等用来定义一个 MySQL 存储过程或触发器的命令：分号在这些命令里充当着 SQL 指令之间的分隔符，是整个存储过程或触发器内部的一部分（不是结尾）。在这几种命令里使用分号不会让它们无法执行。

1. *MULTI_STATEMENTS* 模式

从 MySQL 4.1 版本开始，与之配套的 C API 提供了一些允许在 C 语言程序里一次执行多条 SQL 命令的办法。但在这么做之前，必须先启用 *MULTI_STATEMENTS* 模式，这一任务有两种办法可以完成：

- 在与 MySQL 服务器建立连接的时候，把 *CLIENT_MULTI_STATEMENTS* 属性作为最后一个参数传递给 *mysql_real_connect()* 函数：

```
mysql_real_connect(conn, "localhost", "username", "xxx",
    "dbname", 0, NULL, CLIENT_MULTI_STATEMENTS);
```

- 在与 MySQL 服务器建立起连接之后，用 *mysql_set_server_option()* 函数把 MySQL 服务器的当前工作状态明确地设置为 *MULTI_STATEMENTS* 模式。注意，这里必须使用 *MULTI_STATEMENTS* 常数的另一个名字：

```
mysql_set_server_option(conn, MYSQL_OPTION_MULTI_STATEMENTS_ON);
```

完成上述准备工作之后，就可以用 *mysql_query()* 函数一次执行多条 SQL 命令了。此时，SQL 命令之间必须用分号（;）字符隔开；如下所示：

```
mysql_query("command1; command2; command3");
```

状态信息。在 *MULTI_STATEMENTS* 模式下，如果在执行完 *INSERT*、*UPDATE* 或 *DELETE* 命令之后想知道这些命令插入、修改或删除了多少条数据记录，可以像往常一样使用 *mysql_affected_rows()* 函数来确定。第一次调用这个函数的返回值是被第一条命令修改了的记录个数。如果想知道第二条命令的执行情况，必须先发出一个 *mysql_next_result()* 函数调用（参见 18.4.3 节），依次类推。

出错处理。在 *MULTI_STATEMENTS* 模式下，如果第一条 SQL 命令在执行时发生错误，MySQL 将放弃执行后面的所有命令。此时，*mysql_query()* 函数将返回一个出错代码，*mysql_error()* 和 *mysql_errno()* 函数可以提供关于出错原因更详细的信息。

如果发生错误的不是第一条 SQL 命令，问题就比较复杂了：即使整个执行过程因一条非法的 SQL 命令半途而废，*mysql_query()* 函数也将返回 0 (OK) 作为返回值（因为第一条 SQL 命令执行成功了）。如果想知道有没有发生过错误以及哪一条 SQL 命令发生了错误，唯一的办法是利用 *mysql_next_result()* 函数（参见下一小节）去遍历所有的结果：在遇到那条出错的 SQL 命令时，*mysql_next_result()* 函数将返回一个出错代码，此时可以调用 *mysql_error()* 函数去获得关于那个错误的出错信息。

2. *MULTI_RESULTS* 模式（处理多组 *SELECT* 查询结果）

MySQL 允许同一个存储过程（stored procedure, SP）进行多次 *SELECT* 查询，而这意味着一条命令（*CALL spname*）完全有可能返回多组查询结果。为了能对多组查询结果进行处理，必须提前启用 *MULTI_RESULTS* 模式：把 *CLIENT_MULTI_RESULTS* 属性传递给 *mysql_real_connect()* 函数。顺便说一句，*MULTI_RESULTS* 模式会在上一小节介绍的 *MULTI_STATEMENTS* 模式下自动启用。

在 *MULTI_RESULTS* 模式下，可以在执行完一条 SQL 命令之后像往常一样立刻处理第一组 *SELECT* 结果（*mysql_store_result()* 等函数都可以使用）。如果想知道后面还有没有结果，可以用 *mysql_more_results()* 函数去进行检查。如果有，可以用 *mysql_next_result()* 函数前进到下一组结果，这个函数有 3 种返回值：

0 (OK, 后面还有结果)

-1 (OK, 后面没有结果)

>0 (出错代码)

在 *mysql_next_result()* 函数调用返回后，就可以用 *mysql_store_result()* 或 *mysql_use_result()* 处理下一组查询结果了。此时，*mysql_error()*、*mysql_warning_count()* 和 *mysql_affected_rows()* 等函数所提供的状态信息都是关于此时正在处理的那条 SQL 命令的。

3. 示例

下面这个例子演示了一次执行多条 SQL 命令并对它们的结果依次进行处理的全过程。为了增加代码的可读性，在构造那些 SQL 命令的时候利用反斜线字符 (\) 把它们写成了多个语句行：

```
// example file multi/main.c
#include <stdio.h>
#include <mysql.h>

int main(int argc, char *argv[])
{
    int      i, next;
    MYSQL   *conn;    // connection to the MySQL server
    MYSQL_RES *result; // manage SELECT results
    MYSQL_ROW row;    // process data record

    // create connection
    conn = mysql_init(NULL);
    mysql_options(conn, MYSQL_READ_DEFAULT_GROUP, "");
    if(mysql_real_connect(
        conn, "localhost", "root", "uranus",
        "mylibrary", 0, NULL, 0) == NULL) {
        fprintf(stderr, "sorry, no database connection ...\\n");
        return 1;
    }
    mysql_set_server_option(conn, MYSQL_OPTION_MULTI_STATEMENTS_ON);

    // execute SQL commands
    const char *sql="SELECT * FROM categories LIMIT 5\\
                    INSERT INTO categories (catName) \\
                    VALUES ('test1'), ('test2')\\
                    SELECT 1+2+dummy\\
                    DELETE FROM categories WHERE catName LIKE 'test%'\\
                    DROP TABLE IF EXISTS dummy";
    if(mysql_query(conn, sql)) {
        fprintf(stderr, "MySQL error: %s\\n", mysql_error(conn));
        fprintf(stderr, "MySQL error number: %i\\n", mysql_errno(conn));
    }
    do // loop over all results
    {
        printf("\\n-----\\n\\n");
        printf("Affected rows: %i\\n", mysql_affected_rows(conn));
        if(mysql_warning_count(conn))
            fprintf(stderr, "MySQL warnings: %i\\n", mysql_warning_count(conn));

        result= mysql_store_result(conn);
        if(result) {
            // display SELECT results
            while((row = mysql_fetch_row(result)) != NULL) {
                printf("result: ");
                for(i=0; i < mysql_num_fields(result); i++) {
                    if(row[i] == NULL)
                        printf("[NULL]\\t");
                    else
                        printf("%s\\t", row[i]));
                }
                printf("\\n");
            }
        }
    }
}
```

```

        printf("%s\t", row[i]);
    }
    printf("\n");
}
mysql_free_result(result);
} else
printf("no result\n");

// read next result
next = mysql_next_result(conn);
if(next>0) {
    printf("\n-----\n");
    printf("mysql_next_result error code: %i\n", next);
    if(mysql_errno(conn))
        fprintf(stderr, "MySQL error: %s\n", mysql_error(conn));
}
} while (!next);

// close connection
mysql_close(conn);
return 0;
}

```

当执行这个程序的时候，将看到如下所示的输出结果：

```

user:~/c/multi> ./multi
-----
Affected rows: -1
result: 1      Computer books          11      2004-12-02 18:37:20
result: 2      Databases                1       2004-12-02 18:37:20
result: 3      Programming              1       2004-12-02 18:37:20
result: 4      Relational Databases     2       2004-12-02 18:37:20
result: 5      Object-oriented databases  2       2004-12-02 18:37:20
-----
Affected rows: 2
no result
-----
Affected rows: -1
result: 3
-----
Affected rows: 2
no result
-----
Affected rows: 0
MySQL warnings: 1
no result

```

18.4.3 预处理语句

在许多数据库应用软件里，经常会出现需要使用不同的参数反复多次地执行同一条查询命令的现象。针对这一情况，MySQL 从 4.1 版本开始支持预处理语句（prepared statement）：预处理语句中的 SQL 命令只需向 MySQL 服务器传递一次，以后再执行这条 SQL 命令时只须把发生了变化的参数值传递过去即可。这种做法不仅能够减少数据传输量，还能够提高 SQL 命令的处理和执行效率，因为 MySQL 服务器现在只需要对经过预处理的 SQL 命令进行一次分析。

使用预处理语句的另一个好处是：数值、日期值、时间值等将不再作为字符串来传递，它们将以 C 语言里的缓存变量的形式来传递，而这些变量都有着与它们所传递的参数值互匹配的数据类型；比如说，MySQL 中的 *INT* 整数将以 C 语言中的 *int* 变量值的形式来传递。不过，为了使用预处理语句，必须对有关的数据结构进行一些相对比较复杂的初始化处理，而这些处理必须发生在执行第一条 SQL

命令之前。

1. 多次执行同一条SQL命令

在下面的例子里，将使用不同的参数值多次执行 *INSERT INTO titles (title, subtitle, langID) VALUE (?,?,?)* 命令。为了达到这一目的，先用 *mysql_stmt_init()* 函数创建了一个 *MYSQL_STMT* 数据结构，然后用 *mysql_stmt_prepare()* 函数对它进行了初始化：把 SQL 命令字符串传递给这个函数，该命令里的每一个参数分别用一个问号 (?) 来表示。

这里使用了一个 *MYSQL_BIND* 类型的数组来描述那条 *INSERT* 命令的 3 个参数（两个字符串，一个整数），这个数组里的元素在对它们进行初始化之前已全部清零 (*memset*)。在描述每一个参数的时候，至少需要为它指定一种数据类型和一个缓存变量。在执行 SQL 命令的时候，有关数据将从缓存变量里读出。可以用来为预处理语句传递参数的各种数据类型 (*buffer_type*) 可以在第 23 章里查到。对那些数组元素进行完初始化之后，还需要用 *mysql_bind_param()* 函数把它们绑定到我们的 *INSERT* 命令上去。

完成上述预处理工作之后，就可以利用 *mysql_stmt_execute()* 函数反复多次地执行那条 *INSERT* 命令了——当然，在此之前还应该把它的参数设置好。注意，如果传递给某个参数的值有可能是 *NULL*，就必须把与这个参数对应的 *bind[n].is_null* 所指向的那个变量设置为 1——这一点在下面给出的示例代码里有演示（详见第二次执行 *INSERT* 命令时的 *langID* 参数）。对于字符串参数值，必须给出它的实际字符个数——为了做到这一点，把 *title_len* 和 *subtitle_len* 变量绑定到了 *MYSQL_BIND* 结构的 *length*（意思是“长度”）元素上¹。

```
// example file prepare1/main.c
#include <stdio.h>
#include <mysql.h>

int main(int argc, char *argv[])
{
    MYSQL *conn;           // connection to MySQL server
    MYSQL_STMT *stmt;      // data structure for the prepared statement
    MYSQL_BIND bind[3];    // description of the parameters

    char *insert =
        "INSERT INTO titles (title, subtitle, langID) VALUES (?, ?, ?)";
    char title_buf[256];
    char subtitle_buf[256];
    unsigned long title_len, subtitle_len;
    int langID;
    my_bool langID_is_null;
    // connection to MySQL
    conn = mysql_init(NULL);
    mysql_options(conn, MYSQL_READ_DEFAULT_FILE, "");
    if(mysql_real_connect(
        conn, "localhost", "root", "uranus",
        "mylibrary", 0, NULL, 0) == NULL) {
        fprintf(stderr, "sorry, no database connection ...\\n");
        return 1;
    }

    // create statement structure
    stmt = mysql_stmt_init(conn);
    // initialize prepared statement
    // bind parameters
    bind[0].buffer_type = MYSQL_TYPE_STRING;
    bind[0].buffer = title_buf;
    bind[0].length = title_len;
    bind[0].is_null = langID_is_null;
    bind[1].buffer_type = MYSQL_TYPE_STRING;
    bind[1].buffer = subtitle_buf;
    bind[1].length = subtitle_len;
    bind[1].is_null = langID_is_null;
    bind[2].buffer_type = MYSQL_TYPE_LONG;
    bind[2].buffer = (char *) &langID;
    bind[2].length = sizeof(langID);
    bind[2].is_null = langID_is_null;
}
```

1. 原文里的 *strlen_title* 和 *strlen_subtitle* 变量在示例代码里根本不存在！——译者注

```

mysql_stmt_prepare(stmt, insert, strlen(insert));

// define parameters of the prepared statement
memset(bind, 0, sizeof(bind));
bind[0].buffer_type = FIELD_TYPE_STRING;
bind[0].buffer = title_buf;
bind[0].buffer_length = 256;
bind[0].length = &title_len;

bind[1].buffer_type = FIELD_TYPE_STRING;
bind[1].buffer = subtitle_buf;
bind[1].buffer_length = 256;
bind[1].length = &subtitle_len;

bind[2].buffer_type = FIELD_TYPE_LONG;
bind[2].buffer = (gptr) &langID;
bind[2].is_null = &langID_is_null;
mysql_stmt_bind_param(stmt, bind);

// execute command for the first time
strcpy(title_buf, "title1");
title_len = strlen(title_buf);
strcpy(subtitle_buf, "test prepared statements");
subtitle_len = strlen(subtitle_buf);
langID=1;
langID_is_null = 0;
mysql_stmt_execute(stmt);
printf("new title with titleId=%d has been inserted\n",
      (int) mysql_insert_id(conn));

// execute again
strcpy(title_buf, "title2");
title_len = strlen(title_buf);
strcpy(subtitle_buf, "test prepared statements");
subtitle_len = strlen(subtitle_buf);
langID_is_null = 1; // langID = NULL
mysql_stmt_execute(stmt);
printf("new title with titleId=%d has been inserted\n",
      (int) mysql_insert_id(conn));

// close statement and connection
mysql_stmt_close(stmt);
mysql_close(conn);
return 0;
}

```

2. 处理SELECT查询

如果打算将 *SELECT* 命令作为预处理语句来执行，就必须把 *SELECT* 查询结果里的各个数据列分别绑定到一个缓存变量上。这么一来，在读取结果记录的时候就可以获得一种与参数传递机制相类似的效果。与前面介绍的通过 *mysql_store_result()* 和 *mysql_fetch_row()* 函数去处理 *SELECT* 查询结果的做法相比，这种“参数传递”机制的好处是：从结果记录里提取出来的数值、日期值等将不再是字符串形式而是 *int* 变量、*MYSQL_TIME* 结构等，极大地简化了对这些数据的处理工作。

在下面的例子里，将从 *titles* 数据表读出一些数据（涉及 4 个数据列：*title*、*titleID*、*catID* 和 *ts*）。与 18.4.2 节里的例子一样，这次在结果数据列与 C 语言变量之间建立绑定关系的工作还是需要通过一个 *MYSQL_BIND* 类型的数组（示例代码中的 *bind[]* 数组）来进行，只不过这次是用 *mysql_stmt_bind_result()* 函数来完成处理的。请注意，对这个函数的调用只能发生在 *mysql_stmt_prepare()* 函数返回之后。

在调用 *mysql_stmt_execute()* 函数执行完 SQL 命令之后，用 *mysql_stmt_store_result()* 函数把所有的

结果记录一次性地取了回来。在这里调用这个函数并不是必要的，而且它还有其他的替代品(*mysql_stmt_use_result()*函数)。之所以在这里使用它是因为它可以简化对 *SELECT* 查询结果的处理工作：可以用 *mysql_stmt_num_rows()* 函数去统计结果记录的总数、可以用 *mysql_stmt_data_seek()* 函数在查询结果里前后移动等。(从 MySQL 的行为上看，调用 *mysql_stmt_store_result()* 函数与调用 *mysql_use_result()* 函数的效果差不多。应该尽量避免使用 *mysql_stmt_use_result()* 函数，它的效果类似于 *mysql_store_result()* 函数。参见表 18-2 中的对比。)

接下来，用 *mysql_stmt_fetch()* 函数把一条结果记录提取到了刚才通过 *MYSQL_BIND* 字段指定的有关变量里。如果没有更多的结果，函数返回 0。这个函数不必要求此前必须执行过 *mysql_stmt_store_result()* 函数调用，它可以根据具体情况从本地缓冲区或者从服务器读取结果记录。

因为刚才执行过 *mysql_stmt_store_result()* 函数，所以现在可以毫无问题地使用 *mysql_stmt_num_rows()* 函数去统计结果记录的总数(但在这个例子没有这样做)；如果想按任意顺序去读取结果记录，在执行 *mysql_stmt_fetch()* 函数之前先调用一次 *mysql_stmt_data_seek()* 函数即可(在这个例子也没有这样做)。关于查询结果的元数据可以用 *mysql_stmt_result_metadata()* 函数去获得。

在输出 *SELECT* 查询结果的时候，一定要检查将被输出的数据值是不是 *NULL*。如果数据值是 *NULL*，相应的 *bind[n].is_null* 所指向的变量将包含着 0。在下面的代码里，就是根据这一点去检查 *catID* 值的——在这个例子里的 *SELECT* 命令所返回的数据列当中，只有 *catID* 数据列是允许包含 *NULL* 值的。

在这个例子里还演示了如何对日期/时间值进行处理。*titles.ts* 数据列是一个 *TIMESTAMP* 数据列，其内容是 *titles* 数据表里的各条数据记录最近一次被修改的时间。在使用 *MYSQL_BIND* 结构把这个数据列绑定到一个 C 语言变量上去的时候，正确的数据类型是 *FIELD_TYPE_TIMESTAMP*。这样一来，就可以把 MySQL 中的 *TIMESTAMP* 值传递到一个 C 语言中的 *MYSQL_TIME* 变量里去，可以从这个 C 语言变量里轻而易举地把日期/时间值的各个组成部分提取出来。*(.year 对应着年份、.month 对应着月份等，请注意前缀的“.”字符！)*

```
// example file prepare2/main.c
#include <stdio.h>
#include <mysql.h>

int main(int argc, char *argv[])
{
    MYSQL      *conn;      // connection to MySQL server
    MYSQL_STMT *stmt;      // prepared statement
    char       *cmd =
        "SELECT title, titleID, catID, ts \
         FROM titles ORDER BY RAND() LIMIT 5";

    MYSQL_BIND   bind[4]; // result column
    char        title_buf[256];
    unsigned long title_len;
    int          titleID, catID;
    my_bool      catID_is_null;
    MYSQL_TIME   ts;

    int          err;      // for the return value of the mysql functions
    // create connection to MySQL
    .. see the previous example

    // for UTF-8 outputs
    mysql_query(conn, "SET NAMES 'utf8'");

    // initialize and prepare prepared statement
    stmt = mysql_stmt_init(conn);
```

```

mysql_stmt_prepare(stmt, cmd, strlen(cmd));

// declare result columns
memset(bind, 0, sizeof(bind));
bind[0].buffer_type = FIELD_TYPE_VAR_STRING; // title
bind[0].buffer = title_buf;
bind[0].buffer_length = 256;
bind[0].length = &title_len;

bind[1].buffer_type = FIELD_TYPE_LONG;      // titleID
bind[1].buffer = (gptr) &titleID;

bind[2].buffer_type = FIELD_TYPE_LONG;      // catID
bind[2].buffer = (gptr) &catID;
bind[2].is_null = &catID_is_null;

bind[3].buffer_type = FIELD_TYPE_TIMESTAMP; // ts
bind[3].buffer = (gptr) &ts;

// bind result columns to variables
mysql_stmt_bind_result(stmt, bind);

// execute command
err = mysql_stmt_execute(stmt);
if(err) {
    fprintf(stderr, "sorry, an error happened ...\\n");
    return 1;
}

// read all results of the MySQL command
mysql_stmt_store_result(stmt);

// loop through all result records
while(!mysql_stmt_fetch(stmt)) {

    printf("titleID=%d \\t", titleID);
    if(catID_is_null)
        printf("catID=NULL \\t");
    else
        printf("catID=%d \\t", catID);
    printf("timestamp=%d-%02d-%02d %02d-%02d-%02d\\t",
           ts.year, ts.month, ts.day,
           ts.hour, ts.minute, ts.second);
    printf("title=%s\\n", title_buf);
}

// end
mysql_stmt_close(stmt);
mysql_close(conn);
return 0;
}

```

18.4.4 字符集设置 (Unicode)

作为一种程序设计语言，C 语言对字符集的识别能力非常低级：凡是以零值字节（8 个二进制位全都是 0 的字节）结尾的一串数据，C 语言都会把它看做是一个字符串。至于个别字节还有什么含义，C 语言及用 C 语言编写出来的程序毫不关心，只知道在传递一串字节时必须保证它们的内容和顺序不能变化而已。

当在控制台窗口里执行一个用 C 语言编写的 MySQL 应用程序时，为了保证它在与 MySQL 服务器通信的时候能够对字符串做出正确的处理，应该让它使用控制台窗口的当前字符集。可是，在很多时候，MySQL 客户程序将沿用 MySQL 服务器的默认字符集，这个字符集往往是 *latin1*。如果需要改用另一种字符集，可以用 *SET NAMES* 命令来进行设置，如下所示：

```
mysql_query(conn, "SET NAMES 'utf8'");
```

当然，除了'utf8'，还可以改用 MySQL 服务器所能支持的任何一种其他的字符集，如'latin1'。如果想让编写出来的程序能够与各种各样的字符集相兼容，办法之一是在程序里通过编程安排一个选项来控制 *SET NAMES* 设置，办法之二是在程序里通过编程去改变 UNIX/Linux 环境变量 *LANG* 和 *LC_xxx*。

注意，*SET NAMES* 命令只对 MySQL 服务器与 C 语言程序之间的通信有影响，它对 MySQL 数据表里实际存储的数据所使用的字符集没有任何影响（MySQL 数据表里的字符集设置是在 *CREATE TABLE* 命令创建它们的时候确定的）。收到 *SET NAMES* 命令之后，MySQL 将自动地把所有的字符串从有关数据表的字符集转换为新设置的字符集。这里要特别提醒大家注意这样一个问题：如果字符串里有新字符集无法表示的字符，那些字符将被替换为一个问号；这意味着比较特殊的字符容易导致信息缺失和不完整。

18.5 处理二进制数据和特殊字符

有两种情况需要大家特别注意：其一是在打算执行的 SQL 命令里包含有零值字节或特殊字符的时候（这种情况在处理 BLOB 数据时经常会遇到）；其二是在 *SELECT* 查询命令所返回的结果里可能包含有零值字节或特殊字符的时候。

mysql_real_escape_string() 函数可以给容易引起问题的字符加上一个反斜线字符作为前缀（如'l、\b、\t、\n、\'等），但因为它采用的具体办法是把原始字符串复制为一个新字符串，所以在编程时必须提前保留足够的空间让这个函数可以完成它的功能，如下所示：

```
// char *s1          target string
// char *s2          initial string (may contain 0 bytes)
// unsigned long len2 the length of s2
mysql_real_escape_string(conn, s1, s2, len2);
```

在汇编 SQL 命令的时候，如果在它的各组成部分里有可能包含着特殊字符，就必须使用 *mysql_real_escape_string()* 函数来预防可能出现的问题。下面的例子演示了如何正确地拼装一条 *INSERT* 命令——拼装结果是 *INSERT INTO publishers (publName) VALUE ('O'Reilly')*。

这个例子还演示了辅助函数 *strmov()* 的用法。这个来自 MySQL C API 的辅助函数有着与 *strcpy()* 函数同样的功能，但它会返回一个指向被复制字符串末尾的指针。因此，*strmov()* 函数特别适合用来逐段地拼凑字符串，而拼凑字符串又几乎是每一个 C 语言程序都不得不去做的一件事。如果想在自己的程序里使用 *strmov()* 函数，必须先用 *include* 指令把 *my_global.h* 和 *m_string.h* 这两个头文件包括到程序里（并且，还必须按照下面这个示例所给出的顺序来进行）。

如果想参照下面这个示例编写一段类似的代码，千万记得要根据具体需要把字符串缓冲区 *tmp* 设置得足够大，还要记得把必要的合法性检查代码也添加上。

```
#include <my_global.h> // necessary so that strmov works
#include <m_string.h> // necessary so that strmov works
#include <mysql.h>
...
char tmp[1000], *tmppos;
char *publname = "O'Reilly";
tmppos = strmov(tmp, "INSERT INTO publishers (publName) VALUES ('");
tmppos += mysql_real_escape_string(
    conn, tmppos, publname, strlen(publname));
tmppos = strmov(tmppos, "')");
*tmppos++ = (char)0;
mysql_query(conn, tmp);
```

如果构造出来的 SQL 命令字符串里包含着零值字节，不要用 `mysql_query()` 函数去执行它！这种 SQL 命令必须用 `mysql_real_query()` 函数去执行。这两个函数之间的唯一区别是：`mysql_real_query()` 函数依据一个明确给出的字符个数（它的最后一个参数）而不是零值字节去判断字符串的长度。如下所示：

```
// sql points to the string with the SQL command
// n gives the length of the string
mysql_real_query(conn, sql, n);
```

二进制数据在被读取时也很容易引起问题，因为 C 语言里的大部分字符串函数都会把它们在读取一条结果记录 (`row[i]` 之类的数组元素) 时遇到的第一个零值字节认为是数据的结束。如果不能肯定数据里不包含零值字节，就必须调用 `mysql_fetch_lengths()` 函数去查明数据的实际长度。这个函数将返回一个 `unsigned long` 类型的数组，它的各个元素分别是当前结果记录各个字段的实际长度。下面的示例演示了这个函数的用法。

提示 还可以利用预处理语句来传输二进制数据。虽然这会给参数的初始化工作增加不少负担，但可以免去必须使用 `mysql_real_escape_string()` 函数去拼装 SQL 命令的麻烦。

预处理语句的另一个好处是可以使用 `mysql_stmt_send_long_data()` 函数，这个函数允许把一项 BLOB 数据分割成几段分次传输给服务器，而这有助于把客户端的内存占用量降到最低。可以在下面这个网址找到一个这个函数的用法示例：<http://dev.mysql.com/doc/mysql/en/mysql-stmt-send-long-data.html>。但令人遗憾的是，MySQL C API 没有提供任何可以用来从 MySQL 服务器分次读取一项 BLOB 数据的函数。

把二进制数据存入和读出数据库

在下面的例子里，将先从当前目录里把 `test.jpg` 文件的内容转存到 `exceptions` 数据库（参见第 8 章）的 `test_blob` 数据表里，再把那条新记录从数据表里读取出来转存到一个名为 `test-copy.jpg` 的新文件，最后删除那条新记录。（注意，为简明起见，这个示例程序在把数据传输到数据库的时候使用了一个长度为 512KB 的静态缓冲区，并且没有对 `test.jpg` 文件的长度是否小于这个缓冲区进行检查。如果是在一个真正的应用程序里，我们当然应该根据实际情况来动态地调整缓冲区的大小。）

```
// example file blob/main.c
#include <stdio.h>
#include <my_global.h> // for strmov
#include <m_string.h> // for strmov
#include <mysql.h>
int main(int argc, char *argv[])
{
    int id;
    FILE *f;
    MYSQL *conn; // connection to MySQL
    MYSQL_RES *result; // SELECT result
    MYSQL_ROW row; // a record of the SELECT result
    size_t fsize;
    char fbuffer[512 * 1024]; // file size maximum 512 kByte
    char tmp[1024 * 1024], *tmppos;
    unsigned long *lengths;
    // create connection
    conn = mysql_init(NULL);
    if(mysql_real_connect(
        conn, "localhost", "root", "uranus",
        "exceptions", 0, NULL, 0) == NULL) {
        fprintf(stderr, "sorry, no database connection ...\\n");
    }
    // insert file into table
    if(mysql_query(conn, "insert into test_blob values('test')") != 0) {
        fprintf(stderr, "error inserting file into table\\n");
    }
    // get file from table
    if(mysql_query(conn, "select * from test_blob") != 0) {
        fprintf(stderr, "error selecting file from table\\n");
    }
    result = mysql_store_result(conn);
    if(result == NULL) {
        fprintf(stderr, "error storing result\\n");
    }
    row = mysql_fetch_row(result);
    if(row == NULL) {
        fprintf(stderr, "error fetching row\\n");
    }
    fsize = mysql_fetch_lengths(result, lengths);
    if(fsize > 0) {
        tmppos = tmp;
        for(id = 0; id < fsize; id++) {
            tmppos += mysql_field_length(result, id);
        }
        if(strmov(tmp, row[0]) != tmp) {
            fprintf(stderr, "error moving data\\n");
        }
    }
    // write file
    f = fopen("test-copy.jpg", "w");
    if(f == NULL) {
        fprintf(stderr, "error opening file\\n");
    }
    if(fwrite(tmp, fsize, 1, f) != 1) {
        fprintf(stderr, "error writing file\\n");
    }
    if(fclose(f) != 0) {
        fprintf(stderr, "error closing file\\n");
    }
    // delete file
    if(mysql_query(conn, "delete from test_blob where id = 1") != 0) {
        fprintf(stderr, "error deleting file from table\\n");
    }
}
```

```

        return 1;
    }

    // read file test.jpg and store in exceptions.test_blob
    f = fopen("test.jpg", "r");
    fsize = fread(fbuffer, 1, sizeof(fbuffer), f);
    fclose(f);
    tmppos = strmov(tmp, "INSERT INTO test_blob (a_blob) VALUES ('')");
    tmppos += mysql_real_escape_string(
        conn, tmppos, fbuffer, fsize);
    tmppos = strmov(tmppos, ")");
    *tmppos++ = (char)0;
    mysql_query(conn, tmp);
    id = (int)mysql_insert_id(conn);
    // read the new record and store the data in the
    // new file test-copy.jpg
    f = fopen("test-copy.jpg", "w");
    sprintf(tmp, "SELECT a_blob FROM test_blob WHERE id = %i", id);
    mysql_query(conn, tmp);
    result = mysql_store_result(conn);
    row = mysql_fetch_row(result);
    lengths = mysql_fetch_lengths(result);
    fwrite(row[0], 1, lengths[0], f);
    fclose(f);

    // delete the new record
    sprintf(tmp, "DELETE FROM test_blob WHERE id = %i", id);
    mysql_query(conn, tmp);

    // release resources
    mysql_free_result(result);
    mysql_close(conn);
    return 0;
}

```

18.6 出错处理

如果在执行时发生错误，大多数 MySQL C API 函数会返回一个出错代码或返回 0。那些返回一个数据结构的函数（例如 `mysql_store_result()`、`mysql_fetch_row()` 等）采用了另一种做法：如果在执行时发生错误，则返回 `NULL` 作为返回值。

如果在执行时没有发生错误，MySQL C API 函数将返回 0 或空字符串（""）作为返回值。这使我们可以像下面这样来检查它们是否执行出错：

```

if(mysql_query(conn, sql)) {
    fprintf(stderr, "%s\n", mysql_error(conn));
    fprintf(stderr, "Fehlernummer %i\n", mysql_errno(conn));
    ...
}

```

有不少 SQL 命令也可以返回一条出错/警告消息。这些出错/警告消息的数量可以在 SQL 命令执行完毕后用 `mysql_warning_count()` 函数来获得。如果想看到出错/警告消息的内容，必须先执行一条 SQL 命令 `SHOW WARNINGS`，然后像对待一条 `SELECT` 命令那样对它的“查询”结果进行处理。

第 19 章**Visual Basic 6/VBA****19**

本章将介绍如何使用 Visual Basic 6 和 VBA（VBA 是 Microsoft Office 软件包里所有软件的一个组成部分）这两种程序设计语言编写 MySQL 应用程序，讨论重点是结合微软公司的数据库接口 ADO 和 MySQL 专用驱动程序 Connector/ODBC 编写程序。除了一些小示例，本章还将介绍一种可以把 Microsoft SQL Server 数据库转换为 MySQL 数据库的软件工具 mssql2mysql 程序。

在注重速度的应用程序里，应该考虑用效率更高的 VBMySQLDirect 来代替 ODBC 和 ADO。本章的末尾将对 VBMySQLDirect 库做一个简单的介绍。

19.1 基础知识和术语

ODBC（数据库开放互连标准）是一种流行于 Windows 环境下的机制，它是人们为了让应用程序能够以一种统一的方式去访问各种数据库系统而定义的一套编程接口。通过 ODBC 去访问 MySQL 数据库的唯一要求是系统必须安装有 Connector/ODBC，它是专为 MySQL 数据库系统而开发的 ODBC 驱动程序。（Connector/ODBC 的早期版本叫做 MyODBC。）

提示 作为学习本章内容的前提和基础，应该按照本书第 7 章给出的步骤把 Connector/ODBC 提前安装就绪。Connector/ODBC 的在线文档内容非常丰富，它目前的 URL 地址是 <http://dev.mysql.com/doc/mysql/en/odbc.html>。还可以在网址 <http://lists.mysql.com/myodbc> 处找到一个 Connector/ODBC 邮件表。

1. 确定 Connector/ODBC 的版本号

如果想检查一下计算机里是否已经安装了 Connector/ODBC，先通过菜单命令 Start(开始) | Settings(设置) | Control Panel(控制面板) | Administrative Tools(系统管理工具) | Data Sources(ODBC)(ODBC 数据源) 打开 ODBC 的对话框，然后看它的 Driver(驱动程序) 选项卡里有没有列出 Connector/ODBC 驱动程序。

本章内容以 Connector/ODBC 3.51.n 为基础（笔者使用的是 Connector/ODBC 3.51.11 版本）。注意，如果想连接 4.1 或更高版本的 MySQL 数据库系统，至少需要安装 Connector/ODBC 3.41 版本。这个 ODBC 驱动程序的早期版本无法支持 MySQL 4.1/5.0 的新密码机制。

Connector/ODBC 3.52 版本即将发布的消息早就传得沸沸扬扬，但截止到 2005 年 3 月，我们连它的 alpha 测试版本都还没有见着。

2. 与微软数据库产品有关的几个术语

如果以前没有接触过微软公司的数据库产品，猜不出含义的缩写词就肯定不止 ODBC 一个。下面

是几个与微软数据库产品有关的常见术语。

- ODBC (Open Database Connectivity, **数据库开放互连标准**)。ODBC 是一种流行于 Windows 环境下的机制，它是人们为了让应用程序能够以一种统一的方式去访问各种数据库系统而定义的一套编程接口。ODBC 从诞生至今已经有许多年头了，但因为有无数的新、老数据库系统和软件开发库的支持，所以它作为一种与开发者无关的标准用来与基于 Windows 的各种数据库系统进行通信和编程而存在的生命期还有很长。这个方案只须为数据库系统安装一个所谓的 ODBC 驱动程序就可以实现，ODBC 驱动程序的作用是在 ODBC 系统和数据库之间创建一个接口。MySQL 数据库系统的 ODBC 驱动程序叫做 Connector/ODBC，它可以从 <http://dev.mysql.com> 网站免费下载。
- DSN (Data Source Name, **数据源名字**)。有许多带有 ODBC 接口的程序只是把 DSN 当做一种创建数据库连接的简便方法，但包括 Access 和 Excel 在内的另外一些程序却是只有通过 DSN 才能建立一条 ODBC 连接。如果打算让 Access 或 Excel 去创建一条与 MySQL 服务器的 ODBC 连接，就必须先通过 ODBC 管理器窗口定义一个数据源（参见第 7 章）。
如果通过一种程序设计语言去访问 MySQL 数据库，那么在不使用 DSN 的情况下也可以与数据库建立起连接。将在本章后面介绍 ADO 编程技巧的时候演示这两种做法（使用 DSN、不使用 DSN）。
- OLE-DB。OLE-DB 是作为 ODBC 的后继者推出的，只可惜它在数据库世界里始终没能真正赢得这样的地位。在微软公司推出了它们的新数据库接口 (.NET Framework 中的 ADO.NET) 之后，OLE-DB 失去了更多的领地，反而是 ODBC 仍被人们认为是一项标准。
OLE-DB 是作为 ADO（见下一条目）的一个辅助接口而被开发出来的。就 MySQL 而言，目前还没有成熟的 OLE-DB 驱动程序。不过，因为 OLE-DB 有一个 ODBC 接口，所以可以通过这个接口去访问那些没有自己的 OLE-DB 驱动程序、但支持 ODBC 的数据库。
- ADO (ActiveX Data Object, **ActiveX 数据对象**)。ADO 是使用 Visual Basic、VBA、Delphi 等程序设计语言编写数据库应用程序时需要用到的各种对象的一个集合。ADO 是那几种程序设计语言与 OLE-DB 之间的接口。对程序员而言，通过 ADO 对象去访问数据库的做法要比使用 OLE-DB 函数直接访问数据库的做法简便得多。在本章后面的内容里，我们将向大家演示如何使用 Visual Basic 语言通过 ADO 对象去访问 MySQL 数据库。
- ADO.NET。ADO.NET 是微软公司新推出的.NET Framework 中的数据库接口部分。（.NET Framework 是一个体积庞大的类库，但只有那些.NET 兼容语言，如 Visual Basic .NET 和 C# 等，才能使用这个家伙。）ADO.NET 与 ADO 在名字上非常相似，但它们在本质上并没有什么共同点；它们俩完全不兼容。
如果打算使用 ADO.NET 来开发 MySQL 应用程序的话，可以选用 Connector/ODBC 作为它们之间的桥梁——ADO.NET 与 ODBC 是兼容的。不过，这项工作用 MySQL 的 ADO.NET 驱动程序来完成会更有效率。这类驱动程序有好几种，第 20 章将介绍的 Connector.NET 就是其中之一；Connector.NET 已经得到了 MySQL 正式认可和支持。
- MDAC (Microsoft Data Access Components, **微软数据访问组件**)。MDAC 是对各种微软数据库组件和函数库（ADO 和 OLE-DB 也包括在内）的一个统称。微软旗下的 Windows、Office、Internet Explorer 等软件产品会把各种版本的 MDAC 随它们一起安装到用户的系统上。MDAC 的最新版本可以从下面这个网址免费下载：<http://msdn.microsoft.com/data/downloads/updates/default.aspx>。

- DAO、RDO。DAO 和 RDO 是与 ADO 功能类似的数据库编程接口。它们是 ADO 的前身，在推出 ADO 之后，微软公司对这两种编程接口的支持已不像以前那么尽心尽力。我们也不打算在这本书里对 DAO 和 RDO 做更多的讨论。

19.2 Connector/ODBC 选项

MySQL 服务器与客户端之间的 ODBC 连接可以通过一系列选项来加以调控。这些选项既可以通过如图 19-1 所示的 ODBC-DSN 对话框来设置，也可以在 ADO 程序代码里通过它们的编号来设置。接下来，我们将按照它们在 DSN 对话框里的顺序对一些比较重要的 Connector/ODBC 选项进行介绍，括号里的数字是各有关选项在程序代码里的对应编号值。

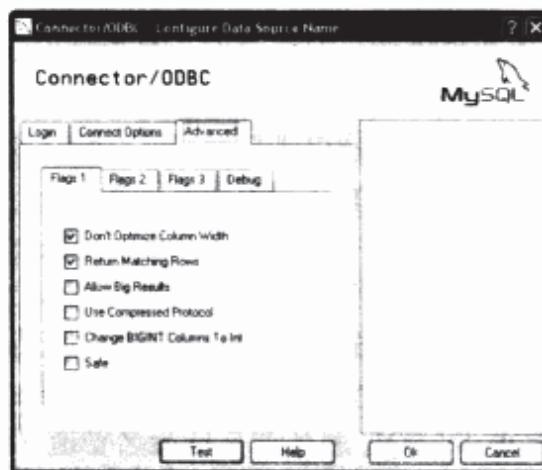


图 19-1 在 DSN 对话框里设置 Connector/ODBC 选项

- Don't Optimize Column Width(1): 不对数据列的宽度进行优化。在默认的情况下，如果 `SELECT col FROM table` 命令返回的字符串都不超过 n 个字符，但 n 小于这个数据列的最大字符串长度，Connector/ODBC 将返回 n 作为最佳数据列宽度。选中本复选框可以避免这种情况。
- Return Matching Rows (2): 返回匹配的数据行。MySQL 服务器可以为许多 SQL 命令（比如 `UPDATE`）返回受它们影响的数据记录（“受影响的记录”）个数，但有不少 ODBC 客户程序不具备正确解释这项信息的能力。如果选中了这个复选框，Connector/ODBC 将返回“被找到的记录”个数而不是“受影响的记录”个数（`UPDATE` 命令的“被找到的记录”个数是 0）。为了让 Connector/ODBC 与某些特定的程序（其中包括 Microsoft Access、带 ADO 组件的 Visual Basic/VBA 等）能够配合工作，就必须选中这个复选框。
- Allow Big Results (8): 允许大结果。如果选中了这个复选框，在 ODBC 程序与 MySQL 服务器之间传输的数据包的尺寸将不受限制。如果执行的 `SELECT` 命令会返回大量的数据或者如果需要对一些大尺寸的 BLOB 数据进行处理，就必须选中这个复选框。
- Use Compressed Protocol (2048): 使用压缩协议。如果选中了这个复选框，在 ODBC 程序与 MySQL 服务器之间传输的数据将使用压缩格式。这有助于减少网络的通信流量，但会加重客户端和服务器的 CPU 负担，因为它们必须对所有的数据进行压缩和解压缩。
- Change BIGINT Column to INT (16384): 把 `BIGINT` 数据列转换为 `INT`。微软公司的函数库大部分都不能处理 64 位整数。这个复选框的作用是把 MySQL 数据表中的 `BIGINT` 字段先自动转换为 32 位整数再进行传输。尤其是在使用 Visual Basic/VBA 语言进行 ADO 编程的时候，这个复选框必须选中。警告：如果选中了这个复选框，64 位整数的前 32 位将全部丢失！

- Safe (131072)**: 安全。这个复选框的作用是让 Connector/ODBC 多进行一些检查。如果 ODBC 程序在处理来自 MySQL 数据库的数据时遇到了问题，就应该把这个复选框选中。（有关文档没有对这个复选框做更准确的描述。）
- Don't Prompt Upon Connect (16)**: 连接时不提示。这个复选框的作用是：如果缺少必要的信息，在与 MySQL 服务器建立连接时将不显示 DSN 对话框。（请注意：即使已经选中了这个复选框，如果在连接某个数据库时没有给出用户名和/或密码或者如果给出的用户名和/或密码是错误的，这个对话框仍将会出现。）
- Enable Dynamic Cursor (32)**: 启用动态光标。如果选中了这个复选框，Connector/ODBC 将支持 Dynamic（意思是“动态”）类型的光标。这些光标是由 ODBC 驱动程序而不是由 MySQL 服务器提供的。出于追求效率的考虑，动态光标在默认的情况下是被禁用的。（这里所说的“光标”是用来管理 *SELECT* 查询结果的，它相当于一个读写指针。详细情况可以在 ADO 文档中介绍 CursorLocation 和 CursorType 属性的内容里查到。）
- Don't Cache Results (1048576)**: 不对结果进行缓存。在默认的情况下，同一条 *SELECT* 命令的全部结果将作为一个数据块从服务器传输到客户端。选中这个复选框将阻止这一做法，但通常只有在 *SELECT* 查询结果的数据量非常大的时候才有必要选中这个复选框。这个复选框仅在 ADO 程序使用的是只允许向前遍历各条结果记录（即所谓的“单向前进”）的光标时才起作用。
- Force Use of Named Pipes (8192)**: 强制使用命名管道。如果选中了这个复选框，客户与服务器之间的通信将使用命名管道而不是 TCP/IP 协议来进行。这个复选框仅适用于 Windows NT/2000/XP 系统，并且要求 MySQL 服务器支持命名管道（但支持命名管道不是 MySQL 服务器的默认行为）。
- Force Use of Forward Only Cursors (2097152)**: 强制使用单向前进光标。如果选中了这个复选框，ODBC 驱动程序将总是返回一个只允许向前遍历各条结果记录（即所谓的“单向前进”）的光标（即使 ADO 请求使用另一种光标的时候也是如此）。
- Trace Driver Calls To myodbc.log (4)**: 把 ODBC 函数的调用情况记入 myodbc.log 日志文件。如果选中了这个复选框，在访问数据库的过程中曾经被调用过的所有 ODBC 函数都将被记载到日志文件 C:\myodbc.log 里去。这个复选框对代码调试工作非常有帮助。
- Save Queries To myodbc.log (524288)**: 把查询命令记入 myodbc.log 日志文件。如果选中了这个复选框，在访问数据库的过程中曾经被执行过的所有 SQL 命令都将被记载到日志文件 C:\myodbc.log 里去。

在许多场合里，只须选中 *Don't Optimize Column Width* 和 *Return Matching Rows* 这两个复选框就已经足够了。

提示 Connector/ODBC 选项的完整清单以及更详细的说明可以在下面这个网址查到：<http://dev.mysql.com/doc/mysql/en/connection-parameters.html>。

19.3 ADO 程序设计与 Visual Basic 6/VBA 简介

本节将讨论如何使用 Visual Basic 6 或 VBA 这两种程序设计语言去编写用来访问 MySQL 数据库的程序代码的问题。将使用 ADO 作为数据库编程接口。对数据库的访问由 Connector/ODBC 负责具

体完成，ADO 与 Connector/ODBC 之间的通信将经过 OLE-DB 和 OLE-DB/ODBC 驱动程序两道环节。完整的通信链路是这样的：VB/VBA→ADO→OLE-DB→OLE-DB/ODBC 驱动程序→Connector/ODBC→MySQL。

如果 MySQL 服务器与 Visual Basic 程序没有¹运行在同一台计算机上，这条通信链路的前 5 个环节将发生在客户端，而通过网络最先与 MySQL 服务器进行通信的将是 Connector/ODBC。

提示 在这本书里，假设用户都对 ADO 有足够的了解。如果对 ADO 一窍不通，既不知道 Connection 对象和 Recordset 对象是什么东西，也不知道绑定控件的用途和用法，在继续阅读本章之前最好先找一本关于 ADO 程序设计的书好好读读。本书没有足够的篇幅去介绍与 ADO 有关的基本知识。

本章中的示例代码已在 Connector/ODBC 3.51.11 加 Visual Basic 6 加 AOD 2.8 的测试环境下通过了所有的测试。这些代码在 VBA 6 加 ADO 老版本（如：ADO 2.1）的环境下也应该可以正常运行，但涉及 ADO 控件的部分代码不在此列。

当然，完全可以使用 Visual Basic 程序的脚本型变体——比如 Active Server Pages (ASP) 脚本或 Windows Scripting Host (WSH) 程序——来完成同样的工作。如果真是那样，需要在声明有关变量时省略 *As typename* 子句并使用像 *Set conn = CreateObject("ADODB.Connection")* 这样的指令来创建 ADO 对象。

1. 前提与限制

- **MySQL 数据表：**如果打算通过 ADO 提供的 *Recordset* 对象或者是绑定控件 (*Insert*、*Update*、*Delete*) 去修改 MySQL 数据表里的数据，就必须给那些 MySQL 数据表配上一个主索引和一个 *TIMESTAMP* 数据列。
- **Connector/ODBC 复选框：**在为 MySQL 数据库配置 DSN 的时候，*Don't Optimize Column Width*、*Return Matching Rows* 和 *Change BIGINT Column to INT* 这 3 个复选框必须选中。如果程序是通过编程而不是通过 DSN 去连接 MySQL 数据库的，则必须在代码里使用与上述 3 个复选框等价的设置，即 *Options = 16387 (1 + 2 + 16384)*。如果还想在自己的 ADO 程序里使用一个动态的服务器端光标 (*CursorType = adOpenDynamic*, *CursorLocation = adUseServer*)，正确的 *Options* 设置值将是 *16419 (1 + 2 + 32 + 16384)*。
- **用 Recordset 对象和绑定控件来修改数据：**如果以上条件都能得到满足，就可以使用 *Recordset* 对象和绑定控件来修改数据。

可是，这么的效果往往并不好。在一些比较简单的的场合，把修改过的数据再存入数据库（不管是通过一个 *Recordsets* 对象、还是通过一个绑定控件）一般不会有什问题；但在比较复杂的应用项目里，这种做法往往会导致一些非常难以追查的问题。这些问题的根源并不总是 MySQL 或 Connector/ODBC 有什么地方不妥，OLE-DB 和/或 ADO 也经常出毛病。许多有经验的 ADO 程序员干脆放弃使用绑定控件，即使在以微软出品的数据库系统作为数据源的时候也不使用它们。

正是因为这个理由，才会在 MySQL 新闻组上看到那么多人建议尽量避免使用 *Recordset* 对象和绑定控件去修改数据，它们只适合用来读取和显示数据。如果真的需要对数据进行修改，应该先构造一条正宗的 SQL 命令 (*UPDATE* 等)，再用 *conn.Execute* 指令去执行它。这两种办法（第一种是通过 *Recordset* 对象去编辑有关数据的简便办法，第二种是使用 SQL 命令的传统套路）在本章的示例中都

1. 原书这里出现严重错误！少了“没有”二字！——译者注

有演示，但最值得推荐的还是第二种变体。

□ 速度：如果按照本节开头给出的通信链接路来实现从 Visual Basic 到 MySQL 的访问，因为 Visual Basic 与 MySQL 之间隔着好几层接口，Visual Basic 在处理字符串的时候速度又比较慢，这种解决方案的速度和效率当然谈不上理想。如果想获得更高的性能，可以考虑使用 VBMySQLDirect 接口，它可以在 Visual Basic 和 MySQL 之间建立起一条直达连接，不需要通过 ODBC 等其他环节。

2. 示例程序

与本书的其他章节一样，可以在 www.apress.com 网站找到大量与本章内容有关的示例程序。那些示例程序都需要系统上安装有 Visual Basic 6 才能顺利运行。此外，大部分示例还需要把 *mylibrary* 数据库安装到系统上。

如果有兴趣试用一下那些示例程序，千万不要忘记在它们的开头部分（通常是在 *Form_Load()* 函数里）把有关的连接参数（主机名、用户名、密码等）修改成符合自己的系统设置情况的样子，那些示例程序现在使用的连接参数是：主机名是 *localhost*，用户名是 *root*，密码是 *uranus*。

19.4 与 MySQL 服务器建立连接

19.4.1 与 MySQL 服务器建立连接：使用 DSN

在下面这段代码里，将先使用一个 DSN 去连接 MySQL 服务器，然后执行一条简单的 *SELECT* 命令。由 *SELECT* 命令返回的结果记录可以通过 *rec* 对象的属性和方法去处理：

```
Dim conn As Connection
Dim rec As Recordset
Set conn = New Connection
conn.ConnectionString = "DSN=mysql-mylibrary"
conn.Open
Set rec = New Recordset
rec.CursorLocation=adUseClient
rec.Open "SELECT * FROM tablename", conn
...
    'process record list
rec.Close 'close record list
conn.Close 'close connection
```

下面解释一下用来设置 *ConnectionString* 属性的那条语句：在最简单的情况下，只需给出一个 DSN 名字即可，用户名和密码都是可选的；各个参数之间必须用分号 (;) 隔开。一个完整的 *ConnectionString* 字符串如下所示：

```
conn.ConnectionString = "DSN=mysql-mylibrary;UID=root;PWD=xxx"
```

ConnectionString 字符串里的 *UID* 和 *PWD* 设置要优先于当初定义这个 DSN 时给出的原始设置。

表 19-1 对使用 DSN 与 MySQL 服务器建立连接时需要用到的 *ConnectionString* 参数做出了解释。

表 19-1 *ConnectionString* 参数（使用 DSN 连接数据库服务器）

参 数	含 义
<i>DSN=name</i>	数据源的名字（DSN 是 Data Source Name 的字头缩写）
<i>UID=name</i>	用来建立连接的 MySQL 用户名
<i>PWD=password</i>	用来建立连接的 MySQL 密码

19.4.2 与 MySQL 服务器建立连接（不使用 DSN）

利用下面演示的 Visual Basic 编程技巧，即使没有提前定义一个 DSN 也可以与 MySQL 服务器建

建立起连接。如果想开发一个适用于任何 MySQL 数据库的通用性程序，这个技巧将非常实用。具体地说，这次需要像下面这样来构造 *ConnectionString* 字符串：

```
conn.ConnectionString = "Provider=MSDASQL;" +_
"DRIVER={MySQL ODBC 3.51 Driver};"+_
"Server=localhost;UID=username;PWD=xxx;" +_
"database=databasename;Option=16387"
```

表 19-2 对不使用 DSN 与 MySQL 服务器建立连接时需要用到的 *ConnectionString* 参数做出了解释。

表 19-2 *ConnectionString* 参数（不使用 DSN 连接数据库服务器）

参数	含义
Provider=MSDASQL	OLE-DB/ODBC 驱动程序的名字（Microsoft Data Access SQL）
Driver=MySQL ODBC 3.51 Driver	MySQL/ODBC 驱动程序的名字
Server=name	MySQL 服务器的主机名或 IP 地址
Port=n	MySQL 服务器的 IP 端口号（默认值是 3306）
UID=name	用来建立连接的 MySQL 用户名
PWD=password	用来建立连接的 MySQL 密码
Database=name	数据库的名字
Prompt=noprompt / complete	在连接失败时是否显示 Connector/ODBC 对话框（否 是）
Options=n	Connector/ODBC 选项

Prompt=complete 的作用是让程序在连接 MySQL 服务器失败时自动显示如本书第 7 章图 7-2 所示的 ODBC 连接对话框，用户可以通过这个对话框重新输入一个密码等。注意，这个对话框里的 Windows DSN Name 字段必须有内容（可以通过编程随便提供一个字符串），否则这个对话框将无法退出。

因为这个对话框很容易把用户弄糊涂（它里面的许多选项与其说是方便，不如说是陷阱），所以最好不要使用 *Prompt=complete* 选项（默认设置是 *Prompt=noprompt*），而是采用下面这种做法。

首先，在开始连接数据库服务器之前先执行一条 *On Error Resume Next* 指令。其次，在尝试连接数据库服务器之后对是否发生错误进行检查。如果发生错误，则通过编程显示一个定制的对话框供用户进一步输入或修改有关的连接参数并重新构造一个新的 *ConnectionString* 字符串。

Options 参数用来给出各种 Connector/ODBC 连接选项。这个参数的设置值是各有关 Connector/ODBC 连接选项的编号累加在一起得到的一个整数值。具体到 Visual Basic/ADO 程序，通常必须把 *Options* 参数的值设置为 16387 (=1+2+16384)，这相当于同时选中 *Don't Optimize Column Width*、*Return Matching Rows* 和 *Change BIGINT Column to INT* 这三个选项。

1. 通过 *DataEnvironment* 对象与 MySQL 服务器建立连接

如果使用的是 Visual Basic Professional 6（个人版）或 Visual Basic Enterprise 6（企业版），还可以通过 *DataEnvironment* 对象来设置各种连接属性，其具体做法是：在 *Connection* 对象的属性对话框里，先在 Provider（厂商）选项卡里选中 Microsoft OLE-DB 作为提供 ODBC Drivers（ODBC 驱动程序）的厂商，然后切换到 Connection（连接）选项卡（如图 19-2 所示），选择一个现有的 DSN 或是单击 Build（创建）按钮表示想用必要的连接属性自行构造一个新的连接字符串。单击 Build 按钮将打开一个类似于 ODBC 对话框的对话框，但在那个对话框里的设置结果将作为一个字符串被复制到 *DataEnvironment* 对话框里去，可以在 *DataEnvironment* 对话框里对那个字符串进行必要的编辑。

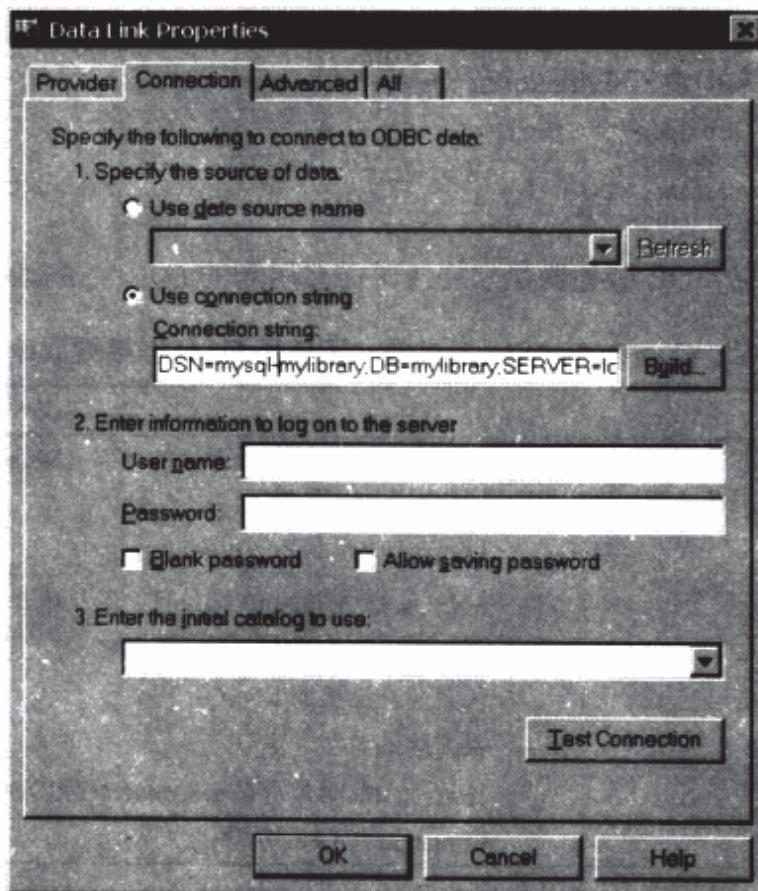


图 19-2 在 *DataEnvironment* 对话框里设置连接参数

以上步骤将为 *Connection* 对象创建一个 *DataEnvironment* 对象，而可以在接下来的程序代码里像下面这样通过后者去访问前者。(在下面这段代码里，*DataEnvironment* 对象的名字是 *DE*，*Connection* 对象的名字是 *conn*。)

```
Dim rec As Recordset
Set rec = New Recordset
If DE.Conn.State = adStateClosed Then
    DE.Conn.Open
End If
rec.CursorLocation=adUseClient
rec.Open "SELECT * FROM database", DE.Conn
...
```

DataEnvironment 对话框不仅可以帮助设置好各种连接参数，还可以帮助构造各种 SQL 查询命令——这个对话框里的 SQL Generator (SQL 命令生成器) 是一个非常方便的工具，用它生成的 SQL 查询命令可以在程序代码里通过相应的 *Command* 对象去访问。

2. *Connection* 对象的属性

Connection 对象（以及它们代表的数据库连接）的属性有很多，刚才介绍的 *ConnectionString* 参数只能对其中的一小部分做出设置——对于那些未经明确设置的大部分属性，ADO/ODBC 将简单地使用它们的默认设置值。

有时候，知道这些选项是如何被设置的会很有用。刚才介绍的通过 *ConnectionString* 参数去设置一些连接选项的情况就是一个很直观的例子——可以通过在程序代码里修改 *ConnectionString* 参数中的连接选项而灵活地与数据库服务器建立连接。事实上，甚至可以通过一个名为 *Extended Properties* 的超级参数去为 *Connection* 对象设置更多的选项：

```

conn.ConnectionString = "Provider=MSDASQL;Driver=MySQL;Server=..."
conn.Open
Debug.Print conn.ConnectionString
Provider=MSDASQL.1;Extended Properties=
"DRIVER={MySQL ODBC 3.51 Driver};DESC=;DATABASE=mylibraryodbc;
 SERVER=localhost;UID=root;PASSWORD=uranus;PORT=;
OPTION=16387;STMT=;"

```

Connection 对象的 *Properties* 属性可以提供更多的信息，而且这些信息更容易提取：

```

Debug.Print conn.Properties("Max Columns in Index")
32

```

Connection 对象的属性及其设置值的完整清单可以通过以下循环获得：

```

Dim p As Property
For Each p In conn.Properties
    Debug.Print p.Name & " = " & p.Value
Next

```

19.4.3 ADO 程序设计技巧

1. 带客户端光标的Recordset对象

Recordset 对象是对一个现有的 *Connection* 对象（参见 19.4.2 节）调用 *open* 方法而打开（创建）的。在本章的所有示例里，*Connection* 对象的变量名都是 *conn*。

在默认的情况下，ADO 将自动提供一个服务器端光标作为默认的光标位置。如果想改用一个客户端光标，就必须在执行 *Open* 指令之前明确地做出相应的设置，如下所示：

```

Dim rec As Recordset
rec.CursorLocation = adUseClient 'client-side cursor
rec.Open "SELECT ... FROM ... WHERE ...",
        conn, adOpenStatic, adLockReadOnly

```

Open 方法的参数主要有以下几个：第一个参数是打算执行的 SQL 代码字符串；第二个参数是代表着 *Connection* 对象的变量；第三个用来设置打算使用的光标的 *CursorType* 类型（因为客户端光标只支持 *adOpenStatic* 类型。如果已经用 *rec.CursorLocation=adUserClient* 表明打算使用一个客户端光标，在这第三个参数的位置上随便写出什么都没有关系）；第四个参数用来设置 *Recordset* 对象里的数据是只读的 (*adLockReadOnly*) 还是允许修改的 (*adLockOptimistic*)。ADO 还提供了 *adLockPessimistic* 和 *adLockBatchOptimistic* 两种锁定方式，但笔者没有对它们进行测试。

如果使用了客户端光标，*Open* 方法将把 SQL 查询命令从数据库里查找出来的所有数据一次性地全部取回到客户端程序。如果数据量比较大，这可能需要花费相当多的时间。但这么做的好处是在程序里对结果数据进行处理的时候，可以随意使用 *Recordset* 对象的所有属性和方法，如前后移动 (*MoveNext* 等属性)、搜索 (*Find* 属性)、在本地进行排序 (*Sort* 属性)、修改 (*column= "new value":rec.Update*) 等。

2. 带服务器端光标的Recordset对象

可以用 *rec.CursorLocation=adUserServer*¹ 明确地表明打算使用一个服务器端光标。但因为服务器端光标是 ADO 默认提供的光标，所以只要没有使用 *rec.CursorLocation=adUserClient* 明确地表明想使用一个客户端光标，实际得到的就将是一个服务器端光标。属性 *CursorType= adOpenForwardOnly* 的含义是只能使用 *MoveNext* 方法依次遍历由 SQL 命令返回的结果记录并——如果没有明确地给出

1. 原书这里显然写错了。*CursorType* 与 *CursorLocation* 是两回事！另外，它还把 *adOpenForwardOnly* 错写成了 *adForwardOnly*——既然写了 *ad*，就必须写成 *adOpenForwardOnly*。——译者注

LockType=adLockReadOnly 的话——对结果记录进行修改。如下所示：

```
Dim rec As Recordset
rec.CursorLocation = adUseServer
rec.Open "SELECT ... FROM ... WHERE ...",
conn, adOpenForwardOnly, adLockOptimistic
```

注意，属性 *CursorType=adOpenForwardOnly* 的含义是只能使用 *MoveNext* 方法以单向前进的方式去遍历由 SQL 命令返回的结果记录。既不能使用 *Bookmarks* 方法，也不能在遍历完所有的结果记录之前统计出它们的总数来（如果真的需要这个数字，只能多执行一条 *SELECT COUNT(*) FROM ...* 查询命令）。

这些限制当然会给结果记录的处理工作带来不便，但这并不意味着服务器端光标没有实用价值。在需要对大量数据依次进行处理的场合，服务器端光标将是更好的选择——这可以避免客户端光标的缺点（所有的结果记录将一次性地被传输到客户端）。

3. Recordset 对象的属性

ADO 为 *Recordset* 对象提供的属性多得让人数不过来，但 *Recordset* 对象并不总是包含着在 *Open* 指令里设置或请求的属性，这是因为如果底层的驱动程序不支持某种特定的光标类型，ADO 就会自做主张地选用另外一种。

如果想知道某个 *Recordset* 对象都实际包含着哪些属性，可以使用 *CursorsTypes.vbp* 程序去进行测试（如图 19-3 所示）：可以通过鼠标对 *Recordset* 对象各有关属性进行设置，这个程序会打开 *Recordset* 对象去确定它实际包含着的属性并把它们显示出来。（请注意，对于服务器端光标，这个程序的检查结果并不总是准确无误的，尤其是图 19-3 中 *Recordset Supports(...)* 部分的信息。这部分信息由 *Recordset* 对象的 *Supports()* 方法返回，但这个方法有时会返回完全错误的结果。）

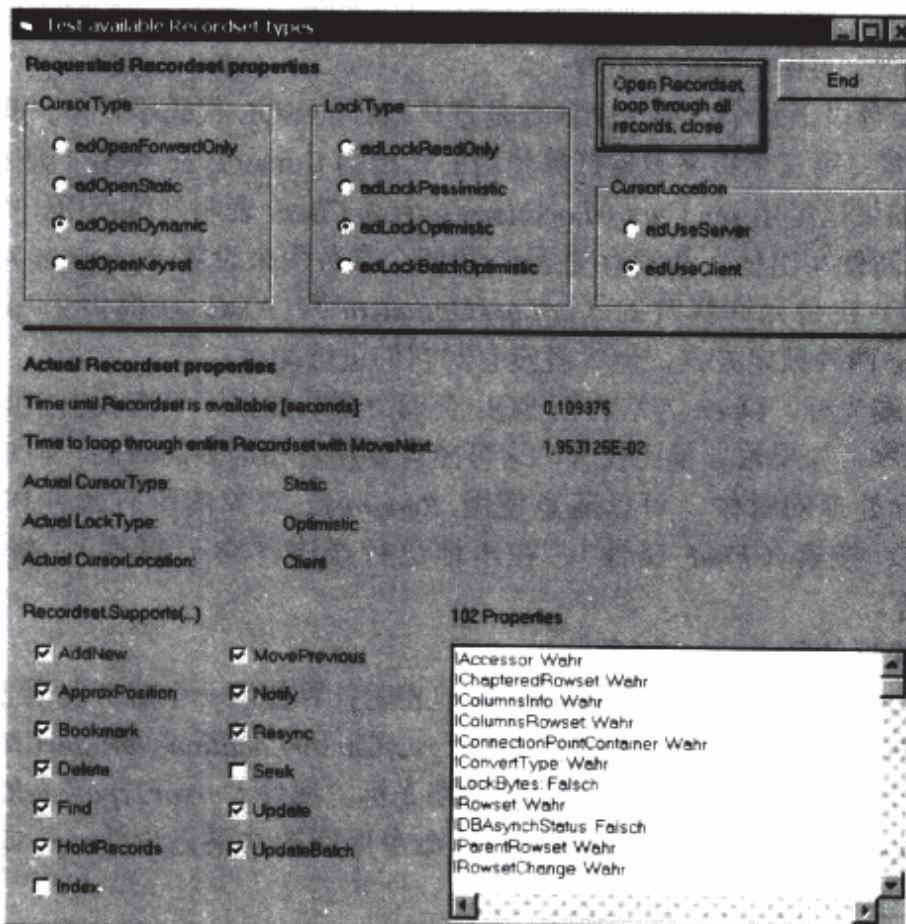


图 19-3 用来测试 Recordset 对象可用属性的 Visual Basic 程序

总的来说，“ADO + Connector/ODBC”支持以下几种光标属性组合：

CursorLocation = Client

CursorType = Static

*LockType = ReadOnly 或 Optimistic 或 BatchOptimistic 或 Pessimistic*¹

CursorLocation = Server

CursorType = ForwardOnly / Static / Dynamic

LockType = 以上所有属性

如果 *CursorLocation=Server*, ODBC 选项 *Enable Dynamic Cursor* (32) 将决定获得的是一个动态光标 (*CursorType=Dynamic*) 还是一个静态光标 (*CursorType=Static*)。

4. 打开一个变量型 Recordset 对象

如果没有对光标的属性进行设置，将默认地获得一个常数型 Recordset 对象 (*CursorLocation = Server*、*CursorType = ForwardOnly*、*LockType = ReadOnly*)，常数型 Recordset 对象里的数据像常数一样不允许修改。换句话说，如果需要像对待变量那样对 Recordset 对象里的数据进行修改的话，就必须明确地对光标属性做出必要的设置。用来打开一个变量型 Recordset 对象的典型代码如下所示：

```
Dim rec As New Recordset
rec.CursorLocation = adUseClient
rec.Open "SELECT cola, colb FROM table", conn, _
adOpenStatic, adLockOptimistic
```

