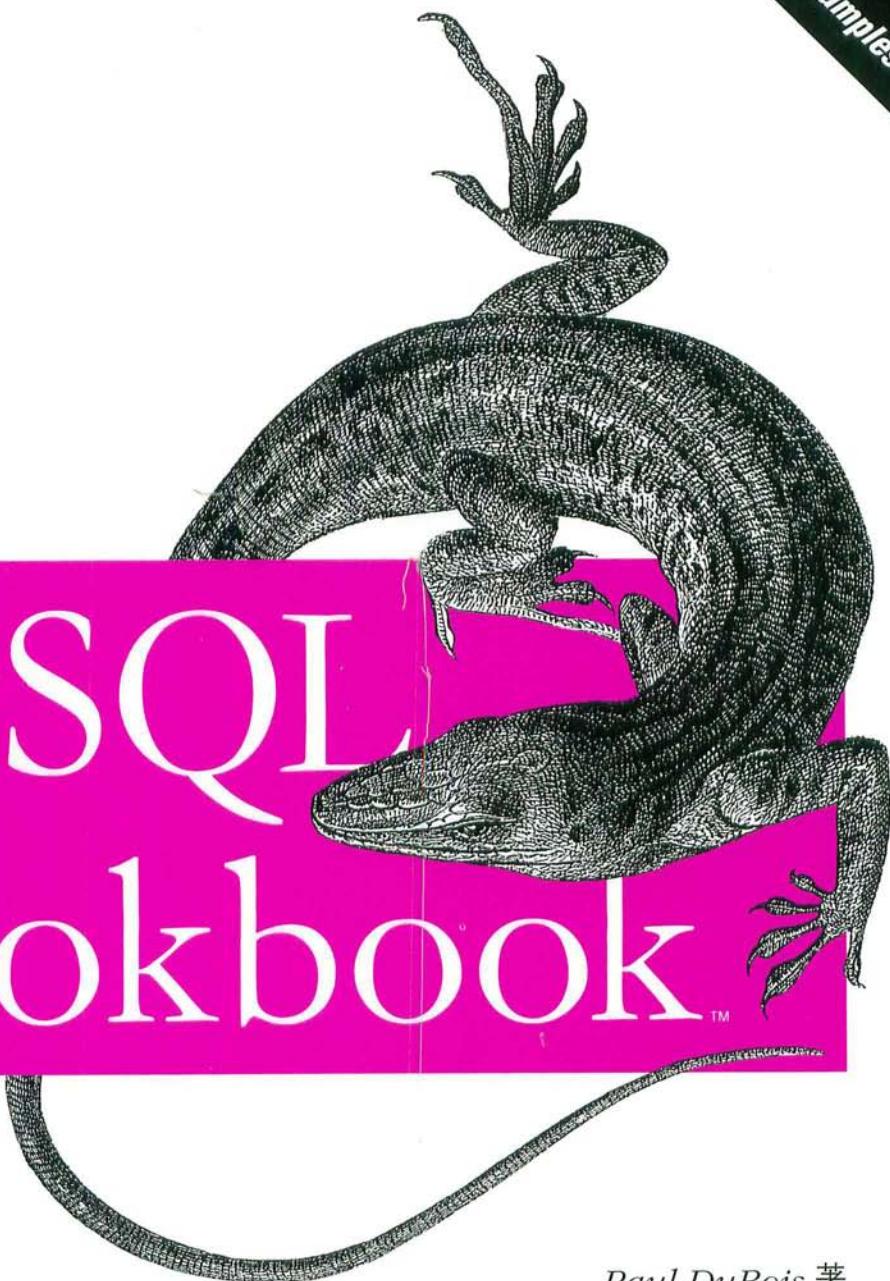


MySQL Cookbook
*Solutions and Examples for
Database Developers and DBAs*

第2版
Includes Ruby Examples

MySQL Cookbook

中文版



Paul DuBois 著
瀚海时光团队 译

O'REILLY®

 電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

O'REILLY®

MySQL Cookbook 第2版 中文版

MySQL Cookbook 2nd Edition

[美] Paul DuBois 著

瀚海时光团队 译

电子工业出版社

Publishing House of Electronics Industry

北京 · BEIJING
www.TopSage.com

内 容 简 介

本书为各个层次的、没有时间和精力来从头解决 MySQL 问题的用户提供了大量简练、精辟的代码段和可用的示例，每节都阐述了代码应该如何工作及原因所在。本书在目前仍然广为流行的 MySQL 4.1 的基础上加入了 MySQL 5.0 的内容及它强大的新特性。读者将掌握用 MySQL 客户端程序执行 SQL 查询的方法，以及通过 API 编写与 MySQL 服务器交互程序的方法。书中有大量使用 Perl、PHP、Python、Java 甚至 Ruby 来检索并显示数据的新示例，还增加了子查询、视图、存储过程、触发器和事件等内容。

本书适合于所有从事数据库技术开发的相关人员阅读，是 MySQL 开发人员案头必备之书。

978-0-596-52708-2 MySQL Cookbook 2nd Edition. Copyright ©2006 by O'Reilly Media, Inc.
Simplified Chinese edition, jointly published by O'Reilly Media, Inc. and Publishing House of
Electronics Industry, 2008. Authorized translation of the English edition, 2006 O'Reilly
Media, Inc., the owner of all rights to publish and sell the same. All rights reserved including the
rights of reproduction in whole or in part in any form.

本书中文简体版专有出版权由 O'Reilly Media, Inc. 授予电子工业出版社，未经许可，不得以任何方式复制或抄袭本书的任何部分。

版权贸易合同登记号 图字：01-2007-2460

图书在版编目（CIP）数据

MySQL Cookbook: 第 2 版: 中文版 / (美) 迪布瓦 (DuBois,P.) 著; 瀚海时光团队译. —
北京: 电子工业出版社, 2008.3

ISBN 978-7-121-05993-3

I. M… II. ①迪…②瀚… 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 (2008) 第 017138 号

责任编辑: 王继花

项目管理: 梁 晶

封面设计: Karen Montgomery 张 健

印 刷: 北京市天竺颖华印刷厂

装 订: 三河市金马印装有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 787×980 1/16 印张: 61.5 字数: 1458 千字

印 次: 2008 年 3 月第 1 次印刷

定 价: 128.00 元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888。

质量投诉请发邮件至 zlts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。服务热线: (010) 88258888。

目录

Table of contents

序言	1
第1章：使用mysql客户端程序	1
1.0 引言	1
1.1 建立 MySQL 用户账号	2
1.2 创建数据库和样表	4
1.3 启动和停止 mysql	6
1.4 使用可选项文件来指定连接参数	8
1.5 保护选项文件以阻止其他用户读取	10
1.6 混合使用命令行和选项文件参数	11
1.7 找不到 mysql 时该怎么做	12
1.8 发起 SQL 语句	13
1.9 取消一条部分输入的语句	14
1.10 重复和编辑 SQL 语句	15
1.11 自动完成数据库名和表名	16
1.12 让 mysql 从文件中读取语句	17
1.13 让 mysql 从其他程序读取语句	20
1.14 一行输入 SQL	21
1.15 使用拷贝粘贴作为 mysql 输入源	22
1.16 预防查询输出超出屏幕范围	22
1.17 发送查询输出到文件或程序	24
1.18 选择表格或制表符定界的查询输出格式	25
1.19 指定任意的输出列分隔符	26
1.20 生成 HTML 或 XML 输出	27
1.21 在查询输出中禁止列头部	29
1.22 使长输出行更具可读性	30
1.23 控制 mysql 的繁冗级别	31
1.24 记录交互式的 mysql 会话	32
1.25 以之前执行的语句创建 mysql 脚本	33
1.26 在 SQL 语句中使用用户自定义的变量	33
1.27 为查询输出行计数	36
1.28 将 mysql 用作计算器	37

1.29 在 Shell 脚本中使用 mysql	38
第 2 章：编写基于 MySQL 的程序.....	45
2.0 引言	45
2.1 连接、选择数据库及断开连接	50
2.2 查错	64
2.3 编写库文件	72
2.4 发起语句并检索结果	85
2.5 处理语句中的特殊字符和 NULL 值.....	100
2.6 处理标识符中特殊字符	112
2.7 识别结果集中的 NULL 值	113
2.8 获取连接参数的技术	117
2.9 结论和建议	129
第 3 章：从表中查询数据	131
3.0 引言	131
3.1 指定查询列/从指定列中查询	133
3.2 指定查询行	134
3.3 格式化显示查询结果	135
3.4 使用列别名来简化程序	138
3.5 合并多列来构建复合值	139
3.6 Where 表达式中的列别名	140
3.7 调试比较表达式	141
3.8 使查询结果唯一化	142
3.9 如何处理 NULL 值	143
3.10 在用户程序中使用 NULL 作为比较参数	145
3.11 结果集排序	146
3.12 使用视图来简化查询	148
3.13 多表查询	149
3.14 从查询结果集头或尾取出部分行	151
3.15 在结果集中间选取部分行	153
3.16 选择合适的 LIMIT 参数	155
3.17 当 LIMIT 需要“错误”的排列顺序时做什么	158
3.18 从表达式中计算 LIMIT 值	159
第 4 章：表管理	161
4.0 引言	161
4.1 克隆表	161
4.2 将查询结果保存到表中	162
4.3 使用临时表	165
4.4 检查或改变某个表的存储引擎	167
4.5 生成唯一的表名	168

第 5 章：与字符串共舞	171
5.0 引言	171
5.1 字符串属性	172
5.2 选择字符串的数据类型	175
5.3 正确设置客户端连接的字符集	178
5.4 串字母	179
5.5 检查一个字符串的字符集或字符排序	182
5.6 改变字符串的字符集或字符排序	183
5.7 更改字符串字母的大小写	185
5.8 更改字符串大小写失败的情况	186
5.9 控制字符串比较中的大小写敏感	188
5.10 使用 SQL 模式进行模式匹配	191
5.11 使用正则表达式进行模式匹配	194
5.12 模式匹配中的大小写问题	198
5.13 分割或者串联字符串	200
5.14 查询子串	203
5.15 使用 FULLTEXT 查询	203
5.16 用短语来进行 FULLTEXT 查询	208
5.17 要求或禁止 FULLTEXT 搜索单词	209
5.18 用 FULLTEXT 索引来执行词组查询	211
第 6 章：使用日期和时间	213
6.0 引言	213
6.1 选择合适的日期或者时间变量类型	214
6.2 修改 MySQL 中的日期格式	216
6.3 设置客户端时区	220
6.4 获取当前日期或时间	222
6.5 使用 TIMESTAMP 来跟踪行修改时间	223
6.6 从日期或者时间值中分解出各部分值	226
6.7 合成日期或者时间值	232
6.8 在时间数据类型和基本单位间进行转换	234
6.9 计算两个日期或时间之间的间隔	238
6.10 增加日期或时间值	243
6.11 计算年龄	248
6.12 将一个日期和时间值切换到另一个时区	253
6.13 找出每月的第一天，最后一天或者天数	254
6.14 通过子串替换来计算日期	257
6.15 计算某个日期为星期几	258
6.16 查出给定某周的某天的日期	259
6.17 执行闰年计算	262
6.18 接近但不是 ISO 格式的日期格式	265
6.19 将日期或时间当成数值	266
6.20 强制 MySQL 将字符串当作时间值	268

6.21 基于时间特性来查询行	269
第 7 章：排序查询结果	273
7.0 引言	273
7.1 使用 ORDER BY 命令排序查询结果	274
7.2 使用表达式排序	278
7.3 显示一组按照其他属性排序的值	280
7.4 字符串排序的大小写区分控制	283
7.5 基于日期的排序	286
7.6 按日历排序	288
7.7 按周历排序	290
7.8 按时钟排序	291
7.9 按数据列的子串排序	292
7.10 按固定长度的子串排序	293
7.11 按可变长度的子串排序	295
7.12 按域名顺序排列主机名	300
7.13 按照数字顺序排序点分式 IP 地址	302
7.14 将数值移动到排序结果的头部或尾部	304
7.15 按照用户定义排序	308
7.16 排序枚举数值	309
第 8 章：生成摘要	313
8.0 引言	313
8.1 使用 COUNT 函数生成摘要	315
8.2 使用 MIN()和 MAX()函数生成摘要	318
8.3 使用 SUM()和 AVG()函数生成摘要	319
8.4 使用 DISTINCT 函数消除重复	321
8.5 查找数值相关的最大值和最小值	323
8.6 控制 MIN()函数和 MAX()函数的字符串大小写区分	325
8.7 将摘要划分为子群	327
8.8 摘要与空值	330
8.9 使用确定的特性选择组群	333
8.10 使用计数确定数值是否唯一	334
8.11 使用表达式结果分组	335
8.12 分类无类别数据	336
8.13 控制摘要显示顺序	340
8.14 查找最小或最大的摘要数值	342
8.15 基于日期的摘要	344
8.16 同时使用每一组的摘要和全体的摘要	346
8.17 生成包括摘要和列表的报告	349

第 9 章：获取和使用元数据	353
9.0 引言	353
9.1 获取受语句影响的数据行数目	355
9.2 获取设置元数据的结果	357
9.3 确定一条语句是否生成了结果集	367
9.4 使用元数据来格式化查询输出	368
9.5 列举或检查数据库或表的扩展	372
9.6 访问表数据列定义	374
9.7 取得 ENUM 和 SET 数据列信息	381
9.8 在应用程序中使用表结构信息	383
9.9 获取服务器元数据	388
9.10 编写适合 MySQL 服务器版本的应用程序	389
9.11 确定默认数据库	390
9.12 监测 MySQL 服务器	391
9.13 确定服务器支持哪个存储引擎	393
第 10 章：数据导入导出	395
10.0 引言	395
10.1 使用 LOAD DATA 和 mysqlimport 导入数据	399
10.2 指定数据文件位置	401
10.3 指定数据文件的结构	403
10.4 处理引号和特殊字符	405
10.5 导入 CSV 文件	406
10.6 读取不同操作系统的文件	407
10.7 处理重复的键值	408
10.8 获取关于错误输入数据的诊断信息	408
10.9 跳过数据文件行	410
10.10 指定输入列顺序	411
10.11 在插入输入值之前对数据文件进行预处理	412
10.12 忽略数据文件列	413
10.13 从 MySQL 中导出查询结果	415
10.14 将表导出为文本文件	417
10.15 以 SQL 格式导出表内容或定义	418
10.16 将表或数据库拷贝到另一个服务器	420
10.17 编写你自己的导出程序	422
10.18 将数据文件从一种格式转化为另一种格式	426
10.19 提取和重排数据文件列	427
10.20 使用 SQL 模式来控制错误的输入数据处理	430
10.21 验证并转换数据	432
10.22 使用模式匹配来验证数据	435
10.23 使用模式来匹配广泛的内容类型	438
10.24 使用模式来匹配数值	439
10.25 使用模式来匹配日期或时间	441

10.26 使用模式来匹配 E-mail 地址或 URL	445
10.27 使用表元数据来验证数据	446
10.28 使用一个查找表来验证数据	449
10.29 将两个数字的年份值转化为四位形式	452
10.30 验证日期和时间合法性	453
10.31 编写时间处理工具	456
10.32 使用不完整的日期	461
10.33 导入非 ISO 格式日期值	462
10.34 使用非 ISO 格式导出日期值	463
10.35 导入和导出 NULL 值	464
10.36 根据数据文件猜测表结构	466
10.37 在 MySQL 和 Access 之间交换数据	469
10.38 在 MySQL 和 Microsoft Excel 之间交换数据	470
10.39 将输出结果导出为 XML	472
10.40 将 XML 导入 MySQL	476
10.41 尾声	478
第 11 章：生成和使用序列	481
11.0 引言	481
11.1 创建一个序列并生成序列值	482
11.2 为序列选择数据类型	485
11.3 序列生成的行删除的效果	487
11.4 查询序列值	490
11.5 对一个已有的序列进行重新计数	494
11.6 扩展序列的取值范围	496
11.7 序列顶部数值的再使用	497
11.8 确保各行按照给定顺序重编号	498
11.9 从某个特定值开始一个序列	499
11.10 序列化一个未序列化的表	500
11.11 使用 AUTO_INCREMENT 栏来创建多重序列	502
11.12 管理多重并发 AUTO_INCREMENT 数值	507
11.13 使用 AUTO_INCREMENT 值将表进行关联	508
11.14 将序列生成器用作计数器	511
11.15 创建循环序列	514
11.16 按行顺序输出数列查询	516
第 12 章：使用多重表	517
12.0 引言	517
12.1 在表中找到与另一个表中的行相匹配的行	518
12.2 查找与其他表不匹配的行	526
12.3 将表与自身进行比较	531

12.4 产生主从列表和摘要	536
12.5 枚举多对多的关系	539
12.6 查找每组行中含有最大或最小值的行	544
12.7 计算队伍排名	548
12.8 使用连接补全或识别列表的缺口	554
12.9 计算连续行的差值	559
12.10 发现累积和与动态均值	561
12.11 使用连接控制查询输出的顺序	565
12.12 在单个查询中整合几个结果集	567
12.13 识别并删除失配或独立行	572
12.14 为不同数据库间的表执行连接	575
12.15 同时使用不同的 MySQL 服务器	576
12.16 在程序中引用连接的输出列名称	579
第 13 章：统计技术	583
13.0 引言	583
13.1 计算描述统计	584
13.2 分组描述统计	587
13.3 产生频率分布	589
13.4 计数缺失值	592
13.5 计算线性回归和相关系数	594
13.6 生成随机数	596
13.7 随机化行集合	598
13.8 从行集合中随机选择条目	601
13.9 分配等级	602
第 14 章：处理重复项	607
14.0 引言	607
14.1 防止在表中发生重复	608
14.2 处理向表中装载行时出现的重复错误	610
14.3 计数和识别重复项	614
14.4 从表中消除重复项	618
14.5 从自连接的结果中消除重复	622
第 15 章：执行事务	627
15.0 引言	627
15.1 使用事务存储引擎	628
15.2 使用 SQL 执行事务	630
15.3 在程序中执行事务	631
15.4 在 Perl 程序中使用事务	634
15.5 在 Ruby 程序中使用事务	636
15.6 在 PHP 程序中使用事务	637



15.7 在 Python 程序中使用事务	638
15.8 在 Java 程序中使用事务	639
15.9 使用事务的替代方法	640
第 16 章：使用存储例程、触发器和事件	643
16.0 引言	643
16.1 创建复合语句对象	645
16.2 使用存储函数封装计算	647
16.3 使用存储过程来“返回”多个值	649
16.4 用触发器来定义动态的默认列值	650
16.5 为其他日期和时间类型模拟 TIMESTAMP 属性	653
16.6 使用触发器记录表的变化	655
16.7 使用事件调度数据库动作	658
第 17 章：关于 Web 应用中 MySQL 的介绍	661
17.0 引言	661
17.1 Web 页面产生的基本原则	663
17.2 使用 Apache 运行 Web 脚本	667
17.3 使用 Tomcat 运行 Web 脚本	678
17.4 在 Web 输出中编码特殊字符	688
第 18 章：在 Web 页面中混合查询结果	697
18.0 引言	697
18.1 以段落文本显示查询结果	698
18.2 以列表形式显示查询结果	700
18.3 以表格形式显示查询结果	712
18.4 将查询结果显示为超链接	717
18.5 根据数据库内容中创建导航索引	721
18.6 存储图片或其他二进制数据	726
18.7 检索图片或其他二进制数据	733
18.8 提供标语广告	736
18.9 提供可下载的查询结果	738
18.10 使用模板系统生成 Web 页面	741
第 19 章：用 MySQL 处理 Web 输入	761
19.0 引言	761
19.1 编写脚本生成 Web 表单	764
19.2 根据数据库内容构建单取表单元素	767
19.3 根据数据库内容构建多取表单元素	783
19.4 将一条数据库记录导入表单	788
19.5 收集 Web 输入	793
19.6 验证 Web 输入	804

19.7 将 Web 输入存入数据库	805
19.8 处理文件上传	808
19.9 执行搜索并显示结果	815
19.10 生成上一页和下一页链接	818
19.11 生成点击排序的表格头单元	822
19.12 Web 页面访问计数	827
19.13 Web 页面访问日志	831
19.14 使用 MySQL 存储 Apache 日志	833
第 20 章：使用基于 MySQL 的 Web 会话管理	841
20.0 引言	841
20.1 在 Perl 应用程序中使用基于 MySQL 的会话	845
20.2 在 Ruby 应用程序中使用基于 MySQL 的存储	850
20.3 在 PHP 会话管理器中使用基于 MySQL 的存储	854
20.4 在 Tomcat 中为会话支持存储使用 MySQL	865
附录 A：获取 MySQL 软件	875
附录 B：从命令行执行程序	881
附录 C：JSP 和 Tomcat 知识的初步内容	889
附录 D：参考资料	917
索引	921

计算机精品学习资料大放送

[软考官方指定教材及同步辅导书下载](#) | [软考历年真题解析与答案](#)

[软考视频](#) | [考试机构](#) | [考试时间安排](#)

[Java 一览无余](#): [Java 视频教程](#) | [Java SE](#) | [Java EE](#)

[.Net 技术精品资料下载汇总](#): [ASP.NET 篇](#)

[.Net 技术精品资料下载汇总](#): [C#语言篇](#)

[.Net 技术精品资料下载汇总](#): [VB.NET 篇](#)

[撼世出击](#): [C/C++编程语言学习资料尽收眼底](#) [电子书+视频教程](#)

[Visual C++\(VC/MFC\)学习电子书及开发工具下载](#)

[Perl/CGI 脚本语言编程学习资源下载地址大全](#)

[Python 语言编程学习资料\(电子书+视频教程\)下载汇总](#)

[最新最全 Ruby、Ruby on Rails 精品电子书等学习资料下载](#)

[数据库管理系统\(DBMS\)精品学习资源汇总](#): [MySQL 篇](#) | [SQL Server 篇](#) | [Oracle 篇](#)

[平面设计优秀资源学习下载](#) | [Flash 优秀资源学习下载](#) | [3D 动画优秀资源学习下载](#)

[最强 HTML/xHTML、CSS 精品学习资料下载汇总](#)

[最新 JavaScript、Ajax 典藏级学习资料下载分类汇总](#)

[网络最强 PHP 开发工具+电子书+视频教程等资料下载汇总](#)

[UML 学习电子资下载汇总](#) [软件设计与开发人员必备](#)

[经典 LinuxCBT 视频教程系列](#) [Linux 快速学习视频教程一帖通](#)

[天罗地网](#): [精品 Linux 学习资料大收集\(电子书+视频教程\)](#) [Linux 参考资源大系](#)

[Linux 系统管理员必备参考资料下载汇总](#)

[Linux shell、内核及系统编程精品资料下载汇总](#)

[UNIX 操作系统精品学习资料<电子书+视频>分类总汇](#)

[FreeBSD/OpenBSD/NetBSD 精品学习资源索引](#) [含书籍+视频](#)

[Solaris/OpenSolaris 电子书、视频等精华资料下载索引](#)

>> [更多精品资料请访问大家论坛计算机区...](#)

序言

Preface

MySQL 数据库管理系统最近几年已经有了很多的追随者，特别是在 Linux 和开源社区中，MySQL 在商业使用方面的市场份额也同样在增长。它因以下几个原因而广为人喜爱：快速，易于安装、使用和管理。它可以运行于多种 Unix 和 Windows 操作系统下，而且基于 MySQL 的程序可以使用多种语言来编写。从历史上看，它尤其流行于创建包含动态内容的数据库支撑的 Web 站点。此外，随着 MySQL 5.0 中诸如视图、触发器、存储过程以及函数等特征的引入，MySQL 对于应用开发其他领域的渗透也正在深入。

随着 MySQL 的流行，用户常有如何解决特定问题的疑问，为此提供答案的需求便也应运而生。这便是本书的目的所在。当你使用 MySQL 遇到特定类型的问题需要攻克时，本书就是一本唾手可得的资料，你可以在其中翻阅到所需要的快捷的解决方案或技术。自然地，因其是一本“食谱”，它包含了相应的方子：你可以直接依循指南而无须从头开发你自己的代码。它以问题和解决方案的格式写成，该格式非常实用，也使得内容易于阅读吸收。本书包含了很多小节，每个小节描述了如何编写一次查询，应用一项技术，或者开发一段脚本来解决特定范围的问题。本书并未试图去开发一个成熟的复杂应用。相反，它试图为你自己在开发这些应用时提供帮助来解决曾难倒你的问题。

例如，一个很常见的问题是，“当我编写查询时在数据值中出现引号和特殊字符该如何处理？”那并不难，但是当你不知从何开始时就有些难。本书阐述了要做什么，它向你揭示了从何开始以及如何继续。这些知识将反复为你提供帮助，因为在你明白它的内涵之后，你就能将这些技术应用到任意类型的数据，例如文本、图片、音频或视频片段、新闻文章、压缩文件或者 PDF 文档。另一个常见问题是，“我能同时访问多个表的数据吗？”答案是“可以”，它很简单，因为只要了解合适的 SQL 语法就可以了。但是直到你看到本书给你

的示例你才能清楚地知道怎么去做。你可以从本书学到的其他知识包括：

- 如何使用 SQL 来查询、排序和统计行。
- 如何发现两表间匹配或不匹配的行。
- 如何执行一次事务。
- 如何计算日期或时间的间隔，包括计算年龄。
- 如何识别或移除重复行。
- 如何将图片存入 MySQL 并在网页中查询出来以供显示。
- 如何合理使用 LOAD DATA 读取你的数据文件或者查明文件里的哪些值不正确。
- 如何使用 strict 模式来阻止错误数据进入你的数据库中。
- 如何将一个表或一个数据库拷贝到另一个服务器。
- 如何生成序列值以用作唯一的行标识符。
- 如何编写存储过程和函数。
- 如何将视图用作“虚拟表”。
- 如何设置触发器，使其在你插入或更新表行时被激活来执行特定的数据处理操作。
- 如何创建按照计划执行的数据库事件。

了解如何使用 MySQL 的其中一个方面是理解怎么和服务器进行通信——也就是怎么使用 SQL 格式化查询语言。因此，本书的一个重点就是使用 SQL 来阐明回答特定类型问题的查询。学习使用 SQL 的一个有用工具是包含在 MySQL 发行包中的 mysql 客户端程序。通过交互式地使用该客户端，你可以发送 SQL 语句到服务器并查看结果。这相当有用，因为它提供了 SQL 的直接接口。实际上，mysql 客户端如此有用，以致整个第 1 章都用来阐述它。

但仅仅发挥 SQL 查询的能力还不够。从数据库中获取的信息常需要进一步处理或者以特定的有用方式展现。如果你有复杂相互关系的查询时，譬如你要将一次查询的结果作为其他查询的基础时该怎么办？或者你需要生成特定格式需求的报表时该怎么办？这些问题将我们带到了本书的其他重点上——怎么通过应用程序接口（API）来编写和 MySQL 服务器交互的程序。当你了解如何在编程语言的上下文中使用 MySQL 时，你就获得了以下方式开发 MySQL 潜能的能力：

- 你可以记忆某次查询的结果，并在以后的某个时候使用它。
- 你可以充分发挥通用编程语言的功能。这就使你可以基于查询的成功或失败，或者基

于返回行的内容作出决定，然后采取相应的动作。

- 你可以随心所欲地格式化并显示查询结果。如果你正在编写一个命令行脚本，你可以生成普通文本。如果是一个基于 Web 的脚本，你可以生成一个 HTML 表格。如果它是一个提取信息用于传输到其他系统的应用，你可以生成一个 XML 形式的数据文件。

当你结合使用 SQL 和某种通用编程语言，你就拥有了一个非常灵活的发起查询并处理查询结果的框架。编程语言为你提供了一系列额外的功能用于执行复杂的数据库操作，从而提高了你的能力。然而这并不意味着本书很复杂。它尽量保持简单性，展示了怎么使用易于理解和掌握的技术来搭建“积木”。

将这些技术集成到你自己的应用中都由你自己来完成，你可以用以构建任意复杂的应用。毕竟，遗传密码仅基于四种核酸，而这些基本元素已被结合用于生成了我们周围的不计其数的物种。同样，音阶中仅有 12 个音符，但是经过熟练的作曲家之手，它们就能组成丰富多彩变化无穷的乐曲。同样的道理，在你掌握一系列简单的技术元素后，展开你的想象力，将它们运用到你想解决的数据库编程问题上，你就可以生成艺术性可能并不很强但肯定有用的应用，并能使你和其他人能有更高的生产率。

本书的读者对象

Who This Book Is For

本书对于任何想使用 MySQL 的人都是有用的，从想使用数据库来为如 blog 或 Wiki 等个人项目服务的个人用户到专业的数据库和 Web 开发者。本书也会吸引那些现在并未使用 MySQL 但有此意向的人。例如，如果你想学习数据库，但是认为大型数据库系统如 Oracle 不是作为学习工具的最好选择时，MySQL 就很适合你。

如果你不熟悉 MySQL，你会在本书中发现很多你从未想过的 MySQL 使用方法。如果你有丰富的使用经验，你可能会熟悉这里提出的许多问题，但你也许之前不得不花时间去解决它，而有了本书就会大大节省时间；利用本书给出的方案，将它们用在你的程序中远胜于你从头来编写代码。

本书甚至对于未使用 MySQL 的人也是有用的。你也许会想这是一本 MySQL 手册而不是 PostgreSQL 手册或者 InterBase 手册怎么能运用到除 MySQL 之外的数据库系统上呢。某种程度上的确如此，因为某些 SQL 构造是 MySQL 特有的。但是很多查询使用的是可移植到其他数据库引擎上的标准 SQL，所以你只须作小幅修改甚至无须改动就可使用它们了。

除此之外，一些编程语言接口提供了数据库无关的访问方法；无论你连接的是什么类型的数据库服务器，都用相同的方式来使用。

本书内容涵盖了介绍性的和高级的主题，所以如果某节所描述的技术对你来说很浅显，那就跳过它。或反之如果你发现很难读懂某节，最好的方法应该是先把它放在一边，在读了一些准备性的章节后再回过头去阅读它。

更高级的读者有时会问，为什么在一本关于 MySQL 的书中却提供了一些和 MySQL 无关的主题的说明性的资料，例如如何设置环境变量。之所以这么做，是想用我的经验去帮助那些刚开始了解 MySQL 的人们。MySQL 吸引人的一点就在于它易于使用，这使其成为没有广泛数据库背景的人们的普遍选择。然而，正是这样一些人常常会被一些简单的问题所难倒以致不能高效率地运用 MySQL，以一个很普通的问题为证，“如何避免在调用 mysql 的时候每次都要输入它的完整路径？”有经验的读者会立即意识到这是一个简单的设置 PATH 环境变量包含 mysql 安装目录的问题。但有的读者不会，特别是习惯于使用图形界面的 Windows 用户以及最近的那些 Mac OS X 用户（他们发现其熟悉的用户界面现在被 Terminal 应用程序中强大甚至可以说是神秘的命令行所加强了）。如果你就是这样的读者，我希望你能发现这些基本章节的有用之处，它们能帮助你冲破阻止你方便地使用 MySQL 的障碍。如果你是一个更高级的用户，则可跳过这些章节。

内容摘要

What's in This Book

很可能当你使用本书时脑海中已经存在一个准备开发的应用但是不太确定怎么实现它的某些部分。在这种情况下，你已经清楚你想解决哪种类型的问题，所以你应该搜索内容目录或者索引来查找某个能满足你需要的章节。理想情况下，该章节正好是你脑海中所想的。如果不是，你应该能找到某个解决类似问题的章节，这样你可以稍作调整就能满足手头的需要。我努力解释涵盖在开发每项技术里的原理以便你能进行修改以满足你应用的特定需求。

使用本书的另一方法就是脑海中不带着任何特定问题通读它。这样同样能给你以帮助，因为你会更深入地理解 MySQL 能做什么，所以我建议你能偶尔翻阅一下本书。如果你已经很熟悉本书并了解它所解决的问题种类那么它就会是一个更高效的工具。

当你进入后面的章节时，有时你会发现这些章节使用了前面章节所介绍的相关主题知识。

这在同一章内同样存在，后面的节常会使用该章前面所讨论的技术。如果你进入了一章，发现其中使用了你并不熟悉的某个技术，先查看内容目录或者索引看看有没有该技术的介绍。你应该会发现之前它已被解释过了。例如，如果你发现一节使用你不理解的 ORDER BY 子句来排序查询结果，翻到第 7 章，该章讨论了不同的排序方法并解释了它们是如何工作的。

以下段落对每章进行了概述以便你能对本书内容有个总体的认识。

第 1 章，使用 mysql 客户端程序，描述了如何使用标准的 MySQL 命令行客户端。mysql 通常是人们所使用的一个或主要的 MySQL 接口，所以了解如何使用它的功能相当重要。使用该程序你可以交互式地发起查询并查看其结果，所以很适于快速试验。你也可以在批处理模式下执行封装好的 SQL 脚本或者将其输出发送给其他的程序。除此之外，本章还讨论了使用 mysql 的其他方法，诸如如何对输出行进行计数或者使比较长的行更具可读性，如何生成不同的输出格式，以及如何记录 mysql 会话的日志等。

第 2 章，编写基于 MySQL 的程序，阐述了 MySQL 编程的基本要素：如何连接到服务器，发起查询，取回结果，以及处理错误。它还讨论了在查询中如何处理特殊字符和 NULL 值，如何编写库文件来封装公用操作代码，描述了收集构造服务器连接所需参数的不同方法。

第 3 章，从表中查询数据，涵盖了 SELECT 语句的多个方面，而 SELECT 语句是从 MySQL 服务器查询数据的主要手段：指定你想检索的列和行，进行比较，处理 NULL 值，以及选择查询结果的某个区域。后面的章节更详细地介绍了本章的一些主题，但本章提供了他们所依赖的概念的一个概观。如果你需要一些行选择方面的介绍性的背景知识或者你还不了解 SQL，你应该仔细阅读本章。

第 4 章，表管理，包括表克隆，将结果集拷贝到其他表，使用临时表，以及检查或改变某个表的存储引擎等。

第 5 章，和字符串共舞，描述了如何处理字符串数据。它涵盖了字符集和字符序、字符串比较、处理大小写敏感问题、模式匹配、分解和组合字符串，以及执行 FULLTEXT 搜索。

第 6 章，使用日期和时间，揭示了如何使用时间相关的数据。它描述了 MySQL 的日期格式以及如何以其他格式来显示日期值。它还包括如何使用 MySQL 特定的 TIMESTAMP 数据类型，如何设定时区，如何在不同时间单位间进行转换，如何执行日期算术操作来计算时间间隔或者从一个日期来生成另一个日期，以及闰年计算等。

第 7 章, 排序查询结果, 描述如何以你想要的顺序放置查询结果行。这包括指定排序方向, 处理 NULL 值, 解决字符串大小写敏感问题, 以及根据日期或部分列值排序等。它还提供了范例来说明如何对特定类型的值进行排序, 例如域名、IP 地址以及 ENUM (枚举) 值。

第 8 章, 生成摘要, 阐述用于评估一系列数据的一般性特征的技术, 例如某数据值是多少或者它的最小值、最大值或平均值是多少。

第 9 章, 获取和使用元数据, 讨论如何获得查询返回数据的信息, 例如结果集的行数或列数, 或者每列的名称和类型。它还揭示了如何询问 MySQL 哪些数据库和表可用, 或者查明表和列的结构。

第 10 章, 数据导入导出, 描述如何在 MySQL 和其他程序间传输信息。这包括怎么将文件从一种格式转化为另一种, 去除或重排数据文件中的列, 检查并验证数据, 改写如日期型等常以多种格式出现的值, 以及当你用 LOAD DATA 将数据值载入 MySQL 中时如何确定哪些数据值会引致问题。

第 11 章, 生成和使用序列, 讨论 AUTO_INCREMENT 列, MySQL 生成序列号的机制。它揭示了如何生成新的序列值或如何确定最新值, 如何重排一列, 如何在一个给定值开始序列, 以及如何建立一个表使它能同时维护多个序列值。它还揭示了如何使用 AUTO_INCREMENT 值来维护表间的主从关系, 包括一些需要避免的陷阱。

第 12 章, 使用多重表, 揭示了如何执行连接, 这是一种将一个表的行和其他表的行联合起来的操作。它阐述了如何比较表来找出匹配或不匹配之处, 如何生成主从列表和摘要, 列举多对多关系, 以及基于另一表的内容来更新或删除一表中的行。

第 13 章, 统计技术, 阐述如何生成描述性的统计表、频率分布、回归以及相关性。它还包括如何随机化一组行或从组中随机选取一行。

第 14 章, 处理重复行, 讨论如何识别、计算并移除重复行——以及 (首要的) 如何避免它们发生。

第 15 章, 执行事务, 揭示了如何处理必须作为一个单位一起执行的多条 SQL 语句。它探讨了如何控制 MySQL 的自动提交 (auto-commit) 模式, 以及如何提交 (commit) 或回滚事务, 并阐述了一些你能用于非事务性的存储引擎的背景知识。

第 16 章, 使用存储的程序、触发器和事件, 描述了如何编写存储在服务器端的存储函数和存储过程, 当表被修改时激活的触发器, 以及在预定的情况下执行的事件。

第 17 章, 关于 Web 应用中 MySQL 的介绍, 让你可以着手编写基于 Web 的 MySQL 脚本。Web 编程使你能从数据库内容生成动态页面或者收集信息存入你的数据库中。本章探讨如何配置 Apache 来运行 Perl、Ruby、PHP 和 Python 脚本, 以及怎么配置 Tomcat 使其能运行用 JSP 标签写成的 Java 脚本。它还提供了后面的章节中 JSP 页面常用到的 Java 标准标签库 (JSTL) 的一个概览。

第 18 章, 在 Web 页面中混合查询结果, 揭示了如何使用查询结果生成不同类型的 HTML 结构, 诸如段落、列表、表格、超链接和导航索引等。它也描述了如何向 MySQL 中存储图片, 以及之后怎么检索并显示他们, 以及如何发送一份可下载的结果集到一个浏览器。本章还包含专门的一节来阐述如何使用模板包来生成 Web 页面。

第 19 章, 用 MySQL 处理 Web 输入, 讨论如何获取 Web 上的用户输入并用它来创建新的数据库行或者作为执行查询的基础。它主要负责表单处理, 包括如何基于你的数据库中包含的信息创建表单元素, 例如单选按钮、弹出菜单, 或者复选框。

第 20 章, 使用基于 MySQL 的 Web 会话管理, 描述如何编写能记住跨越多个请求的信息的 Web 应用, 使用 MySQL 作为后端的存储。当你想逐步收集信息或者你需要基于用户先前的作为来作出决定时这很有用。

附录 A, 获得 MySQL 软件, 指出哪儿能得到本书示例的源码, 以及从哪儿获取需要的软件来用 MySQL 编写你自己的数据库程序。

附录 B, 从命令行执行命令, 提供在命令行提示中执行命令的背景, 以及如何设置 PATH 等环境变量。

附录 C, JSP 和 Tomcat 知识的初步内容, 提供了 JSP 的一般性叙述以及 Tomcat Web 服务器的安装指令。如果你需要安装 Tomcat 或者对此不是很熟悉, 或者你从未用 JSP 标签写过页面那么就请阅读本附录。

附录 D, 参考资料, 列出了提供本书所涵盖主题相关的额外信息的资料来源。它还列出了一些书籍, 提供了本书使用的编程语言的相关背景介绍。

本书使用的 MySQL API

MySQL APIs Used in This Book

很多语言都存在 MySQL 编程接口, 包括 C、C++、Eiffel、Java、Pascal、Perl、PHP、Python、Ruby、Smalltalk、以及 Tcl。在此情形下, 写一本 MySQL 入门书对作者而言是比较大的

挑战。无疑本书应该提供一些章节用 MySQL 来完成许多有趣且有用的事情，但是本书应该使用哪个或哪些 API 呢？如果每个问题用每种语言都实现一下，那么就会导致或者只能涵盖非常少的解决方案，或者非常非常大部头的书籍！当不同语言的实现互相之间有很多类同之处时还会导致大量的冗余。另一方面，使用多种语言也是很值得的，因为通常在解决特定类型问题方面一种语言相比另一语言更为合适。

为解决此两难局面，我从可用的 API 中挑选出了少数并用它们来编写本书的解决方案。允许从以下的 API 进行选择，这就合理地控制了便于管理的 API 数目范围：

- Perl 和 Ruby DBI 模块；
- PHP，使用 PEAR DB 模块；
- Python，使用 DB-API 模块；
- Java，使用 JDBC 接口。

为什么是这些语言？Perl 和 PHP 易于选择。Perl 毫无疑问是 Web 上最广泛使用的语言，它能如此受欢迎是因为其诸如文本处理能力方面的优点。特别地，它很流行于编写 MySQL 程序。PHP 也被广泛采用。PHP 的优点之一就是你可以用它很方便地访问数据库，这一点使其成为 MySQL 脚本编写的很自然的选择。在 MySQL 编程方面 Python 和 Java 或许不如 Perl 或 PHP 流行，但它们中每种都有大量的追随者。特别是在 Java 社区，MySQL 在开发者中有大量的追随者，他们使用 JSP 技术来构建数据库支撑的 Web 应用。在第 1 版中并未包含 Ruby，但我在第 2 版中引入了它，因为 Ruby 现在已经相当流行，它有一个模仿 Perl 模块的易于使用的数据库访问模块。

我相信采纳的这些语言结合起来能够满足大多数现有 MySQL 程序员的需要。如果你更喜欢某种本书未使用的语言，你仍然可以使用本书，但一定要注意第 2 章，要让你熟悉本书的主要 API。了解如何使用这里的编程接口来执行数据库操作将有助于你理解后面章节中的解决方案，以便你能将其迁移到其他的语言。

本书使用的约定

Conventions Used in This Book

本书通篇都使用以下字体约定：

等宽字体 (Constant Width)

用于程序清单，还有段落中引用的程序元素譬如变量或函数名、数据库、数据类型、环境变量、语句和关键字。

等宽粗体 (*Constant Width bold*)

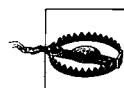
用于指出运行命令时你所输入的文本。

等宽斜体 (*Constant Width italic*)

用于指出变量输入；你应代之以你自己选择的某个值。



提示：本图标用于表示一个提示、建议或一般性的注解。



警告：本图标表示一个警告。

命令常和提示符一起出现以表明它所被使用的上下文。你从命令行发起的命令以一个%提示符显示：

```
% chmod 600 my.cnf
```

该提示符是 Unix 用户所惯见的一个，但并非该命令只能在 Unix 下工作。除非另外说明，通常以%提示符显示的命令也能在 Windows 下工作。

如果你要在 Unix 下以 root 用户执行一条命令，提示符会换为#：

```
# perl -MCPAN -e shell
```

Windows 下特有的命令使用 C:\>提示符：

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql"
```

在 mysql 客户端程序所发起的 SQL 语句以 mysql>提示符显示并以分号结束：

```
mysql> SELECT * FROM my_table;
```

对于使用 mysql 时你所见到的显示查询结果的示例，我有时会用省略号截去输出以表示结果集的行数超出显示。下面的查询生成多行输出，他们中间的部分已经被略去：

```
mysql> SELECT name, abbrev FROM states ORDER BY name;
+-----+-----+
| name        | abbrev |
+-----+-----+
| Alabama      | AL     |
```

```
+-----+-----+
| Alaska      | AK      |
| Arizona     | AZ      |
|
| West Virginia | WV      |
| Wisconsin    | WI      |
| Wyoming      | WY      |
+-----+-----+
```

仅显示 SQL 语句语法的示例并不包含 mysql> 提示符，但他们必须在语句结尾加上分号以使结构清晰可读。例如，这是一条语句：

```
CREATE TABLE t1 (i INT)
SELECT * FROM t2;
```

而这个示例代表两条语句：

```
CREATE TABLE t1 (i INT);
SELECT * FROM t2;
```

分号是用于 mysql 中作为语句结束符的符号。但它并非 SQL 本身的组成部分，所以当你在你所写的程序里（例如，使用 Perl 或 Java）发起 SQL 语句时，不应该包含结尾的分号。

MySQL Cookbook 的网站

The MySQL Cookbook Companion Web Site

本书有一个网站，你可以访问它来获得本书中示例的源代码和样本数据：

<http://www.kitebird.com/mysql-cookbook/>

样本数据和勘误表也在奥莱理（O'Reilly）网站上列了出来。

主要的软件发行包被命名为 *recipes*，你在本书中会发现很多对它的引用。你可以使用它以节约大量的输入。例如，当你在书中看见一条 CREATE TABLE 语句描述数据表的格式时，你通常可以在 *recipes* 发行包的 *tables* 目录下发现一个 SQL 批处理文件，可以用其创建数据表而不是手工输入定义。改变位置到 *tables* 目录，然后执行下面的命令，*filename* 是包含 CREATE TABLE 语句的文件名：

```
% mysql cookbook < filename
```

如果你需要指定 MySQL 用户名或密码选项，应将他们放在数据库名之前。

要获得更多关于 *recipes* 发行包的信息，请参考附录 A。

Kitebird 网站也使本书的一些示例可在线使用，所以你可以在浏览器中试用它们。

版本和平台注意事项

Version and Platform Notes

本书代码的开发都发生在 MySQL 5.0 和 5.1 下。因为在 MySQL 的一般性功能上添加了新的特性，所以一些示例在旧版本下不能执行。例如，MySQL 5.0 引入了视图、触发器和存储过程以及函数，以及 INFORMATION_SCHEMA 元数据数据库。MySQL 5.1 引入了事件。

有时，我也会指出在 MySQL 4.1 中你可以使用的一些背景知识，以弥补缺少相关 5.0 特征带来的问题。例如，INFORMATION_SCHEMA 数据库广泛用于获得表结构信息，但在 MySQL 5.0 之前并未出现。有时这些信息可以使用如 SHOW COLUMNS 等语言来获得，那是第 1 版中所采用的解决方案。这些解决方案仍可以从 Kitebird 网站获得，它们作为第 1 版发行包的部分而存在。如果你有旧版本的 MySQL，你会发现获得旧发行包的一份拷贝很有用。

本书目前所使用的数据库 API 模块的版本是 Perl DBI 1.52、DBD::mysql 3.0、Ruby DBI 0.1.1、PEAR DB 1.7.6、MySQLdb 1.2.0 和 MySQL Connector/J 3.1。Perl DBI 需要 Perl 5.6 或更高版本，尽管从 DBI 1.51 开始就需要 Perl 5.8 或更高版本了。Ruby DBI 要求 Ruby 1.8.0 或更高版本。本书的大多数 PHP 脚本可以运行在 PHP 4.1 或 PHP 5 之上（我推荐 PHP 5）。MySQLdb 要求 Python 2.3.4 或更高版本。MySQL Connector/J 要求 Java SDK 1.2 或更高（推荐使用 1.4 或更高）版本。

我并不假设你正在使用 Unix，尽管那是我自己的首选开发平台。（本书中，“Unix”泛指 Unix 同类系统诸如 Linux 和 Mac OS X。）本书的大多数资料都适用于 Unix 和 Windows。我用于开发本书解决方案的操作系统是 Mac OS X、Gentoo Linux 和 Windows XP。如果你使用 Windows，我假定你正使用的是较新的版本如 Windows 2000 或 XP。本书讨论的某些特性在较古老的非 NT 版本如 Windows Me 或 98 上并不支持。

我假定 MySQL 已经安装好并可供你使用。我还假设如果你有编写你自己的基于 MySQL 的程序的计划，你应该相当熟悉你将使用的语言。如果你要安装软件，可参考附录 A。如果你需要本书使用的编程语言的背景知识，可参考附录 D。

第 1 版读者升级时的注意事项

Upgrade Note for First Edition Readers

如果你有本书的第 1 版，要注意第 2 版的 PHP 库文件和第 1 版的同名，但是与其并不兼容。如果你正使用依赖于第 1 版库文件的 PHP 脚本，但你却安装了第 2 版的库文件，这些脚本文件将会中断。为避免此情况发生，可使用如下过程：

1. 改变位置到你安装第 1 版库文件的目录下，执行下面的每个拷贝命令：

```
% cp Cookbook.php Cookbook1.php  
% cp Cookbook_Utils.php Cookbook_Utils1.php  
% cp Cookbook_Webutils.php Cookbook_Webutils1.php  
% cp Cookbook_Session.php Cookbook_Session1.php
```

2. 找到包含第 1 版库文件的所有 PHP 脚本，将其改为包含*1.php 文件。例如一个包含 `Cookbook_Utils.php` 的脚本应变为包含 `Cookbook_Utils1.php`。（有些库文件本身也引用了其他库文件，所以你也要编辑这些*1.php 文件。）
3. 测试已修改的脚本以确定使用*1.php 文件能正常工作。

现在你可以在第 1 版文件之上安装同名的第 2 版库文件了。第 2 版的 PHP 脚本将使用这些新的库文件，而旧的脚本将会继续正常工作。

其他资源

Additional Resources

任何引人跟随的语言多从它的用户社区的贡献中受益匪浅，因为使用语言的人产生了大量他人可用的代码。Perl 尤其受到广泛的支持网络的帮助，该支持网络提供了很多未随 Perl 本身发布的外部模块。它被称为广泛 Perl 档案网络 (CPAN)，一种用于组织发布 Perl 代码和文档的机制。CPAN 包含了数据库访问，Web 编程以及 XML 处理等模块，这些和本书有直接相关性。其他语言也同样有外部支持：Ruby 有 Ruby 应用档案中心，PHP 有 PEAR 档案中心，Python 有一个名为 Vaults of Parnassus 的模块档案中心。对于 Java，一个好的出发点就是 Sun 的 Java 站点。下表显示了你可以访问以获取更多信息的站点。

API 语言	可以找到外部支持的位置
Perl	http://cpan.perl.org/
Ruby	http://raa.ruby-lang.org/
PHP	http://pear.php.net/
Python	http://www.python.org/
Java	http://java.sun.com/

使用代码示例

Using Code Examples

本书的目的是帮你完成工作。一般而言，你可以在你的程序和文档中使用本书的代码。你不需要联系我们以获取授权，除非你正复制代码的重要部分。例如，编写使用本书几个代

码片段的程序不需要授权。销售或发行奥莱理 (O'Reilly) 书籍中示例的 CD-ROM 则需要授权。引用本书或示例代码来回答问题不需要授权。将本书的重要的示例代码加入到你的产品文档中就需要授权。

我们感谢但并不强求标明对本书的引用 (attribution)。引用标注通常包括标题、作者、出版社，以及 ISBN。例如：“MySQL Cookbook, 第 2 版, Paul DuBois 著. Copyright 2007 O'Reilly Media, Inc., 978-0-596-52708-2.”

如果你感觉对代码示例的使用已经超出了合理的范围或者上述权限，请联系我们 permissions@oreilly.com。

联系博文视点

How to Contact Us

我们已尽力核验本书所提供的信息，尽管如此，仍不能保证本书完全没有瑕疵，而网络世界的变化之快，也使得本书永不过时的保证成为不可能。如果读者发现本书内容上的错误，不管是赘字、错字、语意不清，甚至是技术错误，我们都竭诚虚心接受读者指教。如果您有任何问题，请按照以下方式与我们联系。

奥莱理软件（北京）有限公司

北京市 海淀区 知春路 49 号 希格玛公寓 B 座 809 室

邮政编码：100080

网页：<http://www.oreilly.com.cn>

E-mail：info@mail.oreilly.com.cn

与本书有关的在线信息如下所示。

<http://www.oreilly.com/catalog/mysqlckbk2> (原书)

<http://www.oreilly.com.cn/book.php?bn=978-7-121-05993-3> (中文版)

北京博文视点资讯有限公司（武汉分部）

湖北省 武汉市 洪山区 吴家湾 湖北信息产业科技大厦 1402 室

邮政编码：430074

电话：(027) 87690813 传真：(027) 87690595

网页：<http://bv.csdn.net>

读者服务信箱：

reader@broadview.com.cn (读者信箱)

bvtougao@gmail.com (投稿信箱)

致谢

Acknowledgments

第 2 版的致谢

Second Edition

感谢技术审校者 Richard Sonnen 对书稿的审阅，他发现了很多问题并提出了许多改进本书的建议。Andy Oram 鼓励我开始第 2 版的写作并成为最初的编辑。当其他事务需要他处理时，Brian Jepson 接手了他的工作。感谢他们二位提供了编辑的监督和指引。Adam Witwer 指导了成书的处理。Mary Anne Mayo 编辑了文字，Sada Preisch 对其进行了校对。Joe Wizda 编写了索引。

感谢我的妻子 Karen，她在本书新版的写作过程中对我提供了颇有价值的支持。

第 1 版的致谢

First Edition

感谢技术审校者 Tim Allwine、David Lane、Hugh Williams 以及 Justim Zobel。他们在组织结构和技术精确性方面都提供了有用的建议和修正。MySQL AB 的几位成员友善地加入了他们的注解：特别是，首席 MySQL 开发者 Monty Widenius 梳理了所有的文字并指出了许多问题。Arjen Lenz、Jani Tolonen、Sergei Golubchik 和 Zak Greant 也审阅了部分原稿。Andy Dustman，《Python MySQLdb 模块》的作者，和 Mark Matthews，《MM.MySQL》和《MySQL Connector/J》的作者，也提供了宝贵建议。我衷心感谢所有人对于原稿的改进；所有剩下的错误都应由我来负责。

执行编辑 Laurie Petrycki，构思了本书并提供了颇有价值的总体编写指南和勘误。Lenny Muellner，工具专家，帮助将我原始格式的手稿转化成可印刷的格式。David Chu 扮演了助理编辑的角色。Ellie Volckhausen 设计了封面，选择的是我所乐于见到的动物形象。Linley Dolby 作为本书的产品编辑和校对者对书的出版付出了心血，Colleen Gorman、Darren Kelly、Jeffrey Holcomb、Brian Sawyer、Claire Cloutier 在质量控制方面提供了支持。

有些作者坐在键盘前写作更高效，而我在远离计算机时才能更好地写作——当然最好有一杯咖啡。因此，我要感谢维罗纳的 Sow's Ear 咖啡厅提供的优雅环境，在那里我度过了很多奋笔疾书的时光。

我的妻子 Karen 提供了相当多的支持和理解，而且是超乎预期的长期努力。她的鼓励令我深受鼓舞，她的耐心让我至为感激。

使用 mysql 客户端程序

Using the mysql Client Program

1.0 引言

Introduction

MySQL 数据库系统使用以 mysqld 服务器为中心的客户端-服务器架构。服务器是事实上操作数据库的程序。客户端程序并不直接操作数据库。相反，它们使用 SQL 语句来向服务器传达你的意图。客户端程序被安装在你想访问 MySQL 的本地机器上，但服务器可以安装在任何地方，只要客户端能连接上。MySQL 生来就是网络数据库系统，所以客户端能够和运行在本机或任何其他机器（也许在星球另一端的某台机器上）上的服务器进行通信。可能因为多个不同目的而写出一些客户端，但每个客户端和服务器的交互都是先连接到服务器，发送 SQL 语句到服务器来执行数据库操作，然后从服务器接收语句执行结果。

这其中的一个客户端就是包含在 MySQL 发布包中的 mysql 程序。当交互使用时，mysql 提示你输入一条语句，将其发送到 MySQL 服务器执行，然后显示结果。此项功能使 mysql 很有用，而且在你进行 MySQL 编程活动时也是一个颇有价值的工具。它可以在你用 script 访问时帮你很方便地快速查看表结构，也可以在你将一条语句用在程序中之前帮你确定该语句是否产生正确的输出，等等。mysql 正好适于这些工作。mysql 还能非交互地使用；例如，从文件或其他程序中读取语句。这可以使你在脚本内、定时的任务中或与其他应用结合起来使用 mysql。

本章描述了 mysql 的功能以便你能更高效地使用它。

- 启动、停止 mysql；
- 指定连接参数，使用可选择的文件；
- 设定你的 PATH 变量以便命令解释器能够找到 mysql（以及其他 MySQL 程序）；

- 交互执行 SQL 语句，使用批量文件；
- 取消和编辑语句；
- 控制 mysql 输出格式。

为使用本书中的实例，你需要一个用于工作的 MySQL 用户账号和数据库。本章的前两节描述了如何使用 mysql 配置这些。作为示范目的，示例假设您按如下方式使用 MySQL：

- MySQL 服务器运行在本机；
- MySQL 用户名和密码分别为 cbuser 和 cbpass；
- 数据库名为 cookbook。

当然，您自己试验时，可以不遵循这些假设。您的服务器可以不在本地运行，也不需要使用和本书中一致的用户名、密码或数据库名称。自然，如果你在你的系统中使用不同的默认值，你需要相应地改变示例。

即使你不使用 cookbook 作为你的数据库名称，我也建议你单独创建一个数据库来试验这儿的示例，而不是在一个你正用于其他用途的数据库中来做试验。否则，你已有的表名可能和示例中使用的表名发生冲突，这样的话你就不得不修改这些示例，而当你使用独立的数据库时就不会发生这种情形。

如果你用其他喜爱的客户端程序来执行查询操作，本章中的某些概念可能并不适用。例如，你可能倾向于图形化的 MySQL Query Browser 程序，它提供了 MySQL 数据库的点击式界面。在这种情况下，某些原理将会不同，譬如你中止 SQL 语句的方法。在 mysql 中，你使用分号 (;) 字符来中止语句，而在 MySQL Query Browser 中有一个 Execute 按钮用于中止语句执行。另一个流行的接口程序是 phpMyAdmin，它允许你通过 Web 浏览器来访问 MySQL。

本章用于建表的脚本位于发行版的 tables 目录下，其他的脚本位于 mysql 目录中。可从附录 A 获得发行版的相关信息。

1.1 建立 MySQL 用户账号

Setting Up a MySQL User Account

问题

你需要创建一个账号用于连接运行在给定主机上的 MySQL 服务器。

解决方案

使用 GRANT 语句建立 MySQL 用户账号。然后使用账号的名称和密码和服务器建立连接。

讨论

连接到 MySQL 服务器需要一个用户名和密码。你还可以指定服务器正运行的主机名称。如果你没有显式指定连接参数，mysql 会假设其为默认值。例如，如果你没有指定主机名，mysql 会假定服务器运行在本机上。

如果其他人已经为你建立了一个账号，那么就用该账号来创建和使用数据库。否则，下面的例子将会揭示如何使用 mysql 程序连接到服务器，并发起一个 GRANT 语句来创建具备访问名为 cookbook 数据库权限的用户账号。在所示的命令中，% 代表显示在你的 shell 或命令解释器中的提示。你所输入的文本被显示为粗体。非粗体的文本（包括提示）是程序输出；你不需要输入这些。mysql 的参数包括 -h localhost，表示连接到运行于本机的 MySQL 服务器；-p 用于告知 mysql 要提示输入密码；-u root 表示作为 MySQL 的 root 用户连接。

```
% mysql -h localhost -p -u root
Enter password: *****
mysql> GRANT ALL ON cookbook.* TO 'cbuser'@'localhost' IDENTIFIED BY 'cbpass';
Query OK, 0 rows affected (0.09 sec)
mysql> QUIT
Bye
```

在你输入如上面第一行的 mysql 命令后，如果你获得一条消息表明 mysql 命令找不到或者这是一个错误的命令，那么可参考 1.7 节。否则，当 mysql 打印出密码提示时，在你看见*****的地方输入 MySQL root 用户密码。（如果 MySQL root 用户没有密码，在密码提示处按下 Enter（或 Return）键就可以了。）然后发起一个如上所示的 GRANT 语句。

要允许 cbuser 账号访问 cookbook 外的数据库，将你在 GRANT 语句中的 cookbook 替换为你所想要的数据库名就可以了。要用一个已有的账号来访问 cookbook 数据库，用那个账号替换 'cbuser'@'localhost' 即可。然而，在该情况下，要略去语句中的 IDENTIFIED BY 'cbpass' 部分，否则你就会改变已有账号的当前密码。

'cbuser'@'localhost' 的主机名部分表明了你想访问 cookbook 数据库时连接到的 MySQL 服务器所在的主机。为建立一个连接到运行于本机的服务器的账号，如所示使用 localhost。如果你计划连接到位于其他主机的服务器，那么替换 GRANT 语句中的主机部分。例如，如果你想连接到名为 xyz.com 的主机上的服务器，GRANT 语句应该如下所示：

```
mysql> GRANT ALL ON cookbook.* TO 'cbuser'@'xyz.com' IDENTIFIED BY 'cbpass';
```

上面所描述的过程对于读者来说可能会产生一些迷惑。那在于，为建立一个能连接到 MySQL 服务器的 cbuser 账号，你必须首先连接到服务器以便你能发起 GRANT 语句。我在这里假设你已能作为 MySQL 的 root 用户连接，因为 GRANT 仅可被具备管理员权限的用户诸如 root 使用来创建其他的用户账号。如果你不能作为 root 用户连接到服务器，那么请你的 MySQL 管理员为你建立 cbuser 账号。

在已创建 cbuser 账号后，验证你能使用其来连接到 MySQL 服务器。从 GRANT 语句中所包含的主机上，运行下面的命令来进行验证（-h 后面的主机应该是正运行 MySQL 服务器的主机）：

```
% mysql -h localhost -p -u cbuser
Enter password: cbpass
```

接下来你可以如 1.2 节所描述的创建 cookbook 数据库，并在其中创建表。（为更简便的启动 mysql，无需每次指定连接参数，你可以将它们放在一个可选的文件里。见 1.4 节。）

MySQL 账号和登录账号

MySQL 账号不同于你操作系统的登录账号。例如，MySQL 的 root 用户和 Unix 系统的 root 用户是独立的，两者毫无瓜葛，即便是两者的用户名都相同。这也意味着它们很可能有不同的密码。这还意味着你不能通过创建你操作系统的登录账号来创建新的 MySQL 账号；而应代之以 GRANT 语句。

1.2 创建数据库和样表

Creating a Database and a Sample Table

问题

你想建立一个数据库，并在其中建表。

解决方案

使用 CREATE DATABASE 语句创建数据库，CREATE TABLE 语句创建你想用的每个表，用 INSERT 语句向表中添加数据行。

讨论

1.1 节中所述的 GRANT 语句创建了访问 cookbook 数据库的权限，但并未创建数据库。在你能使用该数据库之前必须显式创建它。本节阐述如何做到这一点，以及如何创建一个表，并将一些采样数据装入其中，这些数据在后面的章节中将会作为示例出现。

如 1.1 节末尾所示连接上 MySQL 服务器。在你成功连接后，创建数据库完成：

```
mysql> CREATE DATABASE cookbook;
```

现在你已经有了一个数据库，接下来你可以在其中创建表。首先，将 cookbook 选定为默认的数据库：

```
mysql> USE cookbook;
```

然后执行下面的语句来创建一个简单表，并插入一些数据行（注 1）：

```
mysql> CREATE TABLE limbs (thing VARCHAR(20), legs INT, arms INT);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('human',2,2);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('insect',6,0);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('squid',0,10);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('octopus',0,8);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('fish',0,0);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('centipede',100,0);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('table',4,0);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('armchair',4,2);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('phonograph',0,1);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('tripod',3,0);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('Peg Leg Pete',1,2);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('space alien',NULL,NULL);
```

表被命名为 limbs，它包含 3 列，分别用于记录不同生命体和对象所拥有的腿数和胳膊数。最后一行的外星生物的生理机能并不可知，所以它的腿数和胳膊数的适当值也不能决定；NULL 表示“未知值”。

可以通过执行一个 SELECT 语句来确认这些行被插进了表中：

```
mysql> SELECT * FROM limbs;
+-----+-----+-----+
| thing      | legs | arms |
+-----+-----+-----+
| human     |    2 |    2 |
| insect     |    6 |    0 |
| squid      |    0 |   10 |
| octopus    |    0 |    8 |
| fish       |    0 |    0 |
| centipede  |  100 |    0 |
| table      |    4 |    0 |
| armchair   |    4 |    2 |
| phonograph |    0 |    1 |
| tripod     |    3 |    0 |
| Peg Leg Pete |  1 |    2 |
| space alien | NULL | NULL |
+-----+-----+-----+
```

注 1：如果你不想输入 INSERT 语句的复杂文本（我不会怪你），跳到第 1.10 节可以获得一个便捷的方法。如果不想输入任何语句，请直接跳到第 1.12 节。

到此为止，你已经创建了一个数据库和一个表。要获悉更多执行 SQL 语句的指令，可参阅 1.8 节。



提示：本书中的语句里诸如 SELECT 或 INSERT 等 SQL 关键字都以大写形式来区分。然而，这仅仅是排版上的惯例。你可以以任意大小写形式来输入关键字。

1.3 启动和停止 mysql

Starting and Stopping mysql

问题

你想启动和停止 mysql 程序。

解决方案

在你的命令行提示中，输入 mysql 并指定必需的连接参数来启动它。要离开 mysql 环境，使用 QUIT 语句。

讨论

要启动 mysql 程序，试试在你的命令行提示处输入其名称。如果 mysql 正确启动，你会看见一段简短的信息，其后紧跟 mysql> 提示表明程序已经准备好接受语句。为作说明，下面展示了欢迎信息（为节省空间，在其他示例里我将不再展示它）：

```
% mysql  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 18427 to server version: 5.0.27-log  
  
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.  
  
mysql>
```

如果你调用了 mysql，但你得到了一条错误信息表示不能找到它或者它是一个错误命令，那就意味着你的命令解释器不知道 mysql 安装在什么位置。在 1.7 节中可见设置 PATH 环境变量的指令，这样你的命令行解释器就能找到命令。

如果 mysql 尝试启动，却马上出现 “access denied” 消息退出，那么你应该指定连接参数。最普遍需要的参数是要连接的主机（运行 MySQL 服务器的主机），你的 MySQL 用户名以及对应的密码。例如：

```
% mysql -h localhost -p -u cbuser  
Enter password: cbpass
```

如果你没有 MySQL 用户名和密码，你必须获得使用 MySQL 服务器的权限，如前面 1.1 节中所描述。

你为 mysql 指定连接参数的方法同样适用于其他的 MySQL 程序诸如 mysqldump 和 mysqladmin。例如，为生成一个包含 cookbook 数据库中的表备份的名为 cookbook.sql 的 dump 文件，如下执行 mysqldump：

```
% mysqldump -h localhost -p -u cbuser cookbook > cookbook.sql  
Enter password: cbpass
```

一些操作要求具备管理员权限的 MySQL 账号。mysqladmin 程序能执行仅 MySQL 的 root 账号可执行的操作，所以你需要如下调用它：

```
% mysqladmin -p -u root shutdown  
Enter password: ← 在此输入 MySQL root 用户密码
```

通常，我会在示例中展示没有连接参数选项的 MySQL 程序命令。我假设你能获取所需要的任意参数，不论是在命令行还是在可选项文件中，这样你就不用每次在调用 mysql，mysqldump 等程序时输入它们了。

下面的表格中显示了连接参数选项的语法和默认值。这些选项均有单破折号短格式和双破折号长格式两种。

参数类型	可选项语法格式	默认值
Hostname	-h <i>hostname</i> --host = <i>hostname</i>	localhost
Username	-u <i>username</i> --user = <i>username</i>	Your login name
Password	-p --password	None

如果你用在某个选项中的值等于其默认值，那么你就可以略去该选项。然而，如表中所示，不存在默认的密码值。为提供密码值，使用-p 或--password 选项，然后当 mysql 提示你的时候输入你的密码：

```
% mysql -p  
Enter password: ← 在此输入你的密码
```

如果你愿意，你可以通过在命令行里直接使用-p *password*（注意，-p 后并无空格）或--password = *password* 来直接指定密码。我并不赞成在多用户机器上使用此方式，因为这时对于运行诸如 ps（它报告进程信息）等工具的其他用户来说密码是可见的。

要终止一个 mysql 会话，发出 QUIT 命令：

```
mysql> QUIT
```

你也可以通过发出 EXIT 命令或按下 Ctrl-D（在 Unix 下）来中止会话。

在 MySQL 中 localhost 的意义

当连接到 MySQL 服务器时你所指定的参数之一就是服务器正运行于其上的主机。大多数程序将主机名 localhost 和 IP 地址 127.0.0.1 视作“本地机器”的同义词。但在 Unix 下，MySQL 程序表现出了不同：依照惯例，它们特殊对待主机名 localhost，这时它们会尝试使用一个 Unix domain socket 文件来连接本地服务器。要强制使用 TCP/IP 连接到本地服务器，那就使用 IP 地址 127.0.0.1 而不是主机名 localhost。可选的，也可以通过指定--protocol=tcp 选项来强制使用 TCP/IP 进行连接。

TCP/IP 连接的默认端口号是 3306。Unix domain Socket 的路径名常常变化，尽管通常情况为 /tmp/mysql.sock。为显式指定套接字文件路径名，使用 -S file_name 或 --socket= file_name 选项。

1.4 使用可选项文件来指定连接参数

Specifying Connection Parameters Using Option Files

问题

你不想在每次调用 mysql 或其他 MySQL 程序时都要在命令行中输入连接参数。

解决方案

将参数放入一个可选项文件中。

讨论

为避免手工输入连接参数，将它们放入一个可选项文件中供 mysql 自动读取。在 Unix 下，你的个人可选项文件位于你的 home 目录下的 .my.cnf。还存在全局的可选项文件，管理员可在其中指定能全局应用于所有用户的参数。你可以使用 /etc/my.cnf 或者 MySQL 安装目录下的 my.cnf 文件。在 Windows 下，你可以使用的可选项文件是你的 MySQL 安装目录（例如，C:\Program Files\MySQL\MySQL Server 5.0）下的 my.ini，或者你的 Windows 目录下的 my.ini（诸如 C:\WINDOWS 或 C:\WINNT），或者 C:\my.cnf 文件。

Windows 资源管理器显示文件时可能会隐藏文件的扩展名，所以名为 my.ini 或 my.cnf 的文件也许只显示为 my。当然，你的 Windows 版本可以允许你禁止扩展名隐藏功能。你也可以选择在控制台窗口中使用 DIR 命令来查看完整的文件名。

以下的示例阐述了用来编写 MySQL 选项文件的格式：

```
# 通用客户端程序连接选项
[client]
host      = localhost
user      = cbuser
password = cbpass

#mysql 程序特定选项
[mysql]
skip-auto-rehash
pager="/usr/bin/less -E" # 指定交互模式的 pager 设置
```

这种格式有如下的特点：

- 行以组（或节）的形式组织。每一组的第一行在方括号里包含组名，余下行指定与该组相关的选项。示例文件中有一个 [client] 组和一个 [mysql] 组。在一组中，以名值对 (*name=value*) 格式编写选项行，其中 *name* 对应一个选项名称（不能以破折号开头），*value* 对应选项的值。如果选项没有值（如 skip-auto-rehash 选项），略去尾部的 =*value* 部分，仅仅列出名称即可。
- 在选项文件中，只允许使用选项的长格式。这是和命令行相比较而言的，在其中可以任意使用长格式或者短格式来指定选项。例如，在命令行中，主机名可以使用 -h *hostname*，也可使用 --host = *hostname* 来指定。在选项文件中，只允许 host = *hostname*。
- 在选项文件中，= 前后允许有空格来分隔选项名和值。而在命令行中，= 前后不允许有任何空格。
- 如果选项值中包含空格或其他特殊字符，你可以用单引号或双引号将其包含起来。pager 选项说明了这一点。
- 如果你不需要某个特定的参数，那么略去对应的行即可。例如，如果通常都是连接到默认的主机 (localhost)，你就不需要 host 行。在 Unix 中，如果你的 MySQL 用户名与你的操作系统登录名相同，你就能略去 user 行。
- 使用选项文件来指定连接参数（如 host、user 和 password）选项已经很普遍了。然而，文件也能列出其他目的的选项。[mysql] 组中的 pager 选项制定了 mysql 在交互模式下用于显示输出的分页程序。它与程序如何连接到服务器无关。
- 通常指定客户端连接参数的选项组是 [client]。此组实际上为所有标准的 MySQL 客户端所使用。通过在该组中列上述选项，你不仅可以更方便地调用 mysql，而且可以调用其他程序如 mysqldump 和 mysqladmin。只要确定你放在该组中的选项可以为所有客户端程序理解即可。例如，如果你将 mysql 所特定的选项如 skip-auto-rehash

或 `pager` 放入 `[client]` 组中，会导致其他使用 `[client]` 组的程序出现“`unknown option`” 错误，这样它们就不能正常运行。

- 你可以在一个选项文件中定义多个组。一个普遍的惯例是对于一个程序而言在 `[client]` 组和以程序自身命名的组中去寻找参数。这就提供了一个方便的方式来列出你需要所有客户端程序都使用的一般性客户端参数，但你还能指定仅适用于某一特定程序的选项。前面的示例选项文件阐述了应用于 `mysql` 程序的此惯例，在其中从 `[client]` 组中获取一般的连接参数，而从 `[mysql]` 组中获取 `skip-auto-rehash` 和 `pager` 选项。
- 如果一个参数在选项文件中多次出现，最后出现的值具备高优先级。通常，你应该在 `[client]` 组后列出特定程序的组以便如果在两个组的选项集中有重合的选项，更普遍的选项将会被特定程序的值所覆盖。
- 以字符`#`或`;`开头的行将被作为注释而忽略。空行也会被忽略。如 `pager` 选项所示，`#` 能用在选项行的末尾来编写注释。
- 选项文件必须是普通文本文件。如果你使用具备一些非文本格式的字处理软件来创建选项文件，要确保将文件显式保存为文本。Windows 用户尤其要注意这些。
- 指定文件或目录路径名称的选项应该使用`/`作为路径名称分隔符，即使在 Windows 下原本使用`\`作为路径分隔符也是如此。另外，也可以以`\\"`来代替`\`（这是必要的，因为`\`在字符串中是 MySQL 的转义字符）。

如果你想查看 `mysql` 程序从选项文件中读取了哪些选项，使用这个命令：

```
% mysql --print-defaults
```

你也可以使用 `my_print_defaults` 工具，它接收其要读取的选项文件组的名称作为参数。例如，`mysql` 要在 `[client]` 和 `[mysql]` 组中查找选项，所以你可以通过使用以下命令来检查它从选项文件中读取了什么值：

```
% my_print_defaults client mysql
```

1.5 保护选项文件以阻止其他用户读取

Protecting Option Files from Other Users

问题

你的选项文件中保存了你的 MySQL 用户名和密码，你不希望其他的用户能读取该文件。

解决方案

设置文件的模式使其只能被你访问。

讨论

在多用户操作系统如 Unix 上，你应该保护你的选项文件以阻止其他用户读取它来发现如何使用你的账号连接到 MySQL。使用 chmod 设置其模式使其只能由你自身访问，从而使文件私有。下面的任一个命令都可做到这一点：

```
% chmod 600 .my.cnf  
% chmod go-rwx .my.cnf
```

在 Windows 下，你能使用 Windows 资源管理器来设置文件权限。

1.6 混合使用命令行和选项文件参数

Mixing Command-Line and Option File Parameters

问题

你不想将 MySQL 密码存储在一个选项文件中，但你又不想手工输入你的用户名和服务器主机。

解决方案

将用户名和主机放入选项文件中，但密码除外。取而代之的是，当你调用 mysql 程序时交互式地指定密码。mysql 在选项文件和命令行两者中查找连接参数。如果两处都指定了某个选项，命令行中指定的优先级较高。

讨论

mysql 首先读取你的选项文件以获悉其中列出了哪些连接参数，之后再检查命令行以获取其他的参数。这就意味着你可以用一种方法来指定某些选项，另一种方法来指定其他的。例如，你可以在选项文件中列出你的用户名和主机名，但在命令行中使用密码项：

```
% mysql -p  
Enter password: ←在此输入你的密码
```

命令行参数比你的选项文件中的参数优先级更高，所以即使因为某种原因你要过载一个选项文件参数，仅需要在命令行中指定即可。例如，你可能在选项文件中列出了你普通的 MySQL 用户名和密码以供多用途。如果你有时需要作为 MySQL root 用户连接，在命令行中指定用户和密码项来过载选项文件值就行了：

```
% mysql -p -u root  
Enter password: ← 在此输入 MySQL root 用户密码
```

当在选项文件中有一个非空密码时要显式指定“无密码”，在命令行中使用-p，然后在mysql 提醒你输入密码时按下回车即可：

```
% mysql -p  
Enter password: ← 在此按下 Enter 回车键
```

1.7 找不到 mysql 时该怎么做

What to Do if mysql Cannot Be Found

问题

当你在命令行中调用 mysql 时，命令行解释器却找不到它。

解决方案

将 mysql 安装目录加入你的 PATH 设置中。这样你就能方便地在任何目录运行 mysql。

讨论

在你调用 mysql 时，如果你的 shell 或命令行解释器不能找到它，你将会看到某种错误信息。在 Unix 下看起来如下所示：

```
% mysql  
mysql: Command not found.
```

或者在 Windows 下如下：

```
C:\> mysql  
Bad command or invalid filename
```

告知你的命令行解释器去哪儿查找 mysql 的一个方法就是每次你运行它时键入它的全路径名。在 Unix 下命令看起来如下所示：

```
% /usr/local/mysql/bin/mysql
```

或者在 Windows 下如下：

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql"
```

键入长路径名很快就变得相当烦人。你可以通过在你执行 mysql 前将所在位置改变到 mysql 安装所在目录来避免这一点。然而，我建议你不要那样做。如果你那样做了，你就会情不自禁地将你所有的数据文件和 SQL 批处理文件放在 mysql 同一目录下了，这就会让本只是程序存放的位置变得毫无必要地混乱不堪。

一个更好的解决方案是确保 mysql 安装目录被包括在 PATH 环境变量值里，PATH 环境变量

列出了命令解释器所要寻找命令的目录路径名称。你然后可以在任何目录仅输入其名称就能调用 mysql，你的命令解释器可以找到它。这减少了许多不必要的路径名输入。

一个明显的额外好处是因为你可以方便地在任何地方运行 mysql，所以你不需要将你的数据文件放在 mysql 所在的目录中。当你不用在一个特定位置运行 mysql 时，你就能以有意义的方式自由组织你的文件，而不是一些人为强加的方式。例如，你可以在你的 home 目录下为你所拥有的每个数据库创建一个目录，并将与给定的数据库相关的工作文件放在相应的目录中。

关于设置 PATH 变量的指南，请参考附录 B。

在这儿我已经指出了 PATH 搜索路径变量的重要性，因为我见过不知道这样一个东西存在的人们提出的许多问题，他们试图在 mysql 安装位置的 bin 目录中完成他们所有 MySQL 相关工作。这在 Windows 用户中尤为普遍。

在 Windows 下，避免输入路径名或改变路径到 mysql 目录的另一个方法是创建一个快捷方式并将其放在一个方便的位置例如桌面上。这样启动 mysql 就很方便了，只需打开快捷方式即可。要指定命令选项或启动目录，便捷快捷方式的属性。如果你不想总是用同样的选项来调用 mysql，为你需要的每个选项集创建一个快捷方式应该很有用。例如，创建一个快捷方式作为一个普通用户连接进行通用的工作，而另一个作为 MySQL root 用户连接进行管理性质的工作。

1.8 发起 SQL 语句

Initiating SQL Statements

问题

你已经启动了 mysql，现在你想发送 SQL 语句到 MySQL 服务器去执行。

解决方案

输入它们，但要让 mysql 知道哪儿是语句的结尾。

讨论

当你调用 mysql 时，它会显示 mysql> 提示符以告知你已经准备接受输入。要在 mysql> 提示符位置发起 SQL 语句，先输入语句，再在尾部加上一个分号 (;) 以表示语句结束，然后按下回车键。显式的语句终结符是必需的；mysql 并不将回车符解释为终结符号，因为

你可以在多个输入行中输入一条语句。分号是最普遍的终结符，但你也可以使用\g (“go”)来代替分号。因此，下面的示例是发起同一语句的等价物，即使它们在输入方式和终结符上都不相同：

```
mysql> SELECT NOW();
+-----+
| NOW()           |
+-----+
| 2005-11-15 16:18:10 |
+-----+
mysql> SELECT
-> NOW()\g
+-----+
| NOW()           |
+-----+
| 2005-11-15 16:18:18 |
+-----+
```

注意对于第二条语句，在第二个输入行中提示符从 mysql> 变成了->。Mysql 以这种方式改变提示符来让你知道它仍在等待输入直到语句终结符为止。

要明白作为语句终结符的字符; 和\g 序列都不是语句本身的组成部分。它们是 mysql 程序所使用的约定，mysql 识别这些终结符并在发送语句到 MySQL 服务器之前将其从输入中剥离。当你编写你自己的程序发送语句到服务器时（下一章中我们将进行介绍）记住这点是很重要的。在这些程序中，你不需要包含任何终结符；语句字符串自身的尾部就表示语句的结尾。事实上，加上一个终结符反而可能导致语句因错误而没有执行。

参考

mysql 也可以从文件或其他程序中读取语句。请参考 1.12 节和 1.13 节。

1.9 取消一条部分输入的语句

Canceling a Partially Entered Statement

问题

你开始输入一条语句，但最终决定不执行它。

解决方案

使用你的行删除符或者\c 来取消语句。

讨论

当你改变主意不想发起一条你正输入的语句时，那就取消它。如果语句在单行中，使用退格符或者你的行删除符来删除整行。(特定的行删除符要依赖于你终端所安装的操作系统；于我以及大多数 Unix/Linux 用户来说，该符号是 Ctrl-U；Windows 用户应使用 Esc。)