5. Recordset 对象里的数据：NULL 和日期/时间

下面是各种 MySQL 数据类型在 ADO Recordset 对象里的表示方法。

□ **NULL 值**：可以用一个 *IsNull(rec!column)* 调用来判断 Recordset 对象 *rec* 里的 *column* 字段（数据列）的值是不是 NULL。

□ **日期/时间**：MySQL 数据库里的 DATE、TIME、DATETIME 和 TIMESTAMP 数据列将被自动转换为 Visual Basic 语言中的 Date 数据类型。这里要特别注意 MySQL 中的 TIME 值：这些值将自动获得当前日期作为前缀。（比如说，MySQL 中的 TIME 值“9:00”将变成“2005/3/17 9:00”。）

□ **BLOB**：MySQL 中的 BLOB 数据将被传递到 Visual Basic 中的 Byte 数组里，这个数组里的个别字节可以通过 *rec!a_blob(n)* 形式去访问，其中 *n*=0 对应着第一个字节。

Visual Basic 通常会把 Byte 数组解释为一个字符串。这就需要大家注意这样一个问题：因为 Visual Basic 内部使用的是 Unicode 字符集，所以长度为 512 字节的 BLOB 数据将对应于一个包含 256 个 Unicode 字符的 Visual Basic 字符串。如果想对这个字符串里的个别字节进行处理，就必须使用 AscB、ChrB、LenB、MidB、LeftB、RightB 和 InStrB 等 Visual Basic 函数。

□ **decimal**：MySQL 数据库里的 DECIMAL 数据列将被自动转换为 Visual Basic 语言中的 Decimal 数据类型。注意，不要把它与 Currency 数据类型弄混了。在 Visual Basic 语言里，Decimal 数据类型通常被看做是 Variant 数据类型的一个子类型。

□ **SET/ENUM**：MySQL 数据库里的 SET 和 ENUM 数据列将被自动转换为相应的字符串。

□ **字符串**：Visual Basic 能够自动而且正确地处理好 MySQL 字符串里的的特殊字符（反斜线“\”、单引号 “‘”、双引号 “” 和零值字节）。

6. 字符串

在 Visual Basic 的内部，字符串都使用 Unicode 来表示。Connector/ODBC 和 MySQL 服务器能够

1. 原书这里似乎少写了 *Pessimistic*，刚才还在正文里提到过！——译者注

把进、出某给定数据表的字符串自动转换为正确的字符集。

7. 把BLOB数据存入和读出一个变量（AppendChunk和GetChunk方法）

如果想存储二进制数据，就必须在 MySQL 数据表里使用 *BLOB* 数据列。可以使用 *AppendChunk* 方法把包含在变量 *bin* 里的数据存入 MySQL 数据表里的 *BLOB* 数据列，如下所示（注意，只有在使用了客户端光标的情况下才能这么做）：

```
Dim rec As New Recordset
rec.CursorLocation = adUseClient
rec.Open "SELECT * FROM test LIMIT 1", conn, adOpenStatic,
adLockPessimistic
rec.AddNew
rec!blobcolumn.AppendChunk bin
rec.Update
```

可以使用 *GetChunk* 方法来读出 *BLOB* 数据列里的数据，此时必须使用 *Actualsize* 属性来确定实际读出的数据到底有多少个字节；如下所示：

```
bin = rec!blobcolumn.GetChunk(rec!blobcolumn.ActualSize)
```

也可以把数据分成几段来传输（通过反复执行 *AppendChunk* 和 *GetChunk* 方法），这在数据量很大的时候很有用。

8. 把BLOB数据存入和读出一个文件

ADO 接口提供的 *Stream* 类可以让用户方便、简明地把 *BLOB* 数据直接存入和读出一个文件。在下面的示例代码里，将使用 *LoadFromFile* 方法把一个文件打开为一个二进制数据流，并把它的内容存入 *rec!pic* 字段（*rec* 是对应于某条 *SELECT* 查询命令的 *Recordset* 对象，*pic* 是这个对象里的某个 *BLOB* 数据列。假设我们刚才已经执行过那条 *SELECT* 命令了）：

```
Dim st As New Stream
st.Type = adTypeBinary
st.Open
st.LoadFromFile "C:\test1.gif"
rec!pic = st.Read
```

从 MySQL 数据表读出 *BLOB* 字段值并存入一个文件的工作可以用 *SaveToFile* 方法来完成，如下所示：

```
Dim st As New Stream
st.Type = adTypeBinary
st.Open
st.Write rec!pic
st.SaveToFile "C:\test2.gif", adSaveCreateOverWrite
```

这两个方法完整的用法示例可以在本章后面的有关内容里找到。

9. 确定新数据记录的AUTO_INCREMENT编号值

在把一条新记录插入数据表之后，经常需要知道这条新记录的 ID 编号（也就是作为数据表主索引的那个 *AUTO_INCREMENT* 数据列最新获得的编号值），但令人遗憾的是刚才用来插入新记录的 *Recordset* 对象提供不出这项信息。在 Visual Basic 程序里解决这个问题的办法是通过 SQL 命令 *SELECT LAST_INSERT_ID()* 确定那个 ID 编号。为了给以后的编程工作提供一些方便，建议大家在自己的程序里提前定义一个如下所示的函数去负责这项工作：

```
Private Function LastInsertedID() As Long
Dim rec As New Recordset
rec.CursorLocation = adUseClient
rec.Open "SELECT LAST_INSERT_ID()", conn
LastInsertedID = rec.Fields(0)
End Function
```

10. 与数据库有关的Visual Basic绑定控件

Visual Basic 提供了几个可以直接绑定到数据库查询命令（更准确地说，是相应的 *Recordset* 对象）的控件。这些控件可以把 *Recordset* 对象里的全部数据内容（或是只有当前结果记录的内容）显示在自己的窗口画面里。这些控件大大简化了显示数据库查询结果的编程工作，有几个控件甚至还允许用户直接对画面里的数据进行修改。在理想的情况下，程序员只需编写很少的代码就可以实现一个相当不错的数据库用户操作界面（如图 19-4 所示）。

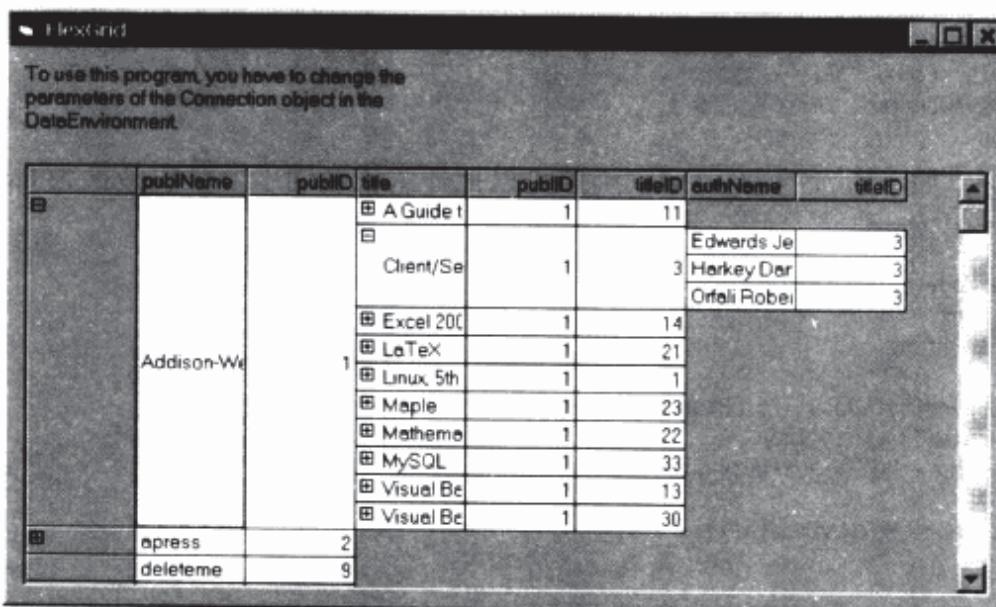


图 19-4 Visual Basic 示例程序：利用 MSHFlexGrid 控件显示 mylibrary 数据库里的数据

可是，绑定控件的实际使用效果往往不那么理想。一般来说，在进行 Connector/ODBC 编程的时候，如果使用了客户端光标，就最好不要使用绑定控件。还有，通过绑定控件去修改数据也不总是那么可靠，如果不是特别必要并经过全面的测试，最好不要使用这种功能。

如果不考虑这些缺点和不足，使用绑定控件还是有好处的：它们既可以使用微软出品的数据库系统作为数据源，也可以使用 MySQL 等其他厂商出品的数据库系统作为数据源。对绑定控件的进一步讨论在任何一本探讨 Visual Basic 数据库编程技术的书刊里都可以找到。

11. 直接执行SQL命令

ADO 接口提供的 *Recordset* 对象大大简化了对数据库查询结果的处理工作，但在某些场合，直接执行 SQL 命令也很有必要。在 Visual Basic 程序里，可以使用 *Connection* 对象的 *Execute* 方法来直接执行 SQL 命令：

```
conn.Execute "INSERT INTO table (a, b) VALUES ('x', 'y')"
```

与使用其他程序设计语言时的情况一样，程序员在使用 Visual Basic 语言编写 MySQL 应用程序时也会遇到必须按照 MySQL 语法规则对各种各样的数据进行转换的问题。在接下来的几个小节里，将介绍下面几个可以帮助完成这种数据转换工作的辅助函数：

□ 浮点数。在使用浮点数的时候必须注意这样一个细节：Visual Basic 通常根据操作系统的国家/语言设置来决定自己使用哪种格式来表示浮点数。比如，如果操作系统的国家/语言设置是 Germany（德国/德语），Visual Basic 就会按照德语习惯把圆周率 Pi 写成 3,1415926（逗号）而不是 3.1415926（小数点）。如果想让自己编写出来的程序在世界各地都能正常工作，就应该把程序里的浮点数用 Visual Basic 函数 *Str()* 全部转换为字符串。

□ **日期。**如下所示的 *Format* 指令可以把存放在变量 *x* 里的日期数据转换为符合 MySQL 要求的格式：

```
Format(x, "'yyyy-mm-dd Hh:Nn:Ss'")
```

□ **字符串。**如下所示的 *Quote()*函数可以给字符串里的反斜线 (\)、单引号 (') 和双引号 (") 字符加上一个反斜线作为前缀，它还可以把字符串里的零值字节替换为 “\0”：

```
Function Quote$(tmp$)
    tmp = Replace(tmp, "\", "\\")
    tmp = Replace(tmp, """", "\""")
    tmp = Replace(tmp, "'", "\'")
    Quote = Replace(tmp, Chr(0), "\0")
End Function
```

□ **二进制数据。**如下所示的 *Hexcode()*函数可以把任意长度的字节数组转换为 0x0102031232 格式的十六进制字符串：

```
Function HexCode(bytedata() As Byte) As String
    Dim i As Long
    Dim tmp As String
    tmp = ""
    For i = LBound(bytedata) To UBound(bytedata)
        If bytedata(i) <= 15 Then
            tmp = tmp + "0" + Hex(bytedata(i))
        Else
            tmp = tmp + Hex(bytedata(i))
        End If
    Next
    HexCode = "0x" + tmp
End Function
```

这个函数的用法如下所示：

```
conn.Execute "INSERT INTO table (col) VALUES (" + HexCode(...) + ")"
```

如下所示的 *HexcodeStr()*函数与 *Hexcode()*函数功能相同，但这次是把输入数据当作字符串来对待：

```
Function HexCodeStr(bytedata As String) As String
    Dim i As Long, b As Long
    Dim tmp As String
    tmp = ""
    For i = 1 To LenB(bytedata)
        b = AscB(MidB(bytedata, i, 1))
        If b <= 15 Then
            tmp = tmp + "0" + Hex(b)
        Else
            tmp = tmp + Hex(b)
        End If
    Next
    HexCodeStr = "0x" + tmp
End Function
```

19.4.4 示例：给 *titles* 数据表增加一个 *authors* 数据列

从 *mylibrary* 数据库生成一份全体图书及其作者的清单需要构造一个复杂的 *SELECT* 命令：两个 *JOIN*（意思是“关联”）操作、一次 *GROUP BY* 分组操作、一次 SQL 统计函数 *GROUP_CONCAT()* 调用：

```
USE mylibrary
SELECT title,
       GROUP_CONCAT(authname ORDER BY authname SEPARATOR ' ; ') AS authors
FROM authors, titles, rel_title_author
WHERE authors.authID = rel_title_author.authID
```

```

AND titles.titleID = rel_title_author.titleID
GROUP BY titles.titleID ORDER BY title
title           authors
A Guide to the SQL Standard      Darween Hugh; Date Chris
A Programmer's Introduction to PHP 4.0 Gilmore W.J.
Alltid den där Annette          Pohl Peter
...

```

如果经常进行这一查询，就应该考虑给 *titles* 数据表增加一个 *authors* 数据列来存放图书作者的姓名。当然，这个数据列里的数据将是冗余的，但这么做的好处是可以简化和加快对数据库的访问。

```
ALTER TABLE titles ADD authors VARCHAR(255)
```

不妨假设 *titles* 数据表已经有了 *authors* 数据列，而用户的任务是编写一个 Visual Basic 程序（它的用户界面如图 19-5 所示）把作者姓名数据填写到 *authors* 数据列里去。这里的思路是：遍历 *titles* 数据表里的所有记录，查出每一本书的全部作者，把各位作者的姓名合并为一个字符串并存入 *authors* 数据列。这个程序提供了两种办法来实现这一任务，下面就来分析一下它们都是如何做到这一点的。

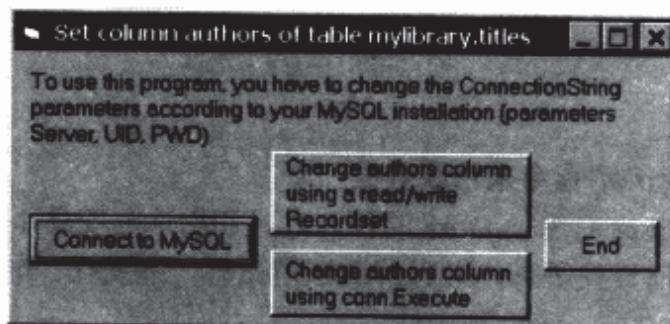


图 19-5 Visual Basic 示例程序：给 *titles* 数据表增加一个 *authors* 数据列

1. 变体1：通过读/写Recordset对象来修改数据

在这个变体里，有关代码全部包含在事件过程 *Command2a_Click* 里，这个过程里的全局变量 *conn* 代表着一条已经创建好了的数据库连接（*Connection* 对象）。

这段代码将打开两个 *Recordset* 对象：一个是 *titles*，它直接对应着与它同名的 *titles* 数据表，我们将通过这个 *Recordset* 对象来插入新数据（*LockType = adLockOptimistic*）；另一个是 *authors*，它包含着一份全体图书及其作者的清单，这个 *Recordset* 对象是在从数据库读入有关数据之后在本地排序的（*Sort* 属性），这么做可以减少对服务器资源的占用。

接下来的循环将遍历 *Recordset* 对象 *titles* 里的所有 *titleID* 值。在每一次循环里，将通过 *Recordset* 对象 *authors* 的 *Find* 属性查找出与当前 *titleID* 值相关的所有作者、把他们的姓名合并到字符串 *authors_str* 里、然后把这个字符串存入数据库：

```

' Example vb6\authors_for_titles\form1.frm
Private Sub Command2a_Click()
    Dim authors_str As String
    Dim titles As New Recordset
    Dim authors As New Recordset
    ' titles Recordset: read/write
    titles.CursorLocation = adUseClient
    titles.Open "SELECT titleID, authors FROM titles", _
               conn, adOpenStatic, adLockOptimistic
    ' authors Recordset: readonly, disconnected
    authors.CursorLocation = adUseClient
    authors.Open "SELECT titleID, authname "
                "FROM authors, rel_title_author " &
                "WHERE authors.authID=rel_title_author.authID",

```

```

        conn, adOpenStatic, adLockReadOnly
authors.Sort = "titleID, authname"
Set authors.ActiveConnection = Nothing

' loop over all titles
While Not titles.EOF
    authors_str = ""
    ' loop over all authors of this title
    authors.MoveFirst
    authors.Find "titleID=" & titles!titleID
    While Not authors.EOF
        If authors_str <> "" Then authors_str = authors_str + "; "
        authors_str = authors_str & authors!authName
        authors.MoveNext
    authors.Find "titleID=" & titles!titleID
    Wend
    ' save author list
    If authors_str <> "" Then
        titles!authors = authors_str
        titles.Update
    End If
    titles.MoveNext
    Wend
End Sub

```

2. 变体2：用conn.Execute()指令来修改数据

在第2种变体里，*Recordset*对象*authors*的打开方式和用途与变体1里的情况完全一样。随后的循环结构在根据当前*titleID*值查找所有作者时的做法稍微有一些变化。作者的姓名将临时存放在变量*authors_str*里。

与变体1相比的新东西是*conn.Execute*指令，这条指令将把*authors_str*变量里的作者姓名用一条`UPDATE`命令存入*titles*数据表。辅助函数*Quote*在这里必不可少，即使某位作者的姓名里包含着像“”（单引号）这样的特殊字符，这个函数也可以保证我们最终构造出来的SQL命令是正确的。

```

Private Sub Command2b_Click()
    Dim authors_str, titleID
    Dim authors As New Recordset
    ' authors Recordset: client-side, readonly, disconnected
    authors.Cursorlocation = adUseClient
    authors.Open "SELECT titleID, authname " &
        "FROM authors, rel_title_author " &
        "WHERE authors.authID=rel_title_author.authID " &
        "ORDER BY titleID, authName",
        conn, adOpenStatic, adLockReadOnly
    authors.Sort = "titleID, authname"
    Set authors.ActiveConnection = Nothing

    ' loop over all titles (titleID)
    While Not authors.EOF
        titleID = authors!titleID
        authors_str = ""
        ' search for all authors associated with current titleID value
        authors_str = ""
        Do While Not authors.EOF
            ' jump to the next titleID from the inner loop
            If authors!titleID <> titleID Then
                Exit Do
            End If
            ' add author name to authors_str
            If Not IsNull(authors!authName) Then
                If authors_str <> "" Then authors_str = authors_str + "; "
                authors_str = authors_str & authors!authName
            End If
        Loop
        If authors_str <> "" Then
            Conn.Execute "UPDATE titles SET authors=" & quote(authors_str) & " WHERE titleID=" & titleID
        End If
    Wend
End Sub

```

```

End If
authors.MoveNext
Loop

' save author list
If authors_str <> "" Then
    conn.Execute "UPDATE titles " &
        "SET authors = '" & Quote(authors_str) & "' " &
        "WHERE titleID=" & titleID
End If
Wend
End Sub
' place \ before ' " and \, replace Chr(0) by \0
Private Function Quote(tmp)
tmp = Replace(tmp, "\", "\\")
tmp = Replace(tmp, """", """")
tmp = Replace(tmp, "'", "\'")
Quote = Replace(tmp, Chr(0), "\0")
End Function

```

19.4.5 示例：添加一本新图书

可以用下面给出的示例程序把关于一本新图书的信息存入 *mylibrary* 数据库。这些信息包括：新图书的名字、它的一位或多作者、它的出版公司（如图 19-6 所示）。这个程序没有检查新图书的作者或出版公司是否已经存在于 *mylibrary* 数据库，因为提供这个示例程序的目的只是为了演示怎样把数据插入几个彼此关联的数据表。

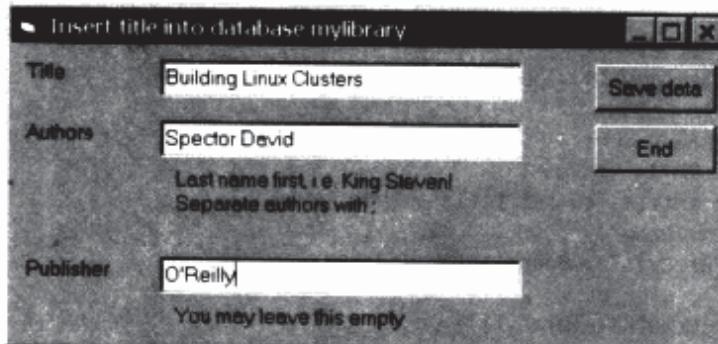


图 19-6 Visual Basic 示例程序：插入一条新图书记录

1. 变体1：通过读/写Recordset对象来修改数据

在下面这段示例代码的开头，一口气打开了 4 个 *Recordset* 对象，将在它们的帮助下把数据存入各有关数据表。在这里使用了一个小技巧：在打开 4 个 *Recordset* 对象的时候都使用了 *LIMIT 1* 子句。如果没有这个 *LIMIT 1* 子句，*SELECT* 命令就会把所有的匹配记录都返回给 *Recordset* 对象，但我们对这些东西并不感兴趣（因为这次的目的是为了修改数据，不是进行查询），想要的只是 4 个分别指向 *authors*、*titles*、*publishers* 和 *rel_title_author* 这 4 个数据表的 ADO 对象而已。可是，为了获得这 4 个 ADO 对象，我们又不得不去执行 4 条 *SELECT* 命令。于是，为了提高效率，我们在这里利用 *LIMIT 1* 子句构造出了 4 个最节省时间的 *SELECT* 命令。（如果想进一步提高这段示例代码的效率，可以把这 4 个 *Recordset* 对象定义为全局变量并在主函数 *Form_Load* 里打开它们。这么一来，就不必在每一个存储事件里再浪费时间去打开这 4 个 *Recordset* 对象了。）

把来自 *txtPublisher*、*txtTitle* 和 *txtAuthor3* 3 个文本输入字段（见图 19-6）的数据存入数据库的命令很容易理解。这里值得一提的是辅助函数 *LastInsertID()*，在稍早曾介绍过它的用途是确定 MySQL 服务器为给定数据表最新生成的 *AUTO_INCREMENT* 编号值。

```

' example vb6\insert_new_title\form1.frm
Private Sub SaveData_WithRecordsets()
    Dim i&, titleID&, authID&, publID&
    Dim authors_array
    Dim authors As New Recordset, titles As New Recordset
    Dim publishers As New Recordset, rel_title_author As New Recordset
    ' open Recordsets
    authors.CursorLocation = adUseClient
    titles.CursorLocation = adUseClient
    publishers.CursorLocation = adUseClient
    rel_title_author.CursorLocation = adUseClient
    authors.Open "SELECT * FROM authors LIMIT 1", _
        conn, adOpenStatic, adLockOptimistic
    titles.Open "SELECT * FROM titles LIMIT 1", _
        conn, adOpenStatic, adLockOptimistic
    publishers.Open "SELECT * FROM publishers LIMIT 1", _
        conn, adOpenStatic, adLockOptimistic
    rel_title_author.Open "SELECT * FROM rel_title_author LIMIT 1", _
        conn, adOpenStatic, adLockOptimistic
    ' save publisher (if given)
    If Trim(txtPublisher) <> "" Then
        publishers.AddNew
        publishers!publName = Trim(txtPublisher)
        publishers.Update
        publID = LastInsertedID()
    End If

    ' save book title (perhaps with publID reference)
    titles.AddNew
    titles>Title = Trim(txtTitle)
    If publID <> 0 Then titles!publID = publID
    titles.Update
    titleID = LastInsertedID()

    ' save authors and make entries to rel_title_author table
    authors_array = Split(txtAuthor, ";")
    For i = LBound(authors_array) To UBound(authors_array)
        authors.AddNew
        authors!authName = Trim(authors_array(i))
        authors.Update
        authID = LastInsertedID()
        rel_title_author.AddNew
        rel_title_author!titleID = titleID
        rel_title_author!authID = authID
        rel_title_author.Update
    Next
End Sub
Private Function LastInsertedID() ... see earlier code

```

2. 变体2：用conn.Execute()指令来修改数据

在第2种变体里，有关代码与介绍如何使用PHP或Perl语言编写MySQL应用程序时给出的示例代码有几分相似：用多个字符串来拼装SQL命令、对字符串里的特殊字符进行转义（这项工作由辅助函数`Quote()`完成）等。除了这些细节，变体2在代码结构方面与变体大同小异：

```

Private Sub SaveData_WithSQLCommands()
    Dim i&, titleID&, authID&, publID&
    Dim authors_array
    ' save publisher (if specified)
    If Trim(txtPublisher) <> "" Then
        conn.Execute "INSERT INTO publishers (publName) " &
            "VALUES ('" & Quote(Trim(txtPublisher)) & "')"
        publID = LastInsertedID()
    End If

```

```

End If

' save book title (with any publID reference)
conn.Execute "INSERT INTO titles (title, publID) " &
    "VALUES ('" & Quote(Trim(txtTitle)) & "','" &
        IIf(publID <> 0, publID, "NULL") & ")"
titleID = LastInsertedID()
' save authors and make any entries in the rel_title_author table
authors_array = Split(txtAuthor, ";")
For i = LBound(authors_array) To UBound(authors_array)
    conn.Execute "INSERT INTO authors (authName) " &
        "VALUES ('" & Quote(Trim(authors_array(i))) & "')"
    authID = LastInsertedID()
    conn.Execute "INSERT INTO rel_title_author " &
        "(titleID, authID) " &
        "VALUES (" & titleID & ", " & authID & ")"
Next
End Sub
Private Function Quote$(tmp$) ... see earlier code

```

19.4.6 示例：把图像文件存入和读出 BLOB 数据列

下面的示例程序从当前子目录读出 test.jpg 文件的内容并把它存入一个数据表 (*Command2_Click*)。在第二步，二进制数据又从数据表里读出并存入 test1.jpg 文件 (*Command3_Click*)。为了检查这一切进行得是否顺利，最后再把新文件显示在一个 *Picture* 控件里(如图 19-7 所示)。为了存放这幅图像，示例程序将首先创建一个数据表，等程序即将结束时再删除它。

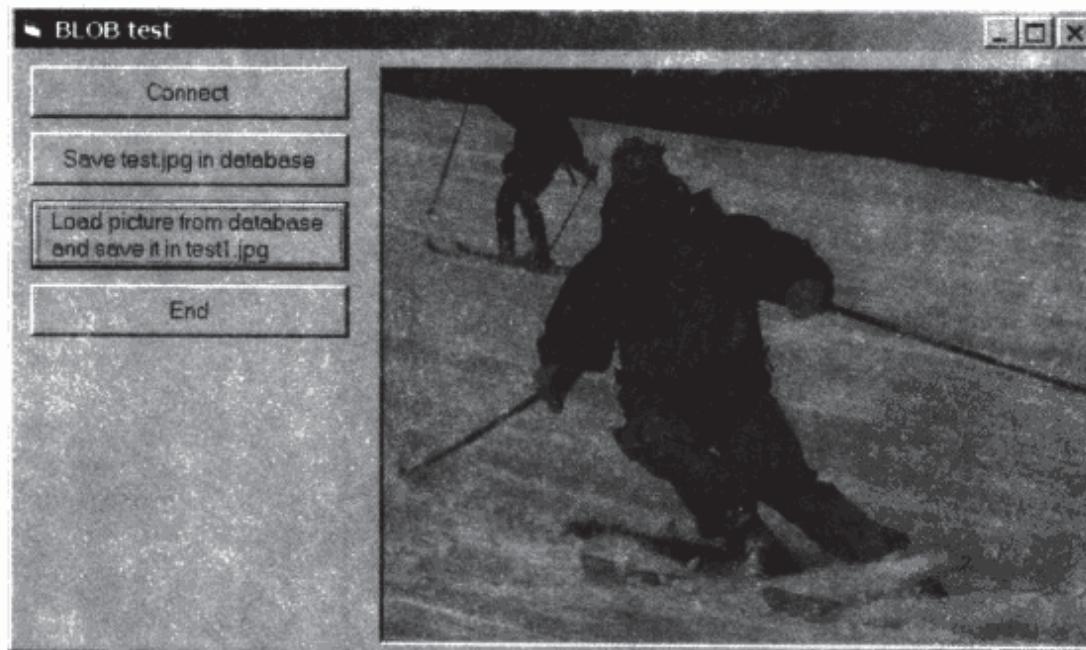


图 19-7 从 BLOB 字段读出并显示一张照片

```

example vb6\blob\form1.frm
Option Explicit
Dim conn As Connection
Dim id As Long

' create connection to MySQL
Private Sub Command1_Click()
    Set conn = New Connection
    'Options: 16395 = 16384 + 8 + 2 + 1
    ' Don't Optimize Column Width + Return Matching Rows +

```

```

' Allow Big Results + Change BIGINT Columns to INT
conn.ConnectionString = "Provider=MSDASQL;" +
    "DRIVER={MySQL ODBC 3.51 Driver};" +
    "Server=localhost;UID=root;PWD=uranus;" +
    "database=mylibraryodbc;Option=16395"
conn.Open
' create table
conn.Execute "CREATE TABLE IF NOT EXISTS testpic " +
    "(id INT NOT NULL AUTO_INCREMENT, pic MEDIUMBLOB, PRIMARY KEY (id))"
Command2.Enabled = True
End Sub

' load file into a Stream object
' and store in the table testpic
Private Sub Command2_Click()
    Dim rec As New Recordset
    Dim fname As String
    Dim st As New Stream
        ' open file
    fname = App.Path + "\test.jpg"
    st.Type = adTypeBinary
    st.Open
    st.LoadFromFile fname

    ' store file in table
    rec.CursorLocation = adUseClient
    rec.Open "SELECT * FROM testpic LIMIT 1", conn, adOpenKeyset,
        adLockOptimistic
    rec.AddNew
    rec!pic = st.Read
    rec.Update
    rec.Close
    st.Close

    ' note id (AUTO_INCREMENT number)
    id = LastInsertedID()
    Command3.Enabled = True
End Sub

' read BLOB from table and store in new file
Private Sub Command3_Click()
    Dim rec As New Recordset
    Dim fname As String
    Dim st As New Stream
        ' Stream object for the new file
    fname = App.Path + "\test1.jpg"
    st.Type = adTypeBinary
    st.Open
    ' read BLOB into the Stream object
    rec.CursorLocation = adUseClient
    rec.Open "SELECT pic FROM testpic WHERE id = " & id, conn,
        adOpenKeyset, adLockReadOnly
    st.Write rec!pic
    rec.Close

    ' store Stream in file
    st.SaveToFile fname, adSaveCreateOverWrite
    st.Close
    ' display file in picture control element
    Picture1.Picture = LoadPicture(fname)
End Sub

```

19.5 转换器：从 Microsoft SQL Server 到 MySQL

如果正在使用 Microsoft SQL Server 或 MSDE (Microsoft Data Engine, 微软数据引擎) 管理着一个数据库并正在考虑把它移植到 MySQL 环境，有两种办法可以把数据移植到一个 MySQL 数据库里去：

- 使用 Access。首先，把数据表导入一个空白的 Access 数据库，然后再把它们导入 MySQL。
- 使用 VBA/Visual Basic 脚本 mssql2mysql (这是笔者开发的一个工具，目前是 1.0 版)。因为不需要经过 Access 这个中间环节，所以这个脚本可以把数据库的结构更准确地复制过去。本节的各个小节都与这个程序有关。这个程序同时也是一个使用 ADO 接口为 MySQL 数据库编写应用程序的好例子，但因为篇幅的关系，这里就不给出它的源代码了。

提示 mssql2mysql 是一个脚本程序，它的最新版本可以在 <http://www.kofler.cc/mysql/mssql2mysql.html> 处找到。MySQL与其他数据库系统之间还有许多其他的商业化或免费的转换器，它们可以在 <http://solution.mysql.com/software/> 处找到。在下面这个网址处还可以找到一篇探讨如何从 Microsoft SQL Server 到 MySQL 的迁移的好文章：<http://dev.mysql.com/tech-resources/articles/migrating-from-microsoft.html>。

19.5.1 mssql2mysql 脚本的特点

- 可以自由获得 (GPL)。
- 能够完成对整个数据库的复制，由用户定义的所有数据表（结构、索引、数据）都没有问题。
- 能够把不符合 MySQL 有关规定的数据表或数据列的名字自动改为符合规定的名字（比如把 *My table* 改为 *My_table*）。
- 能够直接在 MySQL 服务器上生成一个内容是 SQL 命令的文本文件或者执行必要的命令。

19.5.2 对系统的要求

- 要想执行这个脚本，系统上必须安装有 VBA 解释器 (Office 2000 及以后的 Microsoft Office 软件包里的各个组件都带有 VBA 解释器) 或程序设计语言 Visual Basic 6。
- 这个脚本需要有 ADO 接口函数库和 SQLDMO 函数库才能正常工作。(SQLDMO 函数库可以用来通过编程去控制 Microsoft SQL Server，这个函数库随 Microsoft SQL Server 一起安装。)
- 如果不把转换后的数据库写入一个文本文件 (*.sql) 而是直接传输给一个现有的 MySQL 服务器，系统里还必须安装有 Connector/ODBC。

这个转换器已在由 SQL Server 2000、MySQL 5.0.2、ADO 8 和 Connector/ODBC 3.51.11 共同构成的测试环境下通过测试。

19.5.3 缺陷与不足

- 这个转换器无法转换以下数据库对象：一致性规则（外键约束条件）、视图、存储过程、用户自定义数据类型、访问权限。
- MySQL 规定每个数据表只能有一个 *AUTO_INCREMENT* 数据列，而且这个数据列还必须是数据表的主键 (*PRIMARY KEY*)。这个转换器没有对这一条件进行检查。因此，在执行这个脚本之前，必须自行检查 SQL Server 数据表是否满足这一条件并对之做出必要的修改。

- SQL Server 中的 *TIMESTAMP* 数据类型与 MySQL 中的 *TIMESTAMP* 数据类型不兼容，所以这个转换器将把前者转换为 MySQL 中的 *TINYBLOB* 数据类型。

19.5.4 使用方法

mssql2mysql 脚本的源代码文本文件可以从 <http://www.kofler.cc/mysql/mssql2mysql.html> 下载。可以用 Visual Basic 6 或者是使用一个带有 VBA 6 支持的软件程序（比如 Excel 2000、Word 2000、Access 2000 等）来执行这个脚本。

- Visual Basic 6：如果使用的是 Visual Basic 6，请创建一个新项目，然后把这个脚本的全部代码复制到某个表单的代码窗口里，最后按下 F5 功能键就可以执行这个程序。
- VBA：如果使用的是一个 VBA 兼容程序，按下 Alt+F11 组合键切换到 VBA 开发环境，在那里先插入一个新的空白模块，然后把这个脚本的代码全部复制到那个新模块里，再按下 F5 功能键打开宏执行窗口并执行那个名为 *main* 的过程即可。

19.5.5 设置有关参数

mssql2mysql 脚本在进行数据库转换时使用的所有参数由一些常数控制，它们都列在了脚本代码的开头部分。前 5 个常数给出了 Microsoft SQL Server 服务器的登录信息，既可以通过 Windows 2000 的用户登录系统进入，也可以明确地给出用户名和密码，如下所示：

```
Const MSSQL_SECURE_LOGIN = True 'login type (True for 2000/XP Security)
Const MSSQL_LOGIN_NAME = ""      'user name (" for 2000/XP Security)
Const MSSQL_PASSWORD = ""        'password (" for 2000/XP Security)
Const MSSQL_HOST = "mars"        'computer ("local)" for localhost)
Const MSSQL_DB_NAME = "pubs"     'database name
```

接下来的 *OUTPUT_TO_FILE* 常数控制着这个脚本是生成一个文本文件来存放 SQL 命令（=1 时）、还是立刻执行这些命令（=0 时）。生成文本文件的好处是如果转换工作遇到问题，可以直接编辑文本文件；缺点是如此生成的文本文件使用的是 Windows 字符集 *cp850*（与 *latin1* 字符集基本相当），万一 Microsoft SQL Server 数据库里的 Unicode 字符串包含着无法用 *latin1* 字符集表示的字符，它们有可能会丢失。

```
Const OUTPUT_TO_FILE = 0           '1 write file,
                                '0 execute SQL commands at once
```

如果决定 *OUTPUT_TO_FILE*=1，则必须在 *OUTPUT_FILENAME* 常数处给出一个结果文件名：

```
Const OUTPUT_FILENAME = "C:\export.sql"
```

如果决定 *OUTPUT_TO_FILE*=0，就必须给出登录 MySQL 服务器所需要的所有参数。请注意，必须具备足够的权限才能执行 *CREATE DATABASE* 等命令：

```
Const MYSQL_USER_NAME = "root"    'user name
Const MYSQL_PASSWORD = "uranus"   'password
Const MYSQL_HOST = "localhost"    'computer name or "localhost"
Const MYSQL_PORT = 3306          'MySQL port
```

接下来的两个常数分别对应着 Connector/ODBC 3.51 和 2.51 版本，请根据具体情况做出选择：

```
Const MyODBCVersion = "MySQL ODBC 3.51 Driver"
Const MyODBCVersion = "MySQL"      'for 2.50.n
```

最后，还有一些常数是用来控制转换细节的选项，它们是：

- **NEW_DB_NAME**。是否需要给 MySQL 数据库起一个新名字。如果这个常数为空，在创建 MySQL 数据库时将沿用 Microsoft SQL Server 数据库的老名字。
- **DROP_DATABASE**。转换开始前是否要把已经存在的同名 MySQL 数据库删掉。
- **MAX_RECORDS**。为每个数据表最多转换多少条记录；0 表示所有的数据记录都将转换。这个选项提供了一种快速测试机制，比如说，可以先为每个数据表转换 10 条记录，然后检查一下转换效果并做出必要的调整。如果没有任何问题，再开始转换整个数据库不迟。这种测试很有必要，它有助于避免在转换工作进行到半截或即将结束时才发现需要调整某个参数或某个数据表出了问题。
- **VARCHAR_MAX**。指定 VARCHAR 数据列的最大字符长度，超过这个长度的字符串数据列将被转换为 TEXT 数据列。如果 MySQL 服务器的版本是 5.0.2 或更高，就至少应该把这个常数设置为 255。比较新的 MySQL 服务器版本可以支持最大长度为 65 535 个字节的 VARCHAR 数据列，但最大字符长度还要取决于具体的字符集——如果使用的是 UTF8 字符集，把 32 000 作为这个常数的上限比较保险。
- **TABLE_ENGINE**。设置 MySQL 数据表的数据表类型（InnoDB 或 MyISAM）。
- **CHARSET**。为数据表里的文本数据列设置一个字符集。
- **COLLATION**。设置一种排序方式。如果这个常数为空，则使用选中的字符集的默认排序方式。

```

Const NEW_DB_NAME = ""           'MySQL database name
Const UNICODE_TO_BLOB = False    'Unicode --> BLOBs?
Const DROP_DATABASE = True      'begin with DROP database?
Const MAX_RECORDS = 0           '0: convert all data
Const VARCHAR_MAX = 255          'larger values only for MySQL-Server >= 5.0.3
Const TABLE_ENGINE = "InnoDB"    'MyISAM or InnoDB
Const CHARSET = "utf8"          'desired character set in the MySQL tables
Const COLLATION = ""            'empty for default sort order

```

19.6 VBMySQLDirect

本章此前介绍的“ADO + Connector/ODBC”编程技巧虽然可以达到目的，但总体效率比较低。如果速度比与 AOD 保持兼容更重要，本节将要介绍的 *VBMySQLDirect* 接口应该更能引起兴趣。这个函数库大大减少了从 Visual Basic 到 MySQL 之间的中间环节，Visual Basic 程序员可以更直接地去访问 MySQL。这个函数库提供的 *MYSQL_CONNECTION*、*MYSQL_RS*、*MYSQL_FIELD* 和 *MYSQL_ERR* 类有着与 ADO 对象 *Connection*、*Recordset*、*Field* 和 *Error* 相似的功能和用途，但它们之间并不兼容。

VBMySQLDirect 本身由 Visual Basic 代码构成，它们来自目前已不再继续开发的 *MyVbQL* 项目。本书测试的 *VBMySQLDirect* 版本是 1.0.2。

VBMySQLDirect 需要 MySQL 客户函数库 libmySQL.dll 的配合，它们构成的从 Visual Basic 到 MySQL 的通信链路是这样的：Visual Basic → *VBMySQLDirect* → libmySQL.dll → MySQL。

与基于“ADO + Connector/ODBC”的解决方案相比，*VBMySQLDirect* 代码的速度要快很多。另一个显而易见的优点是 *VBMySQLDirect* 方案比“ADO + Connector/ODBC”方案更加简明。换句话说，就是可以用更少的函数实现同样的功能，因而出现编程错误的概率也就小了很多。此外，*VBMySQLDirect* 程序的发行和安装工作也比较简单，因为它们不需要用户安装 ADO 即可使用（ADO 会让安装文件的体积膨胀许多）。不过，基于 *VBMySQLDirect* 的解决方案也有以下几个不足：

- *VBMySQLDirect* 与 ADO 不兼容，把现有的 ADO 代码转换为 *VBMySQLDirect* 代码需要花费一些时间和精力。

❑ *MYSQL_RS* 对象不支持 Visual Basic 语言提供的数据库绑定控件，所以在 *VBMYSQLDirect* 程序里不能像在 ADO 程序里那样使用数据库控件。（不过，在 *VBMYSQLDirect* 的网站上可以找到一个能够让 *VBMYSQLDirect* 与 Crystal Reports 软件进行通信的接口函数库，可以利用这个函数库在 *VBMYSQLDirect* 程序里使用这个流行的报表生成工具。）

VBMYSQLDirect 可以按照 GNU Library General Public License 许可证中的有关规定免费获得，但与之配合的 libmysql.dll 函数库在发行时使用的却是更为严格的 GPL 许可证。这意味着用“*VBMYSQLDirect + libmysql.dll*”开发出来的商业化应用程序必须遵守与 Connector/ODBC 程序同样的规则：如果利用 *VBMYSQLDirect* 开发出来的程序不符合 GPL 对开源的定义，顾客就必须获得一份 MySQL 服务器的商用许可证才可以使用它们。

19.6.1 安装

VBMYSQLDirect 的安装程序可以在 <http://www.vbmysql.com/projects/vbmysqldirect/> 处找到。安装过程先把 *VBMYSQLDirect* 的源代码和一些示例程序复制到 Programs\VBMySQLDirect 目录，然后把库文件 vbmysqldirect.dll 和 libmysql.dll 安装到 Windows\System32 目录，最后把 vbmysqldirect.dll 注册到 Windows 注册表里。请注意，*VBMYSQLDirect* 1.0.2 自带的 libmysql.dll 库文件的版本比较老（还是 MySQL 4.1.n 时期的）。

注意 安装 *VBMYSQLDirect* 之后，系统里会用到 libmysql.dll 库文件的其他软件可能会工作不正常。

比如说，如果此前曾按本书第 2 章给处的步骤安装过 PHP，计算机里现在就会有两个 libmysql.dll 文件，一个在 C:\Windows 目录里（来自 PHP），另一个在 Windows\System32 目录里（来自 *VBMYSQLDirect*）。

等 Apache 服务器下次启动的时候，因为它会先去 Windows\System32 目录、后去 C:\Windows 目录寻找它需要的 DLL，所以它实际加载的将是老版本的 libmysql.dll 文件，而这将导致 PHP 发生错误，因为它需要新版本的 libmysql.dll 文件才能正常工作。

解决这个问题的办法是让 Apache 退出运行并重新命名 Windows\System32\libmysql.dll 文件或干脆删掉这个文件。再次启动 Apache 之后，PHP 就可以重新用上 Windows\libmysql.dll 文件了。这么一来，现在 *VBMYSQLDirect* 也将使用这个新版本的 libmysql.dll 文件。根据 *VBMYSQLDirect* 论坛上的一份帖子的说法，虽然“*VBMYSQLDirect + 新版 libmysql.dll 文件*”的组合尚未经过全面的官方测试，但能够正常工作。在笔者进行的测试中，这种组合没有发生任何问题。

19.6.2 应用

要想在自己的 Visual Basic 项目里使用 *VBMYSQLDirect*，只须为它的库文件创建一个引用链接即可。可以在菜单命令 Project（项目）| References（引用）打开的对话框里，在 *VBMYSQL Direct v1.0* 名下找到这个函数库。有了这个步骤，就可以像平时那样在 Visual Basic 项目里使用 *VBMYSQLDirect* 提供的 4 个 *MYSQL_xxx* 对象了。

19.6.3 示例

本节示例完成的是一项“老”工作：把图书作者的姓名字符串存入 titles 数据表里的 authors 数据列。但这次的做法是先创建一个 *MYSQL_CONNECTION* 对象并把主机名、用户名、密码和数据库名

等参数传递给 *OpenConnection* 方法，如下所示：

```
' authors_for_titles_VBMySQLDirect\form1.frm
' create connection to MySQL
Dim conn As New MYSQL_CONNECTION
Private Sub Command1_Click()
    conn.OpenConnection "localhost", "root", "xxx", "mylibrary"
    Command2.Enabled = True
End Sub
```

这一次，将使用 *Execute* 方法从 *MYSQL_CONNECTION* 对象获得一个 *MYSQL_RS* 对象。注意，在访问当前结果记录的个别字段时，这次需要使用 *rs.Fields("columnname").Value* 属性（而不是 ADO 编程时的 *rs!columnName*）。

MYSQL_RS 对象里的数据修改起来很简单：给各有关字段（数据列）的 *Value* 属性分别赋一个新值、再用 *Update* 方法把它们存入数据库即可。（还可以使用 *AddNew* 来插入新记录或者使用 *Delete* 方法来删除一条记录，但因为我们的这个例子不涉及这些操作，所以在代码里没有给出这方面的演示。）

注意 在用完 *MYSQL_RS* 对象后一定要用 *Close* 方法明确地把它们关闭。如果忘了这么做，这些对象占用的内存资源将一直无法释放。

```
' change the titles table
Private Sub Command2_Click()
    Dim authors_str, titleID
    Dim titles As MYSQL_RS
    Dim authors As MYSQL_RS
    ' titles Recordset for a loop over all titles
    ' and for changing the authors column
    Set titles = conn.Execute(
        "SELECT titleID, authors FROM titles")
    ' loop over all book titles
    While Not titles.EOF
        titleID = titles.Fields("titleID").Value
        ' authors recordset to read the authors of a title
        Set authors = conn.Execute(
            "SELECT authname FROM authors, rel_title_author " &
            "WHERE authors.authID=rel_title_author.authID " &
            " AND rel_title_author.titleID = " & titleID & " " &
            "ORDER BY authName")
        ' loop over all authors
        authors_str = ""
        Do While Not authors.EOF
            ' assemble string with the author names
            If Not IsNull(authors.Fields("authName").Value) Then
                If authors_str <> "" Then authors_str = authors_str + "; "
                authors_str = authors_str & authors.Fields("authName").Value
            End If
            authors.MoveNext
        Loop
        authors.CloseRecordset
        ' save authors field in the titles recordset object
        If Len(authors_str) > 254 Then
            authors_str = Left(authors_str, 254)
        titles.Fields("authors").Value = authors_str
        titles.Update
        titles.MoveNext
    Wend
    titles.CloseRecordset
End Sub
```

第 20 章**Visual Basic .NET 和 C#**

20

本章将介绍如何使用 Visual Basic 2005（简称 VB .NET 2）和 C#这两种程序设计语言编写 MySQL 应用程序。这两种语言的数据库编程工作都使用了数据库接口 ADO .NET 和 MySQL 驱动程序包 Connector/Net。

注解 本章中的大部分示例都使用了 VB .NET 作为程序设计语言。C#程序员可以在本书的配套网站 (www.apress.com) 上找到本章绝大多数示例程序的 C# 源代码。

本章所有的示例程序都是用 VS.NET 开发环境开发的；在编写这些示例程序时使用的是各有关函数库和导入文件的默认设置。

用 ADO .NET 编写 MySQL 应用程序的理论和实践并非仅适用于 MySQL：在与数据库建立起连接之后，后面的代码与任何其他的 ADO .NET 程序几乎没有什么区别。如果说有什么不同的话，那就是本章示例程序所使用的类大都有着 *OdbcXxx* 和 *MySQLXxx* 形式的名字，而 ADO .NET 教科书里介绍的类一般都有着 *OleDbXxx* 和 *SqlXxx* 形式的名字。

20.1 ADO .NET 与 MySQL 之间的通信

通过 ADO .NET 去访问 MySQL 数据库的办法有很多，这与喜欢 C# 还是喜欢 VB .NET 并没有多大的关系。下面是几种比较常见的候选解决方案。

- **Connector/ODBC + ODBC 数据泵。** 在这种变体里，ADO .NET 与 MySQL 之间的通信是通过 ODBC 实现的。MySQL 与 ODBC 之间的接口由 Connector/ODBC 负责提供，向 ADO .NET 提供数据的是 ODBC 数据泵（相应的名空间是 *Microsoft.Data.Odbc*）¹。.NET 1.1 及更高版本的 ADO .NET 里已经集成了这个数据泵。如此形成的通信链路是：VB .NET 或 C# → ADO .NET → ODBC 数据泵 → Connector/ODBC → MySQL。
- **Connector/ODBC + OLE-DB 数据泵。** 在这个变体里，ADO .NET 与 MySQL 之间的通信也是通过 ODBC 实现的，只是在 MySQL 与 ODBC 之间增加了一个 OLE-DB 环节。如此形成的通信链路是：VB .NET 或 C# → ADO .NET → OLEDB 数据泵 → OLEDB/ODBC 数据泵 → Connector/ODBC → MySQL。
- **ADO .NET 专用的受控数据泵。** 为了提高 ADO .NET 与 MySQL 之间的通信效率，最好使用一

1. 数据泵（data provider）是 ADO .NET 对数据库驱动程序的称呼，有些书就把它直译为“数据提供者”。

种专为 ADO .NET 开发的 MySQL 数据泵。如此形成的通信链路是: VB .NET 或 C# → ADO .NET → MySQL 数据泵 → MySQL。

使用这种数据泵的最大好处是能够形成一条最短的通信链路, 而这意味着各环节之间的兼容性问题更少、通信效率更高。这种数据泵的另一个优点是向程序员提供了许多针对 MySQL 时间系统而设计和优化的函数(如支持压缩通信协议的函数等)。不过, 从另一个角度看, 这些“额外的”函数也正是这一变体的最大缺点: 程序代码缺乏可移植性, 想改用另外一种数据泵相当困难, 甚至连改用另外一种数据库系统都很不容易。

还好, 目前已经有了许多种这样的数据泵可供程序员们选择。截止到 2003 年 2 月, 笔者在因特网上找到了以下几种 ADO .NET 专用的受控数据泵:

- **Connector.Net (GPL)**。这个数据泵是 MySQL 公司官方推出的 .Net 专用数据泵, 它是在以前的 ByteFX 数据泵的基础上发展而来的。详细情况见 <http://dev.mysql.com>。
- **MySQLDriverCS (开源, GPL)**。这是 MySQL 官方数据泵的开源版本。详细情况见 <http://source.net/projects/mysqldrivercs/>。
- **MySQLDriver.Net**。由 CoreLab 公司推出的这个数据泵也是一个商业化的数据库驱动程序包。详细情况见 <http://www.crlabs.com/mysqlnet/>。

想凭一己之力对这里提到的所有变体做出全面测试是不可能的。笔者只对两种变体进行了测试: 一种是 Connector.Net, 另一种是 Connector/ODBC + ODBC 数据泵(参见 20.1.1 和 20.1.2 节)。

注解 还有一种办法是根本不使用 ADO .NET, 而是使用 VB .NET 或 C# 中的 ADO 函数库。因为有 COM 可以作为中间环节, 所以这个方案本身是可行的。但因为 ADO 在效率和安全方面都不如 ADO .NET, 与 .NET 框架下的其他组件配合得也不是很好, 所以没有把它开列在上面, 也不打算继续探讨这方面的问题。

20.1.1 通过 Connector/Net 连接数据库

1. 安装 Connector/Net

Connector/Net 是在以前的 ByteFX 数据泵的基础上发展而来的, 它目前是 MySQL 公司为 MySQL 软件正式推出和支持的 ADO .NET 驱动程序集。如果拿不准应该选用哪一种驱动程序, 就应该把 Connector/Net 当作第一选择。这个驱动程序集可以从 <http://dev.mysql.com> 网站下载并在 GPL 许可证下免费使用。如果正在开发一个商业化项目, 购买这个软件的顾客将必须持有 MySQL 服务器的商用许可证才允许运行它。

下载的 Connector/Net 驱动程序集是一个*.ZIP 压缩文档, 其内容是一个*.msi 文件。这个*.msi 文件先把 Connector/Net 驱动程序以及各种各样的示例文件安装到 Programs \MySQL\MySQL Connector Net n.n 目录, 然后把真正的驱动程序(一个名为 MySql.Data.dll 的库文件)注册到总装缓存区(global assembly cache, GAC)里去(注意: 这个注册操作目前只在 .NET Framework 1.0 和 1.1 版本下才能成功)。

2. 使用 Connector/Net

为了在自己的项目里使用 Connector/Net, 必须通过 VS.NET 开发环境的菜单命令 Project(项目) | Add Reference(添加引用) 打开的对话框为库文件 System.Data.dll 和 MySql.Data.dll 创建一个引用链接。在其他配置相同的情况下(Visual Studio 2005 Beta 1、Connector/Net 1.0.4 等), 笔者向 .NET Framework 2.0 的 GAC 注册 MySql.Data.dll 文件的尝试没有成功——这个库文件根本没有出现在

“.NET”对话框里。如果也遇到了同样的问题，就需要以手动方式来注册这个库文件：切换到 Browse（浏览）设置页，找到并打开下面这个文件：Programs\MySQL\MySQL Connector Net 1.0.4\bin\ .NET 1.1\ MySql.Data.dll。

为了避免在编写代码时不得不反复写出 *MySql.Data.MySQLClient.xxx*，应该在程序的开头部分放上一条 *Imports* (VB .NET) 或 *using* (C#) 指令，如下所示：

```
Imports MySql.Data.MySqlClient ' VB.NET
using MySql.Data.MySqlClient; // C#
```

3. 连接数据库

在使用 Connector/Net 的情况下，与数据库建立连接的工作需要通过 MySQLConnection 对象来进行。包含连接数据的字符串由多个部分构成，各部分之间要用分号 (;) 隔开。表 20-1 对最重要的参数进行了解释。注意，许多参数可以接受几种不同形式的参数名（比如说，*User Id*、*Uid*、*Username* 和 *User name* 其实是同一个参数的不同名字）。各有关参数的各种命名形式以及它们的默认设置值可以在 Connector/Net 的在线文档里查到（以 *ConnectionString* 作为关键字进行一下搜索）。

表 20-1 Connector/Net 的连接参数

参 数	含 义
<i>Data Source</i> 或 <i>Hostname</i>	MySQL 服务器的主机名或 IP 地址
<i>Initial Catalog</i> 或 <i>Database</i>	数据库的名字
<i>User Id</i> 或 <i>Username</i>	用户名
<i>Password</i>	密码
<i>Connection Timeout</i>	连接倒计时时间（默认值是 15 秒）；未能在计时结束前建立起连接将触发一个错误
<i>Allow Batch</i>	是否允许一次执行多条 SQL 命令（默认值是 <i>true</i> ：允许）；那些 SQL 命令之间必须用分号隔开

用来创建一条数据库连接的代码如下所示：

```
Dim myconn As New MySqlConnection(
    "Data Source=localhost;Initial Catalog=mylibrary;" +
    "User ID=root;PWD=xxxxxx")
myconn.Open()
```

注解 在可以用在连接字符串里的参数当中，有一个在正式文档里没有记载，它显然来自早期的 ByteFX 驱动程序：*UseCompression=true* 的效果是本程序与 MySQL 服务器之间的通信将采用压缩格式。这在低速网络上会很有用。*UseCompression=true* 要求本程序包含着一个指向库文件 ICSharpCode.SharpZipLib.dll 的引用链接。这个函数库可以在安装 Connector/Net 后在 Programs\MySQL\MySQL Connector Net n.n.n\src 目录里找到。

4. 应用程序的发行与安装

基于 Connector/Net 的应用程序要求系统必须安装有 MySql.Data.dll 库文件才能正常运行。因此，如果想把开发出来的应用程序复制给其他人使用，就应该保证对方的计算机里安装有 MySql.Data.dll（最保险的办法是把这个库文件和应用程序一起安装到同一个目录里去）。

5. VB .NET示例

下面这个示例程序将创建一条与本地 MySQL 服务器的连接，获得一份全体出版公司的排序名单

(有关数据来自 *mylibrary* 数据库的 *publishers* 数据表), 然后把它们输出到一个文本显示框里(如图 20-1 所示):

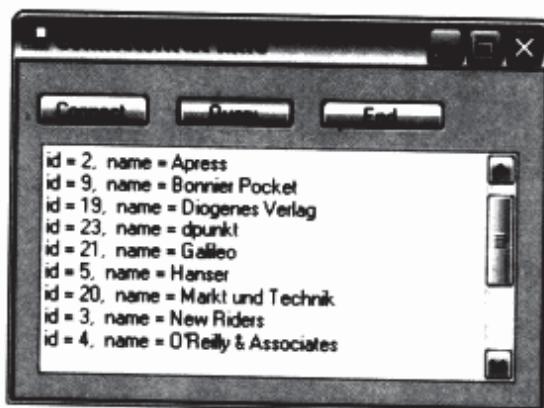


图 20-1 Connector/Net 示例程序

```
' example code vbnet\connector_net_intro
Imports MySql.Data.MySqlClient
Public Class Form1
    Dim myconn As MySqlConnection

    ' create connection
    Private Sub Button1_Click(...) Handles Button1.Click
        Try
            myconn = New MySqlConnection(
                "Data Source=localhost;Initial Catalog=mylibrary;" +
                "User ID=root;PWD=xxxxx")
            myconn.Open()
        Catch myerror As MySqlException
            MsgBox("Database connection error: " & myerror.Message)
            Me.Close()
        End Try
        Button2.Enabled = True
    End Sub

    ' execute SQL query, display result in text box
    Private Sub Button2_Click(...) Handles Button2.Click
        Dim com As MySqlCommand
        Dim dr As MySqlDataReader
        com = New MySqlCommand(
            "SELECT publID, publName FROM publishers ORDER BY publName",
            myconn)
        dr = com.ExecuteReader()
        While dr.Read()
            TextBox1.AppendText("id = " & dr!publID &
                ", name = " & dr!publName & vbCrLf)
        End While
        dr.Close()
    End Sub

    ' program end
    Private Sub Button3_Click(...) Handles Button3.Click
        Me.Close()
    End Sub
End Class
```

6. C#示例

同样功能的 C#程序代码如下所示:

```

// example code csharp\connector_net_intro
using ...;
using MySql.Data.MySqlClient;

namespace connector_net_intro {
    partial class Form1 : Form {
        MySqlConnection myconn;
        public Form1() {
            InitializeComponent();
        }

        // create connection
        private void button1_Click(object sender, EventArgs e) {
            try {
                myconn = new MySqlConnection(
                    "Data Source=localhost;Initial Catalog=mylibrary;" +
                    "User ID=root;PWD=uranus");
                myconn.Open();
            }
            catch (MySqlException myerror) {
                MessageBox.Show("MySQL connection error: " + myerror.Message);
                this.Close();
            }
            button2.Enabled = true;
        }

        // execute SQL query, display result in text box
        private void button2_Click(object sender, EventArgs e) {
            MySqlCommand com;
            MySqlDataReader dr;
            com = new MySqlCommand(
                "SELECT publID, publName FROM publishers ORDER BY publName",
                myconn);
            dr = com.ExecuteReader();
            while (dr.Read())
                textBox1.AppendText("id = " + dr["publID"] +
                    ", name = " + dr["publName"] + Environment.NewLine);
            dr.Close();
        }

        // program end
        private void button3_Click(object sender, EventArgs e) {
            this.Close();
        }
    }
}

```

20.1.2 用 ODBC 数据泵连接数据库

1. 连接数据库

通过 ODBC 数据泵连接服务器的代码与使用 ADO 完成这一任务的代码（见第 19 章）非常相似。主要的区别有两点：一是现在使用的是 *OdbcConnection* 对象；二是现在可以把连接字符串直接传递给构造器，连接字符串本身与我们在上一章编写 ADO 程序时使用的完全一样。

作为一个原则，*Options* 参数的设置值必须是 3。这等于是同时选中 *Don't Optimize Column Width (1)*（不对数据列的宽度进行优化）和 *Return matching rows (2)*（返回匹配的数据行）这两个选项。在基于 ADO .NET 的数据库应用程序里不需要选中 *Change BIGINT Column to INT (16384)*（把 *BIGINT* 数据列转换为 *INT*）选项。

```

' VB.NET
Dim odbcconn As New System.Data.Odbc.OdbcConnection(
    "Driver={MySQL ODBC 3.51 Driver};Server=localhost;" +
    "Database=mylibraryodbc;UID=root;PWD=uranus;Options=3")
odbcconn.Open()

// C#
OdbcConnection odbcconn = new System.Data.Odbc.OdbcConnection(
    "Driver={MySQL ODBC 3.51 Driver};Server=localhost;" +
    "Database=mylibraryodbc;UID=root;PWD=uranus;Options=3");
odbcconn.Open();

```

2. VB .NET示例

下面这个示例程序将把 *mylibrary.publisher* 数据表所收录的所有出版公司列成一个按字母表顺序排序的清单显示在一个控制台窗口里（如图 20-2 所示）。在与数据库建立起连接之后，我们先用 *OdbcCommand()*方法执行了一条 *SELECT* 查询命令，然后在一个循环里使用了一个 *OdbcDataReader* 对象来处理这条 *SELECT* 命令的查询结果。

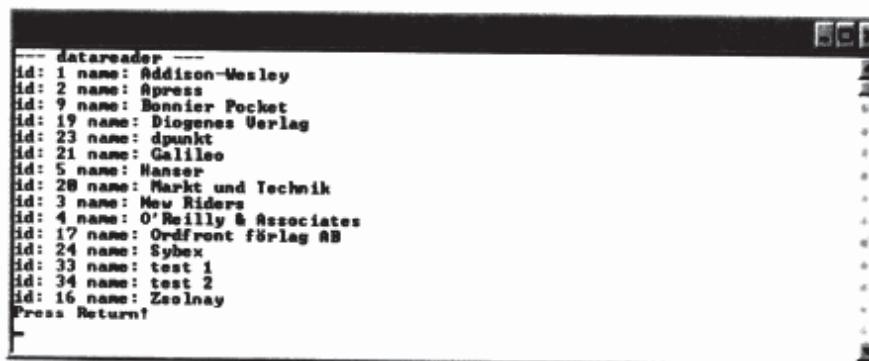


图 20-2 ODBC 示例程序

```

' example vbnet/odbc_connectintro/module1.vb
Option Strict On
Imports System.Data
Imports MicrosoftSystem.Data.Odbc
Module Module1
    Dim odbcconn As OdbcConnection
    Sub Main()
        odbcconn = New OdbcConnection(
            "Driver=MySQL ODBC 3.51 Driver;Server=localhost;" +
            "Database=mylibrary;UID=root;PWD=xxxxx;Options=16387")
        odbcconn.Open()
        read_publishers_datareader()
        odbcconn.Close()
    End Sub

    ' example for OdbcDataReader
    Sub read_publishers_datareader()
        Dim com As OdbcCommand
        Dim dr As OdbcDataReader
        com = New OdbcCommand(
            "SELECT publID, publName FROM publishers ORDER BY publName", _
            odbcconn)
        dr = com.ExecuteReader()
        While dr.Read()
            Console.WriteLine("id: {0} name: {1}", dr!publID, dr!publName)
        End While
        dr.Close()
    End Sub
End Module

```

3. C#示例

同样能的 C#程序代码如下所示：

```
// example csharp\odbc_connect\Class1.cs
using System;
using System.Data.Odbc;

namespace odbc_connect {
    class mainclass {
        [STAThread]
        static void Main(string[] args) {
            Odbctest tst = new Odbctest();
            tst.ReadPublishersDataset();
            Console.WriteLine("Return drücken");
            Console.ReadLine();
        }
    }
    class Odbctest {
        OdbcConnection odbcconn;

        public Odbctest() { // use constructor to connect
            odbcconn = new OdbcConnection(
                "Driver={MySQL ODBC 3.51 Driver};Server=localhost;" +
                "Database=mylibrary;UID=root;PWD=xxxxx;Options=16387");
            odbcconn.Open();
        }

        public void ReadPublishersDataset() {
            OdbcCommand com = new OdbcCommand(
                "SELECT publID, publName FROM publishers ORDER BY publName",
                odbcconn);
            OdbcDataReader dr = com.ExecuteReader();
            while(dr.Read()) {
                Console.WriteLine("id: {0} name: {1}",
                    dr["publID"], dr["publName"]);
            }
            dr.Close();
        }
    }
}
```

20.2 编程技巧

从现在开始，本章后面内容里的示例都将假设与 MySQL 服务器的连接是用 Connector/Net 建立的，在建立连接时打开的 *MySqlConnection* 对象的名字是 *myconn*。换句话说，在后面的各有关示例里，变量名 *myconn* 总是代表着一个适当的 MySQL 数据库。

如果想改用另外一种数据泵来测试本章后面内容里的示例程序，必须做好以下几项准备工作：为使用的函数库在 VS.NET 开发环境里创建一个引用链接；对示例程序开头部分的 *Imports* 或 *using* 指令做出必要的修改；把示例代码中 *MySqlXxx* 替换为 *OdbcXxx* 或是在使用的数据泵里负责实现同样功能的对象名/方法名/属性名；根据使用的数据泵的语法要求对连接字符串做必要的修改；根据使用的数据泵的具体情况，或许还需要对示例代码的某些细节（如：投射操作符、SQL 命令的参数语法等）做一些小的改动。

20.2.1 执行 SQL 命令 (*MySqlCommand* 对象)

要想执行 SQL 命令，必须先有一个 *MySqlCommand* 对象。创建 *MySqlCommand* 对象的办法有两

种：一是把 *MySqlConnection* 对象和 SQL 命令一起传递给 *MySqlCommand* 对象的构造器；二是把它们分开来处理（如果想只用一个 *MySqlCommand* 对象、但执行多条 SQL 命令，第二种办法将非常有用。）

执行 SQL 命令的具体办法有 3 种：

- *ExecuteNonQuery()*方法。这个方法用来执行不会立刻返回一个结果的各种 SQL 命令（即 *UPDATE*、*INSERT* 和 *DELETE* 命令），如下所示：

```
Dim com As New MySqlCommand()
com.Connection = myconn
com.CommandText = "INSERT INTO table (colname) VALUES ('abc')"
com.ExecuteNonQuery()
```

- *ExecuteReader()*方法。如果想在执行一条 *SELECT* 查询命令后使用 *MySqlDataReader*¹ 对象去读取它的查询结果，就需要使用这个方法（请参见本章稍后内容里的示例）。

- *ExecuteScalar()*方法。这个方法用来执行仅返回一个离散值的 SQL 命令（比如 *SELECT COUNT(*) FROM table*）；马上就会在 20.2.2 节看到一个例子。

如果在执行 SQL 命令的过程中发生错误，有关代码就会抛出一个 *MySqlException* 对象，这种对象可以用一个 *Try-Catch* 语法结构来捕获：

```
Try
    com.ExecuteNonQuery()
Catch e As MySqlException
    MsgBox(e.Message)
    Stop
End Try
```

在构造 SQL 命令的时候，程序员必须严格遵守有关的 MySQL 语法规则：字符串里的特殊字符如 “'”、“””、“\” 和零值字节必须被转义为 “\”、“\\"”、“\\” 和 “\\0”；日期/时间值必须有正确的格式（2005-12-31 23:59:59）等。如果需要连续执行多条彼此相似的 *INSERT* 或 *DELETE* 命令，还可以利用带参数的 SQL 命令来提高效率。

AUTO_INCREMENT 数据列

用 *INSERT* 命令最近一次插入的记录的 ID 编号可以用 SQL 命令 *SELECT LAST_INSERT_ID()* 来确定。我们刚才提到的 *ExecuteScalar()* 方法可以把这个 ID 编号值返回为一个 *Long* 类型的整数：

```
Dim n As Long
Dim com As New MySqlCommand()
com.Connection = myconn
com.CommandText = "INSERT INTO table (columns ...) VALUES (...)"
com.ExecuteNonQuery()
com.CommandText = "SELECT LAST_INSERT_ID()"
n = CLng(com.ExecuteScalar())
```

20.2.2 带参数的 SQL 命令 (*MySqlParameter* 对象)

SQL 命令里的参数必须写成 *?name* 的形式。这里又有两种具体做法。第一种做法是直接用 *Add("?name", data)* 方法给出 SQL 命令的参数值，然后执行这条命令。这种做法尤其适合那些只须执行一次的 SQL 命令：

1. 原书在后面的讨论里多次把 *MySqlDataReader* 简单地写成 *DataReader*。这在理论上没有错——毕竟 *MySqlDataReader* 对象是从 *DataReader* 对象派生出来的，但在教科书里这么写还是有些不太妥当，所以我们在翻译时使用的是 *MySqlDataReader*。当然，在必须译成 *DataReader* 的地方没有乱改。——译者注

```

Dim com1 As MySqlCommand
com1 = myconn.CreateCommand()
com1.CommandText =
    "INSERT INTO authors (authName, authID, ts) VALUES(?name, ?id, ?ts)"
com1.Parameters.Add("?name", "authorname")
com1.Parameters.Add("?id", 12345)
com1.Parameters.Add("?ts", DateTime.Now())
com1.ExecuteNonQuery()

```

第二种做法是先为 SQL 命令的各个参数分别定义一个数据类型 (*MySqlDbType.xxxx*, 其中 *xxxx* 可以是任何一种 MySQL 数据类型) 并绑定到一个 *MySQLParameter* 对象上, 然后用 *Prepare()* 方法对 SQL 命令做执行前的预处理, 再通过对各有关 *MySQLParameter* 对象进行赋值的办法去传递参数值, 最后执行这条命令。这里所说的“赋值”指的是对 *MySQLParameter* 对象的 *Value* 属性进行设置, 而且只能发生在对 SQL 命令进行完预处理之后。

这第二种做法非常适合那些需要执行不止一次的 SQL 命令, 它可以最大限度地提高工作效率。如果正在访问的 MySQL 服务器是 4.1 或更高的版本, 它就会把这些 SQL 命令当作预处理语句来执行。这么做的优点有很多: 可以加快处理速度、可以对传递来的所有参数值进行类型检查、可以把参数值自动转换为最适当的 MySQL 格式等。这些优点在对二进制数据 (*BLOB*) 进行处理的时候体现得尤其明显。

```

' prepare command
Dim com2 As MySqlCommand
Dim pname, pid, pts As MySqlParameter
com2 = myconn.CreateCommand()
com2.CommandText =
    "INSERT INTO authors (authName, authID, ts) VALUES(?name, ?id, ?ts)"
pname = com2.Parameters.Add("?name", MySqlDbType.VarChar)
pid = com2.Parameters.Add("?id", MySqlDbType.Int32)
pts = com2.Parameters.Add("?ts", MySqlDbType.Timestamp)
com2.Prepare()
' execute command for the first time
pname.Value = "test 124"
pid.Value = 124
pts.Value = DateTime.Now
com2.ExecuteNonQuery()
' execute command a second time
pname.Value = "test 125"
pid.Value = 125
pts.Value = DateTime.Now
com2.ExecuteNonQuery()

```

注意, 把 *NULL* 值传递给某个参数的工作必须使用 *DBNull.Value* 常数来进行, 如下所示:

```
com.Parameters(0).Value = DBNull.Value
```

与上面那段示例代码等价的 C# 代码如下所示:

```

// prepare command
MySqlCommand com2;
MySqlParameter pname, pid, pts;
com2 = myconn.CreateCommand();
com2.CommandText =
    "INSERT INTO authors (authName, authID, ts) VALUES(?name, ?id, ?ts)";
pname = com2.Parameters.Add("?name", MySqlDbType.VarChar);
pid = com2.Parameters.Add("?id", MySqlDbType.Int32);
pts = com2.Parameters.Add("?ts", MySqlDbType.Timestamp);
com2.Prepare();

// execute command for first time
pname.Value = "test 124";
pid.Value = 124;

```

```

pts.Value = DateTime.Now;
com2.ExecuteNonQuery();

// execute command for second time
pname.Value = "test 125";
pid.Value = 125;
pts.Value = DateTime.Now;
com2.ExecuteNonQuery();

```

20.2.3 处理离散的 SELECT 查询结果 (ExecuteScalar()方法)

像 *SELECT COUNT(*) FROM ...* 这样的查询命令属于一种特例情况：它们只返回一条结果记录，结果记录只有一个字段（数据列），字段的内容是一个离散值。对于这类 SQL 命令，使用 *MySqlDataReader* 对象去读取查询结果未免有些浪费资源。为了避免创建 *MySqlDataReader* 对象，可以采用把包含着这类命令的 *MySqlCommand* 对象传递给 *ExecuteScalar()* 方法的办法去执行它们，这个方法将返回一个 *object* 对象作为结果。在 VB.NET 程序里，可以用 *CInt()* 函数来提取 *object* 对象里的 *Integer* 值；如果 *object* 对象里包含的是字符串值，用 *CStr()* 函数来提取；日期/时间值可以用 *CDate()* 函数来提取等。

```

Dim n As Integer
Dim com As New MySqlCommand("SELECT COUNT(*) FROM table", myconn)
n = CInt(com.ExecuteScalar())

```

在 C# 程序里，需要使用诸如 *(int)*、*(string)* 或 *(date)* 之类的投射操作符来代替这些函数：

```

DateTime dt;
com = new MySqlCommand("SELECT MAX(ts) FROM authors ", myconn);
dt = (DateTime) com.ExecuteScalar();

```

在许多时候，往往需要在 C# 程序里使用两个投射操作符才能获得真正需要的结果：下面这个例子里的 *com.ExecuteScalar()* 调用将返回一个 *object* 对象，这个对象包含着一个 *long* 类型的值。如果想把这个值保存到一个 *int* 类型的变量里，就必须先把 *object* 转换为 *long*，再把 *long* 转换为 *int*。C# 语言把第一个投射操作符——它在下面这个例子里是 *(long)*——完成的工作形象地称为“脱壳”。

```

int id;
com = new MySqlCommand("SELECT LAST_INSERT_ID()");
id = (int)(long)com.ExecuteScalar();

```

20.2.4 读取 SELECT 查询结果 (*MySqlDataReader* 对象)

MySqlDataReader 类向程序员提供了一个读取 *SELECT* 查询结果的简单办法。*MySqlDataReader* 对象不支持在查询结果里的自由遍历（换句话说，这种对象只能用来向前遍历结果记录），不允许对结果记录进行修改。作为回报，*MySqlDataReader* 对象占用的资源比较少，对 *SELECT* 查询结果的访问效率比较高。

如果用 *ExecuteReader()* 方法去执行一条 SQL 命令，获得的就将是一个 *MySqlDataReader* 对象。此时，查询结果中的第一条及下一条结果记录需要用 *Read()* 方法去读取，根据是否读到一条记录，这个方法将返回 *True* 或 *False* 作为它本身的返回值。如果不再需要某个 *MySqlDataReader* 对象，应该尽早使用 *Close()* 方法把它释放掉。（否则，它将一直占用着内存和其他相关资源。）

在下面这段代码里，用了一个循环来遍历所有的结果记录。从结果记录里提取出来的数据先临时存放在一个 *StringWriter* 对象里，最后被输出到了一个控制台窗口：

```

Dim sstr As New IO.StringWriter
Dim com As New MySqlCommand(
    "SELECT publID, publName FROM publishers", myconn)
Dim dr As MySqlDataReader = com.ExecuteReader()
While dr.Read()
    sstr.WriteLine("id: {0} name: {1}", dr!publID, dr!publName)
End While
Console.WriteLine(sstr)
sstr.Close()
dr.Close()

```

MySqlDataReader 对象没有提供任何可以用来提前确定结果记录总数的办法。如果想知道这个数字是多少，就必须在读取所有的结果记录时进行计数或是另外进行一次 *SELECT COUNT(*) ...* 查询。

□ **访问结果记录的数据字段。** 在 VB .NET 程序里，对数据字段进行访问需要以 *dr!columnname* 的方式来进行。如此获得的结果是一个 *object* 对象，实际数据还需要使用 *CInt()*、*CStr()* 等函数来提取。

在 C# 程序里，访问数据字段的语法是 *dr["columnname"]*，从 *object* 对象里提取实际数据的办法是使用相应的投射操作符。

作为一种替代办法，还可以使用 *GetByte(n)*、*GetDateTime(n)* 和 *GetInt32(n)* 等函数来读取结果记录里的数据，这里的 *n* 是结果字段（数据列）的编号（第一个字段的编号是 0）。

□ **访问结果记录的元数据。** 可以用 *GetName(n)* 和 *GetDataTypeName(n)* 函数来获得各结果字段（数据列）的名字和数据类型。如果想知道某个查询命令是否真的返回了结果，对相应的 *dr.HasRows* 属性进行一下测试就可以找出答案。结果字段（数据列）的个数可以通过 *dr.FieldCount* 属性查知。

1. 需要特殊处理的数据类型

□ **NULL 值：** MySQL 数据库里的 *NULL* 值对应着 ADO .NET 中的 *System.DBNull* 常数。有许多办法可以用来测试当前结果字段是否包含 *NULL* 值：

```

If Convert.IsDBNull(dr!column) Then ...
If dr!column Is DBNull.Value Then ...
If dr.IsDBNull(columnnumber) Then ...