如果你已经在多行输入了一条语句，行删除符只能删除最后一行。要完全删去语句，输入 \c，然后按下回车。这将使你回到 mysql> 提示符处：

```
mysql> SELECT *  
-> FROM limbs  
-> ORDER BY\c  
mysql>
```

有时 \c 看来并未起到作用（也就是说，mysql> 提示符没有重现），这使你产生一种陷入语句中无法逃脱的感觉。如果 \c 没有效果，起因通常是你开始输入了一个引用的字符串，但是还没有输入匹配的引号来结束该字符串。让 mysql 的提示符来帮你指引道路吧：

- 如果提示符从 mysql> 变成了 " >，这意味着 mysql 正期待着尾部双引号的输入。如果提示符是 '> 或 ' >，mysql 正在期待着尾部单引号或 backtick。输入适当的相匹配的引号来结束字符串，然后输入 \c，再按下回车。
- 如果提示符是 /*>，你正位于输入 C 语言风格注释 /* ... */ 的中间。输入 */ 来结束注释，然后输入 \c，再按下回车。

1.10 重复和编辑 SQL 语句

Repeating and Editing SQL Statements

问题

你刚输入的语句包含一个错误，你想在不重新输入整个语句的前提下修订它。或者你想不重新输入来重复一条先前的语句。

解决方案

使用 mysql 内建的输入行编辑能力。

讨论

如果你发起一条很长的语句，可是发现它包含了一个语法错误，你应该怎么办？从零开始输入完整的修正后的语句？没有必要：mysql 维护了语句历史并提供了输入行编辑能力。这就可以使你恢复语句以便可以轻松地对其进行修改并重新执行它们。

存在许多许多编辑功能，但多数人在多数编辑过程中仅仅使用命令的一个很小的子集。一个有用命令的基本子集如下表所示。有代表性的，你使用 Up 方向键来恢复前一行，Left 方向键和 Right 方向键在行内移动，Backspace 或 Delete 来删除字符。要在行中加入新字符，将光标移动到合适的位置，然后输入它们即可。当你完成编辑，按下回车来发起语句（当你做这些时，光标不一定要在行末）。

编辑键	键的作用
Up 方向键	向上翻动语句历史
Down 方向键	向下翻动语句历史
Left 方向键	行内左移
Right 方向键	行内右移
Ctrl-A	移到行起始处
Ctrl-E	移到行末
Backspace	删除前一个字符
Ctrl-D	删除光标所包括的字符

在 Windows 中，方向键和退格符编辑功能如上表描述，Home 和 End 分别代替了 Ctrl-A 和 Ctrl-E，按 F7 将会展示给你一个最近所用命令的菜单。

输入行编辑并不仅仅对修正错误有用。你可以用它来尝试一条语句的不同形式，而无须每次都重新输入整个语句。它对于输入一系列类似的语句来说也很便捷。例如，如果你想使用语句历史来发起 1.2 节中所示的一系列 INSERT 语句来创建 limbs 表，首先输入起始的 INSERT 语句。然后，要发起每个后继语句，按下 Up 方向键来恢复前一语句，光标位置位于尾部，使用退格键来删除列值，输入新值，然后按下回车。

mysql 中的输入行编辑功能基于 GNU Readline 库之上构建。你可以阅读它的文档来查看可用编辑功能的更多信息。Readline 文档是 bash 指南的一部分，其在线位置为 <http://www.gnu.org/manual/>。

1.11 自动完成数据库名和表名

Using Autocompletion for Database and Table Names

问题

你希望有一个方式来更快的输入数据库和表名。

解决方案

使用 mysql 的名称自动完成工具。

讨论

通常，当你交互使用 mysql 时，在启动时会读取数据库名称和你默认的（当前的）数据库中的表和列名称的列表。Mysql 存储了这些信息并具有提供名称的能力，这对以更少的击键来输入语句来说是很有用的：

1. 输入部分的数据库，表或列名，然后按下 Tab 键。
2. 如果该部分名称是唯一的，mysql 将会为你补全。否则，你可以再按下 Tab 键来查看可能的匹配。
3. 多输入几个字符，再按一次 Tab 键补全它，或者按两次来查看新的匹配集。

mysql 的名称自动完成能力基于默认数据库中的表名，因此直到你在命令行中或者使用 USE 语句选择了一个数据库后才可以在 mysql 会话中使用。

自动完成帮你缩减了你要输入的数量。然而，如果你不使用此特性，从 MySQL 服务器读取名称完成信息可能会导致效率低下，因为当你的数据库中有大量数据表时它会导致 mysql 启动非常缓慢。要告知 mysql 不要读取此信息以便它能更快地启动，在 mysql 命令行中指定 -A (或 --skip-auto-rehash) 选项即可。作为替代方案，也可将 skip-auto-rehash 行放入 MySQL 选项文件的 [mysql] 组中：

```
[mysql]
skip-auto-rehash
```

要强制 mysql 读取名称完成信息，即使它是在非自动完成模式下调用的，应在 mysql> 提示符位置发起 REHASH 或 \# 命令。

1.12 让 mysql 从文件中读取语句

Telling mysql to Read Commands from a File

问题

你需要 mysql 从文件中读取语句以避免手工输入它们。

解决方案

重定向 mysql 的输入，或者使用 SOURCE 命令。

讨论

默认情况下，mysql 程序交互式地从终端读取输入，但是你可以使用其他的输入源如文件，另一个程序或者命令行参数等以批处理的方式为其提供语句。你也可以使用拷贝粘贴的方式来作为语句输入源。本节讨论如何从文件中读取语句。接下来的几节讨论如何从其他来源获取输入。

要以批处理的方式为执行 mysql 创建 SQL 脚本，应将你的语句放入一个文本文件内。然后调用 mysql，并重定向其输入从该文件读取：

```
% mysql cookbook < filename
```

从输入文件中读取的语句会替代你已经手工输入的部分，所以它们必须以分号（或\g）来结束，就如你手工输入它们一样。交互式和批处理方式两者间的区别之一在于默认的输出样式。在交互方式下，默认为表格（包装起来）格式。而在批处理方式下，默认是用制表符来定界的列值。然而，你可以使用合适的命令选项来选择你想要的任意一种输出样式。

当你需要多次发起一组给定的语句集合时，批处理方式很方便，因为你不需要每次都手工输入它们。例如，批处理方式可以很轻松地设置无须用户干预来运行的定时工作。SQL 脚本对于向其他人分发语句也很有用。事实上这也就是本书的 SQL 示例分发的方式。这里的很多示例都可以用随书的源码发行包中的脚本文件来运行。你可以在批处理方式下将这些文件作为 mysql 的输入来避免亲自输入语句。例如，当某一节展示了一条描述特定表结构的 CREATE TABLE 语句时，你通常会在该节的发行包中发现一个 SQL 批处理文件，可用其来创建（还可能向表中载入数据）表。回忆 1.2 节所显示的用于创建和组装 limbs 表的语句。当你手工输入这些语句时它们会显示出来，但该节发行包中包含了用于执行同一任务的 limbs.sql 文件。文件如下所示：

```
DROP TABLE IF EXISTS limbs;
CREATE TABLE limbs
(
    thing VARCHAR(20),    # 是什么事物
    legs  INT,           # 拥有的腿数
    arms  INT            # 拥有的臂数
);

INSERT INTO limbs (thing,legs,arms) VALUES('human',2,2);
INSERT INTO limbs (thing,legs,arms) VALUES('insect',6,0);
INSERT INTO limbs (thing,legs,arms) VALUES('squid',0,10);
INSERT INTO limbs (thing,legs,arms) VALUES('octopus',0,8);
INSERT INTO limbs (thing,legs,arms) VALUES('fish',0,0);
INSERT INTO limbs (thing,legs,arms) VALUES('centipede',100,0);
```

```
INSERT INTO limbs (thing,legs,arms) VALUES('table',4,0);
INSERT INTO limbs (thing,legs,arms) VALUES('armchair',4,2);
INSERT INTO limbs (thing,legs,arms) VALUES('phonograph',0,1);
INSERT INTO limbs (thing,legs,arms) VALUES('tripod',3,0);
INSERT INTO limbs (thing,legs,arms) VALUES('Peg Leg Pete',1,2);
INSERT INTO limbs (thing,legs,arms) VALUES('space alien',NULL,NULL);
```

要在批处理方式下执行 SQL 脚本文件中的语句，将所在路径变换到包含表创建脚本的该节发行包所在的表目录，然后执行此命令：

```
% mysql cookbook < limbs.sql
```

你会注意到脚本中包含了一条语句，该语句用于在创建表并向其中载入数据之前检查该表是否存在，如果存在会先删去该表。这就使你在利用该表做实验时不用担心改变其内容，因为无论任何时候你都可以通过再次执行该脚本将表恢复到它的基线状态。

刚刚显示的命令阐述了如何在命令行中为 mysql 指定一个输入文件。你可以在一个 mysql 会话中选用 SOURCE *filename* 命令（或者\.*filename*，它们是同义的）来读取一个 SQL 语句文件。假设 SQL 脚本文件 test.sql 包含以下语句：

```
SELECT NOW();
SELECT COUNT(*) FROM limbs;
```

你可以在 mysql 中执行该文件：

```
mysql> SOURCE test.sql;
+-----+
| NOW()          |
+-----+
| 2006-07-04 10:35:08 |
+-----+
1 row in set (0.00 sec)

+-----+
| COUNT(*)      |
+-----+
|      12       |
+-----+
1 row in set (0.01 sec)
```

SQL 脚本自身能包含 SOURCE 或\.*命令*来包含其他脚本。这为你提供了额外的灵活度，但也增加了危险，导致可能产生一个来源的循环。通常，你应该避免这些循环。如果你想搞恶作剧，试图故意创建一个循环来看看 mysql 能嵌套多少层输入文件，下面就是怎么做的办法。首先，手工发送以下两条语句创建一个 counter 表记录源文件深度并初始化嵌套层次为 0：

```
mysql> CREATE TABLE counter (depth INT);
mysql> INSERT INTO counter SET depth = 0;
```

接下来创建一个包含以下行的脚本文件 `loop.sql` (确保每行以分号结束):

```
UPDATE counter SET depth = depth + 1;
SELECT depth FROM counter;
SOURCE loop.sql;
```

最后, 调用 `mysql`, 并发送一个 `SOURCE` 命令来读取脚本文件:

```
% mysql cookbook
mysql> SOURCE loop.sql;
```

`loop.sql` 中的头两条语句增加嵌套的 `counter` 表列并显示当前的 `depth` 值。在第三条语句中, `loop.sql` 以自身为来源, 这样就创建了一个输入循环。你将看见输出连续不断, 通过循环每次计数显示都会增加。最终 `mysql` 将会出现文件描述符越界并以报错终结:

```
ERROR:
Failed to open file 'loop.sql', error: 24
```

`error 24` 是什么? 使用 MySQL 的 `perror` (打印错误) 工具来查看吧:

```
% perror 24
OS error code 24: Too many open files
```

由此可见, 你已经超出了操作系统可以打开的文件数目的界限。

1.13 让 `mysql` 从其他程序读取语句

Telling `mysql` to Read Statements from Other Programs

问题

你想以另一程序的输出作为 `mysql` 的输入。

解决方案

使用 `pipe` (管道)。

讨论

上一节中使用了以下命令来展示 `mysql` 如何从文件中读取 SQL 语句:

```
% mysql cookbook < limbs.sql
```

`mysql` 还可以读取一个 `pipe` (管道), 这就意味着它能接收其他程序的输出作为其输入。举个小例子, 前面的命令等价于这个:

```
% cat limbs.sql | mysql cookbook
```

在你告诉我荣获本周的“cat 无效使用奖”之前（注 2），请允许我观察一下你用其他的命令来替代 cat。要点在于生成包含以分号结尾的 SQL 语句的输出的任意命令都可用作 mysql 的输入来源。这在很多方面都是有用的。例如，mysqldump 工具通过编写一系列重建数据库的 SQL 语句来生成数据库备份。要处理 mysqldump 输出，将其供给 mysql。这意味着你可以结合使用 mysqldump 和 mysql 将一个数据库通过网络拷贝到另一个 MySQL 服务器：

```
% mysqldump cookbook | mysql -h some.other.host.com cookbook
```

当你需要用测试数据填充某个表而不想手工输入 INSERT 语句时，程序生成的 SQL 也是有用的。取而代之的，编写一个生成语句的简短程序，然后使用管道将其输出发送到 mysql：

```
% generate-test-data | mysql cookbook
```

参考

第 10 章进一步探讨了 mysqldump。

1.14 一行输入所有 SQL

Entering an SQL One-Liner

问题

你想直接在 mysql 命令行中指定要执行的语句。

解决方案

mysql 能从其参数列表中读取语句。

讨论

要直接从命令行中执行一条语句，使用 -e（或者 --execute）选项来指定。例如，要查出 limbs 表中有多少行，运行如下命令即可：

```
% mysql -e "SELECT COUNT(*) FROM limbs" cookbook
+-----+
| COUNT(*) |
+-----+
|      12   |
+-----+
```

要用此方式执行多条语句，以分号分隔即可：

注 2：在 Windows 下，等价物应该是“type 无效使用奖”：
C:/>type limbs.sql | mysql cookbook

```
% mysql -e "SELECT COUNT(*) FROM limbs;SELECT NOW()" cookbook
+-----+
| COUNT(*) |
+-----+
|      12   |
+-----+
+-----+
| NOW()          |
+-----+
| 2006-07-04 10:42:22 |
+-----+
```

默认的，如果显示到终端，由以-e 选项指定的语句生成的结果集是以表格格式显示的，否则以制表符定界格式展现。想了解这些不同格式，可见 1.17 节。要选择一个特定的格式，可见 1.18 节。

1.15 使用拷贝粘贴作为 mysql 输入源

Using Copy and Paste as a mysql Input Source

问题

你想利用你的图形用户界面（GUI）使 mysql 更易于使用。

解决方案

使用拷贝粘贴为 mysql 提供要执行的语句。这样你就能利用你 GUI 的能力为 mysql 展现的终端接口提供补充。

讨论

在 Window 环境下，拷贝粘贴非常有用，它能帮你一次运行多个程序并在它们之间传输信息。如果你在一个视窗中打开了包含语句的文档，你可以从那里拷贝语句然后将它们粘贴进你正运行 mysql 的视窗中。这和你自己手工输入语句并无二致，只是更快了。对于你频繁发出的语句，保持他们在一个独立的窗口中是一个很好的方法，这样就能确保他们总是触手可及。

1.16 预防查询输出超出屏幕范围

Preventing Query Output from Scrolling off the Screen

问题

查询输出超出了屏幕范围以致你看不见。

解决方案

让 mysql 每次都显示一页输出，或者在允许滚动的视窗中运行 mysql。

讨论

如果一条语句生成多行输出，通常这些行会超出屏幕的范围。为预防此情形，为 mysql 指定--pager 选项使其每次都显示一页输出（注 3）。`--pager = program`使 mysql 使用一个特定程序作为你的分页工具：

```
% mysql --pager=/usr/bin/less
```

--pager 自身（没带可选值）让 mysql 使用你默认的分页工具，它由 PAGER 环境变量指定：

```
% mysql --pager
```

如果你未设置 PAGER 变量，那么你必须定义它或者使用第一种形式的命令来显式指定一个分页程序。要定义 PAGER，使用附录 B 中所给出的指令来设置环境变量。

在一个 mysql 会话中，你可以分别使用\p 或\n 来开启或关闭分页功能。无参的\p 使用你的 PAGER 变量中指定的程序来开启分页功能。有参的\p 使用参数所指定的分页程序来开启分页功能：

```
mysql> \p
PAGER set to /bin/more
mysql> \p /usr/bin/less
PAGER set to /usr/bin/less
mysql> \n
PAGER set to stdout
```

处理长结果集的另一方法是使用允许你滚动浏览先前输出的终端程序。这些程序，例如 X Window 系统下的 xterm，Mac OS X 下的 Terminal 或 Windows 下的控制台窗口等，允许你设定保存在回滚缓存中的输出行数目。在 Windows 下，你可以使用以下步骤来设置一个允许回滚的控制台窗口：

1. 定位控制面板中的 Console 项或 cmd.exe 到以下目录：WINDOWS\system32。
2. 在 Console 项上点击右键或者鼠标拖动其到你想放置快捷方式的地方（例如桌面）来创建一个快捷方式。
3. 在快捷方式上点击右键，从弹出菜单中选择属性（Properties）项。

注 3：--pager 选项在 Windows 下不可用。

4. 在属性 (Properties) 窗口中选择布局 (Layout) 标签页。

5. 将屏幕缓存高度设为你想存储的行数，然后点击确定 (OK) 按钮。

现在你就可以启动快捷方式来获得一个可滚动的控制台窗口，在该窗口中命令产生的输出可使用滚动条检索。

1.17 发送查询输出到文件或程序

Sending Query Output to a File or to a Program

问题

你想让 mysql 输出到其他地方而非屏幕。

解决方案

重定向 mysql 的输出，或者使用管道 (pipe)。

讨论

mysql 按照你是以交互式还是非交互式的方式运行其来选择默认的输出格式。在交互式方式下，mysql 通常将输出发送到终端，并以表格形式展现查询结果：

```
mysql> SELECT * FROM limbs;
+-----+-----+-----+
| thing      | legs | arms |
+-----+-----+-----+
| human      |    2 |    2 |
| insect      |    6 |    0 |
| squid       |    0 |   10 |
| octopus    |    0 |    8 |
| fish        |    0 |    0 |
| centipede   |  100 |    0 |
| table       |    4 |    0 |
| armchair    |    4 |    2 |
| phonograph  |    0 |    1 |
| tripod      |    3 |    0 |
| Peg Leg Pete |    1 |    2 |
| space alien  | NULL | NULL |
+-----+-----+-----+
12 rows in set (0.00 sec)
```

在非交互式模式下（也就是说，输入或输出被重定向时），mysql 以制表符定界的格式展现其输出：

```
% echo "SELECT * FROM limbs" | mysql cookbook
thing  legs  arms
human  2     2
insect 6     0
squid  0     10
octopus 0    8
```

```
fish      0      0
centipede 100    0
table     4      0
armchair  4      2
phonograph 0      1
tripod    3      0
Peg Leg Pete 1      2
space alien NULL   NULL
```

然而，不管在哪个上下文下，你都能使用合适的命令选项来选择任意的 mysql 输出格式。本节描述如何将 mysql 输出发送到其他地方而非终端屏幕。接下来的几节探讨不同的 mysql 输出格式，以及在默认的输出格式不符合你的要求时，如何根据你的需要显式地选择他们。

要将 mysql 的输出保存到文件中，使用你 shell 的标准重定向功能：

```
% mysql cookbook > outputfile
```

如果你交互式地运行 mysql，并重定向其输出，你将会看不到你正输入的东西，所以通常此种情况下你也会从文件中获取输入语句（或者另一程序）：

```
% mysql cookbook < inputfile > outputfile
```

你也可以将语句输出发送给另一程序。例如，如果你想将查询输出通过邮件发送给某人，你可能以如下方式来执行：

```
% mysql cookbook < inputfile | mail paul
```

注意因为在该上下文下 mysql 是以非交互方式运行的，它生成了制表符定界的输出，而这种输出邮件接收者会发现比表格输出更难读取。1.18 节说明了如何解决这个问题。

1.18 选择表格或制表符定界的查询输出格式

Selecting Tabular or Tab-Delimited Query Output Format

问题

当你想获得制表符定界的输出时，mysql 却生成表格式的输出，或者相反。

解决方案

使用合适的命令选项来显示选择需要的格式。

讨论

当你非交互式使用 mysql 时（例如从文件中读取语句或者将结果发送到管道），默认是以制表符定界的格式进行输出。然而有时它需要以表格格式输出。（这些格式在 1.17 节中进行了描述。）例如，如果你想打印或邮寄语句结果，制表符分隔的输出就不太适合。应该

使用-t (或--table) 选项来生成更具可读性的表格输出:

```
% mysql -t cookbook < inputfile | lpr  
% mysql -t cookbook < inputfile | mail paul
```

反向操作是在交互模式下生成批量 (制表符定界) 输出。要达成这点, 使用-B 或--batch。

1.19 指定任意的输出列分隔符

Specifying Arbitrary Output Column Delimiters

问题

你想让 mysql 使用除制表符外的定界符来生成语句输出。

解决方案

mysql 自身并未提供设置输出定界符的功能, 但你可以对 mysql 输出进行加工来重新格式化。

讨论

在非交互模式下, mysql 以制表符来分隔输出列, 并且没有用于指定输出分隔符的选项。在某些场景下, 可能需要使用一个不同的分隔符来产生输出。假设你想创建一个输出文件供某个程序使用, 而该程序希望用冒号 (:) 而不是制表符来分隔值。在 Unix 下, 你可以使用如 tr 或 sed 等工具将制表符转化成任意分隔符。例如, 要将制表符转为冒号, 以下任一命令都能起到作用 (TAB 表示你应输入 tab 符号的位置) (注 4):

```
% mysql cookbook < inputfile | sed -e "s/TAB/:/g" > outputfile  
% mysql cookbook < inputfile | tr " TAB" ":" > outputfile  
% mysql cookbook < inputfile | tr "\011" ":" > outputfile
```

sed 比 tr 更为强大, 因为它能理解正则表达式并允许多次替换。当你想生成以逗号分隔值的输出格式时这就很有用, 此时需要三次替换:

1. 通过重复引号来去除数据中出现的所有引号字符, 以免当你使用最终的 CSV 文件时它们被解释成列分隔符。
2. 将制表符改变为逗号。
3. 以引号来包含列值。

Sed 允许以一条命令执行所有的三次替换:

注 4: 某些版本的 tr 语法是不同的: 请查阅你本地的文档。而且, 有些 shell 将制表符用于特殊目的, 如文件名补全。对于这些 shell, 在此之前使用 Ctrl-V 将一个文字的 tab 输入命令中。

```
% mysql cookbook < inputfile \
    | sed -e 's/""/"/g' -e 's/TAB /,"/g' -e 's/^"/' -e 's/$"/'/ > outputfile
```

不得不说，这显得很晦涩难懂。你可以使用其他可读性好的语言来实现同一效果。下面是一段 Perl 脚本，它实现了和 sed 命令相同的功能（它将以制表符定界的输入转化为 CSV 输出），还包含注释以说明其如何工作：

```
#!/usr/bin/perl -w
# csv.pl - 将制表符定界的输入转化为以逗号分隔的值输出
while (<>)      # 读取下一个输入行
{
    s/""/"/g;      # 使列值中的任何引号成双
    s/\t/, "/g;   # 在列值间放入 ","
    s/^"/;        # 在第一个值前加上 "
    s/$"/";       # 在最后一个值后加上 "
    print;         # 输出结果
}
```

如果你将脚本命名为 csv.pl，可以以如下方式使用它：

```
% mysql cookbook < inputfile | csv.pl > outputfile
```

如果你在某版本的 Windows 下（不能将.pl 文件和 Perl 关联起来）执行命令，就有必要显式调用 Perl：

```
C:\> mysql cookbook < inputfile | perl csv.pl > outputfile
```

tr 和 sed 通常在 Windows 下不可用。如果你需要一个跨平台的解决方案，Perl 也许是更为适合的选择，因为它在 Unix 和 Windows 下皆可运行。（在 Unix 系统上，Perl 通常是预装的。在 Windows 下，你可以自行安装。）

参考

一个生成 CSV 输出的更好的方法是使用 Perl Text::CSV_XS 模块，它就是为此目的而设计的。这个模块在第 10 章中有讨论，在那里它被用来构建一个通用的文件格式化工具。

1.20 生成 HTML 或 XML 输出

Producing HTML or XML Output

问题

你想将查询结果转化为 HTML 或 XML。

解决方案

Mysql 可以做到这点。使用 mysql -H 或 mysql -X。

讨论

如果你使用了-H(或者--html)选项mysql就会为每次查询结果集生成一个HTML表格。这就为你生成采样输出提供了一个便捷的方式，可以将其包含在Web页面中显示语句执行的结果。下面是一个示例，它显示了表格格式和HTML表格输出之间的差异(HTML输出中已经加入一些换行符使其更具可读性)：

```
% mysql -e "SELECT * FROM limbs WHERE legs=0" cookbook
+-----+-----+
| thing | legs | arms |
+-----+-----+
| squid | 0   | 10  |
| octopus | 0 | 8   |
| fish   | 0   | 0   |
| phonograph | 0 | 1   |
+-----+
% mysql -H -e "SELECT * FROM limbs WHERE legs=0" cookbook
<TABLE BORDER=1>
<TR><TH>thing</TH><TH>legs</TH><TH>arms</TH></TR>
<TR><TD>squid</TD><TD>0</TD><TD>10</TD></TR>
<TR><TD>octopus</TD><TD>0</TD><TD>8</TD></TR>
<TR><TD>fish</TD><TD>0</TD><TD>0</TD></TR>
<TR><TD>phonograph</TD><TD>0</TD><TD>1</TD></TR>
</TABLE>
```

表格的第一行包含列头部。如果你不需要头部行，可见1.21节。

如果你使用-x(或者--xml)选项，mysql会用语句的结果集创建一个XML文档：

```
% mysql -X -e "SELECT * FROM limbs WHERE legs=0" cookbook
<?xml version="1.0"?>

<resultset statement="select * from limbs where legs=0
">
  <row>
    <field name="thing">squid</field>
    <field name="legs">0</field>
    <field name="arms">10</field>
  </row>

  <row>
    <field name="thing">octopus</field>
    <field name="legs">0</field>
    <field name="arms">8</field>
  </row>

  <row>
    <field name="thing">fish</field>
    <field name="legs">0</field>
    <field name="arms">0</field>
  </row>
```

```
<row>
  <field name="thing">phonograph</field>
  <field name="legs">0</field>
  <field name="arms">1</field>
</row>
</resultset>
```

你也可以编写自己的 XML 生成器直接将查询结果转化为 XML。见 10.39 节。

-H, --html, -x 和--xml 选项仅供生成结果集的语句来产生输出。对于如 INSERT 或 UPDATE 等语句就没有作用。

参考

要想了解怎么编写程序将查询结果转化为 HTML, 可参考第 17 章。

1.21 在查询输出中禁止列头部

Suppressing Column Headings in Query Output

问题

你不想在查询输出中包含列头部。

解决方案

用合适的命令选项将 headings 关掉。通常, 该选项--skip-column-names, 但你也能使用--ss。

讨论

以制表符定界的格式对于生成可以导入其他程序的数据文件来说相当方便。然而, 每次查询输出的第一行默认都列出了列头部, 而这并不总是你所想要的。假设你有一个名为 summarize 的程序, 它能为一列数生成不同形式的描述性统计。如果你正使用 mysql 为此程序产生需要的输出, 你不会想要头部的行, 因为它会对结果造成干扰。也就是说, 如果你运行如下的命令, 输出就会不正确, 因为 summarize 将列头部也计算在内:

```
% mysql -e "SELECT arms FROM limbs" cookbook | summarize
```

要创建仅包含数据值的输出, 就要--skip-column-names 选项禁止列头部行:

```
% mysql --skip-column-names -e "SELECT arms FROM limbs" cookbook | summarize
```

你也可以通过两次指定“silent”选项（-s 或--silent）来达到和--skip-column-names 同等的效果：

```
% mysql -ss -e "SELECT arms FROM limbs" cookbook | summarize
```

1.22 使长输出行更具可读性

Making Long Output Lines More Readable

问题

你的一个查询生成了很长的输出行，它杂乱无章地显示在你的屏幕上。

解决方案

使用垂直的输出格式。

讨论

有些语句产生的输出行如此之长以致占据了终端上超过一行的空间，这就导致查询结果难以阅读。下面的示例显示了过长的查询输出行在你屏幕上的展现：

```
mysql> SHOW FULL COLUMNS FROM limbs;
+-----+-----+-----+-----+-----+-----+-----+
| Field | Type      | Collation | Null | Key | Default | Extra | Privileges
|        |           |           |       |     |         |       | Comment |
+-----+-----+-----+-----+-----+-----+-----+
| thing | varchar(20) | latin1_swedish_ci | YES |     | NULL    |       | select,insert,update,references |
| legs  | int(11)    | NULL        | YES |     | NULL    |       | select,insert,update,references |
| arms  | int(11)    | NULL        | YES |     | NULL    |       | select,insert,update,references |
+-----+-----+-----+-----+-----+-----+-----+
```

为避免此问题，将每列值显示在单独的一行来生成“垂直的”输出。这是通过使用\G 而不是用字符 a；或\g 来终止一条语句来实现的。下面是前面的语句的查询结果用垂直格式显示的样子：

```
mysql> SHOW FULL COLUMNS FROM limbs\G
***** 1. row *****
Field: thing
Type: varchar(20)
Collation: latin1_swedish_ci
Null: YES
Key:
Default: NULL
```

```
        Extra:  
Privileges: select,insert,update,references  
Comment:  
***** 2. row *****  
    Field: legs  
    Type: int(11)  
Collation: NULL  
    Null: YES  
    Key:  
Default: NULL  
    Extra:  
Privileges: select,insert,update,references  
Comment:  
***** 3. row *****  
    Field: arms  
    Type: int(11)  
Collation: NULL  
    Null: YES  
    Key:  
Default: NULL  
    Extra:  
Privileges: select,insert,update,references  
Comment:
```

要从命令行指定垂直输出，当你调用 mysql 时使用-E（或者--vertical）选项。这影响会话中发起的所有语句，当使用 mysql 来执行一条脚本时会很有用。（如果你在 SQL 脚本文件中使用通常的分号终结符来编写语句，你可以在命令行中选择使用-E 来选择正常的或者垂直输出。）

1.23 控制 mysql 的繁冗级别

Controlling mysql's Verbosity Level

问题

你需要 mysql 产生更多的输出，或者相反。

解决方案

使用-v 或-s 选项来获得更多或更少的输出。

讨论

当你非交互地运行 mysql 时，不仅默认的输出格式发生了变化，输出也变得更简洁。例如，mysql 并不打印行数或者提示语句执行所费的时间。要让 mysql 显示得更详细，使用-v 或--verbose。这些选项可以指定多次来增加详细程度。尝试一下下面的命令看看输出有何不同：

```
% echo "SELECT NOW()" | mysql  
% echo "SELECT NOW()" | mysql -v
```

```
% echo "SELECT NOW()" | mysql -vv  
% echo "SELECT NOW()" | mysql -vvv
```

和-v 和--verbose 相对的是-s 和--silent，它们也可以多次使用来达到叠加的效果。

1.24 记录交互式的 mysql 会话

Logging interactions in MySQL sessions

问题

你想保留一份在 mysql 会话中所做事情的记录。

解决方案

创建一个 tee 文件。

讨论

如果你维护一个交互式 MySQL 会话的日志，以后就可以用作参考以察看你做了什么、如何做的。在 Unix 下，可以使用脚本程序来保存终端会话的日志。这对于任意命令都有作用，所以也可以用到交互式的 mysql 会话中。但是，脚本会替记录的每一行都加上一个回车符，它还包括任何你所做的回退和更正，就如同你正在输入一样。记录交互式会话而不在日志文件中加入额外垃圾信息的方法（在 Unix 和 Windows 下都能起作用）是用--tee 选项来启动 mysql，指定记录会话的文件名（注 5）：

```
% mysql --tee=tmp.out cookbook
```

要在 mysql 里控制会话记录与否，可分别使用\T 或\t 来开启或关闭 tee 的输出。如果你想记录会话的某个特定部分这就很有用：

```
mysql> \T tmp.out  
Logging to file 'tmp.out'  
mysql> \t  
Outfile disabled.
```

tee 文件包含你所输入的语句以及这些语句的输出，所以它是一个保存完整记录的便捷方法。它很有用，例如，当你想打印或邮寄一个会话或其中一部分，或者要截取语句输出将其作为一个示例加入某个文档中时。在你将语句放入一个脚本文件之前执行语句以确定其语法正确，这也是一个好方法；以后你也可以从 tee 文件来创建脚本，通过编辑该文件，移除除了你想保留的语句外的其他所有东西就可以了。

注 5：它被称为“tee”，因为它类似于 Unix 的 tee 工具。在 Unix 中，阅读 tee 的参考手册页以获取更多背景知识：`% man tee`。

mysql 将会话输出附在 tee 文件的尾部而不是覆盖它。如果你想某个已有的文件仅包含单一会话的内容，在调用 mysql 之前先移除该文件。

1.25 以之前执行的语句创建 mysql 脚本

Creating mysql Scripts from Previously Executed Statements

问题

你想重用在先前的 mysql 会话中执行的语句。

解决方案

使用先前会话的 tee 文件，或者查看 mysql 的语句历史文件。

讨论

创建批处理文件的一个方法是在一个文本编辑器里从头开始输入你的语句到文件中，并期望你在输入的时候不会犯任何错误。但是使用你已经验证为正确的语句来说更为简单。怎么做？首先，在交互模式下使用 mysql “手动” 尝试语句来确定它们能正常工作。然后从你会话的记录中抽取语句来创建批处理文件。两个信息来源对于创建 SQL 脚本来说特别有用：

- 通过在 mysql 中使用--tee 命令选项或\T 命令你可以记录 mysql 会话的所有或部分。
- 在 Unix 下，第二个选项是使用你的历史文件。mysql 维护一份你语句的记录，保存在你的 home 目录的.mysql_history 文件里。

tee 文件会话日志有更多的上下文信息，因为它包含了语句的输入和输出，而不仅仅是语句文本。这些附加信息使得定位你所想要的会话部分更为方便。（当然，你一定要移除 tee 文件中多余的部分来创建批处理文件。）相反，历史文件更为简略。它仅包含你所执行的语句，所以很少有无关的行要删除以获得你所想要的语句。选择任意一种最适合你需要的信息来源。

1.26 在 SQL 语句中使用用户自定义的变量

Using User-Defined Variables in SQL Statements

问题

你想保存表达式所产生的值以便在后来的语句中可以引用它。

解决方案

使用一个用户定义的变量来存储该值以备后用。

讨论

你可以将 SELECT 语句的返回值赋给一个用户定义的变量，并且以后可以在你的 mysql 会话中引用该变量。这为保存一条语句返回的结果然后在其他的语句中引用它提供了一个方法。在 SELECT 语句中赋值给用户变量的语法是 `@ var_name := value`，这里 `var_name` 是变量名，`value` 是你正检索的值。变量可以用在以后语句中表达式可用的任何地方，例如 WHERE 子句或 INSERT 语句中。

用户变量起作用的一个普遍情形是当你需要在以公共的键值关联的多表中发起连续语句时。假设你有一个 customers 表，其 `cust_id` 列可用于区分每个客户，还有一个 orders 表也有一个 `cust_id` 列用于表明每个订单和哪个客户关联。如果你有一个客户名称，你想删除客户记录以及该客户的所有订单，你需要找到那个客户响应的 `cust_id`，然后删除 customers 表和 orders 表中匹配该 ID 的行。完成此任务的一个方法是首先将 ID 值保存在一个变量里，然后在 DELETE 语句中引用该变量：

```
mysql> SELECT @id := cust_id FROM customers WHERE cust_id='customer name';
mysql> DELETE FROM orders WHERE cust_id = @id;
mysql> DELETE FROM customers WHERE cust_id = @id;
```

上面的 SELECT 语句将列值赋给一个变量，但变量也可以由任意的表达式来赋值。下面的语句得出 limbs 表中 arms 列和 legs 列和的最大值并将其赋给变量 `@max_limbs`：

```
mysql> SELECT @max_limbs := MAX(arms+legs) FROM limbs;
```

变量的另一用途是保存在一个 AUTO_INCREMENT 列的表中插入新的一行后 `LAST_INSERT_ID()` 的结果：

```
mysql> SELECT @last_id := LAST_INSERT_ID();
```

`LAST_INSERT_ID()` 返回新的 AUTO_INCREMENT 列值。通过将其保存在变量中，你可以在之后的语句中多次引用该值，即使你发起其他的语句创建他们自身的 AUTO_INCREMENT 值，从而改变了 `LAST_INSERT_ID()` 返回的值。此技术在第 11 章中有进一步的讨论。

用户变量拥有单一的值。如果你使用返回多行的语句来将值赋给一个变量，那么只有最后一行的值被赋予了该变量：

```
mysql> SELECT @name := thing FROM limbs WHERE legs = 0;
+-----+
```

```

| @name := thing |
+-----+
| squid      |
| octopus   |
| fish       |
| phonograph|
+-----+
mysql> SELECT @name;
+-----+
| @name      |
+-----+
| phonograph|
+-----+

```

如果语句没有返回任何行，赋值也不会发生，变量会保持它先前的值。如果该变量之前从未被使用，那么值为 NULL：

```

mysql> SELECT @name2 := thing FROM limbs WHERE legs < 0;
Empty set (0.00 sec)
mysql> SELECT @name2;
+-----+
| @name2 |
+-----+
| NULL   |
+-----+

```

要将一个变量显式地设为一个特定值，使用 SET 语句。SET 语法可以使用 := 或 = 来进行赋值：

```

mysql> SET @sum = 4 + 7;
mysql> SELECT @sum;
+-----+
| @sum |
+-----+
| 11   |
+-----+

```

SET 也可以用来将一个 SELECT 的结果赋给一个变量，这里你要将 SELECT 写成一个子查询的形式（也就是说，在括号内），并且它返回单一的值。例如：

```
mysql> SET @max_limbs = (SELECT MAX(arms+legs) FROM limbs);
```

给定变量的值会一直持续到你赋给它另一个值或者你的 mysql 会话结束为止，任一个先出现都可。

用户变量的名称不是大小写敏感的：

```

mysql> SET @x = 1, @X = 2; SELECT @x, @X;
+-----+-----+
| @x  | @X  |
+-----+-----+
| 2   | 2   |
+-----+-----+

```



提示：在 MySQL 5.0 之前，用户变量名是大小写敏感的。

用户变量可以出现在仅允许表达式的地方，而不是必须是常量或文字标识符的地方。尽管尝试使用变量来作为表名等看起来很诱人，但是并不能正常工作。例如，你可以尝试使用如下的变量来生成一个临时的表名，但是它将不能工作：

```
mysql> SET @tbl_name = CONCAT('tbl_', FLOOR(RAND()*1000000));
mysql> CREATE TABLE @tbl_name (int_col INT);
ERROR 1064: You have an error in your SQL syntax near '@tbl_name
(int_col INT)'
```

用户变量是 MySQL 针对标准 SQL 的特定的扩展。在其他的数据库引擎中他们不能正常工作。

1.27 为查询输出行计数

Numbering Query Output Lines

问题

你想记下查询结果的行数。

解决方案

对 mysql 的输出进行处理或者使用用户定义的变量。

讨论

在 Unix 下当你想记录查询的输出行数时，结合使用 mysql 的--skip-column-names 选项和 cat -n 会很有用：

```
% mysql --skip-column-names -e "SELECT thing, arms FROM limbs" cookbook | cat -n
 1 human    2
 2 insect    0
 3 squid    10
 4 octopus 8
 5 fish     0
 6 centipede      0
 7 table    0
 8 armchair   2
 9 phonograph  1
10 tripod    0
11 Peg Leg Pete  2
12 NULL
```

另一个选择是使用用户变量。包含变量的表达式对于查询结果的每一行都会进行计算，你可以使用该属性在输出中提供一个行号列。

```
mysql> SET @n = 0;
mysql> SELECT @n := @n+1 AS rounum, thing, arms, legs FROM limbs;
+-----+-----+-----+
| rounum | thing      | arms | legs |
+-----+-----+-----+
|     1 | human      |    2 |    2 |
|     2 | insect      |    0 |    6 |
|     3 | squid       |   10 |    0 |
|     4 | octopus    |    8 |    0 |
|     5 | fish        |    0 |    0 |
|     6 | centipede  |    0 |  100 |
|     7 | table       |    0 |    4 |
|     8 | armchair   |    2 |    4 |
|     9 | phonograph |    1 |    0 |
|    10 | tripod     |    0 |    3 |
|    11 | Peg Leg Pete |  2 |    1 |
|    12 | space alien | NULL | NULL |
+-----+-----+-----+
```

1.28 将 mysql 用作计算器

Using mysql as a Calculator

问题

你需要一个快捷的方法计算一个表达式。

解决方案

将 mysql 用作计算器。MySQL 并不要求每个 SELECT 语句都关联到某个表，所以你可以选择任意表达式的结果。

讨论

SELECT 语句会关联到某些表，你可以从中查询多行数据。然而，在 MySQL 中，SELECT 根本不是必须引用表，这也就意味着你可以将 mysql 程序作为计算器来计算表达式：

```
mysql> SELECT (17 + 23) / SQRT(64);
+-----+
| (17 + 23) / SQRT(64) |
+-----+
|      5.00000000 |
+-----+
```

这对于检查比较关系如何工作很有用。例如，要确定给定的字符串比较关系是否大小写敏感，试验如下一条语句：

```
mysql> SELECT 'ABC' = 'abc';
+-----+
| 'ABC' = 'abc' |
+-----+
|          1      |
+-----+
```

此次比较的结果是 1 (意指 “true”；一般来说，非 0 值代表 true)。求得值为 false 的表达式返回 0：

```
mysql> SELECT 'ABC' = 'abcd';
+-----+
| 'ABC' = 'abcd' |
+-----+
|          0      |
+-----+
```

用户变量可以存储中间的计算结果。下面的语句以这种方式使用变量来计算宾馆账单的所有花费：

```
mysql> SET @daily_room_charge = 100.00;
mysql> SET @num_of_nights = 3;
mysql> SET @tax_percent = 8;
mysql> SET @total_room_charge = @daily_room_charge * @num_of_nights;
mysql> SET @tax = (@total_room_charge * @tax_percent) / 100;
mysql> SET @total = @total_room_charge + @tax;
mysql> SELECT @total;
+-----+
| @total |
+-----+
|    324  |
+-----+
```

1.29 在 Shell 脚本中使用 mysql

Using mysql in Shell Scripts

问题

你想在一个 shell 脚本中调用 mysql 而不是交互式地使用它。

解决方案

对此并无规则可循。只要确保为命令提供合适的参数。

讨论

如果需要在程序内处理查询结果，你应使用为你所正用的语言所特别设计的 MySQL 编程接口（例如，在 Perl 脚本内，你使用 DBI 接口；见 2.1 节）。但对于简单，简短快捷的任务，直接从 shell 脚本内调用 mysql 会更方便，可能要用其他命令做一些后续处理。例如，

编写 MySQL 服务器状态测试器的一个简便方法就是如本节所述的使用调用 mysql 的 shell 脚本。Shell 脚本对于创建程序原型也很有用，这个原型你在后面可以转化来用作编程接口。

对于 Unix shell 脚本，我建议你坚持在 Bourne shell 族内指定处理程序，诸如 sh, bash，或 ksh。(csh 和 tcsh shell 更适合交互式使用而非脚本。) 本节提供了一些示例说明如何为 /bin/sh 编写 Unix 脚本，并对 Windows 上的脚本编写作了详细的解释。

如果你需要在命令行解释器中执行程序的指南，或者确定你的 PATH 环境变量已经合理设置从而告知你的命令行解释器在哪些目录搜寻安装的程序，请参考附录 B。

这儿所讨论的脚本可以在 recipes 发行包的 mysql 目录下找到。

在 Unix 下编写 shell 脚本

下面是一个报告 MySQL 服务器当前运行时间的 shell 脚本。它执行一个 SHOW STATUS 语句获得 Uptime 状态变量的值，其中包含以秒数计的服务器运行时间（注 6）：

```
#!/bin/sh
# mysql_uptime.sh - 以秒数来报告服务器正常运行时间

mysql --skip-column-names -B -e "SHOW /*!50002 GLOBAL */ STATUS LIKE 'Uptime'"
```

mysql_uptime.sh 执行 mysql，它使用--skip-column-names 来去除列头部行，-B 来生成批量（定界符分隔的）输出，-e 来表示要执行的语句字符串。以#!开头的脚本第一行是特殊的。它表示用于调用来执行脚本剩余部分的程序的路径名称，这里是 /bin/sh。要使用这个脚本，创建一个包含上述代码行的名为 mysql_uptime.sh 的文件，用 chmod +x 使其可执行，然后运行它。最终的输出如下所示：

```
% ./mysql_uptime.sh
Uptime 1260142
```

这儿所示的命令以./开始，表示脚本位于你的当前目录下。如果你将脚本移动到你的 PATH 设定中的一个目录下，你可以在任何地方调用它，当在你运行脚本时应该略去开头的 ./。

如果你想要一份以天数，小时数，分钟数，秒数方式而不仅仅是秒数方式列出时间的报告，你可以使用 mysql STATUS 语句的输出，它提供了如下信息：

```
mysql> STATUS;
Connection id: 12347
```

注 6：要获取 /*!50002 GLOBAL */ 注释的解释，请参考第 9.12 节。

```
Current database:      cookbook
Current user:         cbuser@localhost
Current pager:        stdout
Using outfile:         ''
Server version:       5.0.27-log
Protocol version:     10
Connection:           Localhost via UNIX socket
Server characterset:  latin1
Db     characterset:  latin1
Client characterset: latin1
Conn.  characterset: latin1
UNIX socket:          /tmp/mysql.sock
Uptime:               14 days 14 hours 2 min 46 sec
```

对于正常运行时间报告来说，信息中仅有的相关部分是以 `Uptime` 开头的行。编写发送一条 `STATUS` 命令到服务器的脚本并用 `grep` 来过滤输出得到所需要的行是很简单的事：

```
#!/bin/sh
# mysql_uptime2.sh - 报告服务器正常运行时间

mysql -e STATUS | grep '^Uptime'
```

结果如下：

```
% ./mysql_uptime2.sh
Uptime:           14 days 14 hours 2 min 46 sec
```

前面的两个脚本使用 `-e` 命令选项指定要执行的语句，但你也可以使用本章先前所述的其他 `mysql` 输入源，例如文件和管道。例如，下面的 `mysql_uptime3.sh` 脚本类似于 `mysql_uptime2.sh` 但使用管道为 `mysql` 提供了输入：

```
#!/bin/sh
# mysql_uptime3.sh - 报告服务器正常运行时间

echo STATUS | mysql | grep '^Uptime'
```

有些 shell 支持一种“本地文档”的概念，它和输入到命令的文件有同样的用处，除了不包括一个显式的文件名以外。（换句话说，文档正好位于脚本之中，而不是存储在一个外部文件中。）要使用一个本地文档为命令提供输入，使用下面的语法：

```
command <<MARKER
input line 1
input line 2
input line 3
...
MARKER
```

`<< MARKER` 表示输入的开始并指出标记变量用于寻找输入的结尾。你用作标记的变量相对

任意选择，但应该与送到命令的输入中的标识符不同。

当你需要指定一条冗长的语句或者多条语句作为输入时，本地文档是-e 选项的有效替代品。在这些情况下，当-e 变得难于使用时，一个本地文档编写起来是更为方便和简单的。假设你有一个日志表 log_tbl，它包含一列 date_added 表示每行什么时候被添加。报告昨天添加的行数的语句如下所示：

```
SELECT COUNT(*) As 'New log entries:'
FROM log_tbl
WHERE date_added = DATE_SUB(CURDATE(), INTERVAL 1 DAY);
```

这条语句可以在脚本中使用-e 指定，但命令行将变得难于阅读因为语句如此之长。在此处本地文档是更为合适的选择，因为你可以用更具可读性的方式来编写语句：

```
#!/bin/sh
# new_log_entries.sh - 统计昨天的日志条目

mysql cookbook <<MYSQL_INPUT
SELECT COUNT(*) As 'New log entries:'
FROM log_tbl
WHERE date_added = DATE_SUB(CURDATE(), INTERVAL 1 DAY);
MYSQL_INPUT
```

当你使用-e 或本地文档，你可以在语句输入内引用 shell 变量——尽管下例说明应尽量避免这么做。假设你有一个简单的脚本 count_rows.sh 用于统计 cookbook 数据库中任意表的行数：

```
#!/bin/sh
# count_rows.sh - 统计 cookbook 数据库表中的行数

#需要命令行中的一个参数
if [ $# -ne 1 ]; then
    echo "Usage: count_rows.sh tbl_name";
    exit 1;
fi

#在查询字符串中使用参数($1)
mysql cookbook <<MYSQL_INPUT
SELECT COUNT(*) AS 'Rows in table:' FROM $1;
MYSQL_INPUT
```

脚本使用\$# shell 变量，它表示命令行参数数目，以及\$1，它代表脚本名称之后的第一个参数。count_rows.sh 确定仅提供一个参数，并用它作为行统计语句中的表名。要运行脚本，可用一个表名参数来调用它：

```
% ./count_rows.sh limbs
Rows in table:
12
```

变量替换对于构建语句很有帮助，你应该谨慎使用此功能。如果你的脚本可以在你的系统上被其他用户所执行，有些人可能以如下恶意的方式来调用它：

```
% ./count_rows.sh "limbs;DROP TABLE limbs"
```

这是“SQL注入”攻击的一种简单形式。在参数替换后，mysql 的最终输入如下所示：

```
SELECT COUNT(*) AS 'Rows in table:' FROM limbs;DROP TABLE limbs;
```

该输入统计了表的行数，接着删去了该表！出于此原因，在你个人私有的脚本中必须谨慎地限制变量替换的使用。有选择地使用 API 重写该脚本，这个 API 可以处理特殊字符，如并无副作用的生成语句。第 2.5 节涵盖了完成此功能的技术。

在 Windows 下编写 shell 脚本

在 Windows 下，你可以在一个批处理文件（一个扩展名为.bat 的文件）内运行 mysql。下面是一个 Windows 批处理文件，mysql_uptime.bat，它类似于先前所示的 mysql_uptime.sh Unix shell 脚本：

```
@ECHO OFF  
REM mysql_uptime.bat -以秒数来报告服务器正常运行时间  
  
mysql --skip-column-names -B -e "SHOW /*!50002 GLOBAL */ STATUS LIKE 'Uptime'"
```

批处理文件调用时可以不用.bat 扩展名：

```
C:\> mysql_uptime  
Uptime 9609
```

然而 Windows 脚本有一些重要限制。例如，不支持本地文档，并且命令参数引用能力有更多的限制。这些问题的解决方法之一就是安装一个更为合理的工作环境；参考工具栏的“发现 Windows 命令行受限了吗？”

发现 Windows 命令行受限了吗?

如果你是一个习惯了作为 Unix 命令行接口一部分的 shell 和工具的 Unix 用户，你或许会认为本章使用的一些命令理应如此，例如 grep、sed 和 tr。这些工具在 Unix 系统上是如此普遍使用，以至于当你发现在 Windows 下它在控制台窗口中进行工作时是必须的时候，却发现找不到它们，这是多么猛烈而又痛苦的打击。

使 Windows 命令行环境更合乎人意的一个方法是为 Windows 安装 Cygnus 工具 (Cygwin) 或者 Unix (UWIN)。这些包包括一些很流行的 Unix shell 以及许多 Unix 用户所期盼的工具。编程工具例如编译器也在每个包下可用。可以在以下位置获取这些包的发布版本：

<http://www.research.att.com/sw/tools/uwin/>

<http://www.cygwin.com/>

这些发布版本可以改变在 Windows 你使用本书的方式，因为它们消除了一些例外情况，在这些情况下命令可以在 Unix 下可用的命令，而在 Windows 下则不行。通过安装 Cygwin 或 UWIN，许多差异已经毫无关系。



编写基于 MySQL 的程序

Writing MySQL-Based Programs

2.0 引言

Introduction

本章探讨如何编写使用 MySQL 的程序。它涵盖了构成后续章节所开发的编程解决方案基础的基本应用编程接口（API）操作。这些操作包括连接到 MySQL 服务器、发起语句，以及检索结果。

基于 MySQL 的客户端程序可以用几种语言来编写。本书包含的语言是 Perl、Ruby、PHP、Python 和 Java。对于这些语言我们将使用下表中相应的接口。

语言	接口
Perl	Perl DBI
Ruby	Ruby DBI
PHP	PEAR DB
Python	DB-API
Java	JDBC

MySQL 客户端 API 提供了以下的能力，每一种能力对应于本章的一节：

连接到 MySQL 服务器，选择一个数据库，以及和服务器断开连接

一个默认的数据库可供使用。除此之外，表现良好的 MySQL 程序在完成任务时会关闭服务器连接。

查错

许多人写的 MySQL 程序根本没有任何错误检查。这样的程序在出错时很难调试。任何数据库操作都可能失败，你应知道怎么发现什么时候出错以及因何出错。此知识可以使你采取合适的措施，例如中止程序或者通知用户。

执行 SQL 语句并检索结果

连接到数据库服务器所有的要点都在于执行 SQL 语句。每个 API 都至少提供了一种方法来完成此功能，并且有几种方法用于处理语句结果。

处理语句中的特殊字符及 NULL 值

编写引用了特殊数据值的语句的一个方法就是在语句字符串中直接嵌入该值。然而，某些字符譬如引号和反斜线有特殊的含义，因此当你构建包含它们的语句时必须采取某种预防措施。对于 NULL 值也是同理。如果你没有合理处理这些情况，你的程序就可能生成错误的 SQL 语句或者产生意外的结果。如果你将外部来源的数据组装入查询中，你可能会受到 SQL 的注入攻击。大多数 API 提供了一种方法，允许你通过符号来引用数据值编写语句。当你执行语句时，你分别提供数据值，API 对任何特殊字符或 NULL 值进行合适编码后将其放入语句中。

识别结果集中的 NULL 值

NULL 值不仅在你构建语句时是特殊的，而且在语句返回的结果集中也是。每个 API 都提供了识别和处理它们的约定。

不论你使用的是何种编程语言，了解如何执行每个基本的数据库 API 操作都是必要的，所以每个操作都用所有的 5 种语言表示出来了。领会每个 API 怎么处理一个给定的操作可以帮你更轻松地了解 API 之间的相似之处，并能更好地理解后面章节中所示的解决方案，即使它们是用你使用得不多的语言所写就。（后面的章节通常仅用一或两种语言来阐述解决方案。）

如果你仅仅对一种特定的 API 感兴趣但却让你了解每个解决方案的几种语言实现，这也许让你无法接受。如果是这样，我建议你以如下方式来了解解决方案：只阅读提供一般性背景的介绍性部分，然后直接去阅读你所感兴趣的語言部分，跳过其他语言。如果你以后有兴趣使用其他语言来编写程序，你总是可以回过头来阅读其他的部分。

本章还讨论下面的主题，它们并非 MySQL API 的直接部分，但能帮你更方便地使用它们：

编写库文件

当你写了一个又一个程序后，你可能会发现有一些你反复执行的特定操作。库文件提供了一种封装这些操作代码的方法，以便你能从多个脚本中执行他们而无须在每

个脚本中包含所有的代码。这减少了代码重复并使你的程序更具可移植性。本节展示了如何为每个 API 编写一个库文件，它包含连接到服务器的例行操作——每个使用 MySQL 的程序都必须执行的操作。(后面的章节为其他操作开发了另外的库程序。)

其他获取连接参数的技术

之前的关于建立到 MySQL 服务器的连接一节中连接依赖于和代码紧密绑定的连接参数。然而，有几种其他的方法可以用于获取参数，这些方法涵盖了从将它们存储于独立的文件中到允许用户在运行时指定它们的方法。

为避免手工输入示例程序，你应该获取解决方案源码发行包。(见附录 A)然后，当一个示例谈及类似“创建一个名为 xyz 的文件，包含以下信息……”的话时，你可以直接使用发行包中对应的文件。本章的脚本位于 api 目录下，除了库文件以外，可以在 lib 目录下找到。

本章示例所使用的表的主要表名为 *profile*。它首次出现在 2.4 节，你应该了解以免跳过了它然后却很迷惑它从何而来。本章后面的一节重置 *profile* 表到后面章节用到的一个已知状态。



提示：这儿讨论的问题可以从命令行运行。要获得调用这儿所涵盖的每种语言所写成的程序的相关指导，可参见附录 B。

假设

要高效地使用本章的材料，你应该确保满足下面的假设：

- 必须为你计划使用的任何语言处理器安装 MySQL 编程支持。如果你要了解如何安装任何 API，可见附录 A。
- 你应该已经设置了一个 MySQL 用户账号以访问服务器和数据库，以用来试验语句。如 1.1 节所述，本书的示例使用用户名和密码分别为 cbuser 和 cbpass 的 MySQL 账号，并且我们将连接到运行在本机的 MySQL 服务器访问名为 cookbook 的数据库。如果你需要创建账号或数据库，可参考该节的指示。
- 这儿的讨论假定你对 API 语言有基本的理解。如果某个解决方案使用你并不熟悉的语言来构建，请参考包括该语言的有用的概念性文字。附录 D 列出了一些或许有用的资源。

- 某些程序的正确执行要求你设置一定的环境变量。参见附录 B，可获取关于设置环境变量的全面信息，参考 2.3 节的应用环境变量搜索库文件的详细说明。

MySQL 客户端 API 架构

所有 MySQL 客户端程序的一个共性是，无论你使用哪种语言，他们都使用某种实现了通信协议的应用编程接口连接到服务器。无论程序目的如何，这一点始终都是正确的，不管它是一个命令行工具也好，是在预定时间自动执行的任务也好，或者是从一个 Web 服务器使用使数据库内容在 Web 上可用的脚本也好。MySQL API 为应用开发者提供了表示数据库操作的标准化方法。每个 API 将你的指令转化为 MySQL 服务器能理解的东西。

服务器本身遵循一种较低级别的协议，我称之为原始协议。在该层次上服务器和它的客户端之间通过网络直接通信。客户端建立到服务器正在侦听的端口的连接，并使用客户服务器协议的最基本的术语与之通信。（基本上，客户端填满数据结构并将他们通过网络推送到服务器。）尝试在此层次上直接与服务器通信效率并不高，写程序来做也是如此。原始协议是一种有效的二进制通信流，但使用很不方便，这是阻止开发者尝试用此方法编写与服务器通信程序的一个因素。更方便的访问 MySQL 服务器的方法是通过原始协议层次之上写就的编程接口。接口处理代表你程序的原始协议的细节，它提供对诸如连接到服务器、发送语句、检索语句结果以及获得语句状态信息等操作的调用。

大多数 MySQL API 并不直接实现原始协议。取而代之的是，他们和被包含在 MySQL 发行包中的 MySQL 客户端库联系在一起并依赖于它。客户端库用 C 写成，因而提供了在 C 程序中与服务器进行通信的接口基础。MySQL 发行包中的大多数标准客户端都用 C 写成，且使用这个 API。你也可以在你自己的程序中使用它，如果你想获得尽可能高效的程序应该考虑那么做。然而，大多数第三方应用开发并不是用 C 来实现的。反之，C 的 API 最常作为其他语言的嵌入库而非直接被使用。这也是 MySQL 通信如何用 Perl、Ruby、PHP、Python 和其他的几种语言来实现的方法。为这些高级语言提供的 MySQL API 被写成一个 C 程序的“封装器”，它们被链接到语言处理器。

这个方法的好处在于它允许语言处理器使用 C 程序代表你和 MySQL 服务器交互，而这个程序提供了你可以更方便地指定数据库操作的接口。例如，诸如 Perl 或 Ruby 等脚本语言

使文本操作更为方便，无须像你在 C 中那样不得不分配字符串缓存或者释放他们。高级语言让你更多专注于你所要做的事情而更少关注你直接用 C 语言编写时所必须考虑的细节。

本书并未涵盖任何 C 的 API 的细节，因为我们从不直接使用它；本书所开发的程序使用构建在 C 的 API 之上的高层次接口，如果你想用 C 来编写 MySQL 客户端程序，下面的信息资源可能会有用：

- MySQL 参考手册包含了一章，描述了 C 的所有 API 的功能。你还应该看一下 MySQL 源码发行包中提供的用 C 写成的标准 MySQL 客户端代码。源码发行包和手册都可以在 MySQL 网站 <http://dev.mysql.com/> 找到，你也可以从 MySQL 出版社获取手册的印刷版本。
- Paul DuBois (Sams) 所著的 MySQL 书籍包含 C 的 API 的参考资料，还包含专门的一章提供了用 C 编写 MySQL 程序的详细教程指南。该章也可在 <http://www.kitebird.com/mysql-book/> 在线查看。本章讨论的示例程序的源代码也可在同一站点找到，供你学习使用。那些程序写出来仅以指导为目的，所以你可以发现他们比 MySQL 源码发行包里的标准客户端程序更容易理解。

MySQL 的 Java 接口并不使用 C 客户端库。他们直接实现原始协议，但在 JDBC 接口之上映射协议实现。你使用标准 JDBC 调用来编写你的 Java 程序，JDBC 将你的数据库操作请求传递给底层 MySQL 接口，该接口将他们转化成使用原始协议与 MySQL 服务器通信的操作。

本书使用的 MySQL 编程接口拥有共同的设计原则：他们都使用两层架构。架构的顶层提供数据库无关的方法，它以可移植的方法实现了数据库访问。这样无论你正使用什么数据库管理系统（无论他是 MySQL、PostgreSQL、Oracle，或者其他的好）表现都相同。较低的层次由一系列驱动组成，每个实现一个特定数据库系统的细节。两层架构使应用程序可以使用一个抽象的接口，该接口不再紧紧绑定在访问任何特定数据库服务器的细节上。这增强了你程序的可移植性，因为你仅需要选择一个不同的较低层次驱动来使用一个不同类型的数据库。当然了，那是理论上的。实际上，完美的可移植性是可望而不可及的：

- 无论你使用的驱动是什么，架构顶层所提供的接口方法都是一致的，但是仍然可能会发出包含仅被某种特定服务器支持的结构的 SQL 语句。对于 MySQL，一个很好的示例是提供数据库和表结构信息的 SHOW 语句。如果你在一个非 MySQL 服务器中使用 SHOW，出现错误就是很可能的结果。

- 较低层次的驱动常常扩展抽象接口使其能更便捷地获得数据库相关的特征。例如，Perl DBI 的 MySQL 驱动使得最新的 AUTO_INCREMENT 值作为数据库句柄的一个属性以便你可以通过 \$dbh->(mysql_insertid) 来访问它。这些特征使你在开始时能更方便地写一个程序，但同时也使其可移植性降低了，在你要将程序移植到使用另一个数据库系统时需要重写部分程序。

尽管这些因素某种程度上有损可移植性，两层架构的普遍可移植性特征为 MySQL 开发者提供了重要的好处。

本书使用的 API 所共有的另一个特性就是他们都是面向对象的。不管你是用 Perl、Ruby、PHP、Python 或 Java 来编写，连接 MySQL 服务器的操作返回一个值，这样可以使你以面向对象的方式来处理语句。例如，当你连接到数据库服务器时，你就获得了一个你可以用于与服务器进一步交互的数据库连接对象。接口还提供其他对象，例如语句对象、结果集对象，或者元数据对象。

现在让我们看看如何使用这些编程接口执行最基本的 MySQL 操作：连接到服务器以及与服务器断开连接。

2.1 连接、选择数据库及断开连接

Connecting, Selecting a Database, and Disconnecting

问题

你需要建立到服务器的连接以访问一个数据库，并且当你完成时需要关闭连接。

解决方案

每个 API 都提供了连接和断开连接的程序。连接程序要求你提供参数指定运行 MySQL 服务器的主机名和你想使用的 MySQL 账户。你还可以选择一个默认的数据库。

讨论

本节的程序展示了怎么执行大多数 MySQL 程序公共的三个基本操作：

建立到 MySQL 服务器的连接

不论你使用哪种 API，使用 MySQL 的每个程序都要做这个。指定连接参数的细节在不同的 API 间有所不同，某些 API 提供了比其他 API 更高的灵活度。然而，存在很

多公共的元素。例如，你必须指定正运行服务器的主机，以及用于访问服务器的 MySQL 账户的用户名和密码。

选择一个数据库

大多数 MySQL 程序选择一个默认的数据库。

和服务器断开连接

每个 API 都提供了关闭打开的连接的方法。最好你用服务器一完成任务就关闭连接，以便能够释放分配给服务连接的资源。否则，如果你的程序在访问了服务器之后执行了额外的计算，连接就会不必要地一直保持打开状态。显示关闭连接更为可取。

如果一个程序在没有关闭连接的情况下简单地中止，MySQL 最终会注意到，且显示关闭连接可以使服务器在程序尾部执行立即关闭的操作。

本节中每个 API 的示例程序展示了如何连接到服务器，选择 `cookbook` 数据库，以及断开连接。

偶尔你会想编写一个不选择数据库的 MySQL 程序。这种情形可能发生在你想发起一条不需要默认数据库的语句时，例如 `SHOW VARIABLES` 或者 `SELECT VERSION()`，又或者你正在编写一个连接到服务器的交互式程序，使用户可以在连接建立之后指定数据库。为涵盖这种情况，每个 API 的讨论也指出了如何在没有选择任何默认数据库的情况下进行连接。

Perl

要用 Perl 编写 MySQL 脚本，你应该先安装好 DBI 模块，还有 MySQL 特定的驱动模块，`DBD::mysql`。如果他们还未安装，附录 A 包含了获取这些模块的信息。

下面是一段连接到 `cookbook` 数据库然后断开连接的简单 Perl 脚本：

```
#!/usr/bin/perl
# connect.pl - 连接到 MySQL 服务器

use strict;
use warnings;
use DBI;

my $dsn = "DBI:mysql:host=localhost;database=cookbook";
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass")
    or die "Cannot connect to server\n";
print "Connected\n";
$dbh->disconnect ();
print "Disconnected\n";
```

要试验脚本，应创建一个包含上述代码的名为 `connect.pl` 的文件，在命令行执行它。（在 Unix 下，如果你的 Perl 程序不在`/usr/bin/perl` 位置，你需要改变脚本首行的路径名。）你应

该能看见程序打印出了两行输出，表明它成功连接和断开连接：

```
% connect.pl  
Connected  
Disconnected
```

如果你需要了解运行 Perl 程序的背景知识，请参考附录 B。

`use strict` 行开启了严格的变量检查，会引起 Perl 抱怨未经声明就被使用的变量。这是一个明智的预防措施，因为它能帮助发现一些未被发现的错误。

`use warnings` 行开启了警告模式，以便 Perl 为任何可导致问题的构建生成警告。我们的示例脚本没有那样的构建，但是形成开启警告以捕获发生在脚本开发过程中的问题的习惯是一个好主意。`use warnings` 类似于指定 Perl `-w` 命令行选项，但在你想看见哪些警告方面提供了更多的控制。（执行 `perldoc warnings` 命令来获取更多信息。）

`use DBI` 语句告知 Perl 程序需要载入 DBI 模块。没有必要显式载入 MySQL 驱动模块 (`DBD::mysql`)，因为当脚本连接到数据库服务器时 DBI 自己就完成了这个任务。

接下来的两行通过设置一个数据源名称 (DSN) 和调用 `DBI connect()` 方法建立了到 MySQL 的连接。`connect()` 的参数是 DSN，MySQL 用户名和密码，以及任何你想指定的连接属性。DSN 是必须的，其他参数是可选的，尽管通常情况下提供一个用户名和密码是必须的。

DSN 指定了使用哪个数据库驱动，其他选项表明连接到哪。对于 MySQL 程序，DSN 格式是 `DBI:mysql:options`。DSN 中的第二个冒号是必须的，即使你并不指定任何选项。

3 个 DSN 部分有如下的含义：

- 第一个部分总是 DBI。它对大小写不敏感；`dbi` 或 `Dbi` 效果相同。
- 第二个部分告知 DBI 使用哪个数据库驱动。对于 MySQL，名称必须是 `mysql`，这里是大小写敏感的。你不能使用 `MySQL`、`MYSQl`，或者任何其他变种。
- 第三个部分，如果存在，是一个分号分隔的名值对列表 (`name = value`)，它制定了额外的连接选项。你所提供的选项对的顺序没有任何关系。出于我们这里所说的目的，最相关的两个选项是主机和数据库。他们指定了 MySQL 服务器正运行的主机名称和你想使用的默认数据库。

给定了这些信息，连接本地主机 `localhost` 上的 `cookbook` 数据库的 DSN 看起来如下所示：

```
DBI:mysql:host=localhost;database=cookbook
```

如果你略去 host 选项，它的默认值是 localhost。因此，这两个 DSN 是等同的：

```
DBI:mysql:host=localhost;database=cookbook  
DBI:mysql:database=cookbook
```

如果你省略 database 选项，connect() 操作不会选择默认的数据库。

connect() 调用的第二和第三个参数是你的 MySQL 用户名和密码。你也可以指定紧跟密码的第四个参数来指定控制发生错误时 DBI 行为的属性。如果没有指定，当错误发生时 DBI 默认会打印出错误信息而不会终止你的脚本。这也是为什么 connect.pl 检查 connect() 是否返回 undef 以指明错误：

```
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass")  
    or die "Cannot connect to server\n";
```

其他的错误处理策略也是可能的。例如，你可以通过在 DBI 调用中禁止 PrintError 属性，启用 RaiseError 替代让 DBI 在发生错误时自动中止脚本。这样你就不需要自己来检查错误（尽管你也丧失了决定你的程序如何从错误中恢复的能力）：

```
my $dbh = DBI->connect ($dsn, $user_name, $password,  
    {PrintError => 0, RaiseError => 1});
```

2.2 节深入讨论了错误处理。

另一个公共属性是 AutoCommit，它设置了用于事务连接的自动提交（auto-commit）模式。在 MySQL 中，对于新连接这是默认开启的，但我们将从这点开始设置它，使初始连接状态明晰：

```
my $dbh = DBI->connect ($dsn, $user_name, $password,  
    {PrintError => 0, RaiseError => 1, AutoCommit => 1});
```

如上所示，connect() 的第四个参数是连接属性名值对哈希表的一个引用。编写这段代码的另一个方法如下：

```
my %conn_attrs = (PrintError => 0, RaiseError => 1, AutoCommit => 1);  
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs);
```

选择你更喜欢的一种方式。本书的脚本使用 %conn_attr 哈希表使 connect() 调用读起来更简单。

如果 connect() 成功，它返回一个包含连接状态信息的数据库句柄。（用 DBI 的说法，对象引用被称为句柄。）后面我们将看到其他的句柄诸如语句句柄，它和特定的语句相关联。本书的 Perl DBI 脚本约定使用 \$dbh 和 \$sth 来表示数据库和语句句柄。

其他的连接参数。对于 localhost 本地连接，你可以在 DSN 中提供 mysql_socket 选项来指定 Unix domain socket 的路径：

```
my $dsn = "DBI:mysql:host=localhost;database=cookbook"
. ";mysql_socket=/var/tmp/mysql.sock";
```

对于非本地 (TCP/IP) 连接, 你可以提供一个 port 选项来指定端口号:

```
my $dsn = "DBI:mysql:host=mysql.example.com;database=cookbook"
. ";port=3307";
```

Ruby

要用 Ruby 来编写 MySQL 脚本, 你应该先安装 DBI 模块, 以及 MySQL 特定的驱动模块。两者都包含在 Ruby DBI 发行包中。如果还没安装, 附录 A 包含了获得 Ruby DBI 的信息。

下面是一段连接到 cookbook 数据库然后断开连接的简单 Ruby 脚本:

```
#!/usr/bin/ruby -w
# connect.rb - 连接到 MySQL 服务器

require "dbi"

begin
  dsn = "DBI:Mysql:host=localhost;database=cookbook"
  dbh = DBI.connect(dsn, "cbuser", "cbpass")
  puts "Connected"
rescue
  puts "Cannot connect to server"
  exit(1)
end
dbh.disconnect
puts "Disconnected"
```

要试验这段脚本, 创建一个包含上述代码的名为 connect.rb 的文件。(在 Unix 下, 如果你的 Ruby 程序不在 /usr/bin/ruby 位置, 你需要改变脚本首行的路径名。) 你应该能看见程序打印出了两行输出表明它成功连接和断开连接:

```
% connect.rb
Connected
Disconnected
```

如果你需要执行 Ruby 程序的背景知识, 请参考附录 B。

-w 选项开启警告模式以便 Ruby 为任何有问题的构建生成警告。我们的示例脚本没有那样的构建, 但是形成使用 -w 来捕获发生在脚本开发过程中的问题的习惯是一个好主意。

require 语句告知 Ruby 程序需要载入 DBI 模块。没有必要显式载入 MySQL 驱动模块, 因为当脚本连接到数据库服务器时 DBI 自己就完成了这个任务。

将数据源名称和MySQL用户名和密码传给 `connect()` 方法就建立了连接。DSN 是必须的。其他参数是可选的，尽管通常情况下必须提供一个用户名和密码。

DSN 指定了使用哪个数据库驱动，其他选项表明连接到哪。对于 MySQL 程序，DSN 为如下格式之一：

```
DBI:MySQL:db_name:host_name  
DBI:MySQL:name=value;name=value ...
```

DSN 组成部分有如下含义：

- 第一部分总是 DBI 或 dbi；
- 第二部分告知 DBI 使用哪个数据库驱动。对于 MySQL，名称为 MySQL；
- 如果有第三部分，它是一个以冒号分隔的数据库名称和主机名，或者是以分号分隔的指定其他连接选项的名值对列表。出于我们这里的目的，最相关的两个选项是主机和数据库。他们指定了 MySQL 服务器正运行的主机名称和你想使用的默认数据库。在 Perl DBI 中，DSN 中的第二个冒号是必需的，即使你并不指定任何选项。

给定了这些信息，连接本地主机 `localhost` 上的 `cookbook` 数据库的 DSN 看起来如下所示：

```
DBI:mysql:host=localhost;database=cookbook
```

如果你略去 `host` 选项，它的默认值是 `localhost`。因此，这两个 DSN 是等同的：

```
DBI:mysql:host=localhost;database=cookbook  
DBI:mysql:database=cookbook
```

如果你省略 `database` 选项，`connect()` 操作不会选择默认的数据库。

如果 `connect()` 成功，它返回一个包含连接状态信息的数据库句柄。本书中的 Ruby DBI 脚本习惯使用 `dbh` 来表示数据库句柄。

如果 `connect()` 方法失败，检测不到特定的返回值。问题发生时 Ruby 程序会抛出异常。要处理错误，应将可能失败的语句放进一个 `begin` 块中，然后使用 `rescue` 子句来包含错误处理代码。脚本顶层发生的异常（也就是，在任何 `begin` 块外的）被默认的异常处理器所捕获，它输出一系列栈痕迹，然后退出。

其他连接参数。对于 `localhost` 本地连接，你可以在 DSN 中提供 `socket` 选项来指定 Unix domain socket 的路径：

```
dsn = "DBI:MySQL:host=localhost;database=cookbook" +  
      ";socket=/var/tmp/mysql.sock"
```

对于非本地 (TCP/IP) 连接，你可以提供一个 port 选项来指定端口号：

```
dsn = "DBI:mysql:host=mysql.example.com;database=cookbook" +
      ";port=3307"
```

PHP

要编写使用 MySQL 的 PHP 脚本，你的 PHP 解释器必须已经编译进了 MySQL 支持。如果没有，你的脚本将不能连接到你的 MySQL 服务器。发生该情况时，检查一下包含在你的 PHP 发行包中的指导来看看如何开启 MySQL 支持。

PHP 实际上有两种扩展来支持 MySQL 使用。第一种，`mysql` 是原始的 MySQL 扩展。它提供了一系列名称以 `mysql_` 开头的函数。第二种，`mysqli`，或者“MySQL improved，”提供了名称以 `mysqli_` 开头的函数。在本书中，你可以使用其中任何一个扩展，但我推荐使用 `mysqli`。

在任何情况下，本书中的 PHP 脚本并不直接使用任一种扩展。相反，他们只用 PHP 扩展和附加仓库 (PEAR) 中的 DB 模块。PEAR DB 模块提供了你所决定使用的底层 MySQL 扩展的接口。这就意味着除了你所选择的 PHP MySQL 扩展外，还必须安装 PEAR。如果尚未安装，附录 A 包含了获取 PEAR 的相关信息。

PHP 脚本通常用于 Web 服务器中。我将假设如果你以该方式使用 PHP，你可以简单地拷贝 PHP 脚本到你服务器的文档树下，在浏览器中请求他们，他们就会执行。例如，如果你在本地主机 `localhost` 以 Apache 作为 Web 服务器，你在 Apache 文档树的最顶层安装了一个 PHP 脚本 `myscript.php`，你应该可以通过请求 URL `http://localhost/myscript.php` 来访问该脚本。

本书使用.php 扩展（后缀名）作为 PHP 脚本文件名称，所以必须要对你的 Web 服务器进行配置以识别.php 扩展。否则，当你在浏览器中请求一个 PHP 脚本时，服务器只会简单地发送脚本的文本内容，这也就是你在浏览器窗口中所将见到的。如果你不希望发生这种情况，特别是在脚本中包含你用来连接到 MySQL 的用户名和密码时。要获得配置 Apache 以使用 PHP 的信息，请参考 17.2 节。

PHP 脚本常常混合使用了 PHP 代码和 HTML，PHP 代码嵌入在特殊的`<?php` 和`?>`标签中。下面是一个简单的示例：

```
<html>
<head><title>A simple page</title></head>
<body>
<p>
<?php
```

```
    print ("I am PHP code, hear me roar!\n");
?>
</p>
</body>
</html>
```

也可以配置 PHP 使其识别“短格式的”标签，通常写成<?和?>形式。本书假设你并不支持短格式标签，所以这儿的 PHP 脚本都不使用他们。

下面是一段连接到 cookbook 数据库然后断开连接的简单 PHP 脚本：

```
<?php
# connect.php - 连接到 MySQL 服务器

require_once "DB.php";

$dsn = "mysqli://cbuser:cblpass@localhost/cookbook";
$conn =& DB::connect ($dsn);
if (PEAR::isError ($conn))
    die ("Cannot connect to server\n");
print ("Connected\n");
$conn->disconnect ();
print ("Disconnected\n");
?>
```

为简便起见，当我展示包含完整 PHP 代码的示例时，我将会略去包围的<?php 和?>标签。（也就是说，如果你在某个 PHP 示例中没有看见标签，可以假想<?php 和?>围绕在所显示的代码块之外。）在 HTML 和 PHP 代码之间切换的示例一定包含标签，使 PHP 代码和非 PHP 代码之间界限分明。

require_once 语句访问使用 PEAR DB 模块所需要的 DB.php 文件。require_once 只是几个 PHP 文件包含语句之一：

- include 指引 PHP 读取命名文件。require 类似于 include，即使 require 发生在从未执行的控制结构中，PHP 也会读取该文件（例如一个条件从不为 true 的 if 块）。
- include_once 和 require_once 类似于 include 和 require，只是如果文件已经被读取过，那就不会再次进行处理。这对于避免常发生于库文件包含其他库文件的情形下的多次声明问题很有用。

\$dsn 是表示如何连接到数据库服务器的数据源名称。它的通用语法如下：

phptype://user_name:password@host_name/db_name

phptype 值是 PHP 驱动类型。对于 MySQL 它应该是 mysql 或 mysqli 以表示使用哪个 MySQL 扩展。你可以选择任一种，只要你的 PHP 解释器已经编译进了所选的扩展。

PEAR DB connect()方法使用 DSN 连接到 MySQL。如果连接尝试成功了，connect() 返回一个连接对象，它能被用于访问其他的 MySQL 相关方法。本书的 PHP 脚本约定使用 \$conn 来表示连接对象。

如果连接尝试失败，connect()返回一个错误 (error) 对象。要确定返回对象是否代表一个错误，可使用 PEAR::isError()方法。

注意 connect()结果的赋值符使用=&操作符而不是=<操作符。=&将一个引用赋给返回值，而=<创建值的一份拷贝。在此上下文中，=<将会创建并不需要的另一个对象。(本书的 PHP 脚本通常使用=&来赋值连接尝试的结果，但在 20.3 节有一个使用=<的实例，以确保被赋值的连接对象比它所发生的函数调用更持久。)

前面示例所示的 DSN 的最后一部分是数据库名。如果不选择一个默认数据库来连接，只要将其从 DSN 尾部略去即可：

```
$dsn = "mysqli://cbuser:cbpass@localhost";
$conn = & DB::connect ($dsn);
```

要试验 connect.php 脚本，将其拷贝到你 Web 服务器的文档树下，使用你的浏览器来请求它。如果你有能从命令行执行的 PHP 解释器的独立版本，你可以在没有 Web 服务器或浏览器的情况下试验该脚本：

```
% php connect.php
Connected
Disconnected
```

如果你需要了解执行 PHP 程序的相关背景知识，请参考附录 B。

作为用字符串格式指定 DSN 的一种替代，你可以使用数组来提供连接参数：

```
$dsn = array
(
    "phptype" => "mysqli",
    "username" => "cbuser",
    "password" => "cbpass",
    "hostspec" => "localhost",
    "database" => "cookbook"
);
$conn = & DB::connect ($dsn);
if (PEAR::isError ($conn))
    print ("Cannot connect to server\n");
```

要使用数组格式 DSN 而不选择默认数据库连接，从数据中略去 database 成员即可。

其他连接参数。要使用特定的 Unix domain socket 文件或者 TCP/IP 端口号，在连接时修改参数。下面的两个示例使用数组格式 DSN 来完成。

对于 localhost 连接，你可以通过在 DSN 数组中包含一个 socket (套接字) 成员来指定 Unix domain socket 的路径名：

```
$dsn = array
(
    "phptype" => "mysqli",
    "username" => "cbuser",
    "password" => "cbpass",
    "hostspec" => "localhost",
    "socket" => "/var/tmp/mysql.sock",
    "database" => "cookbook"
);
$conn = & DB::connect ($dsn);
if (PEAR::isError ($conn))
    print ("Cannot connect to server\n");
```

对于非 localhost (TCP/IP) 连接，你可以通过在 DSN 数组中包含一个 port (端口) 成员来指定端口号：

```
$dsn = array
(
    "phptype" => "mysqli",
    "username" => "cbuser",
    "password" => "cbpass",
    "hostspec" => "mysql.example.com",
    "port" => 3307,
    "database" => "cookbook"
);
$conn = & DB::connect ($dsn);
if (PEAR::isError ($conn))
    print ("Cannot connect to server\n");
```

你可以使用 PHP 初始化文件 (通常名为 `php.ini`) 来指定一个默认的主机名、用户名、密码、套接字路径，或者端口号。对于 `mysql` 扩展，设置 `mysql.default_host`、`mysql.default_user`、`mysql.default_password`、`mysql.default_socket`，或者 `mysql.default_port` 配置变量的值。对于 `mysqli`，相应的变量名以 `mysqli` 开头 (密码变量是 `mysql_default_pw`)。这些变量在全局范围内影响 PHP 脚本：对于没有指定那些参数的脚本，使用 `php.ini` 中的默认值。

Python

要用 Python 来编写 MySQL 程序，你需要为 Python 的 DB-API 接口提供 MySQL 连接的 `MySQLdb` 模块。如果没有安装该模块，附录 A 包含了获得 `MySQLdb` 的信息。

要使用 DB-API 接口，导入你想使用的数据库驱动模块 (对于 MySQL 程序是 `MySQLdb`)。然后通过调用驱动的 `connect()` 方法创建一个数据库连接对象。这个对象提供了访问其他 DB-API 方法的途径，例如为到数据库服务器的连接提供服务的 `close()` 方法。下面是一个简短的 Python 程序，`connect.py`，阐述了这些操作：

```
#!/usr/bin/python
# connect.py - 连接到 MySQL 服务器

import sys
import MySQLdb

try:
    conn = MySQLdb.connect (db = "cookbook",
                           host = "localhost",
                           user = "cbuser",
                           passwd = "cbpass")
    print "Connected"
except:
    print "Cannot connect to server"
    sys.exit (1)

conn.close ()
print "Disconnected"
```

要试验这个脚本，创建一个包含上述代码的名为 `connect.py` 的文件。（在 Unix 下，如果你的 Python 程序不在`/usr/bin/python` 位置，需要改变脚本首行的路径名。）你应该能看见程序打印出了两行输出表明它成功连接和断开连接：

```
% connect.py
Connected
Disconnected
```

如果你需要执行 Python 程序的背景知识，请参考附录 B。

`import` 行给予脚本访问 `sys` 模块（为 `sys.exit()` 方法所需要）和 `MySQLdb` 模块的能力。然后脚本通过调用 `connect()` 获得一个连接对象 `conn` 来建立到 MySQL 服务器的连接。本书的 Python 脚本习惯使用 `conn` 来表示连接对象。

如果 `connect()` 方法失败，没有特殊的返回值供检测。当发生问题时 Python 程序抛出异常。要处理错误，应将可能失败的语句放入一个 `try` 语句，然后使用一个 `except` 子句来包含错误处理代码。发生在脚本顶层的异常（也就是说，在任何 `try` 语句之外的）被默认的异常处理器捕获，它会输出一个堆路径然后退出。

因为 `connect()` 调用使用命名参数，他们的顺序并不重要。如果你从 `connect()` 调用中略去 `host` 参数，它的默认值是 `localhost`。如果你略去 `db` 参数或者传给其一个`" "` 值（空字符串），`connect()` 操作不选择默认数据库。然而如果你传了一个 `None` 值，调用将会失败。

其他连接参数。对于 `localhost` 本地连接，你可以提供一个 `unix_socket` 参数来指定到 Unix domain socket 文件的路径：

```
conn = MySQLdb.connect (db = "cookbook",
                       host = "localhost",
                       unix_socket = "/var/tmp/mysql.sock",
```

```
user = "cbuser",
passwd = "cbpass")
```

对于非本地主机 (TCP/IP) 连接, 你可以提供一个 port 参数来指定端口号:

```
conn = MySQLdb.connect (db = "cookbook",
                       host = "mysql.example.com",
                       port = 3307,
                       user = "cbuser",
                       passwd = "cbpass")
```

Java

Java 中的数据库程序使用 JDBC 接口以及你想访问的特定数据库引擎的驱动写成。也就是说,JDBC 架构提供了和数据库特定驱动协力使用的通用接口。Java 类似于 Ruby 和 Python, 因为你不需要对测试特定的方法调用进行测试来获取表示错误的返回值。相反, 当抛出异常时, 你提供了需要调用的处理器。

Java 编程需要一个软件开发包 (SDK), 然后要将你的 JAVA_HOME 环境变量设为你 SDK 的安装位置。要编写基于 MySQL 的 Java 程序, 你还需要一个 MySQL 特定的 JDBC 驱动。本书中的程序使用 MySQL Connector/J, 是 MySQL AB 所提供的驱动。如果尚未安装, 附录 A 有获取 MySQL Connector/J 的相关信息。附录 B 有获取 SDK 及设置 JAVA_HOME 的信息。

下面的 Java 程序 Connect.java 说明了如何连接到 MySQL 服务器, 断开连接, 以及选择 cookbook 作为默认数据库:

```
// Connect.java - 连接到 MySQL 服务器

import java.sql.*;

public class Connect
{
    public static void main (String[] args)
    {
        Connection conn = null;
        String url = "jdbc:mysql://localhost/cookbook";
        String userName = "cbuser";
        String password = "cbpass";

        try
        {
            Class.forName ("com.mysql.jdbc.Driver").newInstance ();
            conn = DriverManager.getConnection (url, userName, password);
            System.out.println ("Connected");
        }
        catch (Exception e)
        {
            System.err.println ("Cannot connect to server");
            System.exit (1);
        }
    }
}
```

```
        }
        if (conn != null)
        {
            try
            {
                conn.close ();
                System.out.println ("Disconnected");
            }
            catch (Exception e) { /* 忽略 close 错误 */ }
        }
    }
}
```

import java.sql.*语句引用了一些类和接口，这些类和接口提供了对你用于管理与数据库服务器交互的不同方面的数据类型的访问。这些是所有的 JDBC 程序都需要的。

连接到服务器的过程分两步。首先，调用 Class.forName() 注册 JDBC 数据库驱动。Class.forName() 方法需要一个驱动器名称；对于 MySQL Connector/J，使用 com.mysql.jdbc.Driver。然后调用 DriverManager.getConnection() 来初始化连接并获取维护连接状态信息的连接对象。本书的 Java 程序约定使用 conn 来表示连接对象。

DriverManager.getConnection() 有三个参数：描述连接到哪儿以及要使用的数据库的 URL、MySQL 用户名以及密码。URL 字符串格式如下：

```
jdbc:driver://host_name/db_name
```

这个格式遵循 Java 的约定，连接到一个网络资源的 URL 以协议描述符开头。对于 JDBC 程序，协议是 jdbc，你还需要一个指定驱动名称的子协议描述符（对于 MySQL 程序是 mysql）。连接 URL 的许多部分是可选的，但打头的协议和子协议描述符不是。如果你略去 host_name，默认的主机值是 localhost。如果你略去数据库名称，连接操作不选择默认的数据库。然而，在任何情况下你都不应该略去任意的斜线。例如，不选择默认数据库连接到本地主机，URL 是：

```
jdbc:mysql:///
```

要试验程序，编译并执行它。class 语句说明了程序的名称，在这里是 Connect。包含程序的文件名应该和此名称一致，并包括一个.java 扩展名，所以此程序的文件名是 Connect.java（注 1）。使用 javac 编译程序：

```
% javac Connect.java
```

如果你选择其他的 Java 编译器，用对应的名称替换 javac 即可。

注 1：如果你拷贝一份 Connect.java 作为一个新程序的基础，你需要修改 class 语句里的类名，使其和你新文件的名称一致。

Java 编译器生成被编译的字节代码来产生一个名为 Connect.class 的类文件。使用 Java 程序执行类型文件（不需要指定.class 扩展名）：

```
% java Connect  
Connected  
Disconnected
```

在编译执行示例程序之前你需要设置你的 CLASSPATH 环境变量。CLASSPATH 值至少应该包括你的当前目录（.）以及 MySQL Connector/J JDBC 驱动的路径。如果你需要运行 Java 程序或设置 CLASSPATH 的背景知识，可参考附录 B。

小心 Class.forName()!