```

在使用 *SELECT* 查询结果的时候（比如像 *stringvar = CStr(dr!textcolumn)* 这样），一定要假设包含在数据字段里的值有可能是 *NULL* 并进行必要的检查和处理：*NULL* 会导致 *CStr()* 方法（VB .NET）或(*string*)投射操作符（C#）发生错误。

□ **日期：** MySQL 中的日期信息 (*DATE*、*TIMESTAMP*) 将自动转换为 ADO .NET 中的 *Date* 对象。

□ **时间：** MySQL 中的时间信息 (*TIME*) 将自动转换为 ADO .NET 中的 *TimeSpan* 对象。

□ **MySqlDateTime 对象：** 还可以利用 ADO .NET 中的 *MySqlDateTime* 对象直接对 MySQL 中的日期/时间数据进行处理，这种对象可以用 *MySqlDataReader* 类的 *GetMySqlDateTime()* 方法获得。日期/时间数据中的日期和时间部分可以用 *MySqlDateTime* 对象的 *Day*、*Hour* 等属性来提取：*IsValidDate* 属性可以告诉用户那是不是一个合法的日期/时间值。（根据它们的具体配置情况，有些 MySQL 服务器允许非法的日期/时间值进入数据库，比如像“31.2.2005”这样）。

□ **定点数：** MySQL 中的定点数 (*DECIMAL*) 将自动转换为 ADO .NET 中的 *Decimal* 对象。

□ **BIGINT：** 与 ADO 不同，ADO .NET 在处理 64 位整数 (*BIGINT*) 的时候不会发生任何问题。这类数值在 ADO .NET 里被表示为 *Long* 对象。

□ **BLOB：** 二进制数据在 ADO .NET 里被表示为 *Byte* 数组。

□ **字符串：** 在 VB .NET 和 C# 程序里，字符串统一采用 UTF16（即 UCS2）编码，这种 Unicode

格式使用两个字节来表示每一个字符。在 ADO .NET 与 MySQL 之间传输的字符串会在进、出 MySQL 数据表的时候被自动转换为正确的字符集编码。

2. 多组 SELECT 查询结果

如果一次执行了多条 SQL 命令 (*SELECT 1; SELECT 2; ...*) 或者如果使用了存储过程，就有可能得到多组 *SELECT* 查询结果。在默认的情况下，只能通过 *MySqlDataReader* 对象访问到第一组查询结果。如果想访问后面的某一组查询结果，必须先用 *NextResult()* 方法前进到那里；如果已经到达最后一组查询结果，这个方法将返回 *False*。

在下面这段代码里，用了一个 *Do* 循环来遍历多组查询结果，用了一个 *While* 循环来遍历每一组查询结果中的记录，用了一个 *For* 循环来遍历每一条结果记录中的字段（数据列），最后把数据全部以字符串的形式显示在了一个控制台窗口里：

```
Dim i As Integer
Dim com As New MySqlCommand(
    "SELECT * FROM authors LIMIT 3;SELECT COUNT(*) FROM authors;SELECT 1+2", _
    myconn)
Dim dr As MySqlDataReader = com.ExecuteReader()
Do
    Console.WriteLine("-----")
    While dr.Read()
        For i = 0 To dr.FieldCount - 1
            If dr.IsDBNull(i) Then
                Console.Write("NULL ")
            Else
                Console.Write(dr.GetString(i) + " ")
            End If
        Next
        Console.WriteLine()
    End While
    Loop While dr.NextResult()
    dr.Close()
```

20.2.5 DataSet、DataTable 和 MySqlDataAdapter 对象

如果想随心所欲地在 *SELECT* 查询结果里前后移动并在其中修改、删除和插入记录，将需要一个 *DataSet* 对象和至少一个包含在其中的 *DataTable* 对象。同一个 *DataSet* 对象可以用来容纳和管理一个或多个 *DataTable* 对象。*DataTable* 对象相当于一个充满着数据的数据表，这个数据表的内容就是 *SELECT* 命令的查询结果。*DataSet* 和 *DataTable* 都是 ADO .NET 里的类，与具体的数据泵无关。因此，不存在独立的 *MySqlDataSet*、*MySqlDataTable* 等对象。不过，为了让 *DataSet* 对象与某个特定的数据泵进行通信，就必须有一个相应的数据库连接类的对象，具体到 MySQL 数据库系统，这个任务就要由一个 *MySqlDataAdapter* 对象来承担。

容纳在一个 *DataSet* 对象里的数据全部驻留在 ADO .NET 程序的内存里。当因为执行了一条 *SELECT* 命令而在一个 *DataSet* 对象里创建出了一个 *DataTable* 对象的时候，这条命令的全部查询结果将一次性地被传输给那个 ADO .NET 程序。此后，对查询结果进行的所有操作都将作用于有关数据的本地复制——如果想对 MySQL 服务器上的原始数据进行刷新，就必须用 *MySqlDataAdapter* 对象调用 *Update()* 方法。

下面这段代码将把一条 *SELECT* 命令查询的结果填充到一个 *DataTable* 对象：先像往常一样为 SQL 命令创建一个 *MySqlCommand* 对象，然后把这个对象传递给一个 *MySqlDataAdapter* 对象。接下来，创建一个空白的 *DataSet* 对象，然后用 *MySqlDataAdapter* 对象的 *Fill()* 方法执行 *SELECT* 命令并把查询结果存入刚创建的 *DataSet* 对象。为了对查询结果进行处理，还需要在这个 *DataSet* 对象里创建一个

新的 *DataTable* 对象，它在这个例子里的名字是 *dtname*。现在，就可以在代码里通过 *ds.Tables("dtname")* 去访问这个 *DataTable* 对象了¹。

```
Dim com As New MySqlCommand("SELECT * FROM tablename", myconn)
Dim da As New MySqlDataAdapter(com)
Dim ds As New DataSet()           'currently empty DataSet ds
da.Fill(ds, " dtname")          'create DataTable "dtname" in ds
Dim dt As DataTable = ds.Tables("dtname")   'access to the DataTable
```

完成上述准备工作之后，现在就可以通过 *dt.Row(rowno)!columnname* 的形式去访问这个 *DataTable* 对象所容纳的数据了。在这里，*Row(n)* 将返回一条结果记录 (*DataRow* 对象)；*Rows.Count* 属性将返回结果记录的总个数。下面是一个可以用来遍历所有结果记录的循环结构：

```
Dim row As DataRow
For Each row In dt.Rows  'loop over all records
    row!columnname = ...  'change data
    Next
```

可以用 *DataRow.Delete()* 方法来删除当前结果记录。插入新记录的工作要稍微复杂一些：必须先用 *DataTable.NewRow()* 方法创建一条新记录，然后编辑它的内容，最后再用 *DataTable.Rows.Add()* 方法把它添加到 *DataTable* 对象里去，如下所示：

```
Dim newrow As DataRow
newrow = dt.NewRow()
newrow!a_big_int = 12345
dt.Rows.Add(newrow)
```

正如刚才提到的那样，对 *DataSet* 对象里的数据进行的所有修改都将作用于有关数据的本地复制（也就是只发生在当前 ADO .NET 程序的内存里），如果想对 MySQL 服务器上的原始数据进行刷新，就必须用 *MySqlDataAdapter* 对象调用 *Update()* 方法。这种刷新由一系列 *INSERT*、*UPDATE* 和 *DELETE* 命令负责具体完成，这些命令由 *MySqlDataAdapter* 对象发送给 MySQL 服务器去执行。可是，*MySqlDataAdapter* 对象不具备生成这些命令的能力——在调用 *Update()* 方法之前，必须先创建一个 *MySqlCommandBuilder* 对象来完成生成这项任务。下面这段代码演示了这一过程：

```
' ... Code to change data in dt
Dim cb As New MySqlCommandBuilder(da)
Try
    da.Update(ds, "test")
Catch e As Exception
    Stop
End Try
```

如果想知道 *MySqlCommandBuilder* 对象都生成了哪些 *INSERT*、*UPDATE* 和 *DELETE* 命令，可以用以下 3 个方法把它们查出来：*cb.GetInsertCommand()*、*cb.GetUpdateCommand()* 和 *cb.GetDeleteCommand()*。

以上这些理论上的好东西在实践中还需要遵守以下限制：

- 如果想把修改后的数据重新存入数据库，就必须保证同一个 *DataSet* 对象所包含的各个 *DataTable* 对象里的数据都来自同一个 MySQL 数据表。换句话说，只要某个 *DataTable* 对象里的数据是一条涉及多个数据表（使用了 *JOIN* 操作）或使用了 SQL 统计函数（包括 *GROUP BY* 子句）的 *SELECT* 命令的查询结果，就不能用修改后的数据（注意，对数据的本地复制进行修改还是允许的）去刷新 MySQL 服务器上的原始数据，这是因为 *MySqlCommandBuilder* 对象无法为这种情况生成准确无误的 SQL 命令。

1. 原文中的 *test* 在下面这段代码里根本就没有出现！*ds.Tables("name")* 也不如 *ds.Tables("dtname")* 更符合上下文。

——译者注

这条限制并不是 ADO .NET 本身的过错，而是因为涉及多个数据表的修改操作往往有一种以上的语义和语法解释，ADO .NET 和 *CommandBuilder* 基类目前的“智力”还无法做出最正确的选择。

- 通过 *DataSet* 对象去刷新数据库原始数据的效果与具体使用的是哪一种数据泵有着非常密切的关系。在笔者进行的测试中，Connector/J 的表现非常让人满意，ODBC 数据泵却经常出问题。由于这种密切的因果关系，代码在改用了另外一种数据泵之后不一定还能工作得像以前那样好；这一点希望大家特别注意。

提示 在理论上先进的技术不一定能在现实世界里拿过来就用，通过 *DataSet* 对象去修改数据库数据就是一个这样的例子。在决定采用这种解决方案之前，千万不要忘记进行全面和详尽的测试。一般来说，用 *DataSet* 对象去读取数据不会出什么问题。如果需要修改数据，最稳妥的办法还是自行构造必要的 SQL 命令、再用 *MySqlCommand* 对象去执行它们（尽量多使用参数）。

示例

在下面这段代码里，将先在一个 *DataSet* 对象里创建一个 *DataTable* 对象来存放查询命令 *SELECT * FROM author* 的结果，然后在接下来的 *For* 循环里把名字以 *test* 开头的作者全部删除掉：

```

Dim com As New MySqlCommand("SELECT * FROM authors", myconn)
Dim da As New MySqlDataAdapter(com)           'interface MySQL<-->DataSet
Dim cb As New MySqlCommandBuilder(da)          'necessary for updates
Dim ds As New DataSet()                      'currently empty DataSet
da.Fill(ds, "authors")                      'creates DataTable "authors" in ds
Dim dt As DataTable = ds.Tables("authors")    'access to the DataTable
Dim row As DataRow
For Each row In dt.Rows                       'loop over all records
    If LCase(Left(CStr(row!authName), 4)) = "test" Then   'alter data
        Console.WriteLine("delete author " & row!authName.ToString())
        row.Delete()
    End If
Next
Try
    da.Update(ds, "authors")
Catch e As Exception
    MsgBox(e.Message)
End Try

```

同样功能的 C# 代码如下所示：

```

MySqlCommand com;
MySqlDataAdapter da;
DataSet ds;
DataTable dt;
MySqlCommandBuilder cb;
com = new MySqlCommand("SELECT * FROM authors", myconn);
da = new MySqlDataAdapter(com);
cb = new MySqlCommandBuilder(da);
ds = new DataSet();
da.Fill(ds, "authors");
dt = ds.Tables["authors"];
foreach (DataRow row in dt.Rows) {
    if (row["authName"].ToString().Substring(0, 4).ToLower() == "test") {
        Console.WriteLine("delete author " + row["authName"].ToString());
        row.Delete();
    }
}

```

```

try {
    da.Update(ds, "authors");
}
catch (Exception e) {
    Console.WriteLine(e.Message);
}

```

20.2.6 辅助函数

MySqlHelper 类提供的一些方法可以帮助程序员更有效率地编写 Connector/Net 程序。

- *ExecuteDataRow()*: 这个方法将创建一条与 MySQL 服务器的连接, 执行一条 *SELECT* 命令, 返回 *SELECT* 查询结果中的第一条记录, 然后断开与 MySQL 服务器的连接。
- *ExecuteReader()*: 这个方法将创建一条与 MySQL 服务器的连接, 执行一条 *SELECT* 命令, 返回 *SELECT* 查询结果中的全部记录(一个 *DataReader* 对象), 然后断开与 MySQL 服务器的连接。
- *UpdateDataSet()*: 这个方法将创建一条与 MySQL 服务器的连接, 用 *DataSet* 对象里的修改去刷新 MySQL 服务器上的数据库原始数据, 然后断开与 MySQL 服务器的连接。
- *ExecuteScalar()*: 这个方法将执行一条 *SELECT* 命令并把它的离散值结果返回为一个 *Object* 对象。这个方法既可以使用一条现有的 MySQL 服务器连接, 也可以先创建一条新连接、等完成任务后再断开它。
- *ExecuteDataSet()*: 这个方法将执行一条 *SELECT* 命令并返回 *SELECT* 查询结果中的全部记录(一个 *DataSet* 对象)。类似于 *ExecuteScalar()*, 这个方法也是既可以用现有的 MySQL 服务器连接, 也可以先创建一条新连接、等完成任务后再断开它。
- *ExecuteNonQuery()*: 这个方法用来执行那些不会返回结果记录的 SQL 命令。类似于 *ExecuteScalar()*, 这个方法也是既可以用现有的 MySQL 服务器连接, 也可以先创建一条新连接、等完成任务后再断开它。

20.2.7 出错处理

如果是 ADO .NET 方法和属性在执行时发生错误, 就会触发一个异常并抛出一个 *Exceptions* 对象; 这类出错情况通常可以用一个 *Try-Catch* 语法结构来捕获。

如果是 Connector/Net 在执行 SQL 命令的过程中发生错误, 就会触发一个异常并抛出一个 *MySqlException* 对象, 这类出错情况通常可以用一个 *Try-Catch* 语法结构来捕获。如果是 *System.Data* 命名空间里的基类发生错误, 就会触发一个异常并抛出一个 *System.Data.Exception* 对象, 这类出错情况通常可以用一个 *Try-Catch* 语法结构来捕获。对于这两种出错情况, *Message* 属性可以提供更多关于错误原因的信息。

20.2.8 Windows.Form 和 ASP.NET 控件

.NET Framework 向 Windows 应用程序和因特网应用程序 (ASP.NET) 提供了许多可以绑定到 *DataSet* 对象上的控件。这些控件可以大大简化对 SQL 查询结果的处理和输出工作。

程序员朋友请记住这样一个基本原则: 如果只使用这些控件来显示数据而不使用它们去修改数据, 就可以为自己减少许多麻烦。万一必须向用户提供通过这些控件去修改数据的功能, 就一定要对它们做全面彻底的测试。(数据绑定控件几乎都是通过 *DataSet* 对象来实现其功能的, 而笔者刚才已经提到过通过 *DataSet* 对象去刷新数据库原始数据很容易出问题。)

因为篇幅的限制, 无法在这里对各种数据绑定控件和它们的用法做详细的介绍, 但希望下面这个示例程序可以让大家对它们的使用步骤和应用效果有一个直观的认识。这个示例程序将把 *mylibrary*

数据库收录的所有出版公司显示在一个下拉列表框里，当用户通过这个下拉列表框选中某个出版公司的时候，由这个出版公司出版的图书就会显示在一个文本输出表格框里（如图 20-3 所示）。

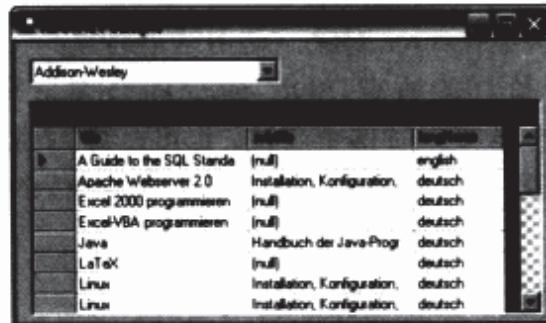


图 20-3 ADO .NET 示例程序

```
' example program vbnet\datacontrols
Imports MySql.Data.MySqlClient
Public Class Form1
    Inherits System.Windows.Forms.Form
    ' Windows Forms Designer generated code ...

    Dim myconn As MySqlConnection
    Dim titlecommand As MySqlCommand
    Dim publparam As MySqlParameter

    Private Sub Form1_Load(...) Handles MyBase.Load
        ' create connection to database
        myconn = New MySqlConnection(
            "Data Source=localhost;Initial Catalog=mylibrary;" +
            "User ID=root;PWD=uranus")
        myconn.Open()

        'prepare command and parameters for title query and Parameter
        titlecommand = New MySqlCommand(
            "SELECT title, subtitle, langName FROM titles, languages " +
            "WHERE publID = ?publ AND titles.langID = languages.langID " +
            "ORDER BY title", _
            myconn)
        publparam = titlecommand.Parameters.Add("?publ", MySqlDbType.Int32)
        titlecommand.Prepare()
        'fill ComboBox with a list of all publishers
        Dim com As New MySqlCommand(
            "SELECT publID, publName FROM publishers ORDER BY publName", _
            myconn)
        Dim da As New MySqlDataAdapter(com)
        Dim ds As New DataSet()
        da.Fill(ds, "publishers")
        ComboBox1.DisplayMember = "publName"
        ComboBox1.ValueMember = "publID"
        ComboBox1.DataSource = ds.Tables("publishers")
    End Sub

    ' nw publisher selected --> create new DataSet with titles
    Private Sub ComboBox1_SelectedIndexChanged(...) _
        Handles ComboBox1.SelectedIndexChanged
        publparam.Value = ComboBox1.SelectedValue
        Dim ds As New DataSet()
        Dim da As New MySqlDataAdapter(titlecommand)
        da.Fill(ds, "titles")
        DataGridView1.DataSource = ds.Tables("titles")
    End Sub
End Class
```

20.2.9 事务

在 ADO .NET 程序里，开始一个事务的办法是调用 *MySqlConnection* 对象的 *BeginTransaction()* 方法。这个方法将返回一个 *MySqlTransaction* 对象，调用这个对象的 *Commit()* 或 *Rollback()* 方法就可以结束这个事务。在事务过程中执行的所有 SQL 命令都必须带有 *Transaction* 属性，可以通过 *MySqlCommand* 构造器的第三个参数来设置这个属性，也可以等 *MySqlCommand* 对象创建出来之后再明确地设置它们的 *Transaction* 属性。下面这段代码演示了在 ADO .NET 程序里使用事务的整个过程：

```
Dim tr As MySqlTransaction
tr = myconn.BeginTransaction()
Dim com As New MySqlCommand("UPDATE ...", myconn, tr)
com.ExecuteNonQuery()
' ... additional commands of the transaction
tr.Commit()
```

20.3 示例：把新图书记录存入 *mylibrary* 数据库

本节示例程序将显示一个简单的输入对话框供人们输入新图书的名字以及新图书的作者姓名和出版公司名称（如图 20-4 所示）。在这个对话框里，作者姓名文本框允许一次输入多位作者的姓名，但它们之间要用分号隔开；出版公司文本框允许空着不填（也就是说，缺少出版公司名称不影响新图书被存入数据库）。在插入一条新图书记录的时候，示例程序会去检查那本新图书的作者和/或出版公司是否早已被收录在了数据库里。如果是，就使用现有的作者和/或出版公司信息只插入新图书。除此以外，这个示例程序没有再做其他的检查和出错处理。

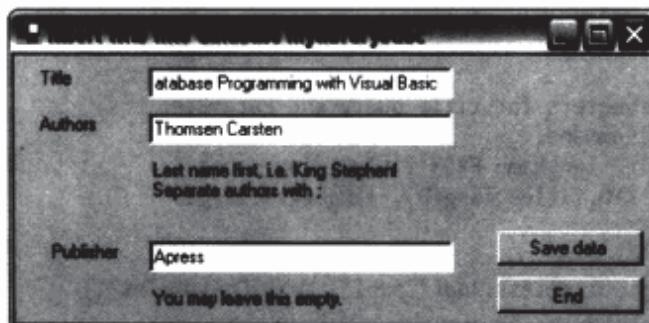


图 20-4 新图书输入表单

程序代码被分为两个过程。在 *Form1_Load* 过程中，用 *INSERT* 和 *SELECT* 命令选出一定数量的 *MySqlCommand* 对象。然后在 *btnSave_Click* 过程中调用这些命令存储输入：

```
' example vbnet\newtitleform
Option Strict On
Imports MySql.Data.MySqlClient
Public Class Form1
    Inherits System.Windows.Forms.Form
    ' Windows Forms Designer generated code ...
Dim mysqlConn As MySqlConnection
    Dim insertPublisherCom, insertAuthorCom,
        insertTitleCom, insertRelAuthTitleCom,
        selectPublisherCom, selectAuthorCom,
        lastIDCom As MySqlCommand

    ' initialization tasks
Private Sub Form1_Load(...) Handles MyBase.Load
    ' create connection to database
```

```

mysqlConn = New MySqlConnection(
    "Data Source=localhost;Initial Catalog=mylibrary;" + _
    "User ID=root;PWD=uranus")
mysqlConn.Open()

' command for storing an author
insertAuthorCom = New MySqlCommand(
    "INSERT INTO authors (authName) VALUES (?authName)", mysqlConn)
insertAuthorCom.Parameters.Add("?authName", MySqlDbType.VarChar)
insertAuthorCom.Prepare()

' command for author search
selectAuthorCom = New MySqlCommand(
    "SELECT authID FROM authors WHERE authName = ?authName LIMIT 1", _
    mysqlConn)
selectAuthorCom.Parameters.Add("?authName", MySqlDbType.VarChar)
selectAuthorCom.Prepare()

' command for storing a title
insertTitleCom = New MySqlCommand(
    "INSERT INTO titles (title, publID) VALUES (?title, ?publID)", _
    mysqlConn)
insertTitleCom.Parameters.Add("?title", MySqlDbType.VarChar)
insertTitleCom.Parameters.Add("?publID", MySqlDbType.Int32)
insertTitleCom.Prepare()

' command for storing a publisher
insertPublisherCom = New MySqlCommand(
    "INSERT INTO publishers (publName) VALUES (?publName)", mysqlConn)
insertPublisherCom.Parameters.Add("?publName", MySqlDbType.VarChar)
insertPublisherCom.Prepare()

' command for publisher search
selectPublisherCom = New MySqlCommand(
    "SELECT publID FROM publishers WHERE publName = ?publName LIMIT 1", _
    mysqlConn)
selectPublisherCom.Parameters.Add("?publName", MySqlDbType.VarChar)
selectPublisherCom.Prepare()

' command for storing rel_title_author entries
insertRelAuthTitleCom = New MySqlCommand(
    "INSERT INTO rel_title_author (titleID, authID) " + _
    "VALUES (?titleID, ?authID)", _
    mysqlConn)
insertRelAuthTitleCom.Parameters.Add("?titleID", MySqlDbType.Int32)
insertRelAuthTitleCom.Parameters.Add("?authID", MySqlDbType.Int32)
insertRelAuthTitleCom.Prepare()

' command for determining LAST_INSERT_ID
lastIDCom = New MySqlCommand(
    "SELECT LAST_INSERT_ID()", mysqlConn)
End Sub

Private Sub btnSave_Click(...) Handles btnSave.Click
    Dim publID, titleID As Integer
    Dim result As Object
    Dim author, authors() As String

    ' Test whether input complete
    txtTitle.Text = Trim(txtTitle.Text)
    txtAuthors.Text = Trim(txtAuthors.Text)
    txtPublisher.Text = Trim(txtPublisher.Text)
    If txtTitle.Text = "" Or txtAuthors.Text = "" Then
        MsgBox("Please specify title and authors!")
        Exit Sub

```

```

End If

' search for publisher or store new publisher
If txtPublisher.Text <> "" Then
    ' does publisher already exist?
    selectPublisherCom.Parameters("?publName").Value = txtPublisher.Text
    result = selectPublisherCom.ExecuteScalar()
    If result Is Nothing Then
        insertPublisherCom.Parameters("?publName").Value = txtPublisher.Text
        insertPublisherCom.ExecuteNonQuery()
        publID = CInt(lastIDCom.ExecuteScalar())
    Else
        publID = CInt(result)
    End If
End If
' store title
insertTitleCom.Parameters("?title").Value = txtTitle.Text
If publID > 0 Then
    insertTitleCom.Parameters("?publID").Value = publID
Else
    insertTitleCom.Parameters("?publID").Value = DBNull.Value
End If
insertTitleCom.ExecuteNonQuery()
titleID = CInt(lastIDCom.ExecuteScalar())

' store authors
authors = Split(txtAuthors.Text, ";")
insertRelAuthTitleCom.Parameters("?titleID").Value = titleID
For Each author In authors
    ' does the author exist already?
    selectAuthorCom.Parameters("?authName").Value = author
    result = selectAuthorCom.ExecuteScalar()
    If result Is Nothing Then
        ' no, store new author
        insertAuthorCom.Parameters("?authName").Value = author
        insertAuthorCom.ExecuteNonQuery()
        insertRelAuthTitleCom.Parameters("?authID") = _
            CInt(lastIDCom.ExecuteScalar())
    Else
        ' yes, determine authID
        insertRelAuthTitleCom.Parameters("?authID").Value = CInt(result)
    End If
    'store rel_title_authors entry
    insertRelAuthTitleCom.ExecuteNonQuery()
Next

MsgBox("Your input has been saved")
txtTitle.Text = ""
txtAuthors.Text = ""
txtPublisher.Text = ""
End Sub
End Class

```

20.4 示例：把图像文件存入和读出一个 BLOB 数据列

下面的示例程序从当前目录读出 test.jpg 文件的内容并把它存入一个数据表 (*btnSave_Click*)。在第二步，二进制数据又从数据表里读出并显示在一个 *PictureBox* 控件里（如图 20-5 所示）、再存入一个 test1.jpg 文件 (*btnLoad_Click*)。



图 20-5 示例：对图像和二进制数据进行处理

在把二进制数据写入和读出一个数据库的过程中，这个示例程序演示了各种各样的编程技巧。为了存放那幅图像，这个示例程序一开始先执行如下所示的 SQL 命令创建一个数据表，等程序即将结束时又删除它：

```
CREATE TABLE IF NOT EXISTS testpic
(id INT NOT NULL AUTO_INCREMENT, pic MEDIUMBLOB, PRIMARY KEY (id))
```

下面是程序代码：

```
' example code vbnet/blob_test
Option Strict On
Imports MySql.Data.MySqlClient
Imports System.IO
Public Class Form1
    Dim myconn As MySqlConnection
    Dim com As MySqlCommand
    Dim id As Integer

    ' create connection to MySQL server, create testpic table
    Private Sub btnConnect_Click(...) Handles btnConnect.Click
        Try
            myconn = New MySqlConnection(
                "Data Source=192.168.80.128;Initial Catalog=mylibrary;" +
                "User ID=root;PWD=uranus")
            myconn.Open()
            com = New MySqlCommand(
                "CREATE TABLE IF NOT EXISTS testpic " +
                "(id INT NOT NULL AUTO_INCREMENT, " +
                " pic MEDIUMBLOB, PRIMARY KEY (id))",
                myconn)
            com.ExecuteNonQuery()
            btnSave.Enabled = True
        Catch myerror As MySqlException
            MsgBox("Database connection error: " & myerror.Message)
            Me.Close()
        End Try
    End Sub
```

```

' load file test.jpg and store in the table testpic
Private Sub btnSave_Click(...) Handles btnSave.Click
    Dim fs As FileStream
    Dim bindata As Byte()
    Dim picpara As MySqlParameter

    ' prepare INSERT command
    com = New MySqlCommand(
        "INSERT INTO testpic (pic) VALUES(?pic)", myconn)
    picpara = com.Parameters.Add("?pic", MySqlDbType.MediumBlob)
    com.Prepare()

    ' read file into the byte field
    fs = New FileStream("test.jpg", FileMode.Open, FileAccess.Read)
    ReDim bindata(CInt(fs.Length))
    fs.Read(bindata, 0, CInt(fs.Length))
    fs.Close()

    ' execute INSERT, bindata as parameter
    picpara.Value = bindata
    com.ExecuteNonQuery()

    ' determine ID of the new record in testpic
    com.CommandText = "SELECT LAST_INSERT_ID()"
    id = CInt(com.ExecuteScalar())
    btnLoad.Enabled = True
End Sub

' read BLOB, store as file test1.jpg and display in PictureBox1
Private Sub btnLoad_Click(...) Handles btnLoad.Click
    Dim fs As FileStream
    Dim bindata As Byte()
    Dim ms As New MemoryStream

    ' execute SELECT command
    com = New MySqlCommand("SELECT pic FROM testpic WHERE id=" & id, myconn)
    bindata = CType(com.ExecuteScalar(), Byte())

    ' write Byte field in MemoryStream,
    ' create from this a new bitmap object
    ms.Write(bindata, 0, bindata.Length)
    PictureBox1.Image = New Bitmap(ms)

    ' write MemoryStream in FileStream as file test1.jpg
    fs = New FileStream("test1.jpg", FileMode.Create, FileAccess.Write)
    ms.WriteTo(fs)
    fs.Close()
End Sub

' delete testpic table
Private Sub btnEnd_Click(...) Handles btnEnd.Click
    com = New MySqlCommand(
        "DROP TABLE IF EXISTS testpic ", myconn)
    com.ExecuteNonQuery()
    Me.Close()
End Sub
End Class

```

Part 5

第五部分

参 考 资 料

本部分内容

- 第 21 章 SQL 语法指南
- 第 22 章 MySQL 工具和选项
- 第 23 章 MySQL API 应用指南

第21章

SQL 语法指南



本章全面介绍了 MySQL 5.0 版本的各种 SQL 操作符、函数及命令。之所以把这些内容安排在这里，是希望使你们——亲爱的读者——对一些最重要和最有用的语法的使用有一个完整的概念。

本章可绝对不是对 MySQL 在线文档的替代，后者不但提供了 MySQL 的源代码，而且对 MySQL 的使用者提供了最好、最完整和最新的用法指南。没有任何一本书能够取代这种在线指南。MySQL 在线文档提供的信息非常丰富，这既是它的最大优点，也是它的最大缺点：重要的命令和语法变体往往淹没在无数的细节当中，而那些细节有 90% 以上与正在寻找的答案没有什么关系。请访问以下网址 <http://dev.mysql.com/doc/mysql/en/index.html>。

注意，虽说 MySQL 的功能模块多得数不胜数，但在某些方面还未能达到 ANSI-SQL/92 标准的要求。

21.1 语法

先从比较简单的内容开始，本节介绍对象名、字符串、日期/时间及二进制数据。

21.1.1 对象命名规则

数据库、数据表、数据列等数据库对象的名字最多可以有 64 个字符的长度。允许使用的字符是 MySQL 使用的字符集当中的所有字母数字符号，以及符号“_”和“\$”。但从实际应用的角度出发，有必要把可使用的字母数字限制在 ASCII 字符中，再加上下划线符号。这样做有两个原因：

- 特殊字符的编码取决于字符集。如果客户与服务器使用的字符集不一致，有可能在访问对象的时候出现问题。
- 数据库和数据表的名字可以储存在文件中，但对文件的命名规则是由操作系统而不是 MySQL 数据库来决定的。这就有可能引起冲突（特别是当数据库需要在各个操作系统之间交换的时候）。

包含特殊字符和保留字的名字：通常情况下，对象名不允许与保留的 SQL 关键字（*select*、*from* 等）相同。而且，许多特殊字符不允许出现在名字里（“-”、“!”、“%”等）。不过，这两个方面的限制都可以通过把名字放入反单引号的方法来解决：

```
CREATE TABLE `special name` (`from` INT, `a!%` INT)
```

复合名字：不在当前数据库里的数据表的名字必须有数据库名作为前缀。同样，如果一个数据列的名字不能够提供唯一的标识，那么就必须增加上这个数据列所在数据表的名字，以及该数据列和数据表所在数据库的名字（比如需要在同一条查询命令里使用多个来自不同的数据表的同名数据列的时候）：

Table names: *tablename* 或者 *db.tablename*

Column names: *colname* 或者 *tblname.colname* 或者 *dbname.tblname.colname*

21.1.2 区分字母大小写

根据是否区分字母的大小写, MySQL 中的对象名可以划分为以下两大类:

□ **区分字母大小写**。数据库名(除了在 Windows 操作系统下)、数据表名(除了在 Windows 操作系统下)、假名以及 MySQL 4.1 版本开始的变量名。

□ **不区分字母大小写**。SQL 命令和函数、数据列名、索引名以及 MySQL 5.0 版本开始的变量名。

在 Windows 环境下, 因为这种操作系统不区分目录名和文件名里的字母大小写情况, 所以 MySQL 对数据库名和数据表名的字母大小写情况也不挑剔。不过, 如果需要在同一条 SQL 命令里多次写出同一个名字, 它们的字母大小写情况必须保持一致——像下面这样的命令是无法正常工作的: *SELECT * FROM authors WHERE Authors.authName = "xxx"*。

从 MySQL 4.0 版本开始, 在创建新的数据库和数据表的时候, Windows 下的 MySQL 只使用小写字母来起名字(不管它们在 *CREATE* 命令中是如何写出的)。这将简化数据库从 Windows 操作系统向 UNIX/Linux 操作系统的迁移。这种自动转换由选项 *lower_case_table_names* 控制, 这个选项在 Windows 环境下的默认设置是 1。

21.1.3 字符串

字符串两端的引号既可以是单引号, 也可以是双引号。在 MySQL 中, 下面的两种表达方法是一样的, 尽管使用单引号的表达方法符合 ANSI-SQL/92 标准。

```
'character string'  
"character string"
```

如果在字符串里有引号, 则要用以下的不同方式来表示:

"abc'abc"	即	abc'abc
"abc""abc"	即	abc"abc
"abc\'abc"	即	abc'abc
"abc\"abc"	即	abc"abc
'abc"abc'	即	abc"abc
'abc''abc'	即	abc'abc
'abc\"abc'	即	abc"abc
'abc\'abc'	即	abc'abc

在一个字符串中, 允许出现普通字符集提供的特殊字符。例如, 如果是在默认字符集为 ISO-8859-1 (*latin1*) 下工作, 则会出现 äöüß 等字符。但是, 有些特殊的字符必须要专门编码:

特殊字符在字符串里的正确写法

\0	0 字节 (代码 0)
\b	后退符 (代码 8)
\t	制表符 (代码 9)
\n	换行符 (代码 10)
\r	回车符 (代码 13)
\"	双引号 (代码 34)
\'	单引号 (代码 39)
\\\	反斜线符 (代码 92)

假如 x 不属于上面的特殊字符, 那么 \x 转义成字符 x。假使一个字符串作为 *BLOB* (二进制对象) 被储

存，0字符、单引号、双引号和反斜杠必须用下面的方式来表达：`\0`, `\'`, `\"`, `\.`

与使用反斜线字符来转义字符串或 *BLOB* 数据中的特殊字符的方法相比，使用十六进制记号来表示整个对象的办法要简单得多。MySQL 允许程序员在 SQL 命令里使用像下面这样给出任意长度的十六进制代码：

`0x4142434445464748494a.`

但是，MySQL 不能以这种格式返回 SQL 查询结果。（如果正在使用 PHP 进行编程，PHP 语言为这个用途提供了一个很方便的函数：`bin2hex`。）

提示 如果要把两个字符串连接在一起，那就要使用函数 *CONCAT()*。（来自其他SQL语言或程序设计语言的操作符“+”和“||”不支持这个功能。）总地来说，MySQL提供的字符串函数还是非常多的。

21.1.4 字符集和排序方式

- 字符集¹和排序方式（collation）可以分开设置。
- 每一个数据表，甚至是一个数据表中的每一列，都可以有它自己的字符集和它自己的排序方式。
- Unicode 是目前最常见的字符集设置（分 UTF8 和 UCS2=UTF16 两种格式）。

在 SQL 命令里，可以对每一个字符串定义字符集，使用函数 *CONVERT()*或是操作符 *_charset* 可以实现这一点。下面是两个相同的例子，给出的是在 UTF8 格式下的内部 Unicode 编码：

```
SELECT HEX(CONVERT('ABCäöü' USING utf8))
        414243C3A4C3B6C3BC
SELECT HEX(_utf8 'ABCäöü')
        414243C3A4C3B6C3BC
```

在许多时候，有必要同时定义字符集和排序方式。（这就决定了哪些字符被认为是相同的以及字符串是如何排序的。）语法 *_charset 'abc' COLLATE collname* 可以完成它。请看下面的例子：

```
SELECT _latin1 'a' = _latin1 'ä'
      0
SELECT _latin1 'a' COLLATE latin1_german1_ci =
      _latin1 'ä' COLLATE latin1_german1_ci
      1
```

21.1.5 数值

十进制的数值表示方法可以有小数点，但在千位数上没有逗号分隔符（如 27345 或是 2.71828）。也可以使用科学计数法（6.0225e23 或 6.626e-34）来表示一些非常大或非常小的数值。

MySQL 还能够处理带有前缀 `0x` 或是 `x'1234` 格式的十六进制数值。根据上下文，这种数值将被解释为字符串或 64 位整数。

```
SELECT 0x142434445464748494a, x'142434445464748494a'
          ABCDEFGHIJ           ABCDEFGHIJ
SELECT 0x41 + 0
          66
```

21.1.6 数值和字符串的自动转换

在对两种不同的数据类型进行操作的时候，MySQL 会尽可能地把它们转换为同一种数据类型。如果两个操作数²中有一个是浮点数，那么整数就将自动转换为浮点数。如果在处理字符串的过程中涉及到计算，

1. 原文这里显然把“字符集”误写成了“字符串”。——译者注

2. 原文这里显然把“操作数”误写成了“操作符”。——译者注

则字符串将被自动转换为数值。（如果字符串的开头部分不能被解释为一个数值，那么 MySQL 在计算时将把它视为 0。）

```
SELECT '3.14abc' + 1
      4.14
```

21.1.7 日期和时间

MySQL 以 2005-12-31 的字符串格式表示日期，以 23:59:59 的字符串格式表示时间。数据类型 *DATETIME* 的表示方法是把这两种格式合并起来，如 2005-12-31 23:59:59。

```
USE exceptions
SELECT * FROM test_date
id  a_date      a_time      a_datetime           a_timestamp
1   2005-12-07  09:06:29  2005-12-07 09:06:29  2005-12-07 09:06:29
```

注意 从 MySQL 4.1 版本开始，*TIMESTAMP* 值的默认格式有所改变：从服务器返回的 *TIMESTAMP* 值的默认格式是 YYYY-MM-DD HH:MM:DD。在 MySQL 4.0 及以前的版本里，*TIMESTAMP* 返回值的格式是 YYYYMMDDHHMMDD。如果想使用旧格式，就需要给日期/时间值加上一个 0（*SELECT ts+0 FROM table*）。

MySQL 5.0 版本有一个很重要的变化。在 *DATE* 和 *DATETIME* 数据列中，只有有效的日期才是可以被接受的。（MySQL 老版本的数据合法性检查相当粗糙：它可以识别出字符串 2005-02-45 是一个非法的日期值，但会认为字符串 2005-02-31 是一个合法的日期值。）日期字符串 '0000-00-00' 是一个特例；在 MySQL 中，这种表示是允许的。

在存储日期/时间值的时候，MySQL 非常灵活：无论是数值（如 20051231）还是字符串都可以接受。在字符串中，允许出现连字符，没有也可以。如果用户给出的年份没有明确是哪一个世纪（只有两位数），MySQL 会自动使用年代范围在 1979~2069 中的年份。因此，下面这些字符串都可以被 MySQL 接受为 *DATETIME* 数据列的值：'2005 12 31'、'20051231'、'2005.12.31' 和 '2005&12&31'。

21.1.8 二进制数据

来自 *BLOB* 字段的二进制数据在 SQL 命令里是作为字符串来处理的（但在排序时会区别对待）。

21.1.9 二进制数值

从 5.0.3 版本开始，MySQL 支持数据类型 *BIT*。二进制数值可以写成 b'110010' 的格式。

21.1.10 注释语句

在 SQL 命令里写出注释的办法有 3 种：

```
SELECT 1  # comment
SELECT 1  /* comment */
SELECT 1  -- comment
```

以 “#” 或 “--”（注意，“--” 的后面至少要有一个空格）开头的注释语句持续到这一行的末尾。“/*” 和 “*/” 之间的注释语句可以跨越多个语句行，就像在 C 语言程序里一样。注释不允许嵌套。

如果在 SQL 代码使用了一些 MySQL 独有的命令，但又希望尽可能地与其他 SQL “方言” 保持相容，就需要使用如下所示的特殊语法：

```
SELECT /*! STRAIGHT_JOIN */ col FROM table ...
```

上面的命令用到了一个 MySQL 对 *SELECT* 命令的独有扩展——只有 MySQL 知道如何解释 *STRAIGHT_JOIN* 关键字。但因为使用 “*/* ... */*” 语法，所以其他的 SQL 语法都会把它视为一个注释。

类似的语法还可以用在各种各样的 MySQL 命令里，例如：

```
CREATE /*!32302 TEMPORARY */ TABLE ...
```

上面该命令里的关键字 *TEMPORARY* 只有 MySQL 3.23.02 或更高版本才会去解释。

21.1.11 SQL 命令末尾的分号

ANSI-SQL 和 MySQL 的 SQL 语法中都不允许分号出现在一条指令的末端。在执行单条命令的时候，MySQL 将遵循这条语法规则。但在以下几种场合，MySQL 将要求 SQL 命令必须以分号结束：

- 如果是在使用 MySQL 自带的命令解释器（即 *mysql* 程序）来执行 SQL 命令，必须使用分号来结束它们。
- 如果是在定义一个存储过程或触发器，必须使用分号把命令分隔开。
- 从 MySQL 4.1 版本开始，客户库允许一次执行多条 SQL 命令，此时需要用分号隔开各条命令。PHP 语言为 *mysqli* 接口提供了 *multi_query()* 方法。对其他的 API 语言，*MULTI_STATEMENT* 方式需要被明确地激活，例如在 C 语言当中，就要使用 *mysql_real_connect(..., CLIENT_MULTI_STATEMENTS)* 函数。

21.2 操作符

MySQL 操作符	
	算术操作符
+ - * /	基本算术运算
%	求余（整数除法的余数）
<i>DIV</i>	另一种整数除法操作符（自 MySQL 4.1 版本开始）
<i>MOD</i>	另一种整数求余操作符（自 MySQL 4.1 版本开始）
	位操作符
	二进制 OR（或）
&	二进制 AND（与）
~	二进制求反（翻转所有 0 位或 1 位）
<<	左移所有的二进制位（相当于乘以 2 ⁿ ）
>>	右移所有的二进制位（相当于除以 2 ⁿ ）
	比较操作符
=	“等于” 比较操作符
<=>	“等于” 比较操作符，允许操作数为 <i>NULL</i> 值
!=<>	“不等于” 比较操作符
<> <= >=	比较操作符：小于、大于、小于或等于、大于或等于
<i>IS [NOT] NULL</i>	<i>NULL</i> 比较操作符
<i>BETWEEN</i>	区间比较操作符
<i>IN</i>	集合比较操作符（如， <i>x IN (1,2,3)</i> 或 <i>x IN ('a','b','c')</i> ）
<i>NOT IN</i>	集合比较操作符（如， <i>x NOT IN ('a','b','c')</i> ）

(续)

MySQL 操作符	
模式匹配操作符	
<i>[NOT] LIKE</i>	简单模式匹配比较操作符（如， <i>x LIKE 'm%'</i> ）
<i>[NOT] REGEXP</i>	扩展模式匹配比较操作符（如， <i>x REGEXP '^x\$'</i> ）
<i>SOUNDS LIKE</i>	对应于 <i>SOUNDEX(a) = SOUNDEX(b)</i> ，自 MySQL 4.1 版本开始
二进制比较操作符	
<i>BINARY</i>	把操作数临时投射为一个二进制数值（如， <i>BINARY x=y</i> ）
逻辑操作符	
<i>!, NOT</i>	逻辑非操作符
<i> , OR</i>	逻辑或操作符
<i>&&, AND</i>	逻辑与操作符
<i>XOR</i>	逻辑异或操作符（自 MySQL 4.1 版本开始）
投射操作符（自 MySQL 4.1 版本开始）	
<i>_charset'abc'</i>	把字符串'abc'投射（临时转换）为 <i>charset</i> 字符集
<i>_charset'abc'COLLATE col</i>	把字符串'abc'投射（临时转换）为 <i>charset</i> 字符集和 <i>col</i> 排序方式

21.2.1 算术操作符与位操作符

如果算术操作符的某个操作数是 *NULL*，则运算结果将是 *NULL*。此外，在 MySQL 数据库系统里，以 0 作为除数的结果也将是 *NULL*（注意：这与许多其他的 SQL “方言”的做法不一样）。

21.2.2 比较操作符

一般来说，比较操作符的返回值是 1（相当于 *TRUE*）或 0（相当于 *FALSE*），与 *NULL* 值进行比较的结果还是 *NULL*。但操作符“*<=>*”和“*IS NULL*”是两个特例，它们在操作数为 *NULL* 时的返回值是 0 或 1。

```
SELECT NULL=NULL, NULL=0
      NULL, NULL
SELECT NULL<=>NULL, NULL<=>0
      1, 0
SELECT NULL IS NULL, NULL IS 0
      1, 0
```

在使用`<`、`<=`、`>`和`=>`以及 *BETWEEN*（当然，还包括所有其他的排序类操作符）等操作符进行的字符串比较操作中，参加比较的数据列的字符集和排序方式将起作用。如果操作数是用引号括起来的字符串，还必须明确地给出它们的字符集和排序方式。MySQL 不能对使用不同字符集的字符串进行比较。

21.2.3 使用 *LIKE* 操作符进行模式匹配

MySQL 为模式匹配操作提供了两个操作符。相对比较简单并且与 ANSI 标准相兼容的是 *LIKE* 操作符。与普通的字符串比较操作一样，*LIKE* 操作符也不区分字母的大小写情况。此外，*LIKE* 操作符支持两种通配符。

LIKE 搜索模式	
<code>_</code>	能与任何一个单个的字符相匹配
<code>%</code>	能与除 <i>NULL</i> 以外的任意长度的字符（包括空字符）序列相匹配
<code>_</code>	下划线字符（ <code>_</code> ）
<code>\%</code>	百分比符号（ <code>%</code> ）

21.2.4 使用 REGEXP 操作符进行模式匹配

REGEXP 操作符以及它的同义操作符 *RLIKE* 可以使用更多的通配符来构造搜索模式。这两种操作符的模式匹配语法比较复杂，但与 *UNIX* 命令 *grep* 和 *sed* 使用的语法非常相似。

REGEXP 搜索模式	
模式的定义	
<i>abc</i>	字符串 <i>abc</i>
<i>(abc)</i>	字符串 <i>abc</i> (构成一个组)
<i>[abc]</i>	字符 <i>a</i> 、 <i>b</i> 、 <i>c</i> 中的某一个
<i>[a-z]</i>	字符 <i>a</i> 到 <i>z</i> 中的某一个
<i>[^abc]</i>	不是这些字符 (可以是其他字符)
.	任意字符
模式的重复匹配次数	
<i>x</i>	表达式 <i>x</i> 必须出现一次
<i>x y</i>	表达式 <i>x</i> 或 <i>y</i> 必须出现一次
<i>x?</i>	表达式 <i>x</i> 可能出现一次 (或者一次也没有)
<i>x*</i>	表达式 <i>x</i> 可能出现任意多次 (或者一次也没有)
<i>x+</i>	表达式 <i>x</i> 可能出现任意多次, 至少要有一次
<i>x {n}</i>	表达式 <i>x</i> 必须出现 <i>n</i> 次
<i>x {,n}</i>	表达式 <i>x</i> 最多出现 <i>n</i> 次
<i>x {n,}</i>	表达式 <i>x</i> 至少出现 <i>n</i> 次
<i>x {n,m}</i>	表达式 <i>x</i> 至少出现 <i>n</i> 次, 最多出现 <i>m</i> 次
<i>^</i>	匹配字符串的开头
<i>\$</i>	匹配字符串的结尾
<i>\x</i>	特殊字符 <i>x</i> (如: 用 <i>\\$</i> 来表示 <i>\$</i>)

与 *LIKE* 操作符的情况一样, *REGEXP* 操作符也不区分字母的大小写。但注意, 只要能在字符串里找到搜索模式, *REGEXP* 操作符就算匹配成功。也就是说, 匹配模式并不需要与整个字符串完全匹配, 只要能与被比较字符串的一个片段相匹配即可。如果的确想要整个字符串匹配, 就要在搜索模式里用上“*^*”和“*\$*”通配符。

提示 上面的表格只给出了 *REGEXP* 模式的最重要元素。完整的描述可以用 *UNIX/Linux* 操作系统的 *man 7 regex* 命令查到, 也可以在因特网上查到, 例如, 访问 <http://linux.ctyme.com/man/alpha7.htm> 的站点就可以得到。

21.2.5 二进制字符串比较

在默认的情况下, 字符串的比较不考虑字母的大小写。因此, 条件表达式 '*a*'='A' 的返回值是 1 (真)。如果要执行一个二进制的比较, 就必须在某个操作数前放上一个 *BINARY* 关键字。*BINARY* 是一个投射操作符, 它可以临时改变某个操作数的数据类型 (在这个例子里面, 它将把一个数值或字符串投射为一个二进制对象)。*BINARY* 既可以在普通的字符串比较操作中使用, 也可以在使用了 *LIKE* 和 *REGEXP* 操作符的模式匹配操作中使用。

```
SELECT 'a'='A', BINARY 'a' = 'A', 'a' = BINARY 'A'
      1, 0, 0
```

21.2.6 逻辑操作符

逻辑操作符的返回值是 0（假）、1（真）或 *NULL*（当有一个操作数是 *NULL* 的时候）。*NOT* 操作符也不例外；也就是说，*NOT NULL* 的返回值仍旧为 *NULL*。

21.3 变量和常数

MySQL 支持的变量类型有很多：

- **普通变量（用户级变量）**。这种变量用一个@字符作为前缀，在 MySQL 会话末端结束其定义。
 - **系统变量和服务器变量**。这些变量包含了 MySQL 服务器的状态或属性。它们有两个前导的@符号作为标志（例如，@@binlog_cache_size）。
- 许多系统变量以两种版本存在：一个特指当前连接（如，@@session.wait_timeout），而另外一个用于 MySQL 服务器（如，@@global.wait_timeout，具有对于这个变量的默认值）。
- **结构化变量**。这些是系统变量的一些特例。MySQL 目前只在需要定义更多的 MyISAM 索引缓存区时才会用到这些变量。
 - **存储过程里的本地变量和参数**。这些变量处于存储过程中，而且只是在存储过程中有效。它们没有特殊的前导标志，因此，给它们起的名字必须与数据表和数据列的名字有所区别。

在 MySQL 4.1 及以前的版本里，MySQL 变量名一直区分字母大小写。但从 MySQL 5.0 版本开始，不再区分字母的大小写。因此，@name、@Name 以及@NAME 都表示同一个变量。

21.3.1 变量赋值

下面的例子给出了为变量赋值的几个语法。请注意，*SET* 命令使用的赋值操作符是“=”，*SELECT* 命令使用的赋值操作符是“:=”。最后一个用法是把一条数据记录的多个字段（数据列）分别赋值给几个变量，这种用法只能在 MySQL 5.0 或更高的版本里使用，而且仅适用于 *SELECT* 命令的返回结果只有一条记录的情况。

```
SET @varname = 3
SELECT @varname := 3
SELECT @varname := COUNT(*) FROM tabelle
SELECT COUNT(*) FROM tabelle INTO @varname
SELECT title, subtitle FROM titles WHERE titleID=... INTO @t, @st
```

21.3.2 使用和查看变量

许多变量都可以使用 *SELECT* 命令来查看：

```
SELECT @varname
      3
SELECT @@binlog_cache_size
@@binlog_cache_size
      32768
```

除了 *SELECT* 命令，还可以使用 *SHOW VARIABLES* 命令来查看系统变量。该命令的优点在于它可以同时显示所有的变量（注意，结果清单里的变量名都没有@@前导标志）。

Variable_name	Value
back_log	50

```

basedir          C:\Programs\MySQL\MySQL Server 5.0
binlog_cache_size 32768
bulk_insert_buffer_size 8388608

```

特别需要注意的是，有些系统变量只能使用 *SELECT* 命令来查看（例如，`@@autocommit`）或只能使用 *SHOW VARIABLES* 命令来查看（例如，`system_time-zone`）。

21.3.3 全局级系统变量与会话级系统变量

MySQL 中的系统变量分为 *SESSION*（会话）和 *GLOBAL*（全局）两大部分。*SESSION* 变量只对当前会话（连接级别）有效，而 *GLOBAL* 变量则对整个服务器全局有效。

```

SELECT @@wait_timeout      -- 会话（连接级别）
SELECT @@session.wait_timeout -- 会话（连接级别）
SELECT @@global.wait_timeout   -- 全局

```

系统变量也允许修改。根据变化是仅对当前连接有效还是对整个服务器有效，可以选用如下所示的不同语法。请注意，在使用 *SET* 命令的时候，可以省略两个`@@`标示符。但不能使用 *SELECT* 命令去改变系统变量。

```

SET @wait_timeout = 10000      -- 会话（连接级别）
SET @@session.wait_timeout = 10000 -- 会话（连接级别）
SET SESSION wait_timeout = 10000 -- 会话（连接级别）
SET @@global.wait_timeout = 10000 -- 全局
SET GLOBAL wait_timeout = 10000 -- 全局

```

拥有 *Super* 权限的用户有权对全局级别的变量进行修改。当一个全局变量被改变时，新的值对所有新的连接有效，但是对已经存在的连接无效。

从另外的一个方面来说，*SESSION* 变量的改变只是对当前的连接有效。当一个新的连接出现的时候，整体变量的默认值在起作用。

提示 本书没有涵盖所有的 MySQL 系统变量。可以在 MySQL 文档中找到有关 *GLOBAL* 和 *SESSION* 变量的完整清单。MySQL 文档有关内容里的关键字 *LOCAL* 与这里所说的 *SESSION* 有着完全一样的含义：<http://dev.mysql.com/doc/mysql/en/system-variables.html>。

有关变量内容的进一步说明可以在以下网址处的 MySQL 文档中找到：

<http://dev.mysql.com/doc/mysql/en/server-parameter.html>;
<http://dev.mysql.com/doc/mysql/en/set-option.html>;
<http://dev.mysql.com/doc/mysql/en/show-variables.html>.

21.3.4 SET PASSWORD 命令

SET 命令也可以用来改变连接密码。但是，*PASSWORD* 本身可不是一个变量！

```

SET PASSWORD = PASSWORD('xxx')
SET PASSWORD FOR user@hostname = PASSWORD('xxx')

```

SET 命令还有一些特殊的格式，在本章后面内容里的 *SET* 标题下再做进一步的论述。

21.3.5 结构化变量

许多系统变量有不止一个实例。MySQL 文档把这些实例称为结构化变量（structured variables）。它们以 *instancename.variablename* 的形式出现。事实上，有成组的变量与这种实例有关。（如果用面向对象的程序设计术语来说，它们相当于对象和对象的属性。）

MySQL 目前只有一组结构化变量，它们控制着 MySQL 中的 MyISAM 索引缓存区：*key_buffer_size*、*key_cache_block_size*、*key_cache_division_limit* 和 *key_cache_age_threshold*。MyISAM 索引缓存区是一个用来临时存放 MyISAM 数据表的索引的内存块，这 4 个变量决定着这个内存块的大小和管理模式。

MySQL 服务器在启动后会自动创建一个这种缓存区对象的实例，这个缓冲区实例的名字是 *default*。可以用 *SET@@default.key_buffer_size=...* 命令来设置这个默认缓存区的长度。命令 *SET@@key_buffer_size=...* 自动指向这个默认的 *default* 实例。

使用下面的命令，可以再创建一个 MyISAM 索引缓存区：

```
SET @@mycache.key_buffer_size = n
```

现在，*mycache* 是 MyISAM 索引缓存区的一个新实例。使用 *SET @@mycache.key_cache_xxx* 命令，可以对这个新 MyISAM 索引缓存区的属性进行设置。然后，使用命令 *CACHE INDEX*，可以把某一个索引赋值给这个新缓存区。如果不想使用这个缓存区，把它的长度设置为 0 即可：

```
SET @@mycache.key_buffer_size = 0
```

提示 只在极少数需要特别追求速度的场合才有必要创建多个 MyISAM 索引缓存区。进一步的资料可以通过下面的网址获得：

<http://dev.mysql.com/doc/mysql/en/myisam-key-cache.html>;
<http://dev.mysql.com/doc/mysql/en/multiple-key-caches.html>;
<http://dev.mysql.com/doc/mysql/en/structure-system-variables.html>.

21.3.6 常数

从 MySQL 4.0 版本开始，MySQL 可以识别 *TRUE* (1) 和 *FALSE* (0) 两个常数。

21.4 MySQL 数据类型

MySQL 数据类型

整 数	
<i>TINYINT(m)</i>	8 位整数 (1 字节)；可选参数 <i>m</i> 负责设置 <i>SELECT</i> 查询结果里的数据列宽度 (最大显示宽度)，但对数值本身的取值范围没有影响
<i>SMALLINT(m)</i>	16 位整数 (2 字节)
<i>MEDIUMINT(m)</i>	24 位整数 (3 字节)
<i>INT(m), INTEGER(m)</i>	32 位整数 (4 字节)
<i>BIGINT(m)</i>	64 位整数 (8 字节)
浮 点 数	
<i>FLOAT(m,d)</i>	浮点数，8 位精度 (4 字节)。参数 <i>m</i> 和 <i>d</i> 都是可选的， <i>m</i> 是浮点数在显示时的总位数、 <i>d</i> 是浮点数在显示时小数点后的位数。在插入数据的时候，数值被相应地四舍五入。超过最大可取值的数值将被替换为最大可取值。参数 <i>m</i> 和 <i>d</i> 的效果是这样的：如果往一个 <i>FLOAT(5,2)</i> 数据列插入浮点数 1234.5678，MySQL 将发出一个警告说格式不匹配 (这种警告消息可以用 <i>SHOW WARNINGS</i> 去查看)，再去显示这个数值时只能看到 999.99 (但存储和计算时仍使用 1234.5678) ¹
<i>DOUBLE(m,d)</i>	浮点数，16 位精度 (8 字节)

1. 原书这里把参数 *m* 和 *d* 的作用说乱了！它们不改变数值本身，只影响显示/查看效果！——译者注

(续)

MySQL 数据类型	
<i>REAL(m,d)</i>	与 <i>DOUBLE</i> 同义
<i>DECIMAL(p,s)</i>	两个参数 <i>p</i> 和 <i>s</i> 规定了全部的位数（精度，最大 65）以及小数点后的位数（范围，最大 30）。MySQL 以二进制格式保存数值。用来保存 <i>DECIMAL</i> 值的字节分小数点前和小数点后两部分计数：先为小数点前和小数点后的二进制数字各分配 4 个字节，这 4 个字节最多可以容纳 9 位十进制数字；如果还有多出来的十进制数字，就再按每次增加一个字节的办法进行分配（每个新增字节容纳 2 个十进制数字，但每 9 个十进制数字容纳在 4 个字节里）
<i>NUMERIC, DEC</i>	与 <i>DECIMAL</i> 同义
	日期/时间
<i>DATE</i>	日期格式为 "2005-12-31"，范围是 1000-01-01 到 9999-12-31 (3 字节)
<i>TIME</i>	时间格式为 "23:59:59"，范围是 +/−838:59:59 (3 字节)
<i>DATETIME</i>	把 <i>DATE</i> 和 <i>TIME</i> 结合起来一起表示，格式为 "2005-12-31 23:59:59" (8 字节)
<i>YEAR</i>	年代范围是 1900~2155 (1 字节)
<i>TIMESTAMP(m)</i>	日期和时间格式 20051231235959 表示的时间在 1970 年和 2038 年之间；选项 <i>m</i> 的值决定了 <i>SELECT</i> 取值结果的位数；例如，设置 <i>m</i> =8，那就只有年、月、日可以显示出来
	字符串
<i>CHAR(n)</i>	固定长度的字符串，最多不超过 255 个字符
<i>NATIONAL CHAR(n)</i>	Unicode 字符串（对应于 <i>CHAR(n) CHARSET utf8</i> 或 <i>NCHAR(n)</i> ）
<i>VARCHAR(n)</i>	可变长度的字符串；在 MySQL 4.1 及以前的版本里，最多 255 个字符。从 MySQL 5.0.3 版本开始，MyISAM 数据表里的 <i>VARCHAR</i> 数据列的最大长度是 65 535 个字节，但最大字符串长度还要取决于具体使用的字符集
<i>NATIONAL VARCHAR(n)</i>	可变长度的 Unicode 字符串（对应于 <i>NCHAR VARCHAR(n), VARCHAR(n) CHARSET utf8</i> ）
<i>TINYTEXT</i>	可变长度的字符串，最大为 255 个字符
<i>TEXT</i>	可变长度的字符串，最大为 $(2^{16}-1)$ 个字符
<i>MEDIUMTEXT</i>	可变长度的字符串，最大为 $(2^{24}-1)$ 个字符
<i>LONGTEXT</i>	可变长度的字符串，最大为 $(2^{32}-1)$ 个字符
	二进制数据
<i>TINYBLOB</i>	可变长度的二进制数据，最大为 255 个字节
<i>BLOB</i>	可变长度的二进制数据，最大为 $(2^{16}-1)$ 个字节
<i>MEDIUMBLOB</i>	可变长度的二进制数据，最大为 $(2^{24}-1)$ 个字节
<i>LONGBLOB</i>	可变长度的二进制数据，最大为 $(2^{32}-1)$ 个字节
	几何数据（从 MySQL 4.1 版本开始）
<i>GOMETRY</i>	一个通用的几何对象；细分的几何类型在本章后面的表格里列出
	其他数据类型
<i>ENUM</i>	集合，最多包含 65 535 个字符串（1 或 2 字节）作为集合元素
<i>SET</i>	集合，最多包含 255 个字符串（1 到 8 字节）作为集合元素
<i>BIT</i>	二进制位（从 MySQL 5.0.3 版本开始）
<i>BOOL</i>	<i>TINYINT(1)</i> 的同义词

在数据列的声明定义里（*CREATE TABLE, ALTER TABLE*），不同的数据列可以使用不同的选项。下表对这些选项进行了汇总。请注意，不是所有的选项都适用于所有的数据类型。

MySQL 数据类型的属性 (选项)

<i>NULL</i>	允许这个数据列包含 <i>NULL</i> 值; 这是默认设置
<i>NOT NULL</i>	不允许这个数据列包含 <i>NULL</i> 值
<i>DEFAULT</i> <i>xxx</i>	如果没有给出其他的输入值, 就使用默认值 <i>xxx</i> 。如果没有明确地设置一个默认值, 那么 MySQL 会自做主张地根据不同情况把 <i>NULL</i> (如果这个数据列允许包含 <i>NULL</i> 值的话) 或 0(数值字段) 或空字符串(<i>VARCHAR</i> 字段) 或 0000-00-00(日期) 或 0000(年) 或集合中的第一个元素(<i>ENUM</i> 字段) 设置为默认值
<i>DEFAULT CURRENT_TIMESTAMP</i>	对 <i>TIMESTAMP</i> 数据列起作用: 当新的记录插入的时候, 当前时间被自动保存
<i>ON UPDATE CURRENT_TIMESTAMP</i>	对 <i>TIMESTAMP</i> 数据列起作用: 当有变化(<i>UPDATE</i>)的时候, 当前时间被自动保存
<i>PRIMARY KEY</i>	把这个数据列定义为主键
<i>AUTO_INCREMENT</i>	导致自动递增这个数据列中插入的数值; 仅适用于单个整数数据列, 而且必须同时给出 <i>NOT NULL</i> 和 <i>PRIMARY KEY</i> 选项(如果这个数据列不是 <i>PRIMARY KEY</i> , 就必须为它定义一个 <i>UNIQUE</i> 索引)
<i>UNSIGNED</i>	无符号整数。注意, 无符号整数的计算结果仍将是无符号整数
<i>ZEROFILL</i>	在 <i>SELECT</i> 查询结果中的整数左侧用 0 来填充以达到长度要求(因此五位数就像下面这样 00123、01234 表示)
<i>BINARY</i>	对于 <i>CHAR</i> 和 <i>VARCHAR</i> 数据列, 比较和排序操作以二进制执行。因此, 大写字母分在小写字母的前面。当结果是以字母顺序排列来显示的时候, 这样的效率很高, 但是实用较少
<i>CHARACTER SET</i> <i>name</i> [<i>COLLATE</i> <i>sort</i>]	字符串的字符集和排序方式(后者是可选的)
<i>COMMENT</i> <i>text</i>	把 <i>text</i> 作为注释存入数据列(从 MySQL 4.1 版本开始)
<i>SERIAL</i>	从 MySQL 4.1 版本开始, 与 <i>BIGINT NOT NULL AUTO_INCREMENT UNIQUE</i> 同义

21.5 SQL 命令汇总表 (按功能分类)

在本章后面的内容里, 我们将按照字母顺序把 SQL 的命令列出。在这里, 先按照 SQL 命令的功能把它们系统地汇总在一起。

数据库查询, 数据处理

<i>SELECT</i>	查询现有记录(搜索数据)
<i>INSERT</i> ~	插入新记录
<i>REPLACE</i>	取代现有的记录
<i>UPDATE</i>	改变现有的记录
<i>DELETE</i>	删除选中的记录
<i>TRUNCATE TABLE</i>	删除一个数据表的全部记录
<i>LOAD DATA</i>	从一个文本文件中插入记录
<i>HANDLER</i>	比 <i>SELECT</i> 更有效地读取数据(从 MySQL 4.0 版本开始)

事务(仅适用于 InnoDB 数据表)

<i>BEGIN</i> 或 <i>START TRANSACTION</i>	开始一个事务
<i>COMMIT</i>	确认当前事务里的所有 SQL 命令
<i>ROLLBACK</i>	放弃当前事务里的所有 SQL 命令
<i>SAVEPOINT</i>	在当前事务里放置一个折返点标记

创建数据库/数据表/视图，改变数据库计划

<i>ALTER DATABASE</i>	对数据库的结构做出改变（从 MySQL 4.1 版本开始）
<i>ALTER TABLE</i>	对数据表中的各个数据列做出改变，添加索引等
<i>ALTER VIEW</i>	改变视图（从 MySQL 5.0 版本开始）
<i>CREATE DATABASE</i>	创建一个新的数据库
<i>CREATE INDEX</i>	为数据表创建一个新的索引
<i>CREATE TABLE</i>	创建一个新的数据表
<i>CREATE VIEW</i>	创建一个视图（从 MySQL 5.0 版本开始）
<i>DROP DATABASE</i>	删除一个完整的数据库
<i>DROP FUNCTION 或 PROCEDURE</i>	删除一个存储过程（从 MySQL 5.0 版本开始）
<i>DROP INDEX</i>	删除一个索引
<i>DROP TABLE</i>	删除一个完整的数据表
<i>DROP VIEW</i>	删除一个视图（从 MySQL 5.0 版本开始）
<i>RENAME TABLE</i>	重命名一个数据表

数据表的管理（通用命令）

<i>ANALYZE TABLE</i>	返回关于索引内部管理工作的信息
<i>CHECK TABLE</i>	检查数据表文件是否已被损坏
<i>FLUSH TABLES</i>	先关闭、再重新打开所有的数据表文件
<i>LOCK TABLE</i>	阻断其他用户对数据表进行写操作
<i>OPTIMIZE TABLE</i>	对数据表的内存使用情况进行优化
<i>UNLOCK TABLES</i>	解除由 <i>LOCK</i> 命令设置的数据表锁定

MyISAM 数据表的管理

<i>BACKUP TABLE</i>	把数据表文件复制到备份目录里
<i>CACHE INDEX</i>	把单个的缓存区分配到数据表索引
<i>LOAD INDEX INTO CACHE</i>	把数据表索引载入到缓存区
<i>REPAIR TABLE</i>	尝试修复被损坏的数据表文件
<i>RESTORE TABLE</i>	用 <i>BACKUP</i> 备份恢复数据表

存储过程和触发器的管理与执行（从 MySQL 5.0 版本开始）

<i>ALTER FUNCTION PROCEDURE</i>	修改一个存储过程
<i>CALL</i>	调用一个存储过程
<i>CREATE FUNCTION PROCEDURE TRIGGER</i>	创建一个存储过程或触发器
<i>DROP FUNCTION PROCEDURE TRIGGER</i>	删除一个存储过程或触发器
<i>SHOW CREATE FUNCTION PROCEDURE</i>	查看一个存储过程的代码
<i>SHOW FUNCTION PROCEDURE STATUS</i>	返回一份现有存储过程的清单

数据库结构信息及其他管理用信息

<i>DESCRIBE</i>	与 <i>SHOW COLUMNS</i> 同义
<i>EXPLAIN</i>	解释 <i>SELECT</i> 命令在 MySQL 服务器的内部是如何执行的
<i>SHOW</i>	查看关于数据库、数据表、视图、字段、存储过程等的信息

管理、访问权限等

<i>FLUSH</i>	先清空 MySQL 的各个临时缓冲区（把信息写入硬盘）、再重新读入那些信息
<i>GRANT</i>	授予一项权限
<i>KILL</i>	终止一个进程
<i>REVOKE</i>	收回一项权限
<i>RESET</i>	删除查询缓存区（query cache）或日志文件
<i>SET</i>	改变 MySQL 系统变量的值
<i>SHOW</i>	查看 MySQL 的工作状态、系统变量、进程等
<i>USE</i>	改变当前数据库

镜像机制（主控服务器）

<i>PURGE MASTER LOGS</i>	删除旧的日志文件
<i>RESET MASTER</i>	删除所有日志文件
<i>SET SQL_LOG_BIN=0/1</i>	禁用/激活二进制日志功能
<i>SHOW BINLOG EVENTS</i>	查看当前日志文件里的事件清单（从 MySQL 4.0 版本开始）
<i>SHOW MASTER LOGS</i>	返回一份全体日志文件的清单
<i>SHOW MASTER STATUS</i>	指定当前活跃的日志文件
<i>SHOW SLAVE HOSTS</i>	返回一份全体已注册从属服务器的清单（从 MySQL 4.0 版本开始）

镜像机制（从属服务器）

<i>CHANGE MASTER TO</i>	改变 master.info 文件里的镜像机制设置
<i>LOAD DATA FROM</i>	从主控服务器把所有的数据表复制到从属服务器（从 MySQL 4.0 版本开始）
<i>LOAD TABLE FROM</i>	从主控服务器把某一个数据表复制到从属服务器
<i>RESET SLAVE</i>	重新对 master.info 文件进行初始化
<i>SHOW SLAVE STATUS</i>	显示 master.info 文件的内容
<i>SLAVE START/STOP</i>	启用/停用镜像机制

21.6 SQL 命令指南（按字母表顺序排列）

在接下来的内容里，使用了以下语法元素：

[option]: 用方括号来给出 SQL 命令的可选部分。

variant1 | variant2 | variant3: 用“|”字符来分隔“n 选一”选项。

ALTER DATABASE [dbname] actions

从 MySQL 4.1 版本开始，可以使用 *ALTER DATABASE* 命令来改变数据库的全局属性。设置保存在文件 *dbname/db.opt* 中。可以使用相同的命令 *ALTER SCHEMA* 来取代 *ALTER DATABASE* 命令。如果没有 *dbname*,

该命令应用于当前数据库。

actions: 目前有两个动作可选。

- [DEFAULT] CHARACTER SET CHARSET* 指定了哪一个字符集是数据库应当使用的默认字符集。(在数据表和数据列的定义里, 可以指定不同的字符集。)
- [DEFAULT] COLLATE collname* 指定了默认的排序方式。

ALTER FUNCTION/PROCEDURE name options

从 MySQL 5.0 版本开始, *ALTER FUNCTION/PROCEDURE* 命令可以改变存储过程 (SP) 的许多细节, 但该命令不能用来改变 SP 的代码。如果想改变 SP 的代码, 只能先删除 SP (*DROP FUNCTION/PROCEDURE*) 然后再重新创建它 (*CREATE FUNCTION/PROCEDURE*)。

options:

- NAME newname* 重命名这个存储过程。
- SQL SECURITY DEFINER/INVOKER* 改变 SP 的安全模式 (参阅 *CREATE FUNCTION*)。
- COMMENT 'newcomment'* 改变储存在 SP 中的注释。

ALTER TABLE tblname tloptions

ALTER TABLE 用于改变一个数据表的各个方面。下面提供有关这个语法的概要。

在这里给出的最简单的语法形式里, *ALTER TABLE* 改变的是数据表的选项。可供改变的选项将在稍后的 *CREATE TABLE* 命令条目里介绍。该命令还可以用来改变数据表的类型(例如从 MyISAM 改为 InnoDB)。

注意, *ALTER TABLE* 命令的许多语法是通过重新创建数据表来实现其功能的, 其基本步骤是: 首先, MySQL 先用新的数据表属性创建一个新数据表 *X*, 然后把所有的记录复制到这个新数据表; 然后, 把现有的数据表重命名为 *Y*, 把数据表 *X* 重命名为 *tblname*; 最后, 删除数据表 *Y*。对于比较大的数据表, 这往往需要很长的时间并临时占用大量的硬盘空间。

ALTER TABLE *tblname* ADD *newcolname* *coltype* *coloptions* [FIRST | AFTER *existingcolumn*]

该命令为数据表添加一个新的数据列。对新数据列的定义与 *CREATE TABLE* 命令里一样。如果这个新数据列的位置没有用 *FIRST* 或 *AFTER* 命令指定, 那么它将位于数据表的最后一列。

下面这个例子将把一个数据类型是 *TIMESTAMP* 的新数据列 *ts* 添加到 *authors* 的数据表中:

ALTER TABLE authors ADD ts TIMESTAMP

ALTER TABLE *tblname* ADD INDEX [*indexname*] (*indexcols* ...)
 ALTER TABLE *tblname* ADD FULLTEXT [*indexname*] (*indexcols* ...)
 ALTER [IGNORE] TABLE *tblname* ADD UNIQUE [*indexname*] (*indexcols* ...)
 ALTER [IGNORE] TABLE *tblname* ADD PRIMARY KEY (*indexcols* ...)
 ALTER TABLE *tblname* ADD SPATIAL INDEX (*indexcol*)

这些命令为一个数据表创建一个新索引。如果没有给出 *indexname*, MySQL 将使用被索引数据列的名字。

如果在创建一个 *UNIQUE* 或主索引的过程中发现了重复的字段, 可选关键字 *IGNORE* 就开始起作用了。如果没有 *IGNORE*, 命令将在遇到一个错误时终止, 不能生成索引。如果使用了 *IGNORE*, 那些重复的记录将被删除。

从 MySQL 4.1 版本开始, 可以为几何数据生成二维坐标索引。数据列 *indexcol* 必须是 *GEOMETRY* 数

据类型并具有 *NOT NULL* 属性。

```
ALTER TABLE tblname ADD [CONSTRAINT [fr_keyname]]
  FOREIGN KEY [c1_keyname]
    (column1) REFERENCES table2 (column2)
  [ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT}]
  [ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT}]
```

该命令为数据表定义了一个外键约束条件，即外键 *tblname.column1* 指向 *table2.column2*，数据表引擎将负责保证所有的引用都是有效的（不会指向一个并不存在的字段）。

该命令将创建两个新索引：一个是建立在 *column1* 和 *column2* 这两个数据列上的外键索引，另一个是建立在 *tblname.column1* 数据列上的普通索引（如果它尚不存在的话）。

如果需要，可以给这些索引起个名字（*c1_keyname* 和 *fr_keyname*）。如果使用的是镜像机制系统，必须这样做；否则，就有可能发生 MySQL 给原始数据库和镜像数据库起的名字不一样的情况，而这将在打算删除外键约束规则时引起一些难以预料的问题。

可选的 *ON DELETE* 和 *ON UPDATE* 子句规定了数据表引擎将如何处理因 *DELETE* 和 *UPDATE* 命令而导致的数据完整性遭到破坏的情况（详见第 8 章）。这里的默认设置是 *STRICT*，意思是不执行有可能破坏数据完整性的 SQL 命令并触发一条出错消息。

目前（MySQL 5.0.n），外键约束条件仅适用于 InnoDB 数据表。数据列 *column2* 上必须有一个索引而且必须与 *column1* 数据列有着同样的数据类型。

```
ALTER TABLE tblname ALTER colname SET DEFAULT value
ALTER TABLE tblname ALTER colname DROP DEFAULT
```

该命令改变了数据列或数据表的默认值，或是删除一个已经存在的默认值。

```
ALTER TABLE tblname CHANGE oldcolname newcolname coltype coloptions
```

此命令改变数据表中数据列的默认值，或是删除一个已经存在的默认值。可能需要参考当初在 *CREATE TABLE* 命令里对数据列做出的声明。如果数据列的名字保持不变，就必须两次写出它（*oldcolname* 和 *newcolname* 完全一样）。假如命令 *ALTER TABLE* 只用来改变数据列的名字，*coltype* 和 *coloptions* 选项都必须完整地给出。

```
ALTER TABLE tblname CONVERT TO
  CHARACTER SET charset [COLLATE collname]
```

该命令改变数据表中所有文本数据列的字符集和排序方式。这种变化不但影响数据表形式上的定义，而且影响到数据表的内容：记录的所有文本字段都被转变。

如果只是想改变数据表的定义，而不想改变它的内容，那么必须把相应的数据列改变成 *BLOB* 类型，然后再使用相关的字符集和排序方式转化成想要的文本数据类型。这样就不会改变数据，因为 MySQL 不会改变 *BLOB* 类型的数据：

```
ALTER TABLE tblname CHANGE colname colname BLOB
ALTER TABLE tblname CHANGE colname colname VARCHAR(100) CHARACTER SET ...
```

如果文本数据列具有索引，必须在第一个 *ALTER TABLE* 命令之前删除索引，然后在第二个 *ALTER TABLE* 命令之后再重新创建它。

```
ALTER TABLE tblname DISABLE KEYS
ALTER TABLE tblname ENABLE KEYS
```

从 MySQL 4.0 版本开始，命令 *ALTER TABLE...DISABLE KEYS* 有下面的作用：在使用 *INSERT*、*UPDATE* 和 *DELETE* 命令的时候，所有的非唯一索引不再自动刷新。恢复这种索引自动刷新功能的命令是 *ALTER TABLE...ENABLE KEYS*。

应该以最有效的方式使用这两个命令，从而实现数据表的扩展修正。（使用命令 *ENABLE KEYS* 重新构建索引所用的时间要比每修改一条记录就刷新一次索引所花费的时间少很多。）

```
ALTER TABLE dbname.tblname DISCARD TABLESPACE
ALTER TABLE dbname.tblname IMPORT TABLESPACE
```

这两个命令只适用于有专用数据文件的 InnoDB 类型的数据表（MySQL 服务器选项 *innodb_file_per_table*）。在 MySQL 5.0 中，不允许从一个数据库目录中把这样的文件复制到另外一个数据库目录或是从一个 MySQL 安装复制到另外一个。在特定的场合，这两个 *ALTER TABLE* 命令可以用来禁用/激活一个表空间（tablespace）。

ALTER TABLE dbname.tblname DISCARD TABLESPACE 命令将删除数据表 *tblname* 及其底层文件 *dbname/tblname.ibd*，但 *tblname.frm* 文件会保留下。*ALTER TABLE dbname.tblname IMPORT TABLESPACE* 将再次激活 *dbname/tblname.ibd* 文件；这个文件必须是在执行 *DISCARD TABLESPACE* 命令之前为运行中的 MySQL 安装制作的备份结果。

关于这些命令的进一步讨论可以访问下面的网址找到（虽然这些命令的实际用途并不多）：<http://dev.mysql.com/doc/mysql/en/multiple-tablespaces.html>。

```
ALTER TABLE tblname DROP colname
ALTER TABLE tblname DROP INDEX indexname
ALTER TABLE tblname DROP PRIMARY KEY
ALTER TABLE tblname DROP FOREIGN KEY foreign_key_name
```

前 3 个命令用于删除一个数据列、一个索引或一个主索引。从 MySQL 4.0.13 版本开始，第 4 个命令删除给定的外键约束条件。可以使用 *SHOW CREATE TABLE* 命令查出打算删除的索引的 *foreign_key_name*。

如果使用的是镜像机制，应该避免删除 *FOREIGN KEY* 规则。因为在 *FOREIGN KEY* 规则定义的时候，MySQL 会创建一个特殊的索引。假如没有明确地命名这个索引，就有可能出现原始数据库和镜像数据库使用不同名字的情况。

```
ALTER TABLE tblname ENGINE tabletype
```

该命令用来改变数据表（数据表引擎）的类型。可供选用的数据表类型包括 InnoDB 和 MyISAM。注意，只有在新的数据表引擎支持数据表全部属性的情况下才可以改变数据表的类型。例如，InnoDB 的数据表引擎目前不支持全文本索引，如果要把一个 MyISAM 类型的数据表转变成 InnoDB 类型的数据表，必须首先删除全文本索引（使用命令 *ALTER TABLE tblname DROP indexname*）。

```
ALTER TABLE tblname MODIFY colname coltype coloptions
```

该命令的作用就像 *ALTER TABLE...CHANGE* 命令（见前面命令）一样。唯一的不同点在于数据列不能够改变，所以只需要给出一次名字。

```
ALTER TABLE tblname ORDER BY colname
```

该命令将重新创建数据表并按 *colname* 数据列对数据记录进行排序。如果经常从 *colname* 排序的数据表中读取记录，该命令的使用将会有助于提高一些效率。该命令对新记录和被修改的记录不起作用，所以该命令一般只用于很少需要修改的数据表。

```
ALTER TABLE tblname RENAME AS newtblname
```

该命令重命名一个数据表（参见 *RENAME TABLE* 命令）。

```
ALTER TABLE tblname TYPE tabletype
```

该命令对应于命令 *ALTER TABLE tblname ENGINE tabletype*。

```
ALTER [aloption] VIEW viewname [(columns)] AS command [chkoption]
```

ALTER VIEW 改变了视图的属性。它与 *CREATE VIEW* 命令具有相同的语法。*ALTER VIEW* 不能够改变一个视图的名字。

```
ANALYZE TABLE tablename1, tablename2, ...
```

ANALYZE TABLE 命令将对一个数据列的被索引值进行分析并把分析结果保存起来，这可以在今后加快通过索引去访问数据记录的速度。

对于 MyISAM 类型的数据表，可以使用工具程序 *myisamchk -a* *tblfile*。

```
BACKUP TABLE tblname TO '/backup/directory'
```

BACKUP TABLE 把文件从指定的 MyISAM 类型的数据表中复制到备份目录里。可以使用 *RESTORE TABLE* 命令重新创建数据表。

在 UNIX/Linux 操作系统下，在账户下执行 MySQL 的账户备份目录必须是可写的。

BACKUP 和 *RESTORE* 命令不适用于 InnoDB 类型的数据表。这两个命令一般认为是“不可取”的，最好不要使用。替代方法是使用外部备份工具程序，诸如 *mysqldump*、*mysqlhotcopy* 或是 InnoDB 的备份工具程序。

```
BEGIN
```

如果是在可以事务的数据表上工作，那么 *BEGIN* 命令将开始一个新事务。随后的 SQL 命令可以使用 *COMMIT* 命令来确认，或是使用 *ROLLBACK* 命令来撤销。（对数据表的所有修改只有在执行了 *COMMIT* 命令之后才会永久生效。）关于事务的进一步讨论和用法示例请参见本书第 10 章的内容。

从 MySQL 4.0.11 版本开始，还可以使用 ANSI 标准中的 *START TRANSACTION* 命令来代替 *BEGIN* 命令。

```
CACHE INDEX indexspec1, indexspec2 ... IN cachename
```

从 MySQL 4.1 版本开始, *CACHE INDEX* 命令决定了 MyISAM 索引是位于哪一个缓存区。只有提前创建过一个这样的缓存区, 该命令才能够有效使用 (参见本章关于结构变量的讨论内容)。

indexspec: 给出需要为哪些索引改变它们的高速缓存, 这些索引需要用如下所示的语法给出:

tablename [[*INDEX / KEY*] (*indexname1, indexname2...*)]

如果没有给出索引的名字, 那么该命令将对 *tablename* 中的所有索引有效。

cachename: 给出 MyISAM 索引缓存区的名字。必须提前使用 *SET @@cachename.key_buffer_size=n* (*n* 为缓存区的字节长度) 来建立一个这样的缓存区。

CALL spname [parameter1, parameter2 ...]

该命令调用一个存储过程。*CALL* 命令仅适用于用户定义的 SP 过程, 不能用来调用 SP 函数; SP 函数必须用普通的 SQL 命令 (如 *SELECT* 命令) 调用。

CHANGE MASTER TO variable1=value1, variable2=value2, ...

使用该命令, 用于从控的镜像机制设置将被执行。设置存储在文件 *master.info* 中。该命令只是适用于镜像机制系统中的从属计算机, 并且要求有 *Super* 权限。它能够识别以下变量名:

MASTER_HOST: 主控服务器的主机名或 IP 地址。

MASTER_USER: 用来与主控服务器进行通信的用户名。

MASTER_PASSWORD: 相关的密码。

MASTER_PORT: 主控服务器主机的通信端口 (通常为 3306 号端口)。

MASTER_LOG_FILE: 主控服务器上的当前日志文件。

MASTER_LOG_POS: 主控服务器日志文件里的当前读取位置。

CHECK TABLE tablename1, tablename2 ... [TYPE=QUICK]

命令 *CHECK TABLE* 将检查给定数据表的数据库文件是否遭到损坏, 但并不对发现的错误进行纠正。对于 MyISAM 类型的数据表, 可以使用外部程序命令 *myisamchk -m tb1file* 来取代该命令。

COMMIT

COMMIT 命令终止一个事务并把此前做出的所有修改写入数据库。(如果不想使用 *COMMIT* 命令, 可以使用 *ROLLBACK* 命令来撤销那些尚未写入数据库的修改。)*BEGIN/COMMIT/ROLLBACK* 只对支持事务的数据表起作用。有关事务进一步的讨论请参阅第 10 章的相关内容。

CREATE DATABASE [IF NOT EXISTS] dbname [options]

CREATE TABLE 生成给定的数据库。(更准确地说, 是生成一个空白的目录, 属于新数据库的那些数据表可以储存在里面。) 注意, 数据库的名字是区分字母大小写的。只有那些有权创建数据库的用户才能使用该命令。与 *CREATE DATABASE* 命令等价的 *CREATE SCHEMA* 命令也可以使用。

options: 从 MySQL 4.1 版本开始, 可以为一个数据表规定默认的字符集:

[DEFAULT] CHARACTER SET charset [COLLATE collname]。

可选的关键字 *DEFAULT* 不起作用 (就是说, 定义或不定义它都没有关系)。如果给定字符集支持一

种以上的排序方式，就可以使用 *COLLATE* 来选择其中之一。假如没有给出 *CHARACTER SET*，那么就会使用服务器的默认字符集。

```
CREATE FUNCTION name ([parameters]) RETURNS datatype [options] code
```

从 MySQL 5.0 版本开始，*CREATE FUNCTION* 命令在当前数据库中创建一个用户定义函数（存储过程）。创建存储过程必须具备 *Super* 权限，执行它们必须具备 *Execute* 权限。

- *name*: SP 函数名字。对于单一数据库中的函数和程序，允许使用相同的名字（参见 *CREATE PROCEDURE*）。
- *parameters*: 如果需要给出多个参数，要用逗号把它们隔开。每一个参数的后面必须是它的数据类型，例如，*para1 INT, para2 BIGINT*。
- *datatype*: 给出了函数返回值的数据类型。所有 MySQL 的数据类型都可以，如 *INT, DOUBLE, VARCHAR(n)*。
- *options*: 下面的选项可以用于函数定义中：
 - *LANGUAGE SQL*: 给出存储过程代码的语言。唯一允许的 *LANGUAGE* 设置就是当前的 SQL。这个设置保持默认。将来的 MySQL 版本会提供用其他程序设计语言来定义 SP 的选项（如：PHP 语言）。
 - *[NOT] DETERMINISTIC*: 如果一个 SP 总是返回具有同样参数的同样结果，那么这个 SP 就被看做具有确定性。（结果取决于数据库内容的 SP 是不确定的。）默认情况下，SP 是不确定的。确定的 SP 能够高效地执行。（例如，在缓存区里储存特殊参数的结果是可能的。）但是，目前的 *DETERMINISTIC* 选项被 MySQL 的优化功能所忽略。
 - *SQL SECURITY DEFINER/INVOKER*: *SECURITY* 模式规定了在哪种访问权限下，SP 才能够执行。使用 *SQL SECURITY DEFINER* 定义后的 SP 具有与定义了 SP 的 MySQL 用户同样的权限。这是默认的安全方式。使用选项 *SQL SECURITY INVOKER* 定义的 SP 具有执行 SP 的 MySQL 用户的访问权限。
 - *COMMENT 'text'*: 注释文本，随 SP 一起储存。
 - *code*: 实际的 SP 代码。通常以 SQL 命令的格式给出。如果 SP 里包含不止一条命令，它们必须放在 *BEGIN* 和 *END* 之间，并且要用分号隔开（详见第 13 章）。

下面是一个例子：

```
CREATE FUNCTION half(a INT) RETURNS INT
BEGIN
    RETURN a/2;
END
```

```
CREATE FUNCTION name RETURNS datatype SONAME libraryname
```

CREATE FUNCTION 命令不仅可以用来定义 SP，还可以用来把外库函数导入 MySQL。人们把这样的函数称为用户定义函数（user-defined function）或简称为 UDF。用 C 或 C++ 语言来设计这样的函数需要具备这些函数将如何在 MySQL 里工作的背景知识（当然，还需要有必要的工具，如编译器）。详细的资料可以从以下站点获得：<http://dev.mysql.com/doc/mysql/en/extending-mysql.html> 以及 <http://mysql-udf.sourceforge.net/>。

```
CREATE [UNIQUE|FULLTEXT] INDEX indexname ON tablename (indexcols ...)
```

CREATE INDEX 命令给一个现有的数据库增加一个索引。对于 *indexname*，通常就使用数据列的名字。

CREATE INDEX 不是一个独立的命令，它只是 *ALTER TABLE ADD INDEX/UNIQUE* 命令的一种语法变体，后者已经在前面的内容里详细介绍过了。

CREATE PROCEDURE name ([parameters]) [options] code

从 MySQL 5.0 版本开始，*CREATE PROCEDURE* 在当前数据库中创建一个用户定义过程（存储过程）。创建存储过程必须具备 *Super* 权限，执行它们必须具备 *Execute* 权限。

该命令的语法非常类似于 *CREATE FUNCTION* 命令。但与 SP 函数不同的是，SP 过程不能直接返回一个结果。因此，参数表的语法与 *CREATE FUNCTION* 命令稍有不同。

parameters: 能够定义多个参数，它们之间用逗号分开。每个参数的定义如下：

[IN or OUT or INOUT] parametername datatype

关键字 *IN*、*OUT* 和 *INOUT* 定义了参数是仅用于输入、仅用于输出、还是同时用于输入和输出（默认设置是 *IN*）。参数 *datatype* 可以是任何一种 MySQL 数据类型如 *INT*、*VARCHAR(n)*、*DOUBLE* 等。

下面是一个例子：

```
CREATE PROCEDURE half(IN a INT, OUT b INT)
BEGIN
    SET b=a/2;
END
```

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tblname
  (colname1 coltype coloptions reference,
   colname2 coltype coloptions reference...
  [ , index1, index2 ...]
  )
[tbloptions]
```

CREATE TABLE 命令在当前数据库中生成一个新的数据表。如果数据库不是当前使用的数据库，那么数据表的名字可以用格式 *dbname.tblname* 来定义。如果数据表已经存在，会产生一个错误信息。但如果使用了 *IF NOT EXISTS*，就不会有错误信息，在这种情况下，现有数据表不会受到影响，没有生成新的数据表。

如果使用了关键字 *TEMPORARY*，就将生成一个临时数据表。如果临时数据表与一个现有数据表同名，而这个数据表不是临时数据表，那么临时数据表会被生成，没有错误信息。此时，旧的数据表被保存，而且由临时数据表“隐藏”了起来。如果想让这个临时数据表仅存在于 RAM 中（为了增加速度），就必须给出 *ENGINE = HEAP*。

创建一个普通的数据表必须具备 *Create* 权限。从 MySQL 4.0 版本开始，创建一个临时数据表必须具备 *Create Temporary Table* 权限。

colname: 数据列的名字。

coltype: 数据列的数据类型。MySQL 数据类型的清单（*INT*、*TEXT* 等）见本章前面的内容。

coloptions: 这里规定了某些属性（选项）：

```
NOT NULL | NULL
UNSIGNED
ZEROFILL
BINARY
DEFAULT defaultval | DEFAULT CURRENT_TIMESTAMP
AUTO_INCREMENT | IDENTITY
PRIMARY KEY
```

CHARACTER SET charset [COLLATE collname]

COMMENT text

reference: 为了保证引用一致性, MySQL 提供了许多关键字来定义各种外键。如 *REFERENCES tablename(idcolumn)*。但这些关键字目前被忽略(并且没有错误信息)。只是在使用 InnoDB 类型的数据表的时候, 针对这种类型数据表的外键定义必须在索引定义的范围里被规定。

index: *KEY* 或 *INDEX* 用来定义一个涉及一个或多个数据列的普通索引。*UNIQUE* 用来定义一个唯一化索引(在一个或多个数据列里, 不存在彼此重复雷同的数据值或数据值组合)。为了方便对索引进行内部管理, 这两种语法变体都允许程序员给索引起一个名字。*PRIMARY KEY* 同样地定义一个 *UNIQUE* 索引。但是, 这时的索引名字是预先确定的: 那就是 *PRIMARY*。用 *FULLTEXT* 定义出来的是一个用于是全文检索的索引(在 MySQL 4.0 里, 只有 MyISAM 数据表支持全文检索)。从 MySQL 4.1 版本开始, 基于 *GEOMETRY* 数据的索引可以用 *SPATIAL INDEX* 来生成; *indexcol* 必须定义有 *NOT NULL* 属性。

KEY\INDEX {indexname} (indexcols ...)

UNIQUE \INDEX {indexname} (indexcols ...)

PRIMARY KEY (indexcols ...)

FULLTEXT {indexname} (indexcols ...)

SPATIAL INDEX {indexname} (indexcol)

Foreign key constraints: 如果正在使用 InnoDB 类型的数据表, 可以在这里声明一些外键约束条件。语法如下所示:

[CONSTRAINT {fr_keyname}]

FOREIGN KEY {c1_keyname} (column1) REFERENCES table2 (column2)

[ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT}]

[ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT}]

这就是说, *tblname.column1* 是一个指向 *table2.column2* 的外键。更详细的讨论请参阅命令 *ALTER TABLE... ADD FOREIGN KEY*。

tbloptions: 在这里, 可以定义各种数据表选项, 不过这里将只给用户展示一些最重要的选项。并不是对每一个数据表所有的选项都能够使用。有关不同类型数据表及其变量的内容见第 8 章。

ENGINE = MYISAM | HEAP | INNODB

ROW_FORMAT= default | dynamic | static | compressed

关键字 *AUTO_INCREMENT* 将为 *AUTO_INCREMENT* 数据列给出一个计数初始值(例如, 如果打算使用 6 位数的编号, 这个值可以是 100000)。*CHECKSUM=1* 的作用是储存每一个数据记录的校验和, 在数据库损坏的时候, 这将有助于重新构建数据库。*PACK_KEYS=1* 产生一个较小的索引文件。这将加快读操作的速度, 但会使修改操作的速度变慢。*DELAY_KEY_WRITE=1* 意味着索引不是在记录每一次产生变化的时候都进行刷新。不过, 随时刷新要好一些。

AUTO_INCREMENT = n

CHECKSUM = 0 | 1

PACK_KEYS = 0 | 1

DELAY_KEY_WRITE = 0 | 1

使用 *COMMENT* 关键字, 可以储存一个简要的文本。例如, 文本可以用来描述数据表的用途。可以使用 *SHOW CREATE DATABASE dbname* 命令来读取注释。

COMMENT='comment'

从 MySQL 4.1 版本开始, 可以为一个数据表规定字符集和排序方式。(也可以为某一个数据列设置这

些参数。) 如果没有规定字符集, 那么就使用数据表和服务器的默认字符集:

[DEFAULT] CHARACTER SET charset [COLLATE collname].

CREATE TABLE 语法包含着一些功能重复的语法元素。比如说, 有两种办法可以定义一个主索引: 其一是作为数据列的属性 (*coloptions*) ; 其二是作为一个独立的索引 (*index*) 。当然, 这两种办法的最终结果是一样的, 喜欢用哪种格式都没有关系。

有时候, MySQL 会自做主张地对程序员给出的数据列定义做出一些修改, 比如说, 它会在程序员没有为数据列明确地定义一个默认值的时候自动提供一个它认为适当的默认值。*(silent column changes; 参见第 9 章。)* 因此, 在创建一个数据表之后, 最好能用 *SHOW TABLE name* 命令去查看一下实际的 MySQL 数据表的定义。

下面是一个例子:

```
CREATE TABLE test (id      INT NOT NULL AUTO_INCREMENT,
                  data    INT NOT NULL,
                  txt     VARCHAR(60),
                  PRIMARY KEY (id))
```

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tblname
  [(newcolname1 coltype coloptions reference,
    newcolname2 coltype coloptions reference ...
    [ , key1, key2 ...]
  )]
  [tbloptions]
  [IGNORE | REPLACE] SELECT ...
```

CREATE TABLE 命令的这个变体在创建数据表的同时还将把一条 *SELECT* 命令的查询结果填充到新创建的数据表里去。新数据表里的各个数据列将沿用来自 *SELECT* 查询结果的数据类型, 所以用不着(也不允许)明确地加以定义。

但可惜的是老数据表里的索引和 *AUTO_INCREMENT* 等属性在新数据表里都不存在。数据列的类型也有可能发生变化, 比如老数据表里的 *VARCHAR* 数据列在新数据表里变成了 *CHAR* 数据列。

如果想为新数据表里的单个数据列定义一个索引(如 *PRIMARY KEY(id)*), 这个变体还是允许这么做的。除此之外, 这个变体还提供了一些选项使用户可以为新数据表定义一些新的数据列(比如一个 *AUTO_INCREMENT* 数据列)。

关键字 *IGNORE* 和 *REPLACE* 定义了 MySQL 在有多条内容重复的记录被该命令放入一个 *UNIQUE* 数据列时应该采取何种措施: *IGNORE* 的含义是保留现有记录, 忽略新记录; *REPLACE* 的含义是用新记录替换掉现有记录; 如果这两个选项都没有给出, 就会导致一条出错消息。

如果要复制数据表, 最好使用命令 *CREATE TABLE table2 LIKE table1* (从 MySQL 4.1 版本开始) 建立一个新的数据表, 然后再使用命令 *INSERT INTO table2 SELECT* FROM table1* 把数据进行复制。

下面是一个例子:

```
CREATE TEMPORARY TABLE tmp
  SELECT id, authName FROM authors WHERE id<20
```

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] newtable LIKE oldtable
```

该命令始见于 MySQL 4.1 版本, 它将使现有数据表 *oldtable* 的声明定义创建一个空白的新数据表 *newtable*。

```
CREATE TRIGGER name time event
  ON tablename
  FOR EACH ROW code
```

命令 *CREATE TRIGGER* 定义的 SQL 代码将在用户对给定数据表执行特定数据库命令之前或之后自动执行。每个事件只允许触发一个触发器过程。执行 *CREATE TRIGGER* 命令必须具备 *Super* 权限。

name: 触发器的名字。目前 (MySQL 5.0.3 版本)，如果触发器是为同一个数据库里的不同数据表定义的，它们的名字允许相同。未来的 MySQL 版本据说要求同一个数据库里的触发器都必须有唯一的名字。

time: BEFORE | AFTER

这个选项决定着触发器代码是在触发器其事件发生之前还是发生之后执行。

event: INSERT | UPDATE | DELETE

这个选项决定着哪些数据库操作会导致触发器代码自动执行。

tablename: 触发器是为哪一个数据表定义的。

code: 触发器代码。语法与存储过程一样。

下面是一个例子：

```
CREATE TRIGGER test_before_insert
  BEFORE INSERT ON test FOR EACH ROW
BEGIN
  IF NEW.percent < 0.0 OR NEW.percent > 1.0 THEN
    SET NEW.percent = NULL;
  END IF;
END
```

`CREATE [aloption] VIEW viewname [(columns)] AS command [chkoption]`

从 MySQL 5.0 版本开始，命令 *CREATE VIEW* 创建一个视图 (*View*)。这是一个基于 *SELECT* 命令的虚拟数据表。如果想取代一个已经存在的视图，按照如下格式执行命令 *CREATE OR REPLACE...*。执行该命令必须具备 *Create View* 权限。

aloption: ALGORITHM = UNDEFINED | MERGE | TEMPTABLE

ALGORITHM 负责给出新视图在 MySQL 内部应该如何表示。截止到本书交稿之时，这个选项还没有在 MySQL 文档里有正式的说明。在默认情况下，MySQL 目前总是使用 *UNDEFINED*（它可以由命令 *SHOW CREATE TABLE viewname* 来决定）。

viewname: 新视图的名字。命名规则与数据表相同。视图的名字允许与其底层的数据表名字不一样。

columns: 作为一般原则，新视图里的各个数据列应该与底层数据表里的各有关数据列有着同样的名字和数据类型，但完全可以利用 *columns* 选项为它们定义一个新的名字和新的数据类型。这个语法与命令 *CREATE TABLE* 一样。

command: 必须在这里给出一条 *SELECT* 命令。视图里的数据其实就是这条 *SELECT* 命令的查询结果。

chkoption: WITH [CASCADED | LOCAL] CHECK OPTION

WITH CHECK OPTION 意味着对视图中的数据记录进行修改只有在满足 *SELECT* 命令的 *WHERE* 子句条件时才允许。当然，使用 *WITH CHECK OPTION* 选项的大前提是新视图是允许修改的。

变量 *WITH LOCAL CHECK OPTION* 对来自其他的 *View* 起作用。*LOCAL* 指的是仅考虑 *CREATE VIEW* 命令的 *WHERE* 子句条件，而不考虑更高级别 *View* 的 *WHERE* 子句条件。

使用 *WITH CASCADED CHECK OPTION* 会出现反作用：现在所有更高级别 *View* 的 *WHERE* 子句条件均被考虑。如果既没有规定 *CASCADED*，也没有规定 *LOCAL*，那么就默认 *CASCADED*。

下面是一个例子：

```
CREATE VIEW v1 AS
  SELECT titleID, title, subtitle FROM titles
  ORDER BY title, subtitle
```

```
DELETE [deleteoptions] FROM tablename
[WHERE condition]
[ORDER BY ordercolumn [DESC]]
[LIMIT maxrecords]
```

DELETE 命令用来从数据表里删除那些满足 *condition* 条件的数据记录。

deleteoptions: LOW_PRIORITY, QUICK, IGNORE

LOW_PRIORITY 选项的作用是，只有在所有的读取操作完成之后，数据记录才可以删除。（这个选项的目的是避免由于 *DELETE* 操作而造成 *SELECT* 查询不必要的延迟。）

QUICK 选项的作用是，在删除过程中不对现有的索引进行优化。这可以提高 *DELETE* 命令的执行速度，但会在某种程度上降低索引的效率。

从 MySQL 4.1 版本开始，*IGNORE* 选项的作用是，即使出现错误，*DELETE* 命令也继续进行。所有的错误转变成警告，可以使用 *SHOW WARNINGS* 命令来读取。

condition: 这个子句定义了哪些记录要被删除。有关 *condition* 的语法，请参阅后面对 *SELECT* 命令的说明。

ordercolumn: 利用 *ORDER BY* 选项，可以对将被删除的数据先进行排序。*ORDER BY* 选项只有在与 *LIMIT* 选项联合使用的时候——例如在只需要删除前 10 条和后 10 条记录的场合（依据某些排序原则）——才有意义。

maxrecords: LIMIT 选项的参数值，本次删除操作最多可以删除多少条记录。

如果这种 *DELETE* 命令里没有 *WHERE* 条件子句，那么数据表的所有记录都将被删除（所以要特别小心）。没有 *WHERE* 条件子句的 *DELETE* 命令不能在事务里使用。如果已经打开了一个事务，那么在 *DELETE* 命令执行之前，MySQL 会自动执行一条 *COMMIT* 命令来结束这个事务。假如希望彻底删除一个大数据表，使用 *TRUNCATE* 命令会更有效。

```
DELETE [deleteoptions] table1, table2 ... FROM table1, table2, table3 ...
[USING columns]
WHERE conditions
```

DELETE 命令的这种变体（从 4.0 版本开始）从数据表 *table1*、*table2* 等里面删除记录，而其他数据表（如 *table3* 等）里的数据被认为是搜索条件。

注意，必须把将要从中删除数据的数据表全部列在关键字 *DELETE* 的后面，列在关键字 *FROM* 后面的数据表则是本次 *DELETE* 操作所涉及的所有数据表——负责构成搜索条件的数据表也包括在内。

deleteoptions: 在这里，可以像对待普通 *DELETE* 命令那样定义各种选项。

columns: 在这里列出各数据表的关联字段（参阅 *SELECT* 命令）。这里需要假设关联字段在两个数据表里的名字是一样的。

conditions: 除了通常的删除准则，在这里可以定义连接子句（例如，*WHERE table1.id = table2.foreignID*）。

```
DESCRIBE tablename [ columnname ]
```

DESCRIBE 返回有关当前数据库中指定的数据表的信息（或是关于这个数据表中某一个特殊的数据列的信息）。除了 *columnname* 选项，这里也可以使用包含着通配符 “_” 和 “%” 的模式。这时，*DESCRIBE* 显示的信息是与这个模式相匹配的那些数据列。*DESCRIBE* 返回的信息与命令 *EXPLAIN* 和 *SHOW COLUMN* 相同。

DO 是 *SELECT* 命令的一个变量，并与其有基本相同的语法。两者之间的区别是命令 *DO* 不返回求值结果。例如，命令 *DO* 可以用来对变量进行赋值，这时它会比 *SELECT* 命令快一点（例如，*DO@var:=3*）。

DROP DATABASE [IF EXISTS] dbname

DROP DATABASE 命令删除一个现有数据库以及它的所有数据。这种删除无法恢复，所以一定要特别小心！如果这个数据库不存在，将看到一条出错消息。使用选项 *IF EXISTS* 可以避免这个错误。

在执行完该命令之后，*dbname* 目录里有着（但不限于）以下这些文件扩展名的所有文件都将不复存在：*.BAK*、*.DAT*、*.HSH*、*.ISD*、*.ISM*、*.MRG*、*.MYD*、*.MYI*、*.db*、*.frm* 以及文件 *db.opt*。

DROP FUNCTION fname

DROP FUNCTION 命令删除一个指定的存储过程，或者是禁用一个此前用 MySQL 的 *CREATE FUNCTION* 命令导入的外部辅助函数。

DROP INDEX indexname ON tablename

DROP INDEX 命令从给定的数据表里删除一个索引，*indexname* 是这个索引的名字（它通常与被索引数据列一样）或 *PRIMARY*（这是主索引的名字）。

DROP PROCEDURE pname

DROP PROCEDURE 命令删除指定的存储过程。

DROP [TEMPORARY] TABLE [IF EXISTS] tablename1, tablename2 ... [options]

DROP TABLE 命令用于删除指定（临时）的数据表，这种删除是不可恢复的。选项 *IF EXISTS* 避免由于数据表不存在而出现的错误信息。

注意，*DROP TABLE* 命令自动结束一个运行中的事务（*COMMIT*）。

DROP TEMPORARY TABLE（从 MySQL 4.1 版本开始）仅删除临时数据表。与普通的 *DROP TABLE* 命令相比较，该命令对正在运行中的事务没有影响。

options: RESTRICT|CASCADE

RESTRICT 和 *CASCADE* 这两个选项目前还都没有实际的作用，它们的设计用途是简化把 MySQL SQL 代码移植到其他品牌的数据库系统的工作。

DROP TRIGGER tablename.triggername

DROP TRIGGER 命令删除指定的触发器。

DROP VIEW [IF EXISTS] viewname1, viewname2 ... [options]

DROP VIEW 命令删除指定的视图，这种删除是不可恢复的。这里可以使用与 *DROP TABLE* 命令相同的选项。

EXPLAIN tablename

EXPLAIN 命令返回到一个数据表，这个数据表具有数据表里所有数据列的信息（字段名字、字段类型、索引、默认值等）。如果使用 *SHOW COLUMNS* 或是 *DESCRIBE* 命令，也可以得到同样的信息。还可以通过使用外部程序，如 *mysqlshow* 程序，来得到同样的信息。

EXPLAIN SELECT selectcommand

EXPLAIN SELECT 命令将返回一个表格，其内容是关于这条 *SELECT* 命令执行细节的信息。这些数据有助于对索引进行速度优化，特别是在决定数据表的哪些数据列应当被索引方面很有帮助。（*selectcommand* 的语法在 *SELECT* 命令下给出。*EXPLAIN SELECT* 命令的使用示例以及有关返回数据表的简要描述请参阅第 8 章。）

FLUSH flushoptions

FLUSH 命令可以用来清空 MySQL 内部使用的各种缓存区，也就是把此前还没有实际写入数据库的所有信息写入数据库里。执行 *FLUSH* 命令必须具备 *RELOAD* 权限。

flushoptions: 在这里，可以指定哪一个缓存区需要被清空。多个选项应该用逗号隔开。

DES_KEY_FILE: 重新加载供 *DES_ENCRYPT* 和 *DES_DECRYPT* 函数使用的密钥文件。

HOSTS: 清空主机缓存区数据表。如果在本地网络里，IP 数值的安排发生改变的时候，这样做是必需的。

LOGS: 先关闭所有的日志文件，然后再重新打开它们。对于变更日志 (*update log*)，这将创建一个新的日志文件，文件末尾要增加一个数值 1 (*name.000003→name.000004*)。对于错误日志 (*error log*)，这将把现有文件重命名为 *name.old*，然后创建一个新的错误日志文件。

QUERY CACHE: 对查询缓存区进行碎片整理，以便更有效地使用它的记忆存储器。缓存区没有清空。（如果想彻底清除这个缓存区，就需要使用 *RESET QUERY CACHE* 命令。）

PRIVILEGES: 重新加载权限数据库 *mysql*（对应于 *mysqladmin reload* 程序）。

STATUS: 设置初始状态变量为 0。

TABLES: 关闭所有打开的数据表。

TABLE[S] tblname1, tblname2, ... : 关闭指定的数据表。

TABLES WITH READ LOCK: 同上面一样，但此时将先对所有的数据表进行锁定，这种锁定将一直持续到明确地发出一条相应的 *UNLOCK* 命令为止。

USER_RESOURCES: 对 *MAX_QUERIES_PER_HOUR* 重置账户。

MAX_UPDATE_PER_HOUR 和 *MAX_CONNECTIONS_PER_HOUR*（参见 *GRANT* 命令里的 *maxlimits* 选项）。

大多数 *FLUSH* 操作还能够通过外部程序 *mysqladmin* 来执行。

```
GRANT privileges ON objects
  TO users [IDENTIFIED BY 'password']
  [REQUIRE ssloptions]
  [WITH GRANT OPTION | maxlimits]
```

GRANT 命令用来分配对各种数据库对象的访问权限，这些权限包括：

ALTER, CREATE, CREATE TEMPORARY TABLES, CREATE VIEW, DELETE, DROP, EXECUTE, FILE, INDEX, LOCK TABLE, PROCESS, REFERENCES, RELOAD, REPLICATION CLIENT, REPLICATION SLAVE, SELECT, SHOW DATABASE, SHOWVIEW, SHUTDOWN, SUPER, UPDATE

如果希望设置全部（或是不设置任何）权限，那么就需要给出 *ALL*（或 *USAGE*）。（第二个变量是很有用的，假如希望创建一个新的 MySQL 用户，而又不想给他授予任何权限的时候，就要用到这个变量。）*GRANT* 权限只能通过 *WITH GRANT OPTION* 选项来设置，也就是说 *ALL* 并不包括 *GRANT* 权限。

如果某些权限只是应用于一个数据表里的某些数据列，那么就把这些数据列放入圆括号内。例如，可以规定 *GRANT SELECT(columnA, columnB)*。

objects: 负责给出数据库和数据表。这里可以使用以下几种语法变体:

<i>databasename.tablename</i>	仅在这个数据库的这个数据表
<i>databasename.spname</i>	仅在这个存储过程
<i>databasename.*</i>	这个数据库的所有数据表
<i>tablename</i>	当前数据库的这个数据表
*	当前数据库的所有数据表
**	全局权限

MySQL 不允许数据库的名字里有通配符。

users: 允许列出一个或多个（需用逗号分隔）用户。如果这些用户在 *users* 数据表里尚不存在，则创建它。这里可以使用以下几种语法变体:

<i>username@hostname</i>	主机 <i>hostname</i> 上的这个用户
' <i>username'@'hostname'</i>	同上，但允许名字里包含特殊字符
<i>username</i>	所有计算机上的这个用户
"@ <i>hostname</i>	<i>Hostname</i> 的所有用户
"	所有计算机上的所有用户

password: 使用 *IDENTIFIED BY* 选项，可以对一个指定账户设置明文密码。在 *GRANT* 进入到 *user* 数据表之前，它使用函数 *PASSWORD* 设置密码。如果指定了多个用户，则需要给出多个密码:

TO user1 IDENTIFIED BY 'pw1', user2 IDENTIFIED BY 'pw2', ...

ssloptions: 如果与 MySQL 的连接是加密的 SSL 或者是用户标识需要 X509 证书，可以在这里为建立连接规定要求的信息。语法如下所示:

REQUIRE SSL |X509 [ISSUER'iss'] [SUBJECT'subj'] [CIPHER'ciph']

REQUIRE SSL 意味着连接方式必须是加密的 SSL 连接（因此，普通连接是不允许的）。*REQUIRE X509* 意味着用户必须拥有能够满足 X509 标准要求的有效标识证书。

ISSUER 选项指定了要求的证书的签发者。（如果没有 *ISSUER* 选项，证书的原件不被认可。）

SUBJECT 指定了证书的 *subject* 字段所要求的内容。（如果没有 *SUBJECT* 选项，证书的内容不被认可。）

CIPHER 选项指定了要求的 SSL 加密算法。（SSL 支持各种算法。如果没有这项说明，那么所有的算法都是允许的，包括那些可能存在安全漏洞的旧的算法。）

maxlimits: 在这里，可以设置允许有关用户每小时最多建立多少次连接、发出多少条 *SELECT* 命令、发出多少条 *INSERT/UPDATE/DELETE* 命令。这 3 个值的默认设置都是 0（意思是没有任何限制）：

MAX_QUERIES_PER_HOUR n

MAX_UPDATES_PER_HOUR n

MAX_CONNECTIONS_PER_HOUR n

如果指定的用户尚不存在，并且 *GRANT* 命令不带 *IDENTIFIED* 选项，那么新的用户就没有密码（这是有安全风险的）。从另外一个方面来看，如果用户已经存在，不带 *IDENTIFIED* 选项的 *GRANT* 不改变密码（这意味着现有的密码不会被 *GRANT* 命令误删误改）。

用 *GRANT* 命令来删除已经授予的权限是不可能的（例如，用较小的权限集再一次执行该命令）。如果希望收回某项权限，就必须使用 *REVOKE* 命令。

GRANT 命令只能由具备 *Grant* 权限的用户使用，有权执行 *GRANT* 命令的用户只能把他自己拥有的权限授予给其他人。如果在启动 MySQL 服务器时使用了 *safe-user-create* 选项，创建一个新用户的操作将不仅需要具备 *Grant* 权限，还必须同时具备 *mysql.user* 数据表上的 *Insert* 权限。

从 MySQL 5.0.3 版本开始，只有具备 *Create User* 权限的用户才能使用 *GRANT* 命令来创建新用户。

```
HANDLER tablename OPEN [AS aliasname]
HANDLER tablename READ FIRST|NEXT [ WHERE condition LIMIT n, m ]
HANDLER tablename READ indexname FIRST|NEXT|PREV|LAST [ WHERE ... LIMIT ... ]
HANDLER tablename CLOSE
```

从 MySQL 4.0.3 版本开始，使用 *HANDLER* 命令可以直接访问 MyISAM 和 InnoDB 类型的数据表，与 *SELECT* 命令相比较，*HANDLER* 命令的效率更高。特别是在一次只处理一个记录或是小的记录组的时候，这一点尤为明显。

使用该命令很简单：只要先用 *HANDLER OPEN* 打开一个数据表，就可以随意执行多少次 *HANDLER READ* 命令了。一般来说，第一次执行 *HANDLER READ* 命令时应该使用选项 *FIRST*，然后用 *NEXT* 选项前进到下一条记录，直到处理工作结束或到达最后一条数据记录为止。该命令的返回结果与 *SELECT **（所有数据列）一样。*HANDLER CLOSE* 结束本次访问。

HANDLER tablename READ 按照记录存储的顺序读取记录。从另外一个方面讲，变量 *HANDLER tablename READ indexname* 使用指定的索引。如果想使用主索引，那就必须使用 *'primary'*（注意，是反引号）格式。

HANDLER 命令不太适合用在通用型的 MySQL 应用程序里，因为用它编写出来的代码在兼容性方面不是很好。*HANDLER* 更适合用来编写专业化的底层工具（例如，通过光标去访问有关数据的备份工具或驱动程序等）。注意，*HANDLER* 命令不对数据表进行锁定，因此，在使用 *HANDLER* 读取某个数据表里的数据的同时，其他用户还可以修改这个数据表里的数据。

HANDLER 命令不应该用在存储过程里，替代的方法是使用光标。*HANDLER* 命令与 *DECLARE* *HANDLER* 命令没有任何瓜葛，后者的用途是在存储过程里定义出错处理代码。

```
HELP
HELP contents
HELP functionname
```

从 MySQL 4.1 版本开始，*HELP* 返回一个简明的帮助文本。也可以使用其缩写符号?来代替 *HELP* 命令。

```
INSERT [options1] [INTO] tablename [(columnlist)]
VALUES (valuelist1), (...), ... [options2]

INSERT [options1] [INTO] tablename
SET column1=value1, column2=value2 ... [options2]

INSERT [options1] [INTO] tablename [ (columnlist) ]
SELECT ...
```

INSERT 命令的用途是往一个现有的数据表里插入新的记录。它有 3 种主要的语法变体。第一种（也是最常

用的一种)是把新的数据记录指定在括号中。因此,典型的 *INSERT* 命令如下所示:

```
INSERT INTO tablename (columnA, columnB, columnC)
VALUES ('a', 1, 2), ('b', 7, 5)
```

结果是把两条新记录插入到了数据表中。允许包含 *NULL* 值的数据列、有默认值的数据列以及将由 MySQL 通过 *AUTO_INCREMENT* 机制自动赋值的数据列可以不在 *columnist* 部分出现。如果在 *columnist* 部分没有给出任何一个数据列的名字,就必须在 *VALUES* 选项里按照所有数据列的个数和顺序给出足够多的数据值。

INSERT 命令的第二种变体只能改变一条记录(不是同时改变几个)。这样的命令如下所示:

```
INSERT INTO tablename SET columnA='a', columnB=1, columnC=2
```

至于 *INSERT* 命令的第三种变体,被插入的数据来自一条 *SELECT* 指令。

options1: 该命令的动作可以由下面的选项来控制:

IGNORE 选项的作用是,对于 *UNIQUE KEY* 数据列,忽略已经存在的插入记录。(如果没有这个选项,会导致错误信息。)

LOW_PRIORITY | *DELAYED* | *HIGH_PRIORITY* 在插入操作进行过程中具有影响。

在 *LOW_PRIORITY* 和 *DELAYED* 选项中,MySQL 要延迟它的存储操作,直到没有正在对数据表的读取访问。选项 *DELAYED* 的作用在于 MySQL 立即返回 *OK*,客户不需要等到存储操作的结束。但是,如果需要使用 *LAST_INSERT_ID()* 函数去确定 *AUTO_INCREMENT* 编号值,就不能够使用 *DELAYED* 选项。如果数据表已被 *LOCK* 命令锁定,也不能够使用 *DELAYED* 选项(这是因为执行 *INSERT DELAYED* 命令需要 MySQL 打开一个新的线程,但已被锁定的数据表已经占用了有关资源,MySQL 无法打开一个新线程)。

要插入的记录储存在 RAM 中,一直到插入操作实际被运行。如果由于某些原因(崩溃、断电),MySQL 要终止运行,那么数据会丢失。

HIGH_PRIORITY 通常没有什么用处,即 *INSERT* 命令插入数据的速度很快。*HIGH_PRIORITY* 起作用的唯一场合是在服务器开始时是使用的选项 *low-priority-updates*。这个选项的效果是 *INSERT* 命令将以 *LOW_PRIORITY* 模式存储。*HIGH_PRIORITY* 选项可以覆盖这项默认设置。

options2: ON DUPLICATE KEY UPDATE column1=value1, column2=value2 ...

在新数据的插入过程中,如果与 *UNIQUE* 或 *PRIMARY* 索引发生冲突,这个选项(从 MySQL 4.1 版本开始)的效果是现有记录将被替换为新记录。比如说,如果 *id* 是一个 *PRIMARY* 数据列并且 *id=1* 的记录项已经存在,那么

```
INSERT INTO tablename (id, data) VALUES (1, 10)
```

```
ON DUPLICATE KEY UPDATE data=data+10
```

与下面的命令具有相同的效果:

```
UPDATE tablename SET data=data+10 WHERE id=1.
```

这里还可以使用 *VALUE(columnname)* 函数来指定将被修改的数据列。这个函数返回相应数据列的值。在使用一条 *UPDATE* 指令修改多条记录的时候,这个函数会很有用,如下所示:

```
INSERT INTO tablename (id, data) VALUES (1, 10), (2, 15)
ON DUPLICATE KEY UPDATE data=data+VALUE(data)
```

注意,在使用默认值的时候,对于 *TIMESTAMP* 和 *AUTO_INCREMENT* 编号值有特殊的规则。(这些规则也对 *UPDATE* 有效。)

口 有默认值的数据列: 如果想让 MySQL 对一个数据列使用默认值,方法是要么在 *INSERT* 命令中不定义这个数据列,要么把一个空白字符串(不是 *NULL*)作为它的值:

```
INSERT INTO table (col1, col2_with_default_value) VALUES ('abc', '')
```

- **TIMESTAMP 数据列:** 如果想让 MySQL 在数据列中插入当前时间，办法之一是在 *INSERT* 命令里不写出这个数据列，办法之二是把 *NULL* (不是一个空白字符串) 作为它的值。如果需要存入一个特定的时间戳值，把这个时间戳值的字符串传递给 *INSERT* 命令即可。
- **AUTO_INCREMENT 数据列:** 这里也一样，要么不写出这个数据列，要么传递 *NULL* 值 (使用这个办法的大前提是 *AUTO_INCREMENT* 编号值将由 MySQL 自动生成)。也可以明确地把一个编号值——当然，这个编号值必须是还没有用过的——传递给 *INSERT* 命令。

JOIN

JOIN 实际上不是一个 SQL 命令。这个关键字更多情况下是作为 *SELECT* 命令的一个部分来使用，作用是从几个数据表里连接数据。*JOIN* 的具体讨论见后面对 *SELECT* 命令的说明。

KILL threadid

该命令终止一个指定的 MySQL 服务器线程 (子进程)。只有拥有 *Super* 权限的用户才可以使用该命令。通过 *SHOW PROCESSLIST* (必须具有 *Process* 权限才能够使用该命令) 可以得到正在运行的线程清单。线程也可以通过使用外部程序 *mysqladmin* 来终止。

```
LOAD DATA [ loadoptions ] INFILE 'filename' [ duplicateopt ]
    INTO TABLE tablename
    [ importopt ]
    [ IGNORE ignorenr LINES ]
    [ (columnlist) ]
```

LOAD DATA 命令读取一个文本文件，并把其中的数据作为数据记录逐行插入到数据表中。*LOAD DATA* 插入数据的时候要比使用一组 *INSERT* 命令快许多。

一般来说，文件 *filename* 几乎总是来自 MySQL 服务器主机的文件系统。(读取 MySQL 服务器主机的文件系统需要具备 *File* 权限。出于信息安全方面的考虑，这个文本文件要么必须来自数据库目录，要么必须对这台计算机上的全体用户可读。)

如果输入的文本文件具有 ASCII 字符集以外的字符，那么在使用 *LOAD DATA* 命令之前，必须要用 *SET NAMES* 命令来设置文本的字符集。

loadoption: LOCAL 选项的作用是，本地客户端计算机上的文件 *filename* 被读取 (也就是说，是在命令 *LOAD DATA* 执行的计算机上读取，而不是服务器上的计算机)。这时，不需要 *FILE* 权限。*(FILE* 权限仅与 MySQL 服务器计算机的文件系统有关。) 注意，*LOAD DATA LOCAL* 可以被禁止，这取决于 MySQL 服务器是如何编译及配置的 (选项 *local-infile*)。

LOW_PRIORITY 选项的作用是，只有当没有其他用户对数据表进行读取的时候，数据才插入到数据表中。

CONCURRENT 选项的作用是，在向数据表插入数据的同时，允许其他客户读取数据。但是，只有 MyISAM 类型的数据表具有这个功能，而且，只有在新的数据是明确地插入到数据表文件的末端的时候才可以起作用。数据表文件中不允许包含任何未使用内存 (内存空洞)。可以使用 *OPTIMIZE TABLE* 命令来保证这一点。

filename: 如果一个文件名没有给出路径，那么 MySQL 就在当前数据库的目录中搜索这个文件 (如，'*bulk.txt*')。

如果这个文件名包含着一个相对路径，MySQL 将把它解释为相对于 *data* 目录的相对路径 (例如：'*mydir/bulk.txt*')。

具有绝对路径的文件名不做任何变化地被采用（例如，'tmp/mydir/bulk.txt'）。

duplicateoptions: 当一个新的数据记录与一个已经存在的记录有相同的 *UNIQUE* 或是 *PRIMARY KEY* 的值的时候，*IGNORE* | *REPLACE* 选项决定了 MySQL 的动作。使用 *IGNORE* 选项，这个现有记录被保留，新的记录被忽略。使用 *REPLACE* 选项，已经存在的记录被新的记录所取代。如果这两个选项都没有使用，则会导致出错信息。

importoptions: 这个选项控制着应该如何对将被导入的文件里的数据格式做出解释。完整的 *importoptions* 选项语法如下所示：