示例程序 Connect.java 以如下方式注册 JDBC 驱动：

```
Class.forName ("com.mysql.jdbc.Driver").newInstance ();
```

你应该能不调用 newInstance() 来注册驱动，如下：

```
Class.forName ("com.mysql.jdbc.Driver");
```

然而，这个调用对于有些 Java 实现并不能正常工作，所以确定使用 newInstance()，否则你就会发现你会制定新的 Java 格言：“一次编写，到处调试。”

某些 JDBC 驱动（MySQL Connector/J 也在其中）允许你在 URL 的末尾指定用户名和密码作为参数。在这种情况下，你略去了 getConnection() 调用的第二和第三个参数。使用该 URL 方式，示例程序中建立连接的代码可以被写成如下形式：

```
// 使用包含在 URL 中的用户名和密码进行连接  
Connection conn = null;  
String url = "jdbc:mysql://localhost/cookbook?user=cbuser&password=cbpass";  
  
try  
{  
    Class.forName ("com.mysql.jdbc.Driver").newInstance ();  
    conn = DriverManager.getConnection (url);  
    System.out.println ("Connected");  
}
```

分隔用户名和密码参数的字符应该是&，而非;。

其他的连接参数。 MySQL Connector/J 不支持 Unix domain socket 文件连接，所以即使主机名是 localhost 的连接也是通过 TCP/IP 建立的。你可以通过在连接 URL 中添加 :port_num 到主机名来指定一个显式的端口号：

```
String url = "jdbc:mysql://mysql.example.com:3307/cookbook";
```

2.2 查错

Checking for Errors

问题

程序出错了，但你不知道是什么错。

解决方案

每个人都有使程序正常工作的问题。但是如果你没有预计到查错的困难，你会让工作更艰苦。添加一些错误检查代码以便你的程序能帮你指明哪些地方出错了。

讨论

在读完 2.1 节后，你现在知道如何连接到 MySQL 服务器了。了解如何查错以及如何从 API 中获取特定的错误信息也是一个好主意，所以这是我们接下来将要介绍的。你可能急于去了解如何做更有趣的事情（例如发起语句和取回结果），但是错误检查是非常重要的。程序时常会失败，特别是在开发时，如果你不知道如何确定问题为什么会发生，你就会变得非常盲目。

发生错误时，MySQL 提供了三个值：

- 一个 MySQL 特定的错误号；
- 一个 MySQL 特定的描述性文本错误信息；
- 遵循 ANSI 和 ODBC 标准的五位长字符值的 SQLSTATE 错误代码。

本节的不同部分展示了如何访问这些信息。本书后面的大多数章节在显示错误信息时仅输出 MySQL 特定值，但本节还揭示了如何访问 SQLSTATE 值。

示例程序说明了如何检查错误但事实上如果你的 MySQL 账户设置合理，执行起来不会有任何问题。因此，你需要稍微修改一下示例以强制使其出现错误来触发错误处理语句。这很容易做到。例如，你可改变建立连接调用，提供一个错误的密码。

一个不和任何特定 API 相关的通用调试是检查 MySQL 服务器的查询日志以查看服务器实际上接收了哪些语句。（这需要你开启查询日志功能并且你有访问日志的权限。）查询日志常给你展示特定的难以看懂的语句，并为你提供线索——你的程序并未构建合适的语句字符串。如果你正在 Web 服务器下运行一个脚本且失败了，请检查 Web 服务器的错误日志。

不要搬起石头砸自己的脚（弄巧成拙）：查错

查错的原则不像某些人期许的那样广为人所重视。许多发送到 MySQL 相关邮件列表里面的消息都是请求帮忙解决由于未知的原因而失败的程序问题。在众多的案例里，这些人为他们程序迷惑的原因是他们从不编写错误检查，因而他们也没有办法获悉是否有问题或者查出到底是什么问题！你不能容忍自己也按这种方式工作。通过查错可以为失败做些准备，以便当错误发生时你可以采取合适的应对。

Perl

DBI 模块提供了两个属性，可以控制当 DBI 方法调用失败时发生什么：

- `PrintError`, 如果允许，会引起 DBI 使用 `warn()` 输出一条错误信息。
- `RaiseError`, 如果允许，会引起 DBI 使用 `die()` 输出一条错误信息。这会中止你的脚本运行。

默认情况下，`PrintError` 是允许的，而 `RaiseError` 是禁止的，所以错误发生时在输出信息后脚本会继续执行。两个属性中任一个或者两个都可在 `connect()` 调用里指定。将属性分别设为 1 或 0 可以开启或禁止它。要指定一个或两个属性，将它们放入一个哈希引用里作为第四个参数传给 `connect()` 调用。

下面的代码仅设置了 `AutoCommit` 属性并使用默认的设置用于错误处理。如果 `connect()` 调用失败，这会产生一条警告信息，但脚本会继续执行：

```
my %conn_attrs = (AutoCommit => 1);
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass", \%conn_attrs);
```

然而，因为如果连接尝试失败时你实际上并不能做什么，通常在 DBI 输出一条信息后退出就行了：

```
my %conn_attrs = (AutoCommit => 1);
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass", \%conn_attrs)
or exit;
```

要输出你自己的错误信息，禁止 `RaiseError`，同时禁止 `PrintError`。然后自己测试一下 DBI 方法调用的结果。当某方法失败时，`$DBI::err`, `$DBI::errstr`, 以及 `$DBI::state` 变量分别包含了 MySQL 错误号，一个描述性的错误字符串，以及 SQLSTATE 值：

```
my %conn_attrs = (PrintError => 0, AutoCommit => 1);
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass", \%conn_attrs)
or die "Connection error:
        \"$DBI::errstr ($DBI::err/$DBI::state)\n\";
```

如果没有发生错误，`$DBI::err` 将会是 0 或 `undef`，`$DBI::errstr` 将会是空字符串或 `undef`，`$DBI::state` 将会是空或 00000。

当你查错时，在调用设置这些变量的 DBI 方法后立即访问它们。如果你在使用它们之前调用了另一个方法，他们的值将会被重置。

如果你正输出你自己的信息，默认的设置（`PrintError` 允许，`RaiseError` 禁止）并不是很有用。在这种情况下，DBI 自动输出一条信息，然后你的脚本输出自己的信息。这是最多的冗余，也是最坏的情况——迷惑了正使用脚本的人。

如果你启用了 `RaiseError`，你可以调用 DBI 方法而不需要检查表示错误的返回值。如果某方法失败，DBI 输出一条错误然后中止你的脚本。如果方法返回了，你可以假定它已经成功执行。这对于脚本编写者来说是最简单的方法：让 DBI 来处理所有的错误检查！然而，如果 `PrintError` 和 `RaiseError` 都启用了，DBI 可能接连调用 `warn()` 和 `die()`，这就导致错误信息被输出两次。为避免此问题，最好在你启用 `RaiseError` 的时候就禁止 `PrintError`。这就是本书一般所使用的方法，如下所述：

```
my %conn_attrs = (PrintError => 0, RaiseError => 1, AutoCommit => 1);
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass", \%conn_attrs);
```

如果你既不想启用 `RaiseError` 进行自动错误检查，也不想所有的检查都由自己来做，你可以采取混合式的方法。可以有选择地启用或禁止不同的句柄的 `PrintError` 和 `RaiseError` 属性。例如，当你调用 `connect()` 时，你可以开启 `RaiseError` 在全局范围内启用 `RaiseError`，然后在每个句柄基础上选择性地禁止它。假设你有一个从命令行参数读取用户名和密码的脚本，然后当用户输入要执行的语句时进行循环。在这种情况下，如果连接失败你可能希望 DBI “死掉”并自动输出错误信息（如果用户没有提供正确的用户名和密码你什么也做不了）。在连接后，从另一方面来说，你不希望仅因为用户输入了一个语法上不正确的语句脚本就退出了。脚本捕获该错误，输出信息然后进行循环获得下一条语句会更好一些。下面的代码揭示了这些是怎么做到的。在示例中使用的 `do()` 方法执行一条语句然后返回 `undef` 以表示一个错误：

```
my $user_name = shift (@ARGV);
my $password = shift (@ARGV);
my %conn_attrs = (PrintError => 0, RaiseError => 1, AutoCommit => 1);
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs);
$dbh->{RaiseError} = 0; # 禁止错误时自动中止脚本
print "Enter queries to be executed, one per line; terminate with Control-D\n";
while (<>)                      # 读取并执行查询
{
    $dbh->do ($) or warn "Query failed: $DBI::errstr ($DBI::err)\n";
}
$dbh->{RaiseError} = 1; # 重新启用错误时自动中止脚本
```

如果启用了 `RaiseError`, 你可以通过在一个 `eval` 块里执行代码来捕获错误而无需中止你的程序。如果块中发生错误, `eval` 会失败并在 `$@` 变量中返回一条消息。有代表性地, 你可以以如下方式来使用 `eval`:

```
eval
{
    # 可能失败的语句放在这儿...
};

if ($@)
{
    print "An error occurred: $@\n";
}
```

本技术一般用于实现事务。要获得示例, 详见第 15.4 节。

混合使用 `eval` 和 `RaiseError`, 不同于单独使用 `RaiseError`, 表现在以下方面:

- 错误仅仅中断 `eval` 块, 而不是整个脚本。
- 任何错误都会中断 `eval` 块, 而 `RaiseError` 仅适用于 DBI 相关的错误。

当你启用 `RaiseError` 使用 `eval` 时, 确保禁止了 `PrintError`。否则, 在 DBI 的某些版本中, 错误仅会引起调用 `warn()` 而不是如你所期望的中断 `eval` 块。

除了使用错误处理属性 `PrintError` 和 `RaiseError` 外, 你可以通过开启 DBI 的跟踪机制获得大量关于你脚本执行的有用信息。以一个表示跟踪级别的参数来调用 `trace()` 方法。级别 1 到 9 以递增的输出详细程度来启用跟踪, 而级别 0 则是禁止跟踪:

```
DBI->trace (1);      # 开启跟踪机制, 最小化输出
DBI->trace (3);      # 提高跟踪级别
DBI->trace (0);      # 禁止跟踪机制
```

单独的数据库和语句句柄也有 `trace()` 方法。这就意味着你可以局部跟踪你所想的单一语句。

跟踪输出通常显示在你的终端 (或者在 Web 脚本情形下, 在你的 Web 服务器的错误日志里)。你可以通过提供表示文件名的第二个参数将跟踪输出写入特定的文件中:

```
DBI->trace (1, "/tmp/trace.out");
```

如果跟踪文件已经存在, 跟踪输出被附加到尾部; 文件的内容并不会先清除。要小心在开发脚本的时候开启了文件跟踪, 但在当你将脚本放入生产环境时却忘记了关闭跟踪。最后, 令你懊恼的是, 你会发现跟踪文件已变得相当大。或者更糟糕的是, 文件系统已被填满, 你却毫无头绪, 不知道为什么!

Ruby

Ruby 通过抛出异常来发出错误信号，然后 Ruby 程序通过在 `begin` 块的 `rescue` 子句里捕获异常来处理错误。Ruby DBI 方法在失败时抛出异常，并且通过 `DBI::DatabaseError` 对象提供了错误信息。要获得 MySQL 错误号，错误信息以及 SQLSTATE 值，访问该对象的 `err`、`errstr` 和 `state` 方法。下面的示例揭示如何在 DBI 脚本中捕获异常并访问错误信息：

```
begin
  dsn = "DBI:Mysql:host=localhost;database=cookbook"
  dbh = DBI.connect(dsn, "cbuser", "cbpass")
  puts "Connected"
rescue DBI::DatabaseError => e
  puts "Cannot connect to server"
  puts "Error code: #{e.err}"
  puts "Error message: #{e.errstr}"
  puts "Error SQLSTATE: #{e.state}"
  exit(1)
end
```

PHP

PEAR DB 方法通过它的返回值来表示成功或失败。如果方法失败，返回值是一个错误对象。如果方法成功，返回值如下：

- `connect()`方法返回与数据库服务器交互的连接对象。
- 执行 SQL 语句的 `query()`方法为诸如返回行的 `SELECT` 语句等返回一个结果集对象，或者为修改行的 `INSERT`、`UPDATE`，或 `DELETE` 语句等返回 `DB_OK` 值。

要确定方法返回值是否是一个错误对象，将其传给 `PEAR::isError()`方法，检查 `PEAR::isError()` 结果，然后采取相应的动作。例如，下面的代码在 `connect()` 成功时输出“Connected”，失败时出现一个通用错误信息，然后退出：

```
$dsn = "mysqli://cbuser:cbpass@localhost/cookbook";
$conn =& DB::connect ($dsn);
if (PEAR::isError ($conn))
  die ("Cannot connect to server\n");
print ("Connected\n");
```

要获得 PEAR DB 方法失败时的更多相关信息，使用错误对象提供的方法：

- `getCode()` 和 `getMessage()` 分别返回一个错误号和错误信息。PEAR 提供了非 MySQL 特定的标准值。
- `getUserInfo()` 和 `getDebugInfo()` 返回 MySQL 特定信息。

下面的列表展示了连接错误发生时，每个方法如何显示 PEAR DB 返回的错误信息：

```
$dsn = "mysqli://cbuser:cbpass@localhost/cookbook";
$conn =& DB::connect ($dsn);
if (PEAR::isError ($conn))
{
    print ("Cannot connect to server.\n");
    printf ("Error code: %d\n", $conn->getCode ());
    printf ("Error message: %s\n", $conn->getMessage ());
    printf ("Error debug info: %s\n", $conn->getDebugInfo ());
    printf ("Error user info: %s\n", $conn->getUserInfo ());
    exit (1);
}
```

Python

Python 通过抛出异常来发出报错信号，Python 程序通过在 `try` 语句的 `except` 子句中捕获异常来处理错误。要获得 MySQL 特定的报错信息，命名一个异常类，并提供一个接受信息的变量。下面是示例：

```
try:
    conn = MySQLdb.connect (db = "cookbook",
                           host = "localhost",
                           user = "cbuser",
                           passwd = "cbpass")
    print "Connected"
except MySQLdb.Error, e:
    print "Cannot connect to server"
    print "Error code:", e.args[0]
    print "Error message:", e.args[1]
    sys.exit (1)
```

发生异常时，`e.args` 的第一和第二个元素分别被设为错误号和错误信息。（注意 `Error` 类通过 `MySQLdb` 驱动模块名访问。）

Java

Java 程序通过捕获异常来处理错误。如果你只想做最少的工作，输出一个栈路径来通知用户问题在哪儿：

```
try
{
    /* ... 一些数据库操作... */
}
catch (Exception e)
{
    e.printStackTrace ();
}
```

方法调用序列揭示了问题所在但并不是必须指出问题是什么。对于程序开发者而言，它也许并非所有有意义的东西。更特别的是，你可以输出和异常关联的错误信息和代码：

- 所有的 Exception 对象支持 getMessage() 方法。JDBC 方法可能使用 SQLException 对象抛出异常；这些类似于 Exception 对象，而且还支持 getErrorCode() 和 getSQLState() 方法。getErrorCode() 和 getMessage() 分别返回 MySQL 特定的错误号和消息字符串。getSQLState() 返回包含 SQLSTATE 值的字符串。
- 你也可以获得关于非致命警告的信息，有些方法使用 SQLWarning 对象生成这些警告。SQLWarning 是 SQLException 的一个子类，但是警告是被堆放在一个列表中而不是立即抛出，所以它们不会中断你的程序，你可以随意输出它们。

下面的示例程序，Error.java，阐述了如何通过输出它能获得的所有错误信息来访问错误消息。它试图连接到 MySQL 服务器上，并在连接尝试失败时输出异常信息。接着它发起一条语句并在语句失败时输出异常和警告信息：

```
// Error.java - 说明 MySQL 的错误处理
import java.sql.*;

public class Error
{
    public static void main (String[] args)
    {
        Connection conn = null;
        String url = "jdbc:mysql://localhost/cookbook";
        String userName = "cbuser";
        String password = "cbpass";

        try
        {
            Class.forName ("com.mysql.jdbc.Driver").newInstance ();
            conn = DriverManager.getConnection (url, userName, password);
            System.out.println ("Connected");
            tryQuery (conn); // 发起一次查询
        }
        catch (Exception e)
        {
            System.err.println ("Cannot connect to server");
            System.err.println (e);
            if (e instanceof SQLException) // JDBC 相关异常?
            {
                // 输出一般性信息，以及数据库相关的任意信息
                // (e 必须从 Exception 转型为 SQLException 以访问 SQLException 的特定
                // 方法)
                System.err.println ("SQLException: " + e.getMessage ());
                System.err.println ("SQLState: " + e.getSQLState ());
            }
        }
    }
}
```

```

        + ((SQLException) e).getSQLState ());
System.err.println ("VendorCode: "
        + ((SQLException) e).getErrorCode ());
    }
}
finally
{
    if (conn != null)
    {
        try
        {
            conn.close ();
            System.out.println ("Disconnected");
        }
        catch (SQLException e)
        {
            // 输出一般性信息, 以及数据库相关的任意信息
            System.err.println ("SQLException: " + e.getMessage ());
            System.err.println ("SQLState: " + e.getSQLState ());
            System.err.println ("VendorCode: " + e.getErrorCode ());
        }
    }
}
}

public static void tryQuery (Connection conn)
{
    try
    {
        // 发起一次简单的查询
        Statement s = conn.createStatement ();
        s.execute ("USE cookbook");
        s.close ();

        // 输出聚集的所有警告
        SQLWarning w = conn.getWarnings ();
        while (w != null)
        {
            System.err.println ("SQLWarning: " + w.getMessage ());
            System.err.println ("SQLState: " + w.getSQLState ());
            System.err.println ("VendorCode: " + w.getErrorCode ());
            w = w.getNextWarning ();
        }
    }
    catch (SQLException e)
    {
        // 输出一般性信息, 以及数据库相关的任意信息
        System.err.println ("SQLException: " + e.getMessage ());
        System.err.println ("SQLState: " + e.getSQLState ());
        System.err.println ("VendorCode: " + e.getErrorCode ());
    }
}
}
}

```

2.3 编写库文件

Writing Library Files

问题

你注意到在多个程序中编写同样的代码来执行公共的操作。

解决方案

将执行那些操作的例行程序放在一个库文件中，让你的程序来访问库文件。然后仅需编写该代码一次。你可能需要设置一个环境变量以便你的脚本能够找到库。

讨论

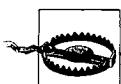
本节描述如何将公共操作代码放入库文件中。封装（或模块化）并不是一个“解决方案”而是一项编程技术。它的主要好处是你不用再在你编写的每个程序里重复代码。取而代之的是，你仅调用库中的一个例行程序。例如，通过将连接到 `cookbook` 数据库的代码放入库程序中，你不需要写出与发起连接相关的所有参数。在你的程序里简单地调用例行程序，然后你就连接上了。

当然建立连接并不是你能封装的唯一操作。本书后面的章节开发放入库文件中的其他工具函数。所有这些文件，包括本节所示的，都可以在 `recipes` 发行包的 `lib` 目录下找到。当你编写自己的程序时，你要大概确认一下你常执行的几项操作以及在适合被包含在库中的候选项。本节所阐述的技术将帮你编写自己的库文件。

库文件除使编写程序更方便外还有其他的好处。例如，如果你在连接到 MySQL 服务器的每个程序里都直接写入连接参数，那么你将它们迁移到使用不同连接参数的另一台机器上时就不得不修改所有的程序。如果取而代之的是你通过调用库里面的程序编写连接到数据库的程序，你就将需要做出变化的位置局部化了：仅需要修改受影响的库程序，而不是所有使用它的程序。

从某种意义上来说代码封装还能提升安全性。如果你将一个私有的库文件设为仅你自己可读，那么只有你运行的脚本能执行文件里的程序。或者假设在你的 Web 服务器的文档树下有一些脚本，一个配置合理的服务器将执行脚本并将它们的输出发送到远程客户端。但是如果服务器不知为何配置失误，后果可能是你的脚本以纯文本方式被发送给客户端，因此显示了你的 MySQL 用户名和密码。（当你意识到的时候已经太晚了。）如果建立到

MySQL 服务器的连接代码被放在位于文档树外的一个库文件中，那些参数就不会被暴露给客户端。



警告：要知道如果你安装一个 Web 服务器可读取的库文件，要是你与其他开发者共享 Web 服务器，你就没有多少安全性可言。任何一个开发者都可以写一个 Web 脚本来读取和显示你的库文件，因为默认情况下，脚本有在 Web 服务器下执行的权限，因此有访问库文件的许可。

接下来的程序示例说明了怎么为每个 API 编写包含连接到 MySQL 服务器上的 cookbook 数据库的库文件。调用程序可以使用 2.2 节讨论的错误检查技术以判断连接尝试是否失败。当连接成功或者不能抛出一个异常时，除 PHP 外，每种语言的连接程序返回一个 database 句柄或者连接对象。PHP 程序返回代表一个连接或错误的对象，因为那是 PEAR DB 连接方法所做的（它并不抛出异常）。

库自身并不能独立使用，所以使用每个库时都是通过一个简短的“test harness”程序来说明的。你可以使用这些程序的任一个作为创建你自己的新程序的基础：拷贝一份文件在 connect 和 disconnect 调用之间加入你自己的代码。

库文件编写不仅包括要将什么放入文件的问题，而且包括辅助性的主题，例如要将文件安装在哪儿以便它能被你的程序所访问，以及（在多用户系统譬如 Unix 上）如何设置它的访问权限以便它的内容不会被暴露在不应该看到的人面前。

选择库文件安装位置

如果你将库文件安装在语言处理器默认搜索的目录下，用该语言编写的程序不需要做任何特殊的事情来访问该库。然而，如果你将库文件安装在语言处理器默认不搜索的地方，你不得不告知你的脚本如何找到库。通常有两个方法来完成此任务：

- 大多数语言提供了可被用于脚本中的语句来将目录添加到语言处理器的搜索路径上，这要求你修改需要用到该库的每个脚本。
- 你可以设置改变语言处理器搜索路径的环境变量或配置变量。这个方法要求使用需要库的脚本的每个用户设置合适的变量。如果语言处理器有一个配置文件，你可以在该文件中设置一个参数来影响所有用户的脚本。

我们使用第二种方法。对于我们的 API 语言，下表列出了相应的变量。在每种情况下，变量值应是一个目录或者目录列表。

语言	变量名	变量类型
Perl	PERL5LIB	环境变量
Ruby	RUBYLIN	环境变量
PHP	include_path	配置变量
Python	PYTHONPATH	环境变量
Java	CLASSPATH	环境变量

要获得设置环境变量的全面信息，请参考附录 B。你可以使用那些指令将环境变量设为下面讨论的值。

假设你想将库文件装入语言处理器默认不搜索的目录中。仅为了说明，我们在 Unix 下使用/usr/local/lib/mcb，在 Windows 下使用 C:\lib\mcb。（要将文件放在其他位置，相应调整变量设置里的路径名。例如，如果你想使用一个不同的目录，或者你想将每个语言的库放在不同的目录中。）

在 Unix 下，如果你将 Perl 库文件放入/usr/local/lib/mcb 目录下，你可以设置 PERL5LIB 环境变量。对于 Bourne shell 族中的一个 shell (sh、bash、ksh)，如下在合适的启动文件中设置变量：

```
export PERL5LIB=/usr/local/lib/mcb
```



提示：如果你正使用原始的 Bourne shell、sh，你可能需要将此分为两个命令：

```
PERL5LIB=/usr/local/lib/mcb
export PERL5LIB
```

对于 C shell 族的 shell (csh、tcsh)，如下在你的.login 文件里设置 PERL5LIB：

```
setenv PERL5LIB /usr/local/lib/mcb
```

在 Windows 下，如果你将 Perl 库放在 C:\lib\mcb 下，你可以如下设置 PERL5LIB：

```
PERL5LIB=C:\lib\mcb
```

在每种情况下，变量设置告知 Perl 除了默认搜索的所有目录外还要查看指定的目录来寻找库文件。如果你将 PERL5LIB 设为多个目录，目录路径名间的分隔符在 Unix 下为冒号 (:) 或在 Windows 下为分号 (;)。

其他的环境变量 (RUBYLIN、PYTHONPATH 和 CLASSPATH) 可使用相同语法指定。



提示：刚刚所讨论的设置的环境变量的方法应该能够满足你从命令行执行脚本的需要。但对于试图在 Web 服务器上执行的脚本，你可能需要同样配置一下服务器以便它能找到库文件。请参见 17.2 节中关于怎么完成此任务的细节。

对于 PHP 来说，搜索路径由 PHP 初始化文件 `php.ini` 中的 `include_path` 变量值定义。在 Unix 中，文件的路径名可能是 `/usr/lib/php.ini` 或 `/usr/local/lib/php.ini`。在 Windows 下，文件可以在 Windows 目录下或主要的 PHP 安装目录下找到。`Include_path` 值定义如下：

```
include_path = "value"
```

`value` 是以与命名目录的环境变量同样的语法来指定的。也就是说，它是一个目录名列表，在 Unix 下名称间以冒号分隔，而在 Windows 下以分号分隔。例如，在 Unix 下，如果你希望 PHP 在当前目录和 `/usr/local/lib/mcb` 下寻找包含的文件，以如下方式设置 `include_path`：

```
include_path = ".:/usr/local/lib/mcb"
```

在 Windows 下，要搜索当前目录和 `C:\lib\mcb`，以如下方式设置 `include_path`：

```
include_path = ".;C:\lib\mcb"
```

如果你修改了 `php.ini` 文件，同时 PHP 正作为一个 Apache 模块执行，你将需要重新启动 Apache 来使你的修改生效。

设置库文件访问权限

如果你正使用一个如 Unix 之类的多用户操作系统，关于文件所有权和访问方式的问题会被提出来要你做出决定：

- 如果某个库文件是私有的且包含只能由你使用的代码，该文件应被放置于你自己的账户下，使其只能由你访问。假设库文件 `mylib` 已经由你所拥有，你可以以如下方式使其私有化：

```
% chmod 600 mylib
```

- 如果库文件仅被你的 Web 服务器所使用，你可以将其安装在一个服务器库目录下，使其仅能被服务器用户 ID 所拥有和访问。要做到这一点你需要是 `root` 用户。例如，如果 Web 服务器以 `wwwusr` 运行，下面的命令使文件为该用户私有：

```
# chown wwwusr mylib  
# chmod 600 mylib
```

- 如果库文件是公共的，你可以将其放在你的编程语言查找库文件时自动搜索的位置。（大多数语言处理器在一些默认的目录集里搜索库）要在这些目录之一里安装文件

你需要是 root 用户。然后你可以使该文件为所有人可读：

```
# chmod 444 mylib
```

现在让我们为每个 API 构建一个库。这儿的每一节阐述了如何编写库文件，并讨论了如何在程序里使用库。

Perl

在 Perl 中，库文件被称为模块，并别具特色地有一个为.pm（“Perl module”）的扩展名。下面是一个示例的模块文件，Cookbook.pm，它实现了一个名为 Cookbook 的模块。（作为惯例，Perl 模块文件的名称和文件中的 package 行的标识符一致。）

```
package Cookbook;
# Cookbook.pm - 具有通过 Perl DBI 模块连接 MySQL 的工具方法的库文件

use strict;
use warnings;
use DBI;

my $db_name = "cookbook";
my $host_name = "localhost";
my $user_name = "cbuser";
my $password = "cbpass";
my $port_num = undef;
my $socket_file = undef;

# 建立一个到 cookbook 数据库的连接，返回一个数据库句柄。
# 如果不能建立连接则抛出一个异常。

sub connect
{
    my $dsn = "DBI:mysql:host=$host_name";
    my %conn_attrs = (PrintError => 0, RaiseError => 1, AutoCommit => 1);

    $dsn .= ";database=$db_name" if defined $db_name;
    $dsn .= ";mysql_socket=$socket_file" if defined $socket_file;
    $dsn .= ";port=$port_num" if defined $port_num;

    return (DBI->connect ($dsn, $user_name, $password, \%conn_attrs));
}

1; # 返回 true
```

模块将建立到 MySQL 服务器连接的代码封装到一个 connect() 方法里，同时 package 为模块建立了一个 Cookbook 命名空间，所以你使用模块名来调用 connect() 方法：

```
$dbh = Cookbook::connect();
```

模块文件的最后一行是一条总返回 true 的语句。这是必需的，因为如果模块没有返回一个 true 值，Perl 会假设模块出现了错误，并在读取它之后退出。

Perl 通过搜索在它的@INC 数组里命名的目录来定位库文件。这个数组包含一个默认的目录列表。要检查你系统上这个变量的值，在命令行中以如下方式调用 Perl：

```
% perl -V
```

命令输出最后的部分显示了@INC 数组里的目录。如果你将库文件安装在这些目录中，你的脚本将会自动找到它。如果你将模块安装在其他地方，则需要通过设置 PERL5LIB 环境变量来告知你的脚本到哪儿去找到它，方式如前所述。

在安装了 Cookbook.pm 模块后，在如下的一个测试套脚本 harness.pl 中试用它：

```
#!/usr/bin/perl
# harness.pl - Cookbook.pm 库的测试套

use strict;
use warnings;
use Cookbook;

my $dbh;
eval
{
    $dbh = Cookbook::connect ();
    print "Connected\n";
};
die "$@" if $@;
$dbh->disconnect ();
print "Disconnected\n";
```

harness.pl 没有使用 DBI 语句。它没有必要，因为 Cookbook.php 库文件本身导入了 DBI 模块，所以任何使用 Cookbook 的脚本也获得了 DBI 的访问权。

如果你不想为显式捕获连接错误而苦恼，你可以更简单地编写脚本体。在这种情况下，Perl 将捕获任意连接异常并在打印出由 connect() 方法所生成的错误信息后终止脚本：

```
my $dbh = Cookbook::connect ();
print "Connected\n";
$dbh->disconnect ();
print "Disconnected\n";
```

Ruby

下面的 Ruby 库文件，Cookbook.rb 定义了实现一个 connect 方法的 Cookbook 类：

```
# Cookbook.rb - 具有通过 Ruby DBI 模块连接 MySQL 的工具方法的库文件

require "dbi"

# 建立一个到 cookbook 数据库的连接,
# 返回一个数据库句柄。如果不能建立连接则抛出一个异常

class Cookbook
  @@host = "localhost"
  @@db_name = "cookbook"
  @@user_name = "cbuser"
  @@password = "cbpass"

  # 连接服务器访问 cookbook 数据库的类方法; 返回数据库句柄对象

  def Cookbook.connect
    return DBI.connect("DBI:MySQL:host=#{@@host};database=#{@@db_name}",
                      @@user_name, @@password)
  end
end
```

connect 方法在库中被定义为 Cookbook.connect，因为 Ruby 类方法被定义为 *class_name.method_name*。

Ruby 通过查找在它的\$LOAD_PATH（也可作\$:）变量中命名的目录来定位库文件，它包含一个默认目录列表的数组。如果要检查你系统上这个变量的值，使用 Ruby 来执行这个语句：

```
puts $LOAD_PATH
```

如果你将库文件安装在那些目录之一中，你的脚本将会自动找到它。如果你将文件安装在其他地方，你需要通过设置 RUBYLIB 环境变量来告知你的脚本到哪儿去找它，正如前面的介绍所述。

在安装了 Cookbook.rb 库文件后，在如下的测试套脚本 harness.rb 中试用它：

```
#!/usr/bin/ruby -w
# harness.rb - Cookbook.rb 库的测试套
require "Cookbook"

begin
  dbh = Cookbook.connect
  print "Connected\n"
rescue DBI::DatabaseError => e
```

```
puts "Cannot connect to server"
puts "Error code: #{e.err}"
puts "Error message: #{e.errstr}"
exit(1)
end
dbh.disconnect
print "Disconnected\n"
```

harness.rb 并没有 DBI 模块的 require 语句。这并不是必需的，因为 Cookbook 模块自身导入了 DBI，所以任意导入 Cookbook 的脚本也获得了 DBI 的访问权。

如果你希望发生错误时，脚本立即终止，而不需要你自己来检查异常，以如下方式编写脚本体：

```
dbh = Cookbook.connect
print "Connected\n"
dbh.disconnect
print "Disconnected\n"
```

PHP

PHP 库文件的内容写得类似于常规的 PHP 脚本。你可以按如下的方式编写实现存在一个 connect() 方法的 Cookbook 类的文件 Cookbook.php：

```
<?php
# Cookbook.php - 具有通过 PHP DB 模块连接 MySQL 的工具方法的库文件

require_once "DB.php";

class Cookbook
{
    # 建立一个到 cookbook 数据库的连接，返回一个 connection 对象，
    # 如果有错误则是一个 error 对象。

    function connect ()
    {
        $dsn = array (
            "phptype"  => "mysqli",
            "username"  => "cbuser",
            "password"  => "cbpass",
            "hostspec"  => "localhost",
            "database"  => "cookbook"
        );
        return (DB::connect ($dsn));
    }

} # 结束 Cookbook
?>
```

尽管本书中的大多数 PHP 示例并不显示<?php 和?>标签，我已在这将它们作为 Cookbook.php 的部分显示了，以强调库文件必须用这些标签来包括所有的 PHP 代码。PHP 解析器在

开始解析时并不对库文件的内容做任何假设，因为你可能包括一个仅包含 HTML 的文件。因此，你必须使用<?php 和?>来显式指定库文件的哪些部分必须被认为是 PHP 代码而不是 HTML，就像你在主脚本中所做的那样。

PHP 通过搜索 PHP 初始化文件里的 `include_path` 变量值中所命名的目录来查找库文件，正如本节前面简介里面所述。假设 `Cookbook.php` 被安装在那些目录中，你可以以如下方式在一个测试套脚本 `harness.php` 中访问它：

```
<?php
# harness.php - Cookbook.php 库的测试套
require_once "Cookbook.php";

$conn =& Cookbook::connect ();
if (PEAR::isError ($conn))
    die ("Cannot connect to server: " . $conn->getMessage () . "\n");
print ("Connected\n");
$conn->disconnect ();
print ("Disconnected\n");

?>
```

`harness.php` 中并无包含 `DB.php` 的语句。它并非必需的，因为 `Cookbook` 模块自身包含了 `DB.php`，这就给予任何包含 `Cookbook.php` 的脚本以 `DB.php` 的访问权。

PHP 库文件应被安装在哪儿？

PHP 脚本常被放置于你的 Web 服务器的文档树中，接着客户端能够直接请求他们。对于 PHP 库文件，我建议你将他们放在文档树之外的某处，特别是如果（类似于 `Cookbook.php`）他们包含用户名和密码时。如果你使用类似于.inc 之类不同的扩展名来作为包含文件的名称的话，这就更为正确了。如果你那样做了，将包含文件安装在文档树下，他们就可能直接被客户端所请求，并被作为普通文本显示，从而暴露了其内容。为避免该情况的发生，重新配置 Apache 以便它能将具有.inc 扩展名的文件视为 PHP 代码，由 PHP 解析器来处理而不是以文本方式来显示。

Python

Python 库被写成模块，并使用 `import` 或 `from` 语句从脚本中来引用。要创建一个连接到 MySQL 的方法，我们可以编写模块文件 `Cookbook.py`：

```
# Cookbook.py - 具有通过 MySQLdb 模块连接 MySQL 的工具方法的库文件
```

```

import MySQLdb

host_name = "localhost"
db_name = "cookbook"
user_name = "cbuser"
password = "cbpass"

# 建立一个到 cookbook 数据库的连接，返回一个 connection 对象，
# 如果不能建立连接则抛出一个异常。

def connect():
    return MySQLdb.connect (db = db_name,
                           host = host_name,
                           user = user_name,
                           passwd = password)

```

文件名决定了模块名称，所以模块被称为 `Cookbook`。模块方法通过模块名被访问，因此你将调用 `Cookbook` 模块的 `connect()` 方法，如下：

```
conn = Cookbook.connect();
```

Python 解析器在 `sys.path` 变量所命名的目录中查找模块。你可以通过交互式运行 Python 并输入一对命令来查明你系统上 `sys.path` 的默认值：

```
% python
>>> import sys
>>> sys.path
```

如果你将 `Cookbook.py` 安装在 `sys.path` 命名的目录中，你的脚本将无需特殊处理就能找到它。如果你将其安装在其他位置，你将需要设置 `PYTHONPATH` 环境变量，正如本节前面的介绍中所述。

在安装了 `Cookbook.py` 库文件后，以如下方式在一个测试套脚本 `harness.py` 中试用它：

```

#!/usr/bin/python
# harness.py - Cookbook.py 库的测试套

import sys
import MySQLdb
import Cookbook

try:
    conn = Cookbook.connect ()
    print "Connected"
except MySQLdb.Error, e:
    print "Cannot connect to server"
    print "Error code:", e.args[0]
    print "Error message:", e.args[1]
    sys.exit (1)

```

```
conn.close ()  
print "Disconnected"
```



提示：Cookbook.py 文件导入了 MySQLdb 模块，但是导入 Cookbook 的脚本并未因此就获得了 MySQLdb 的访问权。如果脚本需要 MySQLdb 相关信息（例如 MySQLdb.Error），脚本必须还导入 MySQLdb。

如果在发生错误时你只想终止脚本而无须自己去检查异常，可以如下方式编写脚本体：

```
conn = Cookbook.connect ()  
print "Connected"  
conn.close ()  
print "Disconnected"
```

Java

Java 库文件在很多方面类似于 Java 程序：

- 源文件里的 class 行表明一个类名；
- 文件名应该和类名一致（带有一个.java 扩展名）；
- 你编译.java 文件来生成一个.class 文件。

Java 库文件在一些方面也和 Java 程序有所不同：

- 不同于普通的程序文件，Java 库文件没有 main() 方法。
- 库文件应该以一个 package 标识符开头，以表明在 Java 命名空间内的类位置。

Java 包标识符的一个公共惯例是以代码作者的反向域名开头，这就使标识符唯一，从而避免了与其他作者所写的类发生冲突。在域命名空间里域名从右到左为从通用到特殊，而 Java 类命名空间从左到右为从通用到特殊。因此，要在 Java 类命名空间内使用域名作为包名的前缀，有必要对其进行反转。以我为例，域名是 kitebird.com，所以如果我写了一个库文件，并将其放在我域名命名空间下的 mcb 中，库以如下的 package 表达式开头：

```
package com.kitebird.mcb;
```

本书开发的 Java 包被放在 com.kitebird.mcb 命名空间内以确保在包命名空间内的唯一性。

下面的库文件，Cookbook.java，定义了实现连接到 cookbook 数据库的 connect() 方法的 Cookbook 类。如果成功，connect() 返回一个 Connection 对象，否则抛出一个异常。为帮助调用者处理错误，Cookbook 类还定义了 getErrorMessage() 和 printErrorMessage() 工具方法，它们分别将错误信息作为字符串返回或将其输出到 System.err。

```

// Cookbook.java - 具有通过 MySQL Connector/J 连接 MySQL 的工具方法的库文件

package com.kitebird.mcb;

import java.sql.*;

public class Cookbook
{
    // 建立一个到 cookbook 数据库的连接，返回一个 connection 对象。
    // 如果不能建立连接则抛出一个异常。

    public static Connection connect () throws Exception
    {
        String url = "jdbc:mysql://localhost/cookbook";
        String user = "cbuser";
        String password = "cbpass";

        Class.forName ("com.mysql.jdbc.Driver").newInstance ();
        return (DriverManager.getConnection (url, user, password));
    }

    // 将错误信息作为字符串返回

    public static String getErrorMessage (Exception e)
    {
        StringBuffer s = new StringBuffer ();
        if (e instanceof SQLException) // JDBC 相关异常?
        {
            // 输出通用信息，以及其他数据库相关的信息
            s.append ("Error message: " + e.getMessage () + "\n");
            s.append ("Error code: " + ((SQLException) e).getErrorCode () + "\n");
        }
        else
        {
            s.append (e + "\n");
        }
        return (s.toString ());
    }

    // 获得错误信息并将其输出到 System.err

    public static void printErrorMessage (Exception e)
    {
        System.err.println (Cookbook.getErrorMessage (e));
    }
}

```

类中的例行程序使用 `static` 关键字声明，这就使它们是编程类方法而非实例方法。这儿之所以这么做，是因为类可以直接被使用而无须创建该类的一个对象，然后再通过对象来调用方法。

要使用 `Cookbook.java` 文件，编译它生成 `Cookbook.class`，然后将类文件装入包标识符对应的目录中。这就意味着 `Cookbook.class` 应该被放入名为 `com/kitebird/mcb`（或者在 Windows 下为 `com\kitebird\mcb`）的目录中，该目录位于你的 `CLASSPATH` 设置的目录下。例如，如果在 Unix 下 `CLASSPATH` 包括 `/usr/local/lib/mcb`，你可以将 `Cookbook.class` 放入 `/usr/local/lib/mcb/com/kitebird/mcb` 目录。（请参考 2.1 节中关于 `CLASSPATH` 变量的 Java 讨论。）

要在 Java 程序中使用 `Cookbook` 类，导入它，然后调用 `Cookbook.connect()` 方法。下面的测试套程序 `Harness.java` 展示了怎么做：

```
// Harness.java - Cookbook 库类的测试套  
  
import java.sql.*;  
import com.kitebird.mcb.Cookbook;  
  
public class Harness  
{  
    public static void main (String[] args)  
    {  
        Connection conn = null;  
        try  
        {  
            conn = Cookbook.connect ();  
            System.out.println ("Connected");  
        }  
        catch (Exception e)  
        {  
            Cookbook.printErrorMessage (e);  
            System.exit (1);  
        }  
        finally  
        {  
            if (conn != null)  
            {  
                try  
                {  
                    conn.close ();  
                    System.out.println ("Disconnected");  
                }  
                catch (Exception e)  
                {  
                    String err = Cookbook.getErrorMessage (e);  
                    System.out.println (err);  
                }  
            }  
        }  
    }  
}
```

`Harness.java` 还展示了当 MySQL 相关的异常发生时如何使用 `Cookbook` 类的错误信息工具方法：

- `printErrorMessage()` 获得异常对象并使用它来输出一条错误信息到 `System.err`。
- `getErrorMessage()` 以字符串返回错误信息。你可以自己显示信息，将其写入一个日志文件中，或者其他的文件中。

2.4 发起语句并检索结果

Issuing Statements and Retrieving Results

问题

你需要你的程序向 MySQL 服务器发送一条 SQL 语句并检索它生成的结果。

解决方案

一些语句仅返回一个状态代码，其他的返回一个结果集（一个行集）。一些 API 提供了发起每种类型语句的不同方法。使用和要执行的语句相应的方法。

讨论

有两类你可以执行的 SQL 语句。部分语句从数据库检索信息，其他的对这些信息做出改变。这两类语句被不同地处理。除此之外，有些 API 提供了几个不同的例行程序来发起语句，这个关系比较复杂。在我们阐述如何用每个 API 发起语句之前，我会先说明一下示例所使用的数据库表，然后进一步讨论两个语句种类并说明每类中处理语句的通用策略。

在第 1 章中，我们创建了一个名为 `limbs` 的表来尝试一些示例语句。在本章中，我们使用一个不同的名为 `profile` 的表。它基于一个“朋友列表”的主意，也就是说，当我们在线时想与之保持联系的人的集合。要维护每个人的属性，我们可以使用如下的表定义：

```
CREATE TABLE profile
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name    CHAR(20) NOT NULL,
    birth   DATE,
    color   ENUM('blue', 'red', 'green', 'brown', 'black', 'white'),
    foods   SET('lutefisk', 'burrito', 'curry', 'eggroll', 'fadge', 'pizza'),
    cats    INT,
    PRIMARY KEY (id)
);
```

`profile` 表表明了对于我们来说关于每个朋友的重要信息：姓名、年龄、喜爱的颜色、喜爱的食物，以及猫的数目——明显就是仅用于一本书的示例的那些“傻瓜”表之一（注 2）。表还包括一个 `id` 列，它包含一个唯一值以便我们能区分不同的行，即使两个朋友拥有相同的姓名。`id` 和 `name` 被声明为 NOT NULL，因为他们每个都要求有个值。其他列允许为 NULL（并且那是他们隐式的默认值）因为我们可能还不知道他们每个要放入的值。也就是说，我们将使用 NULL 来表示“unknown”。

尽管我们想保持年龄记录，但表中并无 `age` 列。取而代之的是，有一个 DATE 类型的 `birth` 列。那是因为年龄是变化的，而生日不会。如果记录了年龄值，我们将不得不持续更新它们。储存出生日期更适合，因为它是固定的，然后我们可以在任何给定时候用它来计算年龄。（6.11 节讨论了年龄计算。）`color` 是一个 ENUM 列；颜色值可以是列出的值中的任何一个。`foods` 是一个 SET，它允许选择集合成员的组合来作为值。按这种方式我们能够为任何人记录多个喜爱的食物。

要创建表，使用 `recipes` 发行包的 `tables` 目录里的脚本 `profile.sql`。改变位置到该目录下，然后运行以下命令：

```
% mysql cookbook < profile.sql
```

创建表的另一个方法就是手动从 `mysql` 程序内发起 CREATE TABLE 语句，但是我推荐你使用脚本，因为它还载入样本数据到表中。那样你就可以试验该表，然后如果你改变了他的内容可以通过重新运行脚本来恢复它。（参见本章的最后一节，关于恢复 `profile` 表的重要性的说明）

`profile.sql` 脚本装入的 `profile` 表的初始内容如下：

```
mysql> SELECT * FROM profile;
+----+-----+-----+-----+-----+-----+
| id | name | birth | color | foods           | cats |
+----+-----+-----+-----+-----+-----+
| 1  | Fred | 1970-04-13 | black | lutefisk,fadge,pizza | 0   |
| 2  | Mort | 1969-09-30 | white | burrito,curry,eggroll | 3   |
| 3  | Brit | 1957-12-01 | red   | burrito,curry,pizza | 1   |
| 4  | Carl | 1973-11-02 | red   | eggroll,pizza       | 4   |
| 5  | Sean | 1963-07-04 | blue  | burrito,curry       | 5   |
| 6  | Alan | 1965-02-14 | red   | curry,fadge        | 1   |
| 7  | Mara | 1968-09-17 | green | lutefisk,fadge     | 1   |
| 8  | Shepard | 1975-09-02 | black | curry,pizza       | 2   |
| 9  | Dick | 1952-08-20 | green | lutefisk,fadge     | 0   |
| 10 | Tony | 1960-05-01 | white | burrito,pizza     | 0   |
+----+-----+-----+-----+-----+-----+
```

注 2：实际上，这些表也不是那么“傻瓜”。表格使用不同的数据类型来表示列，这些便于阐述后面如何解决特殊数据类型的问题。

虽然 profile 表中的多数列允许 NULL 值，样本数据集中没有一行实际包含 NULL。那是因为 NULL 使语句处理起来变得复杂，在 2.5 节和 2.7 节将会处理这些复杂的部分。

SQL 语句种类

SQL 语句可以被分为两大类：

- 不返回结果集（也就是一个行集）的语句。这类语句包括 INSERT、DELETE 和 UPDATE。作为一个通用规则，这类语句通常以某种方式改变了数据库。也有一些例外，譬如 USE db_name，它改变了你会话的默认（当前）数据库而不改变数据库本身。本节所使用示例的修改数据语句是一个 UPDATE：

```
UPDATE profile SET cats = cats+1 WHERE name = 'Fred'
```

我们将介绍如何发起这个语句并确定它所影响的行数。

- 返回一个结果集的语句，例如 SELECT、SHOW、EXPLAIN 和 DESCRIBE。通常这些语句我是指 SELECT 语句，但是你应该理解这类包括所有返回行的语句。本节中所使用的示例行检索语句是一个 SELECT：

```
SELECT id, name, cats FROM profile
```

我们将介绍如何发起这个语句，获取结果集中的行，并确定结果集中的行数和列数。

（如果你需要诸如列名或数据类型之类的信息，你必须访问结果集元数据。9.2 节对此进行了详细说明。）

处理一条 SQL 语句的第一步是将其发送到 MySQL 服务器供执行。某些 API（例如，Perl，Ruby 和 Java 的 API）了解两类语句间的区别并为他们的执行提供不同的调用。其他 API（例如 PHP 和 Python 的 API）只有一类简单的调用，它可供任何语句使用。然而，所有 API 的一个共性就是你不需要使用任何特殊字符来表示语句的结尾。终结符不是必需的，因为语句字符串的结尾隐式地终止了语句。这和你在 mysql 程序中发起语句的方式不同，在那里你使用分号（;）或\g 来终止语句。（它也和我通常在示例中所展示的 SQL 语句语法有所不同，因为我常使用分号以使语句结尾清晰可见。）

当你向服务器发送一条语句时，应该有准备来处理当它未能正确执行时的错误。**不要忽视做这个！**如果一条语句失败了，而你继续基于其成功的假设执行，你的程序将不会正常工作。对于多数部分，错误检查代码在本节中并没有出现，但那只是为简略起见。包含示例的 recipes 发行包中的示例脚本包含了错误处理，它基于 2.2 节所阐述的技术。

如果一条语句无错且执行了，接下来的一步依赖于你所发起的语句类型。如果它是不返回结果集的语句，就没有什么需要做的了，除非你想检查该语句影响了多少行。如果语句返回了一个结果集，你可以获取它的行，然后关闭结果集。如果你正执行一条语句，而在其所属的上下文中你并不知道它是否返回一个结果集，9.3节探讨了如何获知。

Perl

Perl DBI 模块为 SQL 语句执行提供了两个基本的方法，依赖于你是否期望获得一个结果集。要发起不返回结果集的一条语句诸如 INSERT 或 UPDATE，使用数据库句柄的 do() 方法。它执行语句并返回它所影响的行数，或者在发生错误时返回 undef。例如，如果 Fred 新获得一只猫，以下语句可用于递增他的 cats 数目：

```
my $count = $dbh->do ("UPDATE profile SET cats = cats+1
                         WHERE name = 'Fred'");
if ($count)    # 如果未发生错误则输出行数
{
    $count += 0;
    print "Number of rows updated: $count\n";
}
```

如果语句成功执行但未影响任何行，do() 返回一个特殊的值，“0E0”（也就是科学表达式的 0 值，被作为一个字符串表示）。“0E0”可被用于测试一条语句的执行状态，因为它在 Boolean 上下文中是 true（不同于 undef）。对于成功的语句，它也可用于统计所影响的行数，因为它在数字上下文中被当成是数字 0。当然，如果你打印出它的值，你将会打出“0E0”，这可能对于使用你程序的人来说有些不可思议：前面的示例显示了一个可确保其不发生的方法：为该值加 0 使其显式地转化为数字形式以便它显示为 0。你还可以使用带 %d 格式指定符的 printf 来引起一个隐式的数值转化：

```
my $count = $dbh->do ("UPDATE profile SET cats = cats+1
                         WHERE name = 'Fred'");
if ($count)    # 如果未发生错误则输出行数
{
    printf "Number of rows updated: %d\n", $count;
}
```

如果允许 RaiseError，当一个 DBI 相关的错误发生时你的脚本会自动中止，所以你不需要为检查\$count 来查看 do() 是否失败而困扰。在那种情况下，你或许可以简化一下代码：

```
my $count = $dbh->do ("UPDATE profile SET cats = cats+1  
                      WHERE name = 'Fred'");  
printf "Number of rows updated: %d\n", $count;
```

要处理返回结果集的语句诸如 SELECT，使用一个包含几个步骤的不同方法：

1. 通过调用使用数据库句柄的 `prepare()` 方法来指定要执行的语句。如果 `prepare()` 方法成功，它会返回一个用于后续操作的 `statement` 句柄。(如果发生错误，开启了 `RaiseError` 时脚本会中止；否则，`prepare()` 返回 `undef`。)
2. 调用 `execute()` 执行语句，生成结果集。
3. 执行循环来获取语句返回的行。DBI 提供了可以用于这个循环中的几个方法，待会儿我们就会介绍他们。
4. 如果你未取出完整的结果集，通过调用 `finish()` 来释放与其关联的资源。

下面的示例说明了这些步骤，使用 `fetchrow_array()` 作为获取行的方法并假设 `RaiseError` 开启以便出现错误来中止脚本：

```
my $sth = $dbh->prepare ("SELECT id, name, cats FROM profile");  
$sth->execute ();  
my $count = 0;  
while (my @val = $sth->fetchrow_array ())  
{  
    print "id: $val[0], name: $val[1], cats: $val[2]\n";  
    ++$count;  
}  
$sth->finish ();  
print "Number of rows returned: $count\n";
```

上面所示的行获取循环后面跟着一个 `finish()` 的调用，它关闭结果集并告知服务器可以释放与其关联的任意资源。如果你取出了结果集中的每一行，你实际上并不需要调用 `finish()`，因为 DBI 注意到你已经到达最后一行然后会为你释放结果集。显式调用 `finish()` 的重要时刻是当你未完全取出结果集时。因此，这个示例可以略去 `finish()` 调用而没有任何恶果。

示例说明了如果你想知道结果集包含多少行，你应该在你取出他们时自己来对其进行计数。不要用 DBI 的 `rows()` 方法来作此用，DBI 文档并不鼓励这个操作。(原因在于 `rows()` 对于 SELECT 语句来说并不一定可靠——并不是因为 DBI 的不足，而是因为不同数据库引擎在表现上的差异。)

DBI 有几种方法来每次获取一行。前面的示例所使用的 `fetchrow_array()` 返回包含下一行的一个数组，或者如果没有多余行时返回一个空的列表。数组元素可通过 `$val[0]`、`$val[1]`、……形式访问，它们在数组中的顺序和在 SELECT 语句中的顺序一致。数组的大小告知你结果集有多少列。

`fetchrow_array()`方法对于显式命名了要选择的列的语句十分有用。(如果你用 `SELECT *` 来检索列, 那么数组中列的位置就没有保证。)

`fetchrow_arrayref()`类似于 `fetchrow_array()`, 除了它返回一个数组的引用, 或者在没有多余行时返回 `undef` 之外。数组元素以 `$ref->[0]`, `$ref->[1]` 等来访问。和 `fetchrow_array()`一样, 值是以语句中命名的顺序展现的:

```
my $sth = $dbh->prepare ("SELECT id, name, cats FROM profile");
$sth->execute ();
my $count = 0;
while (my $ref = $sth->fetchrow_arrayref ())
{
    print "id: $ref->[0], name: $ref->[1], cats: $ref->[2]\n";
    ++$count;
}
print "Number of rows returned: $count\n";
```

`fetchrow_hashref()`返回一个哈希表结构的引用, 或者在没有其他行时返回 `undef`:

```
my $sth = $dbh->prepare ("SELECT id, name, cats FROM profile");
$sth->execute ();
my $count = 0;
while (my $ref = $sth->fetchrow_hashref ())
{
    print "id: $ref->{id}, name: $ref->{name}, cats: $ref->{cats}\n";
    ++$count;
}
print "Number of rows returned: $count\n";
```

要访问哈希表的元素, 使用被语句所选择的列名 (`$ref->{id}`、`$ref->{name}` 等等)。`fetchrow_hashref()`对于 `SELECT *` 语句尤其有用, 因为你可以访问行元素而无需知道返回的列的顺序, 你只需要知道他们名称。从另一方面来说, 建立一个哈希表比一个数组代价更高些, 所以 `fetchrow_hashref()` 比 `fetchrow_array()` 或 `fetchrow_arrayref()` 慢得多。如果他们有相同的名称, 它还可能“丢失”行元素, 因为列名必须是唯一的。下面的语句选择两个值, 但是 `fetchrow_hashref()` 将返回仅包含一个名为 `id` 的元素的哈希表结构:

```
SELECT id, id FROM profile
```

要避免这个问题, 你可以使用列别名来确保名称相同的列在结果集中有不同的名称。下面的语句检索和上一语句相同的列, 但赋给他们以不同的名称 `id` 和 `id2`:

```
SELECT id, id AS id2 FROM profile
```

无可否认，这个语句显得相当愚蠢。然而，如果你正从多个表中检索列，你很容易就会遇上在结果集中有同名列的问题。发生此情况的示例可参考 12.16 节。

除了刚刚描述的语句执行过程的方法外，DBI 还提供了几个高级的检索方法，可以在一次调用中发起一条语句并返回结果集。所有这些是数据库句柄方法，他们在返回结果集前内部处理了 statement 句柄的创建和释放，不同点在于他们返回的结果的形式。部分返回整个结果集，其他的返回结果集单一的行或列，如下表所述：

方法	返回值
selectrow_array()	数组形式的结果集的第一行
selectrow_arrayref()	数组引用形式的结果集的第一行
selectrow_hashref()	哈希表引用形式的结果集的第一行
selectcol_arrayref()	数组引用形式的结果集的第一列
selectall_arrayref()	以内含数组引用的数组的引用形式出现的整个结果集
selectall_hashref()	以内含哈希引用的哈希表的引用形式出现的整个结果集

这些方法多数返回一个引用。`selectrow_array()`例外，它选择结果集的第一行，并返回一个数组或一个标量，这依赖于你怎么调用它。在数组上下文中，`selectrow_array()`以数组形式返回整行（如果没有选中行则返回空列表）。这对于你期望的仅获得一行的语句很有用：

```
my @val = $dbh->selectrow_array ("SELECT name, birth, foods FROM profile  
WHERE id = 3");
```

当在数组上下文中调用 `selectrow_array()`时，返回值可用于确定结果集的大小。列数是数组中的元素数目，行数是 1 或 0：

```
my $ncols = @val;  
my $nrows = ($ncols ? 1 : 0);
```

你也可以在标量上下文中调用 `selectrow_array()`，在这种情况下仅返回行的第一列。这对于返回单一值的语句来说特别方便：

```
my $buddy_count = $dbh->selectrow_array ("SELECT COUNT(*) FROM profile");
```

如果一条语句没有返回结果，`selectrow_array()`返回一个空数组或者 `undef`，这依赖于你是在数组上下文还是标量上下文中调用它。

`selectrow_arrayref()` 和 `selectrow_hashref()` 选择结果集的第一行并返回对它的引用或者如果没有选中行时返回 `undef`。要访问列值，以你对待 `fetchrow_arrayref()` 或

`fetchrow_hashref()`相同的方式来对待引用。你也可以使用引用来获得行数和列数：

```
my $ref = $dbh->selectrow_arrayref ($stmt);
my $ncols = (defined ($ref) ? @{$ref} : 0);
my $nrows = ($ncols ? 1 : 0);

my $ref = $dbh->selectrow_hashref ($stmt);
my $ncols = (defined ($ref) ? keys (%{$ref}) : 0);
my $nrows = ($ncols ? 1 : 0);
```

使用 `selectcol_arrayref()`，会返回一个单列数组的引用，代表结果集的第一列。假定为一个非 `undef` 的返回值，数组元素可用 `$ref->[i]` 来访问第 *i* 行的值。行数是数组里的元素数目，列数是 1 或 0：

```
my $ref = $dbh->selectcol_arrayref ($stmt);
my $nrows = (defined ($ref) ? @{$ref} : 0);
my $ncols = ($nrows ? 1 : 0);
```

`selectall_arrayref()` 返回一个数组的引用，数组包含结果集每行的元素。每个元素是一个数组引用。要访问结果集的第 *i* 行，使用 `$ref->[i]` 来获得该行的引用。然后以与 `fetchrow_arrayref()` 的返回值相同的方式来对待行引用以访问行中不同列的值。结果集的行数和列数可用如下方式获得：

```
my $ref = $dbh->selectall_arrayref ($stmt);
my $nrows = (defined ($ref) ? @{$ref} : 0);
my $ncols = ($nrows ? @{$ref->[0]} : 0);
```

`selectall_hashref()` 有些类似于 `selectall_arrayref()`，但返回一个哈希表的引用，它的每个元素是结果的一行的一个哈希引用。要调用它，可以指定一个用作哈希键值的列的指数为参数。例如，如果你正检索 `profile` 表的行，主键是列 `id`：

```
my $ref = $dbh->selectall_hashref ("SELECT * FROM profile", "id");
```

然后使用哈希表的键值来访问行。例如，如果一行的列键值为 12，该行的哈希引用可用 `$ref->{12}` 来访问。如果该行的值以列名为键值，你可以用它来访问不同的列元素（例如，`$ref->{12}->{name}`）。结果集的行数和列数可用如下方式获得：

```
my @keys = (defined ($ref) ? keys (%{$ref}) : ());
my $nrows = scalar (@keys);
my $ncols = ($nrows ? keys (%{$ref->{ $keys[0] }}) : 0);
```

当你需要多次处理一个结果集时，不同的 `selectall_XXX()` 方法很有用，因为 Perl DBI 没有提供方法来“重新获得”一个结果集。通过将整个结果集赋给一个变量，你可以多次迭代访问它的元素。

当你禁止了 `RaiseError`, 使用高级方法时需要小心些。在这种情况下, 方法的返回值并不是总能使你区分错误和空的结果集。例如, 如果你在标量上下文中调用 `selectrow_array()` 来检索单一值, 一个 `undef` 返回值就会使你模棱两可, 因为它可能表示三种情况: 一个错误, 一个空的结果集, 或者一个包含单一的 `NULL` 值的结果集。如果你需要对错误进行检测, 你应该检查 `$DBI::errstr`, `$DBI::err`, 或 `$DBI::state` 的值。

Ruby

和 Perl DBI 一样, Ruby DBI 为 SQL 语句执行提供了两种方法。使用任一方法, 如果语句执行方法出错失败, 都会抛出一个异常。

对于诸如 `INSERT` 或 `UPDATE` 等不返回结果集的语句, 调用 `do` 数据库句柄方法, 它的返回值表示所影响的行数:

```
count = dbh.do("UPDATE profile SET cats = cats+1 WHERE name = 'Fred'")  
puts "Number of rows updated: #{count}"
```

对于诸如 `SELECT` 等返回结果集的语句, 调用 `execute` 数据库句柄方法。`execute` 返回一个 `statement` 句柄用于获取结果集的行。`statement` 句柄自身有几个方法可以以不同的方式来获取行。在你用完 `statement` 句柄后, 调用它的 `finish` 方法。(不同于 Perl DBI, 仅在你没有获取整个结果集时必须调用 `finish`, 在 Ruby 中你应该为你所创建的每个 `statement` 句柄都调用 `finish`。) 要确定结果集的行数, 在你获取他们时就对其进行计数。

下面的示例执行一个 `SELECT` 语句并在一个 `while` 循环中使用 `statement` 句柄的 `fetch` 方法:

```
count = 0  
sth = dbh.execute("SELECT id, name, cats FROM profile")  
while row = sth.fetch do  
    printf "id: %s, name: %s, cats: %s\n", row[0], row[1], row[2]  
    count += 1  
end  
sth.finish  
puts "Number of rows returned: #{count}"
```

你还可以使用 `fetch` 作为顺序返回每行的迭代器:

```
count = 0  
sth = dbh.execute("SELECT id, name, cats FROM profile")  
sth.fetch do |row|  
    printf "id: %s, name: %s, cats: %s\n", row[0], row[1], row[2]  
    count += 1  
end  
sth.finish  
puts "Number of rows returned: #{count}"
```

在迭代上下文中(正如刚才所示), `each` 方法与 `fetch` 方法效果一致。

fetch 方法返回 DBI::Row 对象。正如刚才所示，你可以用位置（从 0 开始）来访问行中的列值。DBI::Row 对象还提供了用名称来访问列的方法：

```
sth.fetch do |row|
  printf "id: %s, name: %s, cats: %s\n",
    row["id"], row["name"], row["cats"]
end
```

要一次性获取所有的行，使用 fetch_all，它返回一个 DBI::Row 对象数组：

```
sth = dbh.execute("SELECT id, name, cats FROM profile")
rows = sth.fetch_all
sth.finish
rows.each do |row|
  printf "id: %s, name: %s, cats: %s\n",
    row["id"], row["name"], row["cats"]
end
```

row.size 告诉你结果集中的列数。

要获取每行作为以列名为键值的哈希表，使用 fetch_hash 方法。它可以在循环中调用或者用作一个迭代器。下面的示例展示了迭代器方法：

```
count = 0
sth = dbh.execute("SELECT id, name, cats FROM profile")
sth.fetch_hash do |row|
  printf "id: %s, name: %s, cats: %s\n",
    row["id"], row["name"], row["cats"]
  count += 1
end
sth.finish
puts "Number of rows returned: #{count}"
```

上面的示例调用 execute 来获得一个 statement 句柄，接着在不再需要时句柄调用 finish。调用 execute 的另一个方法是用一个代码块。在这种情况下，它将 statement 句柄传给块，然后自动在那个句柄上调用 finish。例如：

```
count = 0
dbh.execute("SELECT id, name, cats FROM profile") do |sth|
  sth.fetch do |row|
    printf "id: %s, name: %s, cats: %s\n", row[0], row[1], row[2]
    count += 1
  end
end
puts "Number of rows returned: #{count}"
```

Ruby DBI 有几个高级的数据库句柄方法用于执行语句来生成结果集：

- select_one 执行一次查询并以数组形式返回第一行（如果结果为空返回 nil）：

```
row = dbh.select_one("SELECT id, name, cats FROM profile WHERE id = 3")
```

- `select_all` 执行一次查询并返回一个 `DBI::Row` 对象数组，一个对象代表结果集的一行。如果结果为空，数组也为空：

```
rows = dbh.select_all( "SELECT id, name, cats FROM profile")
```

当你需要多次处理一个结果集时 `select_all` 方法很有用，因为 Ruby DBI 没有提供其他方法来“重新获取”一个结果集。通过获取完整的结果集作为一个行对象数组，你可以多次迭代访问它的元素。如果你仅需要操作一次行集，你可以直接应用一个迭代器到 `select_all` 上：

```
dbh.select_all("SELECT id, name, cats FROM profile").each do |row|
  printf "id: %s, name: %s, cats: %s\n",
         row["id"], row["name"], row["cats"]
end
```

PHP

在 PHP 中，对于发起返回结果集的语句以及不返回的语句，PEAR DB 没有不同的方法。取而代之的是，`query()` 方法执行所有种类的语句。`query()` 接收一个语句字符串参数并返回一个结果值，你可以用 `PEAR::isError()` 来测试以判断语句是否失败。如果 `PEAR::isError()` 为 `true`，它意味着你的语句出错了：可能是句法不正确，你没有访问语句中的表的权限，或者是一些阻止语句执行的其他问题。

如果语句执行合理，在那时你所做的将依赖于语句的类型。对于诸如 `INSERT` 或 `UPDATE` 等不返回行的语句，语句已经完成。如果需要，你可以调用连接对象的 `affectedRows()` 方法来确定多少行发生了变化：

```
$result = & $conn->query ("UPDATE profile SET cats = cats+1
                           WHERE name = 'Fred'");
if (PEAR::isError ($result))
  die ("Oops, the statement failed");
printf ("Number of rows updated: %d\n", $conn->affectedRows());
```

对于诸如 `SELECT` 等返回了结果集的语句，`query()` 方法返回一个结果集对象。通常，你使用这个对象在一个循环中调用一个获取行的方法，然后调用它的 `free()` 方法来释放结果集。下面是一个示例，它展示了如何发起一个 `SELECT` 语句并用结果集对象来获取行：

```
$result = & $conn->query ("SELECT id, name, cats FROM profile");
if (PEAR::isError ($result))
  die ("Oops, the statement failed");
while ($row =& $result->fetchRow (DB_FETCHMODE_ORDERED))
  print ("id: $row[0], name: $row[1], cats: $row[2]\n");
printf ("Number of rows returned: %d\n", $result->numRows ());
$result->free ();
```

示例说明了结果集对象所拥有的几个方法：

- 要获得结果集中的行，可执行一个循环，在其中调用 `fetchRow()` 方法。
- 要获得结果集中行的数目统计，可调用 `numRows()`。
- 当没有其他行时，可调用 `free()` 方法。

`fetchRow()` 方法返回结果集的下一行，或者在没有其他行时返回 `NULL`。`fetchRow()` 接收一个参数，它代表该方法应返回的值的类型。如上面的示例所示，带有一个 `DB_FETCHMODE_ORDERED` 参数，`fetchRow()` 返回一个元素数组，它们对应于语句所选择的列并可使用数值下标来访问。数组的大小表示结果集中的列数。

使用参数 `DB_FETCHMODE_ASSOC`，`fetchRow()` 返回一个关联数组，其中包含通过列名访问的值：

```
$result = & $conn->query ("SELECT id, name, cats FROM profile");
if (PEAR::isError ($result))
    die ("Oops, the statement failed");
while ($row = & $result->fetchRow (DB_FETCHMODE_ASSOC))
{
    printf ("id: %s, name: %s, cats: %s\n",
           $row["id"], $row["name"], $row["cats"]);
}
printf ("Number of rows returned: %d\n", $result->numRows ());
$result->free ();
```

使用参数 `DB_FETCHMODE_OBJECT`，`fetchRow()` 返回一个对象，它的成员可以使用列名访问：

```
$result = & $conn->query ("SELECT id, name, cats FROM profile");
if (PEAR::isError ($result))
    die ("Oops, the statement failed");
while ($row = & $result->fetchRow (DB_FETCHMODE_OBJECT))
    print ("id: $row->id, name: $row->name, cats: $row->cats\n");
printf ("Number of rows returned: %d\n", $result->numRows ());
$result->free ();
```

如果你无参调用 `fetchRow()`，它使用默认的获取模式。通常，该值是 `DB_FETCHMODE_ORDERED`，除非你调用连接对象的 `setFetchMode()` 方法改变了它。例如，要使用 `DB_FETCHMODE_ASSOC` 作为一个连接的默认获取模式，可按如下方式做：

```
$conn->setFetchMode (DB_FETCHMODE_ASSOC);
```

Python

Python DB-API 对于返回结果集和不返回结果集的 SQL 语句没有不同的调用方法。要在 Python 中处理一个语句，你用你的数据库连接对象获得一个游标对象。然后使用游标的 `execute()` 方法向服务器发送语句。如果语句出错失败，`execute()` 抛出一个异常。否则，

如果没有结果集，语句结束。然后你可以使用游标的 `rowcount` 属性来确定多少行发生了改变：

```
cursor = conn.cursor ()  
cursor.execute ("UPDATE profile SET cats = cats+1 WHERE name = 'Fred'")  
print "Number of rows updated: %d" % cursor.rowcount
```

注意 `rowcount` 是一个属性而不是一个方法。使用 `rowcount` 来引用它，而不是 `rowcount()`，否则将会抛出异常。



提示：Python DB-API 规范表示数据库连接开始时应该禁止 `auto-commit` 模式，所以当连接到 MySQL 服务器时，MySQLdb 禁止了 `auto-commit`。一个暗示是如果你使用事务型表，当你关闭连接时对它们的修改将会被回滚，除非你先提交（commit）了修改。对于非事务型数据表如 MyISAM 表是自动提交的，所以这种情况不会发生。要想获得关于 `auto-commit` 模式的更多信息，请参见第 15 章（15.7 节）。

如果语句返回了一个结果集，获取它的行，然后关闭结果集。DB-API 提供了一系列方法用于检索行。`fetchone()` 顺序返回下一行（如果没有其他行时返回 `None`）：

```
cursor = conn.cursor ()  
cursor.execute ("SELECT id, name, cats FROM profile")  
while 1:  
    row = cursor.fetchone ()  
    if row == None:  
        break  
    print "id: %s, name: %s, cats: %s" % (row[0], row[1], row[2])  
print "Number of rows returned: %d" % cursor.rowcount  
cursor.close ()
```

正如你可以从上例所看到的，`rowcount` 属性对于 `SELECT` 语句也是有用的；他表示结果集的行数。

`len(row)` 告诉你结果集的列数。

另一个行获取方法 `fetchall()` 以一个行序列的方式返回整个结果集。你可以迭代整个序列来访问行：

```
cursor = conn.cursor ()  
cursor.execute ("SELECT id, name, cats FROM profile")  
rows = cursor.fetchall ()  
for row in rows:  
    print "id: %s, name: %s, cats: %s" % (row[0], row[1], row[2])  
print "Number of rows returned: %d" % cursor.rowcount  
cursor.close ()
```

DB-API 没有提供方法来重新获得一个结果集，所以当你需要多次迭代访问结果集的行或

者直接访问单个值时，`fetchall()`就比较方便。例如，如果 `rows` 包含结果集，你可以用 `rows[1][2]` 来访问第二行中第三列的值（索引以 0 开始，而不是 1）。

要通过列名来访问行值，当你创建游标对象时指定 `DictCursor` 游标类型。这就使得行作为命名元素的 Python 字典对象返回：

```
cursor = conn.cursor (MySQLdb.cursors.DictCursor)
cursor.execute ("SELECT id, name, cats FROM profile")
for row in cursor.fetchall ():
    print "id: %s, name: %s, cats: %s" % (row["id"], row["name"], row["cats"])
print "Number of rows returned: %d" % cursor.rowcount
cursor.close ()
```