```

| FIELDS
|   | TERMINATED BY 'fieldtermstring'
|   | ENCLOSED BY 'enclosechar'
|   | ESCAPED BY 'escchar'
|   | LINES TERMINATED BY 'linetermstring'

```

fieldtermstring 选项负责给出用来分隔同一数据行里的各个数据列的字符串（即数据列分隔符；比如一个制表符）。

enclosechar 负责给出文本文件中各条记录之前和之后应该具有的字符（即数据行分隔符；通常情况下是用于字符串的单引号或双引号）。如果一个记录项是由这个字符开始的，那么这个字符就从开始和末尾被删除。那些不是由 *enclosechar* 字符开始的条目仍然会接受。因此，文本文件中字符的使用在某种程度上说是可选的。

escchar 选项负责给出哪个字符是用来标识特殊字符的（即转义前导字符，通常是反斜线字符）。假如特殊字符出现在文本文件的字符串中，它们还被用于分隔数据列和数据行，这样做是必需的。（更进一步说，如果一个字符被定义为 *escchar*，在反斜线需要由 *escchar* 替换的地方，MySQL 认为格式 “\0” 为零字节。）

linetermstring 负责给出用来结束每个文本行的字符串（即文本行结束符）。在 DOS/Windows 环境下，这种文本文件必须以字符串 “\r\n” 来结束每一行。

在上面提到的这 4 个字符串里都允许出现特殊字符，但必须像下面这样写出：

\0	零值字节
\b	退格符
\n	换行符
\r	回车符
\s	空格符
\t	制表符
'	单引号 (')
"	双引号 ("")
\\"	反斜线符

此外，字符串还可以以十六进制格式给出（如，用 0x22 来代替 “\”）。

如果没有给出任何字符串，则使用如下所示的默认设置：

FIELDS TERMINATED BY '\r' ENCLOSED BY '' ESCAPED BY '\\'

LINES TERMINATED BY '\r\n'

ignorenr: 这个值规定了在文本文件的开头将有多少个文本行被丢弃。假如第一个文本行包含着数据表的标题，这将会非常有用。

columnlist: 如果文本文件里数据列的排序不是与数据表中的数据列严格对应，那么可以在这里规定哪些文件的数据列对应着哪些数据表的数据列。数据列里的清单必须是在圆括号内设置：例如，(firstname,

lastname, birthdate)。

如果在输入过程中没有考虑 *TIMESTAMP* 数据列，或者插入了 *NULL* 值，那么 MySQL 就插入实际的时间。使用 *AUTO_INCREMENT* 数据列，MySQL 给出类似的动作。

作为其输出结果的一部分，*LOAD DATA* 命令将显示一个整数，这个数字是在本次数据导入过程中引起的警告消息的总次数。

从 MySQL 4.1 版本开始，可以使用命令 *SHOW WARNINGS* 和 *SHOW ERRORS* 去查看由 *LOAD DATA* 命令引起的所有警告和错误。

还可以使用外部程序 *mysqlimport* 来取代 *LOAD DATA* 命令。这个程序创建一个指向 MySQL 的链接，然后使用 *LOAD DATA*。与 *LOAD DATA* 相反的命令是 *SELECT ... INTO OUTFILE* 命令。使用它，可以把数据表输出到一个文本文件中。进一步的信息和实例请参见第 14 章。

LOAD DATA FROM MASTER

从 MySQL 4.0 版本开始，该命令把所有 MyISAM 类型的数据表从镜像机制中的主控服务器复制到从属服务器上去。注意，*mysql* 数据库里的数据表不会被复制。在复制完成以后，从属服务器将自动进入镜像工作状态（也就是说，变量 *MASTER_LOG_FILE* 和 *MASTER_LOG_POS* 将被自动置位。它们在正常情况下必须用 *CHANGE MASTER TO* 命令来设置）。

在许多时候要用到该命令，特别是在没有使用 InnoDB 类型的数据表的时候，该命令可以很方便地对镜像机制进行设置。镜像机制的用户应该具有 *Select*、*Reload* 以及 *Super* 权限。

LOAD INDEX INTO CACHE indexspec1, indexspec2 ...

该命令把所有给定的 MyISAM 数据表里的索引加载到一个高速缓存区里。不过，这个功能极少用到，一般只在需要反复进行各种性能测试的场合才会用到它。

indexspec: 规定了要被加载的 MyISAM 类型的数据表。可以使用下面的命令：

tablename [[INDEX|KEY] (*indexname1* ...)] [IGNORE LEAVES]

目前，该命令把所有的数据表索引加载到缓存区里。因此，各个索引的详细说明只是在将来才有意义。*IGNORE LEAVES* 选项是用于索引的一部分被加载。

LOAD TABLE *dbname.tablename* FROM MASTER

如果一个数据表还没有存在于从属服务器中，那么该命令把这个镜像机制系统中的数据表从主控服务器复制到从属服务器中。该命令主要是提供给 MySQL 程序员简化代码调试工作用的，不过，在检测到错误以后，用该命令去修复镜像机制系统也很方便。该命令的执行需要镜像机制用户具有 *Select*、*Reload* 以及 *Super* 权限。*LOAD TABLE* 命令只是适用于 MyISAM 类型的数据表。

LOCK TABLE *table1* [AS *aliasname*] *locktype*, *table2* [AS *alias2*] *locktype*, ...

LOCK TABLE 命令用来防止其他的 MySQL 用户对指定的数据表进行写入或读取操作。如果一个数据表已经被另外一个用户锁定，那么该命令就延迟到锁定解除以后再执行（但 MySQL 并没有为此提供一个设置倒计时时间的设置，所以这目前只在理论上可行）。

数据表的 *LOCK*（锁定）保证了在几个命令执行的过程中，不会由其他用户对数据进行改变。在先执

行 *SELECT* 查询、再根据其查询结果使用 *UPDATE* 命令对数据表进行改变的情况下，*LOCK* 的使用是必需的。（但在单独使用 *UPDATE* 命令的情况下，*LOCK* 不是必需的。这是因为单独执行的 *UPDATE* 命令永远是由 MySQL 服务器一次执行完毕，其他用户不可能在它的执行期间有机会去修改数据。）

LOCK TABLE 命令不能够在 InnoDB 类型的数据表上使用。对于 InnoDB 类型的数据表，可以通过使用事务和命令 *SELECT ... IN SHARE MODE* 以及 *SELECT ... FOR UPDATE* 来更高效地对数据表实施锁定。

注意，*LOCK TABLE* 命令将以 *COMMIT* 方式结束一个正在运行的事务。未来的 MySQL 版本已经计划为 InnoDB 数据表提供一些可以在事务的外部执行的专用 *LOCK* 命令变体。

locktype: 在 MySQL 5.0 版本中，有以下 4 种 *LOCK* 类型。

- *READ*: 所有的 MySQL 用户可以读取数据表，但不可以对数据表做出任何更改（包括执行 *LOCK* 命令的用户在内）。只有在数据表没有由其他的 *WRITE LOCK* 阻止的时候，*READ LOCK* 才被使用。（但是，已经存在的 *READ LOCK* 命令对新的 *READ LOCK* 命令不会有任何的妨碍作用。因此，可能会出现几个用户在同一个数据表上同时使用 *READ LOCK* 命令的情况。）
- *READ LOCAL*: 同 *READ* 选项一样，但允许不会对现有数据记录做出修改的 *INSERT* 命令得到执行。
- *WRITE*: 只有当前用户可以读取和改变数据表，所有其他的用户被彻底阻止。这些用户既不能够改变锁定的数据表中的记录，也不能够对数据表进行读取操作。*WRITE LOCK* 只有在这个数据表没有被其他的 *LOCK* 命令（读或写）阻止的时候才得以使用。在 *WRITE LOCK* 解除之前，其他用户既不能使用 *READ LOCK*，也不能够使用 *WRITE LOCK* 命令。
- *LOW_PRIORITY WRITE*: 就像 *WRITE* 命令一样，所不同的是，在等待期间（即直到所有的其他 *READ* 和 *WRITE LOCK* 结束），其他用户可能会申请获得一个新的 *READ LOCK*。这也就意味着，只有在没有其他任何用户申请 *READ LOCK* 的情况下，*LOCK* 命令才能够执行。

将来的 MySQL 版本中，有可能会出现另外两个专门用于 InnoDB 类型数据表的变量。由于这两个变量目前尚未正式启用，因此，下面的相关讨论可能不一定准确。

IN SHARE MODE: 这个 *LOCK* 类型与 *SELECT * FROM table* 有同样的效果，保护整个数据表不被其他的连接所改变。（在锁定解除之前，其他连接的 *INSERT*、*UPDATE* 以及 *DELETE* 命令被阻止。）

与 *SELECT* 命令相比，这个变量有两个优点：InnoDB 数据表引擎可以高速执行该命令；*LOCK* 命令与其他的数据库系统兼容（*Oracle*、*PostgreSQL* 等）。

IN EXCLUSIVE MODE: 如果这个选项与选项 *LOCK IN SHARE MODE* 或选项 *LOCK FOR UPDATE* 一同使用，那么这种类型的 *LOCK* 会更加严格，甚至可以阻止其他连接的 *SELECT* 命令。

数据表的 *LOCK* 可以增加几个数据库命令一个接一个执行的速度（当然，这是以在这个过程中阻止其他用户为代价的）。

MySQL 数据库依靠线程来管理数据表的 *LOCK*。每一个连接与它自己的线程相关。每一个线程仅可以有一个 *LOCK* 命令。（但有可能包含几个数据表。）一旦 *UNLOCK* 或 *LOCK* 对任何其他数据表执行，所有以前的锁定就失效了。

为了提高效率，应该尽可能地让 *LOCK* 简明扼要，并尽可能迅速地使用 *UNLOCK* 命令结束它们。在当前进程结束的时候（比如在服务器和客户之间的连接意外掉线的时候），*LOCK* 将自动结束。

OPTIMIZE TABLE tablename

从 MySQL 4.1.3 版本开始，*OPTIMIZE TABLE* 命令把 MyISAM 和 InnoDB 类型的数据表中不使用的存储空间删除掉，并保证同一条数据记录里的相关数据都存储在一个连续的块里。

OPTIMIZE TABLE 命令应该定期地在那些不断改变内容（有许多 *UPDATE* 和 *DELETE* 命令）的数据表上执行。这将加快数据访问。使用 MyISAM 类型的数据表，数据库文件还可以做得更小一些。而 InnoDB

数据表的表空间原则上不能做得很小。

PROCEDURE procname

MySQL 允许人们使用外部程序来扩展它的功能。它们的代码必须用 C++ 程序设计语言来表示。要想在 *SELECT* 命令当中使用这项功能，就必须用到关键字 *PROCEDURE*。

作为这个程序的一个例子，MySQL 程序代码包含着函数 *ANALYSE*。这个程序能够用于分析数据表的内容，希望测定一个较好的数据表精确度。这个函数的调入如下：

```
SELECT * FROM tablename PROCEDURE ANALYSE()
```

编写用户定义函数 (*user-defined function*, 简称 *UDF*, 参见 *CREATE FUNCTION* 命令) 需要大量有关 MySQL 的背景知识。详细资料见 MySQL 文档：<http://dev.mysql.com/doc/mysql/en/extending-mysql.html>。

存储过程是一种更好和更简单的 UDF 替代品，这一功能始于 MySQL 5.0 版本。

PURGE MASTER LOGS TO 'hostname-bin.n'

该命令删除所有比指定文件旧的二进制日志文件。只有在确定这些日志文件不再需要的时候，也就是说，只有当所有的从属计算机与它们的数据库同步的时候，才可以执行该命令，而且该命令只是在镜像机制系统中的主控计算机上执行。执行该命令需要 *Super* 权限。请参见 *RESET MASTER* 命令。

RENAME TABLE oldtablename TO newtablename

RENAME TABLE 命令用来给一个现有数据表起一个新名字。它也可以一次重新命名多个数据表，就像 *a TO b, c TO d, ...* 这样。

没有命令能够对整个的数据库给出一个新的名字。如果使用的是 MyISAM 类型的数据表，那就可以按照下面的方法进行：先停止 MySQL 服务器，重命名数据库目录，然后再重新启动服务器。注意，可能不得不改变 *mysql* 数据库的访问权限。使用 InnoDB 类型的数据表，必须做一个备份 (*mysqldump*)，然后再把数据表输入到一个新的数据库中。

REPAIR TABLE tablename1, tablename2, ... [TYPE = QUICK]

REPAIR TABLE 命令尝试修复一个受损的数据表文件。如果使用了选项 *TYPE = QUICK*，则只有索引被再一次创建。

REPAIR TABLE 命令只能用于 MyISAM 类型的数据表。如果不使用该命令，也可以使用外部程序 *myisamchk -r tblfile*。（如果 *REPAIR TABLE* 命令的返回值不是 *OK*，那么可以尝试程序 *myisamchk -o*。这个程序比 *REPAIR TABLE* 命令更具修复的可能性。）

REPLACE [INTO]

REPLACE 与 *INSERT* 命令很相似。唯一的区别是在如何处理那些关键字与现有记录的关键字发生重复的新记录方面。在这种情况下，现有记录被删除，新的记录储存在数据表里。由于镜像机制的行为已经明确定义，所以 *REPLACE* 命令没有 *IGNORE* 选项，而 *INSERT* 命令是有 *IGNORE* 选项的。

RESET MASTER

该命令删除包含索引文件 `hostname-bin.index` 的所有二进制日志文件。使用该命令，可以在一个特定的时间里重新启动镜像。因此，*RESET SLAVE* 必须在所有的从属系统上执行。在开始执行该命令之前，必须保证从属系统上的数据库唯一对应于主控系统上的数据库。该命令的使用需要 *reload* 权限。

如果只想删除旧的（不再需要的）文件，那么使用命令 *PURGE MASTER LOGS*。

RESET QUERY CACHE

该命令删除所有查询缓存区的条目。它需要具有 *reload* 权限。

RESET SLAVE

该命令将重新初始化镜像机制中的从属系统。`master.info` 文件的内容（以及当前日志文件和它的位置）将被删除。执行该命令必须具备 *reload* 权限。

该命令只是在下面的情况下才有意义：假如出现问题以后，数据库要建立在基于以前映射的从属系统上，使得从属系统可以通过镜像与它自身同步；或者是当 *RESET MASTER* 命令在主控系统上执行（因此，所有的日志文件在那里被删除）的时候。在这种情况下，在从属系统上应该是先执行 *SLAVE STOP* 命令，然后再执行 *SLAVE START* 命令。

RESTORE TABLE *tblname* FROM '/backup/directory'

RESTORE TABLE 命令把指定数据表的文件从备份目录中复制到当前数据库的数据目录中。*RESTORE TABLE* 命令与 *BACKUP TABLE* 命令的意思相反。

BACKUP 和 *RESTORE* 命令不能用于 InnoDB 类型的数据表。这两个命令已经有点儿过时了，所以应该尽量避免使用它们。它们的替代品是 `mysqldump`、`mysqlhotcopy` 等外部备份工具以及 InnoDB 数据表的备份工具程序。

REVOKE *privileges* ON *objects* FROM *users*

REVOKE 命令的用途刚好与 *GRANT* 命令相反。该命令可以收回以前授予的权限。该命令中的 *privileges*、*objects* 和 *users* 参数的用法与 *GRANT* 命令中的有关参数一样。唯一的区别体现在 *Grant* 权限上：从用户那里收回这个权限的 *REVOKE* 命令是 *REVOKE GRANT OPTION ON ... FROM ...*。

GRANT 命令把新的用户插入到 `mysql.user` 数据表里，但 *REVOKE* 命令是不能删除这个用户的。可以使用 *REVOKE* 命令把这个用户的所有权限撤回，但是不能不让这个用户建立与 MySQL 的连接。（如果的确想把用户的这个权利也剥夺掉，必须通过 *DELETE* 命令从 `user` 数据库中把这个条目彻底删除。）

注意，在 MySQL 的访问系统里，不能够禁止更高一级所允许的权限。如果允许用户 *x* 访问数据库 *d*，那么就不能使用 *REVOKE* 命令来拒绝该用户访问数据表 *d.t*。如果想允许用户 *x* 访问数据库 *d* 中除了数据表 *t* 以外的所有数据表，那么必须要先禁止该用户对整个数据库的访问，然后再允许他对数据库中各个数据表的访问（除了数据表 *t*）。*REVOKE* 命令还没有聪明到可以自行完成这些操作的程度。

ROLLBACK

ROLLBACK 命令将撤销最近一个事务。（与 *ROLLBACK* 命令相对的是 *COMMIT* 命令，后者将确认最近一个事务并把修改永久性地写入数据库。*BEGIN/COMMIT/ROLLBACK* 只能用于支持事务的数据表类型。关于事务的更多资料和示例请参见第 10 章。

ROLLBACK TO SAVEPOINT name

ROLLBACK 结束当前事务。在给定 *SAVEPOINT* 之后执行的所有 SQL 命令都将被撤销，而在给定 *SAVEPOINT* 之前执行的命令将被接受。该命令始见于 MySQL 4.0.14 版本。

SAVEPOINT name

该命令始见于 MySQL 4.0.14 版本，我们可以用它在事务里设置一个折返点标记，从这个位置开始的后续事务操作可以用 *ROLLBACK TO SAVEPOINT* 命令撤销。*SAVEPOINT* 命令仅在这个事务内有效。在事务结束的时候，它们将被清除。

```
SELECT [selectoptions] column1 [[AS] alias1], column2 [[AS] alias2] ...
[ FROM tablelist ]
[ WHERE condition ]
[ GROUP BY groupfield [ASC |DESC] ]
[ HAVING condition ]
[ ORDER BY ordercolumn1 [DESC], ordercolumn2 [DESC] ... ]
[ LIMIT [offset,] rows ]
[ PROCEDURE procname]
[ LOCK IN SHARE MODE | FOR UPDATE ]
```

SELECT 命令用于数据库查询。它以表格的形式返回查询结果。*SELECT* 命令的常见形式如下所示：

SELECT column1, column2, column3 FROM table ORDER BY column1

SELECT 命令有许多语法变体，有些变体可以用来处理简单的表达式，如下所示：

SELECT HOUR(NOW())

注意，*SELECT* 命令的各组成部分必须按以下顺序给出：

□ *selectoptions*: 该命令的动作可以由许多的选项来控制。

DISTINCT | *DISTINCTROW* 选项规定了当查询返回几个相同的记录的时候，MySQL 应该如何处理。*DISTINCT* 和 *DISTINCTROW* 的含义是内容重复的结果记录只显示一次。*ALL* 的含义是显示所有的记录（这是默认设置）。

从 MySQL 4.1 版本开始，还可以为 *DISTINCT* 选项设置一种排序方式（这也是比较两个字符串是否相等的基础）：语法是 *DISTINCT column COLLATE collname*。

HIGH_PRIORITY 选项的效果是优先执行比数据修改和数据插入命令的优先级更高的查询命令。*HIGH_PRIORITY* 选项应该只是用于那些需要很快执行的查询。

SQL_SMALL_RESULT | *SQL_BIG_RESULT* 控制着查询结果集是大还是小，这有助于 MySQL 的优化处理。这两个选项只是在具有 *GROUP BY* 和 *DISTINCT* 的查询中使用。

SQL_BUFFER_RESULT 选项的作用是把查询结果储存在一个临时的数据表中。如果对查询结果进行处理需要花费较长的时间，就应该使用这个选项以避免在此期间数据表被长时间锁定的问题。

SQL_CACHE 和 *SQL_NO_CACHE* 规定了 *SELECT* 命令的结果是否存储于缓存区中或者这样的存储是否应该被防止。默认情况下是 *SQL_CACHE* 设置，除非查询缓存区是依照要求的模式执行 (*QUERY_CACHE_TYPE=2*)。

SQL_CALC_FOUND_ROWS 选项的作用是，即使使用 *LIMIT* 限制了结果，MySQL 仍然测定出被发现的记录总数。然后可以使用第二个查询命令 *SELECT FOUND_ROWS()* 来获得这个数值。

STRAIGHT_JOIN 选项的作用是，从不止一个数据表的查询中采集的数据应该按照 *FROM* 表达式的先后顺序排列。(如果没有 *STRAIGHT_JOIN* 选项，MySQL 会自己尽可能地找到最优的排列顺序。*STRAIGHT_JOIN* 选项绕过了这个优化运算。)

□ *column*: 通常在这里给出数据列的名字。如果查询包含几个数据表，那么格式就是 *table.column*。

如果查询包含着由 *FROM* 表达式指定的数据表的所有数据列，那么可以简单地使用*来节省打字的时间。(但注意，如果不想要所有的数据列，这样做会降低执行的效率。)

除了数据列的名字，还可以使用多种表达式或函数：比如说，对输出内容进行排版(*DATE_FORMAT(...)* 函数)、对表达式进行计算(*COUNT(...)*函数)等。

使用 *AS*，可以给数据列一个新的名字。特别是在用到函数的时候，这是一个很实用的方法。例如，*HOUR(column) AS hr*。这里的关键字 *AS* 可以省略，即 *HOUR(column) hr* 从语法上来说是正确的(但可读性显然要差一些)。

在 *SELECT* 命令其余的大部分地方，可以使用别名(如，*ORDER BY hr*)。但别名不能够用于 *WHERE* 子句中。

从 MySQL 4.1 版本开始，可以使用 *COLLATE* 选项来设置所需要的排序方式(如，*column COLLATE collname AS alias*)。

□ *tablelist*: 在最简单的情况下，对于查询中的所有数据表可以简单列出(用逗号分隔)。如果没有关联条件(参见下面的 *WHERE*)，MySQL 返回的清单是一个所有相关数据表数据记录的所有可能组合。

有可能在这里规定用以关联数据表的子句，下面的形式是几个例子：

```
table1 LEFT [OUTER] JOIN table2 ON table1.xyID = table2.xyID
table1 LEFT [OUTER] JOIN table2 USING (xyID)
table1 NATURAL [LEFT |OUTER|] JOIN
```

有许多语法方面同义的列表可以在第 9 章当中找到，在那里，对几个数据表关联的内容有比较详细的论述。

在 *tablelist* 选项中，可以为每一个数据表给出一个别名。关键字 *AS* 是可选的：

```
table1 [AS] t1, table2 [AS] t2
```

□ *condition*: 查询结果必须满足的条件在这里出现。条件中可以包含比较(*column1>10* 或是 *column1=column2*)或模式表达式(*column LIKE '%xy'*)。可以用 *AND*、*OR* 或是 *NOT* 把几个条件连在一起。

MySQL 允许带有 *IN* 的选择条件子句：

WHERE id IN(1,2,3) 等同于 *WHERE id=1 OR id=2 OR id=3*。

WHERE id NOT IN(1,2) 等同于 *WHERE NOT (id=1 OR id=2)*。

■ **全文检索**: 条件也可以使用 *MATCH(col1,col2) AGAINST('word1, word2, word3')* 的方式来构建。

在 *col1* 和 *col2* 中对 *word1*、*word2*、*word3* 进行的全文本搜索会执行(这里假定数据列 *col1* 和 *col2* 的全文本索引已经创建)。

AGAINST 选项也支持布尔表达式的搜索，例如格式 *AGAINST('+word1 +word2 -word3' IN BOOLEAN MODE)* 就支持该搜索。这里的加号表示逻辑 *AND* 操作，减号表示指定的字可能在记录中不出现。搜索表达式的全部语法参见第 10 章。

■ **WHERE 与 HAVING**: 条件还可以使用 *WHERE* 或 *HAVING* 来构建。*WHERE* 条件子句是直接应用于 *FROM* 中命名的数据表的数据列。

HAVING 条件子句只是在 *WHERE* 条件子句后作用于查询的中间结果。*HAVING* 子句的优点在于可以对

函数的结果规定条件（例如，在 *GROUP BY* 查询中的函数 *SUM(column1)*）。别名能够在 *HAVING* 条件子句中使用 (*AS xxx*)，但别名不能够在 *WHERE* 条件子句中使用。

如果能够使用 *WHERE* 子句或 *HAVING* 子句构建出相同的条件，则应该使用 *WHERE* 子句来表示，因为这样可以得到更好的优化。

- *groupfield*: 使用 *GROUP BY*，可以定义一个组的数据列。如果对于这个组的数据列，查询结果返回几个具有相同值的记录，那么这些记录被收录进一个新的单个的记录。通常情况下，在 *column* 部分与 *GROUP BY* 一起使用的是所谓的统计函数，这些统计函数的计算可以在归组字段上完成（如 *COUNT*、*SUM*、*MIN*、*MAX*）。

在默认的情况下，归组结果是按照数据列的 *ORDER BY* 定义来排序的。作为可选项，排序方式也可以由 *ASC* 或 *DESC* 来决定。

从 MySQL 4.1 版本开始，可以使用 *COLLATE* 选项来设置一种排序方式（如，*GROUP BY column COLLATE collname*）。

- *ordercolumn*: 使用 *ORDER BY* 子句，可以对多个数据列或表达式进行排序。通常情况下，排序操作按递增顺序进行（A、B、C、…或 1、2、3、…）。如果使用了选项 *DESC*（降序），则按照递减顺序排序。
- *[offset,] row*: 使用 *LIMIT* 选项，查询结果可以减少到任意规定的数目。在查询结果需要以页面方式显示或是需要限制结果记录数目的时候，这个选项是非常有用的。查询结果开始显示的位置由 *offset* 给出（0 表示第一个数据记录），而 *row* 规定了显示出的结果记录中的最多数据行。
- *procname*: 这个选项实现指定用户程序的调用（参见前面对 *PROCEDURE* 命令的说明）。

MySQL 不支持 *SELECT ... INTO table* 的命令格式，尽管其他的许多 SQL 语言都可以识别这个格式。对于 MySQL 数据库系统，在许多情况下，可以使用 *INSERT INTO ... SELECT* 或 *CREATE TABLE tablename ... SELECT ...* 来弥补这一点。

1. 锁定 (InnoDB Tables)

如果在使用事务，那么使用 *LOCK IN SHARE MODE* 选项，可以把所有 *SELECT* 命令找到的记录用 *shared lock* 选项锁止，直到事务完成。这样做有两个用处：第一，只有在有可能影响到查询结果的事务进行完以后，*SELECT* 查询才会执行；第二，只有在事务结束以后，相应的记录才有可能被其他连接所改变（尽管在此之前，它们可以由 *SELECT* 命令来读取）。

FOR UPDATE 的使用更加严格，它是使用 *exclusive lock* 选项来锁止找到的记录。与 *shared lock* 选项相比较，其他执行 *SELECT ... LOCK IN SHARE MODE* 的连接必须要等到事务结束以后（当然，只有在相同的记录被 *SELECT* 查询影响到的时候）。

2. 子查询 (SubSELECT)

从 4.1 版本开始，MySQL 数据库开始支持所谓的 *SubSELECT*（子查询）功能。这意味着一个查询的结果可以作为另外一个查询的条件。这方面的例子请参见第 10 章。下面是子查询语法的各种变体：

SELECT ... WHERE col = [ANY / ALL / SOME] (SELECT ...)

在这个用法中，第二个 *SELECT* 查询必须返回一个单个的值（一个数据行和一个数据列）。这个值被用于比较操作符 *col = ...*。也允许使用其他的比较操作符，例如 *col > ...*、*col <=* 以及 *col <>*。

SELECT ... WHERE col = ANY / SOME (SELECT ...)

这里，第二个 *SELECT* 查询可以返回多个值。*ANY* 或同义关键字 *SOME* 产生所有满足条件的值。那么，整个的查询返回不止一个结果。*col = ANY ...* 与 *col IN* 的意义相同（参见下面）。

SELECT ... WHERE col = ALL (SELECT ...)

表达式 *comparisonoperator ALL* 在第二个 *SELECT* 查询的所有结果是真的时候，或者是第二个 *SELECT* 查询根本就没有返回结果的时候，它的值为 *TRUE*。

SELECT ... WHERE col [NOT] IN (SELECT ...)

使用这种用法，第二个 *SELECT* 查询可以返回各个独立值的一个清单。然后，这个清单可以用 *SELECT ... WHERE col IN (n1,n2,n3)* 命令来进行处理。也可以使用 *NOT IN* 来代替上面变量中的 *IN*。

SELECT ROW(value1, value2 ...) = [ANY] (SELECT col1, col2 ...)

这个查询用来检验有没有可以满足某一特定标准的记录。结果可以是 1（真）或 *NULL*（假）。作为一个比较标准，这里不是一个单一的值，而是一组值。如果结果记录满足 *ROW* 记录，那么整个查询返回 1，否则返回 *NULL*。

如果使用了可选关键字 *ANY* 或它的同义词 *SOME*，那么第二个 *SELECT* 查询可以返回多个值。假如这些值中至少有一个能够满足 *ROW* 记录，整个查询返回 1。

SELECT ... WHERE [NOT] EXISTS (SELECT ...)

使用这种用法，每一个在第一次 *SELECT* 查询中找到的记录都执行第二次查询。它返回一个数据表作为外层 *SELECT* 查询的基础。因此，外层 *SELECT* 命令不访问原先已经存在的数据表，取代这个数据表的是原先 *SELECT* 命令的结果。这类数据表被称为衍生数据表（derived tables）。SQL 语法规规定这类数据表必须使用 *AS name* 来命名。

SELECT 命令可以相互嵌套，因此，这样的命令对阅读和理解都比较困难。*SELECT* 命令也可以放在 *UPDATE* 或 *DELETE* 命令的 *WHERE* 条件表达式中使用，来决定哪些记录需要修改或删除。

```
SELECT [selectoptions] columnlist
INTO @var1, @var2 ...
[ FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ... LIMIT ... ]
```

从 MySQL 5.0 版本开始，使用 *SELECT* 命令的这种语法，把结果记录的所有数据列储存在变量 *var1*、*var2* 等当中。只有在查询仅返回一个记录的时候才能够使用这种语法（如有必要，可以使用 *LIMIT 1*）。

INTO @var1, @var2 ... 也可以放在一个指令的末尾。下面的两个例子具有相同的作用：

```
SELECT title, subtitle INTO @mytitle, @mysub FROM titles WHERE titleID=1
SELECT title, subtitle FROM titles WHERE titleID=1 INTO @mytitle, @mysub
```

```
SELECT [selectoptions] columnlist
INTO OUTFILE 'filename' exportopt
[ FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ... LIMIT ... ]
```

使用 *SELECT* 命令的这种语法，把记录写入一个文本文件中。这里，只是讨论那些应用于这种语法的选项。关于这种语法的其他方面，请参阅 *SELECT* 命令的有关说明。

□ *filename*: 这个文件由 MySQL 服务器的文件系统生成。出于安全因素考虑，这个文件不应该已经存在。而且，必须具有 *FILE* 权限来执行这个 *SELECT* 命令。

□ *exportopt*: 这里规定了文本文件是怎样格式化的。整个的选项块如下所示：

```
{FIELDS
  {TERMINATED BY 'fieldtermstring'}
  {[OPTIONALLY] ENCLOSED BY 'enclosechar'}
  {ESCAPED BY 'escchar'}/}
  {LINES TERMINATED BY 'linetermstring'}
```

fieldtermstring 选项规定了在一个数据行中分隔数据列的字符串（如一个制表符）。

enclosechar 选项规定了每一个条目之前和之后放置的字符，即具有 *ENCLOSED BY \\"* 的 *'I23'*。使用 *OPTIONALLY* 选项，这个字符只能用于 *CHAR*、*VARCHAR*、*TEXT*、*BLOB*、*TIME*、*DATE*、*SET*、以及 *ENUM* 类型的数据列中（而且并不是适用于每一个数值格式，例如 *TIMESTAMP*）。

escchar 选项规定了用于标注特殊字符的字符（通常是反斜线）。特别是当出现在文本文件字符串中的特殊字符同时用于分隔数据单元的时候，这样做是必须的。

如果定义了 *escchar* 选项，那么逃逸字符就总是用于它自身 (\) 及 ASCII 码 0 (\0)。如果 *enclosechar* 为空，那么逃逸字符就作为 *fieldtermstring* 和 *linetermstring* 的首字符的标识符（如，\t 和 \n）。另外，如果 *enclosechar* 不为空，那么 *escchar* 仅用于 *enclosechar*（如，\")，而不用于 *fieldtermstring* 和 *linetermstring*。（这已经不再需要了，因为这个字符串的结尾使用了唯一可以识别的 *enclosechar* 选项。）

linetermstring 选项规定了带有这样字符串的数据行要被终止。在 DOS/Windows 操作系统下的文本文件中必须使用字符串 '\r\n'。

这 4 个字符串都允许包含特殊字符，例如 \b 代表退格键。这些特殊字符的清单可以在前面介绍 *LOAD DATA* 命令的内容里找到。此外，字符串还可以用十六进制记号写出（例如用 0x22 代替 \"）。

类似于使用 *LOAD DATA* 命令时的情况，如下所示的设置是默认的：

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

LINES TERMINATED BY '\r\n'

如果想把由 *SELECT ... INTO OUTFILE* 生成的文件再一次输入到一个数据表中，那么就使用 *LOAD DATA* 命令。该命令的作用与 *SELECT ... INTO OUTFILE* 相反。有关这两个命令进一步的资料和示例请参阅第 14 章。

```
SELECT [selectoptions] column
      INTO DUMPFILE 'filename'
      [ FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ... LIMIT ... ]
```

原则上来说，*SELECT ... INTO DUMPFILE* 与 *SELECT ... INTO OUTFILE* 具有相同的作用（见上面的内容）。区别在于这里的数据储存没有用任何字符来指示数据列或数据行的分隔。

SELECT ... INTO DUMPFILE 命令用于把一个单个的 *BLOB* 对象储存到一个文件里。*SELECT* 查询应该返回的结果应该是准确的一个数据列和一个数据行。但如果出现查询返回的结果不只一个数据单元（由于一些莫名其妙的原因），就会收到一个错误信息：*ERROR1172: Result consisted of more than one row.*

```
(SELECT selectoptions) UNION [ALL] (SELECT selectoptions) unionoptions
```

从 MySQL 4.0 版本开始，可以使用 *UNION* 选项把两个或更多的 *SELECT* 查询结果结合在一起。因此，可以获得一个结果数据表，在这个表里面，单个的查询结果都被串在了一起。由于单个的查询能够影响到不同的数据表，所以，必须保证数据列的数目以及它们的数据类型都是相同的。

可选关键字 *ALL* 的作用是，完全相同的结果（就是说，在不止一个 *SELECT* 查询出现的结果）按照它们相应的重复性出现在最后的结果中。如果没有可选关键字 *ALL*，重复的结果就要被删除（就像 *SELECT* 命令中的 *DISTINCT* 选项一样）。

使用 *SELECT* 命令，所有前面介绍过的选项，如选择数据列、设置排序方式等，都是允许的。通过选项 *unionoptions*，也可以规定最终的结果是如何分类 (*ORDERED BY*) 以及如何减少的 (*LIMIT*)。

```
SET @variable1 = expression1, @variable2 = expression2 ...
```

MySQL 允许用户对自己的用户变量进行管理。这些变量由名字前的 @ 符号来指示。由于这些变量是由每一个客户连接分别管理，所以不会在客户中出现命名冲突。这类变量的内容在连接结束的时候消失。

放弃使用 *SET* 命令，可以使用 *SELECT* 命令对用户变量赋值。语法是 *SELECT @variable:= expression*（注意，必须使用 :=，而不是 =）。

```
SET [options] @@systemvariable = expression
```

如果变量名字前面没有@符号或是有两个@符号，则说明 *SET* 命令是对系统变量进行设置。

□ *options*: MySQL 在系统变量中区分两种层次的有效性：*GLOBAL*（对整个 MySQL 服务器有效）和 *SESSION*（只对当前连接有效）。默认设置是 *SESSION*。

如果不使用 *GLOBAL* 和 *SESSION*，可以把 *global.* 或是 *session.* 作为变量名字的前缀。因此，命令 *SET GLOBAL name = ...* 与 *SET global.name = ...* 具有相同的意思。

只有具有 *Super* 权限的用户才能够对全局级的变量做出改变。*GLOBAL* 改变后，只是对新的连接有效，而对现有连接无效。

```
SET [OPTION] option=value
```

SET 命令也可以用来修改部分 MySQL 选项和密码。尽管看上去其语法与变量赋值一样，但这里讨论的大部分选项都无法通过 *SHOW VARIABLE* 命令去查看。不过，*SELECT@@name* 命令在绝大多数场合都可以使用。

例如，使用

```
SET SQL_LOW_PRIORITY_UPDATES = 0 / 1
```

可以对 MySQL 执行查询命令和修改命令的顺序进行定义。默认的行为 (1) 是优先执行数据修改命令。（这样做的目的在于，一个较长的 *SELECT* 命令不会阻碍数据修改命令，通常情况下，数据修改命令执行得很快。）如果设置的是 0，那么只有当所有的 *SELECT* 命令执行完以后，才能执行数据修改命令。

3. 重要的 *SET* 选项

下面给出了最重要的一些 *SET* 语法变体，它们按照字母顺序排列：

SET AUTOCOMMIT=0 或者 *SET AUTOCOMMIT=1* 把事务的自动提交模式切换为关闭或开启状态。自动提交模式只影响那些支持事务的数据表（参见第 10 章）。

SET CHARACTER SET'csname' 将把字符集 *csname* 赋值给 *character_set_client* 和 *character_set_result* 这两个会话变量，还将把会话变量 *character_set_connection* 赋值为 *character_set_database* 参数的值。*SET CHARACTER SET DEFAULT* 重新把 3 个变量设置成它们的默认值。有关 *character_set_xxx* 的内容参见第 10 章。

SET FOREIGN_KEY_CHECKS=0 或是 *1* 将会禁止或是激活外键检查功能（参见第 8 章）。

如果使用了镜像机制，可以使用 *SET SQL_LOG_BIN=0* 暂时禁用主控系统上的二进制日志，此后做出的修改将不会被镜像到从属服务器上去。*SET SQL_LOG_BIN=1* 重新启用日志功能。

SET NAMES 'csname' 是 *SET CHARACTER SET* 语法的一个变体。区别是字符集 *csname* 被分派到 3 个会话变量 *character_set_client*、*character_set_result* 以及 *character_set_connection* 上。

SET PASSWORD 对改变密码提供了一个非常简便的方式，不用再进行比较麻烦的访问 *mysql* 数据库中数据表的操作：

```
SET PASSWORD = PASSWORD('some password')
```

如果有足够的权限，甚至可以使用 *SET* 命令来设置另外一个用户的密码：

```
SET PASSWORD FOR username@hostname = PASSWORD('newPassword')
```

SET SQL_QUERY_CACHE=0 / 1 / 2 / ON / OFF / DEMAND 设置了查询缓存区的方式（参见第 14 章）。

SET TRANSACTION ISOLATION LEVEL 设置了事务的隔离级别。只有在具有事务能力的数据表上才有这个设置。其语法如下所示：

```
SET [SESSION|GLOBAL] TRANSACTION ISOLATION LEVEL
  READ UNCOMMITTED | READ COMMITTED |
  REPEATABLE READ | SERIALIZABLE
```

SET SESSION 对当前连接改变事务程度，而 *SET GLOBAL* 对所有的未来连接都改变事务程度（但不包括当前的连接）。如果既没有规定 *SESSION* 也没有规定 *GLOBAL*，那么设置只对将来开始的事务有效。（注意，在 *SET TRANSACTION* 命令中的 *SESSION* 和 *GLOBAL* 选项与它们在 *SET @@systemvariable* 命令中的作用稍有不同。）

本书第 10 章中讨论了 4 种隔离程度。对于 InnoDB 类型的数据表，默认是 *REPEATABLE READ*。隔离程度也可以由变量 *@@[global.]tx_isolation* 来读取。

4. 其他 *SET* 选项

下面列出了能够使用 *SET* 命令来修改的所有选项：

```
SET BIG_TABLES = 0 | 1
SET CHARACTER_SET character_set_name | DEFAULT
SET IDENTITY = #
SET INSERT_ID = #
SET LAST_INSERT_ID = #
SET LOW_PRIORITY_UPDATES = 0 | 1
SET MAX_JOIN_SIZE = value | DEFAULT
SET QUERY_CACHE_TYPE = 0 | 1 | 2
SET QUERY_CACHE_TYPE = OFF | ON | DEMAND
SET SQL_AUTO_IS_NULL = 0 | 1
SET SQL_BIG_SELECTS = 0 | 1
SET SQL_BUFFER_RESULT = 0 | 1
SET SQL_LOG_OFF = 0 | 1
SET SQL_LOG_UPDATE = 0 | 1
SET SQL_QUOTE_SHOW_CREATE = 0 | 1
SET SQL_SAFE_UPDATES = 0 | 1
SET SQL_SELECT_LIMIT = value | DEFAULT
SET TIMESTAMP = timestamp_value | DEFAULT
```

对这些设置选项（大部分很少用到）的解释请参见 MySQL 文档对 *SET* 命令的说明：

<http://dev.mysql.com/doc/mysql/en/set-option.html>。

SHOW BINLOG EVENTS [IN logname] [FROM pos] [LIMIT offset, rows]

如果该命令的执行不带选项，那么它返回当前激活的日志文件的全部内容。选项允许对其他日志文件的说明，或者是限制输出。请注意，该命令也可以用来读取外部 MySQL 服务器的日志文件。

SHOW CHARACTER SET [LIKE pattern]

从 MySQL 4.1 版本开始，*SHOW CHARACTER SET* 命令返回一个所有可支持的字符集和它们默认的排序方式的清单。

SHOW COLLATION[LIKE pattern]

从 MySQL 4.1 版本开始，*SHOW COLLATION* 命令返回一个所有可支持的排序方式清单。

SHOW COLUMN TYPES

将来，*SHOW COLUMN TYPES* 命令将返回一个支持数据表中数据列定义的所有数据类型的清单。（在

MySQL 5.0.2 的试验版本中，结果的清单尚不完善。)

```
SHOW [FULL] COLUMNS FROM tablename  
[ FROM databasename ] [ LIKE pattern ]
```

SHOW COLUMNS 命令返回一个描述数据表中所有数据列信息的表格（数据列名字、数据列类型、索引、默认值等）。使用 *LIKE* 选项，数据列的清单可以通过带有通配符“_”和“%”的搜索模式进行过滤。可选关键字 *FULL* 的作用是把当前用户的访问权限也显示在数据列上。要想获得同样的信息，也可以使用命令 *SHOW FIELDS FROM tablename*, *EXPLAIN tablename* 或者 *DESCRIBE tablename*，还可以使用外部程序 *mysqlshow*。更详细的信息可以从虚拟数据表 *information_schema.columns*（从 MySQL 5.0 版本开始）里查到。

```
SHOW CREATE DATABASE tablename
```

从 MySQL 4.1 版本开始，*SHOW CREATE DATABASE* 命令显示可以用来重新创建指定数据库的 SQL 命令。

```
SHOW CREATE FUNCTION/PROCEDURE name
```

从 MySQL 5.0 版本开始，*SHOW CREATE FUNCTION/PROCEDURE* 命令显示可以用来重新创建给定数据表或视图的 SQL 命令。

```
SHOW CREATE TABLE name
```

SHOW CREATE TABLE 命令显示可以用来重新创建指定表或视图的 SQL 命令。

在命令的结果里，所有的对象名都被括在两个反单引号之间，如'tablename'，或'columnname'。假如不需要这样的结果，需要提前执行 *SET SQL_QUOTE_SHOW_CREATE=0* 命令。

```
SHOW CREATE VIEW name
```

SHOW CREATE VIEW 命令显示可以用来重新创建指定视图的 SQL 命令。该命令需要 *Create View* 权限。

```
SHOW DATABASES [ LIKE pattern ]
```

SHOW DATABASES 命令返回一个用户可以访问的所有数据库的清单。该清单可以通过带有通配符“_”和“%”的搜索模式进行过滤。同样的信息也可以通过使用外部程序 *mysqlshow* 来获得。

对于具有 *Show Databases* 权限的用户，*SHOW DATABASES* 命令返回一个所有数据库的清单，包括那些用户不能够访问的数据库。

```
SHOW [STORAGE] ENGINES
```

从 MySQL 4.1 版本开始，*SHOW ENGINES* 命令显示所有数据表引擎的清单（MyISAM、InnoDB 等），包括驱动器是否由当前的 MySQL 版本支持的信息。

```
SHOW [COUNT(*)] ERRORS [LIMIT {offset, } count]
```

从 MySQL 4.1 版本开始, *SHOW ERRORS* 命令返回一个由最近执行的命令所触发的出错清单。和 *SELECT* 命令一样, 使用 *LIMIT* 选项可以对结果进行限制。

SHOW FIELDS

参见 *SHOW COLUMNS* 命令。

SHOW FUNCTION STATUS [LIKE 'pattern']

从 MySQL 5.0 版本开始, 该命令返回一个所有函数 (SP) 的清单。该清单覆盖所有的数据库。作为可选项, 该清单可以减少到所有名字满足搜索模式的函数 (允许使用 SQL 的通配符 “_” 和 “%”)。可以使用 *SHOW PROCEDURE STATUS* 命令来显示所有程序的清单。

SHOW GRANTS FOR user@host

SHOW GRANTS 命令显示对一个特定用户的所有访问权限的清单。*user* 和 *host* 必须要准确定义, 因为这些字符串要储存在各种 *mysql* 访问数据表中。不允许使用通配符。

SHOW INDEX FROM table

SHOW INDEX 命令返回一个关于给定数据表的所有索引信息的数据表。

SHOW INNODB STATUS

SHOW INNODB STATUS 命令返回的信息是关于 InnoDB 类型的数据表驱动程序的各种内部工作机制。数据可以用于速度优化。更多信息请访问以下站点: <http://dev.mysql.com/doc/mysql/en/innodb-tuning.html>。

SHOW KEYS

参见 *SHOW INDEX* 命令。

SHOW [BDB] LOGS

该命令显示当前使用的 BDB 日志文件是哪一个。(如果当前没有使用 BDB 数据表, 那么该命令就不返回结果。)

SHOW MASTER LOGS

该命令返回所有二进制日志文件的清单。它只能够在镜像系统的主控计算机上执行。

SHOW MASTER STATUS

该命令用来显示哪一个是当前的日志文件, 它在这个文件中的位置以及哪些数据库是在日志之外 (配置设

置为 binlog-do-db 和 binlog-ignore-db)。它只能够在镜像系统的主控计算机上执行。

SHOW PRIVILEGES

从 MySQL 4.1 版本开始，该命令返回一个所有可支持的权限清单，并具有简单的描述。

SHOW PROCEDURE STATUS [LIKE 'pattern']

从 MySQL 5.0 版本开始，该命令返回一个所有程序 (SP) 的清单。该清单由所有数据库的 SP 组成。作为可选项，该清单可以减少到所有名字满足搜索模式的程序 (允许使用 SQL 的通配符 “_” 和 “%”)。可以使用 *SHOW FUNCTION STATUS* 命令来显示所有函数的清单。

SHOW [FULL] PROCESSLIST

该命令返回一个 MySQL 服务器里运行的所有线程 (子处理过程) 的清单。如果赋予了 *PROCESS* 权限，那么所有的线程都被显示出来，否则，只是显示用户的线程。

选项 *FULL* 的作用是，对于每一个线程，最近执行的命令的全部文本要被显示。没有这个选项，只有前 100 个字符会被显示出来。

也可以使用外部命令 *mysqladmin* 来显示进程清单。

SHOW SLAVE HOSTS

该命令返回一个复制主数据库的所有从属服务器清单。该命令只能够用于镜像机制系统中的主控计算机上。它只是在从属服务器上起作用，对于这些从属服务器来说，主机名字在配置文件中有明确规定，其格式为 report-host = hostname。

SHOW SLAVE STATUS

该命令提供了镜像机制系统中的状态信息，包括显示有关文件 *master.info* 的所有信息。该命令只能够用于镜像机制系统中的从属计算机上。

SHOW STATUS

该命令返回一个各种 MySQL 变量的清单，这些变量提供了有关 MySQL 当前状态的信息(例如 *Connections*、*Open_files*、*Uptime*)。这些相同的信息也可以由外部程序 *mysqladmin* 来显示。对所有变量的描述请参阅 MySQL 文档对 *SHOW STATUS* 命令的说明：<http://dev.mysql.com/doc/mysql/en/show-status.html>。

SHOW TABLE STATUS [FROM database] [LIKE pattern]

SHOW TABLE STATUS 命令返回有关当前活跃的所有数据表或指定的数据库的信息：数据表的类型、记录的数目、平均记录长度、*Create_time*、*Update_time* 等。这些相同的信息也可以由外部程序 *mysqladmin* 来显示。使用 *pattern* 选项可以限制数据表的清单；允许在 *pattern* 选项里使用 SQL 的通配符 “_” 和 “%”。

`SHOW TABLE TYPES` see also `SHOW ENGINES`

从 MySQL 4.1 版本开始, `SHOW TABLE TYPES` 返回一个所有可支持的数据表类型的清单 (*MyISAM*、*HEAP*、*InnoDB* 等)。

`SHOW TABLES [FROM database] [LIKE pattern]`

`SHOW TABLE` 命令返回一个当前 (或指定) 数据库中的所有数据表和 *Views* 的清单。作为可选项, 该所有数据表的清单可以减少到全部满足搜索模式 *pattern* 的数据表 (允许使用 SQL 的通配符 “_” 和 “%”)。关于数据表结构的更多信息, 可以使用 `DESCRIBE TABLE` 和 `SHOW COLUMNS` 命令来获得。数据表的清单也可以使用外部程序 `mysqlshow` 来显示。

`SHOW [options] VARIABLES [LIKE pattern]`

该命令将返回一份非常长的系统变量清单, 这些系统变量是由 MySQL 定义并具有它们的值 (如 *ansi_mode*、*sort_buffer*、*tmpdir*、*wait_timeout*, 几乎没有名字)。为了限制这个清单, 可以规定一个模式 (如, `LIKE'char%'`)。

可以在启动 MySQL 的时候设置这种变量, 或者是在启动以后用 `SET` 命令来设置。变量清单也可以使用外部程序 `mysqladmin` 来显示。

options: 在这里, 可以规定 *GLOBAL* 选项或是 *SESSION* 选项。*GLOBAL* 选项的作用是显示在全局有效的默认值, 而 *SESSION* 选项意味着只显示对当前连接有效的值。默认设置是 *SESSION* 选项。

有关变量进一步的描述请参阅 MySQL 文档: <http://dev.mysql.com/doc/mysql/en/server-system-variables.html>。

`SHOW [COUNT(*)] WARNINGS [LIMIT [offset,] count]`

从 MySQL 4.1 版本开始, 该命令返回一份由最近执行的 SQL 命令所引起的所有警告的清单。

`SLAVE START/STOP [IO_THREAD | SQL_THREAD]`

这些命令开始或结束镜像机制 (这里留给读者来决定哪个是开始, 哪个是结束)。它们只能在镜像机制系统中的从属计算机上执行。

默认情况下, 两个线程启动用于复制: *IO* 线程 (把二进制日志数据从主控计算机复制到从属计算机) 和 *SQL* 线程 (执行日志文件的 SQL 命令)。使用 *IO_THREAD* 或 *SQL_THREAD* 的选项说明, 这两个线程可以独立地启动或停止 (只是在调试时这样做才有意义)。

`START TRANSACTION`

如果是在一个具有事务能力的数据表 (*InnoDB*) 上工作, 那么 `START TRANSACTION` 命令就开始了一个新的事务。该命令符合 ANSI-99 的标准, 但是它始于 MySQL 4.0.11 版本。在旧的版本中, 必须使用具有同样意义的 `BEGIN` 命令。

`TRUNCATE TABLE tablename`

TRUNCATE 命令和不带 *WHERE* 子句的 *DELETE* 命令的效果相同，即删除数据表中的所有记录。完成这一操作的方式是先删除整个数据表，然后再重新创建它。（这样做要比逐条删除记录快许多。）

TRUNCATE 命令不能在事务里使用，但 *TRUNCATE* 命令与 *COMMIT* 命令有一个共同点：先把尚未写入数据库的修改全部写入数据库再正式执行。*TRUNCATE* 命令也可以用 *ROLLBACK* 命令来撤销（把数据表恢复为原样）。

UNION see also *SELECT UNION*

使用 *UNION* 命令，可以集合几个 *SELECT* 查询的结果。

UNLOCK TABLES

UNLOCK TABLES 命令删除所有用户设置的 *LOCK* 选项。该命令对所有的数据库有效（如，与哪一个数据库是当前数据库没有关系）。

```
UPDATE [updateoptions] tablename SET col1=value1, col2=value2 ...
[ WHERE condition ]
[ ORDER BY columns ]
[ LIMIT maxrecords ]
```

UPDATE 命令修改由 *WHERE* 子句所指定的数据表记录的某些字段。没有在 *SET* 命令里出现的字段将保持不变。在 *value* 选项里可以引用现有字段。例如，一个 *UPDATE* 命令可以以下面的形式出现：

```
UPDATE products SET price = price + 5 WHERE productID=3
```

警告：如果没有 *WHERE* 子句，数据表中的所有数据记录都要被改变。（在上面的例子中，所有产品的价格都要增加 5）。

- *updateoptions*: 在这里，有可能给出选项 *LOW PRIORITY* 和 *IGNORE*。其作用与 *INSERT* 相同。
- *condition*: 这个条件选项指定了哪些记录会因为本次修改而受到影响。*condition* 的语法参见 *SELECT* 命令的相关内容。
- *columns*: 使用 *ORDER BY* 选项，可以在进行修改之前把记录清单分类。但只有与 *LIMIT* 选项一起使用的时候，这样做才有意义，例如，要修改前面 10 条或最后 10 条记录（根据某些准则排序）。这项功能是从 MySQL 4.0 版本开始的。
- *maxrecords*: 使用 *LIMIT* 选项，指定了有可能修改的记录的最大数值。

```
UPDATE [updateoptions] table1, table2, table3
SET table1.col1=table2.col2 ...
[ WHERE condition ] [ ORDER BY columns ] [ LIMIT maxrecords ]
```

从 MySQL 4.0 版本开始，*UPDATE* 命令可以包括不止一个数据表。这个语法要求把本次修改操作所涉及的所有数据表列在关键字 *UPDATE* 的后面，但只有在 *SET* 子句里被赋值的数据列（数据表）会被修改。数据表之间的关联关系由 *WHERE* 子句来设置。

USE *databasename*

USE 命令把给定数据库设置为当前 MySQL 连接的默认数据库。在这条连接被关闭之前（或者在执行下一条 *USE* 命令之前），后续的 SQL 命令将默认地以给定数据库 *databasename* 里的数据表作为操作对象。

21.7 SQL 函数指南

这里讨论的函数既可以用于 *SELECT* 查询，也可以用于其他的 SQL 命令。我们先从几个示例开始。在第一个例子里，将使用 *CONCAT()* 函数合并两个数据表的数据列来创建一个新的字符串。在第二个例子里，函数 *PASSWORD()* 用来在一个数据列中储存一个经过加密的密码。在第三个例子里，函数 *DATE_FORMAT()* 用来对日期/时间值进行排版。

```
SELECT CONCAT(firstname, ' ', lastname) FROM users
Peter Smith
...
INSERT INTO logins (username, userpassword)
VALUES ('smith', PASSWORD('xxx'))
SELECT DATE_FORMAT(a_date, '%Y %M %e')
FROM exceptions.test_date
2005 December 7
```

提示 本节的目的在于使用户对有关的函数有一个总体的认识。关于这些函数更多的资料请查阅 MySQL 文档。其中的一部分函数已经在本书的有关内容里介绍过。

21.7.1 算术函数

算术函数	
<i>ABS(x)</i>	计算绝对值（非负数）
<i>ACOS(x), ASIN(x)</i>	计算反正弦和反余弦值
<i>ATAN(x), ATAN2(x, y)</i>	计算反正切值
<i>CEILING(x)</i>	大于或等于 <i>x</i> 的最小整数
<i>COS(x)</i>	计算余弦值； <i>x</i> 以弧度给出
<i>COT(x)</i>	计算余切值
<i>DEGREES(x)</i>	把弧度转化为角度（乘以 180/pi）
<i>EXP(x)</i>	返回 e^x
<i>FLOOR(x)</i>	小于或等于 <i>x</i> 的最大整数
<i>LOG(x)</i>	返回自然对数（即，以 e 为底）
<i>LOG10(x)&</i>	返回以 10 为底的对数
<i>MOD(x, y)</i>	返回整数除法的余数，与 $x \% y$ 等价
<i>PI()</i>	返回 3.1415927
<i>POW(x, y)</i>	返回 x^y
<i>POWER(x, y)</i>	与 <i>POW(x, y)</i> 等价
<i>RADIANS(x)</i>	把角度转化为弧度（乘以 $\pi / 180$ ）
<i>RAND()</i>	返回一个 0.0~1.0 之间的随机数
<i>RAND(n)</i>	返回一个可重复生成的数（不是特别随机）
<i>ROUND(x)</i>	返回一个最近的整数
<i>ROUND(x, y)</i>	返回一个有小数点后有 <i>y</i> 位的值
<i>SIGN(x)</i>	根据 <i>x</i> 是负数、0、正数而分别返回 -1、0、1
<i>SIN(x)</i>	计算正弦值
<i>SQRT(x)</i>	计算平方根
<i>TAN(x)</i>	计算正切值

(续)

算术函数

<i>TRUNCATE(x)</i>	删除小数点后的数字
<i>TRUNCATE(x, y)</i>	小数点后保留 y 位 (例如, <i>TRUNCATE(1.236439, 2)</i> 返回 1.23)

一般来说, 如果用户提供的参数值是无效或非法的 (如, *SQRT(-1)*) , 这些函数都将返回 *NULL*。

21.7.2 比较函数、测试函数、分支函数**比较函数**

<i>COALESCE(x, y, z, ...)</i>	返回第一个非 <i>NULL</i> 参数
<i>GREATEST(x, y, z, ...)</i>	返回参数中的最大值或最大字符串
<i>IF(expr, val1, val2)</i>	如果表达式 <i>expr</i> 为真, 则返回 <i>val1</i> , 否则, 返回 <i>val2</i>
<i>IFNULL(expr1, expr2)</i>	如果表达式 <i>expr1</i> 为 <i>NULL</i> , 则返回 <i>expr2</i> , 否则, 返回 <i>expr1</i>
<i>INTERVAL(x, n1, n2, ...)</i>	若 <i>x<n1</i> , 则返回 0; 若 <i>x>n2</i> , 则返回 1; 依此类推。所有的输入参数必须是整数, 并且要保证 <i>n1<n2</i>
<i>ISNULL(x)</i>	根据 <i>x</i> 是否为 <i>IS NULL</i> , 返回 1 或 0
<i>LEAST(x, y, z, ...)</i>	返回参数中的最小值或最小字符串
<i>STRCMP(s1, s2)</i>	若在排序方式中 <i>s1=s2</i> , 则返回 0; 若 <i>s1<s2</i> , 则返回 -1; 若 <i>s1>s2</i> , 则返回 1。从 MySQL 4.0 版本开始, 这个函数还会考虑到当前字符集的影响。在默认情况下, 该函数不再区分字母的大小写 (这一点与 MySQL 3.32 版本不同)

检测、分支

<i>IF(expr, result1, result2)</i>	如果表达式 <i>expr</i> 为真, 则返回 <i>result1</i> , 否则, 返回 <i>result2</i>
<i>CASE expr</i>	如果表达式 <i>expr=val1</i> , 则返回 <i>result1</i> , 如果表达式 <i>expr=val2</i> , 返回 <i>result2</i>
<i>WHEN val1 THEN result1</i>	如果没有满足的条件, 则结果为 <i>resultn</i>
<i>WHEN val2 THEN result2</i>	
<i>ELSE resultn.</i>	
<i>CASE</i>	若条件 <i>cond1</i> 为真, 则返回 <i>result1</i> , 依此类推
<i>WHEN cond1 THEN result1</i>	
<i>WHEN cond2 THEN result2</i>	
<i>ELSE resultn</i>	
<i>END</i>	

21.7.3 类型转换 (投射)**类型转换**

<i>CAST(x AS type)</i>	把 <i>x</i> 转化成指定类型。 <i>CAST</i> 函数对下列类型起作用: <i>BINARY</i> 、 <i>CHAR</i> 、 <i>DATE</i> 、 <i>DATETIME</i> 、 <i>SIGNED [INTEGER]</i> 、 <i>TIME</i> 以及 <i>UNSIGNED [INTEGER]</i>
<i>CONVERT(x, type)</i>	与 <i>CAST(x AS type)</i> 的作用相同
<i>CONVERT(s USING cs)</i>	在字符集 <i>cs</i> 中表示字符 <i>s</i> (从 MySQL 4.1 版本开始)

21.7.4 字符串处理

绝大多数字符串函数也可以用来处理二进制数据。从 MySQL 4.1 版本开始 (带 Unicode 支持), 与位

置和长度有关的字符串函数(*LEFT()*、*MID()*等)都以字符而不是字节为计量单位。也就是说,从MySQL 4.1版本开始,不管*column*数据列使用的是哪一种字符集,*MID(column, 3, 1)*总是返回第三个字符(而不再是第三个字节)。

处理字符串	
<i>CHAR_LENGTH(s)</i>	返回 <i>s</i> 中的字符数目; <i>CHAR_LENGTH</i> 也适用于多字节字符集(如, Unicode 字符集)
<i>CONCAT(s1,s2,s3, ...)</i>	合并字符串
<i>CONCAT_WS(x,s1,s2, ...)</i>	作用同 <i>CONCAT</i> 函数,区别在于 <i>x</i> 是插入在每两个字符串中; <i>CONCAT_WS(' ','a','b','c')</i> 返回'a,b,c'
<i>ELT(n,s1,s2, ...)</i>	返回第 <i>n</i> 个字符串; <i>ELT(2,'a','b','c')</i> 返回'b'
<i>EXPORT_SET(x,s1,s2)</i>	根据 <i>x</i> 的二进制位编码从字符串 <i>s1</i> 和 <i>s2</i> 创建一个新的字符串, <i>x</i> 被解释为一个 64 位整数
<i>FIELD(s,s1,s2, ...)</i>	把字符串 <i>s</i> 与字符串 <i>s1</i> 、 <i>s2</i> 等进行比较, 并返回第一个匹配字符串的序号; <i>FIELD('b','a','b','c')</i> 返回 2
<i>FIND_IN_SET(s1,s2)</i>	在字符串 <i>s2</i> 中搜索字符串 <i>s1</i> ; 字符串 <i>s2</i> 中包含着一个逗号分隔的字符串清单; <i>FIND_IN_SET('b','a','b','c')</i> 返回 2
<i>INSERT(s1,pos,0,s2)</i>	把字符串 <i>s2</i> 插入到字符串 <i>s1</i> 的 <i>pos</i> 位置上; <i>INSERT('ABCDEF',3,0,'abc')</i> 函数返回值为'ABabcDEF'
<i>INSERT(s1,pos,len,s2)</i>	把字符串 <i>s2</i> 插入到字符串 <i>s1</i> 的 <i>pos</i> 位置上, 并用新的字符取代字符串 <i>s2</i> 的 <i>len</i> 个字符; <i>INSERT('ABCDEF',3,2,'abc')</i> 函数返回值为'ABabcEF'
<i>INSTR(s,sub)</i>	返回字符串 <i>s</i> 中的 <i>sub</i> 位置; <i>INSTR('abcde','bc')</i> 返回 2
<i>LCASE(s)</i>	把大写字母转换为小写字母
<i>LEFT(s,n)</i>	返回字符串 <i>s</i> 的前 <i>n</i> 个字符
<i>LENGTH(s)</i>	返回储存字符串 <i>s</i> 所必需的字节数; 如果使用的是多字节字符集(如 Unicode), 那么就必须使用 <i>CHAR_LENGTH</i> 来决定这个字符的数目
<i>LOCATE(sub,s)</i>	返回字符串 <i>s</i> 中的 <i>sub</i> 位置; <i>LOCATE('bc','abcde')</i> 返回 2
<i>LOCATE(sub,s,n)</i>	同上面的函数一样, 但是对 <i>sub</i> 的搜索仅开始于字符串 <i>s</i> 的第 <i>n</i> 个字符
<i>LOWER(s)</i>	把大写字母改变为小写字母
<i>LPAD(s,len,fill)</i>	把补足字符 <i>fill</i> 插入到字符串 <i>s</i> 的开头, 使得字符串 <i>s</i> 的长度满足 <i>len</i> ; <i>LPAD('ab',5,'*)</i> 返回'***ab'
<i>LTRIM(s)</i>	删除字符串开始处的空格
<i>MAKE_SET(x,s1,s2, ...)</i>	生成一个新的字符串, 新字符串由 <i>s1</i> 、 <i>s2</i> ... <i>sn</i> 按照表达式 <i>x</i> 给出的顺序拼接而成。比如说, <i>MAKE_SET(1+2+4, 'a','b','c','d')</i> 将返回'a,b,d' ¹
<i>MID(s,pos,len)</i>	从字符串 <i>s</i> 中的 <i>pos</i> 位置开始截取 <i>len</i> 个字符; <i>MID('abcde',3,2)</i> 函数返回'cd'
<i>POSITION(sub IN s)</i>	与函数 <i>LOCATE(sub,s)</i> 的作用相同
<i>QUOTE(s)</i>	从 MySQL 4.0 版本开始, 这个函数返回一个适用于 SQL 命令的字符串; 特殊字符, 如'和", 都使用带一个退格键的反斜杠作为前缀
<i>REPEAT(s,n)</i>	把字符串 <i>s</i> 重复 <i>n</i> 次; <i>REPEAT('ab',3)</i> 函数返回'ababab'
<i>REPLACE(s,find,rpl)</i>	把字符串 <i>s</i> 中的所有 <i>find</i> 字符串由 <i>rpl</i> 来取代; <i>REPLACE('abcde','b','xy')</i> 函数返回'axycede'
<i>REVERSE(s)</i>	翻转该字符串
<i>RIGHT(s,n)</i>	返回字符串 <i>s</i> 的最后 <i>n</i> 个字符

1. 原文“1+2+8”可能是“1+2+4”的笔误。——译者注

(续)

处理字符串

<i>RPAD(s,len,fill)</i>	把补足字符 <i>fill</i> 插入到字符串 <i>s</i> 的末端，使得字符串 <i>s</i> 的长度满足 <i>len</i> : <i>RPAD('ab',5,'*')</i> 返回 'ab***'
<i>RTRIM(s)</i>	删除字符串末端的空格
<i>SPACE(n)</i>	返回 <i>n</i> 个空格字符
<i>SUBSTRING(s,pos)</i>	返回字符串 <i>s</i> 从位置 <i>pos</i> 处的右边部分
<i>SUBSTRING(s,pos,len)</i>	作用同上，但仅返回 <i>len</i> 长度的字符（等同于 <i>MID(s,pos,len)</i> ）
<i>SUBSTRING_INDEX(s,f,n)</i>	搜索字符串 <i>s</i> 中 <i>f</i> 的第 <i>n</i> 次出现，并返回这个位置左边的字符串（唯一的）；如果 <i>n</i> 为负值，则搜索是从字符串的末端开始，并返回这个位置右边的字符串 <i>SUBSTRING_INDEX('abcabc','b',2)</i> 返回 'abca' <i>SUBSTRING_INDEX('abcabc','b',-2)</i> 返回 'cabc'
<i>TRIM(s)</i>	从开始到末端删除字符串 <i>s</i> 中的空格
<i>TRIM(f FROM s)</i>	从开始到末端删除字符串 <i>s</i> 中的 <i>f</i>
<i>UCASE(s) / UPPER(s)</i>	把小写字母转换为大写字母

对数值和字符串进行转换

<i>ASCII(s)</i>	返回字符串 <i>s</i> 第一个字符的字节编码。因此， <i>ASCII('A')</i> 返回 65；参见 <i>ORD</i> 函数
<i>BIN(x)</i>	返回 <i>x</i> 的二进制编码： <i>BIN(12)</i> 返回 '1010'
<i>CHAR(x,y,z, ...)</i>	返回的字符串由编码 <i>x</i> , <i>y</i> , ... 形成： <i>CHAR(65,66)</i> 返回 'AB'
<i>CHARSET(s)</i>	从 MySQL 4.1 版本开始，返回字符串 <i>s</i> 所使用的字符集的名字
<i>CONV(x,from,to)</i>	把 <i>x</i> 由 <i>from</i> 进制转变成 <i>to</i> 进制： <i>CONV(25,10,16)</i> 返回十六进制的 '19'
<i>CONVERT(s USING cs)</i>	从 MySQL 4.1 版本开始，表示字符集 <i>cs</i> 中的字符串 <i>s</i>
<i>FORMAT(x,n)</i>	格式化 <i>x</i> 为千位数上由逗号分隔、小数点后保留 <i>n</i> 位的数： <i>FORMAT(12345.678,2)</i> 返回 '12,345.68'
<i>HEX(x)</i>	返回 <i>x</i> 的十六进制编码； <i>x</i> 可以是一个 64 位整数或（从 MySQL 4.0 版本开始）一个字符串；在后一种情况下，每一个字符都要转变成一个 8 位的十六进制编码： <i>HEX('abc')</i> 返回 '414243'
<i>INET_NTOA(n)</i>	把 <i>n</i> 转变成一个至少有 4 个组的 IP 地址： <i>INET_NTOA(1852797041)</i> 返回 '110.111.112.113'。 <i>INET_ATON(ipadr)</i> 把一个 IP 地址转变成相应的 32 位或 64 位整数： <i>INET_ATON('110.111.112.113')</i> 返回 1852797041
<i>OCT(x)</i>	返回 <i>x</i> 的八进制编码
<i>ORD(s)</i>	同 <i>ASCII(s)</i> 函数一样，返回第一个字符的编码，但是这个函数也适用于多字节字符
<i>SOUNDEX(s)</i>	返回的字符串应该匹配类似发音的英语单词： <i>SOUNDEX('har')</i> 和 <i>SOUNDEX('head')</i> 都返回 'H300'；有关 <i>SOUNDEX</i> 函数算法的更多资料可以参阅 Joe Celko 所著的 <i>for Smarties</i>

字符串的加密与密码管理

<i>AES_DECRYPT(crypt, key)</i>	使用 AES 算法 (Rijndael) 并使用密钥 <i>key</i> 对 <i>crypt</i> 解密
<i>AES_ENCRYPT(str, key)</i>	使用密钥 <i>key</i> 对字符串 <i>str</i> 加密
<i>DES_ENCRYPT(str [,keyno keystr])</i>	使用 DES 算法对字符串 <i>str</i> 加密；这个函数始于 MySQL 4.0 版本，而且仅当 MySQL 是由 SSL 函数汇编的时候才能够使用；如果没有第二个可选参数，DES 密钥文件中的第一个密钥就用于加密；作为可选项，密钥的数值或名字可以指定

(续)

字符串的加密与密码管理	
<i>DES_DECRYPT</i> (<i>crypt</i> [, <i>keyno</i> <i>keystr</i>])	使用 DES 算法对 <i>crypt</i> 解密
<i>DECODE</i> (<i>crypt</i> , <i>pw</i>)	使用密码 <i>pw</i> 对 <i>crypt</i> 解密
<i>ENCODE</i> (<i>str</i> , <i>pw</i>)	使用 <i>pw</i> 作为密码对字符串 <i>str</i> 加密；结果为一个二进制字符串，该字符串可以用 <i>DECODE</i> 函数来解密
<i>ENCRYPT</i> (<i>pw</i>)	使用 UNIX 的 <i>crypt</i> 函数对密码加密；如果不支持这个函数，则返回 <i>ENCRYPT NULL</i>
<i>PASSWORD</i> (<i>pw</i>)	使用 <i>USER</i> 数据表中用以储存的密码的算法加密密码；结果是一个 16 位的字符串；注意，从 MySQL 4.1 版本开始， <i>PASSWORD</i> 使用了更强大的加密并返回一个 45 位的字符串
<i>OLD_PASSWORD</i> (<i>pw</i>)	对 MySQL3.23.n 和 MySQL 4.0.n 版本下的 <i>PASSWORD</i> 进行密码加密；函数始于 MySQL 4.1 版本

校验和的计算函数	
<i>CRC32</i> (<i>s</i>)	从 MySQL 4.1 版本开始，对字符串 <i>s</i> 进行一个校验计算（循环冗余校验值）
<i>MD5</i> (<i>str</i>)	对字符串计算 MD5 校验和
<i>SHA</i> (<i>str</i>), <i>SHA1</i> (<i>str</i>)	从 MySQL 4.0 版本开始，使用 SHA1 算法（指定于 RFC3174）计算一个 160 位的校验和。SHA 被认为要比 MD5 安全许多。结果是返回包含 40 位十六进制编码的一个字符串；SHA 与 SHA1 相同

21.7.5 日期/时间函数

下面列出的函数中，有些是从 MySQL 4.0 或 4.1 版本才开始出现的。

确定当前时间	
<i>CURDATE</i> ()	返回当前日期，如，'2005-12-31' 同义函数： <i>CURRENT_DATE</i> ()
<i>CURTIME</i> ()	以字符串或整数返回当前时间，这取决于上下文，如，'23:59:59'或 235959（整数） 同义函数： <i>CURRENT_TIME</i> ()
<i>NOW</i> ()	以格式'2005-12-31 23:59:59'返回当前时间 同义函数： <i>CURRENT_TIMESTAMP</i> ()、 <i>LOCALTIME</i> ()、 <i>LOCALTIMESTAMP</i> ()、 <i>SYSDATE</i> ()
<i>UNIX_TIMESTAMP</i> ()	以 UNIX 时间戳格式（32 位整数）返回当前系统时间
<i>UTC_DATE</i> ()	返回 UTC（Universal Coordinated Time，国际协调时间）日期
<i>UTC_TIME</i> ()	返回 UTC 时间
<i>UTC_DATETIME</i> ()	返回 UTC 日期和时间

日期/时间值的计算、排版输出、转换	
<i>ADDDATE</i> (<i>d</i> , <i>n</i>)	在起始时间 <i>d</i> 中加入 <i>n</i> 天
<i>ADDDATE</i> (...)	在起始时间 <i>d</i> 中加入一段时间（见下面部分） 同义： <i>DATE_ADD</i> ()
<i>ADDTIME</i> (<i>d</i> , <i>t</i>)	在起始时间 <i>d</i> (<i>DATETIME</i>) 中加入时间 <i>t</i> (<i>TIME</i>)

(续)

日期/时间值的计算、排版输出、转换

<i>CONVERT_TZ(d, tz1, tz2)</i>	把时间 <i>d</i> 从时区 <i>tz1</i> 转化成时区 <i>tz2</i> 。对于给定的时区，语法由操作系统来决定。在 Windows 操作系统下，必须给出与 UTC 的时差（如，'+2:00'），而在 Linux 操作系统下，经过配置以后，可以使用时区名字（如，'America/New_York'；参见第 10 章）
<i>DATE(d)</i>	仅返回 <i>DATETIME</i> 表达式的日期部分（即该函数删除了时间部分）
<i>DATEDIFF(d1, d2)</i>	返回 <i>d1</i> 和 <i>d2</i> 之间的天数。在计算中忽略时间部分
<i>DATE_FORMAT(d, form)</i>	按照格式字符串 <i>f</i> 格式化 <i>d</i> ；见下面部分
<i>DAYNAME(date)</i>	返回'Monday'、'Tuesday'等
<i>DAYOFMONTH(date)</i>	返回一个月当中的某一天（1~31）
<i>DAYOFWEEK(date)</i>	返回星期几（1=星期日，直到 7=星期六）
<i>DAYOFYEAR(date)</i>	返回一年当中的某一天（1~366）
<i>EXTRACT(i FROM date)</i>	返回希望的时间段内截取出的一个值
<i>EXTRACT (YEAR FROM '2003-12-31')</i>	返回 2003
<i>FROM_DAYS(n)</i>	返回公元 0 年后 <i>n</i> 天的日期值
<i>FROM_DAYS(3660)</i>	返回'0010-01-08'
<i>FROM_UNIXTIME(t)</i>	把 UNIX 当前时间数值 <i>t</i> 转换成日期
<i>FROM_UNIXTIME(0)</i>	返回'1970-01-01 01:00:00'
<i>FROM_UNIXTIME(t,f)</i>	同上面的函数一样，但使用的格式与 <i>DATE_FORMAT</i> 一样
<i>GET_FORMAT(...)</i>	对 <i>DATE_FORMAT</i> 函数返回预先指定的格式编码（见下面的表）
<i>HOUR(time)</i>	返回小时值（0~23）
<i>LAST_DAY(d)</i>	返回由日期 <i>d</i> 指定的当月的最后一天。 <i>LAST_DAY('2005-02-01')</i> 返回'2005-02-28'
<i>MAKEDATE(y, dayofyear)</i>	从输入的年和日中创建一个日期 <i>DATE</i> 表达式
<i>MAKETIME(h, m, s)</i>	从输入的小时、分钟和秒中创建一个时间 <i>TIME</i> 表达式
<i>MICROSECOND(d)</i>	返回微秒值（0~999999）
<i>MINUTE(time)</i>	返回分钟值（0~59）
<i>MONTH(date)</i>	返回月份值（1~12）
<i>MONTHNAME(date)</i>	返回月份的名字（'January'等）
<i>PERIOD_ADD(s, n)</i>	添加 <i>n</i> 月到起始日期，日期必须以格式'YYYYMM'给出（如，'200512'表示 2005 年 12 月）
<i>PERIOD_DIFF(s, e)</i>	返回起始和结束日期之间的月份值。必须以格式'YYYYMM'给出这两个时间
<i>QUARTER(date)</i>	返回季度值
<i>SECOND(time)</i>	返回秒值（0~59）
<i>SEC_TO_TIME(n)</i>	把 <i>n</i> 秒转换为 <i>hh:mm:ss</i> 格式的时间值
<i>SEC_TO_TIME(3603)</i>	返回'01:00:03'
<i>STR_TO_DATE(s, form)</i>	依据格式编码 <i>form</i> 翻译字符串 <i>s</i> 。函数 <i>STR_TO_DATE</i> 与函数 <i>DATE_FORMAT</i> 相反
<i>SUBDATE(d, n)</i>	从起始时间 <i>d</i> 中减去 <i>n</i> 天
<i>SUBDATE(d)</i>	从起始时间 <i>d</i> 中减去一段时间（见下面）
<i>SUBTIME(d, t)</i>	同义函数： <i>DATE_ADD()</i> 从起始时间 <i>d</i> (<i>DATETIME</i>) 中减去时间 <i>t</i> (<i>TIME</i>)
<i>TIMESTAMP(s)</i>	返回字符串中给出的起始时间作为 <i>TIMESTAMP</i> 的值。 <i>TIMESTAMP('2005-12-31')</i> 返回'2005-12-31 00:00:00'

(续)

日期/时间值的计算、排版输出、转换	
<i>TIMESTAMP(s, time)</i>	返回 <i>s+time</i> 作为 <i>TIMESTAMP</i> 的值
<i>TIMESTAMPADD(i, n, s)</i>	添加 <i>n</i> 次时间段 <i>i</i> (如 <i>MONTH</i>) 到起始时间 <i>s</i> 上
<i>TIMESTAMPDIFF(i, s, e)</i>	返回起始时间 <i>s</i> 和结束时间 <i>e</i> 之间的时间段 <i>i</i> 的数值。函数 <i>TIMESTAMPDIFF(HOUR, '2005-12-31', '2006-01-01')</i> 返回值为 24
<i>TIME_FORMAT(time, f)</i>	与函数 <i>DATE_FORMAT</i> 一样, 但是仅对时间值有效。函数 <i>TIME_TO_SEC(time)</i> 返回午夜之后的秒值
<i>TO_DAYS(date)</i>	返回从公元 0 年开始计算的天数
<i>UNIX_TIMESTAMP(d)</i>	返回给定日期的当前时间值
<i>WEEK(date)</i>	返回星期值 (1 表示一年当中的第一个星期, 每个星期的首日是星期日)
<i>WEEK (date, mode)</i>	返回星期值 (0~53 或 1~53)。参数 <i>mode</i> 负责设置每个星期的首日是星期几以及星期的定义。例如, <i>mode=0</i> 表示星期始于星期日、返回值 1 代表着一年当中的第 1 个星期。如果没有给出 <i>mode</i> 参数, 则使用服务器的默认设置 (<i>my.cnf/my.ini</i> 文件中的 <i>default_week_format</i> 选项)。 <i>mode</i> 参数的可取值及其含义可以在下面这个网址查到: http://dev.mysql.com/doc/mysql/en/date-and-time-functions.html
<i>WEEKDAY(date)</i>	返回星期几 (0=星期一, 1=星期二, 依此类推)
<i>WEEKOFYEAR(date)</i>	返回历法当中的星期 (1~53)
<i>YEAR(date)</i>	返回年份值
<i>YEARWEEK(date, mode)</i>	根据上下文返回一个整数或字符串, 其内容是年份值和星期值。 <i>mode</i> 的设置与在函数 <i>week</i> 中一样。 <i>YEARWEEK('2005-12-31')</i> 返回 200552

MySQL 的日期/时间函数对数据合法性的检查功能比较弱, 必须由程序员/用户来保证向它们提供的日期/时间参数值是合法的。比如说, 它们会认为 '2005-02-31' 是一个合法的日期值并返回一个“正确的”计算结果, 但这种结果显然没有任何意义。

对返回值是一个时间或一个日期 (或两者兼而有之) 的所有函数来说, 返回值的格式将取决于上下文: 这些函数一般会返回一个字符串 (如, '2005-12-31 23:59:59') ; 但如果被用于数值计算, 它们将返回一个数值, 比如说, *NOW() + 0* 将返回一个 20051231235959 形式的整数值。

1. *ADDDATE()*、*SUBDATE()*、*EXTRACT()*、*TIMESTAMPADD()* 等函数中的时间段

函数 *ADDDATE(date, INTERVAL n i)* 把时间段 *i* 添加 *n* 次到起始日期 *date* 中。可供选用的时间段名字见下面的表格。下面的第三个例子演示了有关函数将如何处理月末日期 (31.12 或 28.2)。

ADDDATE('2005-12-31', INTERVAL 3 DAY) 返回 '2006-01-03'。

ADDDATE('2005-12-31', INTERVAL '3:30' HOUR_MINUTE) 返回 '2005-12-31 03:30:00'。

ADDDATE('2005-12-31', INTERVAL 2 MONTH) 返回 '2006-02-28'。

SUBDATE 函数与 *ADDDATE* 函数类似, 它是把一个给定的时间段减去 *n* 次。

EXTRACT 函数是从时间中抽取出给定的时间段: *EXTRACT(MONTH FROM '2005-12-31')* 返回 12。

TIMESTAMPADD 函数也与 *ADDDATE* 函数类似, 但它使用不同的语法来定义时间段:

TIMESTAMPADD(DAY, 5, '2005-12-31') 返回 '2006-01-05'。

TIMESTAMPDIFF 函数返回的是各种给定时间段两端时间之间的差值:

TIMESTAMPDIFF(DAY, '2005-12-31', '2006-01-15') 返回值为 46。

ADDDATE()、SUBDATE()、EXTRACT()、TIMESTAMPADD()等函数中的时间段

MICROSECOND	n
SECOND	n
MINUTE	n
HOUR	n
DAY	n
MONTH	n
YEAR	n
SECOND_MICROSECOND	'ss.mmmmmmm'
MINUTE_SECOND	'mm:ss'
HOUR_MINUTE	'hh:mm'
HOUR_SECOND	'hh:mm:ss'
DAY_HOUR	'dd hh'
DAY_MINUTE	'dd hh:mm'
DAY_SECOND	'dd hh:mm:ss'
YEAR_MONTH	'yy-mm'

2. 日期/时间数据设置格式

DATE_FORMAT(date, format) 函数可以帮助我们把日期/时间值显示为 MySQL 格式以外的其他格式。下面的两个例子给出了其语法：

DATE_FORMAT('2003-12-31', '%M %d %Y') 返回 'December 31 2003'。

DATE_FORMAT('2003-12-31', '%D of %M') 返回 '31st of December'。

星期几的名字、月份的名字等，总是以英语给出，而不考虑 MySQL 的语言设置 (*language* 选项)。

STR_TO_DATE 是与 *DATE_FORMAT* 相反的函数：*SELECT STR_TO_DATE('December 31 2005', '%M %d %Y')* 返回 '2005-12-31'。

DATE_FORMAT()、TIME_FORMAT()和 FROM_UNIXTIME()函数中的日期符号

%W	星期几	<i>Monday~Sunday</i>
%a	以缩写表示星期几	<i>Mon~Sun</i>
%e	几号	<i>1~31</i>
%d	以两位数表示几号	<i>01~31</i>
%D	具有后缀表示几号	<i>1st、2nd、3rd、4th 等</i>
%w	以数字表示星期几	<i>0 (星期日) ~ 6 (星期六)</i>
%j	以三位数表示一年中的天数	<i>001~366</i>
%U	以两位数表示一年中的第几个星期（以星期日作为星期首日）	<i>00~52</i>
%u	以两位数表示一年中的第几个星期（以星期一作为星期首日）	<i>00~52</i>
%M	月份值	<i>January~December</i>
%b	以缩写表示月份值	<i>Jan~Dec</i>
%c	以数字表示月份值	<i>1~i</i>
%m	以两位数表示月份值	<i>01~12</i>
%Y	以四位数表示年份	<i>2002、2003 等</i>
%y	以两位数表示年份	<i>00、01 等</i>
%%	标示符%	%

关于星期值还要注意：对于每年第一个星期日之前的那些天，%U 将返回 0；从第一个星期日到星期六，它将返回 1；然后是 2；依此类推。符号%u 的返回值与此类似，只是星期首日从星期日变成了星期一而已。

DATE_FORMAT()、TIME_FORMAT()和FROM_UNIXTIME()函数中的时间符号

%f	微秒值	000000~999999
%S 或 %s	以两位数表示秒	00~59
%i	以两位数表示分	00~59
%k	小时值（24 小时制）	0~23
%H	以两位数表示的小时值，0~23 点	00~23
%l	小时值（12 小时制）	1~12
%h 或 %l	以两位数表示的小时值，到 12 点	01~12
%T	24 小时制	00:00:00~23:59:59
%r	12 小时制	12:00:00AM~11:59:59PM
%p	上午或下午	AM、PM

为了避免需要经常重新创建格式字符串，从 MySQL 4.1 版本开始，可以从 *GET_FORMAT()* 函数那里得到些帮助，如下所示：

```
SELECT DATE_FORMAT(NOW(), GET_FORMAT(DATE, 'EUR'))
31.12.2005
```

下面这个表格对 *GET_FORMAT()* 函数的格式代码以及实际排版效果进行了汇总：

GET_FORMAT()函数的格式代码

GET_FORMAT(DATE, 'USA')	'%m.%d.%Y'	12.31.2005
GET_FORMAT(DATE, 'EUR')	'%d.%m.%Y'	31.12.2005
GET_FORMAT(DATE, 'ISO')	'%Y-%m-%d'	2005-12-31
GET_FORMAT(DATE, 'JIS')	'%Y-%m-%d'	2005-12-31
GET_FORMAT(DATE, 'INTERNAL')	'%Y%m%d'	20051231
GET_FORMAT(TIME, 'USA')	'%h:%i:%s %p'	11:59:59 PM
GET_FORMAT(TIME, 'EUR')	'%H:%i:%s'	23:59:59
GET_FORMAT(TIME, 'ISO')	'%H:%i:%s'	23:59:59
GET_FORMAT(TIME, 'JIS')	'%H:%i:%s'	23:59:59
GET_FORMAT(TIME, 'INTERNAL')	'%H%i%s'	235959
GET_FORMAT(DATETIME, 'USA')	'%h:%i:%s %p %h:%i:%s %p'	
GET_FORMAT(DATETIME, 'EUR')	'%H:%i:%s %H:%i:%s'	
GET_FORMAT(DATETIME, 'ISO')	'%H:%i:%s %H:%i:%s'	
GET_FORMAT(DATETIME, 'JIS')	'%H:%i:%s %H:%i:%s'	
GET_FORMAT(DATETIME, 'INTERNAL')	'%H%i%s%H%i%s'	

21.7.6 GROUP BY 函数

下面的函数可以用在 *SELECT* 查询命令里（经常要与 *GROUP BY* 子句一起使用）：

```

USE mylibrary
SELECT catName, COUNT(titleID) FROM titles, categories
WHERE titles.catID=categories.catID
GROUP BY catName
ORDER BY catName
catName          COUNT(titleID)
Children's books      3
Computer books       5
Databases            2
...

```

从 MySQL 4.1 版本开始，排序方式也可以在统计函数里进行定义，如 *MAX(column COLLATE collname)*。

统计函数	
<i>AVG(expr)</i>	计算表达式 <i>expr</i> 的平均值
<i>BIT_AND(expr)</i>	完成表达式 <i>expr</i> 的二进制 <i>AND</i> 操作
<i>BIT_OR(expr)</i>	完成表达式 <i>expr</i> 的二进制 <i>OR</i> 操作
<i>BIT_XOR(expr)</i>	完成表达式 <i>expr</i> 的二进制 <i>XOR</i> 操作
<i>COUNT(expr)</i>	返回表达式 <i>expr</i> 的个数值
<i>COUNT(DISTINCT expr)</i>	返回不同表达式 <i>expr</i> 的个数值
<i>GROUP_CONCAT(expr)</i>	连接字符串（从 MySQL 4.1 版本开始）。表达式 <i>expr</i> 的完整语法如下： [DISTINCT] <i>expr1</i> , <i>expr2</i> ... [ORDER BY <i>column</i> [DESC]] [SEPARATOR'...']。 <i>ORDER BY</i> 把字符串在连接之前分类。 <i>SEPARATOR</i> 规定了分隔符（默认是逗号）。有关 <i>GROUP_CONCAT</i> 的举例参见第 9 章
<i>MAX(expr)</i>	返回表达式 <i>expr</i> 的最大值
<i>MIN(expr)</i>	返回表达式 <i>expr</i> 的最小值
<i>STD(expr)</i>	计算表达式 <i>expr</i> 的标准误差
<i>STDDEV(expr)</i>	同 <i>STD(expr)</i>
<i>SUM(expr)</i>	计算表达式 <i>expr</i> 的总和
<i>VARIANCE(expr)</i>	计算表达式 <i>expr</i> 的均方差（从 MySQL 4.1 版本开始）

21.7.7 其他函数

杂项函数	
<i>BIT_COUNT(x)</i>	返回 <i>x</i> 的二进制表示形式里的“1”位的个数
<i>COALESEC(list)</i>	返回清单中第一个不是 <i>NULL</i> 的元素
<i>LOAD_FILE(filename)</i>	从本地文件系统加载一个文件

管理用函数	
<i>BENCHMARK(n, expr)</i>	执行 <i>n</i> 次表达式 <i>expr</i> ，并测算逝去的时间
<i>CONNECTION_ID()</i>	返回当前数据库连接的 ID 编号值
<i>CURRENT_USER()</i>	以标准签证格式（用 IP 地址取代主机名，如 <i>radha@127.0.0.1</i> ）返回当前用户的名字
<i>DATABASE()</i>	返回当前数据库的名字
<i>FOUND_ROWS()</i>	返回 <i>SELECT LIMIT</i> 查询命令实际找到的记录个数（从 MySQL 4.0 版本开始），但前提是这条 <i>SELECT LIMIT</i> 命令使用了 <i>SQL_CALC_FOUND_ROWS</i> 选项

(续)

管理用函数

<i>GET_LOCK(name, time)</i>	定义一个名字是 <i>name</i> 、持续时间长达 <i>time</i> 秒的锁定；参见第 10 章
<i>IDENTITY()</i>	从 MySQL 4.0 版本开始，与 <i>LAST_INSERT_ID()</i> 函数同义
<i>IS_FREE_LOCK(name)</i>	测试是否可以获得 <i>name</i> 的锁定；如果当前使用着锁定（即在 <i>GET_LOCK</i> 命令执行之前），则返回 0，否则返回 1
<i>LAST_INSERT_ID()</i>	返回数据库当前连接里最近生成的 <i>AUTO_INCREMENT</i> 数值
<i>RELEASE_LOCK(name)</i>	解除对 <i>name</i> 的锁定
<i>SESSION_USER()</i>	与 <i>USER()</i> 函数同义
<i>SYSTEM_USER()</i>	与 <i>USER()</i> 函数同义
<i>USER()</i>	返回当前用户及关联的主机名字（如，“root@localhost”）
<i>VERSION()</i>	以字符串格式返回 MySQL 的版本号

21.8 GIS 数据类型与 GIS 函数

这里讨论的 GIS 数据类型和函数是从 MySQL 4.1 版本开始支持的。列在下面这个表格里的 GIS 数据类型可以用来声明数据表里的数据列（就像声明一个 *INT* 或 *VARCHAR* 数据列那样）。目前，这些数据类型只能用在 MyISAM 类型的数据表中，不能够用于 InnoDB 类型的数据表。作为可选项，几何数据列可以带有 *SPATIAL INDEX* 索引，它独立于使用的 GIS 数据类型。

GIS 数据类型

<i>GEOMETRY</i>	可以接受下面的所有数据类型
<i>POINT</i>	平面上的一个点。例如， <i>POINT(10 10)</i>
<i>MULTIPOINT</i>	平面上的多个点。例如， <i>POINT(10 10, 0 20, -3 2)</i>
<i>LINESTRING</i>	平面上的一个线段。例如， <i>LINESTRING(0 0, 1 1, 3 3)</i>
<i>MULTILINESTRING</i>	平面上的多个线段。例如， <i>MULTILINESTRING((0 0, 5 5), (10 10, 20 20, 40 40), (10 10, 2 0))</i>
<i>POLYGON</i>	平面上的一个闭合多边形，允许有空洞。例如， <i>POLYGON((1 1, 9 1, 9 9, 1 9, 1 1), (3 3, 3 6, 6 6, 6 3, 3 3))</i>
<i>MULTIPOLYGON</i>	平面上的多个多边形。例如， <i>MULTIPOLYGON(((0 0, 5 0, 5 5, 0 5, 0 0)), ((10 10, 30 30, 30 10, 10 10)))</i>
<i>GEOMETRYCOLLECTION</i>	多个几何对象。例如， <i>GEOMETRYCOLLECTION(POINT(100 100), POINT(10 10), LINESTRING(1 1, 100 1, 100 100))</i>

转换函数

<i>ASTEXT(geom)</i>	以 <i>well-known text</i> 格式返回一个集合对象
<i>ASBINARY(geom)</i>	以 <i>well-known binary</i> 格式返回一个集合对象
<i>GEOMFROMTEXT(txt [, srid])</i>	在 <i>well-known text(WKT)</i> 字符串之外创建内部 MySQL 集合格式。作为可选项，可以给出一个对于坐标系统的识别器，它由 MySQL 储存，如果不定义，可以被忽略
<i>GEOMFROMWKB(bindata)</i>	由 <i>WKB</i> 格式中的二进制数据创建内部 MySQL 几何格式

普通几何函数（适用于所有的 GIS 数据类型）

<i>DIMENSION(g)</i>	返回这个对象的尺寸。可能的结果有： 1 表示这个对象为空， 0 表示点（长度=0，面积=0）， 1 表示线（长度>0，面积=0）， 2 表示多边形等，（长度>0，面积>0）
<i>ENVELOPE(g)</i>	返回几何对象的边界框架（见下面）。结果具有数据类型 <i>POLYGON</i>
<i>GEOMETRYTYPE(g)</i>	以字符串返回几何对象的类型：(<i>POINT</i> , <i>LINESTRING</i> , <i>POLYGON</i> , ...)
<i>SRID(g)</i>	返回坐标系统识别器。（尽管 MySQL 数据库储存一个对于所有几何对象的识别器，但它只是确定这个信息的价值）

POINT（点）函数

<i>X(pt)</i>	返回 X 坐标
<i>Y(pt)</i>	返回 Y 坐标

LINESTRING（线段）函数（包括部分 MULTILINestring（多个线段）函数）

<i>GLENGTH(ls)</i>	以浮点数返回线长
<i>ISCLOSED(ls)</i>	如果开始点与结束点是同一个点，则返回 1，否则返回 0
<i>NUMPOINT(ls)</i>	返回直线包含的点的数目
<i>STARTPOINT(ls)</i>	返回开始点（ <i>POINT</i> 对象）
<i>ENDPOINT(ls)</i>	返回最后的点
<i>POINTN(ls, n)</i>	返回第 n 个点

POLYGON（多边形）函数（包括部分 MULTIPOLYGON（多个多边形）函数）

<i>AREA(p)</i>	以浮点数返回多边形的面积
<i>EXTERIORRING(p)</i>	以 <i>LINESTRING</i> 对象返回多边形的外环
<i>EXTERIORRING(p, n)</i>	以 <i>LINESTRING</i> 对象返回多边形的内环
<i>NUMINTERIORRING(p)</i>	返回内环的数目（空洞）

GEOMETRYCOLLECTION 函数

<i>GEOMETRYN(gc, n)</i>	在位置 n 返回几何对象
<i>NUMGEOMETRIES(gc)</i>	返回集合里的对象数目

OpenGIS 以两种方法提供了下面的分析函数。第一种方法仅考虑几何对象的边框（*MBRname*，*MBR* 表示 *minimum bounding rectangle*，最小边界矩形）。第二种方法是精确使用方法，它要计算所有的几何数据（*name*）。MySQL 数据库形式上可以使用这两种方法，但从内部说，它们是一致的，仅考虑几何对象的边框。

GIS 数据类型的分析函数

<i>[MBR]CONTAINS(g1, g2)</i>	如果 g2 完全包含在 g1 中，返回 TRUE
<i>[MBR]WITHIN(g1, g2)</i>	如果 g1 完全包含在 g2 中，返回 TRUE。（ <i>CONTAINS(a, b)</i> 与 <i>WITHIN(b, a)</i> 同义）
<i>[MBR]EQUAL(g1, g2)</i>	如果 g1 与 g2 相等，返回 TRUE
<i>[MBR]INTERSECTS(g1, g2)</i>	如果 g1 与 g2 交叉，返回 TRUE

(续)

GIS 数据类型的分析函数

<i>[MBR]OVERLAPS(g1, g2)</i>	如果 <i>g1</i> 与 <i>g2</i> 重叠, 返回 TRUE
<i>[MBR]TOUCHES(g1, g2)</i>	如果 <i>g1</i> 与 <i>g2</i> 相接, 返回 TRUE
<i>[MBR]DISJOINT(g1, g2)</i>	如果 <i>g1</i> 与 <i>g2</i> 既不重叠、也不相接, 返回 TRUE

21.9 与存储过程和触发器有关的语言元素

几乎可以在存储过程或触发器中使用本章讨论的所有 SQL 命令或函数。MySQL 数据库还提供了一些附加的语言元素, 可以使用这些语言元素来声明系统变量和光标、编写循环语句、构造查询命令等。这些语言元素的语法见下面这些表格。

命令、循环和查询

<i>[blockname:] BEGIN</i>	<i>BEGIN</i> 开始一个语句块; <i>END</i> 结束这个语句块。先声明变量、光标等,
<i>DECLARE variables, cursors,</i>	然后是必要的 SQL 命令和附加语言元素
<i>conditions, handlers etc; commands ...;</i>	
<i>END [blockname];</i>	
<i>LEAVE blockname;</i>	如果语句块有名字 (<i>blockname</i>), 就可以用 <i>LEAVE</i> 命令提前跳出它。 程序将从 <i>END blockname</i> 语句的下一条命令开始继续执行
<i>RETURN result;</i>	如果这是一个函数, 就可以用 <i>RETURN</i> 来返回它的调用结果并结束本次调用

查询

<i>IF condition1 THEN</i>	<i>IF/IF END</i> 创建一个简单的条件分支。允许使用任意个数的 <i>ELSE-IF</i> 子句, 但只允许有一个用来结束整个语句的 <i>ELSE</i> 分支
<i> commands ...;</i>	
<i>{ELSE IF condition2 THEN</i>	
<i> commands ...;}</i>	
<i>{ELSE</i>	
<i> commands ...;}</i>	
<i>END IF;</i>	
<i>CASE expression</i>	用 <i>CASE/END CASE</i> 语句构成的多重分支结构有着更好的可读性。这
<i>When value1 THEN</i>	个语句有好几种变体。允许使用任意多个 <i>WHEN</i> 子句 (每一个 <i>CASE</i> 分
<i> commands ...;</i>	支都可以改写为一个 <i>IF/END IF</i> 结构)
<i>{ELSE</i>	
<i> commands ...;}</i>	
<i>END CASE;</i>	

循环

<i>[loopname:] REPEAT</i>	<i>REPEAT</i> 循环将一直循环执行到给定条件得到满足为止 (至少一次)
<i> commands ...;</i>	
<i> UNTIL condition</i>	
<i> END REPEAT [loopname];</i>	
<i>[loopname:] WHILE condition DO</i>	只要给定条件得到满足, <i>WHILE</i> 循环就将循环一直执行下去。在第一
<i> commands ...;</i>	次循环之前, 如果条件表达的求值结果是 <i>FALSE (0)</i> , 则循环根本就不
<i> END WHILE [loopname];</i>	执行

(续)

循环

<i>loopname: LOOP</i>	<i>LOOP/END LOOP</i> 创建一个无限循环。它必须使用 <i>BREAK</i> 来结束
<i>commands ...;</i>	
<i>END LOOP loopname;</i>	
<i>LEAVE loopname;</i>	提前结束一个循环
<i>ITERATE loopname;</i>	在不改变循环条件的情况下重复执行一次循环体

下面这些 *DECLARE* 指令必须出现在语句块的开头，并且必须按这里给出顺序执行（先声明各有关变量、再声明各有关光标，然后是一些条件表达式和句柄例程）。

声明各有关变量、光标、条件表达式、句柄处理例程（*DECLARE* 命令）

<i>DECLARE varname1, varname2, ... datatype [DEFAULT value];</i>	把局部变量 <i>varname1</i> 、 <i>varname2</i> 等声明为特定的数据类型（例如 <i>INT</i> ）。允许（但不要求）为这些变量设置一个默认值。局部变量只能在它们被定义的语句块里使用，它们的名字不能与数据表或数据列的名字相同
<i>DECLARE cursorname CURSOR FOR select-command;</i>	声明一个光标，这个光标将被用来处理 <i>SELECT</i> 命令的查询结果
<i>DECLARE name CONDITION FOR condition1, c2, c3 ...;</i>	声明一个或多个出错条件表达式。出错条件表达式可以是以下几种形式： <i>SQLSTATE 'code'</i> <i>n</i> (MySQL 出错代码) <i>SQLWARNING</i> <i>NOT FOUND</i> <i>SQLEXCEPTION</i>
<i>DECLARE CONTINUE / EXIT HANDLER FOR condition1, c2, c3 ... command;</i>	声明一个出错处理句柄例程。 <i>CONTINUE</i> 选项的效果是：出错后继续执行后面的代码； <i>EXIT</i> 选项的效果是：出错后退出当前语句块。整个语句块将从 <i>command</i> 部分开始执行。此外，在声明出错条件表达式 (<i>condition1</i> 、 <i>c2</i> 、 <i>c3</i> 等) 时还允许使用 <i>CONDITION</i> 语法

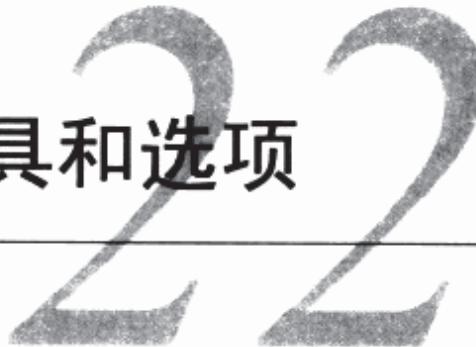
下面这个表格演示了声明和使用一个光标的全过程：

光标的声明和使用

<i>DECLARE done INT DEFAULT 0;</i>	为句柄例程定义一个变量
<i>DECLARE var1, var2 ... datatype;</i>	为 <i>SELECT</i> 命令声明一些变量
<i>DECLARE mycursor CURSOR FOR select command;</i>	声明一个光标
<i>DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1;</i>	声明一个句柄例程，这个句柄例程将在最后一条结果记录被读取之后被触发
<i>OPEN mycursor;</i>	激活（或者说打开）光标
<i>myloop: LOOP FETCH mycursor INTO var1, var2 ...; IF done=1 THEN LEAVE myloop; END IF; END LOOP myloop;</i>	这个循环将把 <i>SELECT</i> 命令的结果记录依次读入 <i>var1</i> 、 <i>var2</i> 等变量。如果已经到达最后一条结果记录，刚才定义的句柄例程将被触发并把 <i>done</i> 赋值为 1，而 <i>done=1</i> 将导致这个循环退出执行
<i>CLOSE mycursor;</i>	关闭光标

第 22 章

MySQL 工具和选项



本章将对最重要的 MySQL 工具及其选项和功能进行汇总，将讨论服务器程序 mysqld、命令解释器 mysql 及管理工具 mysqladmin、myisamchk 等。

这些工具有许多共同的选项，在对配置文件的处理方式上也完全一样。因此，在这一章里的讨论将从它们共同具备的特点开始。

22.1 概述

本章将要介绍的命令有一个共同的特点：它们都可以在一个命令窗口（Windows 系统）或一个控制台窗口（UNIX/Linux 系统）作为一个外部程序来启动运行。这些程序的全部操作都是在文本模式下进行的，所以使用起来不是很方便。但这些命令很适合用在脚本里以自动化的方式完成各种系统管理任务。

下面这个表格列出了本章将要介绍的命令以及它们的基本用途。

MySQL 服务器和配套管理工具

mysqld	MySQL 服务器程序。这个程序几乎总是随系统一起启动：在 Windows 环境下，它通常是作为一项系统服务随系统启动；在 UNIX/Linux 系环境下，它通常是由一个 Init-V 启动脚本调用 mysqld_safe 脚本随系统启动
mysqld_safe	在 UNIX/Linux 系统上安全地启动 MySQL 服务器
mysql	以交互方式执行 SQL 命令的工具程序
mysqladmin	一个用来完成各种系统维护和管理任务（查看工作状态、设置各种权限、执行关机命令等）的工具程序
mysqldump	把一个 MySQL 数据库的内容保存为一个文本文件
mysqlimport	把文本文件里的数据读入 MySQL 数据表
mysqlshow	用来查看关于数据库、数据表和数据列的信息
myisamchk	检查和修复 MyISAM 数据表文件
myisampack	对 MyISAM 数据表进行压缩并生成只读的数据表

22.2 通用选项和配置文件

本章将要介绍的程序有一个共同点：某些特定的选项是在运行这些程序时都可以使用的。这些选项可以在这些程序共用的一个配置文件里给出，因而可以减少在运行这些程序时的打字量。

22.2.1 通用选项

有些选项在 MySQL 服务器（mysqld）和 MySQL 客户端工具程序（mysql、mysqladmin、mysqldump、

`mysqlimport` 等) 里都有着同样的含义和作用, 把这些选项称为 MySQL 工具程序的“通用”选项。作为 UNIX/Linux 系统上的一种常见现象, MySQL 工具程序的通用选项也有长、短两种格式之分, 前者的标志是有两个连字符 (--) 作为前缀, 后者的标志是只有一个连字符 (-) 作为前缀。请注意, 这些选项的短格式是区分字母大小写的。

MySQL 服务器 (`mysqld`) 和 MySQL 客户端工具程序 (`mysql`、`mysqladmin` 等) 的通用选项

<code>--help</code>	显示一些简短的帮助信息
<code>--print-defaults</code>	查看各个选项的默认值; 这些默认值可以来自选项文件或环境变量
<code>--nodefaults</code>	在启动时不读取任何配置文件
<code>--defaults-file = filename</code>	在启动时只读取这个配置文件
<code>--defaults-extra-file = filename</code>	在启动时先读入全局级配置文件, 然后读入这个选项给定的 <code>filename</code> 文件, 最后 (仅适用于 UNIX/Linux 系统) 读入用户级配置文件
<code>--port = n</code>	为 MySQL 程序指定一个 TCP/IP 通信端口 (通常是 3306 号端口)
<code>--socket = filename</code>	为 MySQL 客户程序与服务器之间的本地通信指定一个套接字文件 (仅适用于 UNIX/Linux 系统; 默认设置一般是 <code>/var/lib/mysql/mysql.sock</code> 文件)。在 Windows 环境下, 如果 MySQL 客户与服务器是通过命名管道进行通信的, <code>--sock</code> 选项给出的将是那个命名管道的名字 (默认设置是 <code>MySQL</code>)
<code>--version</code>	显示 MySQL 程序的版本信息

MySQL 客户端工具程序的通用选项

<code>-u un</code>	<code>--user = username</code>	连接 MySQL 服务器时使用的 MySQL 用户名
<code>-p</code>	<code>--password</code>	在启动后立刻提示用户输入密码
<code>-pxxx</code>	<code>--password = xxx</code>	直接传递密码。 <code>-p</code> 的后面不允许有空格, 这是本选项与其他选项不同的地方。与以交互方式输入密码时的情况相比, 这种做法虽然比较简便, 却会带来相当大的安全风险, 所以应该尽量地避免。在某些操作系统上, 任何用户都可以从进程表看到这个密码
<code>-h hn</code>	<code>--host = hostname</code>	MySQL 服务器主机的名字或 IP 地址 (默认设置是 <code>localhost</code> , 意思是 MySQL 服务器运行在本地计算机上)
	<code>--protocol = name</code>	决定用户与服务器间的通信协议。允许的设置有 <code>tcp</code> 、 <code>socket</code> 、 <code>pipe</code> 和 <code>memory</code>
<code>-W</code>	<code>--pipe</code>	使用一个命名管道 (named pipe) 来连接 MySQL 服务器。这个选项仅适用于 Windows XP/2000 环境下的本地连接, 而且 <code>my.ini</code> 文件必须包含着 <code>enable-named-pipe</code> 选项
<code>-C</code>	<code>--compress</code>	对数据进行压缩以使客户程序与服务器之间的数据流量减至最小
	<code>--default-character-set = name</code>	指定与 MySQL 服务器进行通信时使用的默认字符集 (例如 <code>latin1</code> 或 <code>utf8</code>)
	<code>--character-sets-dir = "dir"</code>	给出存放着字符集文件的目录 (例如 " <code>C:/Programs/MySQL/ MySQL Server 5.0/share.charssets</code> ")。这个选项只在需要的字符集没有被预编译到 MySQL 客户程序里的时候才必须给出

为了与 MySQL 服务器建立起连接, MySQL 客户程序在启动时至少需要给出 `-u` 和 `-p` 两个选项, 如下所示:

```
> mysql -u username -p
Enter Password: XXXXXX
```

当然,如果 MySQL 服务器不要求客户必须提供密码,客户不给出密码也可以建立起连接。关于 MySQL 用户和权限管理问题的深入讨论可以参见本书第 11 章。

注意 在 Windows 系统上执行 SQL 命令,如果创建带有选项的目录,一定要把路径中的反斜线字符 (\) 替换为斜线 (/) 字符;如果文件名里包含空格,还必须把整个路径用引号括起来,如--basedir = "C:/Programs/MySQL/MySQL Server 5.0"。

22.2.2 设置配置文件的选项

如果需要经常使用某些特定的选项去启动各种 MySQL 程序,把那些选项统一归纳在几个配置文件里可以减少许多麻烦。mysql、mysqladmin、mysqld、mysqldump、mysqlimport、myisamchk 和 mysqld_safe 等 MySQL 工具程序都可以从配置文件里读取它们所需要的选项。

下面这个表格列出了 MySQL 配置文件的存放位置,MySQL 程序会在启动时按照这个表格里的顺序去依次读取各有关配置文件。如果选项的设置值发生冲突,后读入的设置值将优先于先读入的设置值。(MySQL 工具程序是否读取配置文件以及读取哪些配置文件还要取决于 --no-defaults、--defaults- 和 --defaults-extra-file 等选项的具体设置情况;参见 22.2.1 节里的说明)

选项的级别	Windows 系统	UNIX/Linux 系统
全局级选项	较新的 MySQL 版本: C:\Programs\MySQL\ MySQLServer n.n\my.ini 较早的 MySQL 版本: C:\my.ini 或 C:\Windows \my.ini	/etc/my.cnf
用户级选项 (仅适用于各种客户端程序)		~/.my.cnf
服务器级选项 (仅适用于 mysqld 程序)	DATADIR\my.cnf	DATADIR/my.cnf

表中的 DATADIR 代表 MySQL 数据目录的默认路径,这个路径是在编译 MySQL 服务器时给定的。在 Windows 系统上,这个路径通常是 C:\mysql\data 目录;在 UNIX/Linux 系统上,这个路径通常是在 /var/lib/mysql 目录。注意,DATADIR 与 MySQL 实际存放数据库文件时使用的目录不必相同,后者往往由数据库管理员在启动 MySQL 服务器时使用 --datadir 选项另行指定,而各种 MySQL 工具程序会先去 DATADIR 目录里寻找配置文件再处理 --datadir 选项。

各种 MySQL 配置文件的选项设置语法都是一样的,如下所示:

```
# Comment
[program name]
option1      # equivalent to: -option1
option2=abc   # equivalent to: --option2=abc
```

配置文件里的选项按各有关程序分组列出,方括号里的 *program name* 必须被替换为具体的程序名。对有关选项进行分组的原则是:

- 适用于所有客户端程序(即不包括服务器程序 mysql)的选项集中放在 [client] 选项组里。
- 仅适用于服务器程序的选项集中放在 [mysqld] 选项组里。(mysql 程序的选项也可以放在 [server] 选项组里。)
- 仅适用于程序 xyz 的选项集中放在 [xyz] 选项组里。

在配置文件给出的选项 (*option*) 必须使用长格式的选项名,但不带连字符 (--) 前缀。(比如说,如

果把--host 选项放在配置文件里，就要把它写成 host）。选项的参数要用等号（=）给出。

下面这个例子给出了一个典型的配置文件：

```
# configuration file /etc/my.cnf (Unix/Linux) or my.ini (Windows)
# options for all MySQL tools
[client]
user=username
password=xxx
host=uranus.sol
# options for mysqldump
[mysqldump]
force
# options for mysql (command interpreter)
[mysql]
safe-updates
select_limit=100
```

注意 如果用户在配置文件里对某个特定程序的选项做出了修改，就必须重新启动那个程序才能让那些修改生效。这一点对MySQL服务器来说更是如此；即对[mysqld]选项组的修改必须重新启动MySQL服务器才能生效。

注意，[client]选项组里的选项必须是所有的MySQL客户端程序都支持的。如果某个MySQL工具发现[client]选项组里有自己不认识的选项，它将立刻退出运行并报告出错。

在UNIX/Linux系统上，如果打算使用~/.my.cnf文件里的用户级选项（比如说，把密码放在那里），还应该保证其他用户不能读取以下文件：

```
user$ chmod 600 ~/.my.cnf
```

在Windows系统上，如果在设置选项的时候给出Windows路径或目录，一定要把路径中的反斜线字符（\）替换为（/）或（\\）。（MySQL的Windows版本把反斜线用作转义前导字符。）

配置文件里的路径名不需要用引号括起来（即使文件路径里包含着空格也是如此），如下所示：

```
basedir = C:/Programs/MySQL/MySQL Server 5.0/
```

注意以上几条规则与在命令行上使用--option = ...直接设置有关选项时的不同之处。

可以使用 my_print_defaults 程序去查看配置文件里的[grp]选项组的设置情况。这个程序非常适合用来编写各种定制的脚本。

22.2.3 内存量的表示方法

对于用来设置各种缓存区长度的选项或变量，可以使用字母 K、M 和 G 来分别代表 KB（=1024 字节）、MB（=1024KB）和 GB（=1024MB）等计量单位。也就是说，key_buffer_size = 16M、key_buffer_size = 16384K 和 key_buffer_size = 16777216 的设置效果都是一样的。

22.2.4 环境变量（系统变量）

有些 MySQL 选项还可以通过操作系统级的环境变量来设置。（在 Windows 系统上，这些变量通常被称为系统变量。）

这里把这些变量当中最重要的几个列在了下面的表格里。在 Windows 系统上，可以通过控制面板中的“系统”对话框（参见第 4 章中的图 4-1）设置这些变量；在 UNIX/Linux 系统上，这些变量可以通过启动脚本（如：/etc/profile 或~/.profile）里的 export 命令来定义。根据具体使用的是哪一种 shell，可能需要使用 declare -x 或 setenv 命令来代替 export。

mysql、mysqladmin、mysqld、mysqldump 等程序会用到的部分环境变量和输入

MYSQL_TCP_PORT	与 MySQL 服务器进行 TCP/IP 通信时使用的端口号（通常是 3306 号端口）。
MYSQL_UNIX_PORT	在 Linux/UNIX 环境下与 MySQL 服务器进行本地通信使用的套接字文件（如 /var/mysql.sock 文件）
TMPDIR	用来存放临时文件的目录：这个目录还可以用来存放各种临时数据表
USER	MySQL 用户名

选项设置值的优先顺序

各有关选项的设置值按以下顺序读入：环境变量、配置文件、程序启动选项。在选项的设置值发生冲突的时候，后读入的设置值将优先于先读入的设置值。比如说，程序启动选项的设置值将优先于环境变量给出的设置值。

22.2.5 选项设置规则

下面是一些与选项设置工作有关的规则和注意事项：

- 允许使用连字符来代替变量名里的下划线字符，即可以把 --variable_name = 123 写成 --variable-name = 123。
- 允许使用同一选项的多种形式来启用/禁用相关功能：--option、--option=1 和--enable-option 都可以用来启用某给定功能；--option=0、--disable-option 和--skip-option 都可以用来禁用某给定功能。
注意，如果 skip 是选项名本身的一部分（比如--skip-grant-table），像--enable-grant-table 或--grant-table=0 这样给出选项将不起作用。
- 有些 MySQL 服务器变量允许最终用户在运行时修改，但最终用户做出的修改仅适用于他本人的当前连接。比如说，只须简单地执行一条 SQL 命令 SET read_buffer_size=16M 就可以改变本人的当前连接上的 read_buffer_size 设置值。对于这类变量，数据库管理员可以在启动 MySQL 服务器时用前缀--maximum 为它们设置一个上限值，例如--maximum-read_buffer_size=32M。
- MySQL 的每一个新版本都会增加一些新的选项。如果不希望让编写的脚本只能用在某个特定的 MySQL 版本里，可以使用新选项--loose 作为一个前缀（例如--loose-optionname =3）。这样，如果--loose 后面的选项存在，它就可以得到正确的设置；如果该选项不存在，这项设置将被忽略并且不报告出错。

22.3 mysqld 程序（服务器）

接下来的几个小节对最重要的 mysqld 选项进行了汇总，这些选项将按它们的基本功能和用途分类。

提示 mysqld 程序完整的选项清单可以通过执行 mysqld --verbose --help 命令获得，那份清单非常长。

MySQL 文档没有把 mysqld 选项集中在同一个地方进行介绍，对它们的描述散见于不同的章节。下面这些网址可以帮助大家找到那些最重要的 mysqld 选项：

<http://dev.mysql.com/doc/mysql/en/server-options.html>
<http://dev.mysql.com/doc/mysql/en/command-line-options.html>
<http://dev.mysql.com/doc/mysql/en/privileges-options.html>
<http://dev.mysql.com/doc/mysql/en/replication-options.html>
<http://dev.mysql.com/doc/mysql/en/innodb-start.html>

作为 MySQL 服务器程序, mysqld 几乎总是随系统一起启动: 在 Windows 环境下, 它通常是作为一项系统服务随系统启动; 在 UNIX/Linux 系环境下, 它通常是由一个 Init-V 启动脚本调用 mysqld_safe 脚本随系统启动。既然 mysqld 程序的启动不需要有人的直接参与, 我们也就无法直接为 mysqld 程序设置各种选项。换句话说, mysqld 程序的选项几乎都是通过配置文件设置的。因此, 在后面的有关表格里, 将按照各有关 mysqld 选项在配置文件里的格式(即不带“--”前缀)来列出它们。

不过, --default-extra-file 和 --user 选项却是两个例外: 它们必须直接传递给 mysqld 程序或 mysqld_safe 脚本, 不能通过配置文件给出。这两个选项的作用是:

- 如果在启动 mysqld 程序时使用了 --default-extra-file=filename 选项, 它将在读取完所有其他的配置文件之后去读取本选项指定的文件。
- 如果在启动 mysqld 程序时使用了 --user=name 选项, 它在启动后将在本选项给定的 UNIX/Linux 账户下执行。这里有一个关键的细节: 要想让 MySQL 服务器在启动后改用另一个账户执行, 就必须使用 root 账户来启动 mysqld 程序(否则它将无法切换到另一个账户)。在默认的情况下, mysqld_safe 脚本将使用 --user = mysql 选项来启动 mysqld 程序。

22.3.1 基本选项

mysqld 程序——目录和文件

basedir = path	使用给定目录作为根目录(安装目录)
character-sets-dir = path	给出存放着字符集文件的目录
datadir = path	从给定目录读取数据库文件
pid-file = filename	为 mysqld 程序指定一个存放其进程 ID 的文件(仅适用于 UNIX/Linux 系统); Init-V 脚本需要使用这个文件里的进程 ID 结束 mysqld 进程
socket = filename	为 MySQL 客户程序与服务器之间的本地通信指定一个套接字文件(仅适用于 UNIX/Linux 系统; 默认设置一般是 /var/lib/mysql/mysql.sock 文件)。在 Windows 环境下, 如果 MySQL 客户与服务器是通过命名管道进行通信的, --socket 选项给出的是该命名管道的名字(默认设置是 MySQL)
lower_case_table_name=1/0	新目录和数据表的名字是否只允许使用小写字母; 这个选项在 Windows 环境下的默认设置是 1(只允许使用小写字母)

mysqld 程序——语言设置

character-sets-server = name	新数据库或数据表的默认字符集。为了与 MySQL 的早期版本保持兼容, 这个字符集也可以用 --default-character-set 选项给出; 但这个选项已经显得有点儿过时了
collation-server = name	新数据库或数据表的默认排序方式
language = name	用指定语言显示出错信息

mysqld 程序——通信、网络、信息安全

enable-named-pipes	允许 Windows 2000/XP 环境下的客户和服务器使用命名管道(named pipe)进行通信。这个命名管道的默认名字是 MySQL, 但可以用 --socket 选项来改变
local-infile [=0]	允许/禁止使用 LOAD DATA LOCAL 语句来处理本地文件
myisam-recover [=opt1, opt2, ...]	在启动时自动修复所有受损的 MyISAM 数据表。这个选项的可取值有 4 种: DEFAULT、BACKUP、QUICK 和 FORCE; 它们与 myisamchk 程序的同名选项作用相同

(续)

mysqld 程序——通信、网络、信息安全

old-passwords	使用 MySQL 3.23 和 4.0 版本中的老算法来加密 mysql 数据库里的密码（默认设置是使用从 MySQL 4.1 版本开始引入的新加密算法）
port = n	为 MySQL 程序指定一个 TCP/IP 通信端口（通常是 3306 号端口）
safe-user-create	只有在 mysql.users 数据表上拥有 Insert 权限的用户才能使用 GRANT 命令去创建一个新用户；这是一种双保险机制（那位用户还必须具备 Grant 权限才能执行 GRANT 命令）
shared-memory	允许使用共享内存（shared memory）进行通信（仅适用于 Windows）
shared-memory-base-name= name	给共享内存块起一个名字（默认的名字是 MySQL）
skip-grant-tables	不使用 mysql 数据库里的信息来进行访问控制（警告：这将允许任何用户去修改任何数据库）
skip-host-cache	不使用高速缓存区来存放主机名和 IP 地址的对应关系
skip-name-resolve	不把 IP 地址解析为主机名；与访问控制（mysql.users 数据表）有关的检查全部通过 IP 地址进行
skip-networking	只允许通过一个套接字文件（UNIX/Linux 系统）或通过命名管道（Windows 系统）进行本地连接，不允许 TCP/IP 连接；这提高了安全性，但阻断了来自网络的外部连接和所有的 Java 客户程序（Java 客户即使在本地连接里也使用 TCP/IP）
user = name	mysqld 程序在启动后将在给定 UNIX/Linux 账户下执行；mysqld 必须从 root 账户启动才能在启动后切换到另一个账户下执行；mysqld_safe 脚本将默认使用 --user = mysql 选项来启动 mysqld 程序

mysqld 程序——内存管理、优化、查询缓存区

bulk_insert_buffer_size = n	为一次插入多条新记录的 INSERT 命令分配的缓存区长度（默认设置是 8MB）
key_buffer_size = n	用来存放索引区块的 RAM 值（默认设置是 8 MB）
join_buffer_size = n	在参加 JOIN 操作的数据列没有索引时为 JOIN 操作分配的缓存区长度（默认设置是 128KB）
max_heap_table_size = n	HEAP 数据表的最大长度（默认设置是 16MB）；超过这个长度的 HEAP 数据表将被存入一个临时文件而不是驻留在内存里
max_connections = n	MySQL 服务器同时处理的数据库连接的最大数量（默认设置是 100）
query_cache_limit = n	允许临时存放在查询缓存区里的查询结果的最大长度（默认设置是 1MB）
query_cache_size = n	查询缓存区的最大长度（默认设置是 0，不开辟查询缓存区）
query_cache_type = 0/1/2	查询缓存区的工作模式：0，禁用查询缓存区；1，启用查询缓存区（默认设置）；2，“按需分配”模式，只响应 SELECT SQL_CACHE 命令
read_buffer_size = n	为从数据表顺序读取数据的读操作保留的缓存区的长度（默认设置是 128KB）；这个选项的设置值在必要时可以用 SQL 命令 SET SESSION read_buffer_size = n 命令加以改变
read_rnd_buffer_size = n	类似于 read_buffer_size 选项，但针对的是将按某种特定顺序（比如使用了 ORDER BY 子句的查询）输出的查询结果记录（默认设置是 256KB）
sort_buffer = n	为排序操作分配的缓存区的长度（默认设置是 2MB）；如果这个缓存区太小，则必须创建一个临时文件来进行排序
table_cache = n	同时打开的数据表的数量（默认设置是 64）
tmp_table_size = n	临时 HEAP 数据表的最大长度（默认设置是 32MB）；超过这个长度的临时数据表将被转换为 MyISAM 数据表并存入一个临时文件

22.3.2 与日志和镜像功能有关的选项

mysqld 程序——日志

<code>log [= file]</code>	把所有的连接以及所有的 SQL 命令记入日志（通用查询日志）；如果没有给出 <code>file</code> 参数，MySQL 将在数据库目录里创建一个 <code>hostname.log</code> 文件作为这种日志文件（ <code>hostname</code> 是服务器主机名）
<code>log-slow-queries [= file]</code>	把执行用时超过 <code>long_query_time</code> 变量值的查询命令记入日志（慢查询日志）；如果没有给出 <code>file</code> 参数，MySQL 将在数据库目录里创建一个 <code>hostname-slow.log</code> 文件作为这种日志文件（ <code>hostname</code> 是服务器主机名）
<code>long_query_time = n</code>	慢查询的执行用时上限（默认设置是 10s）
<code>long_queries_not_using_indexes</code>	把慢查询以及在执行时没有使用索引的查询命令全都记入日志（其余同 <code>--log-slow-queries</code> 选项）
<code>log-bin [= filename]</code>	把对数据进行修改的所有 SQL 命令（也就是 <code>INSERT</code> 、 <code>UPDATE</code> 和 <code>DELETE</code> 命令）以二进制格式记入日志（二进制变更日志， <code>binary update log</code> ）。这种日志的文件名是 <code>filename.n</code> 或默认的 <code>hostname.n</code> ，其中 <code>n</code> 是一个 6 位数字的整数（日志文件按顺序编号）
<code>log-bin-index = filename</code>	二进制日志功能的索引文件名。在默认的情况下，这个索引文件与二进制日志文件的名字相同，但后缀名是 <code>.index</code> 而不是 <code>.nnnnnn</code>
<code>max_binlog_size = n</code>	二进制日志文件的最大长度（默认设置是 1GB）。在前一个二进制日志文件里的信息量超过这个最大长度之前，MySQL 服务器会自动提供一个新的二进制日志文件接续上
<code>binlog-do-db = dbname</code>	只把给定数据库里的变化情况记入二进制日志文件，其他数据库里的变化情况不记载。如果需要记载多个数据库里的变化情况，就必须在配置文件使用多个本选项来设置，每个数据库一行
<code>binlog-ignore-db = dbname</code>	不把给定数据库里的变化情况记入二进制日志文件
<code>sync_binlog = n</code>	每经过 <code>n</code> 次日志写操作就把日志文件写入硬盘一次（对日志信息进行一次同步）。 <code>n=1</code> 是最安全的做法，但效率最低。默认设置是 <code>n=0</code> ，意思是让操作系统来负责二进制日志文件的同步工作
<code>log-update [= file]</code>	把日志信息以文本格式记入日志文件（仅适用于 MySQL 4.1 及更早的版本）。MySQL 从 5.0 版本开始不再支持这种日志功能。新版本都使用二进制日志功能
<code>log_error = file</code>	记载出错情况的日志文件名（出错日志）。这种日志功能无法禁用。如果没有给出 <code>file</code> 参数，MySQL 将使用 <code>hostname.err</code> 作为这种日志文件的名字

mysqld 程序——镜像（主控镜像服务器）

<code>server_id = n</code>	给服务器分配一个独一无二的 ID 编号： <code>n</code> 的取值范围是 1~2 ³¹
<code>log-bin = name</code>	启用二进制日志功能。这种日志的文件名是 <code>filename.n</code> 或默认的 <code>hostname.n</code> ，其中 <code>n</code> 是一个 6 位数字的整数（日志文件按顺序编号）
<code>binlog-do/ignore-db = dbname</code>	只把给定数据库里的变化情况记入二进制日志文件
	不把给定数据库里的变化情况记入二进制日志文件

mysqld 程序——镜像（从属镜像服务器）

<code>server_id = n</code>	给服务器分配一个唯一的 ID 编号
<code>log-slave-updates</code>	启用从属服务器上的日志功能，使这台计算机可以用来构成一个镜像链（ <code>A→B→C</code> ）
<code>master-host = hostname</code>	主控服务器的主机名或 IP 地址。如果从属服务器上存在 <code>master.info</code> 文件（镜像关系定义文件），它将忽略此选项

(续)

mysqld 程序——镜像（从属镜像服务器）

master-user = replicusername	从属服务器用来连接主控服务器的用户名。如果从属服务器上存在 master.info 文件，它将忽略此选项
master-password = pword	从属服务器用来连接主控服务器的密码。如果从属服务器上存在 master.info 文件，它将忽略此选项
master-port = n	从属服务器用来连接主控服务器的 TCP/IP 端口（默认设置是 3306 号端口）。如果从属服务器上存在 master.info 文件，它将忽略此选项
master-connect-retry = n	如果与主控服务器的连接没有成功，则等待 n 秒（s）后再进行重试（默认设置是 60s）。如果从属服务器上存在 master.info 文件，它将忽略此选项
master-ssl-xxx = xxx	对主、从服务器之间的 SSL 通信进行配置
read-only = 0/1	0：允许从属服务器独立地执行 SQL 命令（默认设置）； 1：从属服务器只能执行来自主控服务器的 SQL 命令
relay-log-purge = 0/1	1：把处理完的 SQL 命令立刻从中继日志文件里删除（默认设置）； 0：不把处理完的 SQL 命令立刻从中继日志文件里删除
replicate-do-table = dbname.tablename	只对给定数据表进行镜像处理；如果想对多个数据表进行镜像处理，就必须在配置文件使用多个本选项来设置，每个数据表一行
replicate-wild-do-table = dbname.tablename	与 --replicate-do-table 选项的含义和用法相同，但数据库和数据表的名字里允许出现通配符“%”（例如：test%.%——对名字以“test”开头的所有数据库里的所有数据表进行镜像处理）
replicate-do-db = dbname	只对这个数据库进行镜像处理
replicate-ignore-table = dbname.tablename	不对这个数据表进行镜像处理
replicate-wild-ignore-table = dbn.tablen	不对这些数据表进行镜像处理
replicate-ignore-db = dbname	不对这个数据库进行镜像处理
replicate-rewrite-db = db1name->db2name	把主控服务器上的 db1name 数据库镜像处理为从属服务器上的 db2name 数据库
report-host = hostname	从属服务器的主机名；这项信息只与 SHOW SLAVE HOSTS 命令有关——主控服务器可以用这条命令生成一份从属服务器名单
slave-compressed-protocol = 1	主、从服务器使用压缩格式进行通信——如果它们都支持这么做的话
slave-skip-errors = n1,n2,... 或 all	即使发生出错代码为 n1、n2 等的错误，镜像处理工作也继续进行（即不管发生什么错误，镜像处理工作也继续进行）。如果配置得当，从属服务器不应该在执行 SQL 命令时发生错误（在主控服务器上执行出错的 SQL 命令不会被发送到从属服务器上做镜像处理）；如果不使用 slave-skip-errors 选项，从属服务器上的镜像工作就可能因为发生错误而中断，中断后需要有人工参与才能继续进行

22.3.3 InnoDB 配置选项**mysqld 程序——InnoDB——基本设置、表空间文件**

skip-innodb	不加载 InnoDB 数据表驱动程序——如果用不着 InnoDB 数据表，可以用这个选项节省一些内存
innodb-file-per-table	为每一个新数据表创建一个表空间文件而不是把数据表都集中保存在中央表空间里（后者是默认设置）。这个选项始见于 MySQL 4.1

(续)

mysqld 程序——InnoDB——基本设置、表空间文件**innodb_open_files = n**

InnoDB 数据表驱动程序最多可以同时打开的文件数（默认设置是 300）。如果使用了 `innodb_file_per_table` 选项并且需要同时打开很多数据表的话，这个数字很可能需要加大。

innodb_data_home_dir = p

InnoDB 主目录，所有与 InnoDB 数据表有关的目录或文件路径都相对于这个路径。在默认的情况下，这个主目录就是 MySQL 的数据目录。

innodb_data_file_path = ts

用来容纳 InnoDB 数据表的表空间：可能涉及一个以上的文件；每一个表空间文件的最大长度都必须以字节（B）、兆字节（MB）或千兆字节（GB）为单位给出；表空间文件的名字必须以分号隔开；最后一个表空间文件还可以带有一个 `autoextend` 属性和一个最大长度（`max:n`）。例如，`ibdata1:1G:ibdata2:1G:autoextend:max:2G` 的意思是：表空间文件 `ibdata1` 的最大长度是 1GB，`ibdata2` 的最大长度也是 1GB，但允许它扩充到 2GB。除文件名以外，还可以用硬盘分区的设备名来定义表空间，此时必须给表空间的最大初始长度值加上 `newraw` 关键字做后缀，给表空间的最大扩充长度值加上 `raw` 关键字做后缀（例如`/dev/hdb1:20Gnewraw` 或 `/dev/hdb1:20Graw`）；MySQL 4.0 及更高版本的默认设置是 `ibdata1:10M:autoextend`。

innodb_autoextend_increment = n

带有 `autoextend` 属性的表空间文件每次加大多少兆字节（默认设置是 8MB）。这个属性不涉及具体的数据表文件，那些文件的增大速度相对是比较小的。

innodb_lock_wait_timeout = n

如果某个事务在等待 `n` 秒（s）后还没有获得所需要的资源，就使用 `ROLLBACK` 命令放弃这个事务。这项设置对于发现和处理那些未能被 InnoDB 数据表驱动程序识别出来的死锁条件有着重要的意义。这个选项的默认设置是 50s。

innodb_fast_shutdown = 0/1

是否以最快的速度关闭 InnoDB，默认设置是 1，意思是不把缓存在 `INSERT` 缓存区的数据写入数据表，那些数据将在 MySQL 服务器下次启动时再写入（这么做没有什么风险，因为 `INSERT` 缓存区是表空间的一个组成部分，数据不会丢失）。把这个选项设置为 0 反而比较危险，因为在计算机关机时，InnoDB 驱动程序很可能没有足够的时间完成它的数据同步工作，操作系统也许会在它完成数据同步工作之前强行结束 InnoDB，而这会导致数据不完整。

mysqld 程序——InnoDB——日志**innodb_log_group_home_dir = lp**

用来存放 InnoDB 日志文件的目录路径（如 `ib_logfile0`、`ib_logfile1` 等）。在默认的情况下，InnoDB 驱动程序将使用 MySQL 数据目录作为自己保存日志文件的位置。

innodb_log_files_in_group = n

使用多少个日志文件（默认设置是 2）。InnoDB 数据表驱动程序将以轮转方式依次填写这些文件；当所有的日志文件都写满以后，以后的日志信息将写入第一个日志文件并覆盖其中的原有数据；依此循环。

innodb_log_file_size = n

InnoDB 日志文件的最大长度（默认设置是 5MB）。这个长度必须以 MB（兆字节）或 GB（千兆字节）为单位进行设置。

innodb_flush_log_at_trx_commit = 0/1/2

这个选项决定着什么时候把日志信息写入日志文件以及什么时候把这些文件物理地写（术语称为“同步”）到硬盘上。设置值 0 的意思是每隔一秒写一次日志并进行同步，这可以减少硬盘写操作次数，但可能造成数据丢失；设置值 1（默认设置）的意思是在每执行完一条 `COMMIT` 命令就写一次日志并进行同步，这可以防止数据丢失，但硬盘写操作可能会很频繁；设置值 2 是一种折衷的办法，即每执行完一条 `COMMIT` 命令写一次日志，每隔一秒进行一次同步。

(续)

mysqld 程序——InnoDB——日志

<code>innodb_flush_method = x</code>	InnoDB 日志文件的同步办法（仅适用于 UNIX/Linux 系统）。这个选项的可取值有两种： <code>fdatasync</code> ，用 <code>fsync()</code> 函数进行同步； <code>O_DSYNC</code> ，用 <code>O_SYNC()</code> 函数进行同步
<code>innodb_log_archive = 1</code>	启用 InnoDB 驱动程序的 archive（档案）日志功能，把日志信息写入 <code>ib_arch_log_n</code> 文件。启用这种日志功能在 InnoDB 与 MySQL 一起使用时没有多大意义（启用 MySQL 服务器的二进制日志功能就足够用了）

mysqld 程序——InnoDB——缓存区的设置和优化

<code>innodb_log_buffer_pool_size = n</code>	为 InnoDB 数据表及其索引而保留的 RAM 内存量（默认设置是 8MB）。这个参数对速度有着相当大的影响，如果计算机上只运行有 MySQL/InnoDB 数据库服务器，就应该把全部内存的 80% 用于这个用途
<code>innodb_log_buffer_size = n</code>	事务日志文件写操作缓存区的最大长度（默认设置是 1MB）
<code>innodb_additional_mem_pool_size = n</code>	为用于内部管理的各种数据结构分配的缓存区最大长度（默认设置是 1MB）
<code>innodb_file_io_threads = n</code>	I/O 操作（硬盘读写操作）的最大线程个数（默认设置是 4）
<code>innodb_thread_concurrency = n</code>	InnoDB 驱动程序能够同时使用的最大线程个数（默认设置是 8）

22.3.4 其他选项

mysqld 程序——其他选项

<code>bind-address = ipaddr</code>	MySQL 服务器的 IP 地址。如果 MySQL 服务器所在的计算机有多个 IP 地址，这个选项将非常重要
<code>default-storage-engine = type</code>	新数据表的默认数据表类型（默认设置是 MyISAM）。这项设置还可以通过 <code>--default-table-type</code> 选项来设置
<code>default-timezone = name</code>	为 MySQL 服务器设置一个地理时区（如果它与本地计算机的地理时区不一样；详见本书第 14 章有关内容）
<code>ft_min_word_len = n</code>	全文索引的最小单词长度。这个选项的默认设置是 4，意思是在创建全文索引时不考虑那些由 3 个或更少的字符构成的单词
<code>Max_allowed_packet = n</code>	客户与服务器之间交换的数据包的最大长度，这个数字至少应该大于客户程序将要处理的最大 <code>BLOB</code> 块的长度。这个选项的默认设置是 1MB
<code>Sql-mode = mode1, mode2, ...</code>	MySQL 服务器将运行在哪一种 SQL 模式下。这个选项的作用是让 MySQL 与其他的数据库系统保持最大程度的兼容。这个选项的可取值包括 <code>ansi</code> 、 <code>db2</code> 、 <code>oracle</code> 、 <code>no_zero_date</code> 、 <code>pipes_as_concat</code> （详见本书第 14 章有关内容）

注意 如果在配置文件里给出的某个选项是 mysqld 程序无法识别的（比如说，因为犯了一个愚蠢的打字错误），MySQL 服务器将不会启动。千万要注意！

22.4 mysqld_safe 脚本（启动 MySQL 服务器）

UNIX/Linux 环境里的 MySQL 服务器通常是由一个 Init-V 脚本（比如 /etc/init.d/mysqld 或

/etc/init.d/mysql 脚本) 启动的, 而这个脚本又是在操作系统启动时自动执行的。不过, 这个脚本一般不会直接启动 MySQL 服务器, 它采取的办法是调用 mysqld_safe 脚本去完成这一任务。

最常见的做法是先以 *root* 用户的权限启动 mysqld_safe 脚本, 再让它使用一个专用的账户 (通常是权限相对较少的 mysql 账户) 去启动 MySQL 服务器。

在启动 MySQL 服务器之后, mysqld_safe 脚本将继续运行——除非数据库管理员明确地 (比如说, 使用 mysqladmin shutdown 命令) 结束了它的运行。如果 MySQL 服务器因为发生了崩溃而意外“死亡”, mysqld_safe 脚本会立刻重新启动它。

从理论上讲, MySQL 服务器程序 mysqld 的选项都可以与 mysqld_safe 脚本一起使用——后者将简单地把这些选项传递给前者。但在实际工作中, 因为 mysqld 程序的选项几乎总是已经在配置文件里设置妥当了的, 所以需要通过 mysqld_safe 脚本向 mysqld 程序传递选项的情况很少发生 (即使有发生, 往往也只是通过 --data-dir、--pid-file 等几个选项改用另一个数据目录或 PID 文件而已)。不过, 除了这些实际属于 mysqld 程序的选项 (请参见 22.3 节) 以外, mysqld_safe 脚本还有一些它独有的选项。

mysqld_safe 脚本——选项

--core-file-size = n	如果 MySQL 服务器发生崩溃, 这个选项将把内存镜像文件 (这个文件可以帮助人们查找和纠正错误) 的长度限制在 n 个字节以内。注意, n 必须以字节为单位给出
--ledir = dirname	MySQL 服务器程序的存放位置 (默认设置是 /usr/sbin)
--mysqld = filename	MySQL 服务器程序的名字 (默认设置是 mysqld)
--mysqld-version = suffix	MySQL 服务器程序的扩展名 (比如说, 如果把这个选项设置为 max, mysqld_safe 脚本就将去启动 mysqld-max 而不是 mysqld)
--nice = n	mysqld 进程的优先级 (man nice 命令可以提供更多信息)

22.5 mysql_install_db 脚本 (安装 mysql 数据库)

在 UNIX/Linux 系统上, mysql 数据库的首次安装和重新初始化工作可以用 mysql_install_db 脚本来完成。在使用这个脚本之前, 必须先停止 MySQL 服务器并删除现有的 mysql 数据库 (这个脚本会在完成对 mysql 数据库的初始化工作之后用 --bootstrap 选项重新启动 MySQL 服务器)。