上面的示例还说明了如何直接将 `fetch_all` 用作一个迭代器。

Java

JDBC 接口为 SQL 语句处理的不同阶段提供了特定的对象类型。在 JDBC 中通过使用某种类型的 Java 对象来发起语句，如果有结果，将作为其他类型的对象返回。

要发起一条语句，第一步是通过调用 `Connection` 对象的 `createStatement()` 方法获得一个 `Statement` 对象：

```
Statement s = conn.createStatement ();
```

现在使用 `Statement` 对象向服务器发送语句，JDBC 提供了几个方法来完成此任务。选择适合于逆向发送的语句类型的一个：`executeUpdate()` 用于不返回结果集的语句，`executeQuery()` 用于返回结果集的语句，当你不确定是否返回的时候使用 `execute()`。如果语句出错每个方法都抛出一个异常。

`executeUpdate()` 方法发送一条不生成结果集的语句到服务器，然后返回一个表示所影响的行数的数值。当你使用完 `statement` 对象后，关闭它。下面的示例说明这个事件序列：

```
Statement s = conn.createStatement ();
int count = s.executeUpdate (
    "UPDATE profile SET cats = cats+1 WHERE name = 'Fred'");
s.close (); // 关闭 statement
System.out.println ("Number of rows updated: " + count);
```

对于返回结果集的语句，使用 `executeQuery()`。然后获得一个结果集对象，接着用它来检索行的值。当你完成后，关闭结果集和语句对象：

```
Statement s = conn.createStatement ();
s.executeQuery ("SELECT id, name, cats FROM profile");
ResultSet rs = s.getResultSet ();
int count = 0;
```

```
while (rs.next ()) // 循环结果集的行
{
    int id = rs.getInt (1); // 提取列 1, 2, 3
    String name = rs.getString (2);
    int cats = rs.getInt (3);
    System.out.println ("id: " + id
                        + ", name: " + name
                        + ", cats: " + cats);

    ++count;
}
rs.close (); // 关闭结果集
s.close (); // 关闭 statement
System.out.println ("Number of rows returned: " + count);
```

你的 Statement 对象的 `getResultSet()` 方法返回的 ResultSet 对象自身有许多方法，例如 `next()` 可以获取行，不同的 `getXXX()` 方法可以访问当前行的列。开始时，结果集位于集合的第一行之前。调用 `next()` 顺序获取每一行直至其返回 `false`，这意味着已经到了集合尾部。要确定结果集中的行数，如先前示例所示，自己进行计数。

要访问列值，使用诸如 `getInt()`, `getString()`, `getFloat()` 和 `getDate()` 等方法。要将列值取出作为一个通用对象，使用 `getObject()`。`getXXX()` 调用可以带上一个表示列位置（从 1 开始，而不是 0）或列名的参数进行调用。先前的示例展示了如何根据位置来检索 `id`、`name` 和 `cats` 列。换言之，要根据名称来访问列，该示例获取行的循环可以按以下方式重写：

```
while (rs.next ()) // 循环结果集的行
{
    int id = rs.getInt ("id");
    String name = rs.getString ("name");
    int cats = rs.getInt ("cats");
    System.out.println ("id: " + id
                        + ", name: " + name
                        + ", cats: " + cats);

    ++count;
}
```

你可以使用任意 `getXXX()` 调用来检索一个给定的列值。例如，你可以使用 `getString()` 来将任意列值以字符串形式检索出来：

```
String id = rs.getString ("id");
String name = rs.getString ("name");
String cats = rs.getString ("cats");
System.out.println ("id: " + id
                    + ", name: " + name
                    + ", cats: " + cats);
```

或者你可以使用 `getObject()` 将值作为通用对象检索出来，并进行必要的转换。下面的示例使用 `toString()` 将对象值转化为可输出的形式：

```
Object id = rs.getObject ("id");
Object name = rs.getObject ("name");
Object cats = rs.getObject ("cats");
System.out.println ("id: " + id.toString ()
+ ", name: " + name.toString ()
+ ", cats: " + cats.toString ());
```

要了解结果集中有多少列，访问结果集的元数据：

```
ResultSet rs = s.getResultSet ();
ResultSetMetaData md = rs.getMetaData (); // 获得结果集元数据
int ncols = md.getColumnCount (); // 从元数据中获得列数
```

第三个 JDBC 语句执行方法 `execute()` 可用于任何类型的语句。当你从外部源接收到一个语句字符串，但不知道其是否生成一个结果集时它尤其有用。`execute()` 的返回值表示语句类型，以便你可以对其进行合适的处理：如果 `execute()` 返回 `true`，那就存在一个结果集，反之则没有。特别的，如果你想以如下方式使用它，这里 `stmtStr` 代表任意的 SQL 语句：

```
Statement s = conn.createStatement ();
if (s.execute (stmtStr))
{
    // 存在一个结果集
    ResultSet rs = s.getResultSet ();

    // ... 在此处理结果集...

    rs.close (); // 关闭结果集
}
else
{
    // 没有结果集，只输出行数目
    System.out.println ("Number of rows affected: " + s.getUpdateCount ());
}
s.close (); // 关闭语句
```

2.5 处理语句中的特殊字符和 NULL 值

Handling Special Characters and NULL Values in Statements

问题

你需要构建涉及数据值的 SQL 语句，而这些数据值包含诸如引号或反斜线等特殊字符或者特殊值如 `NULL`。或者你正使用从外部源获得的数据来构建语句，且想避免收到 SQL 的注入攻击。

解决方案

使用你 API 的占位符机制或者引用函数来使数据插入变得安全。

讨论

本章到此为止，我们的语句使用的都是“安全的”数据值，而无需特殊对待。例如，我们可以通过将数据值放入语句字符串中在程序中轻松地构建下面的 SQL 语句：

```
SELECT * FROM profile WHERE age > 40 AND color = 'green'  
INSERT INTO profile (name,color) VALUES('Gary','blue')
```

然而，某些数据值并不能轻松地被处理，如果你不仔细的话就会导致问题。语句可能使用包含特殊字符如引号，反斜线，二进制数据或者 NULL 值的值。下面的讨论描述了这些类型的值引起的问题以及处理他们的适当方法。

假设你想执行这个 INSERT 语句：

```
INSERT INTO profile (name,birth,color,foods,cats)  
VALUES('Alison','1973-01-12','blue','eggroll',4);
```

这并没有什么不同寻常之处。但是如果你将 name 列值修改为如 De'Mont 这样的包含一个单引号的值，语句语法上就不正确了：

```
INSERT INTO profile (name,birth,color,foods,cats)  
VALUES('De'Mont','1973-01-12','blue','eggroll',4);
```

问题在于在一个单引号括起来的字符串里面有一个单引号。要使语句合法，应该在引号前面使用一个单引号或用一个反斜线进行转义：

```
INSERT INTO profile (name,birth,color,foods,cats)  
VALUES('De''Mont','1973-01-12','blue','eggroll',4);  
INSERT INTO profile (name,birth,color,foods,cats)  
VALUES('De\'Mont','1973-01-12','blue','eggroll',4);
```

另外，你也可以使用双引号来引用 name 值而不是使用单引号（假设 ANSI_QUOTESSQL 模式被禁止）：

```
INSERT INTO profile (name,birth,color,foods,cats)  
VALUES("De'Mont",'1973-01-12','blue','eggroll',4);
```

如果你正在你的程序中逐字编写一条语句，你可以手工转义或引用 name 值，因为你知道值是什么。但是如果一个变量保存了 name 值，你并不一定知道变量值是什么。甚或更糟的是，单引号并非你必须处理的唯一字符，双引号和反斜线也会引起问题。同时如果你想

存储诸如图像或音频等二进制数据到你的数据库中，那值可能什么都包含——不只是引号或反斜线，还包括其他字符诸如 null (0 值的字节)。在 Web 环境下合理处理特殊字符显得尤为重要，因为语句是使用表单输入来构建的（例如，如果你正搜索匹配远程用户输入的搜索条件的行）。你必须能用同样一种方式来处理任意输入，因为你不可能预知用户将提供何种信息。事实上，恶意用户有意尝试输入包含问题字符的垃圾值从而危及你服务器的安全，这种事很普遍。这是一种发现可被利用的不安全脚本的标准技术。

SQL NULL 值并不是一个特殊字符，但它也需要特别对待。在 SQL 中，NULL 表示“没有值。”这根据上下文有几种涵义，例如“未知的”、“缺少的”、“超出范围的”等等。为避免处理 NULL 值所引入的复杂性，我们的语句到现在还没有使用过 NULL 值，但现在是解决这个问题的时候了。例如，如果你不知道 De'Mont 喜爱的颜色，你可以将 color 列的值设为 NULL——但不是按如下方式编写语句：

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES ('De''Mont', '1973-01-12', 'NULL', 'eggroll', 4);
```

而是 NULL 值不应该用引号括起来：

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES ('De''Mont', '1973-01-12', NULL, 'eggroll', 4);
```

如果你正在你的程序中逐字输入语句，你只需要简单地写下不带引号的 NULL 值。但如果 color 值来自于一个变量，相应的处理就有些复杂了。你必须知道变量值的相关内容以确定当你构建语句时是否要用引号将它括起来。

有两种方法可用于处理特殊字符如引号和反斜线，以及特殊值如 NULL：

- 在语句字符串中使用占位符来指代数据值，当你执行语句时数据值将绑定到占位符上。一般情况下这是首选的方法，因为 API 本身将会为你完成所有或大部分工作，包括如有必要会为值提供引号、引用或转义数据值中的特殊字符，以及将一个特定值解释为未用引号括起来的 NULL 值。
- 使用一个引用函数（如果你的 API 提供了一个）将数据值转化为适用于语句字符串的一种安全格式。

本节说明如何使用这些技术为每个 API 处理特殊的字符和 NULL 值。这儿的示例之一说明了如何向 profile 表中插入一个 name 值为 De'Mont，color 值为 NULL 的行。然而，这

儿所述的原理有其一般性，可以处理任意特殊字符，包括那些二进制数据中的。（参考第 18 章，可见如何处理图像——它也是一种二进制数据——的示例。）而且，原理不仅限于 INSERT 语句，对于其他类型语句同样适用，例如 SELECT。这儿所述的其他示例之一阐述了如何使用占位符来执行一个 SELECT 语句。

特殊字符还出现于这里没有涵盖的其他上下文中：

- 这里所述的占位符和引用技术仅用于数据值，而不适用于标识符诸如数据库或表名。关于引用标识符问题的讨论，请参考 2.6 节。
- 本节涵盖了将特殊字符插入你的数据库中的主题。这儿未涵盖的一个相关主题是其反向操作——将从你的数据库中返回值的特殊字符进行转换以显示在不同的上下文中。例如，如果你正生成 HTML 页面，其中包含从你的数据库中取出的值，你需要将那些值中的<和>字符转化为 HTML 元素<和>；以确保能正确地显示。17.4 节讨论了这个主题。

使用占位符

占位符可以使你免于逐字输入数据值到 SQL 语句中。所以应使用占位符来编写语句——表示值将进入哪儿的特殊字符。一个普遍的占位符字符是?。对于使用该字符的 API，INSERT 语句可以使用占位符重写如下：

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES(?, ?, ?, ?, ?)
```

然后你将语句字符串传入数据库，并单独提供数据值。值被绑定到占位符以取代他们，从而产生包含数据值的语句。

使用占位符的好处之一在于参数绑定操作自动处理了诸如引号和反斜线等字符的转义。当你正插入诸如图像等二进制数据进入你的数据库中，或者使用诸如远程用户通过一个 Web 页面的表单提交的输入等具备未知内容的数据值时，这一点尤其有用。通常你还可以绑定一些特殊值到一个占位符上，以表明你在最终的语句中需要一个 SQL NULL 值。

占位符的第二个好处在于你可以预先“准备”一条语句，然后通过在其每次执行时绑定不同的值达到重用的目的。某些数据库接口具有此功能，它允许在执行语句之前进行某些预解析甚至执行。对于一条以后会被多次执行的语句，这会缩减其开销，因为任何可以在执行之前做的事情仅需执行一次，而不是每次执行都执行一遍。准备好的语句因此提高了语句的重用性。

语句因为包含了占位符而不是特定的数据值而变得更为通用。如果你正反复执行一个操作，你应该可以重用一条准备好的语句并在你每次执行它时绑定不同的值到其上即可。这样做可以获得性能提升，至少对于支持查询计划的数据库系统来说是如此。例如，如果一个程序执行时多次发起一个特定类型的 SELECT 语句，该数据库系统可以为语句创建一个计划，然后每次重用它，而不是反复重新构建计划。MySQL 并不预先构建查询计划，所以使用准备好的语句你不能获得任何性能提升。然而，如果你将一个程序迁移到一个不使用查询计划的数据库，并且你已经使用准备好的语句编写你的程序，你就能自动获得准备好的语句的好处。你没有必要从非准备好的语句进行转化来获得这个好处。

第三个好处在于使用基于占位符的语句的代码可读性更好，尽管这有些主观因素。当你读完本节时，将这儿所用的语句和 2.4 节未使用占位符的语句做个比较，看看你更倾向于哪个。

生成占位符列表

你不能将一个数据值的数组绑定到一个单一的占位符上。每个值都必须和一个独立的占位符绑定。如果你想为一个数据值列表使用占位符，而这个列表在数目上是可变的，你必须构建一个占位符列表。例如，在 Perl 中，下面的语句创建了一个包含由逗号分隔的 n 个占位符的字符串：

```
$str = join (",", ("?") x n);
```

X 是重复操作符，当应用于一个列表时，生成列表的 n 个拷贝，所以 join() 调用将这些列表聚合起来生成一个单一的字符串，其中包含 n 个逗号分隔的?字符。当你试图将一个数据值绑定到一个语句字符串中的占位符列表时就很方便，因为数组的大小表明了需要多少占位符字符：

```
$str = join (",", ("?") x @values);
```

在 Ruby 中，* 操作符有同样的效果：

```
str = ([?] * values.size).join(",")
```

另一个更为易懂的生成占位符列表的 Perl 方法如下：

```
$str = "?" if @values;
for (my $i = 1; $i < @values; $i++)
{
    $str .= ",?";
}
```

这个方法中 Perl 特定的语法较少，因此更易于转化为其他语言。例如，Python 中等价的方法如下：

```
str = ""
if len (values) > 0:
    str = "?"
for i in range (1, len (values)):
    str = str + ","?
```

使用一个引用函数

有些 API 提供了一个引用函数，它接收数据值作为其参数并返回一个被合理引用和转义的值，使其适于安全地插入一条 SQL 语句中。这与使用占位符相比显得不太普遍，但它对构建你不想立即执行的语句比较有用。然而，当你使用这样一个引用函数时，你不需要开启一个到数据库服务器的连接，因为直到获悉数据库驱动时才能确定合适的引用规则。(有些数据库系统的引用规则和其他的系统不同。)

Perl

要在 Perl DBI 脚本中使用占位符，将?放在 SQL 语句中你想插入数据值的每个位置，然后将值绑定到语句中。你可以通过将值传给 `do()` 或 `execute()`，或者调用特定的用于占位符替换的 DBI 方法来绑定值。

使用 `do()`，你可以在同一调用中传递语句字符串和数据值来为 De'Mont 添加 profile 数据行：

```
my $count = $dbh->do ("INSERT INTO profile (name,birth,color,foods,cats)
    VALUES(?, ?, ?, ?, ?)",
    undef,
    "De' Mont", "1973-01-12", undef, "eggroll", 4);
```

语句字符串后面的参数应为 `undef`，其后应紧跟数据值，使每个占位符对应一个。(紧跟语句字符串的 `undef` 参数是一个历史遗留问题，但是它必须出现。)

另外，可以使用 `prepare()` 和 `execute()`。将语句字符串传给 `prepare()` 获得一个 statement 句柄，然后使用该句柄通过 `execute()` 传递数据值：

```
my $sth = $dbh->prepare ("INSERT INTO profile (name,birth,color,foods,cats)
    VALUES(?, ?, ?, ?, ?)");
my $count = $sth->execute ("De' Mont", "1973-01-12", undef, "eggroll", 4);
```

如果你需要反复发起同一语句，你可以使用一次 `prepare()`，然后每次在你需要执行语句时调用 `execute()`。

在任一情形下，DBI 生成的最终语句如下所示：

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De\' Mont','1973-01-12',NULL,'eggroll','4')
```

注意 DBI 是如何在数据值周围添加引号的，即使在原先的语句字符串中?占位符周围并没有引号。(占位符机制在数字值周围也添加引号，但那是可以的，因为 MySQL 服务器会执行必要的类型转换将字符串转化为数值。) 还要注意一下 DBI 的约定，当你将 undef 绑定到一个占位符时，DBI 会将 NULL 放入语句中并正确地在其周围添加引号。

你也同样可以为其他类型的语句使用这些方法。例如，下面的 SELECT 语句使用占位符来查找 cats 值大于 2 的数据行：

```
my $sth = $dbh->prepare ("SELECT * FROM profile WHERE cats > ?");
$sth->execute (2);
while (my $ref = $sth->fetchrow_hashref ())
{
    print "id: $ref->{id}, name: $ref->{name}, cats: $ref->{cats}\n";
}
```

绑定值到占位符的另一个方法是使用 bind_param() 调用。它接收两个参数，占位符位置和要绑定到该位置占位符的值。(占位符位置从 1 开始，而不是 0。) 前面的 INSERT 和 SELECT 可以使用 bind_param() 重写如下：

```
my $sth = $dbh->prepare ("INSERT INTO profile (name,birth,color,foods,cats)
                           VALUES(?, ?, ?, ?, ?)");
$sth->bind_param (1, "De'Mont");
$sth->bind_param (2, "1973-01-12");
$sth->bind_param (3, undef);
$sth->bind_param (4, "eggroll");
$sth->bind_param (5, 4);
my $count = $sth->execute ();

my $sth = $dbh->prepare ("SELECT * FROM profile WHERE cats > ?");
$sth->bind_param (1, 2);
$sth->execute ();
while (my $ref = $sth->fetchrow_hashref ())
{
    print "id: $ref->{id}, name: $ref->{name}, cats: $ref->{cats}\n";
}
```

不管你对占位符使用哪种方法，都不要在?字符周围放任何引号，即使是表示字符串的占位符也不要，DBI 如有必要会自己添加引号。事实上，如果你在占位符字符周围放了引号，DBI 会将其解释为文本字符常量"?，而不是一个占位符。

高级查询方法诸如 selectrow_array() 和 selectall_arrayref() 也可以和占位符一起使用。和 do() 方法相似，参数是语句字符串和 undef，后面紧跟要绑定到出现于语句字符串中的占位符的数据值。示例如下：

```
my $ref = $dbh->selectall_arrayref (
    "SELECT name, birth, foods FROM profile
```

```
WHERE id > ? AND color = ?",
undef, 3, "green");
```

Perl DBI 还提供了一个数据库句柄方法 `quote()` 作为使用占位符的可选项。下面是如何使用 `quote()` 创建一个语句字符串将一个新行插入到 `profile` 表的示例：

```
my $stmt = sprintf ("INSERT INTO profile (name,birth,color,foods,cats)
VALUES(%s,%s,%s,%s,%s)",
$dbh->quote ("De'Mont"),
$dbh->quote ("1973-01-12"),
$dbh->quote (undef),
$dbh->quote ("eggroll"),
$dbh->quote (4));
my $count = $dbh->do ($stmt);
```

这段代码生成的语句字符串和你使用占位符时生成的一样。格式化符号 `%s` 未用引号括起来是因为 `quote()` 如有必要会自动添加：非 `undef` 的值会加上引号插入，`undef` 值会无引号以 `NULL` 插入。

Ruby

Ruby DBI 使用 `?` 作为 SQL 语句中的占位符字符，并用 `nil` 作为绑定 SQL `NULL` 值到占位符的值。

要用 `do` 来使用占位符，将语句字符串以及其后要绑定到占位符的数据值传给它：

```
count = dbh.do("INSERT INTO profile (name,birth,color,foods,cats)
VALUES(?, ?, ?, ?, ?)",
"De'Mont", "1973-01-12", nil, "eggroll", 4)
```

可选的，将语句字符串传给 `prepare` 来获得一个 `statement` 句柄，然后使用该句柄用数据值来调用 `execute`：

```
sth = dbh.prepare("INSERT INTO profile (name,birth,color,foods,cats)
VALUES(?, ?, ?, ?, ?)")
count = sth.execute("De'Mont", "1973-01-12", nil, "eggroll", 4)
```

不管你如何构建语句，DBI 包含一个适当的转义后的引号和适当的未引用的 `NULL` 值：

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De\'Mont', '1973-01-12', NULL, 'eggroll', 4)
```

使用 `prepare` 加 `execute` 的方法对于用不同的数据值多次执行的语句来说很有用。对于仅执行一次的语句，你可以跳过 `prepare` 步骤。将语句字符串传给数据库句柄的 `execute` 方法，后面紧跟着数据值。下面的示例使用此方法处理了一条 `SELECT` 语句：

```
sth = dbh.execute("SELECT * FROM profile WHERE cats > ?", 2)
sth.fetch do |row|
  printf "id: %s, name: %s, cats: %s\n", row["id"], row["name"], row["cats"]
```

```
end  
sth.finish
```

Ruby DBI 占位符机制在数据值被绑定到语句字符串时，如有必要会在这些值周围加上引号，所以在字符串中不要在?字符周围放置引号。

Ruby DBI `quote()` 数据库句柄方法是占位符的一个可选方案。下例使用 `quote()` 为 De'Mont 生成 INSERT 语句：

```
stmt = sprintf "INSERT INTO profile (name,birth,color,foods,cats)  
VALUES (%s,%s,%s,%s,%s)",  
            dbh.quote("De'Mont"),  
            dbh.quote("1973-01-12"),  
            dbh.quote(nil),  
            dbh.quote("eggroll"),  
            dbh.quote(4)  
count = dbh.do(stmt)
```

这段代码生成的语句字符串和你使用占位符时生成的一样。格式化符号`%s`未用引号括起来是因为 `quote()` 如有必要会自动添加：非 `nil` 的值会加上引号插入，`nil` 值会无引号以 `NULL` 插入。

PHP

PEAR DB 模块允许占位符被用于执行 SQL 语句的 `query()` 方法中，或者你可以使用 `prepare()` 准备一条语句，然后 `execute()` 来提供数据值并执行准备好的语句。PEARDB 使用`?`作为 SQL 语句中的占位符字符，当绑定 SQL `NULL` 值到一个占位符时用 PHP `NULL` 作为所用的值。

使用 `query()`，传给它一个语句字符串，后面紧跟一个数组，该数组中包含要绑定到占位符的数据值：

```
$result =& $conn->query ("INSERT INTO profile (name,birth,color,foods,cats)  
VALUES (?, ?, ?, ?, ?)",  
array ("De'Mont", "1973-01-12", NULL, "eggroll", 4));  
if (PEAR::isError ($result))  
    die ("Oops, the statement failed");
```

也可以将语句字符串传给 `prepare()` 获得一个语句对象，将这个对象和数据值数组传给 `execute()`：

```
$stmt =& $conn->prepare ("INSERT INTO profile (name,birth,color,foods,cats)  
VALUES (?, ?, ?, ?, ?)");  
if (PEAR::isError ($stmt))  
    die ("Oops, statement preparation failed");  
$result =& $conn->execute ($stmt,  
array ("De'Mont", "1973-01-12", NULL, "eggroll", 4));  
if (PEAR::isError ($result))  
    die ("Oops, statement execution failed");
```

使用任一种方法构建的语句都包含一个适当的转义后的引号和适当的未引用的 NULL 值：

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De\'Mont','1973-01-12',NULL,'eggroll',4)
```

如果只有一个占位符，数据值数组仅有一个成员：

```
$result = & $conn->query ("SELECT * FROM profile WHERE cats > ?", array (2));
if (PEAR::isError ($result))
    die ("Oops, the statement failed");
while ($row = & $result->fetchRow (DB_FETCHMODE_ASSOC))
{
    printf ("id: %s, name: %s, cats: %s\n",
           $row["id"], $row["name"], $row["cats"]);
}
$result->free ();
```

在这种情况下，你可以不使用数组来指定数据值。下面的两个 query() 调用是等同的：

```
$result = & $conn->query ("SELECT * FROM profile WHERE cats > ?", array (2));
$result = & $conn->query ("SELECT * FROM profile WHERE cats > ? ", 2)
```

PEARDB 占位符机制在数据值被绑定到语句字符串时，如有必要会在这些值周围加上引号，所以在字符串中不要在?字符周围放置引号。

PEARDB quoteSmart() 方法可被用于替代占位符来引用数据值。下面的示例在使用 quoteSmart() 构建语句字符串后为 De'Mont 插入了一行数据：

```
$stmt = sprintf ("INSERT INTO profile (name,birth,color,foods,cats)
                  VALUES(%s,%s,%s,%s,%s)",
                  $conn->quoteSmart ("De'Mont"),
                  $conn->quoteSmart ("1973-01-12"),
                  $conn->quoteSmart (NULL),
                  $conn->quoteSmart ("eggroll"),
                  $conn->quoteSmart (4));
$result = & $conn->query ($stmt);
if (PEAR::isError ($result))
    die ("Oops, the statement failed");
```

PEAR DB 还有一个 escapeSimple() 引用方法，但它比不上 quoteSmart()。例如，它不能正确处理 NULL 值。

Python

Python 的 MySQLdb 模块在 SQL 语句字符串中使用格式化符号来实现占位符。要使用占位符，可调用带两个参数的 execute() 方法：一个包含格式化符号的语句字符串和一个包含要绑定到语句字符串的值的序列。要为 De'Mont 添加一个 profile 表数据行，代码如下：

```
cursor = conn.cursor ()
cursor.execute """
    INSERT INTO profile (name,birth,color,foods,cats)
    VALUES(%s,%s,%s,%s,%s)
    """, ("De'Mont", "1973-01-12", None, "eggroll", 4))
```

某些 Python DB-API 驱动模块支持几个格式化符号（例如 %d 用于整数，%f 用于浮点数）。使用 MySQLdb，你应该使用 %s 占位符将所有数据值作为字符串来进行格式化。MySQL 将执行必要的类型转换。要将一个文本%字符放入语句中，在语句字符串中使用 %%。

参数绑定机制必要时会在数据值周围加上引号。DB-API 将 None 逻辑上视作与 SQL NULL 值等同，所以你可以将 None 绑定到一个占位符上从而在语句字符串中生成一个 NULL。前面的 execute() 调用发送到服务器的语句如下：

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES ('De\'Mont', '1973-01-12', NULL, 'eggroll', 4)
```

如果仅有一个值 val 绑定到一个占位符上，你可以使用语法(val,)将其写成一个序列。下面的 SELECT 语句说明了这点：

```
cursor = conn.cursor ()
cursor.execute ("SELECT * FROM profile WHERE cats = %s", (2,))
for row in cursor.fetchall ()�
    print row
cursor.close ()
```

Python 占位符机制在数据值被绑定到语句字符串时，如有必要会在这些值周围加上引号，所以在字符串中不要在 %s 格式化符号周围放置引号。

使用 MySQLdb 引用数据值的一个可选方法是使用 literal() 方法。要使用 literal() 为 De'Mont 生成 INSERT 语句，应该这么做：

```
cursor = conn.cursor ()
stmt = """
    INSERT INTO profile (name,birth,color,foods,cats)
    VALUES(%s,%s,%s,%s,%s)
    """ % \
        (conn.literal ("De'Mont"), \
         conn.literal ("1973-01-12"), \
         conn.literal (None), \
         conn.literal ("eggroll"), \
         conn.literal (4))
cursor.execute (stmt)
```

Java

如果你使用准备好的语句，JDBC 为占位符提供了支持。回忆一下在 JDBC 中发起未预先准备好的语句的过程是创建一个 Statement 对象，然后把语句字符串传给语句发起的方法之一 executeUpdate()、executeQuery() 或 execute()。

而使用预先准备好的语句，是通过将包含?占位符字符的语句字符串传给你的连接对象的 `prepareStatement()` 方法创建一个 `PreparedStatement` 对象，然后使用 `setXXX()` 方法将你的数据值绑定到语句上。最后，通过使用一个空参数列表来调用 `executeUpdate()`、`executeQuery()` 或 `execute()` 执行语句。

下面是一个示例，它使用 `executeUpdate()` 发起一条 `INSERT` 语句为 De'Mont 向 `profile` 表中插入一行：

```
PreparedStatement s;
int count;
s = conn.prepareStatement (
    "INSERT INTO profile (name,birth,color,foods,cats)"
    + " VALUES(?, ?, ?, ?, ?)");
s.setString (1, "De'Mont");           // 绑定数据值到占位符
s.setString (2, "1973-01-12");
s.setNull (3, java.sql.Types.CHAR);
s.setString (4, "eggroll");
s.setInt (5, 4);
count = s.executeUpdate ();
s.close (); // 关闭语句
```

绑定数据值到语句的 `setXXX()` 方法接收两个参数：占位符位置.(从 1 开始，而不是 0) 和绑定到占位符的值。选择每个绑定值调用以匹配值以绑定列的数据类型：`setString()` 用于绑定一个字符串到 `name` 列，`setInt()` 用于绑定一个整数到 `cats` 列，等等。(实际上，我在此有意使用 `setString()` 日期值如 `birth` 当作一个字符串。)

JDBC和其他 API 之间的差异之一在于你无须通过指定某个特殊值(例如在 Perl 中的 `undef` 或 Ruby 中的 `nil`) 以将 `NULL` 绑定到一个占位符上。代之的是，你调用一个特殊的方法 `setNull()`，其中第二个参数表示列的类型(对于字符串是 `java.sql.Types.CHAR`，对于整数是 `java.sql.Types.INTEGER`，等等)。

`setXXX()` 调用如有必要会在数据值周围加上引号，所以在字符串中不要在语句字符串中的?占位符字符周围放置引号。

对于一条返回结果集的语句，准备的过程是相似的，但是你使用 `executeQuery()` 来执行预先准备好的语句：

```
PreparedStatement s;
s = conn.prepareStatement ("SELECT * FROM profile WHERE cats > ?");
s.setInt (1, 2); // 将 2 绑定到第一个占位符
s.executeQuery ();
// ...在此处理结果集 ...
s.close (); // 关闭语句
```

2.6 处理标识符中特殊字符

Handling Special Characters in Identifiers

问题

你需要构建涉及包含特殊字符的标识符的 SQL 语句。

解决方案

引用标识符以便他们能被安全地插入语句字符串中。

讨论

2.5 节讨论了如何使用占位符或引用方法来处理数据值中的特殊字符。特殊字符也可能出现于诸如数据库、表或列名等标识符中。例如，表名 `some table` 包含一个空格，这在默认情况下是不允许的：

```
mysql> CREATE TABLE some table (i INT);
ERROR 1064 (42000): You have an error in your SQL syntax near 'table (i INT)'
```

标识符中的特殊字符和数据值的字符处理起来有所不同。要使标识符能安全地插入一条 SQL 语句中，通过将其用反引号括起来进行引用：

```
mysql> CREATE TABLE `some table` (i INT);
Query OK, 0 rows affected (0.04 sec)
```

如果一个引用字符出现在标识符中，引用这样的标识符时内部的引号使用成对的引用字符。例如，用``abc` `def``来引用 `abc` `def``。

在 MySQL 中，后引号总是允许用于标识符引用的。如果 `ANSI_QUOTES` SQL 模式开启，双引号字符对于引用标识符也是合法的。这样，在 `ANSI_QUOTES` SQL 模式开启时下面的两条语句是等价的：

```
CREATE TABLE `some table` (i INT);
CREATE TABLE "some table" (i INT);
```

如果必须知道哪个标识符引用字符是可用的，发起一个 `SELECT @@sql_mode` 语句来查询 SQL 模式，并检查其值是否包含 `ANSI_QUOTES`。

要知道尽管 MySQL 中的字符串通常可以使用单引号或双引号字符进行引用(`'abc'`, `"abc"`)，但当 `ANSI_QUOTES` 开启时那并不是正确的。在这种情况下，MySQL 将 `'abc'` 解释为一个字符串，而 `"abc"` 解释为一个标识符，所以你只能将单引号用于字符串。

在一个程序中，如果你的 API 提供了一个标识符引用程序，你就可以使用它，或者如果没有提供，你可以自己写一个。Perl DBI 有一个 `quote_identifier()` 方法返回一个相应的

引用标识符。对于没有类似方法的 API，你可以通过将其用后引号括起来引用一个标识符，当标识符中存在后引号时使其成双即可。下面是一个完成该任务的 Ruby 程序：

```
def quote_identifier(ident)
  return `"` + ident.gsub(/\`/, ``) + `"`
end
```

如果你可以假设标识符内部没有后引号，你可以简单地将其用后引号括起来即可。

注意：如果你编写自己的标识符应用程序，记住其他的 DBMS 可能需要不同的引用习惯。

在某些上下文中，标识符可能被用作数据值，且也应作为数据值进行处理。如果你从元数据数据库 INFORMATION_SCHEMA 查询信息，通常通过在 WHERE 子句中指定数据库对象名来表示返回哪些行。例如，这条语句查询 cookbook 数据库中的 profile 表的列名：

```
SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'cookbook' AND TABLE_NAME = 'item';
```

数据库和表名在这里被用作数据值，而不是标识符。你在程序中构建这条语句时，应使用占位符来将其参数化，而不是用标识符引用。例如，在 Ruby 中你可能如下撰写：

```
names = dbh.select_all(
  "SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
  WHERE TABLE_SCHEMA = ? AND TABLE_NAME = ?",
  db_name, tbl_name)
```

2.7 识别结果集中的 NULL 值

Identifying NULL values in results from your API

问题

查询结果包含 NULL 值，但是你不太确定怎么获悉他们的位置。

解决方案

你的 API 可能有一些特殊的值表示 NULL。你只需要知道它是什么以及怎么对它进行测试。

讨论

2.5 节描述了当你向数据库服务器发送语句时如何引用 NULL 值。本节中，我们会反过来处理如何识别和处理从数据库服务器返回的 NULL 值的问题。通常，要了解 API 将 NULL 值映射到哪个特殊值上或者调用什么方法是问题所在。这些值如下表所示：

语言	检测 NULL 的值或方法
Perl DBI	undef 值
Ruby DBI	nil 值
PHP PEAR DB	NULL 或 unset 值
Python DB-API	None 值
Java JDBC	wasNull() 方法

下面的部分展示了一个 NULL 值检测的一个简单应用。示例检索一个结果集并输出其中的所有值，并将 NULL 值映射到可输出的字符串“NULL”。

为确保 profile 表包含一些 NULL 值的行，使用 mysql 来发起下面的 INSERT 语句，并接着发起 SELECT 语句以确认结果行有所期望的值：

```
mysql> INSERT INTO profile (name) VALUES('Juan');
mysql> SELECT * FROM profile WHERE name = 'Juan';
+----+-----+-----+-----+-----+
| id | name | birth | color | foods |
+----+-----+-----+-----+-----+
| 11 | Juan | NULL | NULL | NULL |
+----+-----+-----+-----+-----+
```

id 列可以包含不同的数字，但其他的列必须如上所示，具有 NULL 值。

Perl

Perl DBI 使用 undef 来表示 NULL 值。使用 defined() 函数来检测该值很方便，当你使用 Perl -w 选项或者通过在你的脚本中包括一行 use warnings 开启了警告功能，这样做就尤为重要。否则，访问 undef 值会导致 Perl 发出如下抱怨：

```
Use of uninitialized value
```

要避免这个警告，在使用可能为 undef 的列值之前使用 defined() 来检测它们。下面的代码从 profile 的列中选取部分并在每行中为任何未定义的值输出“NULL”。这就使 NULL 值在输出中显式可见而不触发任何警告信息：

```
my $sth = $dbh->prepare ("SELECT name, birth, foods FROM profile");
$sth->execute ();
while (my $ref = $sth->fetchrow_hashref ())
{
    printf "name: %s, birth: %s, foods: %s\n",
        defined ($ref->{name}) ? $ref->{name} : "NULL",
        defined ($ref->{birth}) ? $ref->{birth} : "NULL",
        defined ($ref->{foods}) ? $ref->{foods} : "NULL";
}
```

不幸的是，所有列值的测试显得很呆板，并且列越多就越糟糕。为避免这点，你可以在输出它们之前在循环中测试并设置未定义的值。你编写的用于执行测试的代码数量是不变的，而不是与被测试的列的数目成比例。循环也不引用任何特定的列名，所以它能更方便地被拷贝粘贴到其他的程序中或用来作为工具程序的基础：

```
my $sth = $dbh->prepare ("SELECT name, birth, foods FROM profile");
$sth->execute ();
while (my $ref = $sth->fetchrow_hashref ())
{
    foreach my $key (keys (%{$ref}))
    {
        $ref->{$key} = "NULL" unless defined ($ref->{$key});
    }
    printf "name: %s, birth: %s, foods: %s\n",
           $ref->{name}, $ref->{birth}, $ref->{foods};
}
```

如果你从行取入一个数组而不是一个哈希表的话，你可以使用 map 来转化任何 undef 值：

```
my $sth = $dbh->prepare ("SELECT name, birth, foods FROM profile");
$sth->execute ();
while (my @val = $sth->fetchrow_array ())
{
    @val = map { defined ($_) ? $_ : "NULL" } @val;
    printf "name: %s, birth: %s, foods: %s\n",
           $val[0], $val[1], $val[2];
}
```

Ruby

Ruby DBI 使用 nil 来表示 NULL 值，它可以通过对值使用 nil? 方法来识别。下面的示例使用 nil? 来确定是否直接输出结果集值或者为 NULL 值输出字符串 “NULL”：

```
dbh.execute("SELECT name, birth, foods FROM profile") do |sth|
    sth.fetch do |row|
        for i in 0...row.length
            row[i] = "NULL" if row[i].nil? # 列值是 NULL?
        end
        printf "id: %s, name: %s, cats: %s\n", row[0], row[1], row[2]
    end
end
```

循环的一个更简短的替代品是 collect! 方法，它轮流接收每个数组元素并用代码块的返回值来替代它：

```
row.collect! { |val| val.nil? ? "NULL" : val }
```

PHP

PHP 用 PHP NULL 值来代表结果集中的 SQL NULL 值。要确定结果集中的某个值是否代表一个 NULL 值，使用 === 三元相等操作符将其与 PHP NULL 值进行比较：

```
if ($val === NULL)
{
    # $val 是一个 NULL 值
}
```

在 PHP 中，三元相等操作符意味着“精确相等”。普通的 == 相等比较操作符在此并不适合。如果你使用 ==，PHP 认为 NULL 值，空字符串和 0 都彼此相等。

使用 === 来测试 NULL 值的一个替代物是使用 iset()：

```
if (!isset ($val))
{
    # $val 是一个 NULL 值
}
```

下面的代码使用 === 操作符来识别结果集中的 NULL 值并将它们作为字符串 "NULL" 输出：

```
$result = & $conn->query ("SELECT name, birth, foods FROM profile");
if (PEAR::isError ($result))
    die ("Oops, the statement failed");
while ($row = & $result->fetchRow ())
{
    foreach ($row as $key => $value)
    {
        if ($row[$key] === NULL)
            $row[$key] = "NULL";
    }
    print ("name: $row[0], birth: $row[1], foods: $row[2]\n");
}
$result->free ();
```

Python

Python DB-API 程序使用 None 来表示结果集中的 NULL。下面的示例展示了如何检测 NULL 值：

```
cursor = conn.cursor ()
cursor.execute ("SELECT name, birth, foods FROM profile")
for row in cursor.fetchall ():
    row = list (row) # 将不可变的元组变成可变列表
    for i in range (0, len (row)):
        if row[i] == None: # 列值为 NULL?
            row[i] = "NULL"
    print "name: %s, birth: %s, foods: %s" % (row[0], row[1], row[2])
cursor.close ()
```

内部的循环通过查找 None 来检测 NULL 列值，并将它们转化为字符串 "NULL"。注意示例

中如何在循环之前将行转化为一个可变的对象，`fetchall()`将行作为序列值返回，它是不可变的（只读的）。

Java

对于 JDBC 程序，如果结果集中的一列可能包含 `NULL` 值，最好是显式地对它们进行检查。完成此任务的方法是获取该值然后调用 `wasNull()`，如果列为 `NULL` 它返回 `true`，否则返回 `false`。例如：

```
Object obj = rs.getObject (index);
if (rs.wasNull ())
{ /* 值是 NULL */ }
```

上面的例子使用 `getObject()`，但原理同样适用于其他 `getXXX()` 调用。

下面是一个实例，它将结果集中的每行作为一个逗号分隔的值列表输出，对于每个 `NULL` 值输出为“`NUL`”：

```
Statement s = conn.createStatement ();
s.executeQuery ("SELECT name, birth, foods FROM profile");
ResultSet rs = s.getResultSet ();
ResultSetMetaData md = rs.getMetaData ();
int ncols = md.getColumnCount ();
while (rs.next ()) // 循环遍历结果集的行
{
    for (int i = 0; i < ncols; i++) // 对列进行循环
    {
        String val = rs.getString (i+1);
        if (i > 0)
            System.out.print (", ");
        if (rs.wasNull ())
            System.out.print ("NULL");
        else
            System.out.print (val);
    }
    System.out.println ();
}
rs.close (); // 关闭结果集
s.close (); // 关闭语句
```

2.8 获取连接参数的技术

Techniques for Obtaining Connection Parameters

问题

你需要为一个脚本获取连接参数以便它能连接到 MySQL 服务器。

解决方案

有许多方法可以做到这点。你可以在下列方法中任意选择一种。

讨论

连接到 MySQL 的任何程序都需要指定连接参数，例如用户名，密码和主机名。到目前为止所示的方案都直接将连接参数放入试图建立连接的代码中，但那并非你的程序获取参数的唯一方式。本节详细阐述你可以使用的技术，并接着说明如何实现他们中的两个。

将参数硬编码到程序中

参数可以在源文件中或者程序所使用的库文件中给出。此技术很是方便，因为用户不需要自己输入值。其缺点就是不太灵活，要改变参数，你必须修改程序。

交互式请求参数

在命令行环境中，你可以向用户询问一系列问题。在 Web 或 GUI 环境中，这可以通过展现一个表单或对话框来达到。不论哪一种方式，对于频繁使用应用的人来说显得很繁冗，因为每次都需要输入参数。

从命令行获取参数

此方法可以用于你交互式运行的命令或在脚本中运行的命令。类似于交互式获取参数的方法，这要求你每次使用 MySQL 的时候提供参数，因此同样烦人。（能显著减轻这个负担的一个要素是很多 shell 允许你轻松地从你的历史列表里记起命令以用于再次执行。）

从执行环境获取参数

使用此方法最常用的方式是在你的 shell 的启动文件之一中设置合适的环境变量（例如对于 sh、bash、ksh 是.profile；对于 csh 或 tcsh 是.login）。然后在你的登录会话期间你所运行的程序能通过检查他们的环境来获得参数值。

从一个独立的文件中获取参数

使用这个方法，你将诸如用户名和密码等信息存储在一个文件中，而程序在连接到 MySQL 服务器之前可以读取这个文件。当允许你避免将值硬编码到程序自身时，从与你的程序分开的文件中读取参数可以使你获得每次你使用程序时无须输入它们的好处。这项技术对于交互式的程序尤为便捷，因为每次你运行程序时都不需要输入参数。同时，将值储存在一个文件里使你可以集中管理多个程序使用的参数，并且

你可以出于安全目的使用文件访问模式。例如，你可以通过将文件模式设为只允许你访问来阻止其他用户读取文件。

MySQL 客户端库本身提供了一个可选项文件机制，尽管并非所有的 API 都提供对它的访问能力。对于那些不能使用的，也有其特定的背景存在。（例如，Java 支持 properties 文件的使用，并提供了读取他们的工具程序。）

使用一组方法

结合以上几种方法一起使用也常很有用，它给予用户以不同方式提供参数的灵活性。例如，MySQL 客户端如 mysql 和 mysqladmin 在几个位置寻找选项文件并进行读取然后检查命令行参数有没有其他的参数。这就使用户可以在一个选项文件或命令行中指定连接参数。

这些获取连接参数的方法包括一些安全事项。下面是详细说明：

- 将连接参数存储在文件内的任一种方法都会危及你系统的安全，除非文件被保护以免受未授权用户的访问。无论参数是存在一个源文件、一个选项文件或者调用一条命令和在命令行指定参数指定的脚本中，这都适用。（仅能被 Web 服务器读取的 Web 脚本在其他用户有服务器的管理员权限时也是不安全的。）
- 在命令行或环境变量中指定的参数也并不是十分安全。当一个程序在运行时，它的命令行参数和环境对于其他运行进程状态命令如 ps -e 的用户是可见的。特别是，将密码存在环境变量中最好限于你是机器的唯一用户或你信任所有其他用户的情形下。

本节余下部分将介绍如何处理命令行参数来获取连接参数以及如何从选项文件中读取参数。

从命令行获取参数

MySQL 命令行参数的一般性约定（也就是标准的客户端如 mysql 和 mysqladmin 所遵循的约定）既允许使用短选项又允许使用长选项来指定参数。例如，用户名 cbuser 可以使用 -u cbuser（或 -ucbuser）或 --user=cbuser 来指定。另外，对于密码选项 (-p 或 --password) 的任一种，可在选项名后略去密码值以表示程序应该交互式地提示输入密码。

接下来的一系列示例程序说明如何处理命令参数来获得主机名、用户名和密码。这些的标准标识是 -h 或 --host、-u 或 --user 以及 -p 或 --password。你可以编写自己的代码对参数列表进行迭代，但是一般来说，使用已有的专为此目的编写的选项处理模块更为方便。

要使一个脚本使用其他选项，如`--port`或`--socket`，你可以使用所示的代码但要扩展选项指定数组以包含额外的选项。如果给定选项值，你还需要稍微修改一下建立连接的代码来使用他们。

对于这儿所述的 API (Perl、Ruby、Python)，程序使用`getopt()`风格的函数。对于 Java，在`recipes`发行包的`api`目录下可以找到这儿没有说明的示例代码，以及它的使用指南。



提示：示例尽可能地模拟标准 MySQL 客户端的选项处理行为。

例外在于选项处理库不允许使密码值可选，并且如果密码选项没有指定密码值，他们没有提供交互式提示用户输入密码的方法。

结果就是，编写出来的示例脚本在你使用`-p`或`--password`时，必须在选项后提供密码值。

Perl

Perl 通过`@ARGV` 数组将命令行参数传给脚本，该数组可以使用`Getopt::Long` 模块的`GetOptions()`函数处理。下面的程序说明如何解析命令参数来获得连接参数。

```
#!/usr/bin/perl
# cmdline.pl - 说明 Perl 中的命令行选项解析

use strict;
use warnings;
use DBI;

use Getopt::Long;
$Getopt::Long::ignorecase = 0; # 选项是大小写相关的
$Getopt::Long::bundling = 1;   # 允许绑定短格式选项

# 连接参数——默认所有的都未初始化 (undef)
my $host_name;
my $password;
my $user_name;

GetOptions (
    # =s 表示选项后需要跟一个字符串值
    "host|h=s"      => \$host_name,
    "password|p=s"   => \$password,
    "user|u=s"       => \$user_name
) or exit (1); # 不需要错误信息；GetOptions()输出其自身的

# 所有剩下的非选项参数都可保留在@ARGV 中，并可在那里做必要的处理

# 构建数据源名称
```

```

my $dsn = "DBI:mysql:database=cookbook";
$dsn .= ";host=$host_name" if defined ($host_name);

# 连接到服务器
my %conn_attrs = (PrintError => 0, RaiseError => 1, AutoCommit => 1);
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs);
print "Connected\n";

$dbh->disconnect ();
print "Disconnected\n";

```

GetOptions()的参数是选项标识符和选项值所放置的脚本变量的引用。选项标识符列出了选项的长格式和短格式（没有前面的破折号），如果选项需要后跟值，标识符后会紧跟=s。例如，“host|h=s”允许--host 和-h，并表示后面需要一个字符串值。你无须传递@ARGV数组，因为GetOptions()隐式使用它。当GetOptions()返回时，@ARGV包含所有余下的非选项参数。

Ruby

Ruby程序通过ARGV数组来访问命令行参数，你可以使用GetoptLong.new()方法来处理该数组。下面的程序使用这个方法解析命令参数获取连接参数：

```

#!/usr/bin/ruby -w
# cmdline.rb - 说明 Ruby 中的命令行选项解析

require "getoptlong"
require "dbi"

# 连接参数-默认所有的都未初始化(nil)
host_name = nil
password = nil
user_name = nil

opts = GetoptLong.new(
  [ "--host",      "-h",    GetoptLong::REQUIRED_ARGUMENT ],
  [ "--password",  "-p",    GetoptLong::REQUIRED_ARGUMENT ],
  [ "--user",       "-u",    GetoptLong::REQUIRED_ARGUMENT ]
)

# 对选项进行遍历，解出所有的值；opt 是长格式选项，arg 是其值
opts.each do |opt, arg|
  case opt
  when "--host"
    host_name = arg
  when "--password"
    password = arg
  when "--user"
    user_name = arg
  end
end

# 所有剩下的非选项参数都保留在ARGV中，并可在这里做必要的处理

```

```

# 构建数据源名称
dsn = "DBI::Mysql::database=cookbook"
dsn << ";host=#{host_name}" unless host_name.nil?

# 连接到服务器
begin
  dbh = DBI.connect(dsn, user_name, password)
  puts "Connected"
rescue DBI::DatabaseError => e
  puts "Cannot connect to server"
  puts "Error code: #{e.err}"
  puts "Error message: #{e.errstr}"
  exit(1)
end

dbh.disconnect()
puts "Disconnected"

```

要处理 ARGV 数组，使用 GetoptLong.new() 方法，并将表示要识别的选项的信息传给它。这个方法的每个参数是一个三值数组：

- 长格式选项名；
- 短格式选项名；
- 表示选项是否需要一个值的标志。可选的标志是 GetoptLong::NO_ARGUMENT（选项不带值），GetoptLong::REQUIRED_ARGUMENT（选项需要值），以及 GetoptLong::OPTIONAL_ARGUMENT（选项值可选）。对于示例程序，所有的选项都需要一个值。

Python

Python 通过 sys.argv 变量将命令参数作为一个列表传给脚本。你可以通过导入 sys 和 getopt 模块来访问此变量并处理它的内容。下面的程序说明如何从命令参数获取参数并用它们来建立到服务器的连接：

```

#!/usr/bin/python
# cmdline.py - 说明 Python 中的命令行选项解析

import sys
import getopt
import MySQLdb

try:
    opts, args = getopt.getopt (sys.argv[1:],
                               "h:p:u:",
                               [ "host=", "password=", "user=" ])
except getopt.error, e:
    # 对于错误，输出程序名以及错误信息文本
    print "%s: %s" % (sys.argv[0], e)
    sys.exit (1)

# 默认的连接参数值（都为空）

```

```

host_name = password = user_name = ""

# 遍历选项，取出所有值
for opt, arg in opts:
    if opt in ("-h", "--host"):
        host_name = arg
    elif opt in ("-p", "--password"):
        password = arg
    elif opt in ("-u", "--user"):
        user_name = arg

#所有剩下的非选项参数都保留在 args 中，并可在这里做必要的处理

try:
    conn = MySQLdb.connect (db = "cookbook",
                           host = host_name,
                           user = user_name,
                           passwd = password)
    print "Connected"
except MySQLdb.Error, e:
    print "Cannot connect to server"
    print "Error:", e.args[1]
    print "Code:", e.args[0]
    sys.exit (1)

conn.close ()
print "Disconnected"

```

`getopt()`接收两个或三个参数：

- 要处理的命令参数列表。这不应该包括程序名`sys.argv[0]`，所以使用`sys.argv[1:]`引用程序名后面的参数列表。
- 命名短格式选项字符的一个字符串。在`cmdline.py`中，这些字符串每个都后跟一个冒号`(:)`以表示选项需要一个紧跟的值。
- 一个长格式选项名称的列表。在`cmdline.py`中，每个名称后跟`=`以表示选项需要一个紧跟的值。

`getopt()`返回两个值。第一个是一个选项/值对列表，第二个是紧跟最后一个选项的所有剩下的非选项参数列表。`Cmdline.py`遍历选项列表以确定有哪些选项以及他们的值。注意尽管你在传给`getopt()`的选项名称里没有指定之前的破折号，函数返回的名称里还是包括了前面的破折号。

从选项文件获取参数

如果你的 API 允许，你可以在一个 MySQL 选项文件中指定连接参数，API 将从文件中读取参数。对于不直接支持选项文件的 API，你可以安排读取参数所被存储的其他类型文件或者编写你自己的读取选项文件的函数。

1.4 节描述了 MySQL 选项文件的格式。我将假设你已经读过了那里的讨论并在这里可以把注意力集中在如何在程序中使用选项文件。在 Unix 下，用户特定的选项根据管理在`~/.my.cnf`（也就是在你的 home 目录中的`.my.cnf`文件中）中指定。然而，MySQL 选项文件机制在存在几个不同文件时（并非必须存在选项文件）可以对其进行查找。标准的搜索顺序是`/etc/my.cnf`、MySQL 安装目录中的`my.cnf`文件，以及当前用户的`~/.my.cnf`文件。在 Windows 下，你可以使用的选项文件是你的 MySQL 安装目录（例如，`C:\Program Files\MySQL\MySQL Server 5.0`）中`my.ini`文件，你的 Windows 目录（如`C:\Windows`或`C:\WINNT`）中的`my.ini`，或者`my.cnf`文件。

如果存在多个选项文件并且一个给定的参数在他们几个中都进行指定，最后发现的值将具有最高优先级。

MySQL 选项文件不被你自己的程序所使用，除非你让他们这么做：

- Perl、Ruby 和 Python 提供了读取选项文件的直接 API 支持，简单地表示你想在连接到服务器的时候使用他们。指定仅读取一个特定的文件或者用于查找多个选项文件的标准搜索顺序是可能的。
- PHP 和 Java 不支持选项文件。对于 PHP，我们将编写一个简单的选项文件解析函数。对于 Java，我们将采取一个不同的方法——使用属性文件。

尽管 Unix 下的用户特定的选项文件的约定名称是当前用户的 home 目录中的`.my.cnf`，也没有规定你的程序必须使用这个特定的文件。你可以任你所好地命名一个选项文件，并可以将它放在任何你所想要的位置。例如，你可以建立一个名为`mcb.cnf`的文件并将其安装在`/usr/local/lib/mcb`目录供脚本使用来访问`cookbook`数据库。在有些场景下，你甚至可能想创建多个选项文件。然后，在任意给定的脚本中，你可以选择适于脚本所需权限类别的文件。例如，你可能有一个选项文件`mcb.cnf`列出了一个完全访问的 MySQL 用户的参数，以及另一个文件`mcb-ro.cnf`列出了一个仅有 MySQL 只读访问权限的用户的连接参数。另一个可能性是在同一个选项文件中列出多个组，然后让你的脚本从合适的组中选择选项。

Perl

Perl DBI 脚本可以使用选项文件。要利用这一点，将相应的选项标识符放在数据源名称字符串的第三个部分：

- 要使用`mysql_read_default_group=groupname`指定一个选项组。这会让 MySQL 搜索标准的选项文件以获得在命名组和`[client]`组中的选项。组名值是不带方括号的，那应该是开始组的行的一部分。例如，如果选项文件中的某个组以`[my_prog]`行开始，它制定了`my_prog`作为组名的值。要搜索标准的文件，但仅在`[client]`组中找到了，组名应该是`client`。

- 要命名一个特定的选项文件，在DSN中使用`mysql_read_default_file=filename`。当你这么做时，MySQL仅会查看那个文件，且仅查找[client]组中的选项。
- 如果你既指定了一个选项文件又指定了一个选项组，MySQL仅会读取命名的文件，但是会寻找所命名的组和[client]组中的选项。

下面的示例让MySQL使用标准的选项文件搜索顺序来查找[cookbook]和[client]组中的选项：

```
my %conn_attrs = (PrintError => 0, RaiseError => 1, AutoCommit => 1);
# 基本的 DSN
my $dsn = "DBI:mysql:database=cookbook";
# 查看标准的选项文件；使用[cookbook]和[client]组
$dsn .= ";mysql_read_default_group=cookbook";
my $dbh = DBI->connect ($dsn, undef, undef, \%conn_attrs);
```

接下来的示例显式地命名位于`$ENV{HOME}`中的选项文件——用户运行脚本的home目录。这样，MySQL将会查看那个文件并使用[client]组中的选项：

```
my %conn_attrs = (PrintError => 0, RaiseError => 1, AutoCommit => 1);
# 基本的 DSN
my $dsn = "DBI:mysql:database=cookbook";
# 查看当前用户所拥有的用户特定的选项文件
$dsn .= ";mysql_read_default_file=$ENV{HOME}/.my.cnf";
my $dbh = DBI->connect ($dsn, undef, undef, \%conn_attrs);
```

如果你将一个空值（`undef`或者空字符串）传给`connect()`调用的用户名和密码参数，`connect()`将使用在选项文件中所发现的值。`connect()`调用中的非空的用户名或密码会过载任何选项文件值。同样的，DSN中所命名的主机名也会过载任何选项文件值。你可以使用此行为以如下方式使DBI脚本都从选项文件和命令行获取连接参数：

- 创建`$host_name`、`$user_name`和`$password`变量并将其初始化为`undef`。然后，如果命令行中出现相应的选项就解析命令行参数将变量设为非`undef`值。（本节之前所示的处理命令行参数的Perl脚本`cmdline.pl`说明了如何做到这点。）
- 在解析命令参数后，构建DSN字符串并调用`connect()`。使用DSN中的`mysql_read_default_group`和`mysql_read_default_file`指定你想如何使用选项文件，并在`$host_name`不为`undef`时，将`host=$host_name`加到DSN中。除此之外，将`$user_name`和`$password`作为用户名和密码参数传给`connect()`，这些默认为`undef`。如果他们在命令行参数中做了设置，他们将会拥有过载选项文件值的非`undef`值。

如果一个脚本遵循此过程，用户在命令行中给出的参数被传给 `connect()` 并比选项文件的内容具有更高的优先级。

Ruby

Ruby DBI 脚本可以使用类似于 Perl DBI 的机制来访问选项文件，下面的示例对应于前面 Perl 讨论中的示例。

这个例子使用标准的选项文件搜索顺序来查找 [cookbook] 和 [client] 两个组中的选项：

```
# 基本的 DSN
dsn = "DBI:mysql:database=cookbook"
# 查看标准选项文件；使用 [cookbook] 和 [client] 组
dsn << ";mysql_read_default_group=cookbook"
dbh = DBI.connect(dsn, nil, nil)
```

下面的示例使用当前用户的 home 目录中的.my.cnf 文件从 [client] 组获取参数：

```
# 基本的 DSN
dsn = "DBI:mysql:database=cookbook"
# 查看当前用户所拥有的用户特定的选项文件
dsn << ";mysql_read_default_file=#{ENV['HOME']}/.my.cnf"
dbh = DBI.connect(dsn, nil, nil)
```

PHP

PHP 没有使用 MySQL 选项文件的原生支持。要突破此限制，可以使用一个读取选项文件的函数，例如下面所列述的 `read_mysql_option_file()` 函数。它接收一个选项文件名和一个选项组名或者一个包含组名的数组作为参数。（组名的命名应该不包括方括号）它接着读取选项文件中的命名组或组集合中的选项。如果没有给定选项组参数，默认的函数将查看 [client] 组。返回值是一个选项名/值对的数组，或者发生错误时返回 FALSE，它并非文件不存在的错误。（注意引用的选项值和紧跟选项值的 #- 风格注释在 MySQL 选项文件中是合法的，但是这个函数并不处理这些结构。）

```
function read_mysql_option_file ($filename, $group_list = "client")
{
    if (is_string ($group_list))          # 将字符串转化为数组
        $group_list = array ($group_list);
    if (!is_array ($group_list))          # 哇...垃圾参数？
        return (FALSE);
    $opt = array ();                      # 选项名 / 值数组
    if (!@($fp = fopen ($filename, "r")))
        # 如果文件不存在,
        return ($opt);                  # 返回一个空列表
    $in_named_group = 0;                  # 当处理一个命名组时设为非零值
    while ($s = fgets ($fp, 1024))
    {
        $s = trim ($s);
        if ($s{0} == '#')              # 忽略注释
            continue;
        if ($s{0} == '[')              # 处理一个命名组
        {
            $group = substr ($s, 1, -1);
            if (!in_array ($group, $group_list))
                continue;
            $in_named_group = 1;
        }
        else if ($in_named_group)
        {
            $key = substr ($s, 0, -1);
            $val = substr ($s, -1);
            $opt[$key] = $val;
        }
    }
    return ($opt);
}
```

```

if (ereg ("^#[;]", $s))                      # 跳过注释
    continue;
if (ereg ("^\[((^])+)\]", $s, $arg))  # 选项组行?
{
    #检查我们是否处于所要的组中
    $in_named_group = 0;
    foreach ($group_list as $key => $group_name)
    {
        if ($arg[1] == $group_name)
        {
            $in_named_group = 1;      # 在所要的组中
            break;
        }
    }
    continue;
}
if (!$in_named_group)           # 不在所要的组中, 跳过该行
    continue;
if (ereg ("^([^\t=]+)\t*[=\t]*(.*)", $s, $arg))
    $opt[$arg[1]] = $arg[2];      # name=value
else if (ereg ("^([^\t]+)", $s, $arg))
    $opt[$arg[1]] = "";          # 仅名称
# else line is malformed
}
return ($opt);
}

```

下面是一些说明如何使用 `read_mysql_option_file()` 的示例。第一个读取用户的选项文件获得 [client] 组的参数，然后使用他们来连接到服务器。第二个读取系统范围的选项文件，`/etc/my.cnf`，并输出所发现的服务器启动参数（也就是在 [mysqld] 和 [server] 组中的参数）：

```

$opt = read_mysql_option_file ("/u/paul/.my.cnf");
$dsn = array
(
    "phptype"  => "mysqli",
    "username"  => $opt["user"],
    "password"  => $opt["password"],
    "hostspec"  => $opt["host"],
    "database"  => "cookbook"
);
$conn =& DB::connect ($dsn);
if (PEAR::isError ($conn))
    print ("Cannot connect to server\n");

$opt = read_mysql_option_file ("/etc/my.cnf", array ("mysqld", "server"));
foreach ($opt as $name => $value)
    print ("$name => $value\n");

```

Python

DB-API 的 MySQLdb 模块提供了使用 MySQL 选项文件的直接支持。使用 `read_default_file` 或 `read_default_group` 参数指定一个选项文件或选项组到 `connect()` 方法。这两

个参数的作用和 Perl DBI connect() 方法的 mysql_read_default_file 和 mysql_read_default_group 选项一样（参考本节前面关于 Perl 的讨论）。要使用标准的选项文件搜索顺序来查找 [cookbook] 和 [client] 组中的选项，照下面的方式做：

```
conn = MySQLdb.connect (db = "cookbook", read_default_group = "cookbook")
```

下面的例子说明如何使用当前用户 home 目录中的 .my.cnf 文件从 [client] 组获取参数：

```
option_file = os.environ["HOME"] + "/" + ".my.cnf"
conn = MySQLdb.connect (db = "cookbook", read_default_file = option_file)
```

要访问 os.environ 你必须导入 os 模块。

Java

MySQL Connector/J JDBC 驱动不支持选项文件。然而，Java 类库提供了支持以读取包含 name=value 格式的行的属性文件。这有些类似于 MySQL 选项文件格式，尽管有一些不同（例如，属性文件不允许 [groupname] 行）。下面是一个简单的属性文件：

```
# 本文件列出了连接到 MySQL 服务器的参数
user=cbuser
password=cbpass
host=localhost
```

下面的程序 ReadPropsFile.java 展现了一种读取名为 Cookbook.properties 的属性文件以获取连接参数的方法。文件必须位于你的 CLASSPATH 环境变量中所指明的目录下，否则你必须使用完整的路径名来指定它（此处的示例假定文件位于 CLASSPATH 目录中）：

```
import java.sql.*;
import java.util.*; // 需要这个以支持属性文件

public class ReadPropsFile
{
    public static void main (String[] args)
    {
        Connection conn = null;
        String url = null;
        String propsFile = "Cookbook.properties";
        Properties props = new Properties ();

        try
        {
            props.load (ReadPropsFile.class.getResourceAsStream (propsFile));
        }
        catch (Exception e)
        {
            System.err.println ("Cannot read properties file");
            System.exit (1);
        }
        try
        {
            // 构建连接 URL，在尾部对用户名和密码进行编码使其作为参数
        }
    }
}
```

```
url = "jdbc:mysql://" + props.getProperty("host") + "/cookbook" + "?user=" + props.getProperty("user") + "&password=" + props.getProperty("password"); Class.forName("com.mysql.jdbc.Driver").newInstance(); conn = DriverManager.getConnection(url); System.out.println("Connected"); } catch (Exception e) { System.err.println("Cannot connect to server"); } finally { try { if (conn != null) { conn.close(); System.out.println("Disconnected"); } } catch (SQLException e) { /* 忽略 close 错误 */ } } }
```

如果你需要在未发现命名属性 `getProperty()` 时返回一个特定的默认值，将该值作为第二个参数传入。例如，要使用 `127.0.0.1` 作为默认的主机值，可按照如下方式调用 `getProperty()`：

```
String hostName = props.getProperty("host", "127.0.0.1");
```

本章先前所开发的 `Cookbook.java` 库文件（见 2.3 节）包含一个另外的库调用，你可以在 `recipes` 发行包的 `lib` 目录中找到对应的文件版本：一个基于这儿所讨论的概念的 `propsConnect()` 程序。要使用它，建立属性文件的内容 `Cookbook.properties`，并将文件拷贝到你安装 `Cookbook.class` 的同一个位置。然后你可以通过导入 `Cookbook` 类和调用 `Cookbook.propsConnect()` 而不是调用 `Cookbook.connect()` 在程序中建立一个连接。

2.9 结论和建议

Conclusion and Words of Advice

本章讨论了我们的每种 API 所提供的处理与 MySQL 服务器进行交互的不同方面的基本操作。这些操作使你可以编写程序发起任何种类的语句并查询结果。到此为止，我们使用的都是简单的语句，因为重点在于 API 而不是 SQL。下面的章节将集中注意力在 SQL 上，以说明如何向数据库询问更复杂的问题。

在你继续之前，最好将本章中使用的 `profile` 表重置为已知的状态。好让后几章的一些语句通过对其进行重新初始化来使用这个表，当你执行那儿所示的语句时，你会获得与那些章节中所显示的一致的结果。要重置该表，切换位置到 `recipes` 发行包的 `tables` 目录，执行下面的命令：

```
% mysql cookbook < profile.sql  
% mysql cookbook < profile2.sql
```

从表中查询数据

Selecting Data from Tables

3.0 引言

Introduction

本章主要关注用于数据查询的 SELECT 语句。在这一章你可以看到如何使用 SELECT 语句从 MySQL 获取需要的信息。如果你原来只了解 SQL 的皮毛或者你想了解更多 MySQL 对 SELECT 语句的特定扩展，那么本章正适合你。

SELECT 语句的书写方法众多，在这里我们只列举部分方法。如果需要使用更多的 SELECT 语法、函数和操作来查询和处理数据库信息，请参考《MySQL Reference Manual》或者一个通用的 MySQL 文本。

SELECT 允许你对行查询的几个方面进行控制：

- 所要查询的表；
- 所要查询的行和列；
- 重命名查询结果列；
- 对查询结果进行排序。

许多有用的查询都非常简单，不需特殊的操作。例如，在有些 SELECT 查询中甚至不需要指名所要查询的数据库表。例如在 1.28 节中我们曾经讨论了如何将 mysql 作为计算器使用。其他不基于表的查询对于某些目的很有用，例如，查看当前数据库服务器版本信息或者默认的数据库名：

```
mysql> SELECT VERSION(), DATABASE();
+-----+-----+
| VERSION() | DATABASE() |
+-----+-----+
| 5.0.27-log | cookbook |
+-----+-----+
```

为回答更多相关问题，通常你需要从一个或多个表中获取信息。本章的多数示例使用名为

mail 的表，该表的数据行记录一系列主机上的用户之间的邮件信息。mail 表定义如下所示：

```
CREATE TABLE mail
(
    t      DATETIME,  # 消息发送的时间
    srcuser CHAR(8),   # 发送者 (来源用户和主机)
    srchost CHAR(20),
    dstuser CHAR(8),   # 接收者 (目标用户和主机)
    dsthost CHAR(20),
    size    BIGINT,    # 消息的字节大小
    INDEX (t)
);
```

其记录的内容如下：

t	srcuser	srchost	dstuser	dsthost	size
2006-05-11 10:15:08	barb	saturn	tricia	mars	58274
2006-05-12 12:48:13	tricia	mars	gene	venus	194925
2006-05-12 15:02:49	phil	mars	phil	saturn	1048
2006-05-13 13:59:18	barb	saturn	tricia	venus	271
2006-05-14 09:31:37	gene	venus	barb	mars	2291
2006-05-14 11:52:17	phil	mars	tricia	saturn	5781
2006-05-14 14:42:21	barb	venus	barb	venus	98151
2006-05-14 17:03:01	tricia	saturn	phil	venus	2394482
2006-05-15 07:17:48	gene	mars	gene	saturn	3824
2006-05-15 08:50:57	phil	venus	phil	venus	978
2006-05-15 10:25:52	gene	mars	tricia	saturn	998532
2006-05-15 17:35:31	gene	saturn	gene	mars	3856
2006-05-16 09:00:28	gene	venus	barb	mars	613
2006-05-16 23:04:19	phil	venus	barb	venus	10294
2006-05-17 12:49:23	phil	mars	tricia	saturn	873
2006-05-19 22:21:51	gene	saturn	gene	venus	23992

为创建并载入 mail 表，将命令行的位置变换到 recipes 发行包的 tables 目录下，然后运行如下命令：

```
% mysql cookbook < mail.sql
```

在这一章里，我们也会时常用到其他数据库表，其中有的在前面的章节中已经使用过，有的则是新出现的。和创建 mail 表类似，可以在 tables 目录下使用和类似的脚本创建其他数据库表。另外，本章所用到的大多数脚本和程序都可以在 select 目录下找到。这个目录下的文件可以让读者更方便地动手尝试例子。

你可以在第 1 章所讨论的 mysql 程序内运行这里所示的大多数语句来执行它们。还有一些示例是在编程语言的上下文中发起语句的。请参考第 2 章来获取编程技术的相关背景知识。

3.1 指定查询列/从指定列中查询

Specifying Which Columns to Select

问题

用户希望可以查询表中一列或多列信息。

解决方案

可以使用精简的 SQL 语句 `SELECT * FROM table_name` 查询所有列，但这样的查询结果是得到所查询的数据表中所有列的一个无序集合。如果希望只对部分列进行查询，或者希望查询结果按照一定的顺序排列，那么你要么以要求的顺序显式地命名列，要么把它们检索出来放入某个数据结构中，而该数据结构中它们的顺序不再相关。

讨论

通过指定数据库表及其中的一列或多列，来查询所需要的信息。最简单的方法是使用 `SELECT * FROM table_name` 获取一个数据库表中每一列的信息。其中的 * 说明对数据库表中的所有列进行查询，例如：

```
mysql> SELECT * FROM mail;
+-----+-----+-----+-----+-----+
| t    | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+
| 2006-05-11 10:15:08 | barb    | saturn   | tricia  | mars    | 58274 |
| 2006-05-12 12:48:13 | tricia  | mars    | gene    | venus   | 194925 |
| 2006-05-12 15:02:49 | phil    | mars    | phil    | saturn  | 1048  |
| 2006-05-13 13:59:18 | barb    | saturn   | tricia  | venus   | 271   |
...

```

使用*来指定查询列非常简单，但是用户不能指定每一列在查询结果中出现的先后顺序。显式地指定查询列的一个好处在于，查询结果可以按照用户指定的任意顺序显示。例如，你希望查询结果集中 ‘hostname’ 在 ‘username’ 之前而不是之后出现，要达到这个目的，可以使用下面的语句指定查询列：

```
mysql> SELECT t, srchost, srcuser, dsthost, dstuser, size FROM mail;
+-----+-----+-----+-----+-----+
| t    | srchost | srcuser | dsthost | dstuser | size |
+-----+-----+-----+-----+-----+
| 2006-05-11 10:15:08 | saturn  | barb    | mars    | tricia  | 58274 |
| 2006-05-12 12:48:13 | mars    | tricia  | venus   | gene    | 194925 |
| 2006-05-12 15:02:49 | mars    | phil    | saturn  | phil    | 1048  |
| 2006-05-13 13:59:18 | saturn  | barb    | venus   | tricia  | 271   |
...

```

显式地指定查询列的另外一个好处在于，在查询结果中可以只显示用户所关注的列，而不是显示每一列的信息。例如：

```
mysql> SELECT t, srcuser, srchost, size FROM mail;
+-----+-----+-----+-----+

```

t	srcuser	srchost	size
2006-05-11 10:15:08	barb	saturn	58274
2006-05-12 12:48:13	tricia	mars	194925
2006-05-12 15:02:49	phil	mars	1048
2006-05-13 13:59:18	barb	saturn	271
...			

上例使用 mysql 程序比较了在 SELECT 语句中使用 * 将所有列指定为查询列,以及通过在 SELECT 语句中显示的指定查询列,影响查询的结果。当你在自己的程序内部执行 SQL 语句进行查询,这两种不同的 SELECT 语句书写方式,将影响你对查询结果的处理过程。如果你使用 * 查询,服务器将按照数据库表定义中各列的先后顺序返回查询结果(这个查询结果中每一列的顺序都将随着数据库表定义的改变而改变)。此时如果你把查询结果中的一行保存到一个数组中,并根据对应的列序号来存取信息,那么你可能因为数据库表定义的改变,而无法确定数组中每一个值应该对应到数据库表中的哪一列。如果显示的指定了查询列,你就可以根据每一列在 SELECT 语句中出现的先后顺序从每一个行中读取相应数据,而不至于混淆。

另外你所使用的编程语言可能提供了这样的 API: 允许你将查询结果的每一行保存到一个特定的数据结构中,然后根据列名来获取相应的信息。例如, Perl 和 Ruby 提供了一个 hash 的方法; PHP 则提供了一个关联数组和一个对象。如果有这样的 API,你就可以使用 SELECT * 进行查询,然后根据列名获取相应数据结构的成员。这样的话,数据库表的每一列在查询结果中出现的先后顺序就无关紧要了,也就是说使用*,或者显示指定查询列的效率对于程序而言是相同的,这也使得 SELECT * 这种简单的查询方式更具诱惑力。然而,当你并不需要使用到数据库中每一列的信息,通过显式地指明查询列,可以避免服务器返回多余信息,从而提高系统运行的效率。在第 9.8 节中“选择除确定数据列以外的所有信息”部分给出的例子中更加细致地解释了避免查询部分列的原因。

3.2 指定查询行

Specifying Which Rows to Select

问题

用户希望查询结果中只出现符合特定条件的行。

解决方案

使用 WHERE 语句设定查询条件,使得查询结果只包含符合查询条件的行。例如指定查询生活在某个城市的客户信息,或者状态为“finished”的任务。

讨论

如果不对 SELECT 查询做任何限制，查询结果都将包括一个数据库表的所有行，但并不是其中的每一行都是用户所需要的。为了进行更加精确的查询，可以通过使用 WHERE 语句，给 SELECT 查询加上一个或者多个条件限制。

所使用的查询条件可以是相等、不相等或者是相关顺序等。对于字符串一类的数据类型，还可以使用模式匹配作为查询条件。下面的查询语句从数据库表 mail 中查询 srchost 的值等于 venus 或者 srchost 值以 ‘s’ 开头的行记录：

```
mysql> SELECT t, srcuser, srchost FROM mail WHERE srchost = 'venus';
+-----+-----+-----+
| t      | srcuser | srchost |
+-----+-----+-----+
| 2006-05-14 09:31:37 | gene    | venus   |
| 2006-05-14 14:42:21 | barb    | venus   |
| 2006-05-15 08:50:57 | phil    | venus   |
| 2006-05-16 09:00:28 | gene    | venus   |
| 2006-05-16 23:04:19 | phil    | venus   |
+-----+-----+-----+
mysql> SELECT t, srcuser, srchost FROM mail WHERE srchost LIKE 's%';
+-----+-----+-----+
| t      | srcuser | srchost |
+-----+-----+-----+
| 2006-05-11 10:15:08 | barb    | saturn  |
| 2006-05-13 13:59:18 | barb    | saturn  |
| 2006-05-14 17:03:01 | tricia  | saturn  |
| 2006-05-15 17:35:31 | gene    | saturn  |
| 2006-05-19 22:21:51 | gene    | saturn  |
+-----+-----+-----+
```

在上面的查询语句中使用操作符 LIKE 进行模式匹配。其中 % 是通配符，代表一个长度为任意值的字符串，我们将在第 5.10 节中详细讨论模式匹配。

在一次查询中，可以使用 WHERE 语句定义作用于多列的多个查询条件，下面的查询语句可以查询到所有 barb 发给 tricia 的邮件：

```
mysql> SELECT * FROM mail WHERE srcuser = 'barb' AND dstuser = 'tricia';
+-----+-----+-----+-----+-----+
| t      | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+
| 2006-05-11 10:15:08 | barb    | saturn  | tricia  | mars    | 58274 |
| 2006-05-13 13:59:18 | barb    | saturn  | tricia  | venus   | 271  |
+-----+-----+-----+-----+-----+
```

3.3 格式化显示查询结果

Giving Better Names to Query Result Columns

问题

用户希望自定义显示查询结果（列名或者对应值）。

解决方案

使用列别名来作为你所选择的名称。

讨论

当你查询一个结果集时，MySQL 会赋予每个输出列一个名称。（这就是 mysql 如何将你所见到的名称显示为结果集输出中列头部的起始行。）根据 CREATE TABLE 或 ALTER TABLE 语句中定义的数据库表列名，MySQL 自动命名查询结果的每一列，并将这些列名作为查询结果显示的表头信息。同时用户也可以根据自己的需要来定义查询结果中每一列的列名，以及该列值的显示格式。

我们将在本节讨论什么是别名（alias），以及如何使用别名在查询语句中重命名查询结果列。如果你需用查询的是列本身的信息，而非列值（例如需要查询列名本身），请参考第 9.2 节。

如果在查询语句中没有重命名列名，MySQL 将直接使用数据库表中定义的列名作为查询结果中的列名。在下面的查询语句中通过列名指定查询三列信息，所指定的列名除了限定了查询列之外，同时也作为显示查询结果的列名：

```
mysql> SELECT t, srcuser, size FROM mail;
+-----+-----+-----+
| t      | srcuser | size  |
+-----+-----+-----+
| 2006-05-11 10:15:08 | barb   | 58274 |
| 2006-05-12 12:48:13 | tricia | 194925|
| 2006-05-12 15:02:49 | phil   | 1048  |
| 2006-05-13 13:59:18 | barb   | 271   |
...
...
```

如果使用一个运算表达式在查询结果中插入一列，那么该表达式就作为查询结果的默认列名。这样可能导致查询结果中有的列名长且含义不明确，如下例中，使用一个运算表达式来格式化查询结果：

```
mysql> SELECT
    -> CONCAT(MONTHNAME(t), ' ', DAYOFMONTH(t), ', ', YEAR(t)),
    -> srcuser, size FROM mail;
+-----+-----+-----+
| CONCAT(MONTHNAME(t), ' ', DAYOFMONTH(t), ', ', YEAR(t)) | srcuser | size  |
+-----+-----+-----+
| May 11, 2006          | barb   | 58274 |
| May 12, 2006          | tricia | 194925|
| May 12, 2006          | phil   | 1048  |
| May 13, 2006          | barb   | 271   |
...
...
```

在上例中，我们有意使用一个冗长的表达式在查询结果中定义一个新列。在实际使用中，可以使用 DATE_FORMAT 可以得到一样的查询结果，而不会使用如上例一样丑陋的查询语

句。但是，就算使用 DATE_FORMAT 函数简化了查询语句，在显示结果中的列名还是很长，并且没有什么实际意义。

```
mysql> SELECT
-> DATE_FORMAT(t, '%M %e, %Y'),
-> srcuser, size FROM mail;
+-----+-----+-----+
| DATE_FORMAT(t, '%M %e, %Y') | srcuser | size |
+-----+-----+-----+
| May 11, 2006                 | barb    | 58274 |
| May 12, 2006                 | tricia  | 194925|
| May 12, 2006                 | phil    | 1048   |
| May 13, 2006                 | barb    | 271    |
...

```

为给一个输出列赋予一个你自己选择的名称，使用 AS name 子句来指定列别名（关键字 AS 是可选的）。下面的查询结果和前面的一致，但要将第一列重命名为 date_sent：

使用别名可以使查询结果中的列名简洁、易读并且可以赋予一定的意义。别名可以是任意的一个单词，也可以是任意一个词组，但同时也要遵守一定的定义规则。例如：如果所使用的别名中包含 SQL 保留字、空格、特殊字符或者仅包含数字，必须给所定义的别名加上引号。如下的查询语句执行与上例相同的查询操作，所不同的是，在查询语句中使用别名定义了一个简单、易读的列名：

```
mysql> SELECT
-> DATE_FORMAT(t, '%M %e, %Y') AS 'Date of message',
-> srcuser AS 'Message sender', size AS 'Number of bytes' FROM mail;
+-----+-----+-----+
| Date of message | Message sender | Number of bytes |
+-----+-----+-----+
| May 11, 2006     | barb        |      58274 |
| May 12, 2006     | tricia      |      194925|
| May 12, 2006     | phil        |      1048   |
| May 13, 2006     | barb        |      271    |
...

```

使用别名不但可以定义从一个数据库表中查询的结果列，也可用来定义其他的所有查询结果，例如：

```
mysql> SELECT '1+1+1' AS 'The expression', 1+1+1 AS 'The result';
+-----+-----+
| The expression | The result |
+-----+-----+
```

```
+-----+-----+
| 1+1+1 |      3 |
+-----+
```

在上面的查询结果中，第一列的值是‘1+1+1’（在查询语句中加上了引号，以说明‘1+1+1’是一个字符串，而非一个表达式），第二列的值是 1+1+1，也就是 3（在查询语句中没有加引号，说明是一个表达式，而非字符串）。这里别名是两个有实际含义的词组，用来说 明查询结果中两列之间的关系。

如果在查询语句中使用一个单词作为别名进行查询出现语法错误时，这个作为别名使用的 单词很有可能是 MySQL 的保留字。可以给它加上引号使语法正确。例如：

```
mysql> SELECT 1 AS INTEGER;
You have an error in your SQL syntax near 'INTEGER' at line 1
mysql> SELECT 1 AS 'INTEGER';
+-----+
| INTEGER |
+-----+
|       1   |
+-----+
```

3.4 使用列别名来简化程序

Using Column Aliases to Make Programs Easier to Write

问题

如果查询结果的某一个列是由一个运算表达式求得的结果，通过列名在用户程序中使用列 值就不是那么方便，特别是在所使用的运算表达式比较复杂的情况下。

解决方案

通过使用别名的方法，给所有使用的查询结果列一个简单的列名，方便引用。

讨论

在上一节中讨论了如何使用列别名使得查询结果的列名更具有可读性。除此之外，使用列 别名对于在用户程序中进行 SQL 查询也是非常有用的。在用户程序中，如果将查询结果的 每一行保存到一个数组中，然后通过每一列对应的下标从该数组中取得所需的列值，那么 使用列别名并不会带来什么好处。因为使用列别名并不会改变查询结果中的列顺序，也 不会在查询结果中删除或插入别的列。但是当用户通过列名来访问输出列时，别名就会产 生很大的差异，因为别名改变了列的名称。例如，当在查询语句中使用表达式 DATE_FORMAT (t, '%M %e, %Y') 来格式化输出从 mail 表中查询到的时间，所使用的表达式也就是 用户 程序用来取得对应列值时所使用的列名。显然，使用这个表达式作为列名来获取列值是 比较繁琐的。在查询语句中如果使用 AS data_sent 语句来给所查询的列一个简单的别名， 用户 程序就可以使用词组 data_sent 来获取列值。如下 Perl DBI 程序将数据库查询结果 保存在 hash 中，然后通过列别名获取列值：

```
$sth = $dbh->prepare ("SELECT srcuser,
                           DATE_FORMAT(t, '%M %e, %Y') AS date_sent
                      FROM mail");
$sth->execute ();
while (my $ref = $sth->fetchrow_hashref ())
{
    printf "user: %s, date sent: %s\n", $ref->(srcuser), $ref->(date_sent);
}
```

在 Java 语言中也有类似的用法如下，其中 `getString()` 方法的参数即是所需列值的列名或列别名：

```
Statement s = conn.createStatement ();
s.executeQuery ("SELECT srcuser,
                  + " DATE_FORMAT(t, '%M %e, %Y') AS date_sent"
                  + " FROM mail");
ResultSet rs = s.getResultSet ();
while (rs.next ()) // 循环迭代结果集的行
{
    String name = rs.getString ("srcuser");
    String dateSent = rs.getString ("date_sent");
    System.out.println ("user: " + name + ", date sent: " + dateSent);
}
rs.close ();
s.close ();
```

在 Ruby 语言中，查询结果中的每一行是作为一个对象存在的，用户程序既可以使用列对应下标，也可以使用列别名来获取列值。在 PHP 中，数据库操作模块 PEAR DB 将查询结果的每一行保存在一些关联数组或者对象中，这两种数据结构都允许通过列别名获取列值。Python 使用一个游标（cursor）类将查询结果的每一行作为一个类似字典的数据结构返回，可以通过列名或者列别名在返回的数据结构中获取列值。

参考

第 2.4 节揭示了对于每种编程语言如何将数据行取出放入数据结构中使你能通过列名来访问列值。除此之外，在 `recipes` 目录的 `select` 子目录中有一些示例说明了对 `mail` 表如何做到这点。

3.5 合并多列来构建复合值

Combining Columns to Construct Composite Values

问题

用户所需要的查询结果需要组合表中多个不同的列值来获得。

解决方案

解决的方法之一就是使用 `CONCAT()` 函数。如上节所述，可以使用列别名给这个组合列一个简单的名字。

讨论

可以通过组合多列的列值来产生一个复杂的查询结果。如下表达式通过组合 `srcuser` 和 `srchost` 两列的列值得到一个邮件地址：

```
CONCAT(srcuser, '@', srchost)
```

这样的组合列，在默认情况下使用进行组合的表达式作为列名，使得默认列名较为繁琐，为此可以使用上一节中提到的方法赋予组合列一个简单的列别名。如下例，使用别名 `sender` 和 `recipient` 来重命名通过组合 `usernames` 和 `hostnames` 得到的组合列的邮件地址：

```
mysql> SELECT
-> DATE_FORMAT(t, '%M %e, %Y') AS date_sent,
-> CONCAT(srcuser, '@', srchost) AS sender,
-> CONCAT(dstuser, '@', dsthost) AS recipient,
-> size FROM mail;
+-----+-----+-----+-----+
| date_sent | sender      | recipient    | size   |
+-----+-----+-----+-----+
| May 11, 2006 | barb@saturn | tricia@mars | 58274 |
| May 12, 2006 | tricia@mars | gene@venus  | 194925 |
| May 12, 2006 | phil@mars   | phil@saturn | 1048   |
| May 13, 2006 | barb@saturn | tricia@venus| 271    |
...
...
```

3.6 Where 表达式中的列别名

WHERE Clauses and Column Aliases

问题

用户需要在 WHERE 表达式中使用列别名。

解决方案

实际上这样的查询语句并不存在，也就是说用户不能这样做。但是有其他办法来达到这一目的。

讨论

如下查询语句，由于在 WHERE 表达式中使用了列别名，导致了查询错误：

```
mysql> SELECT t, srcuser, dstuser, size/1024 AS kilobytes
-> FROM mail WHERE kilobytes > 500;
ERROR 1054 (42S22): Unknown column 'kilobytes' in 'where clause'
```

上面的查询出现错误的原因是，在 WHERE 语句中使用了列别名 `kilobytes` 来定义所要查询的行。解决的办法是用原始的列名（数据库表中定义的列名或者表达式）来代替别名，此处即用 `size/1024` 代替 `kilobytes`：

```
mysql> SELECT t, srcuser, dstuser, size/1024 AS kilobytes
-> FROM mail WHERE size/1024 > 500;
```

```
+-----+-----+-----+-----+
| t | srcuser | dstuser | kilobytes |
+-----+-----+-----+-----+
| 2006-05-14 17:03:01 | tricia | phil | 2338.3613 |
| 2006-05-15 10:25:52 | gene | tricia | 975.1289 |
+-----+-----+-----+-----+
```

3.7 调试比较表达式

Debugging Comparison Expressions

问题

用户或许想知道在 WHERE 语句中设定的查询条件如何发挥作用，或者想确保在 WHERE 语句中设置了正确的查询条件。

解决方案

一个有效而简单的方法是，将在 WHERE 语句中设定的查询条件和查询结果一起显示给用户。

讨论

为了使查询操作更有效，可以在 WHERE 语句中加上查询条件，使得查询结果中只出现符合查询条件的行：

```
mysql> SELECT * FROM mail WHERE srcuser < 'c' AND size > 5000;
+-----+-----+-----+-----+-----+
| t | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+
| 2006-05-11 10:15:08 | barb | saturn | tricia | mars | 58274 |
| 2006-05-14 14:42:21 | barb | venus | barb | venus | 98151 |
+-----+-----+-----+-----+-----+
```

但有时需要看到比较结果自身（例如，你不太确定比较是否以你所期望的方式工作时）。要达成此目的，移除 WHERE 子句，并将比较表达式放入输出列列表中，或许还包含你正比较的值：

```
mysql> SELECT srcuser, srcuser < 'c', size, size > 5000 FROM mail;
+-----+-----+-----+-----+
| srcuser | srcuser < 'c' | size | size > 5000 |
+-----+-----+-----+-----+
| barb | 1 | 58274 | 1 |
| tricia | 0 | 194925 | 1 |
| phil | 0 | 1048 | 0 |
| barb | 1 | 271 | 0 |
...
...
```

在这些结果中，1 代表 true，0 代表 false。

3.8 使查询结果唯一化

Removing Duplicate Rows

问题

相同的行在查询结果中重复出现多次，如何才能过滤冗余行，使查询结果唯一化？

解决方案

使用 DISTINCT 关键字。

讨论

在查询结果中同一行信息可能重复出现多次。例如，为了从 mail 表中查询所有的发件人，可以使用如下查询语句：

```
mysql> SELECT srcuser FROM mail;
+-----+
| srcuser |
+-----+
| barb   |
| tricia |
| phil   |
| barb   |
| gene   |
| phil   |
| barb   |
| tricia |
| gene   |
| phil   |
| gene   |
| gene   |
| phil   |
| phil   |
| gene   |
+-----+
```

这个查询结果包含太多的冗余信息，在查询语句中加入 DISTINCT 关键字，过滤冗余信息，产生一个具有唯一性的查询结果：

```
mysql> SELECT DISTINCT srcuser FROM mail;
+-----+
| srcuser |
+-----+
| barb   |
| tricia |
| phil   |
| gene   |
+-----+
```

上例中 DISTINCT 关键字作用于一列查询结果，同时它也可以作用于多列查询结果。如下例中，查询在 mail 表中出现的所有日期信息：

```
mysql> SELECT DISTINCT YEAR(t), MONTH(t), DAYOFMONTH(t) FROM mail;
+-----+-----+-----+
| YEAR(t) | MONTH(t) | DAYOFMONTH(t) |
+-----+-----+-----+
| 2006 | 5 | 11 |
| 2006 | 5 | 12 |
| 2006 | 5 | 13 |
| 2006 | 5 | 14 |
| 2006 | 5 | 15 |
| 2006 | 5 | 16 |
| 2006 | 5 | 17 |
| 2006 | 5 | 19 |
+-----+-----+-----+
```

要统计查询结果中互不相同的结果(唯一结果)的数量,可以使用关键字 COUNT(DISTINCT),例如:

```
mysql> SELECT COUNT(DISTINCT srcuser) FROM mail;
+-----+
| COUNT(DISTINCT srcuser) |
+-----+
| 4 |
+-----+
```

参考

第 8 章再次阐述了 DISTINCT 和 COUNT(DISTINCT)。第 14 章更为详细地讨论了如何移除重复行。

3.9 如何处理 NULL 值

Working with NULL Values

问题

如何将列值与 NULL 进行比较?

解决方案

选择使用合适的比较操作符: IS NULL、IS NOT NULL 或者<=>。

讨论

涉及到 NULL 的比较操作都很特殊。用户不能使用 Value=NULL 或者 Value!=NULL 来判断某一个列值等于或者不等于 NULL。如果直接以 Value=NULL 或者 Value!=NULL 作为数据库查询条件, 所得到的结果通常也为 NULL, 因为这样的比较操作并没有定义。即使使用 NULL=NULL 作为查询条件, 得到的结果仍然是 NULL (你不能判断一个未定义值是否等于另一个未定义值)。

要从数据库中查询等于或不等于 NULL 的列值, 需要使用关键字 IS NULL 或者 IS NOT NULL。假设有数据库表 taxpayer, 该表包含列 name 和 ID, id 列中的 NULL 值表示该值未定义:

```
mysql> SELECT * FROM taxpayer;
+-----+-----+
| name | id   |
+-----+-----+
| bernina | 198-48 |
| bertha | NULL  |
| ben    | NULL  |
| bill   | 475-83 |
+-----+-----+
```

使用如下的查询语句，用户并不能看到期望的查询结果：

```
mysql> SELECT * FROM taxpayer WHERE id = NULL;
Empty set (0.00 sec)
mysql> SELECT * FROM taxpayer WHERE id != NULL;
Empty set (0.01 sec)
```

要查找 id 等于或者不等于 NULL 的列值，需要使用 IS NULL 或者 IS NOT NULL 重写上面的查询语句：

```
mysql> SELECT * FROM taxpayer WHERE id IS NULL;
+-----+-----+
| name | id   |
+-----+-----+
| bertha | NULL  |
| ben    | NULL  |
+-----+-----+
mysql> SELECT * FROM taxpayer WHERE id IS NOT NULL;
+-----+-----+
| name | id   |
+-----+-----+
| bernina | 198-48 |
| bill   | 475-83 |
+-----+-----+
```

另外也可以使用<=>来比较两个 NULL 值，与=、!=不同，NULL <=> NULL 结果为 true，而不是未定义。

```
mysql> SELECT NULL = NULL, NULL <=> NULL;
+-----+-----+
| NULL = NULL | NULL <=> NULL |
+-----+-----+
|          NULL |           1 |
+-----+-----+
```

有时可以根据应用程序中上下文的需要，将数据库中的 NULL 值在应用程序中映射为各种不同的特殊定义值。例如，在 taxpayer 表中，如果 id 列中 NULL 表示“unknown（未定义）”，那么在查询语句中可以使用函数 IF()，将 id 列中的 NULL 值在查询结果中显示为 Unknown：

```
mysql> SELECT name, IF(id IS NULL, 'Unknown', id) AS 'id' FROM taxpayer;
+-----+-----+
| name | id   |
+-----+-----+
| bernina | 198-48 |
| bertha | Unknown |
| ben    | Unknown |
+-----+-----+
```

```
| bill      | 475-83   |  
+-----+-----+
```

`IF(X,X,X)` 函数可以作用于任何值类型，但是对于 `NULL`，该函数的作用尤为突出。因为 `NULL` 可以映射或者解释为很多特殊含义，例如：未知的、缺少的、还未确定的、越界，等等。你可以根据应用程序上下文给 `NULL` 赋予任何的特殊含义。

上例查询中的 `IF()` 也可以用 `IFNULL()` 来替换，使查询语句更加简练。`IFNULL()` 函数检测传给它的第一个参数是否为 `NULL`，如果不是 `NULL` 则返回该值，否则将第二个参数作为返回值：

```
mysql> SELECT name, IFNULL(id,'Unknown') AS 'id' FROM taxpayer;  
+-----+-----+  
| name    | id     |  
+-----+-----+  
| bernina | 198-48 |  
| bertha  | Unknown |  
| ben     | Unknown |  
| bill    | 475-83 |  
+-----+-----+
```

换句话说，以下两个函数定义是等价的：

```
IF(expr1 IS NOT NULL,expr1,expr2)  
IFNULL(expr1,expr2)
```

相比而言，从可读性来说 `IF()` 比 `IFNULL()` 更具有可读性。但是从运行效率来看，`IFNULL()` 则略胜一筹。因为在 `IF()` 中，`expr1` 需要计算两次，而在 `IFNULL()` 中则只需要计算一次。

参考

在排序和统计操作中 `NULL` 值也表现特殊。详见第 7.14 节和第 8.8 节。

3.10 在用户程序中使用 `NULL` 作为比较参数

Writing Comparisons Involving `NULL` in Programs

问题

你正在编写寻找包含特定值的数据行的程序，但是当值为 `NULL` 时程序出错了。

解决方案

根据查询条件中所比较的值等于或者不等于 `NULL`，来选择合适的操作符。

讨论

当 NULL 出现在用户程序定义的数据库查询条件下，在选择合适的比较操作符时，会给用户程序带来某些潜在的危险。如果某个用户程序变量中保存的值是 NULL，那么在数据库查询中使用该变量作为查询条件时，需要选择正确的比较操作符，以保证程序的正确性。例如，在 Perl 中，保留字 `undef` 代表 NULL 值，如果需要从 `taxpayer` 表中查询某一行，其中 `$id` 列的值等于某一特定值，如下定义的查询语句可能带来错误的结果：

```
$sth = $dbh->prepare ("SELECT * FROM taxpayer WHERE id = ?");  
$sth->execute ($id);
```

当 `$id` 所保存的值为 `undef` 时语句出错，因为所定义的查询语句实际是：

```
SELECT * FROM taxpayer WHERE id = NULL
```

比较操作 `id=NULL` 的返回结果永远不可能为 `true`，所以上例使用的查询语句不会返回任何结果。考虑到 `$id` 的值可能为 `undef`，用户需要使用正确的比较操作符来定义查询条件：

```
$operator = (defined ($id) ? "=" : "IS");  
$sth = $dbh->prepare ("SELECT * FROM taxpayer WHERE id $operator ?");  
$sth->execute ($id);
```

当 `$id` 的值分别为 `undef(NULL)` 和 `43(not NULL)` 时，上例中的查询语句实际为：

```
SELECT * FROM taxpayer WHERE id IS NULL  
SELECT * FROM taxpayer WHERE id = 43
```

对于 `!=NULL`，可以将 `$operator` 定义修改如下：

```
$operator = (defined ($id) ? "!=" : "IS NOT");
```

也可以从另外的角度来避免这个问题——在定义数据库时，如果不是特别的需要，就不允许列值为 NULL。例如，在 `taxpayer` 表中，可以使用如下定义禁止使用 NULL 作为列值：

```
# taxpayer2.sql  
  
# taxpayer 表，使用 NOT NULL 列定义  
  
DROP TABLE IF EXISTS taxpayer;  
CREATE TABLE taxpayer  
{  
    name  CHAR(20) NOT NULL,  
    id    CHAR(20) NOT NULL  
};
```

3.11 结果集排序

Sorting a Result Set

问题

如何按照用户期望的顺序输出查询结果。

解决方案

MySQL 并不能了解你的想法。在查询语句中使用 ORDER BY 语句来定义查询结果的排序依据，使查询结果按照用户期望的顺序输出。

讨论

当进行数据库查询时，除非在查询语句中指定了查询结果输出顺序，否则 MySQL 服务器采用随机顺序输出查询结果。有许多使用排序技术的方法。在第 7 章中详细探讨了这个主题。最简单的，就是在查询语句中使用一个 ORDER BY 子句对查询结果进行排序。可以在 ORDER BY 子句中指定数据库表的一列或多列作为排序依据，例如下面的查询语句使用列 size 作为排序依据：

```
mysql> SELECT * FROM mail WHERE size > 100000 ORDER BY size;
+-----+-----+-----+-----+-----+
| t   | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+
| 2006-05-12 12:48:13 | tricia | mars    | gene    | venus   | 194925 |
| 2006-05-15 10:25:52 | gene   | mars    | tricia  | saturn  | 998532 |
| 2006-05-14 17:03:01 | tricia | saturn  | phil    | venus   | 2394482 |
+-----+-----+-----+-----+-----+
```

再给出一个例子，以几个不同的列作为查询结果的排序依据，查询结果首先根据 host 列排序，然后再在上一次排序的结果上根据 user 列排序：

```
mysql> SELECT * FROM mail WHERE dstuser = 'tricia'
      -> ORDER BY srchost, srcuser;
+-----+-----+-----+-----+-----+
| t   | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+
| 2006-05-15 10:25:52 | gene   | mars    | tricia  | saturn  | 998532 |
| 2006-05-14 11:52:17 | phil   | mars    | tricia  | saturn  | 5781  |
| 2006-05-17 12:49:23 | phil   | mars    | tricia  | saturn  | 873   |
| 2006-05-11 10:15:08 | barb   | saturn  | tricia  | mars    | 58274 |
| 2006-05-13 13:59:18 | barb   | saturn  | tricia  | venus   | 271   |
+-----+-----+-----+-----+-----+
```

在 ORDER BY 子句的最后，即定义的排序依据的列名之后加上关键字 DESC 使得查询结果按逆序排序：

```
mysql> SELECT * FROM mail WHERE size > 50000 ORDER BY size DESC;
+-----+-----+-----+-----+-----+
| t   | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+
| 2006-05-14 17:03:01 | tricia | saturn  | phil    | venus   | 2394482 |
| 2006-05-15 10:25:52 | gene   | mars    | tricia  | saturn  | 998532 |
| 2006-05-12 12:48:13 | tricia | mars    | gene    | venus   | 194925 |
| 2006-05-14 14:42:21 | barb   | venus   | barb   | venus   | 98151  |
| 2006-05-11 10:15:08 | barb   | saturn  | tricia  | mars    | 58274  |
+-----+-----+-----+-----+-----+
```

3.12 使用视图来简化查询

Using Views to Simplify Table Access

问题

在有的查询中，可能需要使用一个运算表达式才能得到期望的结果。当这样的查询频繁出现时，是否有方法简化这样的查询语句，而不需要每一次进行查询时都使用复杂的运算表达式？

解决方案

使用视图，并根据在查询语句中使用的运算表达式来定义视图中的列。

讨论

在第 3.5 节中，我们从数据库表 mail 中查询到了几列值，其中多半都是通过使用表达式计算得到的：

```
mysql> SELECT
    -> DATE_FORMAT(t, '%M %e, %Y') AS date_sent,
    -> CONCAT(srcuser, '@', srchost) AS sender,
    -> CONCAT(dstuser, '@', dsthost) AS recipient,
    -> size FROM mail;
+-----+-----+-----+-----+
| date_sent | sender      | recipient    | size   |
+-----+-----+-----+-----+
| May 11, 2006 | barb@saturn | tricia@mars | 58274 |
| May 12, 2006 | tricia@mars | gene@venus   | 194925 |
| May 12, 2006 | phil@mars   | phil@saturn | 1048  |
| May 13, 2006 | barb@saturn | tricia@venus | 271   |
...
...
```

当需要频繁使用类似上例的查询时，就需要每一次都在查询语句中重复写入类似的复杂的运算表达式。在这样的情况下，就可以使用视图来简化查询语句。视图是一个虚拟的数据表，它并不包含任何实际的数据。实际上，一个视图可以看作是一个特定的 SELECT 语句。下面的视图 mail_view 与定义它的 SELECT 语句是等价的：

```
mysql> CREATE VIEW mail_view AS
    -> SELECT
    -> DATE_FORMAT(t, '%M %e, %Y') AS date_sent,
    -> CONCAT(srcuser, '@', srchost) AS sender,
    -> CONCAT(dstuser, '@', dsthost) AS recipient,
    -> size FROM mail;
```

查询视图与查询其他的普通数据库表是一样的。例如，用户可以查询视图中的一列或多列信息，可以使用 WHERE 子句查询特定行，使用 ORDER BY 子句对查询结果进行排序，等等：

```
mysql> SELECT date_sent, sender, size FROM mail_view
    -> WHERE size > 100000 ORDER BY size;
+-----+-----+-----+
| date_sent | sender      | size   |
+-----+-----+-----+
```

```
| May 12, 2006 | tricia@mars    | 194925 |
| May 15, 2006 | gene@mars     | 998532 |
| May 14, 2006 | tricia@saturn | 2394482 |
+-----+-----+-----+
```

3.13 多表查询

Selecting Data from More Than One Table

问题

你需要从多个表中查询数据。

解决方案

使用联合 (join) 操作，或者使用子查询。

讨论

到目前为止，我们的讨论都仅限于单表查询，但是在实际应用中，用户有时候需要同时从多个数据库表中查询信息，进行多表查询。有两种方法可实现多表查询：使用联合 (join) 操作或者子查询。join 将一个表中的行和另一个表的行进行匹配，并使你能查询出包含一个或两个表的表列的输出行。子查询是嵌套在另外一个查询过程中的查询，其查询结果通常作为包含该子查询的查询的条件。

本节将通过几个具体的例子来说明联合操作和子查询的使用。别处还有其他的示例：子查询在贯穿全书的不同示例中都有使用（例如，第 3.16 和 3.17 节）。第 12 章更为详细的探讨了联合，包括从多于 2 个表中进行查询。

下面的例子使用在第 2 章中介绍过的表 profile，其中记录了某人的联系人信息。现在引入另外一张表 profile_contact，其中包含了 profile 中记录的联系人所使用的即时消息系统类型及账号，其定义如下：

```
CREATE TABLE profile_contact
(
    profile_id    INT UNSIGNED NOT NULL, # 从 profile 表而来的 ID
    service       CHAR(20) NOT NULL,        # 信息服务名称
    contact_name CHAR(25) NOT NULL,        # 联系人姓名
    INDEX (profile_id)
);
```

在两个表中都有 profile_id 列，通过该列可以将两个表中具有相同的 profile_id 值的两行关联起来。profile_contact 表中的 service 和 contact_name 列分别记录了联系人所使用的即时消息服务类型以及账号。假设在 profile_contact 中包含以下信息：

```
mysql> SELECT * FROM profile_contact ORDER BY profile_id, service;
+-----+-----+-----+
| profile_id | service | contact_name |
+-----+-----+-----+
| 1 | AIM | user1-aimid |
| 1 | MSN | user1-msnid |
| 2 | AIM | user2-aimid |
| 2 | MSN | user2-msnid |
| 2 | Yahoo | user2-yahoooid |
| 4 | Yahoo | user4-yahoooid |
+-----+-----+-----+
```

对于这两张表，有时候会遇到这样的查询要求：查询 profile 表中记录的每一个联系人的名字、所使用的即时消息服务类型以及相应的账号。我们可以使用联合操作来实现这样的查询。使用联合操作从两张表中同时进行查找，并将 profile_contact 表中 profile_id 值和 profile 表中 id 值相等的行关联起来，作为一行查询结果输出：

```
mysql> SELECT id, name, service, contact_name
-> FROM profile INNER JOIN profile_contact ON id = profile_id;
+-----+-----+-----+
| id | name | service | contact_name |
+-----+-----+-----+
| 1 | Fred | AIM | user1-aimid |
| 1 | Fred | MSN | user1-msnid |
| 2 | Mort | AIM | user2-aimid |
| 2 | Mort | MSN | user2-msnid |
| 2 | Mort | Yahoo | user2-yahoooid |
| 4 | Carl | Yahoo | user4-yahoooid |
+-----+-----+-----+
```

上面的查询语句中，From 子句指定了所要查询的表（这里是 profile 和 profile_contact），ON 子句指定了两个表的行的绑定规则。在显示的查询结果中 id 列和 name 列来自 profile 表，而 service 和 contact_name 列来自 profile_contact 表，它们按照 id=profile_id 的规则绑定到一行。

现在涉及到两张表的查询还有很多，这里再举一例。例如这样的查询需求：列出 profile_contact 表中关于 Mort 的所有信息。要从 profile_contact 表中查询 Mort 的相关信息，首先就要知道对应的在 profile 中的 Mort 的 id 信息。可以使用子查询通过 name 列直接查询需要的结果，而不需要用户自己查询对应的 profile 表中的 id 值。查询语句如下：

```
mysql> SELECT * FROM profile_contact
-> WHERE profile_id = (SELECT id FROM profile WHERE name = 'Mort');
+-----+-----+-----+
| profile_id | service | contact_name |
+-----+-----+-----+
| 2 | AIM | user2-aimid |
| 2 | MSN | user2-msnid |
| 2 | Yahoo | user2-yahoooid |
+-----+-----+-----+
```

在上例的查询语句中，在括号中的 SELECT 语句就是一个子查询。

3.14 从查询结果集头或尾取出部分行

Selecting Rows from the Beginning or End of a Result Set

问题

怎样才能只显示查询结果的部分行，例如第一行或者最后 5 行？

解决方案

使用 `LIMIT` 子句，并且经常要跟 `ORDER BY` 子句配合使用。

讨论

MySQL 支持用户使用 `LIMIT` 子句通知服务器只返回查询结果的部分行。`LIMIT` 子句是 MySQL 对 SQL 语句的一个有用的扩展，当用户只关心一次查询结果中的部分行时，`LIMIT` 子句显得尤为有用。此时用户可以通过 `LIMIT` 子句使查询结果只显示查询结果最开始的几行或者是其中的任意部分行。`LIMIT` 子句主要用于以下类型的查询：

- 当需要查询的结果是：第一个或者最后一个，最大或者最小，最新或者最旧，最便宜或者最贵，等等；
- 当查询结果行数较多时，将查询结果划分为小块，每次只取出其中一块返回显示给用户。这项技术在 Web 程序中较为常见：将一个跨越多个页面的查询结果划分为多次显示，每次显示一个页面。每次显示查询结果的一部分，可以使查询结果更具有可读性。在 3.15 节中对此有详细的说明。

下面的例子使用了第 2 章中介绍的 `profile` 表，其具体内容如下：

```
mysql> SELECT * FROM profile;
+----+-----+-----+-----+-----+
| id | name | birth | color | foods           | cats |
+----+-----+-----+-----+-----+
| 1  | Fred | 1970-04-13 | black | lutefisk,fadge,pizza | 0   |
| 2  | Mort | 1969-09-30 | white | burrito,curry,eggroll | 3   |
| 3  | Brit | 1957-12-01 | red   | burrito,curry,pizza  | 1   |
| 4  | Carl | 1973-11-02 | red   | eggroll,pizza        | 4   |
| 5  | Sean | 1963-07-04 | blue  | burrito,curry        | 5   |
| 6  | Alan | 1965-02-14 | red   | curry,fadge          | 1   |
| 7  | Mara | 1968-09-17 | green | lutefisk,fadge       | 1   |
| 8  | Shepard | 1975-09-02 | black | curry,pizza          | 2   |
| 9  | Dick | 1952-08-20 | green | lutefisk,fadge       | 0   |
| 10 | Tony | 1960-05-01 | white | burrito,pizza        | 0   |
+----+-----+-----+-----+-----+
```

在 `SELECT` 查询语句的最后加上 `LIMIT n`，可以获得查询结果的头 `n` 行：

```
mysql> SELECT * FROM profile LIMIT 1;
+----+-----+-----+-----+-----+
| id | name | birth | color | foods           | cats |
+----+-----+-----+-----+-----+
```

```

+---+-----+-----+-----+-----+
| 1 | Fred | 1970-04-13 | black | lutefisk,fadge,pizza | 0 |
+---+-----+-----+-----+-----+
mysql> SELECT * FROM profile LIMIT 5;
+---+-----+-----+-----+-----+
| id | name | birth      | color | foods           | cats |
+---+-----+-----+-----+-----+
| 1 | Fred | 1970-04-13 | black | lutefisk,fadge,pizza | 0 |
| 2 | Mort | 1969-09-30 | white | burrito,curry,eggroll | 3 |
| 3 | Brit | 1957-12-01 | red   | burrito,curry,pizza | 1 |
| 4 | Carl | 1973-11-02 | red   | eggroll,pizza       | 4 |
| 5 | Sean | 1963-07-04 | blue  | burrito,curry       | 5 |
+---+-----+-----+-----+-----+

```

需要注意的是，`LIMIT n` 的含义是“最多返回 n 行”，如果用户指定了 `LIMIT 10`，但是查询结果中实际上只有 3 行，那么服务器将给用户返回 3 行查询结果。

上例的查询结果是无序的，所以从查询结果中取出头 n 行可能也是无意义的。更常见的做法是首先使用 `ORDER BY` 子句对查询结果进行排序，然后再使用 `LIMIT` 子句从查询结果中取出具有最大或者最小特征的行。例如，为了找到出生最早的人，首先根据 `birth` 列对查询结果排序，然后使用 `LIMIT 1` 从查询结果中取出第一行：

```

mysql> SELECT * FROM profile ORDER BY birth LIMIT 1;
+---+-----+-----+-----+-----+
| id | name | birth      | color | foods           | cats |
+---+-----+-----+-----+-----+
| 9 | Dick | 1952-08-20 | green | lutefisk,fadge | 0 |
+---+-----+-----+-----+-----+

```

如果要获取查询结果的最后 n 行，则可以对查询结果进行逆序排序。例如，需要查找最晚出生的人，查询语句类似上例，不同的是对查询结果按照 `birth` 列值递减进行排序：

```

mysql> SELECT * FROM profile ORDER BY birth DESC LIMIT 1;
+---+-----+-----+-----+-----+
| id | name      | birth      | color | foods           | cats |
+---+-----+-----+-----+-----+
| 8 | Shepard | 1975-09-02 | black | curry,pizza | 2 |
+---+-----+-----+-----+-----+

```

要找出日历年最早或最迟的出生日，根据 `birth` 值的月份和日期进行排序：则可以将查询结果先根据生日的月和日进行排序：

```

mysql> SELECT name, DATE_FORMAT(birth, '%m-%d') AS birthday
    -> FROM profile ORDER BY birthday LIMIT 1;
+-----+
| name | birthday |
+-----+
| Alan | 02-14   |
+-----+

```

如果上例查询中不使用 LIMIT 子句，而是人为的忽略除了第一行以外的查询结果，用户也一样可以得到需要的信息。使用 LIMIT 子句的优势在于，数据库服务器只通过网络向用户返回查询结果中的第一行，而不会返回其他行。这样就避免了网络带宽的浪费，同时提高了用户程序的运行效率。

参考

当在查询中使用 `LIMIT n` 来查询最大或者最小的 `n` 个值时，查询结果可能和用户期望有些出入。3.16 节中讨论了针对不同目的如何正确使用 LIMIT 子句。

LIMIT 子句可以和 `RAND()` 函数合用，从数据库中随机查询并返回相关数据，在第 13 章中做了详细说明。

LIMIT 子句也可以应用于 `DELETE` 和 `UPDATE` 语句，对所要删除或者修改的行进行限制。这种情况下，LIMIT 子句和 WHERE 子句组合使用可以产生更好的效果。例如：在一个数据库表中有 5 行一样的数据，其中 4 行是冗余的。此时可以在 `DELETE` 语句中使用一个 WHERE 子句找到这 5 行，然后通过在语句的最后加上 `LIMIT 4` 删除冗余的 4 行。这样可以保持数据库信息的唯一性，避免冗余。在 14.4 节中读者可以看到更多关于使用 LIMIT 删冗余行的讨论。

3.15 在结果集中间选取部分行

Selecting Rows from the Middle of a Result Set

问题

如何从查询结果的中间获取部分行，而不是头 `n` 行或者尾 `n` 行？例如获取查询结果的第 21 到第 40 行。

解决方案

同样可以使用 LIMIT 子句来解决这个问题。需要在 LIMIT 子句中指定从查询结果的第几行开始，一共返回多少行。

讨论

`LIMIT n` 指定服务器返回查询结果的头 `n` 行。同时 LIMIT 子句也可以接收两个参数，使用户可以获取一个查询结果中的任意部分行。这种情况下，LIMIT 的两个参数分别指定了从查询结果的第几行开始返回，以及一共返回多少行。这就意味着用户可以跳过查询结果的前两行，而直接从第三行开始返回查询结果。使用函数 `MAX()` 或 `MIN()` 可以轻松地查询最大或者最小的值，但并不能查询第三大或者第三长的值，这一功能现在可以使用 LIMIT 子句来实现。

```
mysql> SELECT * FROM profile ORDER BY birth LIMIT 2,1;
+-----+-----+-----+-----+-----+
```

```
| id | name | birth      | color | foods          | cats |
+---+----+-----+-----+-----+-----+
| 10 | Tony | 1960-05-01 | white | burrito,pizza |   0  |
+---+----+-----+-----+-----+-----+
mysql> SELECT * FROM profile ORDER BY birth DESC LIMIT 2,1;
+---+----+-----+-----+-----+-----+
| id | name | birth      | color | foods          | cats |
+---+----+-----+-----+-----+-----+
|  1 | Fred | 1970-04-13 | black | lutefisk,fadge,pizza |   0  |
+---+----+-----+-----+-----+-----+
```

使用带有两个参数的 LIMIT 子句可以将查询结果划分为多个小的结果 (sections)。例如，每次查询只从查询结果中返回 20 行记录，每一次查询都使用不同的 LIMIT 参数就可以分多次返回整个表的信息：

```
查询首 20 行
SELECT ... FROM ... ORDER BY ... LIMIT 0, 20;
跳过 20 行, 查询下一个 20 行
SELECT ... FROM ... ORDER BY ... LIMIT 20, 20;
跳过 40 行, 查询下一个 20 行
SELECT ... FROM ... ORDER BY ... LIMIT 40, 20;
...
...
```

Web 程序开发者经常使用这样的技巧将一个查询结果划分为多个更小、更容易管理的结果，并使用多个 Web 页面来显示查询结果。在第 19.10 节中将进一步讨论这个技巧的应用。

为了决定将整个查询结果分多少次返回，首先可以通过 COUNT 函数计算查询结果中一共有多少行。例如，需要查询 profile 表，将结果根据 name 列排序，并且每次只显示 4 行结果，首先可以通过如下语句计算查询结果中一共有多少行：

```
mysql> SELECT COUNT(*) FROM profile;
+-----+
| COUNT(*) |
+-----+
|      10 |
+-----+
```

根据计算结果可知，可以分 3 次返回查询结果（虽然最后一个少于 4 行），可以使用如下语句进行具体查询：

```
SELECT * FROM profile ORDER BY name LIMIT 0, 4;
SELECT * FROM profile ORDER BY name LIMIT 4, 4;
SELECT * FROM profile ORDER BY name LIMIT 8, 4;
```

也可以在一个语句中计算查询结果中的总行数，同时按照划分规则返回部分行。例如，查询 profile 表并根据 name 列排序，返回查询结果的头 4 行，并同时计算整个查询结果的总行数：

```
SELECT SQL_CALC_FOUND_ROWS * FROM profile ORDER BY name LIMIT 4;
SELECT FOUND_ROWS();
```

在第一个 SQL 语句中，关键字 SQL_CALC_FOUND_ROWS 指定服务器计算整个查询结果的行数，而其后的 LIMIT 4 则指定最终只向用户返回查询结果中的头 4 行。第一个语句的计算结果可以在下面的 SQL 语句中使用 FOUND_ROWS() 函数获得。如果得到的结果大于 4，则说明在查询结果中还有其他行未返回给用户。

3.16 选择合适的 LIMIT 参数

Choosing Appropriate LIMIT Values

问题

LIMIT 看起来并未做你想要它做的事。

解决方案

确定你理解了你正问的是什么问题。可能 LIMIT 暴露了你从未考虑过的数据中的一些有趣的微妙之处。

讨论

LIMIT *n* 子句和 ORDER BY 子句一起，可以很方便从查询结果中得到含有某个最大或者最小值的 *n* 行记录。但并不能保证每一次这样的查询结果都是正确的，例如当所要查询的行中有重复或者相等的值时，就不能保证查询结果的正确性。这种情况下需要一次预查询来帮助确定应该使用的 LIMIT 参数值。

我们通过下面的例子说明为什么需要使用预查询。下例查询 2001 年赛季中胜利次数大于 15 次的美国大联盟投手(读者可以在 recipes 发行包的 tables 目录中的 al_winner.sql 文件中找到相关信息)：

```
mysql> SELECT name, wins FROM al_winner
-> ORDER BY wins DESC, name;
+-----+-----+
| name      | wins |
+-----+-----+
| Mulder, Mark | 21 |
| Clemens, Roger | 20 |
| Moyer, Jamie | 20 |
| Garcia, Freddy | 18 |
| Hudson, Tim | 18 |
| Abbott, Paul | 17 |
| Mays, Joe | 17 |
| Mussina, Mike | 17 |
| Sabathia, C.C. | 17 |
| Zito, Barry | 17 |
| Buehrle, Mark | 16 |
| Milton, Eric | 15 |
| Pettitte, Andy | 15 |
| Radke, Brad | 15 |
| Sele, Aaron | 15 |
+-----+-----+
```

在上例中查询语句最后加上 LIMIT 1 即可查询得到胜利次数最多的队。这里表中 wins 列的最大值 21 是唯一的，通过查询可以得到正确的结果，但是如果现在使用 LIMIT 4 来查询获胜次数最多的 4 支队，那么得到一个不确定的查询结果：

- 如果你只是想得到查询结果的头 4 行，那么只需要对查询结果排序之后，在查询语句结尾加上 LIMIT 4 即可：

```
mysql> SELECT name, wins FROM al_winner
      -> ORDER BY wins DESC, name
      -> LIMIT 4;
+-----+-----+
| name | wins |
+-----+-----+
| Mulder, Mark | 21 |
| Clemens, Roger | 20 |
| Moyer, Jamie | 20 |
| Garcia, Freddy | 18 |
+-----+-----+
```

但是这个查询结果并不完全符合使用者的意图，因为 LIMIT 操作将具有相同 wins 列值（这里为 18）的两行数据分割开，并且只返回其中一行。

- 为了避免类似这样将具有相同值的多行分割开，并且只返回其中部分行的情况发生，可以使用一个大于或等于第 4 行中 wins 的值作为查询条件。分两步来完成，首先查询第 4 行的 wins 值，然后在第二个查询中使用一个大于或等于第 4 行 wins 的值作为查询条件：

```
mysql> SELECT wins FROM al_winner
      -> ORDER BY wins DESC, name
      -> LIMIT 3, 1;
+-----+
| wins |
+-----+
| 18 |
+-----+
mysql> SELECT name, wins FROM al_winner
      -> WHERE wins >= 18
      -> ORDER BY wins DESC, name;
+-----+-----+
| name | wins |
+-----+-----+
| Mulder, Mark | 21 |
| Clemens, Roger | 20 |
| Moyer, Jamie | 20 |
| Garcia, Freddy | 18 |
| Hudson, Tim | 18 |
+-----+-----+
```

为了在一次查询操作中完成上例中两次查询的过程，并避免手工查询引起相同值分割的 wins 值，可以将第一次查询作为第二次查询的子查询：

```

mysql> SELECT name, wins FROM al_winner
      -> WHERE wins >=
      ->   (SELECT wins FROM al_winner
      ->     ORDER BY wins DESC, name
      ->     LIMIT 3, 1)
      ->   ORDER BY wins DESC, name;
+-----+-----+
| name | wins |
+-----+-----+
| Mulder, Mark | 21 |
| Clemens, Roger | 20 |
| Moyer, Jamie | 20 |
| Garcia, Freddy | 18 |
| Hudson, Tim | 18 |
+-----+-----+

```

- 如果查询的目的是得到具有头 4 个最大获胜次数的队，则使用的又是另外的查询语句了。首先使用 DISTINCT 和 LIMIT 查询到第 4 大的获胜次数，也即第 4 大的 wins 值，然后使用该值作为第二个查询的 WHERE 字句条件：

```

mysql> SELECT DISTINCT wins FROM al_winner
      -> ORDER BY wins DESC, name
      -> LIMIT 3, 1;
+-----+
| wins |
+-----+
| 17 |
+-----+
mysql> SELECT name, wins FROM al_winner
      -> WHERE wins >= 17
      -> ORDER BY wins DESC, name;
+-----+-----+
| name | wins |
+-----+-----+
| Mulder, Mark | 21 |
| Clemens, Roger | 20 |
| Moyer, Jamie | 20 |
| Garcia, Freddy | 18 |
| Hudson, Tim | 18 |
| Abbott, Paul | 17 |
| Mays, Joe | 17 |
| Mussina, Mike | 17 |
| Sabathia, C.C. | 17 |
| Zito, Barry | 17 |
+-----+-----+

```

像前面的例子一样，这里的两个独立的查询，也可以使用子查询合并为一个：

```

mysql> SELECT name, wins FROM al_winner
      -> WHERE wins >=
      ->   (SELECT DISTINCT wins FROM al_winner
      ->     ORDER BY wins DESC, name
      ->     LIMIT 3, 1)
      ->   ORDER BY wins DESC, name;
+-----+-----+

```

name	wins
Mulder, Mark	21
Clemens, Roger	20
Moyer, Jamie	20
Garcia, Freddy	18
Hudson, Tim	18
Abbott, Paul	17
Mays, Joe	17
Mussina, Mike	17
Sabathia, C.C.	17
Zito, Barry	17

在本节的例子中，针对相同的数据，使用不同的查询方法，就得到不同的结果。因此为了得到期望的查询结果，需要使用能够正确表达使用者意图的 `LIMIT` 子句。

3.17 当 `LIMIT` 需要“错误”的排列顺序时做什么

What to Do When `LIMIT` Requires the Wrong Sort Order

问题

`LIMIT` 和用于排序的 `ORDER BY` 配合使用，通常都能在查询中发挥很好的作用。但是有时候使用 `ORDER BY` 子句得到的排序结果和你想要的结果正好相反。

解决方案

在子查询中使用 `LIMIT` 子句取得所需要的行，然后在外层查询中对子查询结果进行排序。

讨论

如果希望取得一个查询结果的最后 4 行记录，只需要将查询结果按照降序（逆序）排列，然后使用 `LIMIT 4` 子句即可。如下查询语句，从 `profile` 表中查询最晚出生的 4 个人，并返回姓名和生日信息：

```
mysql> SELECT name, birth FROM profile ORDER BY birth DESC LIMIT 4;
+-----+-----+
| name | birth |
+-----+-----+
| Shepard | 1975-09-02 |
| Carl | 1973-11-02 |
| Fred | 1970-04-13 |
| Mort | 1969-09-30 |
+-----+-----+
```

但是这样的查询方法，要求首先对查询结果进行降序排列，最后的查询结果也按照降序返

回给用户。如果要求查询结果按照升序排列又该如何处理呢？解决方法之一是使用两个查询语句。首先使用 COUNT() 函数计算表中一共有几行记录。

```
mysql> SELECT COUNT(*) FROM profile;
+-----+
| COUNT(*) |
+-----+
|      10 |
+-----+
```

然后，使用升序来排列查询结果，并使用带有两个参数的 LIMIT 子句从排序结果中取出所需行。

```
mysql> SELECT name, birth FROM profile ORDER BY birth LIMIT 6, 4;
+-----+-----+
| nam | birth   |
+-----+-----+
| Mort | 1969-09-30 |
| Fred | 1970-04-13 |
| Carl | 1973-11-02 |
| Shepard | 1975-09-02 |
+-----+-----+
```

这样的解决方法要求计算表中的总行数，然后还需要计算从第几行开始返回查询结果，用户也许并不想这么做。我们还有第二种解决方案：首先在一个子查询中使用 LIMIT 子句取得所需要的行，然后在外层查询中对该结果排序，并返回给用户。如下：

```
mysql> SELECT * FROM
-> (SELECT name, birth FROM profile ORDER BY birth DESC LIMIT 4) AS t
-> ORDER BY birth;
+-----+-----+
| name | birth   |
+-----+-----+
| Mort | 1969-09-30 |
| Fred | 1970-04-13 |
| Carl | 1973-11-02 |
| Shepard | 1975-09-02 |
+-----+-----+
```

在查询语句中，FROM 子句之后必须是表名，因此在上例中使用 AS t 对子查询结果赋予别名，作为 FROM 的一个临时表名。

3.18 从表达式中计算 LIMIT 值

Calculating LIMIT Values from Expressions

问题

如何使用表达式计算 LIMIT 子句所使用的参数？

解决方案

实际上，在 sql 语句中，LIMIT 子句只能直接以数字作为参数，而并不能使用表达式作为 LIMIT 子句的参数；除非是在应用程序中，使用用户程序语句计算出 LIMIT 参数，然后再将计算结果插入到 sql 语句中。

讨论

sql 语句中只允许数字作为 LIMIT 子句的参数，如下 sql 语句都是错误的：

```
SELECT * FROM profile LIMIT 5+5;  
SELECT * FROM profile LIMIT @skip_count, @show_count;
```

在 sql 中如果使用表达式作为 LIMIT 子句的参数，会造成同样的错误。因此在应用程序中，应该首先使用程序语句计算出所需参数值，然后将该值插入到 sql 语句中。如下例，在 Perl 或者 PHP 程序中，使用类似方式书写的 sql 语句，将造成运行时错误：

```
$str = "SELECT * FROM profile LIMIT $x + $y";
```

先计算出最终使用的 LIMIT 子句参数可以避免错误发生：

```
$z = $x + $y;  
$str = "SELECT * FROM profile LIMIT $z";
```

或者通过用户程序构造一个 sql 语句（不要忘了把使用的表达式放到括号里）：

```
$str = "SELECT * FROM profile LIMIT " . ($x + $y);
```

如果需要使用带有两个参数的 LIMIT 子句，需要分别计算好两个参数，然后再将其插入到 sql 语句中。

表管理

Table Management

4.0 引言

Introduction

本章的内容主要包括如何创建以及迁移表：

- 克隆表；
- 表间拷贝；
- 使用临时表；
- 生成唯一表名；
- 确定一张表使用什么存储引擎，或者把一张表转换到其他存储引擎。

要创建并装载本章示例中所使用的 `mail` 表，在控制台下进入 `recipes` 发行包的 `table` 目录，运行下面的命令：

```
% mysql cookbook < mail.sql
```

4.1 克隆表

Cloning a Table

问题

你需要创建一个恰好与某个已有表结构一致的表。

解决方案

使用 `CREATE TABLE ... LIKE` 语句来克隆表结构；使用 `INSERT INTO SELECT` 语句克隆部分或者全部表数据。

讨论

如果需要新建一张和现有的某张表结构一致的表，可以使用如下语句进行克隆：

```
CREATE TABLE new_table LIKE original_table;
```

这样得到的新表和原有的表除了以下几点之外，在结构属性上一模一样：CREATE TABLE ... LIKE 语句不克隆源表的外键定义，不克隆源表可能使用的 DATA DIRECTORY 和 INDEX DIRECTORY 表选项。

如上方法生成的新表不包含任何数据。如果需要同时拷贝源表的数据，可以使用 INSERT INTO ... SELECT 语句：

```
INSERT INTO new_table SELECT * FROM original_table;
```

如果只想复制源表的部分数据到新表，可以在完成复制的 sql 语句中加上相应的 WHERE 子句来指定需要复制的行。例如，下面的语句根据 mail 表克隆了 mail2 表，并将 mail 表中发送人为 barb 的数据行复制到 mail2 中：

```
mysql> CREATE TABLE mail2 LIKE mail;
mysql> INSERT INTO mail2 SELECT * FROM mail WHERE srcuser = 'barb';
mysql> SELECT * FROM mail2;
+-----+-----+-----+-----+-----+
| t   | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+
| 2006-05-11 10:15:08 | barb    | saturn  | tricia | mars    | 58274 |
| 2006-05-13 13:59:18 | barb    | Saturn  | tricia | venus   | 271  |
| 2006-05-14 14:42:21 | barb    | venus   | barb   | venus   | 98151 |
+-----+-----+-----+-----+-----+
```

关于 INSERT ... SELECT 语句的更多详细介绍，请参考第 4.2 节。

4.2 将查询结果保存到表中

Saving a Query Result in a Table

问题

如何将 SELECT 语句查询的结果保存到一张表中，而不是直接返回并显示给用户？

解决方案

如果用来保存查询结果的表已经存在，可以使用 INSERT INTO ... SELECT 语句将查询结果插入表中。如果所要使用的表还不存在，需要使用 CREATE TABLE ... SELECT 语句根据查询结果新建一张表。

讨论

MYSQL 通常将 SELECT 查询结果直接返回给用户。例如，当你在 mysql 客户端上执行了一条 sql 语句，数据库服务器会将处理结果返回 mysql 客户端。紧接着，客户端将结果显示在屏幕上。也可以选择将 SELECT 语句的查询结果保存到表中，这在有些情况下也是非常有用的，例如：

- 用来复制一张表的全部或者部分数据。例如，在编写数据库相关的应用程序时，如果用户程序需要处理某张表，出于安全起见，那么可以复制该表用于开发和调试，这样

就不用担心程序开发过程中错误地修改了表。当所要处理的表数据量很大时，在开发过程中所用的临时表只需要复制实际使用表的部分数据，这样可以减少数据库交互所用的时间，从而提高开发速度。

- 如果数据装载操作基于可能是错误的数据，可以将新的数据行暂存到临时表中，然后对数据进行检查，并在必要的时候修正错误的数据。直到对所装载的数据满意之后，再将改行由临时表拷贝到主表中。
- 有的应用程序同时维护着两个表，一个很大的存储表用于保存数据，和一个较小的工作表。应用程序频繁地将数据插入到工作表中，并定时地将数据从工作表复制到存储表，同时清空工作表。
- 如果你打算在同一张表上重复多次数据摘要操作，首先查询一次摘要信息并保存到另外一张临时表中，以后用这张临时表进行数据分析，这比每一次都在原始的数据表上重复进行开销很大的摘要操作要更高效。

本节将讨论如何将查询结果保存到一张表中。在例子中使用 `src_tb1` 来指代源表，从中查询数据；使用 `dst_tb1` 指代目的表，将查询结果保存到其中。

如果用于保存查询结果的目的表已经存在，可以使用 `INSERT ... SELECT` 语句将查询结果复制到其中。假设 `dst_tb1` 中含有一个整数列 `i` 和一个字符串列 `s`，下面的查询语句将 `src_tb1` 表中 `val` 和 `name` 两列的所有行分别复制到 `dst_tb1` 的 `i` 和 `s` 两列中：

```
INSERT INTO dst_tb1 (i, s) SELECT val, name FROM src_tb1;
```

在 sql 语句中，向目的表插入的列数与源表查询结果的列数必须相等。并且目的表和源表之间的列对应关系必须按照 sql 语句中的出现顺序决定，而不是由列名确定。将一个表的所有的列复制到另外一个表中是一种特殊情况，此时可以省略列名，sql 语句可以简写如下：

```
INSERT INTO dst_tb1 SELECT * FROM src_tb1;
```

如果只要复制一行，在 sql 语句中加上一个 `WHERE` 子句，对查找过程进行限制即可，如下：

```
INSERT INTO dst_tb1 SELECT * FROM src_tb1  
WHERE val > 100 AND name LIKE 'A%';
```

其中的 `SELECT` 子句也可以和普通的查询语句一样，使用表达式来计算值。例如，下面的 sql 语句计算 `src_tb1` 中每一个 `name` 出现的次数，然后将 `name` 和出现的次数保存到 `dst_tb1` 中：

```
INSERT INTO dst_tb1 (i, s) SELECT COUNT(*), name  
FROM src_tb1 GROUP BY name;
```

如果目的表事先不存在，就需要首先使用 `CREATE TABLE` 语句创建一个，然后再使用 `INSERT...SELECT` 语句向新创建的目的表插入查询结果。也可以使用一个 sql 语句完成：

CREATE TABLE...SELECT，该语句直接根据查询结果创建目的表。如下语句创建了目的表 dst_tbl，同时将 src_tbl 的内容复制到其中：

```
CREATE TABLE dst_tbl SELECT * FROM src_tbl;
```

MYSQL 根据 src_tbl 表查询结果中的列名、列数和列类型创建目的表 dst_tbl。如果只需要从源表中复制部分列，则需要在 SELECT 子句中加上 WHERE 子句。如果需要创建一个空的目的表，只需要使用一个使得查询结果永远为空的 WHERE 子句即可：

```
CREATE TABLE dst_tbl SELECT * FROM src_tbl WHERE 0;
```

如果只复制部分列，在需要在 SELECT 子句中指明需要复制的列名。例如，src_tbl 中有列 a、b、c 和 d，你可以只复制 b 和 d 两列到目的表：

```
CREATE TABLE dst_tbl SELECT b, d FROM src_tbl;
```

如果希望目的表中列的顺序和源表不同，则需要在 SELECT 子句中显式的指明各列间顺序。如果源表中列 a、b 和 c 按顺序出现，而用户希望复制完成之后各列在目的表中出现顺序为 c、a 和 b，可以使用如下语句：

```
CREATE TABLE dst_tbl SELECT c, a, b FROM src_tbl;
```

在 CREATE TABLE 子句部分插入列定义（类似于类似于创建表时的列定义），可以在目的表中加入查询结果中没有的列。如下语句，在 dst_tbl 中创建了 4 列，其中 a、b 和 c3 列来自源表查询结果，同时加上了定义为 AUTO_INCREMENT 的 id 列：

```
CREATE TABLE dst_tbl
(
    id INT NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id)
)
SELECT a, b, c FROM src_tbl;
```

上例中，目的表 dst_tbl 中 4 列出现顺序为 id、a、b、c。其中 a、b 和 c 散列根据源表查询结果赋值，而 id 列值从 1 开始连续自加，为行序号（详见第 11.1 节）。

如果查询子句中需要使用一个表达式来计算某列值，比较谨慎的做法是使用别名来重命名该列。假设 src_tbl 中每一行保存了一张发票的信息，可以使用如下语句得到表中记录的每一张发票的摘要信息，包括发票编号和总金额。其中查询结果的第二列是一个表达式，默认情况下使用表达式本身作为列名。为了避免冗长的列名在后续工作中带来麻烦，我们使用别名 total_cost 重命名该列：

```
CREATE TABLE dst_tbl
SELECT inv_no, SUM(unit_cost*quantity) AS total_cost
FROM src_tbl
GROUP BY inv_no;
```

使用 CREATE TABLE ... SELECT 来创建目的表，在带来方便的同时也有一些限制。这主要是由于如果使用查询结果来创建表，对新表的描述和定义能力不如直接使用 CREATE

TABLE 语句那么强大。对于新创建的目的表 MySQL 不能确定其某些列属性，例如列索引和默认值。可以使用如下技巧，使得目的表中包含这些信息：

- 如果希望目的表和源表一模一样，那么可以使用在第 4.1 节中讨论的克隆方法。
- 如果希望在目的表中使用索引，可以在 sql 语句中指明。例如，在 src_tbl 中 id 列是主键，state 和 city 两列作为索引，可以赋予 dst_tbl 同样的属性。

```
CREATE TABLE dst_tbl (PRIMARY KEY (id), INDEX(state,city))
SELECT * FROM src_tbl;
```
- 有的列属性不能克隆到目的表，例如 AUTO_INCREMENT 属性和列的默认值。对于这些属性，可以在目的表创建并完成数据复制之后，使用 ALTER TABLE 来设置目的表相应的属性。例如，src_tbl 的 id 列不但具有 PRIMARY KEY 属性，同时还具有 AUTO_INCREMENT 属性，可以这样来完成克隆：

```
CREATE TABLE dst_tbl (PRIMARY KEY (id)) SELECT * FROM src_tbl;
ALTER TABLE dst_tbl MODIFY id INT UNSIGNED NOT NULL AUTO_INCREMENT;
```

4.3 使用临时表

Creating Temporary Tables

问题

如何创建仅供临时使用，并且自动删除的表？

解决方案

创建 TEMPORARY 表，并在使用结束后让 MySQL 自动删除该表。

讨论

有的操作可能需要临时表，并在使用结束后删除。用户可以在使用完临时表之后，使用 DROP TABLE 语句来删除该表，另一个选择是使用 CREATE TEMPORARY TABLE 语句。这个 sql 语句的功能和 CREATE TABLE 类似，只不过它创建的是一张临时表，在和 mysql 服务器的连接关闭之后，创建的临时表将被自动删除。这样，用户在创建了临时表之后，就不需要手动删除。TEMPORARY 关键字可以用在各种创建表的语句中：

- 普通的建表语句：

```
CREATE TEMPORARY TABLE tbl_name (...列定义...);
```

- 克隆表：

```
CREATE TEMPORARY TABLE new_table LIKE original_table;
```

- 根据查询结果建表：

```
CREATE TEMPORARY TABLE tbl_name SELECT ... ;
```

由于临时表是与各个数据库连接相关的，所以不同的数据库连接可以创建同名的临时表，这些表之间不会相互影响。因此在应用程序中需要创建临时表时，只需要保证临时表名在应用程序内部唯一即可，而不用关心所使用的临时表名在数据库服务器上是否唯一（关于表的命名，请参考第 4.5 节）。

临时表具有的另外一个特性是，临时表可以使用普通表的表名。这样做的结果是，在临时表的生命周期内，它将屏蔽与之同名的普通表。利用这个特性，可以创建一个普通表的临时备份，对临时表可以做任意操作，而不影响真实数据。下面的 DELETE 语句从一个临时表中删除了一些行，但是对其对应的普通表没有任何影响。

```
mysql> CREATE TEMPORARY TABLE mail SELECT * FROM mail;
mysql> SELECT COUNT(*) FROM mail;
+-----+
| COUNT(*) |
+-----+
|      16 |
+-----+
mysql> DELETE FROM mail;
mysql> SELECT COUNT(*) FROM mail;
+-----+
| COUNT(*) |
+-----+
|      0 |
+-----+
mysql> DROP TABLE mail;
mysql> SELECT COUNT(*) FROM mail;
+-----+
| COUNT(*) |
+-----+
|      16 |
+-----+
```

尽管使用 CREATE TEMPORARY TABLE 创建的临时表能带来很多便利，但是有些情况下也得小心使用：

- 只有在最后一次使用之后，数据库才会自动删除临时表。如果一个临时表在一个数据库连接内多次重复使用，那么每一次重复使用之前应该先使用 DROP TABLE 显式地删除该临时表（在一个数据库连接中，如果重复两次使用同一个表名创建临时表，而未删除第一次创建的临时表，将导致数据库错误）。
- 应用程序中使用一个与普通表同名的临时表时，应用程序只会对该临时表就行修改。但是如果数据库连接错误断开，而所使用的应用程序自动重新创建了数据库连接，那

么以后对数据库的修改，都将直接作用于普通表，而不是临时表（已经被自动删除）。

- 有的应用程序 API 提供数据库的持久连接或者连接池，可以使用这些技术保证临时表的持久性。因为这样的数据库连接，在应用程序运行结束之后并不会断开，而会被其他的应用程序继续使用。在用户的应用程序关闭之后，就不能再操作所创建的临时表，也就意味着如果不被显示删除，它将一直存在于数据库中。为此，如果应用程序需要使用类似的数据库连接，并需要创建自己的临时表，比较明智的做法是在创建临时表之前执行下面的语句：

```
DROP TEMPORARY TABLE IF EXISTS tbl_name
```

TEMPORARY 关键字用在这里，可以避免错误的删除同名的普通表。例如，临时表 *tbl_name* 已经被删除，而数据库中存在一个 *tbl_name* 的普通表，如果不使用 TEMPORARY 关键字，那么普通表 *tbl_name* 将被删除。

4.4 检查或改变某个表的存储引擎

Checking or Changing a Table's Storage Engine

问题

你需要检测一张表使用哪个存储引擎，以便你确定哪些引擎功能是可用的。或者，你发现另外的引擎所具有的功能特性跟适合你对某表的操作需求，因此你需要更改一张表所用的存储引擎。

解决方案

为了确定一张表所用的存储引擎，你可以使用很多语句中的任意一个。使用含有 ENGINE 子句的 ALTER TABLE 语句来更改表引擎。

讨论

MySQL 支持多种存储引擎，每一中都有不同的特性。例如，InnoDB 和 BDB 支持事务，而 MyISAM 不支持（事务）。如果你需要确定一张表是否支持事务的话，就确定它所使用的引擎。如果你需要在一个事务中使用改表，但是对应的引擎不支持，你可以将改表转换到一个支持事务的引擎上。

检测 INFORMATION_SCHEMA，或使用 SHOW TABLE STATUS，或使用 SHOW CREATE TABLE 语句，来确定一张表当前使用的引擎。如下获取 mail 表的信息：

```
mysql> SELECT ENGINE FROM INFORMATION_SCHEMA.TABLES  
      -> WHERE TABLE_SCHEMA = 'cookbook' AND TABLE_NAME = 'mail';  
+-----+  
| ENGINE |  
+-----+  
| MyISAM |
```

```
+-----+  
mysql> SHOW TABLE STATUS LIKE 'mail'\G  
***** 1. row *****  
      Name: mail  
    Engine: MyISAM  
...  
  
mysql> SHOW CREATE TABLE mail\G  
***** 1. row *****  
     Table: mail  
Create Table: CREATE TABLE `mail` (  
  `t` datetime DEFAULT NULL,  
  `srcuser` char(8) DEFAULT NULL,  
  `srchost` char(20) DEFAULT NULL,  
  `dstuser` char(8) DEFAULT NULL,  
  `dsthost` char(20) DEFAULT NULL,  
  `size` bigint(20) DEFAULT NULL,  
  KEY `t` (`t`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

使用 `ALTER TABLE` 以及一个 `ENGINE` 子句，来改变一张表所用的引擎。例如，使用下面的语句，使 `mail` 表使用 InnoDB 引擎：

```
ALTER TABLE mail ENGINE = InnoDB;
```

注意，改变一个很大的表所用的存储引擎可能会耗费很长时间，并花费大量 CPU 以及 I/O 资源。

参考

为确定 MySQL 服务器支持的存储引擎类型，请参考第 9.13 节。

4.5 生成唯一的表名

Generating Unique Table Names

问题

你需要创建一张表，并保证其表名不与现有表名重复。（如何保证表名唯一？）

解决方案

如果你能创建一张 `TEMPORARY` 表，就无所谓表名是否重复了。否则，试着生成一个对客户端程序唯一的值，并将该值合并到表名中。

讨论

MySQL 是一个多客户端服务器，因此一个创建临时表的脚本可能被多个客户端同时执行，你必须避免多个同时执行的脚本所创建的表名出现冲突。如果这个脚本使用 `CREATE TEMPORARY TABLE` 创建表，不会有问题，因为不同的客户端可以没有冲突地创建重名的临时表。

如果不使用 `TEMPORARY` 表，你最好保证每一个客户端使用脚本创建的表名都是唯一的，并保证删除不再继续使用临时表。为了做到这一点，在表名中加入对于每一个脚本调用都唯一的值。一个 `timestamp` 值不能胜任，因为很有可能两个脚本在同一时刻被调用。一个随机数可能会好一些。例如，在 Java 中，你可以像下面这样使用 `java.util.Random()` 类来创建表名：

```
import java.util.Random;
import java.lang.Math;

Random rand = new Random ();
int n = rand.nextInt (); // 生成随机数
n = Math.abs (n); // 取绝对值
String tblName = "tmp_tbl_" + n;
```

不幸的是，随机数只在一定程度上避免了表名冲突，而不能完全避免。进程号（PID）是唯一值的一个更好来源。PID 被不断重用，但对同一时刻的两个进程（PID）绝不会相同，因此对于一组正在执行的进程而言，给定 PID 值是肯定唯一的。你可以运用这一点生成唯一表名：

Perl:

```
my $tbl_name = "tmp_tbl_$$";
```

Ruby:

```
tbl_name = "tmp_tbl_" + Process.pid.to_s
```

PHP:

```
$tbl_name = "tmp_tbl_" . posix_getpid ();
```

Python:

```
import os
tbl_name = "tmp_tbl_%d" % os.getpid ()
```

注意，就算你使用类似 PID 这样，对一个给定的脚本调用保证唯一的值来生成表名，所生成的表名仍有可能与现有的表名冲突。这种情况可能出现在以前的一个脚本调用过程使用相同的 PID，创建了一个一样的表名，但是这个进程在删除所创建的临时表之前就崩溃了。

另一方面，这样的表已经不再被使用了，因为创建它的进程已经消失了。这种情况下，如果这样的表确实存在，使用下面的语句安全的删除它：

```
DROP TABLE IF EXISTS tbl_name
```

然后你就可以继续创建新表了。

连接标识符是唯一值的另一个来源。MySQL 重用这些值，但是两个同时存在的服务器连接具有不同的 ID。执行下面的语句，获取你的连接 ID：

```
SELECT CONNECTION_ID();
```

某些 MySQL API 可以直接获取连接 ID，而不需要执行任何语句。例如，在 Perl DBI，使用数据库句柄的 mysql_thread_id 属性：

```
my $tbl_name = "tmp_tbl_" . $dbh->{mysql_thread_id};
```

在 Ruby DBI 中，这么做：

```
tbl_name = "tmp_tbl_" + dbh.func(:thread_id).to_s
```

与字符串共舞

Working with Strings

5.0 引言

Introduction

和大多数其他类型的数据一样，字符串类型的数据也可以进行相等、不相等或相对次序等的比较。不过，字符串还要注意一些额外的特性：

- 字符串可以分成二进制和非二进制两类。二进制字符串用于原始数据，例如图像、音乐文件或加密数值等。非二进制字符串用于字符数据例如文本，并且需要被关联到一组字符集和 Collation（排序）上。
- 字符集决定哪些字符在字符串内是合法的，而 Collation 则决定比较字符次序时是否大小写敏感，是否使用特定语言中的某些规则等。
- 用于二进制字符串的数据类型是 `BINARY`、`VARBINARY` 和 `BLOB`，用于非二进制字符串的数据类型是 `CHAR`、`VARCHAR` 和 `TEXT`，其中后三者具有 `CHARACTER SET` 和 `COLLATE` 属性。请参考 5.2 节——选择字符串的数据类型。
- 可以将二进制字符串转换成非二进制字符串，或者相反，也可以将非二进制字符串从一种字符集或 Collation 转换到另一种。
- 可以以完整的形式来使用一个字符串，也可以从中抽取出子字符串，还可以把它同其他字符串相联结。
- 可以对字符串进行模式匹配操作。
- 可以用 `FULLTEXT` 搜索对大量的文本进行高效检索。

本章将讨论如何使用以上所有特性，以便你可以存放、获取和操作字符串以满足你的应用程序的各种需要。

本章所用到表的创建脚本可以在 `recipes` 发行的 `tables` 目录下找到。

5.1 字符串属性

String Properties

属性之一是字符串是二进制的还是非二进制的。

- 二进制字符串是一个字节序列，它可以包含任何类型的信息，例如图像、MP3 文件以及压缩过或加密过的数据等。二进制字符串不与字符集关联，即便你存放的是类似 abc 这样看起来像是普通文本的东西。二进制字符串的比较是通过逐个字节地比较字节数值来进行的。
- 非二进制字符串是一个 Collation 列，它存放包含特定字符集和 Collation 的文本。字符集规定了能够存入字符串的字符，而 Collation 则规定了对字符进行比较和排序时的特性。

非二进制字符串的特征之一是它们有一个字符集。要想看看系统中都有哪些字符集，可以使用下面的命令：

```
mysql> SHOW CHARACTER SET;
+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+
| big5   | Big5 Traditional Chinese | big5_chinese_ci    | 2      |
| dec8   | DEC West European       | dec8_swedish_ci    | 1      |
| cp850  | DOS West European       | cp850_general_ci  | 1      |
| hp8    | HP West European        | hp8_english_ci    | 1      |
| koi8r  | KOI8-R Relcom Russian  | koi8r_general_ci  | 1      |
| latin1 | cp1252 West European   | latin1_swedish_ci | 1      |
| latin2 | ISO 8859-2 Central European | latin2_general_ci | 1      |
...
| utf8   | UTF-8 Unicode           | utf8_general_ci   | 3      |
| ucs2   | UCS-2 Unicode           | ucs2_general_ci  | 2      |
...
```

在 MySQL 中，默认的字符集是 latin1。如果你需要将来自多种语言的字符存放到同一列中，请考虑使用 Unicode 字符集（utf8 或 ucs2）中的一种，因为只有它们才能表示来自多语言的字符。

有些字符集只包含单字节字符，而有些字符集则包含多字节字符。有些多字节字符集中的字符由固定数目的字节来表示，而有些多字节字符集中的字符长度却是可变的。例如：Unicode 数据可以用 ucs2 字符集来保存，其中所有字符均占两个字节，也可以用 utf8 字符集来存放，其中每个字符占一至三个字节不等。

你可以使用 LENGTH() 和 CHAR_LENGTH() 函数来判断给定字符串中是否包含多字节字符，这两个函数分别返回字符串中的字节数和字符数。如果 LENGTH() 大于 CHAR_LENGTH()，那么字符串中一定有多字节字符。

- 在 Unicode 字符集 ucs2 之中，所有的字符都使用两个字节来编码，即便它们在其他字符集如 latin1 中是一个字节。所以，每个 ucs2 字符串都包含有多字节字符。

```

mysql> SET @s = CONVERT('abc' USING ucs2);
mysql> SELECT LENGTH(@s), CHAR_LENGTH(@s);
+-----+-----+
| LENGTH(@s) | CHAR_LENGTH(@s) |
+-----+-----+
|       6 |          3 |
+-----+-----+

```

- 虽然 Unicode 字符集 utf8 包含多字节字符，但是一个具体的 utf8 字符串却有可能只包含单字节字符，如下例所示：

```

mysql> SET @s = CONVERT('abc' USING utf8);
mysql> SELECT LENGTH(@s), CHAR_LENGTH(@s);
+-----+-----+
| LENGTH(@s) | CHAR_LENGTH(@s) |
+-----+-----+
|       3 |          3 |
+-----+-----+

```

非二进制字符串的另一个特征是 Collation，它决定字符集中字符排序的次序。可以使用 SHOW COLLATION 命令来查看有哪些 Collation 可以使用。添加一个 LIKE 子句可以只看特定字符集中的 Collation：

```

mysql> SHOW COLLATION LIKE 'latin1%';
+-----+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| latin1_german1_ci | latin1 | 5 | Yes | Yes | 1 |
| latin1_swedish_ci | latin1 | 8 | Yes | Yes | 1 |
| latin1_danish_ci | latin1 | 15 | | Yes | 1 |
| latin1_german2_ci | latin1 | 31 | | Yes | 2 |
| latin1_bin | latin1 | 47 | | Yes | 1 |
| latin1_general_ci | latin1 | 48 | | Yes | 1 |
| latin1_general_cs | latin1 | 49 | | Yes | 1 |
| latin1_spanish_ci | latin1 | 94 | | Yes | 1 |
+-----+-----+-----+-----+-----+-----+

```

当没有指明使用何种 Collation 时，Default 列中值为 Yes 的 Collation 就是该字符集中字符串默认的 Collation。如上表所示，字符集 latin1 的默认 Collation 就是 latin1_swedish_ci（默认 Collation 也可以用 SHOW CHARACTER SET 来查看）。

Collation 可以是大小写敏感的（a 与 A 被认为是不同的字符）、大小写不敏感的（a 与 A 被认为是相同的字符）或二进制的（两个字符的异同取决于它们的数值是否相等）。名称以 ci、cs 或 bin 结尾的 Collation 分别是大小写不敏感、大小写敏感和二进制的。

从都是使用数值进行比较这点来看，一个二进制 Collation 为非二进制字符串提供排序次序跟二进制字符串排序次序是有点儿像。不过这里仍然存在着区别：二进制字符串的比较总是以单个字节为单元，而二进制 Collation 对非二进制字符串的比较却是以字符为单元，根据字符集不同，有些字符可能会是多字节的。

下面一些例子将说明 Collation 是如何影响排序次序的。假设数据库内有张表，表中有一个 latin1 字符集的字符串列以及如下所示的行：

```
mysql> CREATE TABLE t (c CHAR(3) CHARACTER SET latin1);
mysql> INSERT INTO t (c) VALUES('AAA'),('bbb'),('aaa'),('BBB');
mysql> SELECT c FROM t;
+---+
| c |
+---+
| AAA |
| bbb |
| aaa |
| BBB |
+---+
```

通过对列使用 COLLATE 操作符，你可以选择在排序时使用哪个 Collation 从而影响结果顺序：

- 一个大小写不敏感的 Collation 会将 a 和 A 排序到一起，并排到所有 b 和 B 的前面。当然，对某一个字母来说，没有将某个写法排在另一个写法前面的必要。如下所示：

```
mysql> SELECT c FROM t ORDER BY c COLLATE latin1_swedish_ci;
+---+
| c |
+---+
| AAA |
| aaa |
| bbb |
| BBB |
+---+
```

- 一个大小写敏感的 Collation 也会将 A 和 a 排在 B 和 b 之前，但还将大写排在小写的前面：

```
mysql> SELECT c FROM t ORDER BY c COLLATE latin1_general_cs;
+---+
| c |
+---+
| AAA |
| aaa |
| BBB |
| bbb |
+---+
```

- 一个二进制 Collation 使用字符的数值为它们排序。假设大写字母的数值比小写字母的小，一个二进制 Collation 排出的结果次序如下：

```
mysql> SELECT c FROM t ORDER BY c COLLATE latin1_bin;
+---+
| c |
+---+
| AAA |
| BBB |
| aaa |
+---+
```

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

注意，因为字符不同大小写的数值不同，一个二进制 Collation 会产生一个大小写敏感的排序。不过，这个排序与大小写敏感 Collation 的排序还是不同的。

如果你需要使用特定语言的排序规则进行比较和排序操作，可以选择该语言专有的 Collation。例如，你使用 utf8 字符集存放字符串，默认的 Collation (utf8_general_ci) 会将 ch 和 ll 分别当作有两个字符的字符串。如果你需要的是将 ch 和 ll 当作分别排在 c 和 l 之后单个字符的传统西班牙 Collation 话，可以使用 utf8_spanish2_ci Collation。如下所示，两个 Collation 会产生不同的排序结果：

```
mysql> CREATE TABLE t (c CHAR(2) CHARACTER SET utf8);  
mysql> INSERT INTO t (c) VALUES('cg'),('ch'),('ci'),('lk'),('ll'),('lm');  
mysql> SELECT c FROM t ORDER BY c COLLATE utf8_general_ci;  
+---+  
| c |  
+---+  
| cg |  
| ch |  
| ci |  
| lk |  
| ll |  
| lm |  
+---+  
mysql> SELECT c FROM t ORDER BY c COLLATE utf8_spanish2_ci;  
+---+  
| c |  
+---+  
| cg |  
| ci |  
| ch |  
| lk |  
| lm |  
| ll |  
+---+
```

5.2 选择字符串的数据类型

Choosing a String Data Type

问题

你需要存储字符串数据，但不能确定最合适的数据类型。

解决方案

根据要存储信息的特征以及你想怎么使用它，来选择数据类型。考虑如下问题：

- 字符串是否是二进制数据?
- 是否大小写敏感?
- 字符串的最大长度是多少?
- 想存储定长值还是变长值?
- 是否需要保留尾部空格?
- 是否有固定的允许值集合?