```
root# /etc/init.d/mysql stop
root# rm -r /usr/lib/mysql/mysql
root# mysql_install_db
```

用来执行 mysql_install_db 脚本的账户必须与用来执行 MySQL 服务器的账户是同一个 (这个账户的名字通常是 mysql)。当然, 用 root 账户去执行这个脚本也是可以的, 但一定要在事后把名为 mysql 的数据库目录以及这个目录里的文件的属主改为 mysql 账户:

```
root# mysql_install_db
root# chown mysql -R /var/lib/mysql/mysql
```

这个脚本通常是作为 MySQL 安装工作的一个步骤自动执行的, 它也可以用来把 mysql 数据库恢复到初始化状态。此外, mysql_install_db 脚本不能在 Windows 系统上使用。

注意 根据具体的网络配置, 由这个脚本输入到 mysql.users 数据表里的本地主机名可能没有域名部分。使用简写的主机名有可能导致无法顺利登录 MySQL 服务器的问题。因此, 最好以手动方式把完整的主机名输入到 mysql.users 数据表里去 (参见第 11 章中的有关内容)。

22.6 mysql_fix_privileges 脚本（更新 mysql 数据库）

MySQL 的每一个新版本都或多或少地对负责管理 MySQL 访问权限的 *mysql* 数据库进行着扩充和改进。如果刚把 MySQL 服务器升级到了一个新版本并打算继续沿用来自老版本的 *mysql* 数据库（参见第 14 章），应该运行一次 *mysql_fix_privileges* 脚本来升级这个数据库（添加必要的新数据列等）。这个脚本只需要一个参数：MySQL 数据库系统的 *root* 用户（注意：不是操作系统的 *root* 用户）的密码。

在这个脚本运行结束后，还应该以手动方式做一些善后处理工作——这个脚本设置的某些新权限不一定符合实际情况。事实上，如果不确定应该如何为某位 MySQL 用户设置新的权限，这个脚本将不为这位用户设置任何新权限。这种做法当然比较稳妥和安全，但某些 MySQL 用户在 MySQL 老版本里拥有的权限却很可能因此而“缩水”。

这个脚本只能在 UNIX/Linux 环境里使用。在 Windows 环境里，可以在 MySQL 安装目录下的 scripts 目录里找到一个与这个脚本同名的*.sql 文件，这个文件可以用 mysql 程序来执行。

22.7 mysql_fix_extensions 脚本（重命名 MyISAM 文件）

在把一些包含 MyISAM 数据表的数据库目录从 Windows 系统移植到 UNIX/Linux 系统的时候，文件后缀名里的字母大小写往往会引起一些意想不到的问题。（UNIX/Linux 操作系统对文件名里的字母大小写情况很挑剔，而 Windows 系统则是只在极少数情况下才区分文件名里的大小写字母。）按照 MySQL 的语法要求，有些文件后缀名必须小写（例如*.frm），有些则必须大写（例如*.MYI 和*.MYD）。

这个问题可以用 Perl 脚本 *mysql_fix_extensions* 来解决。这个脚本只需要一个参数：MySQL 数据目录的路径名（比如：/var/lib/mysql 或 C:\mysql\data）。这个脚本没有任何选项。

22.8 mysql 程序（SQL 命令解释器）

mysql 程序是 MySQL 自带的命令解释器，它的基本用途是以交互方式执行 SQL 命令。这个程序也可以在批处理模式下运行，这种模式可以用来完成一些比较复杂的系统管理任务（比如把查询结果生成为 HTML 表格等）。在批处理模式下，*mysql* 程序将使用< file 语法从给定的批处理文件里读入并执行 SQL 命令。批处理文件允许包含任何一种 SQL 命令，唯一的要求是它们都必须后缀一个分号作为结束标记。如果需要在批处理文件里加上一些注释，在每行注释文本的开头加上一个“#”字符即可。（从“#”字符开始直到行尾的所有内容都将被认为是注释文本。）

在启动 *mysql* 程序的时候，有无数的选项可供选用。除了这些选项，还可以在用来启动 *mysql* 程序的命令行上给出一个数据库的名字，而这个数据库将成为本次 *mysql* 会话的默认数据库（相当于 *USE databasename* 命令）。

mysql 程序——语法

mysql [options] [databasename] [< commands.sql]

mysql 程序——基本选项

-e cmd	--execute = cmd	执行给定命令；命令必须用引号括起来，多条命令必须用分号隔开； <i>mysql</i> 程序将在执行完命令后退出
--------	-----------------	--

-i	--ignore-space	忽略函数名与其参数之间的空格。如果没有使用这个选项，就必须把函数调用写成 <i>SUM(price)</i> 的形式；使用了这个选项，还允许把函数调用写成 <i>SUM(price)</i> 的形式
----	----------------	---

(续)

mysql 程序——基本选项

-L	--skip-line-numbers	在显示出错信息时不显示行号。这里的行号是执行出错的命令在一个批处理文件里的语句行编号，知道这个行号对调试工作会很有帮助
-U	--i-am-a-dummy 或 --safe-updates	mysql 程序将拒绝执行不带 WHERE 或 LIMIT 子句的 UPDATE 和 DELETE 命令；这个选项还将给 SELECT 命令的查询结果以及同一条 SQL 命令里的 JOIN 操作次数设置一个上限
-V	--version	mysql 程序将显示自己的版本号；然后退出执行
	--tee = filename	把所有的输入和输出以追加方式复制到给定的日志文件里。这个选项仅适用于 mysql 程序的交互模式（不能在它的批处理模式下使用）
	--no-tee	不使用日志功能（这是默认设置）

mysql 程序——设置格式与输出选项

-B	--batch	以批处理方式运行 mysql 程序。查询结果中的数据列将以制表符分隔（而不是空格和线段），并且只显示查询结果，不显示状态信息
-E	--vertical	把查询结果中的数据列按纵向方式，后一个数据列在前一个数据列之下（不像平时那样后一个数据列在前一个数据列的右边）。这种显示方式仅适合显示那些数据列比较多、数据行却很少（最理想的情况是只有一个数据行）的查询结果。与 --batch 选项类似，这个选项也将只显示查询结果，不显示状态信息
-H	--html	在输出查询结果时把它们的格式设为 HTML 表格。与 --batch 选项类似，这个选项也将只显示查询结果，不显示状态信息
-N	--skip-column-names	在显示查询结果的时候不显示数据列标题
-r	--raw	在显示查询结果的时候不对零值字节、换行符和 “\” 进行转义，而是把它们按原样显示出来（平时的做法是把这几个字符显示为 “\0”、“\t” 和 “\\”）。这个选项必须与 --batch 选项同时使用才有效
-s	--silent	比正常模式少显示一些状态信息：不使用开销比较大的表格排版功能
-t	--table	用空格和线段对输出表格进行排版（这是默认设置）
-v	--verbose	比正常模式多显示一些状态信息
-X	--xml	在输出查询结果时把它们排版为 XML 表格

mysql 程序——常用命令（交互模式）

\c	clear	放弃正在输入的命令。在一条命令的末尾给出 \c 的效果是取消这一条命令，回车后将不是执行这一条命令而是开始输入下一条命令
\e	edit	调用一个外部编辑器来修改当前命令，外部编辑器由环境变量 EDITOR 指定。这个选项仅适用于 UNIX/Linux 系统。不过，从外部编辑器返回到 mysql 程序之后，刚才编辑的那条命令并不显示在 mysql 程序的命令行上，这种效果很容易让人感到困惑
\g	go	执行当前命令（效果相当于 “;” 加回车键）
\h	help	显示帮助信息（一份命令清单以及对它们的简单说明）
\p	print	把当前命令完整地显示在屏幕上
\q	exit 或 quit	退出 mysql 程序（在 UNIX/Linux 系统上，按下 Ctrl+D 组合键也有同样效果）
\r	connect	结束与 MySQL 服务器的当前连接并重新创建一个新连接。connect 命令有两个可选参数，如果需要给出，则前一个必须是数据库名，后一个必须是 MySQL 服务器的主机名

(续)

mysql 程序——常用命令（交互模式）		
\s	status	显示关于 MySQL 服务器的状态信息
\T[fn]	tee [filename]	把所有的输入和输出以追加方式复制到给定的日志文件里。如果没有给出日志文件名，则沿用上一条 tee 命令所使用的日志文件。如果日志文件已经存在，把后来的输入和输出追加到日志文件的末尾
\t	notee	结束 tee 命令，但用户随时可以使用 tee 或 \T 命令恢复使用日志功能
\u db	use database	把给定数据库选为当前的默认数据库
\#	rehash	把所有的 mysql 命令、最重要的 SQL 关键字以及当前数据库里的所有数据表和数据列的名字生成一份清单，mysql 程序将在其内部使用这份清单来实现“关键字自动补足功能”——用户只须输入某个关键字/名字的前几个字符再按下键盘上的 Tab 键，mysql 程序就会把它自动补足为一个完整的关键字。这个选项仅适用于 UNIX/Linux 系统
\. fn	sourec filename	从给定文件读出 SQL 命令并执行；文件里的各条命令之间必须用分号隔开

22.9 mysqladmin 程序（日常管理）

mysqladmin 程序能够在 MySQL 数据库上完成多种日常管理工作，如创建新数据库、修改密码等。mysqladmin 程序能够一次执行多条命令，这些命令将按照它们在 mysqladmin 命令行上的先后顺序依次执行。

mysqladmin 命令的名字允许以简写方式给出，只要不引起二义性就行。比如说，可以把 flush-logs 命令简写为 flush-l、把 kill 命令简写为 k 等。

大多数 mysqladmin 命令都有对应的 SQL 命令，如 CREATE DATABASE、DROP DATABASE、FLUSH、KILL、SHOW 等。在接下来的几个表格里，将在介绍各有关 mysqladmin 命令的时候把与之对应的 SQL 命令放在末尾的括号里。对 SQL 命令的描述和解释见第 21 章。

mysqladmin 程序——语法

mysqladmin [options] command1 command2 ...

mysqladmin 程序——选项

-f	--force	不显示出错消息（比如说，在执行 drop database 命令时）；即使发生错误也继续执行后面的命令
-i n	--sleep = n	每隔 n 秒重复执行一次命令（比如说，周期性地显示状态信息或 ping 命令的执行结果）。此时，mysqladmin 程序的执行将进入一个无限循环，结束这个循环的办法是：在 UNIX/Linux 系统上，按下 Ctrl+C 组合键；在 Windows 系统上，关闭这个命令窗口
-r	--relative	只显示发生了相对变化的状态信息。这个选项需要与 --sleep 选项和 extend-status 命令一起使用
-E	--vertical	这个选项与 --relative 选项的效果相同，只是输出内容将全部显示在同一行上（它可能会非常长）
-t n	--timeout = n	连接倒计时时间：如果经过 n 秒之后还没与 MySQL 服务器建立起连接，则退出 mysqladmin 程序
-w n	--wait = n	连接尝试次数：如果经过 n 次尝试之后还没与 MySQL 服务器建立起连接，则退出 mysqladmin 程序

mysqladmin 程序——常用命令

create dbname	创建一个新数据库（对应于 SQL 命令 <i>CREATE DATABASE</i> ）
drop dbname	删除一个现有的数据库（对应于 SQL 命令 <i>DROP DATABASE</i> ）
extended-status	查看 MySQL 服务器的所有状态变量（ <i>SHOW STATUS</i> ）
flush-hosts	清空主机缓存表（ <i>FLUSH HOSTS</i> ）
flush-logs	先关闭、再打开所有的日志文件（ <i>FLUSH LOGS</i> ）。对于变更日志，这个命令将创建一个新文件，作为文件后缀名的序号将增加 1（如 <i>file.003→file.004</i> ）
flush-status	对许多状态变量进行清零（ <i>FLUSH STATUS</i> ）
flush-tables	关闭所有已被打开的数据表（ <i>FLUSH TABLES</i> ）
flush-threads	清空线程缓存区
flush-privileges	重新加载 mysql 数据库（ <i>FLUSH PRIVILEGES</i> ）
kill id1, id2, ...	结束给定的线程（ <i>KILL</i> ）
password newpassw	修改当前用户的密码（ <i>SET PASSWORD</i> ）。这条命令在 Windows 系统上很容易引起问题，最好使用 SQL 命令 <i>SET PASSWORD</i> 来代替它（详见第 14 章）
ping	检查是否能与 MySQL 服务器建立连接
processlist	显示所有的进程（ <i>SHOW THREADS</i> ）
reload	重新加载 mysql 数据库（ <i>FLUSH PRIVILEGES</i> ）
refresh	先关闭、再打开所有的数据表文件和日志文件
shutdown	关闭 MySQL 服务器
start-slave / stop-slave	启动/停止一个从属镜像服务器的进程
status	查看 MySQL 服务器的一部分状态变量
variables	查看 MySQL 服务器的系统变量（ <i>SHOW VARIABLES</i> ）
version	查看 MySQL 服务器的版本信息

22.10 mysqldump 程序（数据的备份/导出）

mysqldump 程序将生成一份 SQL 命令清单，清单里的 SQL 命令可以精确地重新创建被备份的数据库。这个程序有 3 种语法变体，它们分别用于把 MySQL 服务器上的单个数据库、多个数据库或全部数据库备份下来。只有第一种变体可以用来备份某个特定的数据表。最后，别忘了把这个程序的输出用> backfile.sql 语句发送给一个文件。

mysqldump 程序——语法

```
mysqldump [options] dbname [tables]
mysqldump [options] --databases [moreoptions] dbname1 [dbname2 ...]
mysqldump [options] --all-databases [moreoptions]
```

mysqldump 程序——选项

-- --add-drop-table	在每条 <i>CREATE TABLE</i> 命令的前面加上一条 <i>DROP TABLE</i> 命令；这意味着备份恢复操作将先删除同名的现有数据表、然后再重新把它们创建出来。这个选项是--opt 选项的一部分，而且是默认设置
-- --add-locks	在各组 <i>INSERT</i> 命令的前后分别加上 <i>LOCK TABLE</i> 和 <i>UNLOCK TABLE</i> 命令；这可以加快备份恢复操作的速度。这个选项是--opt 选项的一部分，而且是默认设置。注意，这个选项不能对 InnoDB 数据表使用

(续)

mysqldump 程序——选项	
--	--all
-A	--all-databases
-B	--databases
-	--compatible = name
--	--complete-inserts
-	--create-options
-	--default-character-set = name
--	--delayed-insert
-K	--disable-keys
-e	--extended-insert
-F	--flush-logs
-f	--force
-	--hex-blob
-x	--lock-all-tables
-l	--lock-tables
--master-data [= n]	
--no-create-db	

(续)

mysqldump 程序——选项

--no-create-info	不在备份文件里创建 <i>CREATE TABLE</i> 命令, 只创建 <i>INSERT</i> 命令
--no-data	不在备份文件里创建 <i>INSERT</i> 命令(只创建 <i>CREATE TABLE</i> 命令, 目的是以后只恢复有关数据库的结构)
--opt	以下选项组合的简写形式: --add-drop-table、-create-options、--add-locks、--disable-keys、--extended-insert、--lock-tables、--quick 和--set-charset。这种组合在大多数场合里都是最佳设置, 所以也是默认设置。如果不希望这样, 就必须使用--skip-opt 选项来禁用这一组合
-q --quick	这个选项将使 mysqldump 程序在从 MySQL 服务器读出一个数据行之后立刻把它写入备份文件。这个选项是--opt 选项的一部分, 而且是默认设置。如果不使用这个选项, mysqldump 程序将先把整个数据表全部读入内存再写入备份文件。使用这个选项的好处是可以减少内存占用量, 坏处是 MySQL 服务器可能会被阻塞很长时间。在备份一个非常大的数据表(数据表的长度超过了本地计算机的内存容量)时, 一定要使用--quick 选项
-Q --quote-names	把数据表和数据列的名字用单引号括起来(例如: 'name')
--set-charset	在执行 mysqldump 程序之前改变字符集设置, 等这个程序执行完毕后再恢复原来的字符集设置。这个选项是--opt 选项的一部分, 而且是默认设置。--skip-char-set 选项将使这个选项失效
--skip-opt	不使用默认选项--opt
--single-transaction	用一个事务来读取所有的数据表。一般来说, 这个选项只在备份 InnoDB 数据表的时候才有必要使用, 它可以确保数据表里的数据在备份期间不会发生变化。--lock-tables 选项将使这个选项失效
-T dir --tab = dir	把备份结果直接写入给定目录。此时 mysqldump 程序将为每个数据表生成两个文件: 一个用来存放数据表的结构 (.sql); 另一个用来存放数据表所容纳的数据 (*.txt), 这个文件的内容是一系列 <i>SELECT ... INTO OUTFILE</i> 命令
-w cnd ----where = condition	只对数据表里符合 WHERE 条件 <i>cnd</i> 或 <i>condition</i> 的数据记录进行备份。这个选项必须整个地用引号括起来, 例如: "-wprices>5" 或 "--where=ID=3"
-X --xml	把数据表的内容备份为一个 XML 文件(没有数据表结构信息)

mysqladmin 程序——排版选项(只能与--tab 选项配合使用)

--fields-terminated-by	参见本书第 21 章: <i>SELECT ... INTO OUTFILE</i>
--fields-enclosed-by	
--fields-optionally-enclosed-by	
--fields-escaped-by	
--lines-terminated-by	

如果使用了--tab 选项, 第二个文件(tablename.txt)将直接包含着给定数据表的全部内容(而不是一系列 *INSERT* 命令)。这么做的优点: 备份的结果文件更加短小紧凑、备份恢复工作将完成得非常快。这么做的缺点: 操作过程比较复杂、每次只能备份或恢复一个数据表。

--fields-xxx 和--lines-xxx 选项都必须用引号括起来。下面这个例子演示了如何把双引号本身作为一个字符传递给有关选项:

```
> mysqldump -u root -p --tab /tmp "--fields-enclosed-by=\"\" ...
```

用 mysqladmin 程序生成的*.txt 文件进行备份恢复的办法有两种：一是使用 mysqlimport 程序（参见 22.11 节）；二是使用 SQL 命令 *LOAD DATA*。

22.11 mysqlimport 程序（文本导入、批量导入）

mysqlimport 程序可以把专用格式的文本文件导入 MySQL 数据表。此时，mysqlimport 程序相当于 SQL 命令 *LOAD DATA* 的一个命令行接口；对这条 SQL 命令的详细介绍见第 14 章。

mysqlimport 程序——语法

```
mysqlimport [options] databasefilename
```

mysqlimport 程序——选项

-d	--delete	在导入操作正式开始之前先把数据表里的现有记录全部删除
-i	--ignore	如果来自备份文件的某条记录会导致数据表里的 <i>UNIQUE</i> 或 <i>PRIMARY KEY</i> 数据列出现重复的键值，则保留现有的记录而丢弃来自备份文件的记录
-L	--local	读取来自本地文件系统（而不是来自 MySQL 服务器）的备份文件
-l	--lock-tables	在导入期间锁定数据表，不让其他客户访问
-r	--replace	如果来自备份文件的某条记录会导致数据表里的 <i>UNIQUE</i> 或 <i>PRIMARY KEY</i> 数据列出现重复的键值，用来自数据文件的记录覆盖现有的记录
	--fields-terminated-by	如何处理特殊字符：参见本书第 21 章： <i>LOAD DATA</i>
	--fields-enclosed-by	
	--fields-optionally-enclosed-by	
	--fields-escaped-by	
	--lines-terminated-by	

22.12 mysqlshow 程序（查看信息）

mysqlshow 程序可以帮助用户快速查明 MySQL 服务器上的数据库、数据表和数据列的基本情况。不带参数的 mysqlshow 命令将返回一份全体 MySQL 数据库的清单，带参数的 mysqlshow 命令将返回给定数据库、数据表或数据列的信息。

mysqlshow 程序——语法

```
mysqlshow [options] [databasefilename [tablename [columnname]]]
```

mysqlimport 程序——选项

-i	--status	显示关于数据表的更多信息（数据表类型、数据记录的平均长度等）
----	----------	--------------------------------

mysqlshow 程序用法示例

```
> mysqlshow -u root -p
Enter password: xxx
Databases:
  books
```

```

myforum
mylibrary
mysql
test
...
> mysqlshow -u root -p mysql
Enter password: xxx
Database: mysql
Tables:
  columns_priv
  db
  host
...
> mysqlshow -u root -p mylibrary authors
Enter password: xxx
Database: mylibrary  Table: authors  Rows: 0
Field      Type           Collation          Null  Key  Default Extra Privileges
authID     int(11)        latin1_german1_ci    PRI   ...   ...
authName   varchar(60)    latin1_german1_ci    MUL   ...   ...
ts         timestamp      YES    ...   ...

```

22.13 myisamchk 程序（修复 MyISAM 文件）

myisamchk 程序能够检查 MyISAM 数据库文件 (*.MYD 和 *.MYI 文件) 是否受损，并对受损的文件和索引进行必要的修复。

myisamchk 命令的参数是数据表的名字，这里只要求给出有关文件的主名字（即带或不带.MYI 后缀名均可），但根据用户具体使用的选项，这个程序可能会对两种 MyISAM 文件 (*.MYD 和 *.MYI 文件) 都进行分析和修复。

myisamchk 程序有两种工作模式：一种是只对数据表文件进行检查（用户没有给出 -r、-o、--recover、--safererecover 等选项）；另一种是对数据表文件进行检查并进行必要的修复（用户给出了 -r、-o、--recover、--safererecover 等选项之一）。请注意，有几个选项在这两种工作模式下都可以使用，但含义不同。

myisamchk 程序——语法

`myisamchk [options] tablename1 tablename2 ...`

myisamchk 程序——选项（只分析数据表文件：用户没有给出 -r 或 -o 选项）

-C	--check	检查数据表文件是否受损；这是 myisamchk 程序在用户没有给出任何选项时的默认动作
-e	--extend-check	对数据表进行最全面的检查（因而速度也最慢）
-F	--fast	只对没有正常关闭的数据表文件进行检查
-C	--check-only-changed	只对上次检查后又发生过修改的数据表进行检查
-f	--force	如果发现错误，myisamchk 程序将自动使用 -r 选项重新启动自己
-i	--information	显示关于数据表的统计信息
-m	--medium-check	对数据表进行比 -C 选项更为全面的检查（因而速度也比较慢）
-U	--update-state	在发现错误时把数据表文件标记为“受损”
-T	--read-only	只进行检查，不对数据表文件进行修改

myisamchk 程序——选项（修复并修改数据表文件：用户给出了-r 或-o 选项）

-B	--backup	为 name.MYI 文件制作一个备份，备份文件的名字是 name.BAK
-e	--extend-check	尝试重新创建每一条数据记录，但这经常导致许多记录出现错误数据或数据被误删，而且这种修复需要花费的时间也相当长。尽量避免使用这个选项
-f	--force	强行覆盖临时文件
-l	--no-symlinks	只对使用给定文件名找到的实际文件进行修复，不追踪符号链接（仅适用于 UNIX/Linux 系统）
-o	--safe-recover	类似于-r 选项，但使用的是另一种算法
-q	--quick	只修复索引文件；不对数据文件进行修复
-q -q		与-q 选项的效果基本一样，但会对数据文件做有限度的修复：如果在索引文件里发现重复的键字（索引值），则对数据文件进行必要的修复；不处理其他错误
-r	--recover	尝试重新创建受损的文件
-t p	--tmpdir = path	使用给定的目录来存放临时文件
-u	--unpack	对经过 myisampack 程序压缩的数据表文件进行解压缩

myisamchk 程序——其他选项

-a	--analyze	对索引中的键值字段分布情况进行分析。这可以加快对数据表的访问速度
-A n	--set-auto-increment [=n]	对 AUTO_INCREMENT 计数器进行设置，此后的 ID 编号值将从 n 开始。如果数据表里已经存在 ID=n 的记录或者是没有给出 n 值，这个选项将把下一个 AUTO_INCREMENT 值设置为“当前最大 ID 值+1”
-d	--description	显示关于数据表的各种信息（数据记录的格式和长度、字符集、索引等）
-R	--sort-records = idxnr	根据给定索引对数据表文件里的数据记录进行排序，索引以数字编号的形式给出（这些编号可以通过-d 选项查知）。这将使那些在索引里相邻的记录在数据表文件里也相邻。如果经常按照数据记录在某个索引里的先后顺序依次对它们进行读取，把这个索引作为-R 选项的参数对数据表进行上述优化后可以加快访问速度
-S	--sort-index	对索引文件里的数据项进行排序

myisamchk 程序——与内存管理有关的变量

-0 key_buffer_size = n	关键字缓存区的长度（默认设置是 512KB）
-0 read_buffer_size = n	读缓存区的长度（默认设置是 256）
-0 sort_buffer_size = n	排序缓存区的长度（默认设置是 2MB）
-0 write_buffer_size = n	写缓存区的长度（默认设置是 256KB）

22.14 myisampack 程序（压缩 MyISAM 文件）

MyISAM 数据库文件 (*.MYD) 在经过 myisampack 程序的压缩后可以大大减少空间占用量（往往可以减少一半以上），在某些特定的情况下还可以提高对数据的访问速度。如果说有什么不好的话，那就是数据在经过压缩后就不能再修改了。

myisampack 程序——语法

```
myisampack [options] tablename1 tablename2 ...
```

myisampack 程序——选项

-b	--backup	为数据表文件 name.MYD 制作一个备份，备份文件的名字是 name.OLD
-f	--force	即使压缩结果文件的长度已经超过了原始文件，也继续进行压缩
-J	--join = 'new_table_name'	把在命令行上给出的所有数据表合并为一个大文件，参加这种合并操作的数据表必须具有同样的数据列定义
-t	--test	以模拟方式运行 myisampack 程序，用户可以看到整个压缩过程中的各种信息，但数据本身并没有发生任何变化
-T p	--tmpdir = path	使用给定的目录来存放临时文件

第 23 章

MySQL API 应用指南

在前面的有关章节里，分别对如何使用 PHP、Perl、Java 和 C 等几种程序设计语言编写 MySQL 应用程序的问题进行了探讨。在这一章里，将对这几种语言的 MySQL API (application programming interface，应用编程接口) 进行总结。

23.1 PHP API (*mysql* 接口)

从 PHP 5 开始，PHP 提供了两种 MySQL 编程接口：一种是由一系列函数构成的 *mysql* 接口；另一种是由一系列的类和方法构成的 *mysqli* 接口。本节将对 *mysql* 接口中用来访问 MySQL 数据库的各种函数以及它们的参数进行描述。对 *mysqli* 接口的总结见 23.2 节。

下面是我们在接下来的讨论中会用到的一些语法元素及其含义：

- 在表格的左栏里，方括号里的语法元素代表着各种可选参数。
- 对于需要用到数组参数的所有函数，我们将用字母 *n* 来代表那个数组的下标，数组下标的取值范围是 0 到 *nmax-1*，*nmax* 是数组元素的总数。
- 下面将要介绍的各有关函数的应用示例集中出现在本书的第 3 章和第 15 章。

与 MySQL 服务器建立连接

<code>\$id = mysql_connect(\$host, \$user, \$pw);</code>	建立一条连接
<code>\$id = mysql_connect(\$host, \$user, \$pw, \$new_link, \$client_flags);</code>	功能同上，只是多了两个可选的参数： <code>\$new_link</code> 参数控制着在已经存在一条类似连接的情况下是否还要创建一条新连接（默认设置是 <code>false</code> ：不创建新连接）； <code>\$client_flag</code> 参数负责给出其他一些必要的连接属性（比如： <code>MYSQL_CLIENT_COMPRESS</code> 等）。这两个可选参数始于 PHP 4.3 版本
<code>\$id = mysql_pconnect(\$host, \$user, \$pw [, \$new_link [, \$client_flags]]);</code>	建立一条永久性连接或尝试使用由另一个 PHP 页面打开并仍处于打开状态的连接
<code>mysql_change_user(\$newuser, \$passwd);</code>	改变当前连接所使用的用户名
<code>mysql_select_db(\$dbname);</code>	确定当前连接所使用的默认数据库
<code>mysql_close([\$id]);</code>	关闭由 <code>\$id</code> 参数给定的连接

如果只与 MySQL 服务器建立了一条连接，`$id` 参数通常可以省略；这里的原则是必须保证省略 `$id` 参数的做法不会引起任何二义性问题。

管理性操作

<code>\$result = mysql_list_dbs([\$id]);</code>	给定一条连接，返回一份现有数据库的清单；可以像对待 SELECT 查询结果那样对 \$result 做进一步处理
<code>\$result = mysql_list_tables(\$dbname [, \$id]);</code>	给定一个数据库，返回一份现有数据表的清单；可以像对待 SELECT 查询结果那样对 \$result 做进一步处理
<code>\$result = mysql_list_fields(\$dbn, \$tbln [, \$id]);</code>	给定一个数据库和一个数据表，返回一份现有数据列的清单；可以像对待 SELECT 查询结果那样对 \$result 做进一步处理
<code>mysql_create_db(\$dbname [, \$id]);</code>	创建一个新的数据库
<code>mysql_drop_db(\$dbname [, \$id]);</code>	删除一个数据库

出错处理

<code>\$n = mysql_errno([\$id]);</code>	最近发生的错误的出错代码
<code>\$txt = mysql_error([\$id]);</code>	最近发生的错误的出错消息

与 MySQL 服务器和本次连接有关的信息

<code>\$txt = mysql_get_client_info([\$id]);</code>	返回一个字符串：客户端 API 函数库的版本信息
<code>\$txt = mysql_get_host_info([\$id]);</code>	返回一个字符串：对 MySQL 服务器主机的描述（包括主机名，比如：“localhost via TCP/IP”）
<code>\$n = mysql_get_proto_info([\$id]);</code>	返回一个整数：通信双方使用的通信协议的编号（如 10）
<code>\$txt = mysql_get_server_info([\$id]);</code>	返回一个字符串：MySQL 服务器的版本信息（如“5.0.2-alpha-standard”）
<code>\$n = mysql_thread_id([\$id]);</code>	返回一个整数：给定连接的线程编号
<code>\$txt = mysql_stat([\$id]);</code>	返回一个字符串：MySQL 服务器的工作状态（如“Uptime: 24763 Threads: 1 Questions: 65 ...”）

执行 SQL 命令

<code>[\$result =] mysql_query(\$sql [, \$id])</code>	对默认数据库执行一条 SQL 命令；如果它是一条 SELECT 命令，\$result 将包含它查找出的全部结果记录
<code>[\$result =] mysql_db_query (\$db, \$sql [, \$id]);</code>	对给定数据库执行一条 SQL 命令；这个数据库将成为后续 SQL 命令的默认数据库
<code>\$result = mysql_unbuffered _query(\$sql [, \$id]);</code>	与 mysql_query() 函数的功能基本相同，但这个函数只能用来执行 SELECT 查询命令。这个函数与 mysql_query() 函数的区别是：这个函数查找出的结果记录将保存在服务器上，只在必要时才传输到客户端；这个函数返回的结果记录的总数只有在遍历它们之后才能确定，无法提前使用 mysql_num_rows() 函数查知
<code>\$sql = addslashes(\$s);</code>	把字符串 \$s 里的零值字节、‘\’、‘”’ 和 ‘\’ 字符分别替换为字符串 ‘\0’、‘\’、‘”’ 和 ‘\\’
<code>\$sql = mysql_escape_string(\$s);</code>	与 addslashes() 函数的功能基本相同，但还会把回车符、换行符和 Ctrl+Z 分别替换为字符串 ‘\n’、‘\r’ 和 ‘\z’
<code>\$sql = mysql_real_escape_string(\$s [, \$id]);</code>	与 mysql_escape_string() 函数的功能基本相同，但还会把给定 MySQL 连接所使用的字符集考虑在内

对 SELECT 查询结果进行处理

<code>\$mysql_data_seek(\$result, \$rownr);</code>	在查询结果里，把给定编号的结果记录选定为当前数据记录。这在效果上相当于直接跳转到给定的结果记录
<code>\$row = mysql_fetch_array(\$result);</code>	返回查询结果里的下一条记录（或 <code>false</code> ）；对各个字段的访问需要通过 <code>row[n]</code> 或 <code>row["fieldname"]</code> 的形式进行，后一种形式还区分字段名的字母大小写情况
<code>\$row = mysql_fetch_assoc(\$result);</code>	与 <code>mysql_fetch_array()</code> 函数的功能基本相同，但对各个字段的访问只能通过 <code>row["fieldname"]</code> 的形式来进行，不允许通过 <code>row[n]</code> 的形式去访问各个字段
<code>\$row = mysql_fetch_row(\$result);</code>	返回查询结果里的下一条记录（或 <code>false</code> ）；对各个字段的访问需要通过 <code>row[n]</code> 的形式来进行
<code>\$row = mysql_fetch_object(\$result);</code>	返回查询结果里的下一条记录（或 <code>false</code> ）；对各个字段的访问需要通过 <code>row->fieldname</code> 的形式来进行
<code>\$data = mysql_result(\$result, \$rownr, \$colnr);</code>	返回第 <code>rownr</code> 行、第 <code>colnr</code> 列处的字段值；这个函数比本表格里其他函数的速度都慢，所以只应该用在特定的场合（比如在只需要读取一个离散值的时候，例如： <code>SELECT COUNT(*)</code> ）
<code>mysql_free_result(\$result);</code>	立刻释放 <code>\$result</code> 参数所代表的查询结果（否则，查询结果将一直驻留在内存里直到这个脚本运行结束）

除 `mysql_free_result()`¹ 函数是把 `$result` 参数当做一个标识符来使用以外，上面这个表格里的所有函数都把 `$result` 当做一个数组来对待，并通过它去访问 `SELECT` 查询结果里的结果记录。总的来说，`mysql_fetch_array()`、`mysql_fetch_row()` 或 `mysql_fetch_object()` 函数都可以用来依次遍历所有的结果记录，每调用一次这几个函数当中的任何一个都可以返回下一条结果记录。如有必要，还可以利用 `mysql_data_seek()` 函数快速跳转到某条特定的结果记录去进行处理。

`mysql_fetch_array()`、`mysql_fetch_row()` 和 `mysql_fetch_object()` 函数的区别仅在于访问记录的单个字段的方式上，是用 `row["fieldname"]`、`row[n]` 还是用 `row->fieldname`。`mysql_fetch_row()` 是这 3 个函数中效率最高的，但它们之间执行速度的差别几乎可以忽略不计。

关于查询结果的元信息

<code>\$n = mysql_num_rows(\$result);</code>	结果记录（数据行）的总数（ <code>SELECT</code> ）
<code>\$n = mysql_num_fields(\$result);</code>	结果字段（数据列）的总数（ <code>SELECT</code> ）
<code>\$n = mysql_affected_rows([\$id]);</code>	被最近一条 SQL 命令改变的数据记录的总数（ <code>INSERT</code> 、 <code>UPDATE</code> 、 <code>DELETE</code> 、 <code>CREATE</code> 、 <code>SELECT</code> 等）
<code>\$auto_id = mysql_insert_id([\$id]);</code>	MySQL 服务器为最近一条 <code>INSERT</code> 命令自动生成的 <code>AUTO_INCREMENT</code> 编号值
<code>\$txt = mysql_info([\$id]);</code>	最近一条 SQL 的状态信息，比如 <code>Rows matched: 65 Changed: 65 Warnings: 0</code> 。 <code>mysql_info()</code> 函数是为了让人们了解那些会影响到大量数据记录的 SQL 命令的执行情况而准备的（ <code>INSERT INTO</code> 、 <code>UPDATE</code> 、 <code>ALTER TABLE</code> 等）

关于查询结果中的各个字段（数据列）的元信息

<code>\$fname = mysql_field_name(\$result, \$n);</code>	第 <code>n</code> 个数据列的字段名
<code>\$tblname = mysql_field_table(\$result, \$n);</code>	提供第 <code>n</code> 个数据列的数据表的名字

1. 原文 “`mysql_treat_result()`”，显然是笔误或印刷错误。——译者注

(续)

关于查询结果中的各个字段（数据列）的元信息

<code>\$typename = mysql_field_type(\$result, \$n);</code>	第 <i>n</i> 个数据列的数据类型（例如：“TINYINT”）
<code>\$length = mysql_field_len(\$result, \$n);</code>	第 <i>n</i> 个数据列的最大长度
<code>\$lengths = mysql_fetch_lengths(\$result);</code>	返回一个数组，它的各个元素分别对应着最近一次读取的那条结果记录的各个字段的长度信息（这些信息可以通过 <code>\$lengths[n]</code> 的形式去访问）
<code>\$flags = mysql_field_flags(\$result, \$n);</code>	返回一个字符串，第 <i>n</i> 个数据列的 MySQL 数据类型和属性信息（例如：“not_null primary_key”）。各项属性信息以空格隔开，最适合用 <code>explode()</code> 函数来提取
<code>\$info = mysql_fetch_field(\$result, \$n);</code>	把关于第 <i>n</i> 个数据列的信息返回为一个对象：各项信息需要通过 <code>info->name</code> 的形式去访问（ <code>name</code> 是属性或数据类型的名字；详见下面的表格）。请注意， <code>\$info</code> 包含的信息与 <code>\$flag</code> 包含的信息有些一样，有些不一样

mysql_field_flags()函数返回的字段（数据列）元信息

<code>auto_increment</code>	对应于 MySQL 中的数据列属性 <code>AUTO_INCREMENT</code>
<code>binary</code>	数据列属性 <code>BINARY</code>
<code>blob</code>	数据类型 <code>BLOB</code> 、 <code>TINYBLOB</code> 等
<code>enum</code>	数据类型 <code>ENUM</code>
<code>multiple_key</code>	这个字段是一个多字段索引的组成部分之一
<code>not_null</code>	数据列属性 <code>NOT NULL</code>
<code>primary_key</code>	数据列属性 <code>PRIMARY KEY</code>
<code>timestamp</code>	数据类型 <code>TIMESTAMP</code> ¹
<code>unique_key</code>	数据列属性 <code>UNIQUE</code>
<code>unsigned</code>	数据列属性 <code>UNSIGNED</code>
<code>zerofill</code>	数据列属性 <code>ZEROFILL</code>

mysql_fetch_field()函数返回的字段（数据列）元信息

<code>info->name</code>	数据列的名字（字段名）
<code>info->table</code>	提供这个字段的数据表的名字
<code>info->max_length</code>	字段的最大长度
<code>info->type</code>	字段的数据类型的名字（例如：“TINYINT”）
<code>info->numeric</code>	1（是）或 0（否）；存放在这个字段里的数据是不是数值
<code>info->blob, not_null,</code> <code>multiple_key, primary_key,</code> <code>unique_key, unsigned, zerofill</code>	1（是）或 0（否）；参见上一个表格里的说明

23.2 PHP API (*mysqli* 接口)

从 PHP 5 开始，除了在 23.1 节介绍的 *mysql* 接口外，PHP 还提供了一个新的 *mysqli* 接口。这个新接口

1. 原书说“`TIMESTAMP`”是属性是不对的。——译者注

的优点主要有：面向对象编程、更丰富的功能、支持 MySQL 新功能（比如预处理语句等）。

mysqli 接口提供了 3 个类。

- *mysqli* 类：这个类的对象负责建立与 MySQL 服务器的连接和执行 SQL 命令。
- *mysqli_result* 类：这个类的对象包含 *SELECT* 查询命令的执行结果。
- *mysqli_stmt* 类：这个类的对象负责定义和执行各种预处理语句。

在接下来的几小节里，将把这 3 个类最重要的属性和方法分别汇总在几个语法表格里。在这些表格中，*\$mysqli* 代表一个 *mysqli* 类的对象，*\$result* 代表一个 *mysqli_result* 类的对象，*\$stmt* 代表一个 *mysqli_stmt* 类的对象。

23.2.1 *mysqli* 类

与 MySQL 服务器建立连接

<i>\$mysqli = new mysqli("servername", "user", "pw", "dbname");</i>	变体 1：用 <i>mysqli</i> 构造器创建连接
<i>\$mysqli->options(...);</i> <i>\$mysqli->ssl_set("key", "cert", "ca", "capath", "cipher");</i> <i>\$mysqli->real_connect("servername", "user", "pw", "dbname", portno, "socketfile", flags);</i>	变体 2：先用 <i>mysqli_init()</i> 方法创建一个 <i>mysqli</i> 对象，再通过 <i>\$mysqli->options(MYSQL_OPT_CONNECT_TIMEOUT, 10)</i> 的方式设置好连接选项，最后用 <i>real_connect()</i> 方法建立起连接。 <i>real_connect()</i> 方法的最后 3 个参数是可选的： <i>flags</i> 变量的可取值包括 <i>MYSQL_CLIENT_COMPRESS</i> 和 <i>MYSQL_CLIENT_SSL</i>
<i>\$err = mysqli_connect_errno();</i>	测试在创建连接的过程中是否发生过错误。返回值 0 意味着连接成功

mysqli 类——执行 SQL 命令

<i>[\$result =] \$mysqli->query(\$sqlstring);</i>	执行一条 SQL 命令。如果它是一条 <i>SELECT</i> 命令，查询结果将被传输到客户端并返回为一个 <i>mysqli_result</i> 对象
<i>\$mysqli->real_query(\$sqlstring);</i>	执行一条 SQL 命令，但不传输查询结果
<i>\$sql = "sqlcmd1;cmd2;cmd3";</i> <i>\$ok = \$mysqli->multi_query(\$sql);</i> <i>if(\$ok)</i> <i>do {</i> <i>\$result = \$mysqli->store_result();</i> <i>... process result</i> <i>while(\$mysqli->next_result());</i>	一次执行多条 SQL 命令。第一组 <i>SELECT</i> 查询结果可以立刻通过 <i>store_result()</i> 方法去访问。如果有一组以上的查询结果，就必须使用 <i>next_result()</i> 方法去激活。如果已经到达最后一组查询结果或发生错误， <i>next_result()</i> 方法将返回 0
<i>\$stmt = \$mysqli->prepare(\$sqlstring);</i>	对一条带参数的 SQL 命令进行预处理并返回一个 <i>mysqli_stmt</i> 对象，SQL 命令的执行以及对查询结果的处理将通过 <i>mysqli_stmt</i> 类提供的各种方法来进行（见下面的表格）

mysqli 类——重要的方法

<i>\$mysqli->autocommit(0 / 1);</i>	设置 <i>autocommit</i> 模式
<i>\$mysqli->close();</i>	关闭与 MySQL 服务器的连接
<i>\$mysqli->commit();</i>	结束一个事务
<i>\$str = \$mysqli->escape_string(\$str);</i>	对字符串里的特殊字符进行转义处理：给它们加上一个反斜线 (\) 作为前缀或者是把它们替换为符合 SQL 语法要求的字符组合
<i>\$mysqli->rollback();</i>	放弃一个事务

***mysqli* 类——重要的属性**

<code>\$n = \$mysqli->affected_rows;</code>	被最近一次执行的 SQL 命令 (INSERT、UPDATE、DELETE 等) 改变了的数据记录总数
<code>\$n = \$mysqli->errno;</code>	最近一次发生的错误的出错代码
<code>\$str = \$mysqli->error;</code>	最近一次发生的错误的出错消息
<code>\$str = \$mysqli->info;</code>	返回一个字符串, 其内容是最进一次执行的 SQL 命令的状态信息 (比如说, 在执行完一条 UPDATE 命令后, 这个字符串将是 <i>Rows matched: nnn Changed: nnn Warnings: nnn</i> 的样子)
<code>\$n = \$mysqli->insert_id;</code>	MySQL 服务器为最近一次执行的 INSERT 命令自动生成的 AUTO_INCREMENT 编号值
<code>\$n = \$mysqli->warning_count;</code>	最近一次执行的 SQL 命令触发的出错警告总次数

23.2.2 *mysqli_result* 类

***mysqli_result* 类——重要的方法**

<code>\$result->close();</code>	释放这个对象占用的内存
<code>\$result->data_seek(n);</code>	把第 <i>n</i> 条结果记录选定为当前结果记录 (<i>n=0</i> 对应着第一条结果记录)。注意, 如果在调用 <i>query()</i> 方法时使用了可选参数 <i>MYSQL_USE_RESULT</i> , 就不能使用这个办法来快速定位
<code>\$row = \$result->fetch_array();</code>	返回查询结果中的下一条记录 (或 FALSE)。对各个字段的访问需要通过 <code>\$row[n]</code> 或 <code>\$row['fieldname']</code> 的形式进行, 后一种形式还区分字段名的字母大小写情况
<code>\$row = \$result->fetch_assoc();</code>	与 <i>fetch_array()</i> 方法的功能基本相同, 但对各个字段的访问只能通过 <code>\$row['fieldname']</code> 的形式来进行, 不允许通过 <code>\$row[n]</code> 的形式去访问各个字段
<code>\$meta = \$result->fetch_fields();</code>	返回一个对象数组, 这个数组的各个元素包含着各个结果字段 (数据列) 的元信息。比如说, <code>\$meta[\$n]->name</code> 将包含着某给定字段的名字
<code>\$row = \$result->fetch_row();</code>	与 <i>fetch_array()</i> 方法的功能基本相同, 但对各个字段的访问只能通过 <code>\$row[n]</code> 的形式来进行
<code>\$row = \$result->fetch_object();</code>	返回查询结果里的下一条记录 (或 FALSE); 对各个字段的访问需要通过 <code>\$row->fieldname</code> 的形式来进行, 并区分字段名的字母大小写情况

***mysqli_result* 类——重要的属性**

<code>\$n = \$result->affected_rows;</code>	被最近一次执行的 SQL 命令 (INSERT、UPDATE、DELETE 等) 改变了的数据记录总数
<code>\$n = \$result->field_count;</code>	<i>SELECT</i> 查询结果中的字段 (数据列) 总数
<code>\$lenarray = \$result->lengths;</code>	返回一个整数数组, 这个数组的各个元素包含着用 <i>fetch_xxx()</i> 方法最近一次读取的结果记录的各个字段值的字符长度 (字符个数)
<code>\$n = \$result->num_rows;</code>	<i>SELECT</i> 查询结果中的记录 (数据行) 总数

23.2.3 mysqli_stmt 类

mysqli_stmt 类——重要的方法

<code>\$stmt->bind_param('idsb... \$var1, \$var2 ...);</code>	把 SQL 命令的参数与一些 PHP 变量绑定起来，这些 PHP 变量的数据类型必须用一个字符来做出声明： <i>i</i> =integer（整数）、 <i>d</i> =double（浮点数）、 <i>s</i> =string（字符串）、 <i>b</i> =binary（二进制字节串，即 <i>BLOB</i> ）
<code>\$stmt->bind_result(\$var1, \$var2 ...);</code>	把 <i>SELECT</i> 查询结果中的字段（数据列）与一些 PHP 变量绑定起来。这个方法只有在 <i>execute()</i> 方法成功返回后才能调用
<code>\$stmt->close();</code>	释放这个对象占用的内存
<code>\$stmt->execute();</code>	执行一条 SQL 命令，它的参数通过 <i>\$var1</i> 、 <i>\$var2</i> 等变量来传递
<code>\$stmt->fetch();</code>	把 <i>SELECT</i> 查询结果中的下一条记录提取到刚才用 <i>bind_result()</i> 方法绑定的 PHP 变量里去。如果已经到达最后一条结果记录， <i>fetch()</i> 方法将返回 <i>FALSE</i> 。 <i>fetch()</i> 方法的默认行为是每次从服务器取回一条结果记录到客户端，但如果此前调用过 <i>store_result()</i> 方法，它将从客户端读取结果记录（参见下一条目）
<code>\$stmt->store_result();</code>	把所有的 <i>SELECT</i> 查询结果一次性地传输到客户端

mysqli_stmt 类——重要的属性

<code>\$n = \$stmt->affected_rows;</code>	被最近一次执行的 SQL 命令（ <i>INSERT</i> 、 <i>UPDATE</i> 、 <i>DELETE</i> 等）改变了的数据记录总数
<code>\$n = \$stmt->num_rows;</code>	<i>SELECT</i> 查询结果中的记录（数据行）总数。注意，这个属性只能与 <i>store_result()</i> 方法配合使用，不能与 <i>use_result()</i> 方法配合使用

23.3 Perl DBI

这份指南未能收录全部的 *DBI* 方法、函数和属性。这里只对使用 Perl 语言开发 MySQL 应用程序时最常用的语法关键字进行了介绍。完整的 Perl *DBI* 应用指南可以在 *perldoc* 文档里查到。

为了提高可读性，在接下来的内容里给各种 Perl 方法都加上了括号。但这里要提醒一句：Perl 语法允许程序员在调用各种方法的时候不写出括号，如 `$dbh->disconnect`。

下面是用 Perl 语言编写出来的 MySQL 应用脚本文件的基本框架：

```
#!/usr/bin/perl -w
use DBI;          # database access
use CGI qw(:standard); # required only with CGI scripts
use CGI::Carp qw(fatalsToBrowser); # only with CGI scripts
...
      # here follows the actual code
```

23.3.1 常用的变量名

Perl DBI 模块是面向对象的，所以本节将要介绍的函数都以某种对象的方法（Perl 语言称为“句柄”）的面目出现。在接下来的内容里，将使用以下变量来表示各种常用的 *Perl DBI* 对象：

DBI 句柄的常用变量名

<code>\$dbh</code>	<i>(database handle)</i>	数据库句柄。代表着一条与数据库的连接
<code>\$sth</code>	<i>(statement handle)</i>	语句句柄。用来对查询结果（ <i>SELECT</i> 查询命令）进行处理
<code>\$h</code>	<i>(handle)</i>	一个通用的句柄。它可以代表 <code>\$dbh</code> 、 <code>\$sth</code> 和 <i>DBI</i> 等句柄中的任何一个
<code>\$drh</code>	<i>(driver handle)</i>	驱动模块句柄。用来完成各种管理性功能

23.3.2 与 MySQL 服务器建立连接

连接

<code>use DBI();</code>	导入 DBI 模块
<code>\$datasource = "DBI:mysql:dbname;" .</code>	给出数据库名和主机名；数据库名可以省略，但它前面的冒号不能省略
<code>"host=hostname";</code>	
<code>\$dbh = DBI->connect (\$datasource,</code>	与给定数据库建立连接
<code>\$username, \$password</code>	
<code>[, %attributes]);</code>	

数据源字符串 `$datasource` 还可以包含其他一些参数，参数之间必须用分号隔开；对这些参数的详细介绍见本书第 22 章。

数据源字符串 `$datasource` 里的可选参数

<code>host=hostname</code>	MySQL 服务器的主机名（默认设置是 <code>localhost</code> ）
<code>port=n</code>	MySQL 客户与服务器进行通信时使用的 IP 端口（默认设置是 3306）
<code>mysql_compression=0/1</code>	采用压缩格式通信（默认设置是 0，不采用压缩格式）
<code>mysql_read_default_file=filename</code>	本次连接将要使用的 MySQL 配置文件的文件名
<code>mysql_read_default_group=mygroup</code>	从配置文件里读入 [mygroup] 选项组里的选项（默认设置是 [client] 选项组）

`connect()` 方法的第四个参数是可选的，这个参数可以用来传递其他一些必要的连接属性。在调用 `connect()` 方法的时候，既可以直接写出这些参数，也可以通过一个数组变量来传递它们，如下所示：

```
$dbh = DBI->connect($source, $user, $pw, {Attr1=>val1, Attr2=>val2});
%attr = (Attr1=>val1, Attr2=>val2);
$dbh = DBI->connect($source, $user, $pw, \%attr);
```

此外，绝大多数连接属性还可以在连接建立起来之后再进行设置或修改，如下所示：