讨论

MySQL 提供了几种二进制和非二进制字符数据类型。这些类型成对出现，如下表所示。

二进制数据类型	非二进制数据类型	最大长度
BINARY	CHAR	255
VARBINARY	VARCHAR	65 535
TINYBLOB	TINYTEXT	255
BLOB	TEXT	65 535
MEDIUMBLOB	MEDIUMTEXT	16 777 215
LONGBLOG	LONGTEXT	4 294 967 295

对于二进制数据类型而言，最大长度是字符串必须能容纳的字节数。对于非二进制类型，则是字符串必须能容纳的字符数（这样对于包含多字节字符的字符串来说，其所需字节数会大于字符数）。

MySQL 用固定长度来储存 BINARY 和 CHAR 数据类型的列值。例如，存在 BINARY(10) 或 CHAR(10) 类型列中的值总是占 10 个字节或 10 个字符。当储存时，如有需要，较短的值会被填满以达到要求的长度。对于 BINARY，填充值是 0x00 (0 值字节，即 ASCII NUL)。CHAR 值会填充空白。在检索时，尾部填充的字节或字符会被从 BINARY 和 CHAR 值中去除。

对于 VARBINARY、VARCHAR 和 BLOB、TEXT 类型，MySQL 仅用需要的存储长度来储存值，直到最大列长度为止。储存或检索时不再需要添加或去除填充值。

如果你想保留出现在储存的原始字符串尾部的填充值，那就应该使用一个没有截除动作发生的数据类型。例如，如果你正储存可能以空格结尾的字符串（非二进制），并且你想保留这些空格，就应使用 VARCHAR 或者 TEXT 数据类型之一。下面的语句说明了 CHAR 和 VARCHAR 列处理尾部空格的差异之处：

```
mysql> CREATE TABLE t (c1 CHAR(10),c2 VARCHAR(10));
mysql> INSERT INTO t (c1,c2) VALUES('abc      ','abc      ');
```

```
mysql> SELECT c1,c2,CHAR_LENGTH(c1),CHAR_LENGTH(c2) FROM t;
+---+---+-----+-----+
| c1 | c2      | CHAR_LENGTH(c1) | CHAR_LENGTH(c2) |
+---+---+-----+-----+
| abc | abc     |      3       |      10      |
+---+---+-----+-----+
```

因此，如果你将包含尾部空格的字符串存入一个 CHAR 列中，当查询该值时，你就会发现这些空格不见了。同样的填充和截取动作也发生 BINARY 列，只是这里的填充值是 0x00。



提示：在 MySQL5.0.3 之前的版本中，VARCHAR 和 VARBINARY 的最大长度为 255。截去尾部填充值也同样适用于 VARCHAR 和 VARBINARY 列，所以如果你想保留尾部空格或 0x00 字节，就应使用 TEXT 或 BLOB 类型之一。

一个表既能包含二进制字符串列，也能包含非二进制字符串列，并且它的非二进制列能使用不同的字符集和 Collation。当你声明一个非二进制字符串列时，如果你需要特定的字符集和 Collation，应使用 CHARACTER SET 和 COLLATE 属性。例如，如果你需要储存 utf8 (Unicode) 和 sjis (Japanese) 字符串，你可以以如下的方式来定义表：

```
CREATE TABLE mytbl
(
    utf8data VARCHAR(100) CHARACTER SET utf8 COLLATE utf8_danish_ci,
    sjisdata VARCHAR(100) CHARACTER SET sjis COLLATE sjis_japanese_ci
);
```

列的定义中可以省去 CHARACTER SET 或 COLLATE，或两者都略去：

- 如果你指定 CHARACTER SET 而略去 COLLATE，将使用该字符集默认的 Collation。
- 如果你指定 COLLATE 而略去 CHARACTER SET，将使用 Collation 名称所指的字符集(名称的第一部分)。例如，utf8_danish_ci 和 sjis_japanese_ci 分别指 utf8 和 sjis。(这也意味着在前面的 CREATE TABLE 语句中可以略去 CHARACTER SET 属性。)
- 如果你两个都省略，该列会被赋以表默认的字符集和 Collation。(表定义能在 CREATE TABLE 语句结尾的圆括号后包含这些属性。如果定义了这些属性，将会应用到没有显式定义自身字符集或 Collation 的列。如果省略，表默认为数据库默认的属性。数据库默认值可以在你用 CREATE DATABASE 语句创建表时指定。如果也没有指定，那么服务器默认值将应用于该数据库。)

除非你在启动服务器时使用--character-set-server 和--collation-server 选项指

定不同值，否则服务器默认字符集和 Collation 为 latin1 和 latin1_swedish_ci。这也意味着，字符串默认使用 latin1 字符集而且不是大小写敏感的。

MySQL 也支持 ENUM 和 SET 字符串类型，它们用于储存有固定取值集合的数据。你同样能将 CHARACTER SET 和 COLLATE 属性用于这些数据类型。

5.3 正确设置客户端连接的字符集

Setting the Client Connection Character Set Properly

问题

当你在执行 SQL 语句或生成查询结果时，并未使用系统默认的字符集。

解决方案

使用 SET NAMES 或者一个等价的方法来将你的连接设置到正确的字符集。

讨论

当你在你的应用程序和服务器之间交互发送信息时，应当告知 MySQL 应使用的合适字符集。例如，默认的字符集是 latin1，但是它并非始终是用来连接服务器的正确字符集。假如你正在处理希腊字符数据，试图使用 latin1 字符集在屏幕上显示数据将会导致满屏乱码。如果你在 utf8 字符集中使用 Unicode 字符串，latin1 字符集可能不足以表示你需要的所有字符。

为了解决这个问题，需要设置你和服务器之间的连接来使用合适的字符集。实现这个目的有许多方法：

- 如果你的客户端程序支持--default-character-set 选项，那么你可以在程序调用时使用它来设置字符集。mysql 就是一个这样的程序。把这个选项放到一个选项配置文件中，这样在你每次连接服务器的时候都会生效：

```
[mysql]
default-character-set=utf8
```

- 建立连接后执行 SET NAMES 语句：

```
mysql> SET NAMES 'utf8';
```

SET NAMES 同时也允许指定连接的 Collation：

```
mysql> SET NAMES 'utf8' COLLATE 'utf8-general_ci';
```

- 一些编程接口提供它们自己的方法来设置字符集。Java 客户端的 MySQL Connector/J

就是一个这样的接口。它能在你建立连接时自动检测服务器端使用的字符集，不过你也可以在连接 URL 中通过使用 `characterEncoding` 属性来显式地指定一个不同字符集。这个属性的值应该是 Java 风格的字符集名称。例如，为了选择 `utf8` 字符集，你应该使用如下的 URL 连接：

```
jdbc:mysql://localhost/cookbook?characterEncoding=UTF-8
```

这个方法比 `SET NAMES` 更可取，因为它在应用程序中来执行字符集转换。如果你使用 `SET NAMES` 那么 MySQL Connector/J 就不知道要使用哪一个字符集。

顺便提一句，确保你的显示设备使用的字符集与 MySQL 使用的字符集相匹配。否则，即使 MySQL 正确地处理了数据，它依然无法被正确显示。假设你正在一个终端窗口上使用 `mysql` 程序，并将 MySQL 设置为使用 `utf8` 以及存储 `utf8` 编码的日文数据。如果你将终端窗口设置为使用 `euc-jp` 编码（也是日文数据，但其对日文字符的编码与 `utf8` 不同），那么数据将不会如你所愿的被正确显示出来。



提示： `ucs2` 不能被用作连接字符集。

5.4 串字母

Writing String Literals

问题

用户需要在 SQL 语句中加入字符串。

解决方案

了解管理字符串值的语法规则来保证其正确性。

讨论

可以使用如下几种方式来定义一个字符串：

- 把字符串文本放在单引号或者双引号内。

```
'my string'  
"my string"
```

需要注意的是，当 `ANSI_QUOTES` SQL 模式启用时，不能把字符串放在双引号内。该模式启用时，服务器将双引号所包含的内容解释为表名或列名，而不是普通字符串。（详见 2.6）因此常用的处理方法都是使用单引号来包含用户字符串。这样数据

库服务器将会忽略 ANSI_QUOTES 设置，正确的将引号内容解释为用户字符串，而非标识符。

- 使用十六进制数标记。每一对十六进制数产生字节字符串。可以使用以下任意一种格式表示 abcd：

```
0x61626364  
X'61626364'  
x'61626364'
```

MySQL 将使用了十六进制标记的字符串视为二进制字符串。尽管没有明确要求，应用程序通常在 SQL 语句中使用十六进制字符串来指代二进制数值：

```
INSERT INTO t SET binary_col = 0xdeadbeef;
```

- 通过使用由一个字符集加下划线（“_”）前缀组成的引入器，为用户字符串指定一个字符集作为解释器：

```
_latin1 'abcd'  
_ucs2 'abcd'
```

引入器通知数据库服务器以何种方式解释后续的用户字符串。例如：服务器将 _latin1 'abcd' 解释为含有四个单字节字符的字符串；因为 ucs2 是双字节字符集，服务器将 _ucs2 'abcd' 解释为含有两个双字节字符的字符串。

使用 5.6 节给出的字符串转换指令，来确保一个二进制或非二进制字符串使用特定的字符集或 Collation。

如果你需要使用引号来包含一个含有引号的字符串，以如下方式将引号放在字符串内，将导致一个语法错误：

```
mysql> SELECT 'I'm asleep';  
ERROR 1064 (42000): You have an error in your SQL syntax near 'asleep'
```

有以下几种方式来处理这个问题：

- 使用双引号来标示含有单引号的字符串（假设 ANSI_QUOTES 模式未启用）：

```
mysql> SELECT "I'm asleep";  
+-----+  
| I'm asleep |  
+-----+  
| I'm asleep |  
+-----+
```

反之亦然，使用单引号来标示含有双引号的字符串：

```
mysql> SELECT 'He said, "Boo!"';  
+-----+  
| He said, "Boo!" |  
+-----+
```

```
| He said, "Boo!" |  
+-----+
```

- 使用与字符串内所包含的引号相同的引号来标示用户字符串，此时需要重复在字符串内出现的引号或者在引号前加反斜线。数据库在处理这样的语句时，会自动去除多余的引号或反斜线。

```
mysql> SELECT 'I''m asleep', 'I\'m wide awake';  
+-----+  
| I'm asleep | I'm wide awake |  
+-----+  
| I'm asleep | I'm wide awake |  
+-----+  
mysql> SELECT "He said, ""Boo!""", "And I said, \"Yikes!\\"";  
+-----+  
| He said, "Boo!" | And I said, "Yikes!" |  
+-----+  
| He said, "Boo!" | And I said, "Yikes!" |  
+-----+
```

反斜线将其后的任意特殊字符转义为普通字符。（它导致暂时的不以正常规则处理字符串，类似于\`和\"都称为转义字符）这就意味着反斜线也是一个特殊字符。如果要在字符串中写反斜线，你必须重复两次：

```
mysql> SELECT 'Install MySQL in C:\\mysql on Windows';  
+-----+  
| Install MySQL in C:\\mysql on Windows |  
+-----+  
| Install MySQL in C:\\mysql on Windows |  
+-----+
```

MySQL 可识别的其他转意字符有：\b（退格）、\n（新行，也称为换行）、\r（回车）、\t（制表）和\0（ASCII 符号，空）。

- 使用十六进制数来表示字符串：

```
mysql> SELECT 0x49276D2061736C656570;  
+-----+  
| 0x49276D2061736C656570 |  
+-----+  
| I'm asleep |  
+-----+
```

参考

如果你在程序内部处理 SQL 语句，可以使用程序变量来引用字符串或者二进制值，通过程序接口来处理引号，例如：使用编程语言的数据库访问 API 提供的占位符机制。详见 2.5 节。或者如 18.6 节所示范例，使用 LOAD_FILE() 函数从文件中导入诸如图片的二进制值。

5.5 检查一个字符串的字符集或字符排序

Checking a String's Character Set or Collation

问题

你想知道一个字符串的字符集或 Collation。

解决方案

使用 `CHARSET()` 或 `COLLATION()` 函数。

讨论

如果你使用下面的定义来创建一个表，你就会知道列中所存储的值的字符集和 Collation 分别为 `utf8` 和 `utf8_danish_ci`：

```
CREATE TABLE t (c CHAR(10) CHARACTER SET utf8 COLLATE utf8_danish_ci);
```

但是有些时候，并不清楚应用到一个字符串的是什么字符集或 Collation。服务器配置影响字母组成的字符串以及一些字符串函数，其他一些字符串函数会返回特定字符集中的值。比较操作发生 Collation 失配错误，或者大小写转换不能正常工作，这些都是字符集或 Collation 错误的表现。本章展示了如何检查一个字符串的字符集或 Collation。5.6 节展示了如何将字符串转换到不同的字符集或 Collation。

为确定一个字符串的字符集或 Collation，可以使用 `CHARSET()` 函数或 `COLLATION()` 函数。例如，你知道 `USER()` 函数会返回一个 Unicode 编码的字符串吗？

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+-----+-----+-----+
| USER() | CHARSET(USER()) | COLLATION(USER()) |
+-----+-----+-----+
| cbuser@localhost | utf8 | utf8_general_ci |
+-----+-----+-----+
```

当配置变化时，从当前配置中获取字符集和 Collation 的字符串值属性也会发生变化。这一点对字母组成的字符串来说是真的：

```
mysql> SET NAMES 'latin1';
mysql> SELECT CHARSET('abc'), COLLATION('abc');
+-----+-----+
| CHARSET('abc') | COLLATION('abc') |
+-----+-----+
| latin1 | latin1_swedish_ci |
+-----+-----+
mysql> SET NAMES latin7 COLLATE 'latin7_bin';
mysql> SELECT CHARSET('abc'), COLLATION('abc');
+-----+-----+
| CHARSET('abc') | COLLATION('abc') |
+-----+-----+
```

```
| latin7           | latin7_bin          |
+-----+-----+
```

对于二进制字符串，`CHARSET()`或`COLLATION()`函数会返回值`binary`，这意味着字符串是基于数字的字节值而不是字符的 Collation 值进行比较和存储的。一些函数会返回二进制字符串，例如`MD5()`和`PASSWORD()`：

```
mysql> SELECT CHARSET(MD5('a')), COLLATION(MD5('a'));
+-----+-----+
| CHARSET(MD5('a')) | COLLATION(MD5('a')) |
+-----+-----+
| binary          | binary          |
+-----+-----+
mysql> SELECT CHARSET(PASSWORD('a')), COLLATION(PASSWORD('a'));
+-----+-----+
| CHARSET(PASSWORD('a')) | COLLATION(PASSWORD('a')) |
+-----+-----+
| binary          | binary          |
+-----+-----+
```

一个函数或字符串表达式产生二进制字符串，如果你尝试在此结果上执行大小写转换，那么就会失败，知道这一点很有用。详见 5.8 节。

5.6 改变字符串的字符集或字符排序

Changing a String's Character Set or Collation

问题

有时候你需要将某个字符串的字符集或 Collation 由一种转换到另一种。

解决方案

使用`CONVERT()`函数转换字符串的字符集，使用`COLLATE`操作符改变字符串的 Collation。

讨论

使用`CONVERT()`函数改变字符串的字符集：

```
mysql> SET @s1 = 'my string';
mysql> SET @s2 = CONVERT(@s1 USING utf8);
mysql> SELECT CHARSET(@s1), CHARSET(@s2);
+-----+-----+
| CHARSET(@s1) | CHARSET(@s2) |
+-----+-----+
| latin1      | utf8        |
+-----+-----+
```

使用`COLLATE`操作符改变字符串的 Collation：

```
mysql> SET @s1 = 'my string';
mysql> SET @s2 = @s1 COLLATE latin1_spanish_ci;
mysql> SELECT COLLATION(@s1), COLLATION(@s2);
+-----+-----+
| COLLATION(@s1) | COLLATION(@s2) |
+-----+-----+
| latin1_swedish_ci | latin1_spanish_ci |
+-----+-----+
```

新的 Collation 对于字符串的字符集必需是合法的。举例来说，Collation utf8_general_ci 可以适用于 utf8 字符串，但是不能用在 latin1 字符串上：

```
mysql> SELECT _latin1 'abc' COLLATE utf8_bin;
ERROR 1253 (42000): COLLATION 'utf8_bin' is not valid for
CHARACTER SET 'latin1'
```

如果需要同时改变字符串的字符集和 Collation，则首先使用 CONVERT() 改变字符集，再对结果使用 COLLATE 操作符：

```
mysql> SET @s1 = 'my string';
mysql> SET @s2 = CONVERT(@s1 USING utf8) COLLATE utf8_spanish_ci;
mysql> SELECT CHARSET(@s1), COLLATION(@s1), CHARSET(@s2), COLLATION(@s2);
+-----+-----+-----+-----+
| CHARSET(@s1) | COLLATION(@s1) | CHARSET(@s2) | COLLATION(@s2) |
+-----+-----+-----+-----+
| latin1 | latin1_swedish_ci | utf8 | utf8_spanish_ci |
+-----+-----+-----+-----+
```

CONVERT() 函数还可以将二进制字符串转变为非二进制字符串或者反向转换。使用 binary 可以产生二进制字符串，而使用字符集名可以产生非二进制字符串：

```
mysql> SET @s1 = 'my string';
mysql> SET @s2 = CONVERT(@s1 USING binary);
mysql> SET @s3 = CONVERT(@s2 USING utf8);
mysql> SELECT CHARSET(@s1), CHARSET(@s2), CHARSET(@s3);
+-----+-----+-----+
| CHARSET(@s1) | CHARSET(@s2) | CHARSET(@s3) |
+-----+-----+-----+
| latin1 | binary | utf8 |
+-----+-----+-----+
```

产生二进制字符串的另一种方式是 BINARY 操作符，这与 CONVERT(str USING binary) 函数是等效的：

```
mysql> SET @s1 = 'my string';
mysql> SET @s2 = BINARY @s2;
mysql> SELECT CHARSET(@s1), CHARSET(@s2);
+-----+-----+
| CHARSET(@s1) | CHARSET(@s2) |
+-----+-----+
| latin1 | binary |
+-----+-----+
```

5.7 更改字符串字母的大小写

Converting the Lettercase of a String

问题

如何改变字符串的大小写？

解决方案

使用 `UPPER()` 或者 `LOWER()` 函数。如果这两个函数失效，请参考 5.8 节中的讨论。

讨论

`UPPER()` 和 `LOWER()` 函数可以改变字符串大小写：

```
mysql> SELECT thing, UPPER(thing), LOWER(thing) FROM limbs;
+-----+-----+-----+
| thing | UPPER(thing) | LOWER(thing) |
+-----+-----+-----+
| human | HUMAN | human |
| insect | INSECT | insect |
| squid | SQUID | squid |
| octopus | OCTOPUS | octopus |
| fish | FISH | fish |
| centipede | CENTIPEDE | centipede |
| table | TABLE | table |
| armchair | ARMCHAIR | armchair |
| phonograph | PHONOGRAPH | phonograph |
| tripod | TRIPOD | tripod |
| Peg Leg Pete. | PEG LEG PETE | peg leg pete |
| space alien | SPACE ALIEN | space alien |
+-----+-----+-----+
```

如果只想修改某些子串的大小写，可以先取出目标子串进行操作，然后再将结果和其他部分串联。如下语句，将字符串 `str` 的第一个字符改为大写，其他部分保持不变：

```
CONCAT(UPPER(LEFT(str,1)),MID(str,2))
```

如果类似的操作重复多次，为了避免重复输入冗长的表达式，可以定义一个函数：

```
mysql> CREATE FUNCTION initial_cap (s VARCHAR(255))
-> RETURNS VARCHAR(255) DETERMINISTIC
-> RETURN CONCAT(UPPER(LEFT(s,1)),MID(s,2));
```

有了 `initial_cap` 函数的帮助，SQL 语句变得简单了很多：

```
mysql> SELECT thing, initial_cap(thing) FROM limbs;
+-----+-----+
| thing | initial_cap(thing) |
+-----+-----+
| human | Human |
| insect | Insect |
| squid | Squid |
```

```

| octopus      | Octopus
| fish         | Fish
| centipede   | Centipede
| table        | Table
| armchair    | Armchair
| phonograph  | Phonograph
| tripod       | Tripod
| Peg Leg Pete | Peg Leg Pete
| space alien  | Space alien
+-----+

```

关于 sql 语句中函数申明的更多信息，请参考第 16 章。

5.8 更改字符串大小写失败的情况

Converting the Lettercase of a Stubborn String

问题

如何处理 UPPER() 和 LOWER() 函数失效的情况？

解决方案

当这两个函数失效时，原因很可能是它们所操作的对象是二进制串。此时，可以首先将该字符串转换为非二进制串，使其具有字符集和 Collation 属性，从而可以改变其大小写。

讨论

通常使用 UPPER() 或 LOWER() 函数来改变字符串大小写：

```

mysql> SET @s = 'aBcD';
mysql> SELECT UPPER(@s), LOWER(@s);
+-----+-----+
| UPPER(@s) | LOWER(@s) |
+-----+-----+
| ABCD     | abcd     |
+-----+-----+

```

但是当某一列的数据类型为 BINARY 或 BLOB 时，作用于该列值的 UPPER 和 LOWER 函数都会失效：

```

mysql> CREATE TABLE t (b BLOB) SELECT 'aBcD' AS b;
mysql> SELECT b, UPPER(b), LOWER(b) FROM t;
+-----+-----+-----+
| b    | UPPER(b) | LOWER(b) |
+-----+-----+-----+
| aBcD | aBcD    | aBcD    |
+-----+-----+-----+

```

导致函数操作失败的原因是，函数作用的对象是二进制串，而二进制串没有字符集和 Collation，也没有字母大小写的区别。这可能会让读者感到疑惑，因为事实上在较老版本

的 MySQL 中，二进制串是可以进行大小写转换的，那为什么现在不行了呢？为此，先让我们了解一段历史：

- 在 MySQL4.1 之前的版本中，所有的字符串，包括二进制串，都由服务器指定一个默认的字符集。根据该字符集，对于二进制串也可以使用 UPPER() 和 LOWER() 函数进行大小写转换：

```
mysql> SET @s = BINARY 'aBcD';
mysql> SELECT @s, LOWER(@s), UPPER(@s);
+-----+-----+-----+
| @s   | LOWER(@s) | UPPER(@s) |
+-----+-----+-----+
| aBcD | abcd     | ABCD      |
+-----+-----+-----+
```

- MySQL4.1 对字符集管理做了较大的修改，其中一条就是字符集和 Collation 只能作用于非二进制串。在 4.1 以后的版本中，二进制串定义为以字节为单位的 0、1 串，就算用户在二进制串中加入了字母，服务器也只能将其识别为普通的 0、1 串，没有大小写的区别。因此，将 UPPER() 函数和 LOWER() 函数作用于二进制串就没有意义了。

```
mysql> SET @s = BINARY 'aBcD';
mysql> SELECT @s, LOWER(@s), UPPER(@s);
+-----+-----+-----+
| @s   | LOWER(@s) | UPPER(@s) |
+-----+-----+-----+
| aBcD | aBcD    | aBcD     |
+-----+-----+-----+
```

为了进行大小写转换，首先要将二进制串转换为非二进制串，并采用同时具有大小写定义的字符集。这样，UPPER() 和 LOWER() 函数就可以利用 Collation 中的大小写匹配规则进行大小写转换。如下例中，采用上述方法对类型定义为 BLOB 的列 b 进行大小写转换：

```
mysql> SELECT b,
-> UPPER(CONVERT(b USING latin1)) AS upper,
-> LOWER(CONVERT(b USING latin1)) AS lower
-> FROM t;
+-----+-----+-----+
| b    | upper | lower |
+-----+-----+-----+
| aBcD | ABCD  | abcd  |
+-----+-----+-----+
```

当所要进行大小写转换的对象是函数返回值时，也可能出现大小写转换失败，因为函数返回值也可能是二进制串。例如不能直接对 MD5() 和 COMPRESS() 的返回值进行大小写转换。

为了确定一个表达式（函数）的返回值是否是二进制串，可以使用 CHARSET() 函数。如下语句中，用户可以确定 VERSION() 函数返回一个非二进制串，而 MD5() 返回一个二进制串：

```
mysql> SELECT CHARSET(VERSION()), CHARSET(MD5('some string'));
+-----+-----+
| CHARSET(VERSION()) | CHARSET(MD5('some string')) |
+-----+-----+
| utf8              | binary           |
+-----+-----+
```

也就是说，可以直接对 VERSION() 函数的返回值进行大小写转换，但是对于 MD5() 函数，首先需要将其结果转换为非二进制格式：

```
mysql> SELECT UPPER(VERSION());
+-----+
| UPPER(VERSION()) |
+-----+
| 5.1.12-BETA-LOG |
+-----+
mysql> SELECT UPPER(CONVERT(MD5('some string') USING latin1));
+-----+
| UPPER(CONVERT(MD5('some string') USING latin1)) |
+-----+
| 5AC749FBEEC93607FC28D666BE85E73A |
+-----+
```

5.9 控制字符串比较中的大小写敏感

Controlling Case Sensitivity in String Comparisons

问题

如何比较两个字符串的大小？

解决方案

使用比较操作符。但需要注意字符串的大小写问题，根据需要选择所作比较是否是大小写敏感的。

讨论

与其他数据类型一样，字符串也可以比较大小：

```
mysql> SELECT 'cat' = 'cat', 'cat' = 'dog';
+-----+-----+
| 'cat' = 'cat' | 'cat' = 'dog' |
+-----+-----+
|          1   |          0   |
+-----+-----+
mysql> SELECT 'cat' != 'cat', 'cat' != 'dog';
+-----+-----+
| 'cat' != 'cat' | 'cat' != 'dog' |
+-----+-----+
|          0   |          1   |
```

```
+-----+-----+
mysql> SELECT 'cat' < 'awk', 'cat' < 'dog';
+-----+-----+
| 'cat' < 'awk' | 'cat' < 'dog' |
+-----+-----+
|          0 |           1 |
+-----+-----+
mysql> SELECT 'cat' BETWEEN 'awk' AND 'egret';
+-----+
| 'cat' BETWEEN 'awk' AND 'egret' |
+-----+
|           1 |
+-----+
```

然而，由于字符串类型本身比较复杂，对其进行比较和排序的过程中有很多其他类型不具有的属性。例如，在字符串比较过程中，要决定是否将字符大小写的区别考虑在内。本节就主要讨论字符串比较中的大小写敏感问题。5.12 节中将讨论字符串匹配中的大小写敏感问题。

所要比较的是二进制串还是非二进制串，决定了字符串比较的很多特性：

- 二进制串是以字节为单位的 0、1 串，它们的比较基于每个字节的数值，并且不存在大小写定义。然而，同一个字符的大小写，对应的二进制数值是不同的。也就是说二进制串的比较是大小写敏感的（例如 a 不等于 A）。如果要使二进制字符串比较过程大小写不敏感，可以先将其转换为非二进制字符串，并使用大小写不敏感的字符集，再进行比较。
- 非二进制串是 Collation 列，它们的比较是基于每一个字符的（在有的字符集中，一个字符可能占用多个字节）。非二进制字符串通过一个字符集定义了其中可以出现的合法字符，并在一个 Collation 中定义了字符之间的大小顺序。该 Collation 也同时定义了字符串比较过程是否大小写敏感。因此，可以在进行比较之前选择合适的 Collation，以决定所进行的字符串比较是否是大小写敏感的。

非二进制字符串采用默认字符集 latin1 和默认 Collation latin1_swedish_ci，在默认情况下字符串比较不是大小写敏感的。

在下面的例子中可以看到，将两个不相等的二进制串转换为非二进制串之后，在大小写不敏感的情况下它们是相等的：

```
mysql> SET @s1 = BINARY 'cat', @s2 = BINARY 'CAT';
mysql> SELECT @s1 = @s2;
+-----+
| @s1 = @s2 |
+-----+
|          0 |
+-----+
mysql> SET @s1 = CONVERT(@s1 USING latin1) COLLATE latin1_swedish_ci;
```

```
mysql> SET @s2 = CONVERT(@s2 USING latin1) COLLATE latin1_swedish_ci;
mysql> SELECT @s1 = @s2;
+-----+
| @s1 = @s2 |
+-----+
|          1 |
+-----+
```

latin1 的默认 Collation 就是 latin1_swedish_ci，上面 sql 语句可以省略 COLLATE：

```
mysql> SET @s1 = CONVERT(@s1 USING latin1);
mysql> SET @s2 = CONVERT(@s2 USING latin1);
mysql> SELECT @s1 = @s2;
+-----+
| @s1 = @s2 |
+-----+
|          1 |
+-----+
```

下例说明了相同字符串，在大小写不敏感（第一个 SELECT 语句）和大小写敏感（第二个 SELECT 语句）两种比较下的不同结果：

```
mysql> SET @s1 = _latin1 'cat', @s2 = _latin1 'CAT';
mysql> SELECT @s1 = @s2;
+-----+
| @s1 = @s2 |
+-----+
|          1 |
+-----+
mysql> SELECT @s1 COLLATE latin1_general_cs = @s2 COLLATE latin1_general_cs
      ->   AS '@s1 = @s2';
+-----+
| @s1 = @s2 |
+-----+
|          0 |
+-----+
```

如果将一个二进制串和一个非二进制串进行比较，默认情况下，数据库会将两个字符串都作为二进制串：

```
mysql> SELECT _latin1 'cat' = BINARY 'CAT';
+-----+
| _latin1 'cat' = BINARY 'CAT' |
+-----+
|          0 |
+-----+
```

因此，如果希望两个非二进制串作为二进制串进行比较，只需要将其中任意一个转换为二进制串即可：

```
mysql> SET @s1 = _latin1 'cat', @s2 = _latin1 'CAT';
mysql> SELECT @s1 = @s2, BINARY @s1 = @s2, @s1 = BINARY @s2;
+-----+
| @s1 = @s2 | BINARY @s1 = @s2 | @s1 = BINARY @s2 |
+-----+
```

```
+-----+-----+-----+
|       1 |       0 |       0 |
+-----+-----+-----+
```

根据所要做的比较操作，可以使用 ALTER TABLE 操作修改表中的列定义，使其具有合适的类型定义。假设有一张表用来保存新文章，定义如下：

```
CREATE TABLE news
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    article BLOB,
    PRIMARY KEY (id)
);
```

其中 article 列的类型定义为 BLOB（二进制字符串）。对该列进行的比较，是以字节为单位，基于数值的比较，可以看作是大小写敏感的。如果要进行基于字符，大小写不敏感的比较，可以像这样用 ALTER TABLE 改变 article 列定义：

```
ALTER TABLE news
    MODIFY article TEXT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

5.10 使用 SQL 模式进行模式匹配

Pattern Matching with SQL Patterns

问题

如何进行字符串模式匹配，而不仅仅是字符比较。

解决方案

使用本节将要讨论的 LIKE 操作符和 SQL 模式定义，或者使用 5.11 节中讨论的正则表达式进行模式匹配。

讨论

模式可以看作是由特殊字符定义的特殊字符串，也就是通常所说的 metacharacters，在使用过程中，它们代表了符合特定模式的字符串，而非其自身。MySQL 提供两种模式匹配方法，一种基于 SQL 模式，另一种基于正则表达式。SQL 模式在多种数据库平台上都可以使用，而正则表达式功能更强大。两种匹配方法使用不同的操作符和 metacharacters。在本节主要讨论基于 SQL 模式的字符串匹配，在 5.11 节中将讨论正则表达式匹配。

本节的例子中用到命名为 metal 的表，其内容如下：

```
SELECT * FROM metal;
+-----+
| name   |
+-----+
| copper |
```

```
+-----+  
| gold |  
| iron |  
| lead |  
| mercury |  
| platinum |  
| silver |  
| tin |  
+-----+
```

SQL 模式使用操作符 LIKE 和 NOT LIKE 来匹配字符串和模式串，而不使用=和!=。在模式串中通常使用两个匹配符：_ 用来匹配任意一个字符，% 用来匹配任意一个字符串（包括空串）。可以使用这两个匹配符构造出很多模式串：

- 以特定字符开头的字符串：

```
mysql> SELECT name FROM metal WHERE name LIKE 'co%';  
+-----+  
| name |  
+-----+  
| copper |  
+-----+
```

- 以特定字符结尾的字符串：

```
mysql> SELECT name FROM metal WHERE name LIKE '%er';  
+-----+  
| name |  
+-----+  
| copper |  
| silver |  
+-----+
```

- 含有特定字符的字符串：

```
mysql> SELECT name FROM metal WHERE name LIKE '%er%';  
+-----+  
| name |  
+-----+  
| copper |  
| mercury |  
| silver |  
+-----+
```

- 在特定位置出现特定字符（如下语句，只有在 name 字符串的第三个字符是 p 时才会与模式串匹配）。

```
mysql> SELECT name FROM metal where name LIKE '__pp%';  
+-----+  
| name |  
+-----+  
| copper |  
+-----+
```

基于 SQL 模式的匹配过程要求字符串与模式串完全匹配，匹配才会成功。如下，第一个匹配返回结果为假，第二个匹配过程才是成功的。

```
'abc' LIKE 'b'  
'abc' LIKE '%b%'
```

可以使用 NOT LIKE 实现与 LIKE 相反的模式匹配过程（不匹配时返回结果为真）。下面的语句查询不含有 i 的字符串：

```
mysql> SELECT name FROM metal WHERE name NOT LIKE '%i%';  
+-----+  
| name |  
+-----+  
| copper |  
| gold |  
| lead |  
| mercury |  
+-----+
```

不管是 LIKE 操作符还是 NOT LIKE 操作符，SQL 模式串与 NULL 值匹配的结果总是 NULL：

```
mysql> SELECT NULL LIKE '%', NULL NOT LIKE '%';  
+-----+-----+  
| NULL LIKE '%' | NULL NOT LIKE '%' |  
+-----+-----+  
| NULL | NULL |  

```

对非字符串值进行模式匹配

和其他数据库系统不同的是，MySQL 支持对非字符串值进行模式匹配，例如数值类型 (number) 和日期类型 (date)。下面的例子中，使用函数获取一个日期类型值的各个域值（年、月、日），并对该值进行模式匹配。下面的模式匹配表达式对 1976 年，或者 4 月，或者 1 号的日期值返回为真。

函数值测试	模式匹配测试
YEAR(d) = 1976	d LIKE '1976-%'
MONTH(d) = 4	d LIKE '%-04-%'
DAYOFMONTH(d) = 1	d LIKE '%-01'

在有些情况下，字符串匹配与子串比较是等价的。例如，使用模式匹配来查找以特定字符串开始或者结尾的字符串，就相当于使用 LEFT() 或者 RIGHT() 函数取得子串，然后直接比较该子串和模式串：

模式匹配	子字符串比较
str LIKE 'abc%'	LEFT(str, 3) = 'abc'
str LIKE '%abc'	RIGHT(str, 3) = 'abc'

如果你正匹配一个被索引的列，这时你要选择使用模式或一个等价的 LEFT() 表达式，你可能会发现模式匹配更快。MySQL 对于以字符串开始的模式，能使用索引来缩小搜索范围。而用 LEFT() 则不能。

5.11 使用正则表达式进行模式匹配

Pattern Matching with Regular Expressions

问题

如何对字符串进行更加复杂的模式匹配？

解决方案

使用 REGEXP 操作符和正则表达式，或者使用 5.10 节中讨论的 SQL 模式。

讨论

很多数据库系统都支持 SQL 模式（见 5.10 节），因此它具有一定的可移植性。但是它也有一定的局限性，例如可以使用 SQL 模式串 %abc% 找到所有含有 abc 的字符串，但是很难使用 SQL 模式串来匹配含有 a、b 或 c 中任意一个字母的字符串。也很难使用 SQL 模式串来区分仅由字母或者数字组成的字符串。为此，MySQL 提供了功能更强大的，基于正则表达式的模式匹配操作——REGEXP（或者是不匹配操作，NOT REGEXP）。下表中为正则表达式中可以使用的模式字符：

模式字符	定义
^	匹配字符串的开始部分
\$	匹配字符串的结束部分
.	匹配任何字符（包括回车和新行）
[...]	括号内任意一个字符
[^...]	除了括号内所列字符之外的任意一个字符
p1 p2 p3	p1、p2 或 p3 中任意一个模式串
*	匹配 0 或多个*之前的任何序列
+	匹配 1 或多个+之前的任何序列
{ n }	n 个{n}之前的任何序列
{ m , n }	最少 m 个，最多 n 个，{m, n} 之前的任何序列

这些模式匹配符，与 vi、grep、sed 或者其他 Unix 系统上的正则表达式中的模式匹配符是相同的，并且和大多数编程语言也类似。（第 10 章中讨论了如何在应用程序中使用模式匹配来进行数值验证和转化）。

5.10 节中讨论了如何使用 SQL 模式来查询以某子串开头、结尾或者含有某子串的字符串。可以使用正则表达式实现相同的功能：

- 以特定子串开头

```
mysql> SELECT name FROM metal WHERE name REGEXP '^co';
+-----+
| name |
+-----+
| copper |
+-----+
```

- 以特定子串结尾

```
mysql> SELECT name FROM metal WHERE name REGEXP 'er$';
+-----+
| name |
+-----+
| copper |
| silver |
+-----+
```

- 含有特定子串

```
mysql> SELECT name FROM metal WHERE name REGEXP 'er';
+-----+
| name |
+-----+
| copper |
| mercury |
| silver |
+-----+
```

- 特定子串出现在某一位置

```
mysql> SELECT name FROM metal WHERE name REGEXP '^..pp';
+-----+
| name |
+-----+
| copper |
+-----+
```

通过正则表达式还可以实现很多 MySQL 模式不具有的功能。例如，使用正则表达式，可以判断一个字符串中是否含有某个字符集合中的任意一个字符：

- 使用方括号（[]）来定义一个字符集合，将所有该集合中所要包括的字符都放入括号内。例如 [abc] 定义了一个集合，它可以与字符 a、b 或 c 中任一字符匹配。

- 也可以用-定义集合区间。例如[a-z]定义的模式集合可以与从 a 到 z 的任意字符匹配；[0-9]定义的集合与从 0 到 9 的任一字符匹配；而[a-zA-Z]与所有字符或者数字匹配；
- 如果要匹配的是所给集合之外的任意字符，可以在方括号内，以^开头定义集合。例如[^0-9]匹配除了数字之外的所有字符。

如下表所示，MySQL 也支持使用 POSIX 字符集定义正则表达式：

POSIX 类	匹配定义
[[:alnum:]]	字符和数字
[[:alpha:]]	字母
[[:blank:]]	空格或制表符 (tab)
[[:cntrl:]]	控制符
[[:digit:]]	数字
[[:graph:]]	图形符号 (不包括空格)
[[:lower:]]	小写字母
[[:print:]]	图形符号 (包括空格)
[[:punct:]]	标点符号
[[:space:]]	空格、制表符、换行、回车换行
[[:upper:]]	大写字母
[[:xdigit:]]	十六进制符 (0-9、a-f、A-F)

POSIX 正则表达式 POSIX 分类用在字符串分组里，所以你可以在方括号内使用它们。下面的语句用来检查 name 中是否含有十六进制字符。

```
mysql> SELECT name, name REGEXP '[[[:xdigit:]]]' FROM metal;
+-----+-----+
| name      | name REGEXP '[[[:xdigit:]]]' |
+-----+-----+
| copper    |          1 |
| gold      |          1 |
| iron      |          0 |
| lead      |          1 |
| mercury   |          1 |
| platinum  |          1 |
| silver    |          1 |
| tin       |          0 |
+-----+-----+
```

正则表达式中也可以定义选择性匹配，语法如下：

```
alternative1|alternative2|...
```

选择性匹配的正则表达式与字符集合的匹配方式类似，目标字符串只要与其中的任意一个选项匹配即可。不同的是字符集合每次只能与集合中的一个字符进行匹配，但是选择性表达式的每一个选项都可以是一个独立的正则表达式。例如下面的语句中可以查找到所有以原音字母 (vowel) 开头或者以 er 结尾的 name 字符串：

```
mysql> SELECT name FROM metal WHERE name REGEXP '^aeiou|er$';
+-----+
| name |
+-----+
| copper |
| iron   |
| silver |
+-----+
```

可以使用括号对正则表达式中的选项进行分组。如果匹配完全由数字组成，或者完全由字母组成的字符串，可以这样定义 sql 语句：

```
mysql> SELECT '0m' REGEXP '^[:digit:]+|[:alpha:]+$';
+-----+
| '0m' REGEXP '^[:digit:]+|[:alpha:]+$' |
+-----+
|          1 |
+-----+
```

执行上面的查询后，会发现结果并不正确。原因是上面的表达式中^和\$的优先级高于|，也就是两个匹配选项分别是^[:digit:] 和 [:alpha:]+\$，那么查询到的结果就是以数字开头，或者以字母结尾的字符串。如果使用括号，将匹配选项归为一组，使得^、\$作用于这个选项分组，查询匹配的结果就是我们想要的：

```
mysql> SELECT '0m' REGEXP '^(:digit:)+(:alpha:)+$';
+-----+
| '0m' REGEXP '^(:digit:)+(:alpha:)+$' |
+-----+
|          0 |
+-----+
```

与 SQL 模式不同，正则表达式匹配不要求目标串与模式串完全匹配。下面的两个正则表达式都与至少含有一个 b 的字符串匹配，但是前一个比较简单，因此更高效：

```
'abc' REGEXP 'b'  
'abc' REGEXP '^.*b.*$'
```

NULL 通过 REGEXP 和 NOT REGEXP 正则表达式匹配的结果都是 NULL：

```
mysql> SELECT NULL REGEXP '.', NULL NOT REGEXP '.';
+-----+
| NULL REGEXP '.' | NULL NOT REGEXP '.' |
+-----+
```

	NULL		NULL	
+-----+	-----+		-----+	

如果一个字符串，含有正则表达式所能匹配的任意一个子串，那么它们就是匹配的。因此要特别注意能够与空串匹配的正则表达式，否则它将与除了 NULL 以外的所有字符串匹配。例如，定义为 a^* 的正则表达式与任意字符串匹配，包括空串。如果目的是与非空且仅由 a 构成的字符串匹配，可以使用正则表达式 $a^+.$ $^+$ 表明匹配一个或多个 $^+$ 之前的任何序列。

和使用 SQL 模式进行字符串匹配一样，有些时候使用正则表达式匹配等同于子串直接比较。在匹配文本字符串时，正则表达式中使用 $^$ 和 $$$ ，等同于使用 LEFT() 和 RIGHT() 函数进行子串比较，如下例：

正则表达式	子串比较
str REGEXP '^abc'	LEFT(str ,3) = 'abc'
str REGEXP 'abc\$'	RIGHT(str ,3) = 'abc'

但是如果匹配的内容不全是文本，就很难用子串比较来代替正则表达式匹配了。例如，希望与由数字开头的非空字符串匹配，可以使用下面的正则表达式：

```
str REGEXP '^[0-9]+'
```

这样的情况就很难使用 LEFT() 函数来完成匹配了（用 LIKE 也很难实现）。



提示：正则表达式与 SQL 模式相比有一点不足，就是正则表达式只能作用于单字节字符集。对于 utf8 或者 sjis 这样的双字节字符集，正则表达式就不能发挥作用了。

5.12 模式匹配中的大小写问题

Controlling Case Sensitivity in Pattern Matching

问题

如何处理模式匹配中的大小写敏感问题。

解决方案

更改字符串属性，决定其在匹配过程中是否大小写敏感。

讨论

和其他的字符串操作一样，字符串是否是二进制的，以及非二进制字符串的 Collation 是

否支持大小写，都会决定字符串模式匹配过程是否大小写敏感。5.9 节中讨论了字符串属性对字符串比较过程的影响。

字符串默认的字符集和 Collation 分别是 latin1 和 latin1_swedish_ci，因此默认情况下字符串的模式匹配是大小写不敏感的：

```
mysql> SELECT 'a' LIKE 'A', 'a' REGEXP 'A';
+-----+-----+
| 'a' LIKE 'A' | 'a' REGEXP 'A' |
+-----+-----+
|           1 |           1 |
+-----+-----+
```

对大小写不敏感的模式匹配过程可能产生出人意料的结果：

```
mysql> SELECT 'a' REGEXP '[[lower:]]', 'a' REGEXP '[[upper:]]';
+-----+-----+
| 'a' REGEXP '[[lower:]]' | 'a' REGEXP '[[upper:]]' |
+-----+-----+
|           1 |           1 |
+-----+-----+
```

对于[:lower:]和[:upper:]，匹配的结果都为真，因为默认情况下是大小写不敏感的。

可以使用字符串比较过程中用过的方法，根据不同的情况决定匹配过程是否大小写敏感。具体的做法是：将字符串转换为二进制类型或者非二进制类型，或者对非二进制类型选择合适的 Collation。

如果希望模式匹配过程大小写敏感，可以为模式串或者目标串指定一个大小写敏感的 Collation。例如，对字符集是 latin1 的字符串，选择字符集 latin1_general_cs：

```
mysql> SET @s = 'a' COLLATE latin1_general_cs;
mysql> SELECT @s LIKE 'A', @s REGEXP 'A';
+-----+-----+
| @s LIKE 'A' | @s REGEXP 'A' |
+-----+-----+
|          0 |          0 |
+-----+-----+
```

如果使用大小写敏感的 Collation，在正则表达式中可以使用[:lower:]或[:upper:]来匹配仅有小写或者大写字母组成的字符串。下面的第二个语句中，模式匹配结果只有在目标串中仅含有大写字母时才为真：

```
mysql> SET @s = 'a', @s_cs = 'a' COLLATE latin1_general_cs;
mysql> SELECT @s REGEXP '[[upper:]]', @s_cs REGEXP '[[upper:]]';
+-----+-----+
| @s REGEXP '[[upper:]]' | @s_cs REGEXP '[[upper:]]' |
+-----+-----+
|          1 |          0 |
+-----+-----+
```

5.13 分割或者串联字符串

Breaking Apart or Combining Strings

问题

如何取得一个字符串的子串，或者串联多个字符串。

解决方案

使用 substring-extraction 函数取得子串，使用 CONCAT() 函数串联字符串。

讨论

可以使用 substring-extraction 函数分割字符串，或者取得子串。例如 LEFT() 函数、MID() 函数和 RIGHT() 函数分别可以从字符串左侧、中间或者右侧取得子串：

```
mysql> SELECT name, LEFT(name, 2), MID(name, 3, 1), RIGHT(name, 3) FROM metal;
+-----+-----+-----+-----+
| name | LEFT(name, 2) | MID(name, 3, 1) | RIGHT(name, 3) |
+-----+-----+-----+-----+
| copper | co          | p            | per          |
| gold   | go          | l            | old          |
| iron   | ir          | o            | ron          |
| lead   | le          | a            | ead          |
| mercury | me          | r            | ury          |
| platinum | pl          | a            | num          |
| silver  | si          | l            | ver          |
| tin    | ti          | n            | tin          |
+-----+-----+-----+-----+
```

LEFT() 和 RIGHT() 函数的第二个参数说明所要取得的子串长度。MID() 函数的第二个参数说明从第几位开始取出子串，第三个参数说明所要取得的子串长度。

SUBSTRING() 函数从某个位置开始向右返回整个子串。如果省略 MID() 函数的第三个参数，其执行结果与 SUBSTRING() 相同。

```
mysql> SELECT name, SUBSTRING(name, 4), MID(name, 4) FROM metal;
+-----+-----+-----+
| name | SUBSTRING(name, 4) | MID(name, 4) |
+-----+-----+-----+
| copper | per          | per          |
| gold   | d             | d             |
| iron   | n             | n             |
| lead   | d             | d             |
| mercury | cury        | cury         |
| platinum | tinum       | tinum        |
| silver  | ver          | ver          |
+-----+-----+-----+
```

```
| tin      |
```

使用 SUBSTRING_INDEX (*str*, *c*, *n*) 可以返回 *str* 中任意指定字符左侧或右侧的整个子串。该函数从 *str* 左侧查找字符 *c* 第 *n* 次出现的位置，然后返回该位置左侧的整个子串。如果 *n* 是负数，从 *str* 右侧开始查找 *c* 第 $|n|$ 次出现的位置，并返回其右侧的整个子串：

```
mysql> SELECT name,
-> SUBSTRING_INDEX(name,'r',1),
-> SUBSTRING_INDEX(name,'i',-1)
-> FROM metal;
```

name	SUBSTRING_INDEX(name,'r',1) SUBSTRING_INDEX(name,'i',-1)
copper	copper
gold	gold
iron	i
lead	lead
mercury	me
platinum	platinum
silver	silve
tin	tin

如果字符串中没有第 *n* 个字符，SUBSTRING_INDEX() 函数将返回整个字符串。另外，该函数是大小写敏感的。

Substrings 函数除了显示字符串子串之外，通常还用于字符串比较。下面的查询结果，包含所有第一个字符大于 *n* 的 name 列值：

```
mysql> SELECT name from metal WHERE LEFT(name,1) >= 'n';
+-----+
| name |
+-----+
| platinum |
| silver   |
| tin      |
+-----+
```

使用 CONCAT() 函数，可以将所有作为参数传递给它的字符串串联，之后返回：

```
mysql> SELECT CONCAT('Hello, ',USER(),' , welcome to MySQL!') AS greeting;
+-----+
| greeting          |
+-----+
| Hello, cbuser@localhost, welcome to MySQL! |
+-----+
mysql> SELECT CONCAT(name, ' ends in "d": ', IF(RIGHT(name,1)='d','YES','NO'))
-> AS 'ends in "d"?'
-> FROM metal;
+-----+
| ends in "d"?    |
+-----+
```

```
+-----+
| copper ends in "d": NO      |
| gold ends in "d": YES       |
| iron ends in "d": NO        |
| lead ends in "d": YES       |
| mercury ends in "d": NO     |
| platinum ends in "d": NO    |
| silver ends in "d": NO      |
| tin ends in "d": NO         |
+-----+
```

串联可以在“原地”改变字符串的值，例如，UPDATE 语句中，在 metal 表中 name 列的每一个值之后加上字符串 ide：

```
mysql> UPDATE metal SET name = CONCAT(name,'ide');
mysql> SELECT name FROM metal;
+-----+
| name      |
+-----+
| copperide |
| goldide   |
| ironide   |
| leadide   |
| mercuryide|
| platinumide|
| silveride |
| tinide    |
+-----+
```

上面的操作是可以撤销的（undo），只需要删除每一个值最后的三个字母即可（这需要使用 CHAR_LENGTH() 函数来获取字符串长度）：

```
mysql> UPDATE metal SET name = LEFT(name,CHAR_LENGTH(name)-3);
mysql> SELECT name FROM metal;
+-----+
| name      |
+-----+
| copper    |
| gold      |
| iron      |
| lead      |
| mercury   |
| platinum |
| silver    |
| tin       |
+-----+
```

CONCAT() 函数也可以作用于 ENUM 或者 SET 类型值，这两种值类型在数据库内被当作字符串来保存。例如，可以使用 CONCAT() 函数，将一个新值追加到一个已有的值末尾，并用“,” 分隔。考虑到已有的值可能是 NULL，在这种情况下就直接将已有的值设为新值，并且不使用“,”：

```
UPDATE tbl_name
SET set_col = IF(set_col IS NULL,val,CONCAT(set_col,',',val));
```

5.14 查询子串

Searching for Substrings

问题

如何确定一个字符串中是否含有某个子串。

解决方案

使用 LOCATE()。

讨论

LOCATE() 函数返回字符串中子字符串的第一个出现位置。如若子字符串不在字符串中，则返回值为 0。可选的第三个参数表示字符串中开始进行查找的位置。

```
mysql> SELECT name, LOCATE('in',name), LOCATE('in',name,3) FROM metal;
+-----+-----+-----+
| name | LOCATE('in',name) | LOCATE('in',name,3) |
+-----+-----+-----+
| copper | 0 | 0 |
| gold | 0 | 0 |
| iron | 0 | 0 |
| lead | 0 | 0 |
| mercury | 0 | 0 |
| platinum | 5 | 5 |
| silver | 0 | 0 |
| tin | 2 | 0 |
+-----+-----+-----+
```

传递给 LOCATE() 的参数的 Collation，决定了该函数是否是大小写敏感的。5.6 节和 5.9 节中已经讨论了如何改变字符串的 Collation，来决定查询过程是否是大小写敏感的。

5.15 使用 FULLTEXT 查询

Using FULLTEXT Searches

问题

如何对大量文本进行查询。

解决方案

使用 FULLTEXT 索引。

讨论

在数据量较大时，使用模式匹配进行查询也能得到结果，但是效率会变得很低。也可以使用模式匹配从多个列中查询相同的字符串，但是效率一样不是很高：

```
SELECT * from tbl_name  
WHERE col1 LIKE 'pat' OR col2 LIKE 'pat' OR col3 LIKE 'pat' ...
```

对大量文本或者多列，可以使用 FULLTEXT 查询替代模式匹配，从而获得较高的效率。使用 FULLTEXT 查询，首先要给表加上 FULLTEXT 索引，然后使用 MATCH 操作符查询作为索引的列。FULLTEXT 索引可以作用于 MyISAM 表中的非二进制类型字符串 (CHAR、VARCHAR 或者 TEXT)。

通过对一个足够大的文本进行搜索来说明 FULLTEXT 查询。如果你没有样本数据，因特网上有一些可免费下载的电子书可供使用。对于这里的例子，我选择的是 King James 版本的圣经 (KJV) 的全文，它的数据量够大并且具有书、章节和诗篇良好组织的特性。由于它的数据量太大，这个样本数据没有放在随机光盘中，但是在 MySQL COOKBOOK 的站点（见附录 A）作为 mcb-kjv 单独可以使用。在 mcb-kjv 中包含了一个名为 kjv.txt 的文件，其中包含了诗篇记录。一些记录的样板如下：

```
O  Genesis 1   1   1   In the beginning God created the heaven and the earth.  
O  Exodus   2   20  13  Thou shalt not kill.  
N  Luke     42  17  32  Remember Lot's wife.
```

每条记录都包含一下信息：

- Book 域，值为 O 或者 N，分别表示旧约或者新约；
- Book 名和编号（1 到 66）；
- 章节和段落编号；
- 正文。

新建一个名为 kjv 的表，用来保存所有记录：

```
CREATE TABLE kjv  
(  
    bsect ENUM('O','N') NOT NULL,          # 书籍域 (旧约或新约)  
    bname VARCHAR(20) NOT NULL,            # 书名  
    bnum TINYINT UNSIGNED NOT NULL,        # 书籍数目  
    cnum TINYINT UNSIGNED NOT NULL,        # 章数  
    vnum TINYINT UNSIGNED NOT NULL,        # 节数  
    vtext TEXT NOT NULL                   # 书节文本  
) ENGINE = MyISAM;
```

将 kjv.txt 导入表中：

```
mysql> LOAD DATA LOCAL INFILE 'kjv.txt' INTO TABLE kjv;
```

你将注意到 kjv 表中同时包含了书名列（Genesis、Exodus...）和书编号列（1、2、...）。这些名字和数字编号有固定的对应关系，并且一个可以从另外一个中获得——这种冗余说明这张表不是一种常见形式。为了消除冗余，可以在（kjv 表）只保存书编号（这比保存书名占用更少空间），然后当查询结果中需要书名的时候将编号和一张将每一个书编号和书名做匹配的较小的表做联合。但是作者想避免使用联合，从而，表中（kjv 表）包含有书名使得查询结果的解释更容易，包含有书编号使得对书进行排序更容易。

导入表数据之后，通过加上 FULLTEXT 索引来准备使用 FULLTEXT 检索。这可以使用一个 ALTER TABLE 语句实现：

```
mysql> ALTER TABLE kjv ADD FULLTEXT (vtext);
```

在初始的 CREATE TABLE 语句中包含索引定义是可能的，但是首先生成一张未索引的表，然后在导入表数据之后通过 ALTER TABLE 再加上索引定义，要比直接将表数据导入到一个具有所有定义的表更快。

为了使用索引实现一个查询，使用 MATCH() 函数来指定索引列并且使用 AGAINST() 函数来定义要查询的文本。例如，你可能在想，“Mizrim 这个名字一共出现了多少次？”为了回答这个问题，使用如下语句来查询 vtext 列：

```
mysql> SELECT COUNT(*) from kjv WHERE MATCH(vtext) AGAINST('Mizraim');
+-----+
| COUNT(*) |
+-----+
|        4 |
+-----+
```

为了查明这都是哪些诗节，可以查询你想看到的列（这里的例子使用 \G，使查询结果更合适的在每一页显示）：

```
mysql> SELECT bname, cnum, vnum, vtext
    -> FROM kjv WHERE MATCH(vtext) AGAINST('Mizraim')\G
***** 1. row *****
bname: Genesis
cnum: 10
vnum: 6
vtext: And the sons of Ham; Cush, and Mizraim, and Phut, and Canaan.
***** 2. row *****
bname: Genesis
cnum: 10
vnum: 13
vtext: And Mizraim begat Ludim, and Anamim, and Lehabim, and Naphtuhim,
***** 3. row *****
bname: 1 Chronicles
cnum: 1
vnum: 8
vtext: The sons of Ham; Cush, and Mizraim, Put, and Canaan.
***** 4. row *****
```

```
byname: 1 Chronicles
cnum: 1
vnum: 11
vtext: And Mizraim begat Ludim, and Anamim, and Lehabim, and Naphtuhim,
```

查询结果以书、章节和诗节的形式按照特定的顺序显示，但是这只是一个意外。默认情况下，`FULLTEXT` 查询计算出一个合适的（比较）队列并用来进行排序。为了确保查询结果按照你的意图进行排序，显式的加上一个 `ORDER BY` 子句：

```
SELECT bname, cnum, vnum, vtext
FROM kjv WHERE MATCH(vtext) AGAINST('search string')
ORDER BY bnum, cnum, vnum;
```

如果你想看到排序队列，可以在输出的列中重复使用 `MATCH()` 和 `AGAINST()`。

你可以包含一些附加条件来进一步精确查询。下面的语句完成了更加精确的查询来找出名字 Abraham 在 KJV、New Testament、Book of Hebrews 和 Hebrews 的第 11 章中出现的频率：

```
mysql> SELECT COUNT(*) from kjv WHERE MATCH(vtext) AGAINST('Abraham');
+-----+
| COUNT(*) |
+-----+
|      216 |
+-----+
mysql> SELECT COUNT(*) from kjv
    -> WHERE MATCH(vtext) AGAINST('Abraham')
    -> AND bsect = 'N';
+-----+
| COUNT(*) |
+-----+
|       66 |
+-----+
mysql> SELECT COUNT(*) from kjv
    -> WHERE MATCH(vtext) AGAINST('Abraham')
    -> AND bname = 'Hebrews';
+-----+
| COUNT(*) |
+-----+
|       10 |
+-----+
mysql> SELECT COUNT(*) from kjv
    -> WHERE MATCH(vtext) AGAINST('Abraham')
    -> AND bname = 'Hebrews' AND cnum = 11;
+-----+
| COUNT(*) |
+-----+
|        2 |
+-----+
```

如果你期望在标准查询中频繁的包含其他一些非 FULLTEXT 的列，你可以通过在这些列上加上规则的索引来提高相关查询性能。例如，为了给 book、chapter 和 verse number 列加上索引，可以这样做：

```
mysql> ALTER TABLE kjv ADD INDEX (bnum), ADD INDEX (cnum), ADD INDEX (vnum);
```

在 FULLTEXT 查询中的查询字符串可以包含多个单词，并且你可能会以为使用额外的词会使查询更精确。但是实际上查询结果范围更大了，因为 FULLTEXT 查询返回的每一行只需要包含查询字符串中的任意一个单词，查询过程对每一个单词进行 OR 查询。可以通过下面的语句来说明这一情况，当加上更多的查询词语时得到更大的诗节统计数字：

```
mysql> SELECT COUNT(*) from kjv
   -> WHERE MATCH(vtext) AGAINST('Abraham');
+-----+
| COUNT(*) |
+-----+
|      216 |
+-----+
mysql> SELECT COUNT(*) from kjv
   -> WHERE MATCH(vtext) AGAINST('Abraham Sarah');
+-----+
| COUNT(*) |
+-----+
|      230 |
+-----+
mysql> SELECT COUNT(*) from kjv
   -> WHERE MATCH(vtext) AGAINST('Abraham Sarah Ishmael Isaac');
+-----+
| COUNT(*) |
+-----+
|      317 |
+-----+
```

如果为了实现查询字符串中每一个单词都出现的查询，请参考 5.17 节。

如果你想使用 FULLTEXT 同时对多列进行查询，可以在建立索引时指定所有需要被查询的列：

```
ALTER TABLE tbl_name ADD FULLTEXT (col1, col2, col3);
```

为了使用索引进行查询，在 MATCH() 参数列表中指定所有列：

```
SELECT ... FROM tbl_name
WHERE MATCH(col1, col2, col3) AGAINST('search string');
```

对于你想要联合查询的每一列，你都需要一个对应的 FULLTEXT 索引。

参考

FULLTEXT 查询提供了一种快速简便的配置出一个基本搜索引擎的方法。使用该功能的一种方式是提供一个基于网络的（数据库表）索引列访问接口。本书的网站（参见附录 A）包含了一个简单的基于网络的 KJV 查询网页来说明这一应用。你可以以此为基础，来实现你自己的搜索引擎对不同的文本进行的处理。

5.16 用短语来进行 FULLTEXT 查询

Using a FULLTEXT Search with Short Words

问题

FULLTEXT 搜索对短的查询词语无效（不能正确进行查询）。

解决方案

改变索引引擎的最短词语长度参数。

讨论

在类似 KJV 这样的文本中某些词语特别重要，例如 “God” 和 “sin.”。然而，如果你在 kjav 表中针对这几个单词进行 FULLTEXT 查询，你将会发现一个奇怪的现象——这两个单词都从整本书中消失了。

```
mysql> SELECT COUNT(*) FROM kjav WHERE MATCH(vtext) AGAINST('God');
+-----+
| COUNT(*) |
+-----+
|          0 |
+-----+
mysql> SELECT COUNT(*) FROM kjav WHERE MATCH(vtext) AGAINST('sin');
+-----+
| COUNT(*) |
+-----+
|          0 |
+-----+
```

索引引擎的一个特性就是它会忽略那些“太常见”的词（换句话说，就是一半以上的行中都出现的词），这就从索引中除去了一些词，例如 “the” 或 “and”。但在这里并不是这样的。你可以通过统计所有的行数，并且使用 SQL 模式匹配来统计包含这两个单词的行数来验证（注 1）：

```
mysql> SELECT COUNT(*) AS 'total verses',
-> COUNT(IF(vtext LIKE '%God%',1,NULL)) AS 'verses containing "God"',
-> COUNT(IF(vtext LIKE '%sin%',1,NULL)) AS 'verses containing "sin"';
```

注 1：使用 COUNT() 函数根据同一组变量值来生成多个不同的计数值。这将在第 8.1 节中进行讨论。

```
-> FROM kjv;
+-----+
| total verses | verses containing "God" | verses containing "sin" |
+-----+
|      31102 |          4118 |        1292 |
+-----+
```

没有一个单词出现在一半以上的诗节中，这显然说明出现的频率不是造成对它们进行 FULLTEXT 搜索失败的原因。真正的原因是，默认情况下索引引擎并不会将少于 4 个字母的单词包括在内。单词最短长度是一个可配置参数，它可以通过设置服务器变量 `ft_min_word_len` 来改变。例如，为了使索引引擎包括 3 个字母长度的单词，可以在/etc/my.cnf 文件中 [mysqld] 组内添加一行（或者在其他你保存服务器设置的选项文件中）：

```
[mysqld]
ft_min_word_len=3
```

作了这一改动之后，重启服务器。然后，重建 FULLTEXT 索引来启用新的设置（注 2）：

```
mysql> REPAIR TABLE kjv QUICK;
```

最后，试用一下新的索引来验证它包含了更短的单词：

```
mysql> SELECT COUNT(*) FROM kjv WHERE MATCH(vtext) AGAINST('God');
+-----+
| COUNT(*) |
+-----+
|      3878 |
+-----+
mysql> SELECT COUNT(*) FROM kjv WHERE MATCH(vtext) AGAINST('sin');
+-----+
| COUNT(*) |
+-----+
|      389 |
+-----+
```

这样就会好多了。

但是为什么 `MATCH()` 函数查询找到了 3 878 行和 389 行，而之前的 `LIKE` 语句查询找到了 4 118 行和 1 292 行呢？这是因为 `LIKE` 语句对字符串子串进行模式匹配，而 `MATCH()` 语句进行的 FULLTEXT 搜索对整个单词进行匹配。

5.17 要求或禁止 FULLTEXT 搜索单词

Requiring or Excluding FULLTEXT Search Words

问题

你想要特别地要求或者禁止一个 FULLTEXT 搜索中的单词。

注 2：如果你修改了 `ft_min_word_len` 值，你必须也使用 `REPAIR TABLE` 语句来重建所有含有 FULLTEXT 索引的表的索引。

解决方案

使用 Boolean 模式搜索。

讨论

通常 FULLTEXT 查询返回含有查询字符串中任意单词的行，甚至允许不包括某些单词。例如，下面的语句查询含有名字 David 或者 Goliath 的行：

```
mysql> SELECT COUNT(*) FROM kjv
      -> WHERE MATCH(vtext) AGAINST('David Goliath');
+-----+
| COUNT(*) |
+-----+
|      934 |
+-----+
```

如果你只想要同时含有这两个名字的行，那么这个结果是不合要求的。解决方法之一是重写查询语句，分别对每一个单词进行查询，并且使用 AND 联合查询条件：

```
mysql> SELECT COUNT(*) FROM kjv
      -> WHERE MATCH(vtext) AGAINST('David')
      -> AND MATCH(vtext) AGAINST('Goliath');
+-----+
| COUNT(*) |
+-----+
|        2 |
+-----+
```

另一种要求多个单词的方法是使用 Boolean 模式查询。为此，在查询语句中的每一个单词前加一个+，同时在查询字符串之后加上 IN BOOLEAN MODE：

```
mysql> SELECT COUNT(*) FROM kjv
      -> WHERE MATCH(vtext) AGAINST('+David +Goliath' IN BOOLEAN MODE)
+-----+
| COUNT(*) |
+-----+
|        2 |
+-----+
```

Boolean 模式查询也允许你在查询中排除某些单词。方法是只需要在每一个要排除的单词之前加一个-。下面的查询是从 kjv 中查询含有名字 David 但是不含有 Goliath 的行，反之亦然：

```
mysql> SELECT COUNT(*) FROM kjv
      -> WHERE MATCH(vtext) AGAINST('+David -Goliath' IN BOOLEAN MODE)
+-----+
| COUNT(*) |
+-----+
|      928 |
+-----+
mysql> SELECT COUNT(*) FROM kjv
```

```
-> WHERE MATCH(vtext) AGAINST('-David +Goliath' IN BOOLEAN MODE)
+-----+
| COUNT(*) |
+-----+
|      4   |
+-----+
```

另外一个 Boolean 查询中有用的特殊字符是*。当加到一个查询单词上时，它充当一个通配符。下面的语句查询的行不仅可以包含 whirl，也可以含有 whirls、whirleth 和 whirlwind 等单词：

```
mysql> SELECT COUNT(*) FROM kjv
      -> WHERE MATCH(vtext) AGAINST('whirl*' IN BOOLEAN MODE);
+-----+
| COUNT(*) |
+-----+
|      28   |
+-----+
```

关于 Boolean FULLTEXT 查询的操作符列表，请参考《MySQL Reference Manual》。

5.18 用 FULLTEXT 索引来执行词组查询

Performing Phrase Searches with a FULLTEXT Index

问题

你想针对一个词组进行一次 FULLTEXT 搜索，也就是说每一个单词互相临近并且按照特定顺序出现。

解决方案

使用 FULLTEXT 词组查询功能。

讨论

你不能使用简单的 FULLTEXT 查询来查找含有特定词组的行：

```
mysql> SELECT COUNT(*) FROM kjv
      -> WHERE MATCH(vtext) AGAINST('still small voice');
+-----+
| COUNT(*) |
+-----+
|      548  |
+-----+
```

这个查询返回了一个结果，但这并不是你想要的。一个 FULLTEXT 搜索针对每一个单独出现的单词计算出一个相关度级别，不管这个单词在 vtext 列的什么地方出现，只要任意一个单词出现了，级别就不为 0。因而，这样的语句通常会找到过多的行。

FULLTEXT 搜索在 Boolean 模式中支持词组搜索。在查询字符串中用双引号把词组括起来，即可使用该功能：

```
mysql> SELECT COUNT(*) FROM kjv
-> WHERE MATCH(vtext) AGAINST('"still small voice" IN BOOLEAN MODE');
+-----+
| COUNT(*) |
+-----+
|       1   |
+-----+
```

当和词组中相同的几个单词按照给定的顺序出现在一列中时，词组匹配成功。

使用日期和时间

Working with Dates and Times

6.0 引言

Introduction

MySQL 提供了多种变量类型用于保存时间和日期值，并提供了一些函数对这一类变量进行操作。MySQL 用特殊的格式来存储日期和时间，因而弄懂这些以免在操作时间数据时产生令人奇怪的结果很重要。本章的讨论包括以下内容：

如何选择时间和日期变量类型

MySQL 为表的数据列提供了多种时间和日期变量类型。了解各种类型的属性，才能作出合适的选择。

如何显示日期和时间

MySQL 提供了默认的显示格式，但是用户也可以使用合适的函数来定制输出格式。

更改客户端时区

服务器仅按照客户端的时区动态决定 `TIMESTAMP` 类型值，而与服务器时区无关。因此，为了得到正确的 `TIMESTAMP` 值，处于不同时区的客户端必须根据需要设定时区值。

获取当前日期或时间

MySQL 提供函数获取当前日期或时间。对于需要知道这些值或者需要知道与之相关的其他变量值的应用程序来说，这些函数都是非常有用的。

使用 `TIMESTAMP` 记录每一行数据的修改时间

`TIMESTAMP` 数据类型具有一些特殊属性，使其能够自动记录每一行信息的创建和修改时间。

分解日期和时间值

用户可以从日期和时间值中分解出自己需要的部分值，例如一个日期值中的月份部分，或者一个时间值中的小时部分。

合成日期或时间值

和分解出日期和时间中每一部分值相反，也可以通过组合每一个部分值，合成一个日期或者时间值。

日期和时间值的转换以及基本单位

在有些有关时间和日期的运算中，如果使用多少天或者多少秒来代表日期或者时间值，也许会比用这些值本身更容易处理。MySQL 提供了一些函数，用于日期和时间值以及一些基本单位（例如，天、秒）之间的转换。

日期和时间运算

可以对时间和日期变量进行加减运算来得到新的时间或者日期值，或者计算两个值之间的时间间隔。在应用程序中和时间和日期相关的运算包括计算年龄、日期和时间间隔等。

将时间和日期值作为控制条件来查询日期

上面提到的时间和日期运算的结果也可以用在 WHERE 子句中，用来指定查询过程中的日期和时间限制。

这一章只介绍了一部分用于操作时间和日期值的函数。在《MySQL Reference Manual》中可以查看到所有可用的函数。因为有很多类似的函数，也就是说对于同一个表达式，可以有多种书写方式。笔者使用不同的函数得到相同的结果，本章所列的很多问题也可以用其他方法来解决，建议读者尝试找到其他的方法。读者也许能找到更加有效，或者可读性更高的解决方法。

本章所用的脚本可以在随书发布的 `veclpes` 源码包中的 `dates` 目录下找到。在 `tables` 目录下可以找到 `sql` 脚本来创建本章中要用到的表。

6.1 选择合适的日期或者时间变量类型

Choosing a Temporal Data Type

问题

如何选择合适的变量类型用于保存日期和时间值。

解决方案

根据所要保存的信息的特征以及使用方式来选择变量类型。需要考虑的问题如下：

- 只需要单独的时间或者日期，还是需要同时保存日期和时间。
- 所要保存的值的起止范围是什么。
- 是否需要数据库自动将列值初始化为当前日期和时间。

讨论

MySQL 提供了 DATE 和 TIME 类型分别用来保存日期和时间值，而 DATETIME 和 TIMESTAMP 类型用来同时保存日期和时间值。这些值类型有如下特征：

- 使用 *CCYY-MM-DD* 格式的字符穿来处理 DATE 类型值，其中 *CC*、*YY*、*MM* 和 *DD* 分别代表世纪、世纪的年份、月和日。DATE 值的有效范围从 1000-01-01 到 9999-12-31。
- 使用格式 *hh:mm:ss* 字符穿来保存 TIME 类型变量值，其中 *hh*、*mm*、*ss* 分别代表小时、分钟和秒。时间值通常被当作是某一天中的某个时间，但实际上在 MySQL 将时间作为一个连续的值，可以从过去的某一个时间点开始计算。因此，TIME 类型的值可能大于 23:59:59，也可能是一个负数（实际上，TIME 类型的取值范围是 -838:59:59 到 838:59:59）。
- DATETIME 和 TIMESTAMP 类型使用 *CCYY-MM-DD hh:mm:ss* 格式的字符穿来保存时间和日期的一个组合值。（MySQL4.1 之前的版本使用 *CCYYMMDDhhmmss* 格式的数值来保存 TIMESTAMP 值，因此基于这样的数据保存格式的应用程序需要升级，才能使用 MySQL4.1 以后的数据库。）

在许多方面，你可以以类似的方式处理 DATETIME 和 TIMESTAMP 数据类型，但要当心这些不同之处：

—DATETIME 类型变量的取值范围为 1000-01-01 00:00:00 到 9999-12-31 23:59:59，而 TIMESTAMP 仅为 1970-01-01 00:00:00 到 2037-12-31 23:59:59。

—TIMESTAMP 类型具有自动初始化和自动更新的特性，这些特性将在第 6.5 节中做详细讨论。

—当客户端向数据库中插入一个 TIMESTAMP 类型值时，服务器将该值从客户端所属时区转换到 UTC，然后将该值保存为一个 UTC 时间值。当一个客户端从服务器查询这个 TIMESTAMP 值时，服务器执行相反的操作，将 UTC 值转换为客户端时区时间值。处在不同时区的客户端通过更改其连接配置，可以使服务器正确的执行 TIMESTAMP 值的时区转换（第 6.3 节）。

本章大部分例子都使用下面的表，其中的列分别是 TIME、DATE、DATETIME 和 TIMESTAMP 类型。（time_val 表中的两列用于计算时间间隔）。

```
mysql> SELECT t1, t2 FROM time_val;
+-----+-----+
| t1   | t2   |
+-----+-----+
| 15:00:00 | 15:00:00 |
| 05:01:30 | 02:30:20 |
| 12:30:20 | 17:30:45 |
+-----+-----+
mysql> SELECT d FROM date_val;
+-----+
| d   |
+-----+
```

```
+-----+
| 1864-02-28 |
| 1900-01-15 |
| 1987-03-05 |
| 1999-12-31 |
| 2000-06-04 |
+-----+
mysql> SELECT dt FROM datetime_val;
+-----+
| dt          |
+-----+
| 1970-01-01 00:00:00 |
| 1987-03-05 12:30:15 |
| 1999-12-31 09:00:00 |
| 2000-06-04 15:45:30 |
+-----+
mysql> SELECT ts FROM timestamp_val;
+-----+
| ts          |
+-----+
| 1970-01-01 00:00:00 |
| 1987-03-05 12:30:15 |
| 1999-12-31 09:00:00 |
| 2000-06-04 15:45:30 |
+-----+
```

建议读者在阅读后面的章节之前建立 `time_val`、`date_val`、`datetime_val` 和 `timestamp_val` 四张表（可以使用 `veclib` 发行包中 `tables` 目录下提供的脚本）。

6.2 修改 MySQL 中的日期格式

Changing MySQL's Date Format

问题

你想改变 MySQL 用来表示日期值的 ISO 格式。

解决方案

实际上没有什么办法。但是，在保存日期类型值时可以将非 ISO 格式的输入重写为 ISO 格式，或者在显示日期类型值时，使用 `DATE_FORMAT()` 函数将其从 ISO 格式转换为其他格式。

讨论

MySQL 保存日期类型值时使用的 `CCYY-MM-DD` 格式遵循 ISO 8601 标准。由于在该格式中，年、月、日值都有固定长度，并且从左到右出现在串值中，因此日期类型值自然的按照时间顺序排序。在第 7 章和第 8 章中将详细讨论日期类型值的排序和分组。

ISO 格式虽然较常见，但是并不被所有的数据库系统支持，因此在不同的数据库系统之间转移数据时可能造成数据错误。此外，用户常常按照自己认为正确的格式来处理日期数据，例如 *MM/DD/YY* 或者 *DD-MM-CCYY* 格式。如果用户期望的日期格式和 MySQL 实际使用的日期格式之间匹配不当，也会造成一些错误。

MySQL 的新用户常常有这样的疑问，如何指定 MySQL 服务器使用给定的格式处理日期数据？例如使用格式 *MM/DD/CCYY*。实际上这是不可能的，用户真正需要考虑的问题是，如何在 MySQL 使用的日期格式和用户期望的日期格式之间进行转换。MySQL 始终只能使用 ISO 格式，包括日期值在数据库中的存储以及数据库查询结果中的日期值都是使用 ISO 格式。

- 对于数据存储，如果要保存的日期值不使用 ISO 格式，那么首先要重写这个日期值。如果不重写，也可以将其保存为字符串类型（例如，保存到一个 CHAR 类型列中）。但是那样做，以后就不能使用日期处理函数来对其进行操作了。

本书在第 10 章中讨论了日期值的重写，以及如何验证日期值的有效性。在有些情况下，如果用户使用的日期值格式接近于 ISO 格式，也可以不重写用户日期数据，而直接保存，让数据库完成转换工作。例如，当用户把数据保存进 DATE 类型列时，MySQL 将字符串值 *87-1-7* 和 *1987-1-7* 以及数值 *870107* 和 *19870107* 保存为 *1987-01-07*

- 在显示日期数据时，可以使用 `DATE_FORMAT()` 函数，将日期值按照非 ISO 格式重写。`DATE_FORMAT()` 函数提供了很灵活的方式来改变日期值的显示格式（详见后面）。用户也可以使用函数（例如 `YEAR()`）分解日期值，然后再按照用户格式显示。第 10 章中有更多相关讨论。

向数据库中录入一个日期值时，重写非 ISO 格式数据的一种方法是使用 `STR_TO_DATE()` 函数。该函数接受两个参数，一个代表日期值的字符串，和一个对该日期字符串进行说明的格式化串。其中格式化串由%c 格式的字符序列组成，其中 c 说明了日期字符串中对应部分的含义。例如，%Y、%M 和%d 分别表示 4 位数字的年份、月名、以及用两位数字表示一个月的第几天。可以使用如下语句，向数据库一个数据类型为 DATE 的列中插入值 May 13, 2007：

```
mysql> INSERT INTO t (d) VALUES(STR_TO_DATE('May 13, 2007', '%M %d, %Y'));
mysql> SELECT d FROM t;
+-----+
| d   |
+-----+
| 2007-05-13 |
+-----+
```

在显示一个日期值时，如果没有特别指定，MySQL 按照 ISO 格式 (*CCYY-MM-DD*) 显示日期值。如果不希望按照 MySQL 的默认格式输出时间和日期值，可以使用 `DATE_FORMAT()` 或者 `TIME_FORMAT()` 函数按照用户期望的格式重写日期或者时间值。如果所期望的格式使用函数重写仍然不能实现，则需要使用存储函数（过程）。

使用 DATE_FORMAT() 函数，可以按照用户期望的格式重写日期值。该函数接收两个参数，一个 DATE、DATETIME 或者 TIMESTAMP 类型值，和一个定义输出格式的格式化串。这个格式化串使用与 STR_TO_DATE() 函数中的格式化串一样的定义方式和特殊字符。下面的语句显示了默认情况下，以及经过 DATE_FORMAT() 函数按照用户格式重写之后的 data_val 表中 d 列值：

```
mysql> SELECT d, DATE_FORMAT(d, '%M %d, %Y') FROM date_val;
+-----+-----+
| d      | DATE_FORMAT(d, '%M %d, %Y') |
+-----+-----+
| 1864-02-28 | February 28, 1864 |
| 1900-01-15 | January 15, 1900 |
| 1987-03-05 | March 05, 1987 |
| 1999-12-31 | December 31, 1999 |
| 2000-06-04 | June 04, 2000 |
+-----+-----+
```

上面 sql 语句中的表达式 DATE_FORMAT(d, '%M %d, %Y')，会在查询结果的表头中显示一个很长的列名，影响了结果的可读性。为此，可以通过使用别名，使查询结果更具有可读性。

```
mysql> SELECT d, DATE_FORMAT(d, '%M %d, %Y') AS date FROM date_val;
+-----+-----+
| d      | date      |
+-----+-----+
| 1864-02-28 | February 28, 1864 |
| 1900-01-15 | January 15, 1900 |
| 1987-03-05 | March 05, 1987 |
| 1999-12-31 | December 31, 1999 |
| 2000-06-04 | June 04, 2000 |
+-----+-----+
```

DATE_FORMAT()、TIME_FORMAT() 和 STR_TO_DATE() 三个函数都接受格式化串作为参数，关于定义格式化串的所有特殊字符及其含义，都可以在《MySQL Reference Manual》中找到。这里列出了一部分使用频率较高的格式化串定义：

格式化字符	含义
%Y	年份，数字形式，4 位数
%y	年份，数字形式，2 位数
%M	完整的月份名称 (January-December)
%b	月份名称的前三个字母 (Jan-Dec)
%m	月份，数字形式 (01..12)
%c	月份，数字形式 (1..12)
%d	该月日期，数字形式 (01..31)
%e	该月日期，数字形式 (1..31)
%W	工作日名称 (Sunday..Saturday)
%r	时间，12 小时制，以 AM 或 PM 结尾
%T	时间，24 小时制

格式化字符	含义
%H	小时，数字形式，2位数 (00..23)
%i	分钟，数字形式，2位数 (00..59)
%s	秒，数字形式，2位数 (00..59)
%%	'%'文字字符

对于 DATE_FORMAT() 函数，只有传递给它的参数同时具有时间和日期值（例如参数为 DATETIME 或者 TIMESTAMP 类型变量）时，与时间相关的格式化串才有效。下面的语句示范了如何使用格式化串来重写并输出一个 DATETIME 类型值：

```
mysql> SELECT dt,
    -> DATE_FORMAT(dt, '%c/%e/%Y %r') AS format1,
    -> DATE_FORMAT(dt, '%M %e, %Y %T') AS format2
    -> FROM datetime_val;
+-----+-----+-----+
| dt      | format1          | format2          |
+-----+-----+-----+
| 1970-01-01 00:00:00 | 1/1/70 12:00:00 AM | January 1, 1970 00:00:00 |
| 1987-03-05 12:30:15 | 3/5/87 12:30:15 PM | March 5, 1987 12:30:15 |
| 1999-12-31 09:00:00 | 12/31/99 09:00:00 AM | December 31, 1999 09:00:00 |
| 2000-06-04 15:45:30 | 6/4/00 03:45:30 PM | June 4, 2000 15:45:30 |
+-----+-----+-----+
```

TIME_FORMAT() 函数与 DATE_FORMAT() 类似，但是 TIME_FORMAT() 函数参数中的格式化字符串只能使用时间相关的格式化字符串定义。可以用来处理 TIME，DATETIME 或者 TIMESTAMP 类型值。

```
mysql> SELECT dt,
    -> TIME_FORMAT(dt, '%r') AS '12-hour time',
    -> TIME_FORMAT(dt, '%T') AS '24-hour time'
    -> FROM datetime_val;
+-----+-----+-----+
| dt      | 12-hour time | 24-hour time |
+-----+-----+-----+
| 1970-01-01 00:00:00 | 12:00:00 AM | 00:00:00 |
| 1987-03-05 12:30:15 | 12:30:15 PM | 12:30:15 |
| 1999-12-31 09:00:00 | 09:00:00 AM | 09:00:00 |
| 2000-06-04 15:45:30 | 03:45:30 PM | 15:45:30 |
+-----+-----+-----+
```

如果使用 DATE_FORMAT() 或者 TIME_FORMAT() 函数不能得到用户希望的结果，那么就需要使用存储函数来实现了。下面的函数将一个 24 小时进制的 TIME 值转化为 12 小时制，并以 am 或 pm 作为后缀，而不是以函数返回结果中的 AM 或者 PM 作为后缀。该函数使用函数 TIME_FORMAT() 完成大部分的转换工作，其中格式化串 %r 产生一个以 AM 或者 PM 为后缀的 12 小时制时间值，然后使用 am 或者 pm 替换后缀：

```
CREATE FUNCTION time_ampm (t TIME)
RETURNS VARCHAR(13) # mm:dd:ss {a.m.|p.m.} 格式
BEGIN
DECLARE ampm CHAR(4);
IF TIME_TO_SEC(t) < 12*60*60 THEN
```

```
    SET ampm = 'a.m.';
ELSE
    SET ampm = 'p.m.';
END IF;
RETURN CONCAT(LEFT(TIME_FORMAT(t, '%r'), 9), ampm);
END;
```

像这样来使用上面定义的存储函数：

```
mysql> SELECT t1, time_ampm(t1) FROM time_val;
+-----+-----+
| t1      | time_ampm(t1) |
+-----+-----+
| 15:00:00 | 03:00:00 p.m. |
| 05:01:30 | 05:01:30 a.m. |
| 12:30:20 | 12:30:20 p.m. |
+-----+-----+
```

关于存储函数定义的方法请参考第 16 章。

6.3 设置客户端时区

Setting the Client Time Zone

问题

如果客户端与服务器处在不同的时区，那么当客户端在服务器上保存一个 `TIMESTAMP` 类型值时，实际保存的结果不是正确的 UTC 时间。

解决方案

当客户端连接到服务器时，通过设置系统变量 `time_zone` 来改变客户端时区值。

讨论

MySQL 根据各个客户端连接时区的不同，对 `TIMESTAMP` 类型值进行相应的处理。当一个客户端在服务器中插入一个 `TIMESTAMP` 值时，服务器将该时间值从客户端连接时区转换到 UTC，并在数据库中保存该 UTC 时间值。（在数据库内部，`TIMESTAMP` 值保存为从 1970-01-01 00:00:00 UTC 开始的秒数。）当客户端从服务器查询一个 `TIMESTAMP` 类型值时，服务器执行一个相反的操作，将该值从 UTC 转换到客户端连接时区。

一个客户端连接的默认时区为所连接到的服务器的时区。服务器在启动时检查所在的操作系统环境，并设置相应配置（如果不希望服务器使用所在操作系统时区，那么可以使用 `--default-time-zone` 选项来启动服务器）。如果所有的客户端都和服务器在相同的时区，那么并不需要什么特别的操作来保证 `TIMESTAMP` 类型值的正确性。但对于一个与服务器不在同一时区的客户端，当它向服务器插入 `TIMESTAMP` 类型值时，实际保存的 UTC 时间就是错误的。

假设服务器和客户端 A 在同一时区，客户端 A 执行了下面的操作：

```
mysql> CREATE TABLE t (ts TIMESTAMP);
mysql> INSERT INTO t (ts) VALUES('2006-06-01 12:30:00');
mysql> SELECT ts FROM t;
+-----+
| ts |
+-----+
| 2006-06-01 12:30:00 |
+-----+
```

这里，客户端 A 从数据库查询到的 TIMESTAMP 值，与它向数据库插入的值相同。另外一个客户端 B 从服务器也可以得到相同的查询结果，但是如果 B 与 A 在不同的时区，那么 B 得到的时间值对于它所处的时区来说就是错误的。相反的，如果 B 在服务器保存了一个 TIMESTAMP 值，这个值如果返回给 A，那么对于 A 的时区而言，也是错误的。

为了避免这样的错误发生，就需要根据客户端时区来转换 TIMESTAMP 类型值，因此需要客户端显式的设置时区。通过设置客户端系统变量 time_zone 可以设置一个客户端连接的时区。假设，服务器有一个先于 UTC 6 小时的全局时区，每一个客户端都将自己和服务器的连接时区初始化为服务器的全局时区：

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
+-----+-----+
| @@global.time_zone | @@session.time_zone |
+-----+-----+
| +06:00           | +06:00           |
+-----+-----+
```

前面提到过的客户端 B 从服务器查询到的 TIMESTAMP 类型值与 A 查询到的一样：

```
mysql> SELECT ts FROM t;
+-----+
| ts |
+-----+
| 2006-06-01 12:30:00 |
+-----+
```

如果 B 只先于 UTC 4 个小时，可以在连接到服务器后这样设置时区：

```
mysql> SET SESSION time_zone = '+04:00';
mysql> SELECT @@global.time_zone, @@session.time_zone;
+-----+-----+
| @@global.time_zone | @@session.time_zone |
+-----+-----+
| +06:00           | +04:00           |
+-----+-----+
```

之后，客户端 B 从服务器查询到的 TIMESTAMP 值，就会从 UTC 自动转换到其所属时区：

```
mysql> SELECT ts FROM t;
+-----+
| ts      |
+-----+
| 2006-06-01 10:30:00 |
+-----+
```

客户端时区也会影响获取当前时间和日期的函数的返回值（见 6.4 节）。

参考

使用 CONVERT_TZ() 函数，可以将一个日期或者时间值在各个时区之间进行转换（详见 6.12 节）。

6.4 获取当前日期或时间

Determining the Current Date or Time

问题

如何获取当前日期或者时间。

解决方案

使用 CURDATE()、CURTIME() 或者 NOW() 函数可以取得该连接所属时区的当前日期和时间。使用 UTC_DATE()、UTC_TIME() 或 UTC_TIMESTAMP() 函数可以获得当前的 UTC 日期和时间。

讨论

有些应用程序需要使用当前日期或者时间，例如日志程序中，需要在每一条日志上加上当前日期和时间。有些日期相关运算也会用到当前时间或日期，例如查询本月的第一天和最后一天，或者计算下一个星期三的日期等。

通过三个函数可以获取当前日期和时间。CURDATE() 和 CURTIME() 函数分别返回当前日期和时间，NOW() 函数同时返回当前日期和时间：

```
mysql> SELECT CURDATE(), CURTIME(), NOW();
+-----+-----+-----+
| CURDATE() | CURTIME() | NOW()        |
+-----+-----+-----+
| 2006-06-03 | 09:41:50 | 2006-06-03 09:41:50 |
+-----+-----+-----+
```

CURRENT_DATE 和 CURRENT_TIME 函数与 CURDATE() 和 CURTIME() 函数是等价的；函数 CURRENT_TIMESTAMP 和 NOW() 是等价的。

前面提到的函数返回值，都是客户端连接所属时区的当前日期和时间（第 6.3 节）。而 `UTC_DATE()` 和 `UTC_TIMESTAMP()` 则返回 UTC 当前日期和时间。

使用 6.6 节中讨论的技术分解上述函数返回结果，来获取当前日期或者时间的部分值（例如现在是几点，今天是本月的第几天等）。

6.5 使用 TIMESTAMP 来跟踪行修改时间

Using `TIMESTAMP` to Track Row Modification Times

问题

如何让数据库自动记录一行记录的创建和最后修改时间。

解决方案

使用 `TIMESTAMP` 数据类型，它具有自动初始化和自动更新的特性。

讨论

`TIMESTAMP` 类型可用于同时保存日期和时间值。前面的章节中讨论了 `TIMESTAMP` 类型的取值范围（第 6.1 节），以及 `TIMESTAMP` 值在保存和查询时如何进行 UTC 转换（第 6.3 节）。本节主要讨论如何使用 `TIMESTAMP` 类型自动记录每一行的创建和最后修改时间：

- 表中的一个 `TIMESTAMP` 类型列可以从下面两个方面加以区别对待：
 - 当插入一个新行时，该列自动初始化为当前日期和时间。也就是说 MySQL 可以自动完成 `TIMESTAMP` 列的初始化，不需要在 `INSERT` 语句中特别处理。（如果在 `INSERT` 语句中将 `TIMESTAMP` 列值设为 `NULL`，服务器也会自动将其设为当前日期和时间。）
 - 当某一列数据发生变化时（使用原值再一次赋值在这里不能视为变化），`TIMESTAMP` 列可以自动更新为当前日期和时间。只有当你真的改变了某列的值时更新才会触发；将列值设为它的当前值并不会更新 `TIMESTAMP`。
- 当用户没有意识到由于改动了其他列数据而导致 `TIMESTAMP` 列发生变化时，这种自动更新的特性有时会让用户摸不着头脑。当然读过本节的用户就不会觉得惊奇了。
- 一张表中可以同时有多个 `TIMESTAMP` 类型列，但是只能有一列具有上述特性。其他列都会被初始化为 0，并且在其他列值发生变化时，也不会自动更新到当前日期和时间。实际上，用户需要使用 SQL 语句来更新其他的 `TIMESTAMP` 类型列值。

- 如果一个 TIMESTAMP 类型列有 NOT NULL 属性，那么把列值设为 NULL，可以将其值更新为当前日期和时间。如果表中有多个 TIMESTAMP 列，对每一列都可以通过把值设为 NULL，来更新到当前日期和时间。

行插入和更新具有的特殊属性，使得 TIMESTAMP 类型非常适合这一类操作，例如自动记录插入和更新的日期和时间。下面讨论如何利用这些特性。

《MySQL Reference Manual》中有定义一个 TIMESTAMP 类型列的详细方法，这里只讨论一些简单情况。在建表时，如果不对 TIMESTAMP 类型列做任何限制，它在默认情况下就具有自动初始化和自动更新的特性。在下面的例子中，使用 SHOW CREATE TABLE 语句，可以看到使用 CREATE TABLE 语句执行以后数据库对 TIMESTAMP 列的完整定义：

```
mysql> CREATE TABLE t (ts TIMESTAMP);
mysql> SHOW CREATE TABLE t\G
***** 1. row *****
Table: t
Create Table: CREATE TABLE `t` (
  `ts` timestamp NOT NULL
  DEFAULT CURRENT_TIMESTAMP
  ON UPDATE CURRENT_TIMESTAMP
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

如果显示的定义了 TIMESTAMP 列的 DEFAULT 或者 ON UPDATE 属性，那么它只会具有显示定义了的属性，不会默认的加上没有定义的。

列定义中的 NOT NULL 属性十分有趣，因为虽然定义说明不能保存 NULL 值，但是仍然可以给该列赋 NULL 值，但最终保存的不是 NULL，因为数据库会自动将其保存为当前日期和时间。

为了让数据库自动记录某一行插入或者最后一次更新的时间，可以在表定义中加上一个 TIMESTAMP 类型列，用它来完成自动记录。当你创建新的一行时 MySQL 将会把该列设为当前的日期和时间，并且无论何时你更新行中另一列的值时都会更新该列。假设用户定义了包含 TIMESTAMP 类型列的表 tsdemo1：

```
CREATE TABLE tsdemo1
(
  ts TIMESTAMP,
  val INT
);
```

此时，TIMESTAMP 类型列具有自动初始化和自动更新的能力。向 tsdemo1 中插入几列，然后再进行查询。(假设每两次插入之间都有时间间隔，以区分插入时间。) 第一个 INSERT 语句表明，可以将 ts 列自动初始化为当前的日期和时间；第二个 INSERT 语句表明，可

以通过将 `ts` 列设为 `NULL`, 从而将其值设为插入时的日期和时间。

```
mysql> INSERT INTO tsdemo1 (val) VALUES(5);
mysql> INSERT INTO tsdemo1 (ts,val) VALUES(NULL,10);
mysql> SELECT * FROM tsdemo1;
+-----+-----+
| ts      | val   |
+-----+-----+
| 2006-06-03 08:21:26 |    5 |
| 2006-06-03 08:21:31 |   10 |
+-----+-----+
```

下面执行一个语句来修改某一行的 `val` 列值, 然后观察这一操作的影响:

```
mysql> UPDATE tsdemo1 SET val = 6 WHERE val = 5;
mysql> SELECT * FROM tsdemo1;
+-----+-----+
| ts      | val   |
+-----+-----+
| 2006-06-03 08:21:52 |    6 |
| 2006-06-03 08:21:31 |   10 |
+-----+-----+
```

结果表明被修改行的 `TIMESTAMP` 列值也同时被更新了。

如果更新了多行数据, 那么每一行的 `TIMESTAMP` 列值都将被更新:

```
mysql> UPDATE tsdemo1 SET val = val + 1;
mysql> SELECT * FROM tsdemo1;
+-----+-----+
| ts      | val   |
+-----+-----+
| 2006-06-03 08:22:00 |    7 |
| 2006-06-03 08:22:00 |   11 |
+-----+-----+
```

如果一个语句没有绝对的改变 `val` 列值, 那就不会引起 `TIEMSTAMP` 列更新。下面的语句将 `val` 列值设为原列值, 然后让我们看一下数据有没有发生变化:

```
mysql> UPDATE tsdemo1 SET val = val;
mysql> SELECT * FROM tsdemo1;
+-----+-----+
| ts      | val   |
+-----+-----+
| 2006-06-03 08:22:00 |    7 |
| 2006-06-03 08:22:00 |   11 |
+-----+-----+
```

如果希望 `TIMESTAMP` 列自动初始化为行插入的日期和时间, 并且不希望该值以后发生变化, 那么可以使 `TIMESTAMP` 列具有 `auto-initialize` 属性, 但是没有 `auto-update` 属性。如下定义:

```
CREATE TABLE tsdemo2
(
    t_create TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    val      INT
);
```

创建了 tsdemo2 之后，向其中插入记录来初始化 TIMESTAMP 列值为当前日期和时间。在 SQL 语句中不需要对 TIMESTAMP 列做处理（或者将其设为 NULL）。

```
mysql> INSERT INTO tsdemo2 (val) VALUES(5);
mysql> INSERT INTO tsdemo2 (t_create, val) VALUES(NULL, 10);
mysql> SELECT * FROM tsdemo2;
+-----+-----+
| t_create | val |
+-----+-----+
| 2006-06-03 08:26:00 | 5 |
| 2006-06-03 08:26:05 | 10 |
+-----+-----+
```

完成插入后，修改 val 列值，然后观察是否引起了 t_create 列自动更新（其值为行插入日期和时间）：

```
mysql> UPDATE tsdemo2 SET val = val + 1;
mysql> SELECT * FROM tsdemo2;
+-----+-----+
| t_create | val |
+-----+-----+
| 2006-06-03 08:26:00 | 6 |
| 2006-06-03 08:26:05 | 11 |
+-----+-----+
```

参考

可以使用触发来代替 TIMESTAMP 的特有属性来完成对行记录的插入和修改时间的记录（参见 16.5 节）。

6.6 从日期或者时间值中分解出各部分值

Extracting Parts of Dates or Times

问题

如何从一个日期或者日期值中分解出一部分值。

解决方案

可以有多种方法：

- 使用一个特定函数从时间或日期值中取出一个特定部分值，例如 MONTH() 函数，或者 MINUTE() 函数。如果只需要从一个时间或日期值中取出一个特定部分值，那么使用这样的函数是最高效的。

- 使用格式化函数 `DATE_FORMAT()` 或者 `TIME_FORMAT()`，和一个特定的格式化串来从一个日期或者时间值中分解出需要的部分。
- 将一个时间或者日期值当作一个字符串，然后使用 `LEFT()` 或者 `MID()` 函数从其中解析出所需要的部分。

讨论

下面将讨论分解时间和日期值的各种方法。

使用成分分解函数来分解日期和时间值

MySQL 提供了很多函数用来从日期和时间值中获取特定部分值。例如，`DATE()` 和 `TIME()` 函数可以获取一个日期或者时间部分值。

```
mysql> SELECT dt, DATE(dt), TIME(dt) FROM datetime_val;
+-----+-----+-----+
| dt      | DATE(dt) | TIME(dt) |
+-----+-----+-----+
| 1970-01-01 00:00:00 | 1970-01-01 | 00:00:00 |
| 1987-03-05 12:30:15 | 1987-03-05 | 12:30:15 |
| 1999-12-31 09:00:00 | 1999-12-31 | 09:00:00 |
| 2000-06-04 15:45:30 | 2000-06-04 | 15:45:30 |
+-----+-----+-----+
```

下面给出了很多成分分解函数，在《MySQL Reference Manual》中可以查到所有这一类的函数。跟日期相关的函数可以作用于 `DATE`、`DATETIME` 或者 `TIMESTAMP` 类型值；和时间相关的函数可以作用于 `TIME`、`DATETIME` 或者 `TIEMSTAMP` 类型值。

函数	返回值
<code>YEAR()</code>	日期值中的年
<code>MONTH()</code>	月份数值 (1..12)
<code>MONTHNAME()</code>	月份名称 (January..December)
<code>DAYOFMONTH()</code>	月份中的天数值 (1..31)
<code>DAYNAME()</code>	一周中的天数 (Sunday..Saturday)
<code>DAYOFWEEK()</code>	一周中的天数 (1..7 对应 Sunday..Saturday)
<code>WEEKDAY()</code>	一周中的天数 (0..6 对应 Monday..Sunday)
<code>DAYOFYEAR()</code>	一年中的天数值 (1..366)
<code>HOUR()</code>	时间中的小时数 (0..23)
<code>MINUTE()</code>	时间中的分钟数 (0..59)
<code>SECOND()</code>	时间中的秒数 (0..59)

给出一个例子：

```
mysql> SELECT dt,
-> YEAR(dt), DAYOFMONTH(dt),
```

```

-> HOUR(dt), SECOND(dt)
-> FROM datetime_val;
+-----+-----+-----+-----+-----+
| dt           | YEAR(dt) | DAYOFMONTH(dt) | HOUR(dt) | SECOND(dt) |
+-----+-----+-----+-----+-----+
| 1970-01-01 00:00:00 |    1970 |          1 |      0 |       0 |
| 1987-03-05 12:30:15 |    1987 |          5 |     12 |      15 |
| 1999-12-31 09:00:00 |    1999 |         31 |      9 |       0 |
| 2000-06-04 15:45:30 |    2000 |          4 |     15 |      30 |
+-----+-----+-----+-----+-----+

```

有些函数从日期或者时间值中直接分解出一个子串，作为返回值，例如 YEAR() 和 DAYOFMONTH() 函数。但是有些函数可以返回在日期或者时间值中没有直接对应子串的部分值。年份中的天数值就是其一：

```

mysql> SELECT d, DAYOFYEAR(d) FROM date_val;
+-----+-----+
| d           | DAYOFYEAR(d) |
+-----+-----+
| 1864-02-28 |      59 |
| 1900-01-15 |      15 |
| 1987-03-05 |      64 |
| 1999-12-31 |     365 |
| 2000-06-04 |     156 |
+-----+-----+

```

另外一个例子是获取星期几，它可以通过名称或数值获得。

- DAYNAME() 函数返回 date 的星期名字。没有函数直接返回 date 星期名字的前三个字母，但是可以通过 LEFT() 函数来获得：

```

mysql> SELECT d, DAYNAME(d), LEFT(DAYNAME(d),3) FROM date_val;
+-----+-----+-----+
| d           | DAYNAME(d) | LEFT(DAYNAME(d),3) |
+-----+-----+-----+
| 1864-02-28 | Sunday    | Sun               |
| 1900-01-15 | Monday    | Mon               |
| 1987-03-05 | Thursday  | Thu               |
| 1999-12-31 | Friday    | Fri               |
| 2000-06-04 | Sunday    | Sun               |
+-----+-----+-----+

```

- DAYOFWEEK() 和 WEEKDAY() 函数返回 date 的星期值，但要注意的是这两个函数返回值的范围不一样。DAYOFWEEK() 返回值对应星期天到星期六为从 1 到 7；而 WEEKDAY() 的返回值则是从 0 到 6 分别代表星期一到星期天：

```

mysql> SELECT d, DAYNAME(d), DAYOFWEEK(d), WEEKDAY(d) FROM date_val;
+-----+-----+-----+-----+
| d           | DAYNAME(d) | DAYOFWEEK(d) | WEEKDAY(d) |
+-----+-----+-----+-----+
| 1864-02-28 | Sunday    |        1 |       6 |
| 1900-01-15 | Monday    |        2 |       0 |
+-----+-----+-----+-----+

```

1987-03-05	Thursday	5
1999-12-31	Friday	6
2000-06-04	Sunday	1

EXTRACT()是另外一个用于分解日期或者时间值的函数：

```
mysql> SELECT dt,
-> EXTRACT(DAY FROM dt),
-> EXTRACT(HOUR FROM dt),
-> FROM datetime_val;
+-----+-----+-----+
| dt | EXTRACT(DAY FROM dt) | EXTRACT(HOUR FROM dt) |
+-----+-----+-----+
| 1970-01-01 00:00:00 | 1 | 0 |
| 1987-03-05 12:30:15 | 5 | 12 |
| 1999-12-31 09:00:00 | 31 | 9 |
| 2000-06-04 15:45:30 | 4 | 15 |
+-----+-----+-----+
```

在该函数中，使用保留字指明要从时间或者日期值分解出的部分，例如 YEAR、MONTH、DAY、HOUR、MINUTE 或 SECOND（可以从《MySQL Reference Manual》中找到所有可用的保留字）。注意所有的保留字都是单数形式。

获取当前时刻的年、月、日、小时、分钟或者秒

本节讨论的函数都可以直接作用于 CURDATE() 和 NOW() 函数的返回值，来获取当前时刻的年、月、日或者是星期几：

```
mysql> SELECT CURDATE(), YEAR(CURDATE()) AS year,
-> MONTH(CURDATE()) AS month, MONTHNAME(CURDATE()) AS monthname,
-> DAYOFMONTH(CURDATE()) AS day, DAYNAME(CURDATE()) AS dayname;
+-----+-----+-----+-----+-----+
| CURDATE() | year | month | monthname | day | dayname |
+-----+-----+-----+-----+-----+
| 2006-06-03 | 2006 | 6 | June | 3 | Saturday |
+-----+-----+-----+-----+-----+
```

类似的，可以通过分解 CURTIME() 或者 NOW() 的返回值，得到当前时刻的小时、分钟或者秒：

```
mysql> SELECT NOW(), HOUR(NOW()) AS hour,
-> MINUTE(NOW()) AS minute, SECOND(NOW()) AS second;
+-----+-----+-----+
| NOW() | hour | minute | second |
+-----+-----+-----+
| 2006-06-03 09:45:32 | 9 | 45 | 32 |
+-----+-----+-----+
```

使用格式化函数分解日期或者时间值

使用 DATE_FORMAT() 或者 TIME_FORMAT() 函数，可以按照给定的格式格式化一个时间或者日期值。通过合适的格式化串，就可以分解出所需要的部分值：

```

mysql> SELECT dt,
-> DATE_FORMAT(dt, '%Y') AS year,
-> DATE_FORMAT(dt, '%d') AS day,
-> TIME_FORMAT(dt, '%H') AS hour,
-> TIME_FORMAT(dt, '%s') AS second
-> FROM datetime_val;
+-----+-----+-----+-----+
| dt | year | day | hour | second |
+-----+-----+-----+-----+
| 1970-01-01 00:00:00 | 1970 | 01 | 00 | 00 |
| 1987-03-05 12:30:15 | 1987 | 05 | 12 | 15 |
| 1999-12-31 09:00:00 | 1999 | 31 | 09 | 00 |
| 2000-06-04 15:45:30 | 2000 | 04 | 15 | 30 |
+-----+-----+-----+-----+

```

使用格式化函数，一次可以得到一个日期或者时间值的多个部分。例如，下面的语句从一个 DATETIME() 值中分解出完整的日期或者时间值：

```

mysql> SELECT dt,
-> DATE_FORMAT(dt, '%Y-%m-%d') AS 'date part',
-> TIME_FORMAT(dt, '%T') AS 'time part'
-> FROM datetime_val;
+-----+-----+
| dt | date part | time part |
+-----+-----+
| 1970-01-01 00:00:00 | 1970-01-01 | 00:00:00 |
| 1987-03-05 12:30:15 | 1987-03-05 | 12:30:15 |
| 1999-12-31 09:00:00 | 1999-12-31 | 09:00:00 |
| 2000-06-04 15:45:30 | 2000-06-04 | 15:45:30 |
+-----+-----+

```

使用格式化函数分解日期或者时间值的一个好处是，可以根据用户指定的格式来显示分解得到的结果。如果不按照 *CCYY-MM-DD* 的格式来显示一个日期值，或者只想显示一个时间值中的小时和分钟部分，可以容易地做到：

```

mysql> SELECT ts,
-> DATE_FORMAT(ts, '%M %e, %Y') AS 'descriptive date',
-> TIME_FORMAT(ts, '%H:%i') AS 'hours/minutes'
-> FROM timestamp_val;
+-----+-----+-----+
| ts | descriptive date | hours/minutes |
+-----+-----+-----+
| 1970-01-01 00:00:00 | January 1, 1970 | 00:00 |
| 1987-03-05 12:30:15 | March 5, 1987 | 12:30 |
| 1999-12-31 09:00:00 | December 31, 1999 | 09:00 |
| 2000-06-04 15:45:30 | June 4, 2000 | 15:45 |
+-----+-----+-----+

```

使用字符串函数分解时间或者日期值

本节到目前为止讨论了如何使用函数来分解一个日期或者时间值，例如 YEAR()、MONTH() 和 DATE_FORMAT() 函数。如果将一个日期或者时间值作为参数传递给一个字符串处理函数，MySQL 会把这样的参数当作字符串来处理。因此，可以使用字符串函数来分解日期

或者时间值。例如使用 LEFT() 或者 MID() 函数从时间日期值中分解出一个子串：

```
mysql> SELECT dt,
-> LEFT(dt,4) AS year,
-> MID(dt,9,2) AS day,
-> RIGHT(dt,2) AS second
-> FROM datetime_val;
+-----+-----+-----+
| dt      | year | day  | second |
+-----+-----+-----+
| 1970-01-01 00:00:00 | 1970 | 01   | 00     |
| 1987-03-05 12:30:15 | 1987 | 05   | 15     |
| 1999-12-31 09:00:00 | 1999 | 31   | 00     |
| 2000-06-04 15:45:30 | 2000 | 04   | 30     |
+-----+-----+-----+
```

使用 LEFT() 或者 RIGHT() 函数，可以从 DATETIME 或者 TIMESTAMP 类型值中分解出完整的日期或者时间值：

```
mysql> SELECT dt,
-> LEFT(dt,10) AS date,
-> RIGHT(dt,8) AS time
-> FROM datetime_val;
+-----+-----+-----+
| dt          | date       | time      |
+-----+-----+-----+
| 1970-01-01 00:00:00 | 1970-01-01 | 00:00:00 |
| 1987-03-05 12:30:15 | 1987-03-05 | 12:30:15 |
| 1999-12-31 09:00:00 | 1999-12-31 | 09:00:00 |
| 2000-06-04 15:45:30 | 2000-06-04 | 15:45:30 |
+-----+-----+-----+
mysql> SELECT ts,
-> LEFT(ts,10) AS date,
-> RIGHT(ts,8) AS time
-> FROM timestamp_val;
+-----+-----+-----+
| ts          | date       | time      |
+-----+-----+-----+
| 1970-01-01 00:00:00 | 1970-01-01 | 00:00:00 |
| 1987-03-05 12:30:15 | 1987-03-05 | 12:30:15 |
| 1999-12-31 09:00:00 | 1999-12-31 | 09:00:00 |
| 2000-06-04 15:45:30 | 2000-06-04 | 15:45:30 |
+-----+-----+-----+
```

相对于直接分解函数和格式化函数来说，MySQL 对使用字符串函数来分解时间或者日期值有更多的限制：

- 如果使用 LEFT()、RIGHT() 或者 MID() 函数，要求所作用的字符串必须是定长的。尽管将 1987-1-1 保存到一个 DATE 类型的列中时，MySQL 会自动将其解释为 1987-01-01，但是不能使用 RIGHT('1987-1-1', 2) 来获取对应时间值的日期部分。如果要获取长度可变的子串，可以使用 SUBSTRING_INDEX() 函数。如果所要分解的日期

或者时间值的格式接近于 ISO 格式，可以首先使用 6.18 节中的技术将其转化为 ISO 标准格式，这样就可以使待处理的时间或者日期值字符串长度固定了。

- 使用字符串函数的另一个限制，是不能取得所给的时间或者日期值中没有出现的值。例如使用字符串函数，不能直接得到一个日期值是星期几，或者是一年的第几天。

6.7 合成日期或者时间值

Synthesizing Dates or Times from Component Values

问题

如何替换一个给定日期值的一部分，从而得到一个新的日期值？如何通过组合一个日期或者时间值的各个部分，从而得到新的日期或者时间值？

解决方案

用户有多种选择：

- 使用 MAKETIME() 函数，组合小时、分钟和秒得到一个 TIME 类型值。
- 使用 DATE_FORMAT() 或者 TIME_FORMAT() 函数，来替换一个时间或者日期值的一部分，从而得到新的时间或者日期值。
- 使用分解函数从时间或者日期值中分解出部分值，然后再使用 CONCAT() 函数组合各部分值，得到新值。

讨论

分解一个时间或者日期值的逆过程，就是通过组合时间或者日期值的各个部分，得到新值。可以通过使用合成函数、格式化函数或者字符串串联合成时间或者日期值。

MAKETIME() 函数接收三个参数：小时、分钟和秒，并组合得到一个时间值：

```
mysql> SELECT MAKETIME(10,30,58), MAKETIME(-5,0,11);
+-----+-----+
| MAKETIME(10,30,58) | MAKETIME(-5,0,11) |
+-----+-----+
| 10:30:58          | -05:00:11       |
+-----+-----+
```

MAKEDATE() 函数可以组合生成一个日期值，它接收的参数为年和当年的第几天：

```
mysql> SELECT MAKEDATE(2007,60);
+-----+
| MAKEDATE(2007,60) |
+-----+
| 2007-03-01        |
+-----+
```

笔者并不认为 MAKEDATE() 函数有多少用处，因为更多的时候是使用年、月和当月的第几天来合成一个日期值。

生成一个日期值通常的流程，都是首先查询或者从外部程序输入一个已有的日期值，然后替换其中的特定部分。例如，要生成某一个日期对应的当月第一天的日期，可以首先使用 DATE_FORMAT() 函数从一个日期值中分解出年和月，然后再在这个部分日期值的末尾加上 01：

```
mysql> SELECT d, DATE_FORMAT(d, '%Y-%m-01') FROM date_val;
+-----+-----+
| d      | DATE_FORMAT(d, '%Y-%m-01') |
+-----+-----+
| 1864-02-28 | 1864-02-01 |
| 1900-01-15 | 1900-01-01 |
| 1987-03-05 | 1987-03-01 |
| 1999-12-31 | 1999-12-01 |
| 2000-06-04 | 2000-06-01 |
+-----+-----+
```

TIME_FORMAT() 也有和 DATE_FORMAT() 类似的用法。如下语句生成秒部分为 00 的时间值：

```
mysql> SELECT t1, TIME_FORMAT(t1, '%H:%i:00') FROM time_val;
+-----+-----+
| t1      | TIME_FORMAT(t1, '%H:%i:00') |
+-----+-----+
| 15:00:00 | 15:00:00 |
| 05:01:30 | 05:01:00 |
| 12:30:20 | 12:30:00 |
+-----+-----+
```

合成时间或者日期值的另一种方法，是组合使用时间日期分解函数和 CONCAT() 函数。但是，这种方法和使用 DATE_FORMAT() 的方法相比略显凌乱，并且得到的结果也略有不同：

```
mysql> SELECT d,
    -> CONCAT(YEAR(d), '-', MONTH(d), '-01')
    -> FROM date_val;
+-----+-----+
| d      | CONCAT(YEAR(d), '-', MONTH(d), '-01') |
+-----+-----+
| 1864-02-28 | 1864-2-01 |
| 1900-01-15 | 1900-1-01 |
| 1987-03-05 | 1987-3-01 |
| 1999-12-31 | 1999-12-01 |
| 2000-06-04 | 2000-6-01 |
+-----+-----+
```

读者可能会注意到，在产生的结果中，有的月份值只有一位数。可以使用 LPAD() 函数，保证所有的月份值都由两位数字组成（这也是 ISO 格式规定）。该函数会在需要的时候在数值右侧添加 0：

```
mysql> SELECT d,
    -> CONCAT(YEAR(d), '-', LPAD(MONTH(d), 2, '0'), '-01')
```

```
-> FROM date_val;
+-----+
| d      | CONCAT(YEAR(d), '-', LPAD(MONTH(d), 2, '0'), '-01') |
+-----+
| 1864-02-28 | 1864-02-01
| 1900-01-15 | 1900-01-01
| 1987-03-05 | 1987-03-01
| 1999-12-31 | 1999-12-01
| 2000-06-04 | 2000-06-01
+-----+
```

6.18 节中讨论了将非 ISO 格式日期转化为 ISO 格式日期的其他方法。

和合成日期值的方法类似，通过组合小时、分钟和秒生成 TIME 值。例如，为了对一个 TIME 类型值进行处理，使其秒部分为 00，首先可以从这个 TIME 值中分解出小时和分钟的部分值，然后使用 CONCAT() 函数与 00 串联：

```
mysql> SELECT t1,
->   CONCAT(LPAD(HOUR(t1), 2, '0'), ':', LPAD(MINUTE(t1), 2, '0'), ':00')
->   AS recombined
->   FROM time_val;
+-----+-----+
| t1    | recombined |
+-----+-----+
| 15:00:00 | 15:00:00   |
| 05:01:30 | 05:01:00   |
| 12:30:20 | 12:30:00   |
+-----+-----+
```

将一个日期值和一个时间值串联，并在中间插入一个空格，就得到了一个 DATETIME 或者 TIMESTAMP 类型值：

```
mysql> SET @d = '2006-02-28';
mysql> SET @t = '13:10:05';
mysql> SELECT @d, @t, CONCAT(@d, ' ', @t);
+-----+-----+-----+
| @d    | @t    | CONCAT(@d, ' ', @t)  |
+-----+-----+-----+
| 2006-02-28 | 13:10:05 | 2006-02-28 13:10:05 |
+-----+-----+-----+
```

6.8 在时间数据类型和基本单位间进行转换

Converting Between Temporal Data Types and Basic Units

问题

如何将一个时间或者日期值转换为相应的单位值，例如秒或者天。这样的转化在与时间相关的计算中很有用，或者是必要的（6.9 节和 6.10 节中详细讨论）。

解决方案

转化采用的方法，主要取决于所要处理的数据类型：

- 在时间类型值和秒之间进行转换，可以使用 TIME_TO_SEC() 函数和 SEC_TO_TIME() 函数；
- 在日期类型值和天之间转换，可以使用 TO_DAYS() 和 FROM_DAYS() 函数；
- 在 TIMESTAMP 或者 DATETIME 类型值和秒之间进行转换，可以使用 UNIX_TIMESTAMP() 函数和 FROM_UNIXTIME() 类型值。

讨论

下面的讨论说明了如何在时间和日期值，与标准单位时间之间进行转换。

在时间值和秒之间进行转换

TIME 类型值是简单基本类型（秒）的一种特殊表现形式，因此可以使用 TIME_TO_SEC() 和 SEC_TO_TIME() 函数，在 TIME 值和秒之间进行转化。

TIME_TO_SEC() 函数将一个 TIME 类型值转换为对应的秒数，SEC_TO_TIME() 则相反。下面的语句通过一个简单的例子来说明两个方向的转换：

```
mysql> SELECT t1,
-> TIME_TO_SEC(t1) AS 'TIME to seconds',
-> SEC_TO_TIME(TIME_TO_SEC(t1)) AS 'TIME to seconds to TIME'
-> FROM time_val;
+-----+-----+-----+
| t1      | TIME to seconds | TIME to seconds to TIME |
+-----+-----+-----+
| 15:00:00 |          54000 | 15:00:00                |
| 05:01:30 |          18090 | 05:01:30                |
| 12:30:20 |          45020 | 12:30:20                |
+-----+-----+-----+
```

通过除法，可以用分钟、小时或者天来表示一个时间值：

```
mysql> SELECT t1,
-> TIME_TO_SEC(t1) AS 'seconds',
-> TIME_TO_SEC(t1)/60 AS 'minutes',
-> TIME_TO_SEC(t1)/(60*60) AS 'hours',
-> TIME_TO_SEC(t1)/(24*60*60) AS 'days'
-> FROM time_val;
+-----+-----+-----+-----+-----+
| t1      | seconds | minutes | hours   | days    |
+-----+-----+-----+-----+-----+
| 15:00:00 | 54000  | 900.0000 | 15.0000 | 0.6250  |
| 05:01:30 | 18090  | 301.5000 | 5.0250  | 0.2094  |
| 12:30:20 | 45020  | 750.3333 | 12.5056 | 0.5211  |
+-----+-----+-----+-----+-----+
```

使用 FLOOR() 函数截除一个数值的小数部分：

```
mysql> SELECT t1,
-> TIME_TO_SEC(t1) AS 'seconds',
-> FLOOR(TIME_TO_SEC(t1)/60) AS 'minutes',
-> FLOOR(TIME_TO_SEC(t1)/(60*60)) AS 'hours',
-> FLOOR(TIME_TO_SEC(t1)/(24*60*60)) AS 'days'
-> FROM time_val;
+-----+-----+-----+-----+
| t1      | seconds | minutes | hours | days |
+-----+-----+-----+-----+
| 15:00:00 |    54000 |     900 |    15 |    0 |
| 05:01:30 |   18090 |     301 |     5 |    0 |
| 12:30:20 |   45020 |     750 |    12 |    0 |
+-----+-----+-----+-----+
```

如果将一个 DATETIME 或者 TIMESTAMP 类型值传递给 TIME_TO_SEC() 函数作为参数，那么其中的日期部分值将会被忽略。利用这一特性可以从 DATETIME 或者 TIMESTAMP 类型值中提取出时间部分值（可以作为第 6.6 节中讨论方法的一个补充）：

```
mysql> SELECT dt,
-> TIME_TO_SEC(dt) AS 'time part in seconds',
-> SEC_TO_TIME(TIME_TO_SEC(dt)) AS 'time part as TIME'
-> FROM datetime_val;
+-----+-----+-----+
| dt          | time part in seconds | time part as TIME |
+-----+-----+-----+
| 1970-01-01 00:00:00 |                      0 | 00:00:00 |
| 1987-03-05 12:30:15 |                  45015 | 12:30:15 |
| 1999-12-31 09:00:00 |                  32400 | 09:00:00 |
| 2000-06-04 15:45:30 |                  56730 | 15:45:30 |
+-----+-----+-----+
mysql> SELECT ts,
-> TIME_TO_SEC(ts) AS 'time part in seconds',
-> SEC_TO_TIME(TIME_TO_SEC(ts)) AS 'time part as TIME'
-> FROM timestamp_val;
+-----+-----+-----+
| ts          | time part in seconds | time part as TIME |
+-----+-----+-----+
| 1970-01-01 00:00:00 |                      0 | 00:00:00 |
| 1987-03-05 12:30:15 |                  45015 | 12:30:15 |
| 1999-12-31 09:00:00 |                  32400 | 09:00:00 |
| 2000-06-04 15:45:30 |                  56730 | 15:45:30 |
+-----+-----+-----+
```

在日期值和天数之间进行转换

使用 TO_DAYS() 和 FROM_DAYS() 函数可以在一个日期值和对应的天数之间进行转化。如果可以忽略截除 DATETIME 或者 TIMESTAMP 中时间部分值引起的误差，也可以使用上述函数在 DATETIME 或者 TIMESTAMP 类型，和天数之间进行转化。

TO_DAYS() 函数将一个日期值转换为对应的天数、FROM_DAYS() 则执行相反的过程。

```

mysql> SELECT d,
-> TO_DAYS(d) AS 'DATE to days',
-> FROM_DAYS(TO_DAYS(d)) AS 'DATE to days to DATE'
-> FROM date_val;
+-----+-----+-----+
| d      | DATE to days | DATE to days to DATE |
+-----+-----+-----+
| 1864-02-28 |       680870 | 1864-02-28          |
| 1900-01-15 |       693975 | 1900-01-15          |
| 1987-03-05 |       725800 | 1987-03-05          |
| 1999-12-31 |       730484 | 1999-12-31          |
| 2000-06-04 |       730640 | 2000-06-04          |
+-----+-----+-----+

```

在使用 `TO_DAYS()` 函数时，最好严格遵守《MySQL Reference Manual》的建议，并避免使用早于格林尼治时间（1582 年）的 `DATE` 类型值。在这个日期之前的 `DATE` 类型值，由于其年和月的长度的变化，很难对第 0 天“day 0”作出有意义的定义。这与 `TIME_TO_SEC()` 函数不同，在 `TIME_TO_SEC()` 函数中，`TIME` 类型值与其对应的秒数在任何情况下都是有明确含义的，并且第 0 秒也是有明确定义的。

如果给 `TO_DAYS()` 函数传递了一个 `DATETIME` 或者 `TIMESTAMP` 类型值，该函数会从参数中提取出日期部分，而忽略时间值。这就提供了一种从 `DATETIME` 或者 `TIMESTAMP` 类型值中分解出日期值的新方法（在第 6.6 节中讨论了其他方法）：

```

mysql> SELECT dt,
-> TO_DAYS(dt) AS 'date part in days',
-> FROM_DAYS(TO_DAYS(dt)) AS 'date part as DATE'
-> FROM datetime_val;
+-----+-----+-----+
| dt            | date part in days | date part as DATE |
+-----+-----+-----+
| 1970-01-01 00:00:00 |           719528 | 1970-01-01          |
| 1987-03-05 12:30:15 |           725800 | 1987-03-05          |
| 1999-12-31 09:00:00 |           730484 | 1999-12-31          |
| 2000-06-04 15:45:30 |           730640 | 2000-06-04          |
+-----+-----+-----+
mysql> SELECT ts,
-> TO_DAYS(ts) AS 'date part in days',
-> FROM_DAYS(TO_DAYS(ts)) AS 'date part as DATE'
-> FROM timestamp_val;
+-----+-----+-----+
| ts            | date part in days | date part as DATE |
+-----+-----+-----+
| 1970-01-01 00:00:00 |           719528 | 1970-01-01          |
| 1987-03-05 12:30:15 |           725800 | 1987-03-05          |
| 1999-12-31 09:00:00 |           730484 | 1999-12-31          |
| 2000-06-04 15:45:30 |           730640 | 2000-06-04          |
+-----+-----+-----+

```

在 `DATETIME` 或者 `TIMESTAMP` 类型值和秒数之间进行转换

对于取值范围和 `TIMESTAMP` 类型（1970 年到 2037 年）相同的 `DATETIME` 或者 `TIMESTAMP` 类型，可以使用 `UNIX_TIMESTAMP()` 函数和 `FROM_UNIXTIME()` 函数，与从 1970 年开始的

秒数进行转换。将一个 TIMESTAMP 或者 DATETIME 类型转换为秒数，比转换为天数更精确，但同时对所要进行转换的值的取值范围也有了限制。(TIME_TO_SEC()函数忽略日期值，因此不能用于这种转换)：

```
mysql> SELECT dt,
-> UNIX_TIMESTAMP(dt) AS seconds,
-> FROM_UNIXTIME(UNIX_TIMESTAMP(dt)) AS timestamp
-> FROM datetime_val;
+-----+-----+-----+
| dt      | seconds | timestamp |
+-----+-----+-----+
| 1970-01-01 00:00:00 | 21600 | 1970-01-01 00:00:00 |
| 1987-03-05 12:30:15 | 541967415 | 1987-03-05 12:30:15 |
| 1999-12-31 09:00:00 | 946652400 | 1999-12-31 09:00:00 |
| 2000-06-04 15:45:30 | 960151530 | 2000-06-04 15:45:30 |
+-----+-----+-----+
```

这些函数的函数名中的“UNIX”，与它们作用的时间值的取值范围的关系在于 1970-01-01 00:00:00 UTC 标志了“Unix epoch”。epoch 是时间 0，在 UNIX 系统中是计算时间的起点。看到这里，读者可能对上面的例子感到奇怪，我们将 UNIX_TIMESTAMP() 函数作用于时间值 1970-01-01 00:00:00，得到的结果居然是 21600。到底哪里不对？结果为什么不是 0？带来差异的原因是 UNIX_TIMESTAMP() 函数把传递给它的时间值从客户端时区转换到 UTC。笔者使用的服务器处在美国中部时区，比 UTC 晚 6 小时（21 600 秒）。

使用 UNIX_TIMESTAMP() 函数也可以把 DATE 类型值转换为秒，相当于给 DATE 类型值一个 00:00:00 的时间部分值：

```
mysql> SELECT
-> CURDATE(),
-> UNIX_TIMESTAMP(CURDATE()),
-> FROM_UNIXTIME(UNIX_TIMESTAMP(CURDATE()))\G
***** 1. row *****
CURDATE(): 2006-05-30
UNIX_TIMESTAMP(CURDATE()): 1148965200
FROM_UNIXTIME(UNIX_TIMESTAMP(CURDATE())): 2006-05-30 00:00:00
```

6.9 计算两个日期或时间之间的间隔

Calculating the Interval Between Two Dates or Times

问题

你想知道两个日期或时间之间有多长。也就是说，你想知道两个时间值之间的间隔。

解决方案

为了计算两个时刻之间的间隔，可以使用时间差函数，或者将所要处理的值转换为基本时间单位（天或者秒），然后计算间隔。根据所要处理的值类型来选择合适的函数。

讨论

下面讨论几种不同的方法，来计算时间间隔。

使用时间差函数

要计算两个日期值之间间隔的天数，可以使用 DATEDIFF() 函数：

```
mysql> SET @d1 = '2010-01-01', @d2 = '2009-12-01';
mysql> SELECT DATEDIFF(@d1,@d2) AS 'd1 - d2', DATEDIFF(@d2,@d1) AS 'd2 - d1';
+-----+-----+
| d1 - d2 | d2 - d1 |
+-----+-----+
|      31 |     -31 |
+-----+-----+
```

DATEDIFF() 函数也可以作用于 date-and-time 类型，但是在处理过程中将忽略时间部分值。也就是说 DATEDIFF() 函数可用于计算两个 DATE、DATETIME 或者 TIMESTAMP 类型值之间间隔的天数。

使用 TIMEDIFF() 函数计算两个 TIME 类型值之间的时间间隔：

```
mysql> SET @t1 = '12:00:00', @t2 = '16:30:00';
mysql> SELECT TIMEDIFF(@t1,@t2) AS 't1 - t2', TIMEDIFF(@t2,@t1) AS 't2 - t1';
+-----+-----+
| t1 - t2 | t2 - t1 |
+-----+-----+
| -04:30:00 | 04:30:00 |
+-----+-----+
```

TIMEDIFF() 函数也可以作用于 date-and-time 类型值。也就是说 TIMEDIFF() 函数接受的参数类型可以是 TIME，也可以是 date-and-time 类型，但是要求两个参数类型相同。

使用 TIME 类型值表示的时间间隔，可以使用 6.6 节中的方法进行分解。例如，为了将一个时间间隔分为时、分、秒值显示，在 SQL 中使用 HOUR()、MINUTE() 和 SECOND() 函数计算时间间隔的各个部分。（不要忘记如果你的时间间隔为负数，你需要将其考虑在内。）要确定 time_val 表中 t1 和 t2 列之间时间间隔的各个部分，下面的 SQL 语句可以达成此目的：

```
mysql> SELECT t1, t2,
-> TIMEDIFF(t2,t1) AS 't2 - t1 as TIME',
-> IF(TIMEDIFF(t2,t1) >= 0,'+', '-') AS sign,
-> HOUR(TIMEDIFF(t2,t1)) AS hour,
-> MINUTE(TIMEDIFF(t2,t1)) AS minute,
-> SECOND(TIMEDIFF(t2,t1)) AS second
-> FROM time_val;
```

t1	t2	t2 - t1 as TIME	sign	hour	minute	second
15:00:00	15:00:00	00:00:00	+	0	0	0
05:01:30	02:30:20	-02:31:10	-	2	31	10
12:30:20	17:30:45	05:00:25	+	5	0	25

对于 date 或者 data-and-time 类型值，还可以使用 `TIMESTAMPDIFF()` 函数来计算时间间隔，在使用该函数时，可以指定函数返回的时间间隔使用的基本单位。函数定义为：

```
TIMESTAMPDIFF(unit, val1, val2)
```

其中 `unit` 为计算时间间隔所用的基本单位（天或者秒），`val1` 和 `val2` 分别是所要计算相互间隔的两个时间值。利用 `TIMESTAMPDIFF()` 函数，可以用多种方式来表示两个时间值之间的间隔：

```
mysql> SET @dt1 = '1900-01-01 00:00:00', @dt2 = '1910-01-01 00:00:00';
mysql> SELECT
    -> TIMESTAMPDIFF(MINUTE,@dt1,@dt2) AS minutes,
    -> TIMESTAMPDIFF(HOUR,@dt1,@dt2) AS hours,
    -> TIMESTAMPDIFF(DAY,@dt1,@dt2) AS days,
    -> TIMESTAMPDIFF(WEEK,@dt1,@dt2) AS weeks,
    -> TIMESTAMPDIFF(YEAR,@dt1,@dt2) AS years;
+-----+-----+-----+-----+
| minutes | hours | days | weeks | years |
+-----+-----+-----+-----+
| 5258880 | 87648 | 3652 | 521 | 10 |
+-----+-----+-----+-----+
```

可以使用的时间间隔基本单位有：`FRAC_SECOND`、`SECOND`、`MINUTE`、`HOUR`、`DAY`、`WEEK`、`MONTH`、`QUARTER` 以及 `YEAR`。注意所有的基本单位都是单数形式的，而不是复数。

注意 `TIMESTAMPDIFF()` 函数的以下特性：

- 如果参数中的第一个时间值大于（晚于）第二个，那么返回值是负数。这与 `DATEDIFF()` 和 `TIMEDIFF()` 中的参数顺序与返回值的关系相反。
- 虽然函数名中有 `TIMESTAMP`，但是参数的有效范围并不受 `TIMESTAMP` 类型限制，可以超出 `TIMESTAMP` 类型的取值范围。
- 只有 MySQL 5.0 以后的版本支持该函数。对之前的 MySQL 服务器，可以使用本节中讨论的其他方法来计算时间间隔。

使用基本时间单位来计算时间间隔

另一种计算时间间隔的方法，是使用基本时间单位，例如天或者秒：

- 将所要处理的日期或者时间值转换为对应的基本时间单位表示（多少天，多少秒）。
- 计算基本单位计量的时间值之间的差值，计算结果仍然用基本单位计量。
- 如果希望最终结果是一个日期或者时间值，那么在把计算结果从基本单位转换回来即可。

在这种计算方法中，需要根据所要计算的时间间隔的值类型，来选择合适的转换函数：

- 在时间值和秒数之间进行转换，使用 `TIME_TO_SEC()` 和 `SEC_TO_TIME()` 函数。
- 在日期值和天数之间转换，使用 `TO_DAYS()` 和 `FROM_DAYS()` 函数。
- 在 `date-and-time` 类型和秒数之间转换，使用 `UNIX_TIMESTAMP()` 和 `FROM_UNIXTIME()` 函数。

有关转换函数（和对转换过程的限制），请参考 6.8 节中的讨论。下面的讨论中都假设读者熟悉转换过程。

利用基本时间单位计算时间间隔

要计算两个时间之间间隔的秒数，首先使用 `TIME_TO_SEC()` 将这两个时间都转换为秒，然后计算差值。如果希望计算结果仍然以一个时间值的形式出现，需要使用 `SEC_TO_TIME()` 对计算结果再做一次转换。下面的语句计算了 `time_val` 表中，`t1` 列和 `t2` 列值之间的时间间隔，分别用秒和 `TIME` 类型值显示了计算结果：

```
mysql> SELECT t1, t2,
-> TIME_TO_SEC(t2) - TIME_TO_SEC(t1) AS 't2 - t1 (in seconds)',
-> SEC_TO_TIME(TIME_TO_SEC(t2) - TIME_TO_SEC(t1)) AS 't2 - t1 (as TIME)'
-> FROM time_val;
+-----+-----+-----+-----+
| t1      | t2      | t2 - t1 (in seconds) | t2 - t1 (as TIME) |
+-----+-----+-----+-----+
| 15:00:00 | 15:00:00 | 0 | 00:00:00          |
| 05:01:30 | 02:30:20 | -9070 | -02:31:10         |
| 12:30:20 | 17:30:45 | 18025 | 05:00:25          |
+-----+-----+-----+-----+
```

使用基本单位计算两个 Date 类型值，或者两个 date-and-time 类型值的时间间隔

当把两个日期都转换为基本单位，然后再计算它们之间的时间间隔，此时所处理的日期的取值范围，就决定了所选用的转换函数：

- 如果所处理的 `DATE`、`DATETIME` 或者 `TIMESTAMP` 类型变量值，都是 1970-01-01 00:00:00（UNIX 计时起始点）以后的，那么可以将其转换为从 UNIX 计时起始点开始的秒数。这样，所计算得到的时间间隔可以精确到秒。
- 而 1582 年以前的日期值，只能以天为单位进行转换，并计算时间间隔。
- 早于这些时间点的日期值导致更多问题。这种情况下，你可能会发现你的编程语言提供了在 SQL 中不可用或很难执行的计算。如果是这样，考虑直接从编程语言 API 处

理日期值。(例如, CPAN 中的 Date::Calc 和 Date::Manip 模块可用在 Perl 脚本中。)

如果要以天为单位来计算两个日期值或 date-and-time 类型值的时间间隔, 可以首先使用 TO_DAYS() 函数将所要处理的值转换为多少天:

```
mysql> SELECT TO_DAYS('1884-01-01') - TO_DAYS('1883-06-05') AS days;
+-----+
| days |
+-----+
| 210 |
+-----+
```

如果要以星期为单位来计算时间间隔, 使用和上例一样的方法, 最后结果再除以 7 即可:

```
mysql> SELECT (TO_DAYS('1884-01-01') - TO_DAYS('1883-06-05')) / 7 AS weeks;
+-----+
| weeks |
+-----+
| 30.0000 |
+-----+
```

不能简单地用除法把天转换为月或者年, 因为月和年的天数都是变化的。如果要以月或者年为单位计算时间间隔, 可以使用本节讨论过的 TIMESTAMPDIFF() 函数:

对于 1970 到 2037 之间的 date-and-time 类型值, 可以使用 UNIX_TIMESTAMP() 函数, 以秒为单位计算时间间隔。例如, 以秒为单位, 来计算相隔两个星期的两个日期之间的时间间隔:

```
mysql> SET @dt1 = '1984-01-01 09:00:00';
mysql> SET @dt2 = @dt1 + INTERVAL 14 DAY;
mysql> SELECT UNIX_TIMESTAMP(@dt2) - UNIX_TIMESTAMP(@dt1) AS seconds;
+-----+
| seconds |
+-----+
| 1209600 |
+-----+
```

使用数学操作, 可以把以秒为单位计算的时间间隔, 转换为以其他单位计量。多少秒可以转换为多少分钟、多少小时、多少天或者是多少个星期:

```
mysql> SET @interval = UNIX_TIMESTAMP(@dt2) - UNIX_TIMESTAMP(@dt1);
mysql> SELECT @interval AS seconds,
       -> @interval / 60 AS minutes,
       -> @interval / (60 * 60) AS hours,
       -> @interval / (24 * 60 * 60) AS days,
       -> @interval / (7 * 24 * 60 * 60) AS weeks;
+-----+-----+-----+-----+-----+
| seconds | minutes | hours | days | weeks |
+-----+-----+-----+-----+-----+
| 1209600 | 20160.0000 | 336.0000 | 14.0000 | 2.0000 |
+-----+-----+-----+-----+-----+
```

可以使用 FLOOR() 函数处理计算结果，避免得到一个小数值。以后的很多例子都会用到这个函数。

对于超出 TIMESTAMP 类型取值范围的 date-and-time 值，可以使用更加通用（但是也更复杂）的方法来计算时间间隔：

- 首先以天为单位求得时间间隔，然后将所得结果和 $24 \times 60 \times 60$ 相乘，即可得到以秒为单位计量的时间间隔。
- 在所得结果上，再加上以秒为单位的两个 date-and-time 值的时间部分的差值，即得到最终结果。

下面，用两个相隔一星期的 date-and-time 值为例来说明：

```
mysql> SET @dt1 = '1800-02-14 07:30:00';
mysql> SET @dt2 = @dt1 + INTERVAL 7 DAY;
mysql> SET @interval =
    -> ((TO_DAYS(@dt2) - TO_DAYS(@dt1)) * 24*60*60)
    -> + TIME_TO_SEC(@dt2) - TIME_TO_SEC(@dt1);
mysql> SELECT @interval AS seconds, SEC_TO_TIME(@interval) AS TIME;
+-----+-----+
| seconds | TIME      |
+-----+-----+
| 604800 | 168:00:00 |
+-----+-----+
```

所需要的是时间间隔，还是两个时刻的跨度

当计算两个日期（或者时间）之间的差值时，需要考虑所需要的是它们之间的间隔，还是它们的跨度。本节讨论的两个日期之间的差值，也就是两个日期之间的间隔。如果需要计算两个日期的跨度，还需要在所得到的时间间隔上加上一个计量单位。例如，2002-01-01 和 2002-01-04 之间的时间间隔是 3 天，但是时间跨度是 4 天。如果使用时间差，没有得到你想要的结果，那么你可能需要加上或者减去一个计量单位。

6.10 增加日期或时间值

Adding Date or Time Values

问题

如何对时间或者日期值做加法。例如，怎样在一个时间上加上几秒，或者如何知道从今天起三个星期以后的日期？

解决方案

有几种方法对时间或者日期做加法：

- 使用时间加法函数；
- 使用+INTERNAL 或者-INTERNAL 函数；

- 转换为基本计时单位，然后求和。

所要进行求和的时间或日期类型决定所用的函数和操作符。

讨论

下面讨论几种进行时间值求和的方法：

使用时间加法函数或者操作符进行时间值求和运算

使用 ADDTIME() 函数，把一个时间值或者一个 date-and-time 类型值和一个时间值相加：

```
mysql> SET @t1 = '12:00:00', @t2 = '15:30:00';
mysql> SELECT ADDTIME(@t1,@t2);
+-----+
| ADDTIME(@t1,@t2) |
+-----+
| 27:30:00 |
+-----+
mysql> SET @dt = '1984-03-01 12:00:00', @t = '12:00:00';
mysql> SELECT ADDTIME(@dt,@t);
+-----+
| ADDTIME(@dt,@t) |
+-----+
| 1984-03-02 00:00:00 |
+-----+
```

使用 TIMESTAMP() 函数，把一个时间值或者一个 date-and-time 类型值和一个时间值相加：

```
mysql> SET @d = '1984-03-01', @t = '15:30:00';
mysql> SELECT TIMESTAMP(@d,@t);
+-----+
| TIMESTAMP(@d,@t) |
+-----+
| 1984-03-01 15:30:00 |
+-----+
mysql> SET @dt = '1984-03-01 12:00:00', @t = '12:00:00';
mysql> SELECT TIMESTAMP(@dt,@t);
+-----+
| TIMESTAMP(@dt,@t) |
+-----+
| 1984-03-02 00:00:00 |
+-----+
```

MySQL 也提供了 DATE_ADD() 和 DATE_SUB() 函数，来对一个日期值和一个时间值进行加法或减法运算。每个函数都以一个日期（或日期和时间）值 d 和一个时间间隔为参数，语法表示如下：

```
DATE_ADD(d, INTERVAL val unit)
DATE_SUB(d, INTERVAL val unit)
```

+INTERVAL 和-INTERVAL 操作符也是类似语法：

```
d + INTERVAL val unit
d - INTERVAL val unit
```

unit 就是计量时间间隔的时间单位, *val* 指明多少个 *unit*。常用的时间单位有 SECOND、MINUTE、HOUR、DAY、MONTH 和 YEAR (在《MySQL Reference Manual》中可以查到所有可用的时间单位)。注意所有的时间单位都是单数, 而不是复数。

使用 DATE_ADD() 或者 DATE_SUB() 函数, 对日期值可以进行如下数学运算:

- 求出即日第三天的日期值

```
mysql> SELECT CURDATE(), DATE_ADD(CURDATE(), INTERVAL 3 DAY);
+-----+-----+
| CURDATE() | DATE_ADD(CURDATE(), INTERVAL 3 DAY) |
+-----+-----+
| 2006-05-22 | 2006-05-25 |
+-----+-----+
```

- 一个星期以前的日期值:

```
mysql> SELECT CURDATE(), DATE_SUB(CURDATE(), INTERVAL 7 DAY);
+-----+-----+
| CURDATE() | DATE_SUB(CURDATE(), INTERVAL 7 DAY) |
+-----+-----+
| 2006-05-22 | 2006-05-15 |
+-----+-----+
```

在 MySQL 5.0 以后的版本中, 可以用 1 WEEK 代替 7 DAY, 但计算的结果将是一个 DATETIME 类型, 而不再是 DATE 类型

- 如果同时需要日期和时间值, 用 DATETIME 或 TIMESTAMP 值来开始。要回答这样一个问题, “60 小时后是什么时间?”, 这么做:

```
mysql> SELECT NOW(), DATE_ADD(NOW(), INTERVAL 60 HOUR);
+-----+-----+
| NOW() | DATE_ADD(NOW(), INTERVAL 60 HOUR) |
+-----+-----+
| 2006-02-04 09:28:10 | 2006-02-06 21:28:10 |
+-----+-----+
```

- 有些时间间隔值同时包含了日期和时间值, 下面的例子在当前时刻上, 加上了 14.5 小时:

```
mysql> SELECT NOW(), DATE_ADD(NOW(), INTERVAL '14:30' HOUR_MINUTE);
+-----+-----+
| NOW() | DATE_ADD(NOW(), INTERVAL '14:30' HOUR_MINUTE) |
+-----+-----+
| 2006-02-04 09:28:31 | 2006-02-04 23:58:31 |
+-----+-----+
```

类似的, 可以在当前时刻加上 3 天 4 小时, 结果如下:

```
mysql> SELECT NOW(), DATE_ADD(NOW(), INTERVAL '3 4' DAY_HOUR);
+-----+-----+
| NOW() | DATE_ADD(NOW(), INTERVAL '3 4' DAY_HOUR) |
+-----+-----+
```

```
| 2006-02-04 09:28:38 | 2006-02-07 13:28:38
+-----+-----+
```

当时间间隔值的符号取反时, DATE_ADD() 函数的实际意义就转换为 DATE_SUB() 了, 反之亦然, 因此 DATE_ADD() 函数和 DATE_SUB() 函数是可互换的。下面对日期值 d 的两个运算是相同的:

```
DATE_ADD(d, INTERVAL -3 MONTH)
DATE_SUB(d, INTERVAL 3 MONTH)
```

也可以使用 +INTERVAL 或者 -INTERVAL 操作符对日期值进行加减运算:

```
mysql> SELECT CURDATE(), CURDATE() + INTERVAL 1 YEAR;
+-----+-----+
| CURDATE() | CURDATE() + INTERVAL 1 YEAR |
+-----+-----+
| 2006-05-22 | 2007-05-22
+-----+-----+
mysql> SELECT NOW(), NOW() - INTERVAL '1 12' DAY_HOUR;
+-----+-----+
| NOW() | NOW() - INTERVAL '1 12' DAY_HOUR |
+-----+-----+
| 2006-05-22 19:00:50 | 2006-05-21 07:00:50
+-----+-----+
```

对于 MySQL 5.0 以及以后版本, 还有一个选择来对日期值或者 date-and-time 类型值进行算术运算, 即 TIMESTAMPADD() 函数。其参数类似于 DATE_ADD() 函数, 两个函数的关系如下等式:

```
TIMESTAMPADD(unit, interval, d) = DATE_ADD(d, INTERVAL interval unit)
```

使用基本时间单位进行时间加法运算

把一个时间值和一个日期或者一个 date-and-time 类型值相加的另外一种方法是, 使用函数将所要处理的值转换为单位时间, 进行加法运算, 然后再转换回来。关于可用函数的背景知识, 请参考第 6.8 节。

使用基本时间单位进行时间加法运算, 类似于用基本时间单位计算时间差, 所不同的就是一个是加法, 一个是减法。要将一个以秒为单位的时间值和另一个 TIME 类型值相加, 首先将 TIME 值转换为以秒为单位, 使得两个操作值使用相同的时间单位, 然后相加, 最后再将结果转换回 TIME 类型值。例如, 两个小时就是 7 200 秒 ($2 \times 60 \times 60$), 下面的语句在 time_val 表中 t1 列的每一个值上加上 2 小时:

```
mysql> SELECT t1,
-> SEC_TO_TIME(TIME_TO_SEC(t1) + 7200) AS 't1 plus 2 hours'
-> FROM time_val;
+-----+-----+
| t1 | t1 plus 2 hours |
+-----+-----+
| 15:00:00 | 17:00:00
| 05:01:30 | 07:01:30
+-----+-----+
```

```
| 12:30:20 | 14:30:20 |  
+-----+-----+
```

如果所要加上的时间值也是一个 TIME 类型值，那也需要首先将其转换为以秒为单位，然后再相加。下面的语句将 time_val 表中每一行上的两个 TIME 类型列值相加：

```
mysql> SELECT t1, t2,  
-> TIME_TO_SEC(t1) + TIME_TO_SEC(t2)  
-> AS 't1 + t2 (in seconds)',  
-> SEC_TO_TIME(TIME_TO_SEC(t1) + TIME_TO_SEC(t2))  
-> AS 't1 + t2 (as TIME)'  
-> FROM time_val;  
+-----+-----+-----+-----+  
| t1      | t2      | t1 + t2 (in seconds) | t1 + t2 (as TIME) |  
+-----+-----+-----+-----+  
| 15:00:00 | 15:00:00 |          108000 | 30:00:00 |  
| 05:01:30 | 02:30:20 |          27110  | 07:31:50 |  
| 12:30:20 | 17:30:45 |          108065 | 30:01:05 |  
+-----+-----+-----+-----+
```

非常值得注意的一点是，TIME 类型值表示的是所经过的时间，而不是一天中的某个时间，因此它并不会在达到 24 小时之后被设为 0。可以从上面语句的第 1 和第 3 个输出行看到这一点。如果希望算术运算的结果是一天的某个时间，在将结果转换回 TIME 的时候，可以使用%modulo 操作以 24 小时为单位对结果取模。一天一共有 $24 \times 60 \times 60$ ，即 86 400 秒，因此将一个以秒为单位的时间值转换为一天 24 小时内的某个时间，可以采用如下 MOD() 函数，或者取模运算符%：

```
MOD(s,86400)  
s % 86400  
s MOD 86400
```



提示：TIME 类型值的取值范围是 -838:59:59 到 838:59:59（也就是 -3020399 到 3020399 秒）。但是 TIME 类型数学运算的结果可能超出这个取值范围，例如把两个 TIME 类型值相加，很容易就会得到一个超出取值范围的 TIME 值，也就无法将结果保存到数据库的一个 TIME 类型列。

下面的三个表达式是等价的。将其第一个应用到前面示例中的时间计算中时会生成下面的结果：

```
mysql> SELECT t1, t2,  
-> MOD(TIME_TO_SEC(t1) + TIME_TO_SEC(t2), 86400)  
-> AS 't1 + t2 (in seconds)',  
-> SEC_TO_TIME(MOD(TIME_TO_SEC(t1) + TIME_TO_SEC(t2), 86400))  
-> AS 't1 + t2 (as TIME)'  
-> FROM time_val;  
+-----+-----+-----+-----+  
| t1      | t2      | t1 + t2 (in seconds) | t1 + t2 (as TIME) |  
+-----+-----+-----+-----+  
| 15:00:00 | 15:00:00 |          21600  | 06:00:00 |
```

```
| 05:01:30 | 02:30:20 | 27110 | 07:31:50 |
| 12:30:20 | 17:30:45 | 21665 | 06:01:05 |
+-----+-----+-----+-----+
```

以基本单位对日期或者 date-and-time 做加法。通过把日期值或者 date-and-time 类型值转换为基本单位，你就可以对它们进行推移得到别的日期。例如，为了把一个日期向前或者向后推移一个星期（7天），可以使用 TO_DAYS() 函数和 FROM_DAYS() 函数：

```
mysql> SET @d = '2006-01-01';
mysql> SELECT @d AS date,
       -> FROM_DAYS(TO_DAYS(@d) + 7) AS 'date + 1 week',
       -> FROM_DAYS(TO_DAYS(@d) - 7) AS 'date - 1 week';
+-----+-----+-----+
| date      | date + 1 week | date - 1 week |
+-----+-----+-----+
| 2006-01-01 | 2006-01-08 | 2005-12-25 |
+-----+-----+-----+
```

如果你不在乎抛弃其中的时间部分值，TO_DAYS() 函数也可以将 DATETIME 或者 TIMESTAMP 类型值转换为（多少）天：

```
mysql> SET @dt = '2006-01-01 12:30:45';
mysql> SELECT @dt AS datetime,
       -> FROM_DAYS(TO_DAYS(@dt) + 7) AS 'datetime + 1 week',
       -> FROM_DAYS(TO_DAYS(@dt) - 7) AS 'datetime - 1 week';
+-----+-----+-----+
| datetime           | datetime + 1 week | datetime - 1 week |
+-----+-----+-----+
| 2006-01-01 12:30:45 | 2006-01-08        | 2005-12-25        |
+-----+-----+-----+
```

为了保留 DATETIME 或者 TIMESTAMP 值中的时间部分，可以使用 UNIX_TIMESTAMP() 和 FROM_UNIXTIME() 函数作为替代。下面的语句把一个 DATETIME 值向前和向后推移了一个小时（3600秒）：

```
mysql> SET @dt = '2006-01-01 09:00:00';
mysql> SELECT @dt AS datetime,
       -> FROM_UNIXTIME(UNIX_TIMESTAMP(@dt) + 3600) AS 'datetime + 1 hour',
       -> FROM_UNIXTIME(UNIX_TIMESTAMP(@dt) - 3600) AS 'datetime - 1 hour';
+-----+-----+-----+
| datetime           | datetime + 1 hour | datetime - 1 hour |
+-----+-----+-----+
| 2006-01-01 09:00:00 | 2006-01-01 10:00:00 | 2006-01-01 08:00:00 |
+-----+-----+-----+
```

前面讨论到的方法，都要求你要处理的初始值和最终结果都在 TIMESTAMP 的有效取值范围内（1970年到2037年）。

6.11 计算年龄

Calculating Ages

问题

如何求得某人的年龄。

解决方案

这个一个日期计算问题。这就是要计算两个日期之间的时间间隔，但又有些歪曲了时间间隔的本意。对于以年来计的年龄，需要考虑到开始和结束两个日期在历年中所处的位置。对于以月计的年龄，同样需要考虑到相关月份（在一年中）所处的位置，以及相关日期在当月的位置。

讨论

判断年龄是时间间隔运算的一种，但是你又不能简单的将其视为计算间隔的天数，然后除以 365。那样做是行不通的，因为闰年破坏了这样的计算（1995-03-01 到 1996-02-29 的时间间隔是 365 天，但是从年龄的角度来看，就不是一年了）。使用 365.25 作为除数要稍微精确一点，但是对于所有日期而言，仍然是不正确的。替代的做法是，求出日期之间相差的年份，然后根据相关日期在历年中所处的位置做调整（假设 Gretchen Smith 生于 1942 年 4 月 14 日，要求出 Gretchen Smith 现在的年龄，我们需要考虑到当前时刻在历年中所处的位置：在今年开始到 4 月 13 号之间，她处在某个年龄，在 4 月 13 号到年末之间她又会大一岁）。本节展示了如何实施以年或者月为单位来计算年龄的一类运算。

计算年龄的最简单的方法，就是使用 `TIMESTAMPDIFF()` 函数，因为你可以传递给它一个出生日期，一个当前日期以及你所希望使用的计量年龄的时间单位：

```
TIMESTAMPDIFF(unit,birth,current)
```

`TIMESTAMP DIFF()` 函数在计算中，对不同的月份和年份的长度以及日期值在历年中的相关位置作了必要的调整。假设我们有一个 `sibling` 表，其中记录了 Gretchen Smith 和她的兄弟 Wilbur 和 Franz 的生日：

name	birth
Gretchen	1942-04-14
Wilbur	1946-11-28
Franz	1953-03-05

使用 `TIMESTAMPDIFF()` 函数，我们可以回答如下问题：

- Smith 家的小孩今天多大？

```
mysql> SELECT name, birth, CURDATE() AS today,  
-> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS 'age in years'  
-> FROM sibling;  
+-----+-----+-----+-----+  
| name | birth | today | age in years |  
+-----+-----+-----+-----+  
| Gretchen | 1942-04-14 | 2006-05-30 | 64 |  
| Wilbur | 1946-11-28 | 2006-05-30 | 59 |
```

```
| Franz | 1953-03-05 | 2006-05-30 | 53 |  
+-----+-----+-----+-----+
```

- Franz 出生时 Gretchen 和 Wilbur 年纪多大

```
mysql> SELECT name, birth, '1953-03-05' AS 'Franz'' birthday,  
-> TIMESTAMPDIFF(YEAR,birth,'1953-03-05') AS 'age in years'  
-> FROM sibling WHERE name != 'Franz';  
+-----+-----+-----+-----+  
| name | birth | Franz' birthday | age in years |  
+-----+-----+-----+-----+  
| Gretchen | 1942-04-14 | 1953-03-05 | 10 |  
| Wilbur | 1946-11-28 | 1953-03-05 | 6 |  
+-----+-----+-----+-----+
```

前面的查询都以年为单位输出年龄，但是你喜欢的话也可以使用其他时间间隔单位。例如，以月为单位的 Smith 家的小孩的年龄可以这样来计算：

```
mysql> SELECT name, birth, CURDATE() AS today,  
-> TIMESTAMPDIFF(MONTH,birth,CURDATE()) AS 'age in months'  
-> FROM sibling;  
+-----+-----+-----+-----+  
| name | birth | today | age in months |  
+-----+-----+-----+-----+  
| Gretchen | 1942-04-14 | 2006-05-30 | 769 |  
| Wilbur | 1946-11-28 | 2006-05-30 | 714 |  
| Franz | 1953-03-05 | 2006-05-30 | 638 |  
+-----+-----+-----+-----+
```

TIMESTAMPDIFF() 函数要求 MySQL 5.0 或者以上版本。对于 MySQL 较老的版本，你可以不使用这个函数计算年龄，但是就像下面的讨论展示的，这需要更多的工作。

通常，给定一个出生日期 birth，对一个目标日期 d 的以年来计的年龄可以这样计算：

```
if (d occurs earlier in the year than birth)  
    age = YEAR(d) - YEAR(birth) - 1  
if (d occurs on or later in the year than birth)  
    age = YEAR(d) - YEAR(birth)
```

两种情况下，计算年份差别的部分都是相同的。它们之间的区别在于两个日期在历年中的相关顺序。然而，这种顺序不能使用 DAYOFYEAR() 函数来判断，因为这个函数只对两个日期所处的年份都具有相同天数的情况才有效。对于不同年份的日期，不同的日期可能具有相同的 DAYOFYEAR() 函数返回值，就像下面的语句所举的例子：

```
mysql> SELECT DAYOFYEAR('1995-03-01'), DAYOFYEAR('1996-02-29');  
+-----+-----+  
| DAYOFYEAR('1995-03-01') | DAYOFYEAR('1996-02-29') |  
+-----+-----+  
| 60 | 60 |  
+-----+-----+
```

ISO 标准的日期字符串的比较服从自然顺序的事实成为了我们此处的救兵，更准确的说，代表月和日的最右边 5 个字母也恰当的进行了比较：

```
mysql> SELECT RIGHT('1995-03-01',5), RIGHT('1996-02-29',5);
+-----+-----+
| RIGHT('1995-03-01',5) | RIGHT('1996-02-29',5) |
+-----+-----+
| 03-01           | 02-29           |
+-----+-----+
mysql> SELECT IF('02-29' < '03-01','02-29','03-01') AS earliest;
+-----+
| earliest |
+-----+
| 02-29   |
+-----+
```

也就是说，你可以像这样来进行两个日期值 d1 和 d2 在历年中哪个更早的测试：

```
RIGHT(d2,5) < RIGHT(d1,5)
```

根据测试的结果，该表达式产生的结果是 1 或者 0，因此，< 比较的结果可以用来实施基于年的年龄计算：

```
YEAR(d2) - YEAR(d1) - (RIGHT(d2,5) < RIGHT(d1,5))
```

如果你想让比较的结果的意义更为明确，可以使用返回值只为 0 或者 1 的 IF() 语句来包装比较语句：

```
YEAR(d2) - YEAR(d1) - IF(RIGHT(d2,5) < RIGHT(d1,5),1,0)
```

下面的语句示范了如何使用这个算式来计算刚进入 1975 年时，生于 1965-03-01 的某人的年龄。它展现了未经调整的以年来计的年龄，调整过的年龄，以及最终的年龄：

```
mysql> SET @birth = '1965-03-01';
mysql> SET @target = '1975-01-01';
mysql> SELECT @birth, @target,
-> YEAR(@target) - YEAR(@birth) AS 'difference',
-> IF(RIGHT(@target,5) < RIGHT(@birth,5),1,0) AS 'adjustment',
-> YEAR(@target) - YEAR(@birth)
-> - IF(RIGHT(@target,5) < RIGHT(@birth,5),1,0)
-> AS 'age';
+-----+-----+-----+-----+
| @birth      | @target      | difference | adjustment | age   |
+-----+-----+-----+-----+
| 1965-03-01 | 1975-01-01 |          10 |          1 |      9 |
+-----+-----+-----+-----+
```

让我们试试在 sibling 表上使用这个 age-in-years 算式。我们可以使用前面用 TIMESTAMPDIFF() 函数回答的问题来测试这个算式。这个算式像下面这样产生了问题的答案：

- Smith 家的孩子现在多大？

```

mysql> SELECT name, birth, CURDATE() AS today,
->   YEAR(CURDATE()) - YEAR(birth)
->   - IF(RIGHT(CURDATE(),5) < RIGHT(birth,5),1,0)
->   AS 'age in years'
->   FROM sibling;
+-----+-----+-----+-----+
| name    | birth     | today      | age in years |
+-----+-----+-----+-----+
| Gretchen | 1942-04-14 | 2006-05-30 |          64 |
| Wilbur   | 1946-11-28 | 2006-05-30 |          59 |
| Franz    | 1953-03-05 | 2006-05-30 |          53 |
+-----+-----+-----+-----+

```

- 当 Franz 出生时 Gretchen 和 Wilbur 多大?

```

mysql> SELECT name, birth, '1953-03-05' AS 'Franz'' birthday',
->   YEAR('1953-03-05') - YEAR(birth)
->   - IF(RIGHT('1953-03-05',5) < RIGHT(birth,5),1,0)
->   AS 'age in years'
->   FROM sibling WHERE name != 'Franz';
+-----+-----+-----+-----+
| name    | birth     | Franz' birthday | age in years |
+-----+-----+-----+-----+
| Gretchen | 1942-04-14 | 1953-03-05 |          10 |
| Wilbur   | 1946-11-28 | 1953-03-05 |           6 |
+-----+-----+-----+-----+

```

当自然地实施这些计算的时候，务必记住，如果使用日期字符串的 *MM-DD* (月-日) 部分进行比较要产生正确的结果，你必须使用 ISO 标准的日期值例如 1987-07-01，而不是接近 ISO 标准的值，例如 1987-7-1。例如，下面的比较对于单纯的字符串比较而言是正确的，但是对于日期比较来说是错误的：

```

mysql> SELECT RIGHT('1987-7-1',5) < RIGHT('1987-10-01',5);
+-----+
| RIGHT('1987-7-1',5) < RIGHT('1987-10-01',5) |
+-----+
|                               0 |
+-----+

```

由于第一个日期值的月和日部分第一位缺少 0，使得基于字符串子串的日期比较失败。如果读者需要使用非 ISO 格式日期值，请参考 6.18 节。

以月为单位计算年龄的公式与以年为单位来计算的公式类似，不同之处仅在于将以年为单位的年龄乘以 12，然后加上所差的月份，最后根据相关日期在一个月中所处的位置进行调整。这种情况下，我们需要使用每个日期分解中的月和日部分，因此需要使用 MONTH() 和 DAYOFMONTH() 函数，而非直接比较日期值字符串中的 *MM-DD* 部分。Smith 家的孩子们的年龄如果以月份来计可以这样计算：

```

mysql> SELECT name, birth, CURDATE() AS today,
->   (YEAR(CURDATE()) - YEAR(birth)) * 12
->   + (MONTH(CURDATE()) - MONTH(birth))

```

```

-> - IF(DAYOFMONTH(CURDATE()) < DAYOFMONTH(birth), 1, 0)
-> AS 'age in months'
-> FROM sibling;
+-----+-----+-----+-----+
| name | birth | today | age in months |
+-----+-----+-----+-----+
| Gretchen | 1942-04-14 | 2006-05-30 | 769 |
| Wilbur | 1946-11-28 | 2006-05-30 | 714 |
| Franz | 1953-03-05 | 2006-05-30 | 638 |
+-----+-----+-----+-----+

```

6.12 将一个日期和时间值切换到另一个时区

Shifting a Date-and-Time Value to a Different Time Zone

问题

如果你有一个 date-and-time 值，如何得到它在另外一个时区的对应值。例如，你将要跟世界上其他地区的人举行电话会议，并且你需要通知参加者在他们自己的时区将在几点举行会议。

解决方案

使用 CONVERT-TZ() 函数。

讨论

CONVERT-TZ() 函数接收三个参数：一个 date-and-time 值，两个时区指示器。该函数在第一个时区内解释 date-and-time 值，然后将这个值转换到第二个时区，并按此产生最终结果。

假设我住在美国的芝加哥，我需要和住在世界上其他地方的一些人开会。下面的表列出了每一个参加会议的人所处的位置，以及他们所处的时区名称。

地点	时区名
Chicago, Illinois, U.S.	US/Central
Berlin, Germany	Europe/Berlin
London, United Kingdom	Eur ope/London
Edmonton, Alberta, Canada	America/Edmonton
Brisbane, Australia	Australia/Brisbane