```
$dbh->{'LongReadLen'} = 1000000;
```

下面的表格列出了最重要的连接属性。

数据源字符串 `$datasource` 里的可选参数

<code>RaiseError=>0/1</code>	是否在连接发生错误时显示一条出错消息（默认设置是 0：不显示出错消息）
<code>PrintError=>0/1</code>	是否在连接发生错误时显示一条出错消息但继续执行（默认设置是 1：显示出错消息并继续执行）
<code>LongReadLen=>n</code>	读取单个数据字段时实际读入的最大字节长度（默认设置是 0：不管数据字段多长都全部读入）
<code>LongTruncOK=>0/1</code>	判断数据字段长度（1 代表太长要被截断；0 代表显示出错消息并继续执行）

断开与数据库的连接

<code>\$dbh->disconnect();</code>	断开与数据库的连接
--------------------------------------	-----------

23.3.3 执行 SQL 命令、处理 SELECT 查询结果

执行 SQL 命令——不会返回结果记录的 SQL 命令

<code>\$n = \$dbh->do("INSERT ...");</code>	执行一条不会返回结果记录的 SQL 命令。 <code>\$n</code> 包含着被这条 SQL 命令改变了的数据记录的总数，这里有 3 种特殊情况： <code>0E0</code> ，没有改变任何记录； <code>-1</code> ，无法确定这个总数； <code>undef</code> ，执行出错
<code>\$n = \$dbh->do(\$sql, \%attr, \@values);</code>	执行一条带参数的 SQL 命令。数组参数 <code>@values</code> 的各个元素是这条 SQL 命令里的各个参数（这些参数用问号“?”表示）的参数值，DBI 会自动调用 <code>quote()</code> 函数对这些参数值进行必要的转义处理； <code>%attr</code> 参数用来传递一些必要的可选属性（或者是 <code>undef</code> ，即不需要使用可选属性）
<code>\$id = \$dbh->('mysql_insertid');</code>	MySQL 服务器为新插入的记录自动生成的 <code>AUTO_INCREMENT</code> 编号值（注意： <code>mysql_insertid</code> 属性仅适用于 MySQL 数据库系统）

执行 SQL 命令——会返回结果记录的 SQL 命令

<code>\$sth = \$dbh->prepare("SELECT ...");</code>	对 SQL 查询命令（通常是 <code>SELECT</code> 命令）进行预处理。后续的所有操作都必须使用语句句柄 <code>\$sth</code> 来进行
<code>\$sth->execute();</code>	执行经过预处理的 SQL 查询命令
<code>\$sth->execute(@values);</code>	执行带参数的 SQL 查询命令。数组参数 <code>@values</code> 的各个元素是这条 SQL 命令里的各个参数（这些参数用问号“?”表示）的参数值
<code>\$sth->fetchxxx();</code>	对查询结果进行处理（参见下面的表格）
<code>\$sth->finish();</code>	释放语句句柄 <code>\$sth</code> 占用的内存和其他资源

如果一条 SQL 命令是用`prepare()`和`execute()`方法执行的并返回了一些记录作为查询结果，那些结果记录可以通过下面这个表格里的各种`fetchxxx()`方法来处理。

处理结果记录

<code>@row = \$sth->fetchrow_array();</code>	把下一条结果记录读入普通数组 <code>@row</code> ；如果已经到达最后一条结果记录或发生错误， <code>@row</code> 将包含一个空数组。对各个结果字段（数据列）的访问需要通过 <code>\$row[n]</code> 的形式进行（ <code>n=0</code> 对应着第一个结果字段）
<code>@row = \$sth->fetch();</code>	与 <code>fetchrow_array()</code> 方法完全等价
<code>\$rowptr = \$sth->fetchrow_arrayref();</code>	与 <code>fetchrow_array()</code> 方法的功能基本相同，但这次返回的不是一个数组而是一个指向这个数组的指针。如果已经到达最后一条结果记录或发生错误，这个方法将返回 <code>undef</code>
<code>\$row = \$sth->fetchrow_hashref();</code>	把下一条结果记录读入关联数组 <code>\$row</code> ；如果已经到达最后一条结果记录或发生错误， <code>\$row</code> 将包含着 <code>undef</code> 值。对各个结果字段（数据列）的访问需要通过 <code>\$row->{'columnname'}</code> 的形式进行并区分字段名的字母大小写情况
<code>\$result = \$sth->fetchall_arrayref();</code>	读取所有的结果记录并返回一个指针数组的指针，这个指针数组里的各个元素（指针）分别指向一个普通数组，每个普通数组包含着一条结果记录。对各个结果字段（数据列）的访问需要通过 <code>\$result->[\$row][\$col]</code> 的形式来进行
<code>\$result = \$sth->fetchall_arrayref({});</code>	同上，但这次将把结果记录存放在关联数组里。对各个结果字段（数据列）的访问需要通过 <code>\$result->[\$row]->{'columnname'}</code> 的形式来进行

把结果字段（数据列）绑定到 Perl 变量上（为调用 *fetch_array()*方法做准备）

<code>\$sth->bind_col(\$n, \\$var);</code>	把第 <i>n</i> 个结果字段绑定到 Perl 变量 <i>\$var</i> 上（第一个结果字段对应着 <i>\$n=1</i> ）。此后，在每次读取下一条结果记录的时候，变量 <i>\$var</i> 都会被自动刷新。对 <i>bind_col()</i> 方法的调用必须发生在 <i>execute()</i> 方法之后。如果发生错误，这个方法将返回 <i>false</i>
<code>\$sth->bind_columns(\\$var1, \\$var2, ...);</code>	与 <i>bind_col()</i> 方法的功能基本相同，但这次是对查询结果里的所有字段（数据列）都进行绑定。此时，程序员必须注意自己是否提供了足够的 Perl 变量

关于 SQL 命令的元信息

<code>\$n = \$sth->{'NUM_OF_FIELDS'};</code>	结果字段（数据列）的总数（在执行完 <i>SELECT</i> 命令后调用）
<code>\$n = \$sth->{'NUM_OF_PARAMS'};</code>	带参数的 SQL 命令里的参数个数
<code>\$sql = \$sth->{'Statement'}</code> ;	SQL 命令的文本内容

关于 *SELECT* 查询结果的元信息——结果字段的名字、数据类型等

<code>\$array_ref = \$sth->{'NAME'}</code> ;	返回一个数组指针，这个数组里的元素包含着各个结果字段的名字。这些名字需要通过 <code>@{\$array_ref}[\$n]</code> 的形式去访问，其中 <i>n</i> 的取值范围是 0 到 <code>\$sth->{'NUM_OF_FIELDS'}</code> -1
<code>\$array_ref = \$sth->{'NAME_lc'}</code> ;	同上，但字段名全部是小写字母
<code>\$array_ref = \$sth->{'NAME_uc'}</code> ;	同上，但字段名全部是大写字母
<code>\$array_ref = \$sth->{'NULLABLE'}</code> ;	各个结果字段是否允许包含 <i>NULL</i> 值：1，允许；0，不允许；2，无法确定
<code>\$array_ref = \$sth->{'PRECISION'}</code> ;	各个结果字段的精度（这个“精度”取的是 ODBC 含义，即数据列的最大宽度）
<code>\$array_ref = \$sth->{'SCALE'}</code> ;	浮点数小数部分的位数
<code>\$array_ref = \$sth->{'TYPE'}</code> ;	各个结果字段的数据类型（以数据类型在 ODBC 标准中的编号表示）。笔者的测试结果是： <i>CHAR</i> : 12; <i>INT</i> : 4; <i>TEXT/BLOB</i> : -1; <i>DATE</i> : 9; <i>TIME</i> : 10; <i>TIMESTAMP</i> : 11; <i>FLOAT</i> : 7; <i>DECIMAL</i> : 3; <i>ENUM/SET</i> : 1

快捷方式

<code>@row = \$dbh->selectrow_array(\$sql);</code>	相当于 <i>prepare()</i> 、 <i>execute()</i> 和 <i>fetchrow_array()</i> 这 3 个方法的组合。注意，这个方法只能返回查询结果中的第一条记录给 <i>@row</i> 数组，后面的结果记录不可访问
<code>\$result = \$dbh->selectrow_array(\$sql);</code>	同上，但 <i>\$result</i> 包含的是第一条结果记录的第一个字段的值
<code>\$result = \$dbh->selectall_arrayref(\$sql);</code>	相当于 <i>prepare()</i> 、 <i>execute()</i> 和 <i>fetchrow_arrayref()</i> 这 3 个方法的组合。对 <i>\$result</i> 的进一步处理参见前面表格里对 <i>fetchrow_arrayref()</i> 方法的说明

给字符串和 BLOB 数据里的特殊字符加上反斜线

<code>\$dbh->quote(\$data);</code>	把 <i>\$data</i> 的内容用单引号全部括起来，把其中的零值字节以及字符 “`” 和 “\” 分别替换为 “\0”、“\`” 和 “\\”。如果 <i>\$data</i> 为空 (<i>undef</i>)， <i>quote()</i> 方法将返回字符串 <i>NULL</i> （注意，不是 <i>NULL</i> 值）
---------------------------------------	--

事务

<code>\$dbh->{AutoCommit} = 0;</code>	关闭 <i>autocommit</i> 模式。此后执行的所有 SQL 命令将构成一个事务
<code>\$dbh->commit();</code>	确认一个事务
<code>\$dbh->rollback();</code>	放弃一个事务

23.3.4 出错处理

与出错处理有关的 DBI 方法

<code>\$h->err();</code>	最近一个错误的出错代码（或者 0：没有错误）
<code>\$h->errstr();</code>	最近一个错误的出错信息（或者空字符串：没有错误）
<code>DBI->trace(\$n [, \$filename]);</code>	启用 DBI 模块的日志功能，把 DBI 内部对数据库数据的访问情况输出到 STDERR 设备或一个给定的文件。参数 \$n 控制着日志信息的详细程度： 1，日志信息量最少；15，日志信息量最多

23.3.5 辅助函数

DBI 函数

<code>@bool = DBI::looks_like_a_number(@data);</code>	测试 @data 数组的各个元素是不是一个数值 (<i>true</i> : 是; <i>false</i> : 不是)， 测试结果将返回在数组 @bool 里
<code>\$result = DBI::neat(\$data [, \$ maxlen]);</code>	把变量 \$data 里的字符串数据排版为适合输出的格式：用单引号把字符串括起来；把其中的非 ASCII 字符替换为 “.”（英文句号）；如果 \$data 字符串的长度超过了参数 \$maxlen 设置的字符个数（默认设置是 400 个字符），把 \$data 字符串截短到只剩 \$maxlen-3 个字符，再加上 “...”（3 个英文句号）作为它的结尾
<code>\$result = DBI::neat_list(@listref, \$maxlen, \$sep);</code>	同上，但这次是对一个字符串列表进行上述排版处理：列表元素之间的分隔符由参数 \$sep 给定（默认为 “,”）

\$dbh 方法

<code>\$ok = \$dbh->ping();</code>	测试与 MySQL 服务器的连接是否可用： <i>true</i> ， 可用； <i>false</i> ， 不可用
---------------------------------------	--

23.3.6 DBD::mysql 驱动程序中的 MySQL 专用扩展模块

如果使用 DBI 模块的目的只是为了访问 MySQL 数据库，还可以利用 DBI 模块提供的 MySQL 专用方法和属性提高 DBI 程序的执行效率。下面是 DBI 模块和 DBD::mysql 驱动程序里的一些最重要的 MySQL 专用方法和属性。这些方法和属性可以简化编程工作，提高 Perl 程序的运行效率。不过，用它们编写出来的代码不具备可移植性，如果想用这些代码去访问另一种数据库系统，代码修改工作的量可能会很大。

在一条给定连接上执行数据库管理任务

<code>\$drh = DBI->install_driver('mysql');</code>	返回一个驱动模块句柄
<code>\$drh->func('createdb', \$database, \$host, \$user, \$password, 'admin');</code>	创建一个新数据库；为此创建的新连接将在完成任务后关闭
<code>\$drh->func('dropdb', \$database, \$host, \$user, \$password, 'admin');</code>	删除一个数据库
<code>\$drh->func('shutdown', \$host, \$user, \$password, 'admin');</code>	关闭 MySQL 服务器
<code>\$drh->func('reload', \$host, \$user, \$password, 'admin');</code>	重新加载所有的 MySQL 数据表（mysql 数据库里用来完成访问控制和权限管理任务的各个数据表也包括在内）

在当前连接上执行数据库管理任务

<code>\$dbh->func('createdb', \$database, 'admin');</code>	创建一个新数据库
---	----------

(续)

在当前连接上执行数据库管理任务

<code>\$dbh->func('dropdb', \$database, 'admin');</code>	删除一个数据库
<code>\$dbh->func('shutdown', 'admin');</code>	关闭 MySQL 服务器
<code>\$dbh->func('reload', 'admin');</code>	重新加载所有的 MySQL 数据表

\$dbh 属性

<code>\$info = \$dbh->{'mysql_hostinfo'};</code>	返回一个字符串，内容是关于 MySQL 服务器主机的信息（比如：“192.168.80.128 via TCP/IP”）
<code>\$info = \$dbh->{'mysql_info'};</code>	返回一个字符串，内容是刚才执行的 SQL 命令的状态信息（比如说，在执行了一条 UPDATE 命令后，这个字符串将是 <i>Rows matched: 13 Changed: 13 Warnings: 0</i> 的样子）
<code>\$id = \$dbh->{'mysql_insertid'};</code>	MySQL 服务器为新插入的记录自动生成的 <i>AUTO_INCREMENT</i> 编号值
<code>\$n = \$dbh->{'mysql_protoinfo'};</code>	返回一个整数：通信双方使用的通信协议的编号（如 10）
<code>\$info = \$dbh->{'mysql_serverinfo'};</code>	返回一个字符串：MySQL 服务器的版本信息（如“5.0.2-alpha-standard”）
<code>\$info = \$dbh->{'mysql_stat'};</code>	返回一个字符串：MySQL 服务器的工作状态（有多少个线程、打开了多少个数据表等）
<code>\$threadid = \$db->{'mysql_thread_id'};</code>	返回一个整数：这条 MySQL 连接的线程 ID 编号

\$sth 方法和属性

<code>\$sth->rows();</code>	刚才执行的 SELECT 命令查找到的数据记录个数；注意：不能与 <code>\$sth->{'mysql_use_result'}</code> 属性一起使用
<code>\$sth->{'mysql_store_result'}=1;</code>	启用 <code>mysql_store_result</code> 模式，MySQL 服务器将把 SELECT 查询结果中的所有记录一次性地全部传输到客户端（这是默认设置）
<code>\$sth->{'mysql_use_result'}=1;</code>	启用 <code>mysql_use_result</code> 模式，MySQL 服务器只在必要时才把 SELECT 查询结果中的一条记录传输到客户端

\$sth 属性——确定 SELECT 查询结果的元数据

<code>\$ar_ref = \$sth->{'mysql_is_auto_increment'};</code>	测试各结果字段（数据列）是否具有 <i>AUTO_INCREMENT</i> 属性。本表格里的所有方法都返回一个数组指针，而该数组的各个元素将表明各结果字段是否具有某种属性。对数组元素的访问需要通过 <code>@{\$ar_ref}[\$n]</code> 进行，其中 <code>\$n</code> 的取值范围是 0 到 <code>\$sth->{'NUM_OF_FIELDS'}</code> -1
<code>\$ar_ref = \$sth->{'mysql_is_blob'};</code>	测试各结果字段（数据列）是否包含 <i>BLOB</i> 数据
<code>\$ar_ref = \$sth->{'mysql_is_key'};</code>	测试各结果字段（数据列）是不是有索引
<code>\$ar_ref = \$sth->{'mysql_is_not_null'};</code>	测试各结果字段（数据列）是否具有 <i>NOT NULL</i> 属性
<code>\$ar_ref = \$sth->{'mysql_is_num'};</code>	测试各结果字段（数据列）是不是存储着数值
<code>\$ar_ref = \$sth->{'mysql_is_pri_key'};</code>	测试各结果字段（数据列）是不是主索引的组成部分
<code>\$ar_ref = \$sth->{'mysql_max_length'};</code>	返回各结果字段（数据列）的最大宽度
<code>\$ar_ref = \$sth->{'mysql_table'};</code>	返回各结果字段（数据列）所属的数据表的名字
<code>\$ar_ref = \$sth->{'mysql_type_name'};</code>	返回各结果字段（数据列）的数据类型

23.4 JDBC (Connector/J)

要想通过 Java 去访问 MySQL，就必须安装一个用来访问 MySQL 数据库的 JDBC 驱动程序；本书假设用户安装的是 Connector/J 3.n 版本。在这个假设得到满足的前提下，就可以使用 JDBC（Java Database Connectivity，Java 数据库互联标准）驱动程序提供的各种类和方法去访问 MySQL 数据库，这些类和方法的名字都是 `java.sql.*` 和 `javax.sql.*` 的形式。JDBC 是一个非常复杂的 API 函数库，本书实在拿不出足够的篇幅来把它完整地介绍给大家，下面的表格只对最重要的 JDBC 类和方法进行了汇总。

23.4.1 与 MySQL 服务器建立连接

利用 `DriverManager` 类来连接 MySQL 数据库

<code>import java.sql.*;</code>	导入 JDBC 基类
<code>Class.forName("com.mysql.jdbc.Driver").newInstance();</code>	加载并注册 Connector/J，它是人们为 MySQL 数据库系统开发的各种 JDBC 驱动程序中的一种
<code>Connection conn = DriverManager.getConnection ("jdbc:mysql://hostname/dbname", "username", "password");</code>	尝试与主机 <code>hostname</code> 上的 <code>dbname</code> 数据库建立连接。在连接字符串里可以传递各种可选的参数，第 17 章里的两个表格对其中最为重要的参数进行了汇总

利用 `DataSource` 类来连接 MySQL 数据库（仅适用于 Java 2 v1.4 及更高版本）

<code>import java.sql.*;</code>	导入 JDBC 基类和 JDBC 扩展类
<code>import javax.sql.*;</code>	
<code>com.mysql.jdbc.jdbc2.optional.MysqlDataSource ds = new com.mysql.jdbc.jdbc2.optional.MysqlDataSource();</code>	先创建一个 <code>com.mysql.jdbc.jdbc2.optional.MysqlDataSource</code> 类的对象，然后设置主机名和数据库名，最后去建立连接
<code>ds.setServerName("hostname");</code>	
<code>ds.setDatabaseName("dbname");</code>	
<code>Connection conn = ds.getConnection("username", "password");</code>	代替 <code>setServerName()</code> 和 <code>setDatabaseName()</code> 方法去设置各有关连接参数；连接字符串（URL）的语法与 <code>DriverManager.getConnection()</code> 方法所使用的连接字符串语法相同

23.4.2 执行 SQL 命令

执行 SQL 命令（`Statement` 对象）

<code>Statement stmt = conn.createStatement();</code>	创建一个 <code>Statement</code> 对象，这是执行 SQL 命令所必须的
<code>Statement stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY, java.sql.ResultSet.CONCUR_READ_ONLY);</code>	定义一个 <code>Statement</code> 对象，用这个对象去执行 <code>SELECT</code> 命令将返回一个只允许向前遍历结果记录（ <code>TYPE_FORWARD_ONLY</code> ）和只读（ <code>CONCUR_READ_ONLY</code> ）的 <code>ResultSet</code> 对象

(续)

执行 SQL 命令 (Statement 对象)

<code>Statement stmt = conn.createStatement(java.sql.ResultSet.TYPE_SCROLL_SENSITIVE, java.sql.ResultSet.CONCUR_UPDATABLE); int n = stmt.executeUpdate("INSERT ..."); stmt.getWarnings();</code>	定义一个 <i>Statement</i> 对象，用这个对象去执行 <i>SELECT</i> 命令将返回一个允许自由遍历结果记录 (<i>TYPE_SCROLL_SENSITIVE</i>) 和允许修改 (<i>CONCUR_UPDATABLE</i>) 的 <i>ResultSet</i> 对象
	执行 <i>INSERT</i> 、 <i>UPDATE</i> 和 <i>DELETE</i> 命令。返回值是被修改的记录总数
	<i>getWarning()</i> 方法可以让我们知道刚才执行的 SQL 命令引起了多少次警告（相当于 SQL 命令 <i>SHOW WARNINGS</i> ）。这个方法将返回一个 <i>SQLWarning</i> 对象，直到所有的警告都被处理完毕
<code>ResultSet res = stmt.executeQuery("SELECT ..."); stmt.addBatch("INSERT ..."); stmt.addBatch("INSERT ..."); int[] n = stmt.executeBatch();</code>	执行一条 <i>SELECT</i> 查询命令并返回一个 <i>ResultSet</i> 对象作为结果 一次执行多条 SQL 命令： <i>executeBatch()</i> 方法将返回一个 <i>int</i> 数组，该数组的各个元素分别是各条 SQL 命令改变的记录总数

确定 *INSERT* 命令插入的新记录的 AUTO_INCREMENT 编号值

<code>stmt.executeUpdate("INSERT ..."); ResultSet newid = stmt.getGeneratedKeys(); if(newid.next()) { int id = newid.getInt(1); long id = ((com.mysql.jdbc.Statement)stmt).getLastInsertID(); }</code>	以下 3 种变体的出发点 <i>getGeneratedKeys()</i> 方法将返回一个 <i>ResultSet</i> 对象，这个对象包含最新生成的一个或多个 ID 编号值。如果 <i>INSERT</i> 命令只插入了一条新记录，那就只有一个 ID 编号值；这个编号值可以利用 <i>next()</i> 和 <i>getInt(1)</i> 方法读出。 <i>getGeneratedKeys()</i> 方法始见于 Java 2 v1.4 版本
	<i>getLastInsertID()</i> 方法也可以返回最新生成的 ID 编号值，但这个方法是 Connector/J 独有的，不具备可移植性
	这种变体使用了一条 SQL 命令来返回最新生成的 ID 编号值。注意，这条 SQL 命令与刚才执行的 <i>INSERT</i> 命令必须执行在同一个事务里

执行预处理语句 (PreparedStatement 对象)

<code>PreparedStatement pstmt = conn.prepareStatement("INSERT ... (?, ?)"); pstmt.setString(1, "O'Reilly"); pstmt.setInt(2, 7878);</code>	声明了一条带有两个参数的 SQL 命令，SQL 命令里的参数用问号 (?) 表示
	传递参数。不同数据类型的参数要用不同的方法来传递，除了如左栏所示的 <i>setString()</i> 和 <i>setInt()</i> ，还有 <i>setNull()</i> 、 <i>setDate()</i> 、 <i>setTime()</i> 、 <i>setFloat()</i> 、 <i>setBinaryStream()</i> 等。这些方法都需要两个参数：SQL 命令里的参数的编号（从 1 开始）和它的参数值
<code>int n = pstmt.executeUpdate(); ResultSet res = pstmt.executeQuery(); pstmt.addBatch(); int n pstmt.executeBatch();</code>	执行 SQL 命令：这个方法与 <i>Statement</i> 类的同名方法完全一样

23.4.3 处理 SELECT 查询结果 (ResultSet 类)

改变 ResultSet 对象

<code>res.deleteRow();</code>	删除当前结果记录
<code>res.updateXxx(n, data);</code>	先修改当前记录的第 <i>n</i> 个字段，然后保存所做的修改
<code>res.updateRow();</code>	
<code>res.moveToInsertRow();</code>	插入一条新记录，修改它的第 <i>n</i> 个字段，然后保存所做的修改
<code>res.updateXxx(n, data);</code>	
<code>res.insertRow();</code>	

处理 ResultSet 对象

<code>res.getInt(n);</code>	返回当前结果记录的第 <i>n</i> 个字段的值。除了使用字段的编号（从 1 开始），还可以使用字段的名字，例如 <code>getDate("birthdate")</code>
<code>res.getString(n);</code>	
<code>res.getBytes(n);</code>	
<code>...</code>	
<code>res.wasNull();</code>	测试最新读入的字段值是不是 <i>NULL</i> 。对那些无法储存 <i>NULL</i> 、因而只能用 0 来代替 <i>NULL</i> 的 Java 基本数据类型来说，这种测试非常有必要—— <code>wasNULL()</code> 方法是区分 0 和 <i>NULL</i> 的唯一手段
<code>res.getBinaryStream(n);</code>	返回一个 <i>InputStream</i> 对象，准备按字节读取二进制数据
<code>res.getCharacterStream(n);</code>	返回一个 <i>Reader</i> 对象，准备按字符读取二进制数据
<code>res.getBlob(n);</code>	返回一个 <i>Blob</i> 对象，准备读取二进制数据
<code>res.getClob(n);</code>	返回一个 <i>Clob</i> 对象，准备读取二进制数据

遍历 ResultSet 对象

<code>res.next();</code>	把 <i>ResultSet</i> 对象中的下一条结果记录选定为当前记录。如果已经到达最后一条结果记录，这个方法将返回 <i>false</i>
<code>res.first();</code>	把 <i>ResultSet</i> 对象中的第一条结果记录选定为当前记录。如果这个 <i>ResultSet</i> 对象里没有包含任何记录，这个方法将返回 <i>false</i>
<code>res.previous();</code>	把上一条记录选定为当前记录
<code>res.last();</code>	把最后一条记录选定为当前记录
<code>res.beforeFirst();</code>	把记录读取光标定位在第一条记录之前，这也是 <i>ResultSet</i> 对象刚由 <code>executeQuery()</code> 方法创建出来时的初始状态。此时调用 <code>next()</code> 方法将把第一条记录选定为当前记录
<code>res.afterLast();</code>	把记录读取光标定位在最后一条记录之后。此时调用 <code>previous()</code> 方法将把最后一条记录选定为当前记录
<code>res.isFirst();</code>	测试当前记录是不是第一条记录/最后一条记录
<code>res.isLast();</code>	
<code>int n = res.getRow();</code>	返回当前记录的编号（第一条记录的编号是 1）
<code>res.absolute(n);</code>	把第 <i>n</i> 条记录选定为当前记录

ResultSet 对象的元数据

<code>ResultSetMetaData meta = res.getMetaData();</code>	返回一个 <i>ResultSetMetaData</i> 对象，这个对象包含着关于 <i>SELECT</i> 查询结果的信息
<code>meta.getColumnCount();</code>	返回结果字段（数据列）的总数

(续)

ResultSet 对象的元数据

<code>meta.getColumnName(i);</code>	返回第 <i>i</i> 个字段的名字（一个 <i>String</i> 类型的值）
<code>meta.getColumnType(i);</code>	返回第 <i>i</i> 个字段的数据类型的编号。这是一个 <i>int</i> 类型的值，是 <i>java.sql.Types</i> 集合所定义的常数之一
<code>meta.getColumnTypeName(i);</code>	返回第 <i>i</i> 个字段的数据类型的名字（一个 <i>String</i> 类型的值）
<code>meta.isNullable(i);</code>	测试第 <i>i</i> 个字段是否允许包含 <i>NULL</i> 值
<code>meta.isAutoIncrement(i);</code>	测试第 <i>i</i> 个字段是不是一个 <i>AUTO_INCREMENT</i> 数据列

23.4.4 事务**事务**

<code>conn.setAutoCommit(false);</code>	启用事务机制（当然，有关的 MySQL 数据表必须支持事务才行）
<code>conn.commit();</code>	确认在当前事务里执行过的所有 SQL 命令并开始下一个事务
<code>conn.rollback();</code>	放弃在当前事务里执行过的所有 SQL 命令并开始一个新事务

23.5 ADO .NET (Connector/Net)

以下表格对 Connector/Net（即 *MySql.Data.dll* 库文件）提供的最重要的类和方法进行了汇总。表格内容里使用的是 Visual Basic 语法。

23.5.1 与 MySQL 服务器建立连接**与 MySQL 服务器建立连接**

<code>Imports MySql.Data.MySqlClient</code>	导入 Connector/Net 类
<code>Dim myconn As MySqlConnection myconn = New MySqlConnection("Data Source=localhost; Initial Catalog=mylibrary; User ID=root;PWD=xxxxxx") myconn.Open()</code>	创建一条与 MySQL 服务器的连接

MySqlConnection 类的方法和属性

<code>BeginTransaction</code>	开始一个事务并返回一个 <i>MySqlTransaction</i> 对象（参见下面的表格）
<code>Close</code>	断开一条连接
<code>ConnectionString</code>	连接字符串，其内容是创建连接时需要用到的各种连接属性
<code>CreateCommand</code>	创建一个 <i> MySqlCommand</i> 对象
<code>Dispose</code>	释放这条连接占用的内存
<code>Ping</code>	测试与 MySQL 服务器的连接是否可用
<code>ServerThread</code>	给定连接的 MySQL 服务器线程编号
<code>ServerVersion</code>	MySQL 服务器的版本信息字符串（如 <i>5.0.2-alpha-standard-log</i> ）
<code>State</code>	关于连接状态的信息（数据类型是 <i>System.Data.ConnectionState</i> ）

23.5.2 执行 SQL 命令与处理 SELECT 查询结果

执行 SQL 命令 (MySqlCommand 类)

<code>Dim com As MySqlCommand</code>	创建一个 <i>MySqlCommand</i> 对象的两种办法
<code>com = myconn.CreateCommand("sql")</code>	
<code>com = New MySqlCommand("sql", _</code>	
<code>myconn)</code>	
<code>com.ExecuteNonQuery()</code>	执行一条不会返回任何结果记录的 SQL 命令(比如 <i>UPDATE</i> 或 <i>INSERT</i> 命令)
<code>obj = com.ExecuteScalar()</code>	返回一个离散值结果。 <i>ExecuteScalar()</i> 方法只能用来执行那些返回且仅返回一个数据行和一个数据列的 <i>SELECT</i> 查询命令。这个方法的返回值是一个 <i>Object</i> 数据类型的对象，必须使用一个转换函数(例如: <i>CInt()</i>)或一个投射操作符(例如: <i>(int)</i>)把它转换为需要的数据格式
<code>dr = com.ExecuteReader()</code>	返回一个 <i>MySqlDataReader</i> 对象(参见下面的表格)
<code>Dim n As Long com.CommandText =</code> <code>"SELECT LAST_INSERT_ID() "n =</code> <code>CLng(com.ExecuteScalar())</code>	返回由 <i>INSERT</i> 命令最新插入的数据记录的 ID 编号值 (<i>AUTO_INCREMENT</i> 数据列)

执行带参数的 SQL 命令 (MySqlParameter 类)

<code>Dim com As MySqlCommand</code>	对带参数的 SQL 命令进行预处理。参数以“?name”形式给出。必须创
<code>Dim p1, p2, p3 As MySqlParameter</code>	建一个 <i>MySqlParameter</i> 对象并在这个对象里对各有关参数的数据类型做
<code>com = myconn.CreateCommand()</code>	出声明。带参数的 SQL 命令必须经过 <i>Prepare()</i> 方法的预处理才能执行
<code>com.CommandText = _</code> <code>"INSERT ...VALUES(?a, ?b, ?c)"</code>	
<code>p1 = com.Parameters.Add("?a", _</code> <code>MySqlDbType.VarChar)</code>	
<code>p2 = com.Parameters.Add("?b", _</code> <code>MySqlDbType.Int32)</code>	
<code>...</code>	
<code>com.Prepare()</code>	
<code>p1.Value = ...</code>	把参数值传递给 <i>MySqlParameter</i> 对象的 <i>Value</i> 属性，然后用前面介绍
<code>p2.Value = ...</code>	过的 <i>Execute()</i> 方法执行那条 SQL 命令
<code>com.ExecuteXxx()</code>	

对 SELECT 查询结果进行处理 (MySqlDataReader 类)

<code>Dim dr As MySqlDataReader</code>	执行一条 SQL 命令并把结果返回为一个 <i>MySqlDataReader</i> 对象。对
<code>dr = com.ExecuteReader()</code>	SELECT 查询结果的访问将按照 <i>forward-only</i> (单向遍历) 和 <i>read-only</i> (只读) 方式进行
<code>dr.HasRows</code>	测试 <i>DataReader</i> 对象里是否包含着数据
<code>dr.FieldCount</code>	返回 <i>DataReader</i> 对象里的结果字段(数据列)总数
<code>While dr.Read()</code> <code>n = CInt(dr!publID)</code> <code>s = CStr(dr!publName)</code> <code>End While</code>	依次输出 <i>DataReader</i> 对象里的结果记录(数据行)。如果已经到达最后一条结果记录， <i>Read()</i> 方法将返回 <i>False</i> 。对当前记录里的各个字段的访问需要通过 <code>dr!columnname</code> (VB.NET 程序) 或 <code>dr["columnname"]</code> (C#程序) 的形式来进行。 <code>dr!columnname</code> 和 <code>dr["columnname"]</code> 的返回值都是 <i>Object</i> 数据类型，必须用 <i>Cdatatype()</i> 函数 (VB.NET 程序) 或 <i>(datatype)</i> 投射操作符 (C#程序) 把它们转换为正确的数据格式

(续)

对 SELECT 查询结果进行处理 (MySqlDataReader 类)

<i>dr.GetName(n)</i>	返回第 <i>n</i> 个结果字段 (数据列) 的名字 (第一个字段对应着 <i>n=0</i>)
<i>dr.GetDataTypeName(n)</i>	返回第 <i>n</i> 个结果字段 (数据列) 的数据类型的名字
<i>dr.IsDBNull(n)</i>	测试当前记录的第 <i>n</i> 个结果字段是否包含着 <i>NULL</i> 值
<i>dr.GetByte(n)</i>	直接读取第 <i>n</i> 个结果字段的值。这几个方法特别适合用来处理二进制数
<i>dr.GetBytes(n, ...)</i>	据
<i>dr.GetChar(n)</i>	
<i>dr.GetDateTime(n)</i>	
<i>...</i>	
<i>bool = dr.NextResult()</i>	前进到下一条 SELECT 命令的查询结果。如果已经到达最后一组 SELECT 查询结果, 这个方法将返回 <i>False</i>
<i>dr.Close()</i>	关闭 <i>DataReader</i> 对象并释放其中的数据

23.5.3 利用 DataSet/DataTable 类修改数据

MySqlDataAdapter 和 MySqlCommandBuilder 对象的用法

<i>Dim da As New _</i> <i>MySqlDataAdapter(com)</i>	在 SQL 命令 <i>com</i> (一个 <i>MySqlCommand</i> 对象) 的基础上创建一个 <i>DataTable</i> 对象并通过一个 <i>DataSet</i> 对象用名字 <i>dtname</i> 把查询结果传递给 这个 <i>DataTable</i> 对象。这样就可以通过这个 <i>DataTable</i> 对象去访问 SQL 命 令 <i>com</i> 的查询结果了
<i>Dim ds As New DataSet()</i> <i>da.Fill(ds, "dtname")</i>	
<i>Dim dt As DataTable = _</i> <i>ds.Tables("dtname")</i>	
<i>n = dt.Count</i>	确定 <i>DataTable</i> 对象里的记录总数
<i>Dim row As DataRow</i> <i>For Each row In dt.Rows</i> <i>var = row!columnname</i> <i>Next</i>	用一个循环去遍历一个 <i>DataTable</i> 对象里的所有记录。在 C# 程序里, 对各个字段 (数据列) 的访问需要通过 <i>dr["columnname"]</i> 的形式来进行
<i>row.Delete()</i>	删除 <i>DataTable</i> 对象里的当前记录
<i>row!columnname = ...</i>	修改 <i>DataTable</i> 对象里的当前记录
<i>row.Update()</i>	
<i>Dim newrow As DataRow</i> <i>newrow = dt.NewRow()</i> <i>newrow!columnname = ...</i> <i>dt.Rows.Add(newrow)</i>	在 <i>DataTable</i> 对象里插入一条新记录
<i>Dim cb As New _</i> <i>MySqlCommandBuilder(da)</i> <i>da.Update(ds, "dtname")</i>	把在客户端对 <i>DataTable</i> 对象进行的修改永久地存入 MySQL 服务器。 <i>MySqlCommandBuilder</i> 对象提供了必要的 SQL 修改命令
<i>cb.GetDeleteCommand()</i> <i>cb.GetInsertCommand()</i> <i>cb.GetUpdateCommand()</i>	把由 <i>MySqlCommandBuilder</i> 对象创建的 <i>DELETE</i> 、 <i>INSERT</i> 或 <i>UPDATE</i> 命令返回为一个 <i>MySqlCommand</i> 对象 (这些命令的 SQL 代码可以通过 <i>CommandText</i> 属性读到)

23.5.4 事务

事务	
<i>Dim tr As MySqlTransaction</i>	创建一个 <i>MySqlTransaction</i> 对象
<i>tr = myconn.BeginTransaction()</i>	
<i>Dim com As New MySqlCommand(</i> “ <i>UPDATE ...</i> ”, <i>myconn, tr</i>) <i>com.ExecuteNonQuery()</i> ... <i>further commands of the transaction</i>	创建一个 <i>MySqlCommand</i> 对象并在这个事务的框架里执行它
<i>tr.Commit()</i>	确认这个事务
<i>tr.Rollback()</i>	放弃这个事务

23.6 C API

以下表格对最重要的 MySQL C API 函数和数据结构进行了汇总。

23.6.1 数据结构

数据结构	
<i>MYSQL *conn;</i>	结构, 用来存放连接数据
<i>MYSQL_RES *result;</i>	结构, 用来存放 <i>SELECT</i> 查询结果
<i>MYSQL_ROW row;</i>	指针, 指向当前结果记录 (数据行)
<i>MYSQL_ROW_OFFSET roffset;</i>	指针, 表明 <i>SELECT</i> 查询结果中的偏移量
<i>MYSQL_FIELD *field;</i>	结构, 用来字段 (数据列) 的元数据 (数据列的名字、数据类型、十进制数字的个数、字符数等); 详见下面的表格
<i>MYSQL_FIELD_OFFSET foffset;</i>	字段在数据记录里的偏移量 (0 对应着第一个字段, 1 对应着第二个字段等)
<i>MYSQL_STMT *stmt;</i>	结构, 用来处理预处理语句
<i>MYSQL_BIND bind[n];</i>	结构, 用来描述预处理语句的参数
<i>MYSQL_TIME mytime;</i>	结构, 用来把日期/时间值传递给预处理语句
<i>my_ulonglong n;</i>	64 位整数, 后面的表格里有一些 MySQL 函数的返回值是这种数据类型

在接下来的语法表里, 将假设变量 *conn*、*result*、*row*、*field*、*roffset*、*foffset* 等都已经像上面这个表格所演示的那样得到了正确的声明。注意, 上表中的数据结构 *MYSQL_ROW* 已经是一个指针了, 所以在声明 *row* 变量时不需要使用 “*” 字符。

<i>MYSQL_FIELD</i> 结构的元素	
<i>char *name;</i>	这个数据列 (字段) 的名字
<i>char *table;</i>	提供这个数据列的数据表的名字。如果这个数据列是计算出来的或者是其他数据列的一个假名, <i>table</i> 将指向那个计算公式或那个 <i>ALIAS</i> 假名定义子句
<i>char *def;</i>	这个数据列的默认值或 <i>NULL</i>
<i>enum enum_field_types type;</i>	这个数据列的数据类型; 可取值包括: <i>FIELD_TYPE_BLOB</i> <i>FIELD_TYPE_DATE</i> <i>FIELD_TYPE_DATETIME</i> <i>FIELD_TYPE_DECIMAL</i>

(续)

MYSQL_FIELD 结构的元素

<i>enum enum_field_types type;</i>	<i>FIELD_TYPE_DOUBLE</i> <i>FIELD_TYPE_ENUM</i> <i>FIELD_TYPE_FLOAT</i> <i>FIELD_TYPE_INT24</i> <i>FIELD_TYPE_LONG</i> <i>FIELD_TYPE_LONGLONG</i> <i>FIELD_TYPE_NULL</i> <i>FIELD_TYPE_SET</i> <i>FIELD_TYPE_SHORT</i> <i>FIELD_TYPE_STRING</i> <i>FIELD_TYPE_TIME</i> <i>FIELD_TYPE_TIMESTAMP</i> <i>FIELD_TYPE_TINY</i> <i>FIELD_TYPE_VAR_STRING</i> <i>FIELD_TYPE_YEAR</i>
<i>unsigned int length;</i>	程序员在定义这个数据列时为它设置的长度
<i>unsigned int max_length;</i>	这个数据列在查询结果中的最大长度。在使用 <i>mysql_use_result()</i> 函数取回查询结果的时候，这个值将永远是 0
<i>unsigned int flags;</i>	关于这个数据列的其他信息，包括： <i>AUTO_INCREMENT_FLAG</i> <i>BINARY_FLAG</i> <i>MULTIPLE_KEY_FLAG</i> <i>NOT_NULL_FLAG</i> <i>PRI_KEY_FLAG</i> <i>UNIQUE_KEY_FLAG</i> <i>UNSIGNED_FLAG</i> <i>ZEROFILL_FLAG</i>
<i>unsigned int decimals;</i>	<i>DECIMAL</i> 数据列的小数部分位数（即： <i>DECIMAL(10, 5)</i> 中的 5）

与预处理语句有关的数据结构**MYSQL_BIND 结构的元素**

<i>enum enum_field_types buffer_type;</i>	参数的数据类型，可取值如上表所示
<i>void *buffer;</i>	指针，指向用来传递参数值的缓冲区变量
<i>unsigned long buffer_length;</i>	缓冲区的最大长度（如果参数是字符串/BLOB 的话）
<i>unsigned long *length;</i>	参数值的实际长度（如果参数是字符串/BLOB 的话）
<i>my_bool *is_null;</i>	测试参数值是不是 <i>NULL</i> 。这个测试不是通过对缓冲区变量的内容直接进行比较而得到结论的。警告：直接读取 <i>is_null</i> 没有实际意义，它只是一个指针，我们需要的信息（“是 <i>NULL</i> ”或者“不是 <i>NULL</i> ”）存放在这个指针指向的变量里
<i>my_bool is_unsigned;</i>	是否需要把参数值解释为无符号整数（如果参数是整数数据类型）
<i>my_bool error;</i>	传递参数值时是否发生了错误（例如：参数值超出了缓冲区的最大长度）

下面这个表格汇总了 *MYSQL_BIND* 结构里的 *buffer_type* 变量的可取值。这个表格的第一列是枚举集合 *enum_field_types* 里的各个元素，第二列是与之对应的 MySQL 数据类型和最匹配的 C 语言数据类型。

enum_field_types 集合 (设置)

<i>MYSQL_TYPE_TINY</i>	MySQL 数据类型: <i>TINYINT</i>	C 语言数据类型: <i>char</i>
<i>MYSQL_TYPE_SHORT</i>	<i>SMALLINT</i>	<i>short int</i>
<i>MYSQL_TYPE_LONG</i>	<i>INT</i>	<i>int</i>
<i>MYSQL_TYPE_LONGLONG</i>	<i>BIGINT</i>	<i>long long int</i>
<i>MYSQL_TYPE_FLOAT</i>	<i>FLOAT</i>	<i>float</i>
<i>MYSQL_TYPE_DOUBLE</i>	<i>DOUBLE</i>	<i>double</i>
<i>MYSQL_TYPE_TIME</i>	<i>TIME</i>	<i>MYSQL_TIME</i>
<i>MYSQL_TYPE_DATE</i>	<i>DATA</i>	<i>MYSQL_TIME</i>
<i>MYSQL_TYPE_DATETIME</i>	<i>DATETIME</i>	<i>MYSQL_TIME</i>
<i>MYSQL_TYPE_TIMESTAMP</i>	<i>TIMESTAMP</i>	<i>MYSQL_TIME</i>
<i>MYSQL_TYPE_STRING</i>	<i>CHAR</i>	<i>char *</i>
<i>MYSQL_TYPE_VAR_STRING</i>	<i>VARCHAR</i>	<i>char *</i>
<i>MYSQL_TYPE_TINY_BLOB</i>	<i>TINY_BLOBS</i>	<i>char *</i>
<i>MYSQL_TYPE_BLOB</i>	<i>BLOBS</i>	<i>char *</i>
<i>MYSQL_TYPE_MEDIUM_BLOB</i>	<i>MEDIUM_BLOBS</i>	<i>char *</i>
<i>MYSQL_TYPE_LONG_BLOB</i>	<i>LONG_BLOBS</i>	<i>char *</i>

MYSQL_TIME 结构的元素

<i>unsigned int year;</i>	年
<i>unsigned int month;</i>	月
<i>unsigned int day;</i>	日
<i>unsigned int hour;</i>	小时
<i>unsigned int minute;</i>	分钟
<i>unsigned int second;</i>	秒
<i>my_bool neg;</i>	一个布尔值: 是否允许使用负数值来表示时间 (例如: <i>TIME_DIFF()</i> 函数的时间差计算结果)
<i>unsigned long second_part;</i>	微秒: 但 MySQL 5.0 版本目前还不能使用微秒

23.6.2 连接与管理**与 MySQL 服务器建立连接**

<i>MYSQL *conn;</i>	对名为 <i>MYSQL</i> 的 C 语言数据结构进行初始化
<i>conn = mysql_init(NULL);</i>	
<i>mysql_options</i> (<i>conn, option, "value"</i>);	设置各种必要的连接选项; <i>option</i> 参数可以是以下可取值之一: <i>MYSQL_OPT_CONNECT_TIMEOUT</i> <i>MYSQL_OPT_LOCAL_INFILE</i> <i>MYSQL_OPT_NAMED_PIPE</i> <i>MYSQL_INIT_COMMAND</i> <i>MYSQL_READ_DEFAULT_FILE</i> <i>MYSQL_READ_DEFAULT_GROUP</i>
	MySQL 在线文档 (http://dev.mysql.com/doc/mysql/en/mysql-options.html) 还列出了许多其他的选项。在这些选项当中, 有些还需要用 " <i>value</i> " 语法给出一个设置值。如果需要设置多个连接选项, 就必须多次调用这个函数来进行设置 (为每个选项调用一次)。 <i>mysql_options()</i> 函数必须在 <i>mysql_real_connect()</i> 函数前执行

(续)

与 MySQL 服务器建立连接

<code>mysql_real_connect(conn, "hostname", "username", "password", "dbname", portnum, "socketname", flags);</code>	尝试与 MySQL 服务器建立连接；如果出错，则返回 <code>NULL</code> 。 <code>flags</code> 参数可以是以下常数的任意组合： <code>CLIENT_COMPRESS</code> <code>CLIENT_FOUND_ROWS</code> <code>CLIENT_IGNORE_SPACE</code> <code>CLIENT_INTERACTIVE</code> <code>CLIENT_LOCAL_FILES</code> <code>CLIENT_MULTI_STATEMENTS</code> <code>CLIENT_MULTI_RESULTS</code> <code>CLIENT_NO_SCHEMA</code> <code>CLIENT_ODBC</code> <code>CLIENT_SSL</code>
<code>mysql_set_server_option(conn, option);</code>	为了与 MySQL 服务器建立一条长期连接而改变服务器选项，但目前只有以下两种服务器选项可供选用： <code>MYSQL_OPTION_MULTI_STATEMENT_ON</code> <code>MYSQL_OPTION_MULTI_STATEMENT_OFF</code>
<code>mysql_change_user(conn, "username", "password", "dbname");</code>	改变一条现有连接上的用户名和默认数据库
<code>mysql_change_db(conn, "dbname");</code>	改变默认数据库；当前用户必须有权访问新默认数据库才能调用成功
<code>mysql_ping(conn);</code>	测试这条连接是否可用；如果不可用，则重新创建这条连接；如果这条连接仍然可用，则返回 0
<code>mysql_close(conn);</code>	关闭这条连接

获取关于当前连接的信息

<code>mysql_characterset_name(conn);</code>	返回一个字符串：当前连接所使用的字符集
<code>mysql_get_client_info();</code>	返回一个字符串：客户库的版本信息（例如：“5.0.2”）
<code>mysql_get_server_info(conn);</code>	返回一个字符串：服务器的版本信息（例如：“5.0.2-alpha-standard”）
<code>mysql_get_host_info(conn);</code>	返回一个字符串
<code>mysql_get_proto_info(conn);</code>	返回一个 <code>unsigned int</code> 整数，连接协议的版本号（例如：10）
<code>mysql_info(conn);</code>	返回一个字符串：最近一条 <code>INSERT</code> 、 <code>UPDATE</code> 、 <code>LOAD DATA</code> 或 <code>ALTER TABLE</code> 命令的执行情况（例如：“Rows matched: 3 Changed: 3 Warnings: 0”）
<code>mysql_stat(conn);</code>	返回一个字符串：服务器状态信息（线程的个数、处于打开状态的数据表的个数等）
<code>mysql_thread_id(conn);</code>	返回一个 <code>unsigned long</code> 整数，当前连接在 MySQL 服务器上的线程号

管理用途的函数

<code>mysql_kill(conn, n);</code>	结束编号为 <code>n</code> 的线程（当前用户必须具备 <code>Process</code> 权限才能调用成功）
<code>mysql_shutdown(conn);</code>	关闭 MySQL 服务器（当前用户必须具备 <code>Execute</code> 权限才能调用成功）

出错处理

<code>mysql_errno(conn);</code>	如果刚才执行 SQL 命令时发生错误，返回一个出错代码（ <i>unsigned int</i> 整数）；如果没有发生错误，返回 0
<code>mysql_error(conn);</code>	如果刚才执行 SQL 命令时发生错误，返回一个出错消息字符串；如果没有发生错误，返回一个空字符串（""）
<code>mysql_warning_count(conn);</code>	返回刚才执行的 SQL 命令引起的警告次数。如果想读取警告消息的文本，必须执行 SQL 命令 <i>SHOW WARNINGS</i>

23.6.3 执行 SQL 命令及处理 SELECT 查询结果**执行 SQL 命令**

<code>mysql_query(conn, "SELECT ...");</code>	执行这条 SQL 命令，如果这条命令在没有触发任何错误的情况下被服务器接受了，则返回 0
<code>mysql_real_query(conn, "SELECT ... ", len);</code>	类似于 <code>mysql_query()</code> 函数，但这个函数允许 SQL 命令里包含零值字节（比如在存储 <i>BLOB</i> 数据的时候）。注意，必须明确地通过 <i>len</i> 参数给出 SQL 命令字符串的长度
<code>mysql_affected_rows(conn);</code>	返回一个 <i>my_ulonglong</i> 类型的整数： <i>INSERT</i> 、 <i>UPDATE</i> 或 <i>DELETE</i> 命令改变的数据记录的总数。这个函数不能用来确定 <i>SELECT</i> 查询结果里的记录总数
<code>mysql_insert_id(conn);</code>	返回一个 <i>my_ulonglong</i> 类型的整数：MySQL 服务器为 <i>INSERT</i> 命令最新插入的那条记录自动生成的 <i>AUTO_INCREMENT</i> 编号值

处理 SELECT 查询结果

<code>result = mysql_store_result(conn);</code>	把所有的结果记录一次性地全部从服务器传输到客户端并把它们保存在一个 <i>MYSQL_RES</i> 结构里
<code>result = mysql_use_result(conn);</code>	<code>mysql_store_result()</code> 函数的一种变体：结果记录将缓存在服务器上，只在必要时才把一条结果记录传输到客户端
<code>mysql_num_fields(result);</code>	返回查询结果中的记录总数
<code>row = mysql_fetch_row(result);</code>	把下一条结果记录传输到一个 <i>MYSQL_ROW</i> 结构里。如果没有记录可供读取（比如已经到达最后一条结果记录的时候），这个函数将返回 <i>NULL</i>
<code>row[n];</code>	返回一个以零值字节结尾的字符串：当前结果记录的第 <i>n</i> 个字段（数据列）的内容。注意， <code>row[n]</code> 可能包含着 <i>NULL</i> 值。如果正在处理的是一些本身可能包含着零值字节的二进制数据，就必须提前调用 <code>mysql_fetch_lengths()</code> 函数来确定当前记录各个结果字段的长度
<code>mysql_free_result(result);</code>	释放 <i>MYSQL_RES</i> 结构 <i>result</i> 。如果当初使用的是 <code>mysql_use_result()</code> 函数，本函数将释放那些缓存在服务器上的结果记录

处理 SELECT 查询结果：关于数据列和字段的元信息

<code>mysql_fetch_lengths(result);</code>	返回一个 <i>unsigned long</i> 数组，这个数组的各个元素分别是当前结果记录的各个字段的字符长度（把 <code>row[n]</code> 视为一个字符串）
<code>field = mysql_fetch_field(result);</code>	返回一个 <i>MYSQL_FIELD</i> 结构：结果字段（数据列）的数据类型。第一次调用这个函数返回的是关于第 1 个结果数据列的元信息，第二次调用返回的是关于第 2 个结果数据列的元信息，依此类推。如果已经到达最后一个结果数据列，这个函数将返回 <i>NULL</i> 。对 <i>MYSQL_FIELD</i> 结构的描述见前面的表格

(续)

处理 SELECT 查询结果：关于数据列和字段的元信息

<code>field=mysql_fetch_field_direct(result, n);</code>	直接返回关于第 <i>n</i> 个数据列的元信息（第 1 个数据列对应 <i>n=0</i> ）
<code>mysql_fetch_fields(result);</code>	把关于所有数据列的元信息返回为一个 <i>MYSQL_FIELD</i> 类型的数组

下面这个表格里的函数只能与 *mysql_store_result()* 函数配合使用（不能与 *mysql_use_result()* 函数配合使用）。

处理 SELECT 查询结果：只能与 *mysql_store_result()* 函数配合使用的部分函数

<code>mysql_num_rows(result);</code>	返回查询结果里的记录总数
<code>mysql_data_seek(result, n);</code>	把数据行光标直接定位到第 <i>n</i> 条结果记录（第一条结果记录对应着 <i>n=0</i> ）。接下来必须用 <i>mysql_fetch_row()</i> 函数来重新读取数据
<code>roffset = mysql_row_tell(result);</code>	返回一个指针，指针指向当前结果记录（这个指针的作用类似于一个书签）
<code>mysql_row_seek(result, roffset);</code>	把数据行光标直接定位到查询结果中的某个特定位置。接下来必须用 <i>mysql_fetch_row()</i> 函数来重新读取数据。 <i>roffset</i> 参数是相对于当前结果记录的偏移量，它必须提前用 <i>mysql_row_tell()</i> 函数加以确定

MULTI_STATEMENT 和 MULTI_RESULT 模式

<code>mysql_real_connect(..., CLIENT_MULTI_STATEMENTS);</code>	在建立连接的同时激活这两种 <i>MULTI_XXX</i> 模式
<code>mysql_set_server_option(conn, MYSQL_OPTION_MULTI_STATEMENTS_ON);</code>	在连接已经建立之后激活这两种 <i>MULTI_XXX</i> 模式
<code>mysql_query("command1;command2;command3");</code>	一次执行多条 SQL 命令，这些命令必须用分号隔开。第一条命令的结果可以像平常那样（用 <i>mysql_affected_row()</i> 、 <i>mysql_store_result()</i> 等函数）来处理
<code>n = mysql_more_results(conn);</code>	测试是否还有更多的查询结果（如果有，则 <i>n=1</i> ）
<code>n = mysql_next_result(conn);</code>	把下一组查询结果选定为当前查询结果，接下来就可以像平常那样使用各有关函数对这组查询结果进行处理了。这个函数的返回值有以下几种情况： <i>n=0</i> , OK, 有下一组结果； <i>n=1</i> , OK, 但没有下一组结果； <i>n>0</i> , 出错代码

辅助函数

<code>n = mysql_real_escape_string(conn, dest, src, srclen);</code>	把字符串 <i>src</i> 复制为字符串 <i>dest</i> 并把特殊字符替换为“\”转义序列（“\0”、“\b”、“\t”、“\n”、“\v”等）， <i>dest</i> 字符串以零值字节结尾。 <i>srclen</i> 参数负责给出 <i>src</i> 字符串里的字符个数。注意， <i>dest</i> 参数必须被声明为一个长度适当的字符串——为了复制一个完全由特殊字符构成的字符串， <i>dest</i> 的长度至少需要达到 <i>srclen*2+1</i> 个字符。这个函数的返回值是 <i>dest</i> 字符串里的字符个数（最末尾的零值字节不计算在内）。比如说，如果 <i>src</i> 字符串是 <i>O'Reilly</i> ，那么 <i>dest</i> 字符串就将是 <i>O\\'Reilly</i> ，而这个函数的返回值 <i>n</i> 将等于 9
<code>n = mysql_hex_string(to, from, len);</code>	把字符串 <i>from</i> 里的每一个字符转换为十六进制编码再写入缓冲区 <i>to</i> ， <i>to</i> 里的结果字符串以零值字节结尾。注意， <i>to</i> 字符串没有 <i>0x</i> 前缀，但这个前缀却是 MySQL 识别十六进制编码的字符串的标志，这意味着在构造 SQL 命令字符串的时候必须亲自动手加上这些必要的 <i>0x</i> 前缀

(续)

辅助函数

<code>n = mysql_hex_string(to, from, len);</code>	<code>len</code> 参数是 <code>from</code> 字符串的字符长度，缓冲区 <code>to</code> 的长度不得小于 <code>len*2+1</code> 个字符 这个函数的返回值是结果字符串的长度（最末尾的零值字节不计算在内）
<code>destpt = strmov(dest, src);</code>	类似于 <code>strcpy()</code> 函数，这个函数也是把字符串 <code>src</code> 复制为字符串 <code>dest</code> ，但这个函数的返回值指向 <code>dest</code> 字符串的末尾。因此，这个函数非常适合用来把多个字符串拼凑为一个大字符串。注意，要想使用 <code>strmov()</code> 函数，就必须在程序源代码文件里在 <code>mysql.h</code> 文件之前先导入 <code>my_global.h</code> 和 <code>m_string.h</code> 两个文件

23.6.4 预处理语句

预处理语句的构造和执行

<code>stmt = mysql_stmt_init(conn);</code>	返回一个 <code>MYSQL_STMT</code> 结构
<code>mysql_stmt_prepare(stmt, sqlcmd, strlen(sqlcmd));</code>	对 <code>MYSQL_STMT</code> 结构进行初始化，其中 <code>sqlcmd</code> 参数是将要执行的 SQL 命令字符串（数据类型是 <code>char[]</code> ）
<code>mysql_stmt_bind_param(stmt, bind);</code>	为 SQL 命令声明各有关参数。 <code>bind</code> 是一个 <code>MYSQL_BIND</code> 数组，该数组的每一个元素分别对应着 SQL 命令的一个参数。 <u>这些参数的值将通过这个 bind 数组里的各有关缓冲区变量来传递</u>
<code>mysql_stmt_execute(stmt);</code>	执行 SQL 命令 ¹
<code>mysql_stmt_close(stmt);</code>	释放 <code>MYSQL_STMT</code> 结构占用的内存

对通过预处理语句而获得的 SELECT 查询结果进行处理

<code>stmt = mysql_stmt_init(conn);</code>	返回一个 <code>MYSQL_STMT</code> 结构
<code>mysql_stmt_prepare(stmt, sqlcmd, strlen(sqlcmd));</code>	对 <code>MYSQL_STMT</code> 结构进行初始化
<code>mysql_stmt_bind_result(stmt, bind);</code>	对 <code>SELECT</code> 查询结果中的字段（数据列）做出声明。 <code>bind</code> 是一个 <code>MYSQL_BIND</code> 数组，该数组的每一个元素分别对应着 <code>SELECT</code> 查询结果中的一个字段。 <u>这些字段的值将通过这个 bind 数组里的各有关缓冲区变量来传递</u>
<code>mysql_stmt_execute(stmt);</code>	执行 SQL 命令
<code>mysql_stmt_store_result(stmt);</code>	把查询结果全部传输到客户端并把它们存放在一个缓冲区里。注意：调用这个函数并不是一个必不可少的步骤。如果没有调用它，查询结果将缓存在服务器端直到用 <code>mysql_stmt_fetch()</code> 函数把最后一条结果记录取走为止。注意，如果没有调用这个函数，就不能使用 <code>mysql_stmt_data_seek()</code> 函数在查询结果里快速移动，也不能在处理完所有的结果记录之前用 <code>mysql_stmt_num_rows()</code> 统计出它们的总数
<code>n = mysql_stmt_num_rows(stmt);</code>	返回结果记录的总数；返回值 <code>n</code> 的数据类型是 <code>my_ulonglong</code>
<code>mysql_stmt_fetch(stmt);</code>	把下一条结果记录读入那个负责传递 <code>SELECT</code> 查询结果的 <code>MYSQL_BIND</code> 数组中的各有关变量。如果已经到达最后一条结果记录或者 <code>SELECT</code> 命令根本没有返回任何结果记录，这个函数将返回 0
<code>mysql_stmt_data_seek(stmt, n);</code>	把第 <code>n</code> 条结果记录选定为当前结果记录。下一次 <code>mysql_stmt_fetch()</code> 函数调用将把这条记录读入各有关变量
<code>result = mysql_stmt_result_metadata(stmt);</code>	返回关于 <code>SELECT</code> 查询结果的元信息。详见前面对 <code>mysql_num_fields()</code> 、 <code>mysql_fetch_fields()</code> 等函数的说明

1. 原文这里显然是把“传递参数”和“传递 `SELECT` 查询结果”两个概念给混在一起了——这需要两个 `MYSQL_BIND` 数组，但原文没有体现出这一点！这是概念性错误。为了不误导读者，我们省略了这句话，但在上一栏和下面的表格里各增加了一句话（见下划线部分）。——译者注

Part 6

第六部分

附录

本部分内容

- 附录 A 术语解释
- 附录 B 本书的配套示例文件
- 附录 C 参考书目

附录 A**术语解释**

在 这篇附录里，我们将对数据库以及相关领域里的一些最重要的概念和术语进行简单的解释。

字符集 (character set): 字符集是用来表示某一种或某几种人类语言中的字符的编码方案。在各种字符串字符集当中，ASCII 字符集是最“古老的”之一。在 ASCII 字符集里，字母 A 被编码为整数值 65。

客户 (client): 狹义的“客户”特指供人们用来访问某项服务的程序。广义的“客户”还包括运行各种客户程序的计算机。具体到数据库领域，客户就是人们用来访问某个数据库的程序。数据库本身由另外一种程序，由所谓的“数据库服务器”负责管理。

客户/服务器体系 (Client/Server Architecture): 绝大多数现代的数据库系统都采用了客户/服务器体系，即由一个服务器程序负责集中管理数据库里的数据和执行 SQL 命令，人们使用各种客户程序通过网络去访问这个服务器。与客户/服务器体系相对的是以 dBase 和 Access 为代表的文件服务器型数据库系统。在文件服务器型数据库系统里，所有的客户通过一个公共文件系统去直接访问数据库文件。（文件服务器型数据库系统的主要缺点是效率相对比较低下，这一点在有许多客户试图同时编辑同一批数据的时候体现得尤其明显。）

集群 (cluster): 在信息理论里，集群通常是指为了完成同一任务而被联成一个网络的多台计算机，而所谓的数据库集群就是用多台计算机共同管理和提供一种数据库服务的数据库系统，这种数据库系统一般都比较庞大，而且往往在速度和信息安全性方面有着比较高的要求。

光标 (cursor): 在 MySQL 数据库系统里，光标的含义相当于一个结果记录读写指针，它使人们可以对查询结果一步一步地做出处理（光标的这种用途在各种存储过程里最为常见）。在 ADO 函数库里，与光标有关的各种属性决定着 ADO 程序都能对 *RecordSet* 对象里的记录进行哪些处理。

数据字典 (data dictionary): MySQL 数据字典使程序员可以使用 *SELECT* 查询命令获得关于数据库和数据表结构的信息以及其他一些关于 MySQL 服务器的管理用数据。从 MySQL 5.0 版本开始，MySQL 提供了一些虚拟的 *INFORMATION_SCHEMA* 数据表来实现各种数据字典功能。

数据记录 (data record): 数据记录就是数据表里的数据行，它经常被简称为“记录”。

数据库 (database): 从广义上讲，人们为了某种目的而收集的任何一批数据（通常还经过一些排序）都可以被称为一个数据库。具体到 MySQL 数据库，它们是一些由一个或多个数据表构成的数据集合体。在日常工作中，人们还使用经常“数据库”这个名词来称呼一个完备的数据库系统，即一个用来管理有关数据的程序。在 MySQL 里，这个程序被称为数据库服务器（MySQL 服务器）。

数据库服务器 (database server): 数据库服务器是一种通过网络向人们提供各种数据库访问服务的程序。数据库服务器的任务是管理数据库里的数据和执行 SQL 命令。

派生数据表 (derived table): 派生数据表特指为了保存 *SELECT* 命令的查询结果而由数据库服务器临时创建出来的数据表。从这个意义上讲，派生数据表不过是嵌套在 *SELECT* 命令里的子查询的一种变体而已。

域名 (domain name): 域名代表着一个完整的网络。比如说，对于一个由 3 台名为 *mars.sol*、*uranus.sol* 和 *jupiter.sol* 的计算机构成的本地网络来说，*mars*、*uranus* 和 *jupiter* 是计算机的名字（主机名），*sol* 是这个网络的域名。

域名服务器 (Domain Name Server, DNS): 在网络的内部，计算机需要使用数字形式的 IP 地址（如 192.168.0.27）来进行通信，但这种数字对人类用户来说很难记忆和使用——人们更善于记忆由主机名和域名组合而成的计算机名（如 *uranus.sol*）。域名服务器的任务就是在 IP 地址和计算机名之间进行转换：把用户输入的计算机名转换为相应的 IP 地址以便计算机能够进行通信，把 IP 地址转换计算机名以便用户能够了解通信情况和查看通信内容。（有些小型网络可能没有 DNS 服务器。在这种网络里，IP 地址和计算机名之间的转换是通过检索一个静态的表格来实现的。在 UNIX/Linux 环境下，这个表格通常保存在/etc/hosts 文件里。）

外键 (foreign key): 每一个外键都是一个独一无二的值（通常是一个整数值），这个值指向另一个数据表里的某个特定的字段（数据列）。也就是说，每一个外键都会在两个数据表之间创建一个链接（关系）。

全文索引 (full-text index): 全文索引是一种专门为了在大段文本里搜索以任意顺序出现的一个或多个单词而创建的索引。这种搜索就是所谓的全文搜索（full-text search），而全文索引可以大大提高全文搜索的效率。

全局程序集缓存区 (Global Assembly Cache, GAC): 这是微软数据库产品的一个独有概念，它相当于.NET 程序会用到的所有*.dll 文件的注册表。与 Windows 操作系统的注册表（directory）¹不同，在 GAC 里可以同时注册同一个函数库的多个版本而不会导致冲突。

主机名 (host name): 主机名是人们为了区分和标识某个网络里的各台计算机而给它们起的名字。在实际工作中，人们往往把主机名和计算机名看做是同义词，但严格地讲，主机名只是计算机名的组成部分之一。比如说，在计算机名 *jupiter.sol* 里，*jupiter* 是主机名，*sol* 是域名。

热备份 (hot backup): “热备份”的“热”字指的是在备份过程中，数据库服务器的运行几乎不受任何干扰。（在普通的备份操作中，数据库服务器不能对被备份数据做任何修改。）就目前而言，对 MySQL 数据库进行热备份只能使用商业化的第三方软件来进行，而且只能对 InnoDB 数据表进行热备份。

头文件 (include file): 头文件是程序代码的一部分（通常由诸如 *include* 之类的指令导入，所以才叫做 *include file*）。各种程序设计语言对头文件的处理方式不外乎两种：一种是在编译/链接期间读入头文件，比如 C 语言；另一种是在程序执行时才读入头文件，如 PHP。

索引 (index): 在数据库领域，“索引”及它的同义词“关键字”（key）特指根据数据表的内容而生成的一个排序清单。索引的基本用途是加快对数据记录的访问速度：即不必搜索整个数据表，只要搜索它的某个索引就可以快速确定某个特定数据记录的位置并把它直接读取出来。数据库意义上的索引与人们日常生活中的图书目录或索引有着同样的目的和效果。

InnoDB 数据表: MySQL 数据表有好几种硬盘存储格式，其中最常见的是 MyISAM 和 InnoDB 格式。InnoDB 数据表比 MyISAM 数据表的功能更多，比如事务和外键约束条件等。InnoDB 得名于 Innobase 公司，其中的“DB”是“database”的简写。

ISP (Internet Service Provider, 因特网服务提供商): ISP 是为网站提供技术支持和托管服务的公司，它们的顾客是那些希望在因特网上建立一个网站但自己没有永久性或高速的因特网连接的公司或个人。在挑选 ISP 的时候，一定要看它能否提供所需要的软件和服务支持。比如说，如果想参照本书建立一个网站，就应该找一家支持 Apache、PHP、MySQL 等的 ISP。当然，如果打算使用微软产品来建立一个这样的网站，就应该找一家提供微软产品的 ISP。

1. 原文这里的“directory”大概是机器翻译的结果。这里不能把它译做“目录”！——译者注

关键字 (key): 在数据库领域里，关键字就是数据表的排序索引。参见它的同义词条目“索引”。

日志 (log/logging): 日志的动词含义是把 SQL 命令记入一个文件，名词含义是用来记载 SQL 命令的文件。日志的用途是把数据库自上次备份之后发生的一切修改记载下来，以便在发生灾难事件之后还可以把数据库重新创建出来。此外，在 MySQL 数据库系统里，日志也是启用镜像机制的前提条件。

MyISAM 数据表: MySQL 数据表有好几种硬盘存储格式，其中最重要的是 MyISAM 格式。MyISAM 是“indexed sequential access method”（基于索引的顺序访问方法）的缩写，意思是在索引的帮助下访问某个文件里的数据记录。另一种同样重要的 MySQL 数据表格式是 InnoDB。

命名管道 (named pipe): 命名管道是 Windows 2000/XP 环境中的一种程序间通信机制，这种通信机制按照“先进先出”(first in, first out; FIFO) 的原则来交换数据。

范式 (normal form): 数据库由一个或多个数据表构成，所以设计一个数据库就是对其中的数据表进行设计，以避免数据冗余，这个目的需要通过一系列规则来保证，满足第 n 组规则的数据库称为“第 n 范式”的。

规范化 (normalization): 这个术语特指为了避免数据冗余而对数据库设计方案进行的优化工作。进行这种优化的目的通常是为了达到第 3 范式。

PHP: PHP 是一种用来编写动态网站程序的脚本语言，PHP 程序的文件名是*.php。*.php 文件由 Web 服务器调用 PHP 解释器在服务器端以解释方式执行，最终生成的 HTML 结果文档将由 PHP 解释器返回给 Web 服务器、再由 Web 服务器发送给客户端的 Web 浏览器。

端口 (port): 使用 TCP/IP 协议传输的数据包永远是发往某个特定目标端口的，所以数据包的目标地址里必须包含一个端口号。常见的服务都有默认的通信端口号，所以往往可以根据端口号来判断出这是哪种类型的通信。比如说，Web 浏览器和服务器之间的数据交换默认使用 80 号端口来进行，MySQL 则默认使用 3306 号来进行通信。

预处理语句: 预处理语句可以大大提高使用不同参数执行同一条 SQL 命令的效率：SQL 命令本身只向服务器发送一次，以后的执行只需发送变化了的参数即可。这不仅减少了数据传输量，还节省了服务器对 SQL 命令进行分析的工作量。

主索引/主键 (primary index / primary key): 主索引的用途是把数据包里的每一条数据记录独一无二地标识出来。主键字段几乎总是由一个整数数据列充当，在插入新记录时，数据库服务器会为这个数据列自动生成一个新的、唯一的键值。

权限 (privilege): 在 MySQL 数据库系统里，权限就是执行一项数据库操作（比如读取或修改某个特定数据包的数据等）的权利。

查询 (query): 在数据库语言里，查询的名词含义通常特指一条 SELECT 命令，动词含义则是执行一条 SELECT 命令。这种命令的语法是 SQL 语言 (structured query language, 结构化查询语言) 的一个组成部分。SQL 规定了应该如何构造将被发送给数据库服务器去执行的命令。

查询 (命令) 缓存区 (query cache): 在许多数据库应用程序里，同样的查询命令往往会被多次执行。为了加快这种查询命令的执行速度。MySQL 从 4.0 版本开始在其内部设立了一个查询缓存区来临时存放各种查询命令和它们的执行结果。这样一来，当用户再次执行同样的查询命令时，MySQL 服务器就可以更加迅速地对它们做出响应。注意，临时存放在这个缓存区里的查询命令及其查询结果会在用户对底层的数据表做出修改时被清除。

引用一致性 (referential integrity): 引用一致性可以保证在两个或更多个关联数据表之间不存在无效的链接。比如说，如果某个图书数据库符合引用一致性的要求，就不可能存在其出版公司在数据库里不存在的图书。反过来讲，如果引用一致性遭到了破坏，就可能出现“图书”数据表里的出版公司编号在“出版公司”数据表里根本不存在的问题。

许多数据库系统会在数据发生变化时自动检查引用一致性是否遭到了破坏，这种检查是通过外键约束条件（也叫引用一致性规则）来进行的。MySQL 目前只支持 InnoDB 数据表上的外键约束条件。

关系 (relation): 关系化数据库的主要特征就是数据库里的多个数据表可以根据它们的内容相互关联。这种关联就是所谓的“关系”。数据表之间的关联关系由 SQL 命令里的 *JOIN* 操作符或 *WHERE* 子句体现。

镜像机制 (replication): 通过镜像机制，对一个数据库的所有修改将在一台以上的数据库服务器上得到执行。这么做的效果是同一个数据库将存在于多台数据库服务器上。镜像机制的优点是可以提高信息的安全性（即使一台数据库服务器发生故障也不会影响数据库的使用）和减少用户的查询等待时间（在主控服务器上做出的修改也将反映在从属服务器上，所有的主控服务器和从属服务器都可以对 SQL 查询做出响应）。

数据库设计方案 (schema): 数据库设计方案对数据库的布局结构——也就是各有关数据表、它们的数据列和数据类型、索引、数据表之间的关系等——做出了描述。

服务器 (server): 狹义的“服务器”特指那些向其他程序（即所谓的“客户”）提供各种服务或数据的程序，广义的“服务器”还包括那些运行着各种服务器程序的计算机。MySQL 服务器（数据库服务器）和 Apache（Web 服务器）都是典型的服务器程序的例子。

套接字文件 (socket file): 套接字文件是 UNIX/Linux 环境下的一种程序间通信机制。套接字文件本身并不是一个真正的文件，它们不包含任何数据并且长度为 0。

UNIX/Linux 环境下的 MySQL 客户与 MySQL 服务器之间的本地连接是通过一个套接字文件发生的，这种通信比使用网络协议 TCP/IP 的效率更高。（Java 客户属于例外，它们不支持套接字文件。）这种用途的套接字文件名通常是 /var/lib/mysql/mysql.sock；这个文件名可以通过 MySQL 配置文件 /etc/my.cnf 加以设置。

存储过程 (stored procedure): 存储过程是由数据库服务器直接执行的一些程序代码。存储过程是用一种在 SQL 语言的基础上扩展了一些控制结构（循环、分支等）的程序设计语言编写的。MySQL 从 5.0 版本开始支持存储过程。

子查询 (subSELECT): 子查询特指嵌套在同一条查询命令里的多条 *SELECT* 子命令。MySQL 从 4.1 版本开始支持这种命令。

数据表 (table): 数据表是数据库的组成部分。数据表的属性由它的数据列或字段（名字和数据类型）以及索引来决定。数据表里的数据行也叫做数据记录（data record）。MySQL 服务器可以把数据表存储为多种格式（如 MyISAM、InnoDB 等）。

时间戳 (timestamp): 人们把数据类型是 *TIMESTAMP* 的数据列里的数据值叫做“时间戳”。在修改一条数据记录的时候，MySQL 服务器会自动地把当前日期和时间存入这种类型的数据列里。从这个意义上讲，时间戳就是数据记录最近一次发生修改的时间。

事务 (transaction): 事务是一组有着内在逻辑联系的 SQL 命令。支持事务的数据库系统要么确认同一个事务里的所有 SQL 命令，要么把它们当做整体全部放弃。也就是说，事务永远不会只完成一部分。事务可以大大提高数据安全性。

事务还可以提高效率，因为在执行多条 SQL 命令的过程中不必再使用 *LOCK* 命令锁定整个数据表。MySQL 目前只支持 InnoDB 数据表上的事务。

触发器 (trigger): 在数据库领域里，这个术语特指那些作为某种特定操作（比如 *UPDATE* 或 *INSERT* 命令）的结果而被自动执行的 SQL 代码（存储过程）。触发器的典型用途之一是保证对数据的修改符合有关的规则。MySQL 从 5.0 版本开始支持触发器。

Unicode: Unicode 是一种以 16 位整数来表示每一个字符的字符集。世界各国语言中的特殊字符几乎都可以用这种字符集来表示。MySQL 从 4.1 版本开始支持 Unicode 字符集。

Unicode 字符集有好几种变体，它们之间的主要区别在于存储 16 位整数时采用的字节顺序。比较常见的 Unicode 字符集有 UTF8（为了避免零值字节而使用 1 个到 3 个字节对字符进行编码）和 UTF16（也叫 UCS2，

每个字符固定使用 2 字节编码) 两种。

URL (Uniform Resource Locator, 统一资源定位器): 这个术语说的其实就是各种因特网地址 (比如: `http://www.kofler.cc/mysql` 或 `ftp://ftp.mysql.com`)。URL 地址里还可以包含其他一些信息, 如变量名或用户名等: `http://my.company/page.html?var=123`、`ftp://username.compamy/directory`。

视图 (view): 在数据库领域里, 视图特指那些按照某种要求用一个或多个数据表生成一个子数据表的 SQL 查询命令。视图可以简化对数据表的访问操作 (把用户感兴趣的数据行和数据列包含在视图内) 和提高信息的安全性 (把需要保密的数据列和数据行排除在视图外)。MySQL 从 5.0 版本开始支持视图。

Web 服务器 (Web server): Web 服务器是通过因特网提供 HTML 页面的程序。目前最为流行的 Web 服务器是 Apache (市场份额在 60% 左右)。