如果会议将在我所处时区的 2006 年 11 月 23 日上午 9 点举行，那么对于其他与会者而言，应该是什么时间？下面的语句使用 CONVERT-TZ() 函数计算对应每一个时区的时间：

```
mysql> SET @dt = '2006-11-23 09:00:00';
mysql> SELECT @dt AS Chicago,
-> CONVERT_TZ(@dt,'US/Central','Europe/Berlin') AS Berlin,
-> CONVERT_TZ(@dt,'US/Central','Europe/London') AS London,
-> CONVERT_TZ(@dt,'US/Central','America/Edmonton') AS Edmonton,
-> CONVERT_TZ(@dt,'US/Central','Australia/Brisbane') AS Brisbane\G
***** 1. row *****
Chicago: 2006-11-23 09:00:00
Berlin: 2006-11-23 16:00:00
London: 2006-11-23 15:00:00
Edmonton: 2006-11-23 08:00:00
Brisbane: 2006-11-24 01:00:00
```

但愿 Brisbane 的参加者不会介意在午夜以后起床（开会）。

在上例中使用了时区名，这要求在你的 mysql 数据库中有支持时区名的时区表（关于建立时区表，请参考《MySQL Reference Manual》）。如果你不能使用时区名，你可以使用相关时区相对于 UTC 的数值。这有些不太好处理，因为你可能需要考虑夏令时问题。使用数值时区的对应语句如下：

```
mysql> SELECT @dt AS Chicago,
-> CONVERT_TZ(@dt,'-06:00','+01:00') AS Berlin,
-> CONVERT_TZ(@dt,'-06:00','+00:00') AS London,
-> CONVERT_TZ(@dt,'-06:00','-07:00') AS Edmonton,
-> CONVERT_TZ(@dt,'-06:00','+10:00') AS Brisbane\G
***** 1. row *****
Chicago: 2006-11-23 09:00:00
Berlin: 2006-11-23 16:00:00
London: 2006-11-23 15:00:00
Edmonton: 2006-11-23 08:00:00
Brisbane: 2006-11-24 01:00:00
```

6.13 找出每月的第一天、最后一天或者天数

Finding the First Day, Last Day, or Length of a Month

问题

给定一个日期，你想要知道这个日期所处月份的第一天或者最后一天的日期，或者是相关月份之前或者之后 n 个月的第一天或者最后一天的日期。与之相关的问题是如何知道一个月有多少天。

解决方案

要得到一个月的第一天的日期，可以使用日期推移（一个日期算术元算程序）。为了得到最后一天的日期，使用 LAST_DAY() 函数。为了计算一个月中有多少天，找出当月最后一天的日期，并将它作为 DAYOFMONTH() 函数的一个参数值。

讨论

有时你有一个日期值，并且你希望得到与之并没有固定关系的日期值。例如，当月的第一天或者最后一天的日期，与当前日期之间的所差的天数并不是一个固定值。

为了找到所给日期所处月份的第一天的日期，将该日期值向后推移 DAYOFMONTH()-1 天。

```
mysql> SELECT d, DATE_SUB(d, INTERVAL DAYOFMONTH(d)-1 DAY) AS '1st of month'  
-> FROM date_val;  
+-----+-----+  
| d      | 1st of month |  
+-----+-----+  
| 1864-02-28 | 1864-02-01 |  
| 1900-01-15 | 1900-01-01 |  
| 1987-03-05 | 1987-03-01 |  
| 1999-12-31 | 1999-12-01 |  
| 2000-06-04 | 2000-06-01 |  
+-----+-----+
```

通常情况下，为了找到一个与给定日期所处月份间隔 n 个月的某一个月的第一天的日期，首先计算该日期所处月份的第一天的日期，然后将得到的日期值推移 n 个月：

```
DATE_ADD(DATE_SUB(d, INTERVAL DAYOFMONTH(d)-1 DAY), INTERVAL n MONTH)
```

例如，为了找到一个给定日期所处月的前一个和后一个月的第一天， n 为 -1 和 1：

```
mysql> SELECT d,  
-> DATE_ADD(DATE_SUB(d, INTERVAL DAYOFMONTH(d)-1 DAY), INTERVAL -1 MONTH)  
-> AS '1st of previous month',  
-> DATE_ADD(DATE_SUB(d, INTERVAL DAYOFMONTH(d)-1 DAY), INTERVAL 1 MONTH)  
-> AS '1st of following month'  
-> FROM date_val;  
+-----+-----+-----+  
| d          | 1st of previous month | 1st of following month |  
+-----+-----+-----+  
| 1864-02-28 | 1864-01-01           | 1864-03-01           |  
| 1900-01-15 | 1899-12-01           | 1900-02-01           |  
| 1987-03-05 | 1987-02-01           | 1987-04-01           |  
| 1999-12-31 | 1999-11-01           | 2000-01-01           |  
| 2000-06-04 | 2000-05-01           | 2000-07-01           |  
+-----+-----+-----+
```

要找到一个给定的日期所处月份的最后一天，相对要简单一些，因为有一个函数可以使用：

```
mysql> SELECT d, LAST_DAY(d) AS 'last of month'  
-> FROM date_val;  
+-----+-----+  
| d          | last of month |  
+-----+-----+  
| 1864-02-28 | 1864-02-29 |  
| 1900-01-15 | 1900-01-31 |
```

```
+-----+-----+
| 1987-03-05 | 1987-03-31      |
| 1999-12-31 | 1999-12-31      |
| 2000-06-04 | 2000-06-30      |
+-----+-----+
```

通常情况下，为了找到与一个给定日期间隔 n 个月的某一个月份的最后一天，首先将该日期推移 n 个月，然后将推移的结果作为 `LAST_DAY()` 函数的参数：

```
LAST_DAY(DATE_ADD(d, INTERVAL n MONTH))
```

例如，为了找到一个给定日期所处月份的前一个和后一个月的最后一天， n 为 -1 和 1：

```
mysql> SELECT d,
-> LAST_DAY(DATE_ADD(d, INTERVAL -1 MONTH))
->   AS 'last of previous month',
-> LAST_DAY(DATE_ADD(d, INTERVAL 1 MONTH))
->   AS 'last of following month'
-> FROM date_val;
+-----+-----+-----+
| d      | last of previous month | last of following month |
+-----+-----+-----+
| 1864-02-28 | 1864-01-31           | 1864-03-31           |
| 1900-01-15 | 1899-12-31           | 1900-02-28           |
| 1987-03-05 | 1987-02-28           | 1987-04-30           |
| 1999-12-31 | 1999-11-30           | 2000-01-31           |
| 2000-06-04 | 2000-05-31           | 2000-07-31           |
+-----+-----+-----+
```

为了计算一个月有多少天，首先用 `LAST_DAY()` 函数计算当月最后一天的日期，然后使用 `DAYOFMONTH()` 函数从结果中分解出 day-of-month 部分：

```
mysql> SELECT d, DAYOFMONTH(LAST_DAY(d)) AS 'days in month' FROM date_val;
+-----+-----+
| d      | days in month |
+-----+-----+
| 1864-02-28 |        29 |
| 1900-01-15 |        31 |
| 1987-03-05 |        31 |
| 1999-12-31 |        31 |
| 2000-06-04 |        30 |
+-----+-----+
```

参考

在本章下面的内容 6.17 节中将讨论在应用程序中不使用 SQL 如何计算一个月的长度（关键的技巧是处理闰年）。

6.14 通过子串替换来计算日期

Calculating Dates by Substring Replacement

问题

给定一个日期 date，因为你知道另外一个日期值与之有一些相同的部分，由此你想利用 date 产生另外一個日期值。

解决方案

把一个日期或者时间值当作一个字符串，然后直接对其中的一些部分进行替换。

讨论

有些情况下，你可以使用字符串子串替换代替日期算术来计算日期值。例如，你可以将一个日期 date 的“日”部分通过字符串替换的方法设为 01，从而得到当月第一天的日期。也可以使用 DATE_FORMAT() 或者 CONCAT() 函数来实现：

```
mysql> SELECT d,
    -> DATE_FORMAT(d, '%Y-%m-01') AS method1,
    -> CONCAT(YEAR(d), '-', LPAD(MONTH(d), 2, '0'), '-01') AS method2
    -> FROM date_val;
+-----+-----+-----+
| d      | method1   | method2   |
+-----+-----+-----+
| 1864-02-28 | 1864-02-01 | 1864-02-01 |
| 1900-01-15 | 1900-01-01 | 1900-01-01 |
| 1987-03-05 | 1987-03-01 | 1987-03-01 |
| 1999-12-31 | 1999-12-01 | 1999-12-01 |
| 2000-06-04 | 2000-06-01 | 2000-06-01 |
+-----+-----+-----+
```

通过字符串替换的技巧，也可以产生一年中的某个特定日期。对于元旦（1 月 1 日），将一个日期中的月和日部分都替换为 01：

```
mysql> SELECT d,
    -> DATE_FORMAT(d, '%Y-01-01') AS method1,
    -> CONCAT(YEAR(d), '-01-01') AS method2
    -> FROM date_val;
+-----+-----+-----+
| d      | method1   | method2   |
+-----+-----+-----+
| 1864-02-28 | 1864-01-01 | 1864-01-01 |
| 1900-01-15 | 1900-01-01 | 1900-01-01 |
| 1987-03-05 | 1987-01-01 | 1987-01-01 |
| 1999-12-31 | 1999-01-01 | 1999-01-01 |
| 2000-06-04 | 2000-01-01 | 2000-01-01 |
+-----+-----+-----+
```

对于圣诞节，则将月和日部分分别替换为 12 和 25：

```

mysql> SELECT d,
-> DATE_FORMAT(d, '%Y-12-25') AS method1,
-> CONCAT(YEAR(d), '-12-25') AS method2
-> FROM date_val;
+-----+-----+-----+
| d      | method1   | method2   |
+-----+-----+-----+
| 1864-02-28 | 1864-12-25 | 1864-12-25 |
| 1900-01-15 | 1900-12-25 | 1900-12-25 |
| 1987-03-05 | 1987-12-25 | 1987-12-25 |
| 1999-12-31 | 1999-12-25 | 1999-12-25 |
| 2000-06-04 | 2000-12-25 | 2000-12-25 |
+-----+-----+-----+

```

将字符串替换和日期推移结合起来，对其他年份的圣诞节日期进行同样的操作。下面的语句展现了两种不同的方式来计算两年以后的圣诞节日期。第一种方法首先得到今年的圣诞节日期，然后将其推移到两年以后。第二种方法首先将当前日期向前推移两年，然后根据推移的结果找到对应年份的圣诞节日期：

```

mysql> SELECT CURDATE(),
-> DATE_ADD(DATE_FORMAT(CURDATE(), '%Y-12-25'), INTERVAL 2 YEAR)
-> AS method1,
-> DATE_FORMAT(DATE_ADD(CURDATE(), INTERVAL 2 YEAR), '%Y-12-25')
-> AS method2;
+-----+-----+-----+
| CURDATE() | method1   | method2   |
+-----+-----+-----+
| 2006-05-22 | 2008-12-25 | 2008-12-25 |
+-----+-----+-----+

```

6.15 计算某个日期为星期几

Finding the Day of the Week for a Date

问题

你想知道一个日期是星期几。

解决方案

使用 DAYNAME() 函数。

讨论

为了得到一个给定日期对应的星期几的名称，使用 DAYNAME() 函数。

```

mysql> SELECT CURDATE(), DAYNAME(CURDATE());
+-----+-----+
| CURDATE() | DAYNAME(CURDATE()) |
+-----+-----+
| 2006-05-22 | Monday      |
+-----+-----+

```

DAYNAME() 函数在和其他与日期相关的技巧联合使用时常常十分有用。例如，为了确定一个月的第一天是星期几，使用 6.13 节中讨论的 first-of-month 表达式作为 DAYNAME() 函数的参数：

```
mysql> SET @d = CURDATE();
mysql> SET @first = DATE_SUB(@d, INTERVAL DAYOFMONTH(@d)-1 DAY);
mysql> SELECT @d AS 'starting date',
-> @first AS '1st of month date',
-> DAYNAME(@first) AS '1st of month day';
+-----+-----+-----+
| starting date | 1st of month date | 1st of month day |
+-----+-----+-----+
| 2006-05-22    | 2006-05-01      | Monday           |
+-----+-----+-----+
```

6.16 查出给定某周的某天的日期

Finding Dates for Any Weekday of a Given Week

问题

你想计算出一个给定日期所处星期中每一个周历日（weekday）的日期。例如，你想要知道与 2006-07-09 属于同一个星期的星期二的那一天的日期。

解决方案

这是一个日期推移程序。计算出给定日期 date 是所处星期的第几天以及其与所要计算的那一天之间相差的天数，然后以此对 date 做推移。

讨论

这一节和下一节描述了，当目标日期由一周的第几天这样的形式给出时，如何将一个日期转换到目标日期。为了解决这个问题，你需要知道 day-of-week 值。假设你从一个目标日期 2006-07-09 开始。如果你想知道与之处于同一个星期的星期二的日期，这个计算取决于 2006-07-09 是星期几。如果是它是星期一，你加上一天就得到 2006-07-10，但是如果它是星期三，你减去一天得到 2006-07-08。

MySQL 在这里提供了两个十分有用的函数。DAYOFWEEK() 将星期天作为一个星期的第一天，返回值由 1 到 7 分别代表星期天到星期六。WEEKDAY() 将星期一作为一个星期的第一天，返回值由 0 到 6 分别代表星期一到星期天。（本节所举例子使用 DAYOFWEEK() 函数）。其他的 day-of-week 运算涉及确定（一个星期中）每一天的名字，这种情况可以使用 DAYNAME() 函数。

从一个星期中的一天来确定另一天的计算取决于（计算工程中的）源日期以及目标日期。笔者发现最简单的一种方法是，首先将引用的日期推移到一个相对于一个星期的起始点比较明确的日期，然后再做进一步的推移：

- 以 DAYOFWEEK() 函数返回值将引用日期向后推移，这样总是得到上一个星期的星期六的日期。
- 将所得到的上一个星期六的日期推移一天得到星期天的日期，推移两天得到星期一的日期，以此类推。

在 SQL 中，这一类操作可以表述如下，对一个日期 d ，当 n 取值从 1 到 7 得到从星期天到星期六的日期：

```
DATE_ADD(DATE_SUB(d, INTERVAL DAYOFWEEK(d) DAY), INTERVAL n DAY)
```

这个表达式将“推移到星期六”和“向前推移”的步骤分解为单独的操作，但是由于 DATE_SUB() 和 DATE_ADD() 函数使用的时间间隔单位都是“天”，该表达式可以简化为一个 DATE_ADD() 表达式：

```
DATE_ADD(d, INTERVAL n - DAYOFWEEK(d) DAY)
```

如果将这一公式应用于 date_val 表中的日期值，使用一个变量 n ， $n=1$ 代表星期天， $n=7$ 代表星期六，来查找一个星期的第一天和最后一天，我们得到如下结果：

```
mysql> SELECT d, DAYNAME(d) AS day,
-> DATE_ADD(d, INTERVAL 1 - DAYOFWEEK(d) DAY) AS Sunday,
-> DATE_ADD(d, INTERVAL 7 - DAYOFWEEK(d) DAY) AS Saturday
-> FROM date_val;
+-----+-----+-----+-----+
| d      | day    | Sunday | Saturday |
+-----+-----+-----+-----+
| 1864-02-28 | Sunday | 1864-02-28 | 1864-03-05 |
| 1900-01-15 | Monday | 1900-01-14 | 1900-01-20 |
| 1987-03-05 | Thursday | 1987-03-01 | 1987-03-07 |
| 1999-12-31 | Friday | 1999-12-26 | 2000-01-01 |
| 2000-06-04 | Sunday | 2000-06-04 | 2000-06-10 |
+-----+-----+-----+-----+
```

如果你想知道一个星期内与目标日期相关的一些 weekday 的日期，需要对上面的处理过程做一些修改。首先，确定所要求的 weekday 在目标日期所在星期中的日期。然后将结果推移到所求的星期内。

计算另外一个星期中的某一个天的日期的问题，可以分解为一个一周内日期推移（使用上述公式）和星期平移。这两个运算可以按任意顺序执行，因为不管是否先将相关日期推移到另外一个星期内，在一个星期内需要的推移次数是一样的。例如，使用前面的公式计算本星期内星期三的日期， n 为 4。为了计算两个星期前的星期三的日期，你可以首先如下做星期内推移：

```
mysql> SET @target =
-> DATE_SUB(DATE_ADD(CURDATE(), INTERVAL 4 - DAYOFWEEK(CURDATE()) DAY),
-> INTERVAL 14 DAY);
mysql> SELECT CURDATE(), @target, DAYNAME(@target);
+-----+-----+-----+
| CURDATE() | @target | DAYNAME(@target) |
+-----+-----+-----+
```

```
| 2006-05-22 | 2006-05-10 | Wednesday |  
+-----+-----+-----+
```

或者你也可以先做星期推移：

```
mysql> SET @target =  
-> DATE_ADD(DATE_SUB(CURDATE(), INTERVAL 14 DAY),  
-> INTERVAL 4-DAYOFWEEK(CURDATE()) DAY);  
mysql> SELECT CURDATE(), @target, DAYNAME(@target);  
+-----+-----+-----+  
| CURDATE() | @target | DAYNAME(@target) |  
+-----+-----+-----+  
| 2006-05-22 | 2006-05-10 | Wednesday |  
+-----+-----+-----+
```

某些应用程序需要判定第 n 个特定周日的日期。例如，如果你管理的工资表以每月第二或第四个星期四作为发薪日，你就需要知道具体的日期。对任一给定月份，一种方法是从该月的第一天开始进行推移。这种方法很容易推算出那一周（本月底一周）中星期四的日期；其中的技巧是计算出需要推移几个星期，才能得到第二和第四个星期四。如果该月的第一天是从星期天到星期四的任意一天，你可以推移一个星期得到第二个星期四。如果该月的第一天是星期五或者之后的某一个周日，你就需要推移两个星期。第四个星期四就是之后的两个星期。

下面的 Perl 程序实现的是查询 2007 年所有发薪日的运算。它执行一个循环来计算这一年中每个月第一天的日期。对每一个月，执行一个语句来判断第二和第四个星期四的日期：

```
my $year = 2007;  
print "MM/CCYY 2nd Thursday 4th Thursday\n";  
foreach my $month (1..12)  
{  
    my $first = sprintf ("%04d-%02d-01", $year, $month);  
    my ($thu2, $thu4) = $dbh->selectrow_array (qq{  
        SELECT  
            DATE_ADD(  
                DATE_ADD(?, INTERVAL 5-DAYOFWEEK(?) DAY),  
                INTERVAL IF(DAYOFWEEK(?) <= 5, 7, 14) DAY),  
            DATE_ADD(  
                DATE_ADD(?, INTERVAL 5-DAYOFWEEK(?) DAY),  
                INTERVAL IF(DAYOFWEEK(?) <= 5, 21, 28) DAY)  
        }, undef, $first, $first, $first, $first, $first, $first);  
    printf "%02d/%04d %s %s\n", $month, $year, $thu2, $thu4;  
}
```

程序输出结果如下：

MM/CCYY	2nd Thursday	4th Thursday
01/2007	2007-01-11	2007-01-25
02/2007	2007-02-08	2007-02-22
03/2007	2007-03-08	2007-03-22

04/2007	2007-04-12	2007-04-26
05/2007	2007-05-10	2007-05-24
06/2007	2007-06-14	2007-06-28
07/2007	2007-07-12	2007-07-26
08/2007	2007-08-09	2007-08-23
09/2007	2007-09-13	2007-09-27
10/2007	2007-10-11	2007-10-25
11/2007	2007-11-08	2007-11-22
12/2007	2007-12-13	2007-12-27

6.17 执行闰年计算

Performing Leap Year Calculations

问题

你需要执行一个与闰年相关的日期计算。例如，基于一个日期是否处于闰年的一个月或者一年的天数。

解决方案

需要知道如何检测某一年是否是闰年，并将结果作为计算过程的一个要素。

讨论

不是所有的月份都有相同的天数这一事实使得日期计算复杂，另外一个头疼的问题是闰年的二月（比其他年份）多一天。下面的方法展示了如何确定一个给定的日期是否处于闰年，以及如何在计算一年或者一月的过程中考虑闰年的情况。

确定一个日期是否处于闰年

为了确定一个日期是否处于闰年，使用 `YEAR()` 函数获取（日期）的年部分值，并且对结果进行检测。一个最常用的检测闰年的方法是“除以 4”，也就是如下使用取模操作符%：

`YEAR(d) % 4 = 0`

然而，从学术上来说这个测试并不正确（例如，1900 可以被 4 整除，但它并不是闰年）。对于某一年如果它是闰年，必须同时符合以下两个条件：

- 必须能够被 4 整除；
- 世纪元年除非同时能被 400 整除，不能用被 100 整除来判断。

第二个限制条件的意思是除了每四个世纪的世纪元年之外，其他的世纪元年都不是闰年。SQL 中你可以如下表达这些限制条件：

```
(YEAR(d) % 4 = 0) AND ((YEAR(d) % 100 != 0) OR (YEAR(d) % 400 = 0))
```

同时使用最常用的检测闰年的方法以及完整的检测过程来检测 date_val 表中的所有日期，产生如下结果：

```
mysql> SELECT
-> d,
-> YEAR(d) % 4 = 0
-> AS 'rule-of-thumb test',
-> (YEAR(d) % 4 = 0) AND ((YEAR(d) % 100 != 0) OR (YEAR(d) % 400 = 0))
-> AS 'complete test'
-> FROM date_val;
+-----+-----+-----+
| d      | rule-of-thumb test | complete test |
+-----+-----+-----+
| 1864-02-28 |           1 |          1 |
| 1900-01-15 |           1 |          0 |
| 1987-03-05 |           0 |          0 |
| 1999-12-31 |           0 |          0 |
| 2000-06-04 |           1 |          1 |
+-----+-----+-----+
```

如同你所看到的，这两个检测并不总是产生一样的结果。特别要指出的是，最常用的测试方法对于 1900 年产生错误结果，而完整测试的结果是正确的，因为它符合世纪元年的约束。

注意，因为闰年的完整测试需要检测世纪值，因此（检测过程）需要 4 位数的年份值。2 位数的年份值对于世纪值是不明确的，使得评估世纪元年约束变得不可能。

如果你在一个应用程序内处理日期值，你可以使用程序语言 API 代替 SQL 来实现闰年测试。取出一个日期值字符串的前 4 位数字得到相应的年份，然后对其进行检测。如果所用的编程语言提供字符串到数值的自动转换，这（检测过程）就简单了。否则，你需要在进行测试之前显式地将年份字符串转化为相应的数值。

Perl、PHP:

```
$year = substr ($date, 0, 4);
$is_leap = ($year % 4 == 0) && ($year % 100 != 0 || $year % 400 == 0);
```

Ruby:

```
year = date[0..3].to_i
is_leap = (year.modulo(4) == 0) &&
          (year.modulo(100) != 0 || year.modulo(400) == 0)
```

Python:

```
year = int (date[0:4])
is_leap = (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)
```

Java:

```
int year = Integer.valueOf (date.substring (0, 4)).intValue ();
boolean is_leap = (year % 4 == 0) && (year % 100 != 0 || year % 400 == 0);
```

在日期长度计算中使用闰年测试

一年通常有 365 天，但是闰年会多一天。为了决定某个日期所处年份的天数，你可以使用刚刚展示的闰年检测方法来断定是否需要加上一天：

```
$year = substr ($date, 0, 4);
$is_leap = ($year % 4 == 0) && ($year % 100 != 0 || $year % 400 == 0);
$days_in_year = ($is_leap ? 366 : 365);
```

为了在 SQL 中计算一年的天数，计算出当年最后一天的日期，然后将其（作为参数）传递给 DAYOFYEAR()：

```
mysql> SET @d = '2006-04-13';
mysql> SELECT DAYOFYEAR(DATE_FORMAT(@d, '%Y-12-31'));
+-----+
| DAYOFYEAR(DATE_FORMAT(@d, '%Y-12-31')) |
+-----+
| 365 |
+-----+
mysql> SET @d = '2008-04-13';
mysql> SELECT DAYOFYEAR(DATE_FORMAT(@d, '%Y-12-31'));
+-----+
| DAYOFYEAR(DATE_FORMAT(@d, '%Y-12-31')) |
+-----+
| 366 |
+-----+
```

在月份天数的计算中使用闰年检测

在 6.13 节中，我们讨论了在 SQL 中如何使用 LAST_DAY() 函数来确定一个月的天数。

对于提供了 API 的程序语言，你可以使用一个非 SQL 函数，通过一个给定的 ISO 格式日期参数，得到该日期所处月份的天数。除了二月份之外，这都是直截了当的，对于二月份，函数需要根据相关年份是否是闰年决定返回 29 或者 28。下面是 Ruby 版本的函数：

```
def days_in_month(date)
  year = date[0..3].to_i
  month = date[5..6].to_i # 月份，从 1 开始
  days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
  days = days_in_month[month-1]
  is_leap = (year.modulo(4) == 0) &&
             (year.modulo(100) != 0 || year.modulo(400) == 0)

  # 闰年二月要加上一天
  days += 1 if month == 2 && is_leap
  return days
end
```

6.18 接近但不是 ISO 格式的日期格式

Canonizing Not-Quite-ISO Date Strings

问题

一个日期所用格式接近但是并不是 ISO 格式。

解决方案

通过将一个日期值（作为参数）传递给一个始终返回一个 ISO 格式日期值的函数来规范化一个日期值。

讨论

在先前的章节 6.7 节中，我们遇到了使用 CONCAT() 函数合成的日期值可能并非完全是 ISO 格式的问题。例如，下面的语句产生的一个月第一天日期值中月份部分可能只有 1 位数字：

```
mysql> SELECT d, CONCAT(YEAR(d), '-', MONTH(d), '-01') FROM date_val;
+-----+-----+
| d      | CONCAT(YEAR(d), '-', MONTH(d), '-01') |
+-----+-----+
| 1864-02-28 | 1864-2-01          |
| 1900-01-15 | 1900-1-01          |
| 1987-03-05 | 1987-3-01          |
| 1999-12-31 | 1999-12-01         |
| 2000-06-04 | 2000-6-01          |
+-----+-----+
```

在那一节中，展现了一个使用 LPAD() 函数的技巧来保证月份值含有 2 位数字：

```
mysql> SELECT d, CONCAT(YEAR(d), '-', LPAD(MONTH(d), 2, '0'), '-01') FROM date_val;
+-----+-----+
| d      | CONCAT(YEAR(d), '-', LPAD(MONTH(d), 2, '0'), '-01') |
+-----+-----+
| 1864-02-28 | 1864-02-01          |
| 1900-01-15 | 1900-01-01          |
| 1987-03-05 | 1987-03-01          |
| 1999-12-31 | 1999-12-01          |
| 2000-06-04 | 2000-06-01          |
+-----+-----+
```

另外一种将近似于 ISO 格式日期值标准化的方法，是在产生 ISO 格式日期的表达式中对其进行处理。对于一个日期值 d，下面任意一个表达式都可以实现这个目标：

```
DATE_ADD(d, INTERVAL 0 DAY)
d + INTERVAL 0 DAY
FROM_DAYS(TO_DAYS(d))
STR_TO_DATE(d, '%Y-%m-%d')
```

例如，`CONCAT()`操作产生的非 ISO 格式的结果可以通过如下几种不同的方法转换为 ISO 格式：

```
mysql> SELECT
-> CONCAT(YEAR(d), '-', MONTH(d), '-01') AS 'non-ISO',
-> DATE_ADD(CONCAT(YEAR(d), '-', MONTH(d), '-01'), INTERVAL 0 DAY) AS 'ISO 1',
-> CONCAT(YEAR(d), '-', MONTH(d), '-01') + INTERVAL 0 DAY AS 'ISO 2',
-> FROM_DAYS(TO_DAYS(CONCAT(YEAR(d), '-', MONTH(d), '-01'))) AS 'ISO 3',
-> STR_TO_DATE(CONCAT(YEAR(d), '-', MONTH(d), '-01'), '%Y-%m-%d') AS 'ISO 4'
-> FROM date_val;
```

non-ISO	ISO 1	ISO 2	ISO 3	ISO 4
1864-2-01	1864-02-01	1864-02-01	1864-02-01	1864-02-01
1900-1-01	1900-01-01	1900-01-01	1900-01-01	1900-01-01
1987-3-01	1987-03-01	1987-03-01	1987-03-01	1987-03-01
1999-12-01	1999-12-01	1999-12-01	1999-12-01	1999-12-01
2000-6-01	2000-06-01	2000-06-01	2000-06-01	2000-06-01

参考

第 10 章讨论了日期验证过程中的闰年计算。

6.19 将日期或时间当成数值

Treating Dates or Times as Numbers

问题

你想把一个日期字符串当作一个数值。

解决方案

实施一个字符串-数值转化。

讨论

在许多情况下，在 MySQL 中将一个日期或者时间值当作一个数值都是可能的。如果你想进行一个日期值相关的算术运算，这（字符串-数值转化）有时是有帮助的。为了将一个时间转化为数值形式，可以通过对其加 0 或者在一个数值上下文中使用时间值。

```
mysql> SELECT t1,
-> t1+0 AS 't1 as number',
-> FLOOR(t1) AS 't1 as number',
-> FLOOR(t1/10000) AS 'hour part'
-> FROM time_val;
```

t1	t1 as number	t1 as number	hour part
15:00:00	150000	150000	15
05:01:30	50130	50130	5

```
| 12:30:20 |          123020 |          123020 |          12 |
+-----+-----+-----+-----+
```

同一类型的转化也可以作用于日期或者 date-and-time 类型。对于 DATETIME 类型列，转化过程产生一个小数部分。如果小数部分不需要，使用 FLOOR() 函数去除小数部分。

```
mysql> SELECT d, d+0 FROM date_val;
+-----+-----+
| d      | d+0    |
+-----+-----+
| 1864-02-28 | 18640228 |
| 1900-01-15 | 190000115 |
| 1987-03-05 | 19870305 |
| 1999-12-31 | 19991231 |
| 2000-06-04 | 20000604 |
+-----+
mysql> SELECT dt, dt+0, FLOOR(dt+0) FROM datetime_val;
+-----+-----+-----+
| dt            | dt+0          | FLOOR(dt+0)   |
+-----+-----+-----+
| 1970-01-01 00:00:00 | 19700101000000.000000 | 19700101000000 |
| 1987-03-05 12:30:15 | 19870305123015.000000 | 19870305123015 |
| 1999-12-31 09:00:00 | 19991231090000.000000 | 19991231090000 |
| 2000-06-04 15:45:30 | 20000604154530.000000 | 20000604154530 |
+-----+
```

通过加 0 产生的结果，与转换为基本时间单位，秒或者天，得到的结果并不完全相同。这个结果，本质上就是你通过取出原始的时间字符串中所有分隔符号得到的结果。同样的，转化为数值形式的过程只对 MySQL 解释为时间值的类型有效。如果你尝试通过加 0 将一个普通的文本字符串转换为数值，你将只能得到时间或者日期值的第一个组成元素。这样的转化将产生一个警告：

```
mysql> SELECT '1999-01-01'+0, '1999-01-01 12:30:45'+0, '12:30:45'+0;
+-----+-----+-----+
| '1999-01-01'+0 | '1999-01-01 12:30:45'+0 | '12:30:45'+0 |
+-----+-----+-----+
|           1999 |                   1999 |          12 |
+-----+-----+-----+
1 row in set, 3 warnings (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+
| Level | Code | Message          |
+-----+-----+
| Warning | 1292 | Truncated incorrect DOUBLE value: '1999-01-01' |
| Warning | 1292 | Truncated incorrect DOUBLE value: '1999-01-01 12:30:45' |
| Warning | 1292 | Truncated incorrect DOUBLE value: '12:30:45' |
+-----+-----+
```

如果你对有些函数的返回结果加 0，例如 DATE_FORMAT() 函数和 TIME_FORMAT() 函数，或者你通过 LEFT() 或者 RIGHT() 函数分解出 DATETIME 或者 TIMESTAMP 类型值的部分值，然后对结果加 0，会发生同样的事情。在 +0 的上下文中，这些函数的返回结果被当作字符

串来处理，而不是时间相关值，并且对其转化为数值只作用于这些字符串的第一个组成部分。

6.20 强制 MySQL 将字符串当作时间值

Forcing MySQL to Treat Strings as Temporal Values

问题

你希望一个字符串被时间化解释。

解决方案

在一个时间相关上下文中使用字符串，以暗示 MySQL 如何处理这些字符串。

讨论

如果你需要使 MySQL 将一个字符串处理为一个日期或者时间，在一个提供不改变字符串值的时间相关上下文的表达式中使用这些字符串。例如，你不能在一个符合 TIME 类型值格式的字符串上加 0 来实现一个 time-to-number 的转化，但是你可以使用 TIME_TO_SEC() 函数和 SEC_TO_TIME() 函数实现转化。下面的两列结果说明了这一点：

```
mysql> SELECT '12:30:45'+0, SEC_TO_TIME(TIME_TO_SEC('12:30:45'))+0;
+-----+-----+
| '12:30:45'+0 | SEC_TO_TIME(TIME_TO_SEC('12:30:45'))+0 |
+-----+-----+
|      12      |                      123045 |
+-----+-----+
```

在第二列中，时间值和秒之间的相互转换过程没有改变时间值本身，但是产生了一个时间相关的上下文，MySQL 在该上下文中将结果作为一个 TIME 类型值。对于日期值，处理过程类似，但使用的是 TO_DAYS() 函数和 FROM_DAYS() 函数。

```
mysql> SELECT '1999-01-01'+0, FROM_DAYS(TO_DAYS('1999-01-01'))+0;
+-----+-----+
| '1999-01-01'+0 | FROM_DAYS(TO_DAYS('1999-01-01'))+0 |
+-----+-----+
|      1999      |                      19990101 |
+-----+-----+
```

对于 date-and-time 类型值，你可以使用 DATE_ADD() 函数来引入一个时间相关上下文：

```
mysql> SELECT
    -> DATE_ADD('1999-01-01 12:30:45', INTERVAL 0 DAY)+0 AS 'numeric datetime';
+-----+
| numeric datetime |
+-----+
| 19990101123045 |
+-----+
```

6.21 基于时间特性来查询行

Selecting Rows Based on Their Temporal Characteristics

问题

你想基于时间约束查询行。

解决方案

在 WHERE 子句中使用一个日期或者时间条件。这可能基于一个列值与一个已知值的直接比较。或者这也可能需要一个函数作用于列值来将其转化为一种更适合进行检测的形式，例如使用 MONTH() 函数来检测一个日期的月份部分值。

讨论

前面大部分基于日期的技巧，都是通过产生日期或者时间值的语句来举例说明的。你可以在 WHERE 子句中使用相同的技巧，对查询语句得到的行做基于日期的限制。例如，你可以通过寻找早于或者晚于一个给定值，处于一个时间范围或者符合某月或者某日的时间值来查询行。

将一个日期和另一个日期进行比较

下面的语句从 date_val 表中查找早于 1900 或者在 1900 年出现的行：

```
mysql> SELECT d FROM date_val where d < '1900-01-01';
+-----+
| d   |
+-----+
| 1864-02-28 |
+-----+
mysql> SELECT d FROM date_val where d BETWEEN '1900-01-01' AND '1999-12-31';
+-----+
| d   |
+-----+
| 1900-01-15 |
| 1987-03-05 |
| 1999-12-31 |
+-----+
```

当你不知道在 WHERE 子句中你想使用的确切日期时，你常常可以通过一个表达式来计算出所要的日期。例如，为了执行一个“历史上的今天”这个语句查询一个历史表中的行来找到整整 50 年前发生的事情，可以这样：

```
SELECT * FROM history WHERE d = DATE_SUB(CURDATE(), INTERVAL 50 YEAR);
```

你在报纸上可以看到用一些列表来展示历史上的新闻事件（本质上，这样的语句就是查询到了第 n 个周年的那些历史事件）。如果你想要查询所有“历史上的今天”发生的事情，而

不仅仅是某一年的“今天”发生的事情，那么查询语句又有些不同了。这种情况下，你需要忽略年，查询与当日的月与日匹配的行。这个话题将在后面的“比较日期值与日历日（当前日期）”部分进行讨论。

日期运算对时间范围检测一样有用。例如，为了查找过去七年内的日期值，使用 DATE_SUB() 函数来计算分界日期：

```
mysql> SELECT d FROM date_val WHERE d >= DATE_SUB(CURDATE(), INTERVAL 7 YEAR);
+-----+
| d   |
+-----+
| 1999-12-31 |
| 2000-06-04 |
+-----+
```

注意 WHERE 子句中的表达式将日期列 d 分离在比较操作符的一侧。通常情况下这是一个很好的做法；如果那一列被索引化，将其分离在比较操作符的一侧使得 MySQL 更高效的执行该语句。为了说明这一点，前面的 WHERE 子句可以以一种逻辑上相同，但是 MySQL 执行效率更低的方式来书写：

```
... WHERE DATE_ADD(d, INTERVAL 7 YEAR) >= CURDATE();
```

这里，d 列被用于一个表达式内。这意味着表中的每一行都必须要被查询，以使得表达式能够被求值并测试，这使得这一列上的任何索引都无效。

有时候，并不容易重写一个比较表达式，使得一个 date 列分离在比较操作符的一侧。例如，下面的 WHERE 子句在比较操作中只使用了一个日期列值的一部分：

```
...WHERE YEAR(d) >= 1987 AND YEAR(d) <= 1991;
```

为了重写第一个比较表达式，删除 YEAR() 函数调用，并使用一个完整的日期值来代替表达式的右侧：

```
...WHERE d >= '1987-01-01' AND YEAR(d) <= 1991;
```

重写第二个比较语句需要一些技巧。你可以像第一个比较语句的处理一样，删除表达式左侧的 YEAR() 函数调用，但是不能简单的在右侧的年份后面加上-01-01。这样做会得到如下一个不正确的结果：

```
...WHERE d >= '1987-01-01' AND d <= '1991-01-01';
```

失败的原因，是从 1991-01-02 到 1991-12-31 的日期导致测试不能通过。为了正确的重写第二个比较表达式，下面的两种方式都是正确的：

```
...WHERE d >= '1987-01-01' AND d <= '1991-12-31';
...WHERE d >= '1987-01-01' AND d < '1992-01-01';
```

其他日期计算的使用常常出现于创建的行有一个生命周期的程序中。这类程序在执行一个过期操作时必须能够决定删除哪些行。你可以使用多种途径来解决这个问题：

- 在创建每一行时，在其中保存一个日期值（通过将一列设为 `TIMESTMAP` 类型或者将其值设为 `NOW()` 来实现，详细内容参考 6.5 节）。为了以后执行一个过期操作，通过比较所记录的创建日期和当期日期来确定哪些行的创建日期太老了（过期了）。例如，判定创建时间超过 n 天的行的语句如下：

```
DELETE FROM mytbl WHERE create_date < DATE_SUB(NOW(), INTERVAL n DAY);
```

- 当创建一行时，使用 `DATE_ADD()` 函数计算到期时间，将到期时间显式的保存到每一行中。对于 n 天内将会过期的行，你可以这样做：

```
INSERT INTO mytbl (expire_date, ...)  
VALUES (DATE_ADD(NOW(), INTERVAL n DAY), ...);
```

为执行这种情况下的到期操作，将到期日期与当前日期进行比较以查看哪些已经到期了：

```
DELETE FROM mytbl WHERE expire_date < NOW();
```

比较一个时间值和其他时间值

时间相关的比较类似于日期相关的比较。例如，为了找到 9 AM 到 2 PM 之间出现的时间，可以使用如下表达式：

```
...WHERE t1 BETWEEN '09:00:00' AND '14:00:00';  
...WHERE HOUR(t1) BETWEEN 9 AND 14;
```

对于一个索引化的 `TIME` 列，第一种方法会更高效。第二种方法具有不但适用于 `TIME` 类型，也适用于 `DATETIME` 和 `TIMESTAMP` 类型的特性。

比较日期值与历日（当前日期）

使用历日检测，解决一年中某个特定日期相关的问题。下面的例子使用查询生日来说明具体如何操作：

- 今天谁生日？这要求匹配一个特定的历日，因此你需要在比较时分解出月和日，但是忽略年：

```
... WHERE MONTH(d) = MONTH(CURDATE()) AND DAYOFMONTH(d) = DAYOFMONTH(CURDATE());
```

这样的语句常作用于生平传记数据，来查询一个在一年内某个特定日期出生的演员、政治家、音乐家的一个列表，等等。

使用 `DAYOFYEAR()` 函数来解决这个“在某一天”的问题看起来很自然，因为这使得查询语句很简单。但是 `DAYOFYEAR()` 函数对闰年不能正确工作。2 月 29 日的出现推移了从 3 月到 12 月的天数。

- 这个月谁生日？这种情况下，只需要对月进行检测：

```
...WHERE MONTH(d) = MONTH(CURDATE());
```

- 下个月谁生日？这里的技巧在于你不能简单的对现在的月份加 1 以得到用于限制日期匹配的月份值。在 12 月，你这样做会得到 13 月。使用下面的技巧，以确保你得到的是 1 (1 月)：

```
...WHERE MONTH(d) = MONTH(DATE_ADD(CURDATE(), INTERVAL 1 MONTH));  
...WHERE MONTH(d) = MOD(MONTH(CURDATE()), 12)+1;
```

排序查询结果

Sorting Query Results

7.0 引言

Introduction

本章涵盖了排序这一非常重要的操作，用来控制如何显示 MySQL 中 SELECT 语句的查询结果。排序通过在查询中增加一条 ORDER BY 子句来实现。如果没有这条子句，MySQL 将以任意顺序返回查询的行结果，因此排序让无序的查询结果具有顺序同时让查询结果更加便于理解和检查。（当你使用 GROUP BY 子句时，排序也将被默认执行，见 8.13 节。）

你能够按照如下几条途径排序查询结果的行：

- 使用单独的一列，综合的几列，或者是列的一部分；
- 使用升序或者降序；
- 使用表达式的结果；
- 使用案例敏感性或案例非敏感性排序；
- 使用临时顺序。

本章在几处使用 driver_log 作为范例；它包含了记录一组卡车司机日常行驶里程日志的数据列，如下所示：

```
mysql> SELECT * FROM driver_log;
+-----+-----+-----+-----+
| rec_id | name  | trav_date | miles |
+-----+-----+-----+-----+
|     1  | Ben   | 2006-08-30 |  152  |
|     2  | Suzi  | 2006-08-29 |  391  |
|     3  | Henry | 2006-08-29 |  300  |
|     4  | Henry | 2006-08-27 |   96  |
|     5  | Ben   | 2006-08-29 |  131  |
|     6  | Henry | 2006-08-26 |  115  |
|     7  | Suzi  | 2006-09-02 |  502  |
|     8  | Henry | 2006-09-01 |  197  |
|     9  | Ben   | 2006-09-02 |   79  |
|    10  | Henry | 2006-08-30 |  203  |
+-----+-----+-----+-----+
```

许多其他的范例使用的是 mail 表（在前面的章节中使用过）：

```
mysql> SELECT * FROM mail;
+-----+-----+-----+-----+-----+-----+
| t      | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+-----+
| 2006-05-11 10:15:08 | barb    | saturn   | tricia  | mars    | 58274 |
| 2006-05-12 12:48:13 | tricia  | mars     | gene    | venus   | 194925 |
| 2006-05-12 15:02:49 | phil    | mars     | phil    | saturn  | 1048  |
| 2006-05-13 13:59:18 | barb    | saturn   | tricia  | venus   | 271   |
| 2006-05-14 09:31:37 | gene    | venus   | barb    | mars    | 2291  |
| 2006-05-14 11:52:17 | phil    | mars     | tricia  | saturn  | 5781  |
| 2006-05-14 14:42:21 | barb    | venus   | barb    | venus   | 98151 |
| 2006-05-14 17:03:01 | tricia  | saturn   | phil    | venus   | 2394482 |
| 2006-05-15 07:17:48 | gene    | mars     | gene    | saturn  | 3824  |
| 2006-05-15 08:50:57 | phil    | venus   | phil    | venus   | 978   |
| 2006-05-15 10:25:52 | gene    | mars     | tricia  | saturn  | 998532 |
| 2006-05-15 17:35:31 | gene    | saturn   | gene    | mars    | 3856  |
| 2006-05-16 09:00:28 | gene    | venus   | barb    | mars    | 613   |
| 2006-05-16 23:04:19 | phil    | venus   | barb    | venus   | 10294 |
| 2006-05-17 12:49:23 | phil    | mars     | tricia  | saturn  | 873   |
| 2006-05-19 22:21:51 | gene    | saturn   | gene    | venus   | 23992 |
+-----+-----+-----+-----+-----+-----+
```

偶尔也使用其他表作为范例。你可以通过在 recipes 发行包中的 tables 目录中找到的描述来创建其中的大部分表。

7.1 使用 ORDER BY 命令排序查询结果

Using ORDER BY to Sort Query Results

问题

查询的输出行没有按照你期望的顺序输出。

解决方案

在查询语句中增加一句 ORDER BY 子句来排序结果的行。

讨论

在引言中展示的 driver_log 和 mail 表的内容是无组织的且难以搞清其含义。仅有 id 和 t 两列的内容例外，它们是有序的，但这仅仅是巧合。表中的数据行确实是趋向于按照它们原来被插入的顺序返回，但仅仅只到表受到删除及更新的操作为止。在那之后插入的行可能在结果集中间的某处返回。许多 MySQL 的用户注意到了这个提取数据行时的混乱，因此他们询问，“我该怎样在我的表中存储数据行以保证当我提取它们时它们是按照特定顺序输出的？”这个问题的答案是，“这是个错误的问题。”存储数据行是服务器的工作，

你应该让服务器完成。除此之外，即便你能制定存储顺序，如果你想在不同的时间察看以不同的顺序存储的结果，它如何帮助你做到？

当你选择数据行，它们被从数据库中取出并以服务器使用的顺序返回。这个顺序可能会改变，即便对那些没有排序数据行的语句，它取决于服务器执行相关语句时所使用的索引，因为索引会影响提取数据的顺序。即便你的数据行看起来很自然的以正确的顺序返回，关系数据库对它返回的数据行的顺序也不做任何保证——除非你告诉它怎么做。如果期望将查询结果的数据行按指定顺序排列，需要在你的 SELECT 语句中增加一条 ORDER BY 子句来排序它们。没有 ORDER BY，当你改变表的内容时，你会发现提取结果的顺序可能会改变。使用 ORDER BY 子句后，MySQL 将一直按照你指定的方法来排序数据行。

ORDER BY 具有以下的综合特性：

- 可以使用单独一列的值或者多列来排序；
- 可以以升序（默认）或者降序排序任何列；
- 可以通过名字或者别名来引用排序列。

这一节展示了一些基本的排序技术，比如如何为排序列命名以及制定排序方向。接下来的一节举例说明了如何使用更复杂的排序。更荒谬的想法是，你甚至可以使用 ORDER BY 让一个结果集无序，这对于随机化数据行或与 LIMIT 一起使用在结果集中随机抽取一个数据行都非常有效。这些使用 ORDER BY 的方法在 13 章中详细描述。

接下来的范例集展示了如何排序单一数据列或者多数据列以及按照升序或者降序排序。范例在 driver_log 中选择了数据行但是以不同的顺序排序，因此你可以比较不同的 ORDER BY 子句的效果。

这次查询生成了单独使用司机姓名列的排序：

```
mysql> SELECT * FROM driver_log ORDER BY name;
+-----+-----+-----+-----+
| rec_id | name  | trav_date | miles |
+-----+-----+-----+-----+
|     1  | Ben   | 2006-08-30 |  152  |
|     9  | Ben   | 2006-09-02 |   79  |
|     5  | Ben   | 2006-08-29 |  131  |
|     8  | Henry | 2006-09-01 |  197  |
|     6  | Henry | 2006-08-26 |  115  |
|     4  | Henry | 2006-08-27 |   96  |
|     3  | Henry | 2006-08-29 |  300  |
|    10  | Henry | 2006-08-30 |  203  |
|     7  | Suzi  | 2006-09-02 |  502  |
|     2  | Suzi  | 2006-08-29 |  391  |
+-----+-----+-----+-----+
```

默认排序方向是升序。你能确定其方向，对于确定的升序排序可以通过在被排序的列名后增加 ASC 来指定：

```
SELECT * FROM driver_log ORDER BY name ASC;
```

与升序相对的是降序，通过在被排序的列名后增加 DESC 来指定：

```
mysql> SELECT * FROM driver_log ORDER BY name DESC;
+-----+-----+-----+-----+
| rec_id | name  | trav_date | miles |
+-----+-----+-----+-----+
|     2  | Suzi   | 2006-08-29 |   391 |
|     7  | Suzi   | 2006-09-02 |   502 |
|    10  | Henry  | 2006-08-30 |   203 |
|     8  | Henry  | 2006-09-01 |   197 |
|     6  | Henry  | 2006-08-26 |   115 |
|     4  | Henry  | 2006-08-27 |    96 |
|     3  | Henry  | 2006-08-29 |   300 |
|     5  | Ben    | 2006-08-29 |   131 |
|     9  | Ben    | 2006-09-02 |    79 |
|     1  | Ben    | 2006-08-30 |   152 |
+-----+-----+-----+-----+
```

如果你仔细检查刚才展示的查询输出，你会注意到尽管数据行按照名字排序，但是任意给定名字的数据行是无序的。（例如，对 Henry 或者 Ben 而言，他们的 trav_date 数值没有按照日期顺序。）这是因为 MySQL 不会主动排序数据，除非你告诉它：

- 查询返回的数据行的全部顺序是不确定的，除非你使用 ORDER BY 子句指定。
- 在按照给定列的数值排序的数据行群体中，其他列的数值的顺序也是不确定的，除非你在 ORDER BY 子句中为它们命名。

为了更全面的控制输出顺序，可以通过逐一列举需要用来排序的每一列来指定一个多列排序，每一列之间以逗号分隔。接下来的查询通过每一行名字中的 name 和 trav_date 的升序来排序结果：

```
mysql> SELECT * FROM driver_log ORDER BY name, trav_date;
+-----+-----+-----+-----+
| rec_id | name  | trav_date | miles |
+-----+-----+-----+-----+
|     5  | Ben   | 2006-08-29 |   131 |
|     1  | Ben   | 2006-08-30 |   152 |
|     9  | Ben   | 2006-09-02 |    79 |
|     6  | Henry | 2006-08-26 |   115 |
|     4  | Henry | 2006-08-27 |    96 |
|     3  | Henry | 2006-08-29 |   300 |
|    10  | Henry | 2006-08-30 |   203 |
|     8  | Henry | 2006-09-01 |   197 |
|     2  | Suzi  | 2006-08-29 |   391 |
|     7  | Suzi  | 2006-09-02 |   502 |
+-----+-----+-----+-----+
```

多列排序也能用降序排序，但是 DESC 必须在每一个列名字之后指定来实现全面的降序排序：

```
mysql> SELECT * FROM driver_log ORDER BY name DESC, trav_date DESC;
+-----+-----+-----+-----+
| rec_id | name   | trav_date | miles |
+-----+-----+-----+-----+
|     7  | Suzi   | 2006-09-02 |  502  |
|     2  | Suzi   | 2006-08-29 |  391  |
|     8  | Henry  | 2006-09-01 |  197  |
|    10  | Henry  | 2006-08-30 |  203  |
|     3  | Henry  | 2006-08-29 |  300  |
|     4  | Henry  | 2006-08-27 |   96  |
|     6  | Henry  | 2006-08-26 | 115   |
|     9  | Ben    | 2006-09-02 |   79  |
|     1  | Ben    | 2006-08-30 |  152  |
|     5  | Ben    | 2006-08-29 |  131  |
+-----+-----+-----+-----+
```

多列的 ORDER BY 子句能够实现混合序的排序，这样某些列可以按照升序排序而另一些则是降序。接下来的查询时按 name 的降序排序同时对每一个 name 按照 trav_date 的升序排序：

```
mysql> SELECT * FROM driver_log ORDER BY name DESC, trav_date;
+-----+-----+-----+-----+
| rec_id | name   | trav_date | miles |
+-----+-----+-----+-----+
|     2  | Suzi   | 2006-08-29 |  391  |
|     7  | Suzi   | 2006-09-02 |  502  |
|     6  | Henry  | 2006-08-26 | 115   |
|     4  | Henry  | 2006-08-27 |   96  |
|     3  | Henry  | 2006-08-29 |  300  |
|    10  | Henry  | 2006-08-30 |  203  |
|     8  | Henry  | 2006-09-01 |  197  |
|     5  | Ben    | 2006-08-29 |  131  |
|     1  | Ben    | 2006-08-30 |  152  |
|     9  | Ben    | 2006-09-02 |   79  |
+-----+-----+-----+-----+
```

在刚才被展示的查询中的 ORDER BY 子句是按名字引用被排序的列。你也能通过使用别名来命名列。也就是说，如果一个输出列有别名，你能在 ORDER BY 子句中引用别名：

```
mysql> SELECT name, trav_date, miles AS distance FROM driver_log
-> ORDER BY distance;
+-----+-----+-----+
| name   | trav_date | distance |
+-----+-----+-----+
| Ben    | 2006-09-02 |      79  |
| Henry  | 2006-08-27 |      96  |
| Henry  | 2006-08-26 |    115   |
| Ben    | 2006-08-29 |    131   |
| Ben    | 2006-08-30 |    152   |
| Henry  | 2006-09-01 |    197   |
+-----+-----+-----+
```

```
| Henry | 2006-08-30 |      203 |
| Henry | 2006-08-29 |      300 |
| Suzi  | 2006-08-29 |      391 |
| Suzi  | 2006-09-02 |      502 |
+-----+
```

通过别名指定的列能够按照升序或者降序排序，就像指定的列：

```
mysql> SELECT name, trav_date, miles AS distance FROM driver_log
-> ORDER BY distance DESC;
+-----+-----+-----+
| name | trav_date | distance |
+-----+-----+-----+
| Suzi | 2006-09-02 |      502 |
| Suzi | 2006-08-29 |      391 |
| Henry | 2006-08-29 |      300 |
| Henry | 2006-08-30 |      203 |
| Henry | 2006-09-01 |      197 |
| Ben   | 2006-08-30 |      152 |
| Ben   | 2006-08-29 |      131 |
| Henry | 2006-08-26 |      115 |
| Henry | 2006-08-27 |       96 |
| Ben   | 2006-09-02 |       79 |
+-----+-----+-----+
```

你应该自行排序查询结果吗？

如果你在自己的程序中使用了 `SELECT` 语句，你能把一个没排序的结果集放到一个数据结构中，然后使用你的编程语言自行排序该数据结构。但是为什么要做重复劳动呢？MySQL 服务程序内建了高效的排序功能，所以你可以让它完成自己分内的工作。

这条规则可能发生例外的情况是，当你需要以不同的方式排序一组数据行。在这种情况下，与其使用好几条仅仅只是 `ORDER BY` 子句不同的查询语句，一次获得所有数据行然后根据需要在你的程序中自行排序可能更快。不过，如果你的查询结果集非常大而且排序它们需要非常大的空间和处理时间，这个方法就显得不是那么有吸引力了。

7.2 使用表达式排序

Using Expressions for Sorting

问题

当你想排序使用某一数据列计算得到的数值排序查询结果，而并非直接使用该列存储的数值时。

解决方案

将计算所需数值的表达式放到 `ORDER BY` 子句中。

讨论

`mail` 表中的一列显示了每一封邮件的大小（以字节）：

```
mysql> SELECT * FROM mail;
+-----+-----+-----+-----+-----+-----+
| t    | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+-----+
| 2006-05-11 10:15:08 | barb   | saturn  | tricia  | mars    | 58274 |
| 2006-05-12 12:48:13 | tricia | mars    | gene    | venus   | 194925 |
| 2006-05-12 15:02:49 | phil   | mars    | phil    | saturn  | 1048  |
| 2006-05-13 13:59:18 | barb   | saturn  | tricia  | venus   | 271   |
...

```

假设你想取得“大”邮件的信息（定义大小超过 50 000 字节的邮件为大邮件），但是你希望它们以千字节为单位被排序并显示出来，而不是以字节。在这种情况下，排序所需的数值可以按照如下表达式计算得到：

```
FLOOR((size+1023)/1024)
```

对上面表达式中的 +1023 觉得疑惑吗？那是因为这里将邮件大小归类到最近的 1 024 字节类的上界。如果没有它，那么邮件大小将被归类到对应的下界（例如，2 047 字节大小的信息可能被报告仅有 1 千字节大小而不是 2 千字节）。这个技术将在 8.12 节中详细讨论。

有两种使用表达式来排序查询结果的方法。第一种，你可以直接将表达式放在 `ORDER BY` 子句中：

```
mysql> SELECT t, srcuser, FLOOR((size+1023)/1024)
-> FROM mail WHERE size > 50000
-> ORDER BY FLOOR((size+1023)/1024);
+-----+-----+-----+
| t    | srcuser | FLOOR((size+1023)/1024) |
+-----+-----+-----+
| 2006-05-11 10:15:08 | barb   | 57   |
| 2006-05-14 14:42:21 | barb   | 96   |
| 2006-05-12 12:48:13 | tricia | 191  |
| 2006-05-15 10:25:52 | gene   | 976  |
| 2006-05-14 17:03:01 | tricia | 2339 |
+-----+-----+-----+
```

第二种，如果你想在输出列中以被命名的表达式排序，你可以给它一个别名然后在 `ORDER BY` 中引用该别名：

```
mysql> SELECT t, srcuser, FLOOR((size+1023)/1024) AS kilobytes
-> FROM mail WHERE size > 50000
-> ORDER BY kilobytes;
+-----+-----+-----+
| t    | srcuse | kilobytes |
+-----+-----+-----+
| 2006-05-11 10:15:08 | barb   | 57   |
| 2006-05-14 14:42:21 | barb   | 96   |
| 2006-05-12 12:48:13 | tricia | 191  |
| 2006-05-15 10:25:52 | gene   | 976  |
+-----+-----+-----+
```

```
| 2006-05-14 17:03:01 | tricia | 2339 |  
+-----+-----+-----+
```

尽管你可以以任一种方式使用 ORDER BY 子句，但是至少有两个理由让你更愿意使用别名：

- 在 ORDER BY 子句中使用别名比重复（相当繁琐）使用表达式更容易——如果你改动了一个地方，那你不得不改动其余所有地方。
- 别名便于显示，提供了更有意义的列名标记。注意前面两次查询结果，显然第二次查询结果的第三列的列名更有意义。

7.3 显示一组按照其他属性排序的值

Displaying One Set of Values While Sorting by Another

问题

如果你想将查询结果按照你没有选择的数值排序。

解决方案

这不是问题。你可以在 ORDER BY 子句中使用不会出现在输出列表中的数据列。

讨论

ORDER BY 子句并不限制只能对那些在输出列表中命名的数据列进行排序。它也可以对隐藏的数值排序（不出现在查询输出中的数值）。这个技术通常应用在当你的数据有不同的表示方法，而你想显示一组按照其他方式排序的结果的时候。例如，你可能不希望以字节的方式显示邮件信息的大小，而是以字串的形式比如以 103KB 代表 103 千字节。你可以使用如下表达式将字节计数的方式转换为所需的方式：

```
CONCAT(FLOOR((size+1023)/1024), 'K')
```

然而，这样的数值是字串，所以它们将按照词汇序排序，而不是数字序。如果你使用它们来排序，那 96KB 将排序在 2 339KB 之后，尽管它代表的实际数值更小：

```
mysql> SELECT t, srcuser,  
-> CONCAT(FLOOR((size+1023)/1024), 'K') AS size_in_K  
-> FROM mail WHERE size > 50000  
-> ORDER BY size_in_K;  
+-----+-----+-----+  
| t      | srcuser | size_in_K |  
+-----+-----+-----+  
| 2006-05-12 12:48:13 | tricia | 191K    |  
| 2006-05-14 17:03:01 | tricia | 2339K   |  
| 2006-05-11 10:15:08 | barb   | 57K     |
```

```
| 2006-05-14 14:42:21 | barb      | 96K        |
| 2006-05-15 10:25:52 | gene      | 976K       |
+-----+-----+-----+
```

以下的语句可以实现期望的输出，显示字符串，但是使用数字值排序：

```
mysql> SELECT t, srcuser,
-> CONCAT(FLOOR((size+1023)/1024),'K') AS size_in_K
-> FROM mail WHERE size > 50000
-> ORDER BY size;
+-----+-----+-----+
| t      | srcuser | size_in_K |
+-----+-----+-----+
| 2006-05-11 10:15:08 | barb    | 57K      |
| 2006-05-14 14:42:21 | barb    | 96K      |
| 2006-05-12 12:48:13 | tricia  | 191K     |
| 2006-05-15 10:25:52 | gene    | 976K     |
| 2006-05-14 17:03:01 | tricia  | 2339K    |
+-----+-----+-----+
```

按照字符串方式显示按数字值方式排序的数值也能够让你摆脱一些非常困难的情形。体育团队的队员通常都被指派了一个运动衣的号码，这个号码你可能认为应该使用数字序的列存储。不要如此轻易的下决定！一个运动员的运动衣可能是 0 号，而另外一个可能是 00 号。如果一个体育团队的队员同时有这两个号，那么你不能将它们视为同样的号码。解决这个问题的方法是将运动衣号码按照字符串存储：

```
CREATE TABLE roster
(
  name          CHAR(30),      # 选手姓名
  jsrsey_num   CHAR(3)       # 运动衣号
);
```

这样运动衣号码将按照你输入的方式显示出来，这样 0 和 00 将按照不同的数值对待。不幸的是，尽管采用字符串表示方式可以解决区分 0 和 00 的问题，它同时也引入了另外一个问题。假设一个体育团队拥有以下的队员：

```
mysql> SELECT name, jersey_num FROM roster;
+-----+-----+
| name    | jersey_num |
+-----+-----+
| Lynne   | 29        |
| Ella    | 0         |
| Elizabeth | 100      |
| Nancy   | 00        |
| Jean    | 8         |
| Sherry  | 47        |
+-----+-----+
```

0

当你试图按照运动衣号码排序队员时，问题来了。如果这些号码按照字符串方式存储，它们将按照词汇序排序，而词汇序通常与数字序是不同的。而这对体育团队而言确实是个问题：

```
mysql> SELECT name, jersey_num FROM roster ORDER BY jersey_num;
+-----+-----+
| name | jersey_num |
+-----+-----+
| Ella | 0          |
| Nancy | 00         |
| Elizabeth | 100      |
| Lynne | 29         |
| Sherry | 47         |
| Jean | 8          |
+-----+-----+
```

显然，数值 100 和 8 的位置不正确。不过这一点可以很轻易的解决。我们可以显示字符串数值，但是使用数字数值排序。为了达成这个目标，可以给 `jersey_num` 值加上一个 0 来强制完成字符串到数字的转换：

```
mysql> SELECT name, jersey_num FROM roster ORDER BY jersey_num+0;
+-----+-----+
| name | jersey_num |
+-----+-----+
| Ella | 0          |
| Nancy | 00         |
| Jean | 8          |
| Lynne | 29         |
| Sherry | 47         |
| Elizabeth | 100      |
+-----+-----+
```

显示一组按照其他属性排序的值的技术还可以应用在别的场合，比如，当你想显示一组由许多列复合而成的数值，但是它们并非按照你希望的顺序排序时。例如，`mail` 表中列举消息发送者使用了单独的 `srcuser` 和 `srchost` 值。如果你希望从 `mail` 表中按照电子邮件地址 `srcuser@srchost` 的格式显示消息发送者，你可以通过如下表达式来创建该数值：

```
CONCAT(srcuser, '@', srchost)
```

然而，如果你希望对待 `hostname` 比对 `username` 更加有意义，这样的数值对排序并不合适。因此，使用其下的列数值来排序结果比使用显示的复合数值本身要更合适：

```
mysql> SELECT t, CONCAT(srcuser, '@', srchost) AS sender, size
-> FROM mail WHERE size > 50000
-> ORDER BY srchost, srcuser;
+-----+-----+-----+
| t           | sender        | size   |
+-----+-----+-----+
| 2006-05-15 10:25:52 | gene@mars    | 998532 |
| 2006-05-12 12:48:13 | tricia@mars | 194925 |
| 2006-05-11 10:15:08 | barb@saturn | 58274  |
+-----+-----+-----+
```

```
| 2006-05-14 17:03:01 | tricia@saturn | 2394482 |
| 2006-05-14 14:42:21 | barb@venus   | 98151  |
+-----+-----+-----+
```

同样的想法也应用在排序人名上。假设你拥有一张包括姓和名的人名表。首先按照姓来排列数据行，当数据列是单独显示时，查询是直截了当的：

```
mysql> SELECT last_name, first_name FROM name
-> ORDER BY last_name, first_name;
+-----+-----+
| last_name | first_name |
+-----+-----+
| Blue      | Vida       |
| Brown     | Kevin       |
| Gray      | Pete        |
| White     | Devon       |
| White     | Rondell    |
+-----+-----+
```

如果你想按照名、空格、姓的字符串方式显示每一个姓名，你可以按照如下方式查询：

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM name ...
```

但是你如何对姓名排序以保证他们是按照姓的顺序输出的？答案是，显示复合姓名，但是在 ORDER BY 字句中引用其他数值排序：

```
mysql> SELECT CONCAT(first_name, ' ', last_name) AS full_name
-> FROM name
-> ORDER BY last_name, first_name;
+-----+
| full_name |
+-----+
| Vida Blue |
| Kevin Brown |
| Pete Gray  |
| Devon White |
| Rondell White |
+-----+
```

7.4 字符串排序的大小写区分控制

Controlling Case Sensitivity of String Sorts

问题

字符串排序操作是大小写敏感的，特别是在当你不希望它们这样的时候，反之亦然。

解决方案

改变被排序数值的比较特征。

讨论

第 5 章字符串操作中讨论了字符串的属性比较是怎样基于该字符串是否是二进制格式的：

- 二进制字符串是二进制序列。它们的比较是通过对每个字节的数字值比较实现的。字符集和字母对比较而言是无意义的。
- 非二进制字符串是 Collation 列。它们具有字符集合 Collation 并且通过使用预先定义好的 Collation 顺序来实现字符与字符的对比。

这些属性同样都可以应用在字符串的排序中，因为排序本身就是基于对比的。为了改变一个字符串列的排序属性，你必须改变它的比较属性。（第 5 章第 2 节详细总结了什么类型的字符串是二进制，什么样的是非二进制的）。

本节的范例是一张同时具有大小写敏感和大小写不敏感的字符串数据列，以及二进制数据列的表：

```
CREATE TABLE str_val
(
    ci_str      CHAR(3) CHARACTER SET latin1 COLLATE latin1_swedish_ci,
    cs_str      CHAR(3) CHARACTER SET latin1 COLLATE latin1_general_cs,
    bin_str     Binary(3)
);
```

假设表格具有如下内容：

ci_str	cs_str	bin_str
AAA	AAA	AAA
aaa	aaa	aaa
bbb	bbb	bbb
BBB	BBB	BBB

每一列都包含同样的数值，但是对每一列以自然顺序排序得到三个不同的结果：

- 大小写不敏感的排序将 a 和 A 排列到了一起，将它们放到了 b 和 B 之前。然而，对于给定的一个字母，将一个字母排序到另一个之前不是必须的，如下所示：

```
mysql> SELECT ci_str FROM str_val ORDER BY ci_str;
+-----+
| ci_str |
+-----+
| AAA    |
| aaa    |
| bbb    |
| BBB    |
+-----+
```

- 大小写敏感的排序将 A 和 a 放到了 B 和 b 之前，并将大写字母排序到小写字母之前：

```
mysql> SELECT cs_str FROM str_val ORDER BY cs_str;
+-----+
| cs_str |
+-----+
| AAA    |
| aaa    |
| BBB    |
| bbb    |
+-----+
```

- 二进制字符串则按照数字序排序。假设大写字母的数字值比相应的小写字母小，则二进制排序结果的顺序如下：

```
mysql> SELECT bin_str FROM str_val ORDER BY bin_str;
+-----+
| bin_str |
+-----+
| AAA    |
| BBB    |
| aaa    |
| bbb    |
+-----+
```

你可以在具有二进制 Collation 的非二进制字符串列数值中得到同样的查询结果，只要该数据列包含单字节字符（例如，CHAR(3) CHARACTER SET latin1 COLLATE latin1_bin）。对多字节字符，二进制 Collation 仍旧产生了数字排序，但是字符数值使用的是多字节数字。

如果需要改变每一列的排序属性，需要使用第 5 章第 9 节介绍的技术来控制字符串比较的工作：

- 如果需要用大小写区分的方式来排序非大小写区分的字符串，需要使用大小写区分的 Collation 为被排序的数值定序：

```
mysql> SELECT ci_str FROM str_val
-> ORDER BY ci_str COLLATE latin1_general_cs;
+-----+
| ci_str |
+-----+
| AAA    |
| aaa    |
| BBB    |
| bbb    |
+-----+
```

- 如果需要以非大小写区分的方式排序大小写区分的字符串，需要使用非大小写区分的 Collation 为被排序的数值定序：

```
mysql> SELECT cs_str FROM str_val
-> ORDER BY cs-str COLLATE latin1_swedish_ci;
```

```
+-----+
| cs_str |
+-----+
| AAA    |
| aaa    |
| bbb    |
| BBB    |
+-----+
```

其他可能的排序所使用的数值都将转换成同样字母，这种转换让字母之间不相关：

```
mysql> SELECT cs_str FROM str_val
      -> ORDER BY UPPER(cs_str);
+-----+
| cs_str |
+-----+
| AAA    |
| aaa    |
| bbb    |
| BBB    |
+-----+
```

- 二进制字符串将使用数字字节值排序，所以没有字母的概念被引入。然而，因为不同情形的字母具有不同的字节值，所以有效的二进制字符串比较是区分大小写的（也就是说，A 和 a 是不同的）。如果要使用大小写区分的字序来排序二进制字符串，需要将它们转换为非二进制字符串并使用正确的 Collation。例如，为实现非大小写区分的排序，可以使用如下的语句：

```
mysql> SELECT bin_str FROM str_val
      -> ORDER BY CONVERT(bin_str USING latin1) COLLATE latin1_swedish_ci;
+-----+
| bin_str |
+-----+
| AAA    |
| aaa    |
| bbb    |
| BBB    |
+-----+
```

如果默认的排序是不区分大小写的 (latin1 排序就是这样的)，你可以省略 COLLATE 语句。

7.5 基于日期的排序

Date-Based Sorting

问题

你可能只是想临时的排序数据行。

解决方案

使用日期或时间数据类型。如果数值的一些部分与你想完成的排序不相关，忽略它们。

讨论

许多数据库表包括日期或者时间信息，而且它对于临时的排序数据行通常是非常必要的。MySQL 知道如何排序临时数据类型，所以对 DATE、DATETIME、TIME 或 TIMESTAMP 这些数据列类型的定序没有任何特殊的诀窍。让我们从一张包含上面每种类型数值的表开始：

```
mysql> SELECT * FROM temporal_val;
+-----+-----+-----+-----+
| d    | dt   | t    | ts   |
+-----+-----+-----+-----+
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 1980-01-01 02:00:00 |
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 2021-01-01 03:00:00 |
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 1975-01-01 04:00:00 |
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 1985-01-01 05:00:00 |
+-----+-----+-----+-----+
```

对任一数据列使用 ORDER BY 字句，将相应数值排序为正确顺序：

```
mysql> SELECT * FROM temporal_val ORDER BY d;
+-----+-----+-----+-----+
| d    | dt   | t    | ts   |
+-----+-----+-----+-----+
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 1985-01-01 05:00:00 |
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 1980-01-01 02:00:00 |
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 1975-01-01 04:00:00 |
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 2021-01-01 03:00:00 |
+-----+-----+-----+-----+
mysql> SELECT * FROM temporal_val ORDER BY dt;
+-----+-----+-----+-----+
| d    | dt   | t    | ts   |
+-----+-----+-----+-----+
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 2021-01-01 03:00:00 |
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 1975-01-01 04:00:00 |
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 1980-01-01 02:00:00 |
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 1985-01-01 05:00:00 |
+-----+-----+-----+-----+
mysql> SELECT * FROM temporal_val ORDER BY t;
+-----+-----+-----+-----+
| d    | dt   | t    | ts   |
+-----+-----+-----+-----+
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 1985-01-01 05:00:00 |
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 1975-01-01 04:00:00 |
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 1980-01-01 02:00:00 |
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 2021-01-01 03:00:00 |
+-----+-----+-----+-----+
mysql> SELECT * FROM temporal_val ORDER BY ts;
+-----+-----+-----+-----+
| d    | dt   | t    | ts   |
+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 1975-01-01 04:00:00 |
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 1980-01-01 02:00:00 |
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 1985-01-01 05:00:00 |
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 2021-01-01 03:00:00 |
+-----+-----+-----+-----+
```

有时，临时排序仅使用日期或时间数据列的一部分。这种情况下，可以使用表达式获取你所需要的部分同时也可以使用表达式排序结果。相关的范例将在后续的章节中给出。

7.6 按日历排序

Sorting by Calendar Day

问题

你可能希望按照年历的日期排序。

解决方案

使用月和天的日期值排序，忽略年的信息。

讨论

按照历法顺序与按照日期排序是不同的。你需要忽略日期的年份信息而仅仅按照月和天对应的年历位置顺序为它们排序。假设你拥有一张事件表，它看起来是按照实际发生的日期排序的：

```
mysql> SELECT date, description FROM event ORDER BY date;
+-----+-----+
| date      | description          |
+-----+-----+
| 1215-06-15 | Signing of the Magna Carta |
| 1732-02-22 | George Washington's birthday |
| 1776-07-14 | Bastille Day           |
| 1789-07-04 | US Independence Day    |
| 1809-02-12 | Abraham Lincoln's birthday |
| 1919-06-28 | Signing of the Treaty of Versailles |
| 1944-06-06 | D-Day at Normandy Beaches   |
| 1957-10-04 | Sputnik launch date       |
| 1958-01-31 | Explorer 1 launch date    |
| 1989-11-09 | Opening of the Berlin Wall  |
+-----+-----+
```

如果需要将这些项目按照日历序列列出，首先需要将它们按照月排序，然后再在每个月内按照日期排序：

```
mysql> SELECT date, description FROM event
-> ORDER BY MONTH(date), DAYOFMONTH(date);
+-----+-----+
| date      | description          |
+-----+-----+
```

```
+-----+-----+
| 1958-01-31 | Explorer 1 launch date
| 1809-02-12 | Abraham Lincoln's birthday
| 1732-02-22 | George Washington's birthday
| 1944-06-06 | D-Day at Normandy Beaches
| 1215-06-15 | Signing of the Magna Carta
| 1919-06-28 | Signing of the Treaty of Versailles
| 1789-07-04 | US Independence Day
| 1776-07-14 | Bastille Day
| 1957-10-04 | Sputnik launch date
| 1989-11-09 | Opening of the Berlin Wall
+-----+-----+
```

MySQL 同样也有一个叫做 DAYOFYEAR() 的函数，你可能会怀疑它对于日历日期的排序是否有用：

```
mysql> SELECT date, description FROM event ORDER BY DAYOFYEAR(date);
+-----+-----+
| date      | description          |
+-----+-----+
| 1958-01-31 | Explorer 1 launch date
| 1809-02-12 | Abraham Lincoln's birthday
| 1732-02-22 | George Washington's birthday
| 1944-06-06 | D-Day at Normandy Beaches
| 1215-06-15 | Signing of the Magna Carta
| 1919-06-28 | Signing of the Treaty of Versailles
| 1789-07-04 | US Independence Day
| 1776-07-14 | Bastille Day
| 1957-10-04 | Sputnik launch date
| 1989-11-09 | Opening of the Berlin Wall
+-----+-----+
```

看起来这个函数工作正常，那仅仅是因为这张表并没有包含会暴露使用 DAYOFYEAR() 函数排序问题的数据行：对于不同的日历日期，它可能会产生同样的数值。例如，闰年的 2 月 29 日和非闰年的 3 月 1 日具有同样的数值：

```
mysql> SELECT DAYOFYEAR('1996-02-29'), DAYOFYEAR('1997-03-01');
+-----+-----+
| DAYOFYEAR('1996-02-29') | DAYOFYEAR('1997-03-01') |
+-----+-----+
|           60 |           60 |
+-----+-----+
```

这个特性意味着 DAYOFYEAR() 函数将不会总是产生正确的日历排序结果。但是它能将实际发生在不同的历日上的日期聚类。

如果一张表采用分离的年、月、日数据列来表示日期，那么排序日历就不需要提取部分日期的信息了。直接排序相关数据列就可以了。对大数据集，使用分离日期数据列排序比基于日期部分信息提取排序要快得多。这并不是凌驾于日期部分信息提取之上，更重要的是，你能分别指出部分日期数据列——这些对于使用单一日期数据列是不可能的。这条原则说明，你应该在设计数据表时就要考虑到数据提取和排序的易用性，特别是针对那些你希望经常使用的数据。

7.7 按周历排序

Sorting by Day of Week

问题

如果你想按照周历排序数据行。

解决方案

使用 DAYOFWEEK() 函数将日期数据列转换为它相应的周历数字值。

讨论

周历排序与日历排序类似，区别只在于你使用了不同的函数来获取相关顺序的数值。

你可以通过使用 DAYNAME() 函数来获取一周的日期，但是那样产生的字符串将按照词汇顺序排序而不是按照周历顺序（星期日、星期一、星期二，等等）。在这里，显示一个按照其余数值排序的数值的技术显得相当有用（参照 7.3 节）。使用 DAYNAME() 函数显示日期名字，但是使用 DAYOFWEEK() 函数按照周历排序，这个函数将对星期日到星期六分别返回数值 1 到 7：

```
mysql> SELECT DAYNAME(date) AS day, date, description
-> FROM event
-> ORDER BY DAYOFWEEK(date);
+-----+-----+-----+
| day   | date      | description          |
+-----+-----+-----+
| Sunday | 1809-02-12 | Abraham Lincoln's birthday |
| Sunday | 1776-07-14 | Bastille Day           |
| Monday | 1215-06-15 | Signing of the Magna Carta |
| Tuesday | 1944-06-06 | D-Day at Normandy Beaches |
| Thursday| 1989-11-09 | Opening of the Berlin Wall |
| Friday  | 1732-02-22 | George Washington's birthday |
| Friday  | 1958-01-31 | Explorer 1 launch date    |
| Friday  | 1957-10-04 | Sputnik launch date       |
| Saturday| 1919-06-28 | Signing of the Treaty of Versailles |
| Saturday| 1789-07-04 | US Independence Day       |
+-----+-----+-----+
```

如果你想按照周历顺序排序数据行，但是希望将星期一作为一周的起始而将星期日作为一周的最后一天，你可以通过使用 MOD() 函数将星期一映射为 0，星期二映射为 1，...，星期日映射为 6：

```
mysql> SELECT DAYNAME(date), date, description
-> FROM event
-> ORDER BY MOD(DAYOFWEEK(date)+5, 7);
+-----+-----+-----+
| DAYNAME(date) | date      | description          |
+-----+-----+-----+
| Monday        | 1215-06-15 | Signing of the Magna Carta |
| Tuesday       | 1944-06-06 | D-Day at Normandy Beaches |
| Thursday      | 1989-11-09 | Opening of the Berlin Wall |
+-----+-----+-----+
```

Friday	1732-02-22	George Washington's birthday	
Friday	1957-10-04	Sputnik launch date	
Friday	1958-01-31	Explorer 1 launch date	
Saturday	1789-07-04	US Independence Day	
Saturday	1919-06-28	Signing of the Treaty of Versailles	
Sunday	1776-07-14	Bastille Day	
Sunday	1809-02-12	Abraham Lincoln's birthday	

下面的表显示使用 DAYOFWEEK() 函数的表达式可以用来将一周的任意一天作为排序的第一天：

列举显示的第一天	DAYOFWEEK()表达式
Sunday	DAYOFWEEK(date)
Monday	MOD(DAYOFWEEK(date)+5, 7)
Tuesday	MOD(DAYOFWEEK(date)+4, 7)
Wednesday	MOD(DAYOFWEEK(date)+3, 7)
Thursday	MOD(DAYOFWEEK(date)+2, 7)
Friday	MOD(DAYOFWEEK(date)+1, 7)
Saturday	MOD(DAYOFWEEK(date)+0, 7)

另外一个可以用来对周历排序的函数是 WEEKDAY()，不过它返回的是一个不同的数值集（星期一为 0，到星期日为 6）。

7.8 按时钟排序

Sorting by Time of Day

问题

如果你希望按照时钟排序数据行。

解决方案

从存储时间信息的数据列中取出小时、分钟和秒的信息，并使用它们来排序。

讨论

每天时钟的排序可以通过不同的方法完成，取决于你的数据列的类型。如果相关数值存储在叫做 timecol 的时间数据列里，那么只需要使用 ORDER BY timecol 字句直接对它们排序。如果需要把 DATETIME 或者 TIMESTAMP 数据值按照时钟顺序放置，那么就提取时间部分的信息并对它们排序。例如，mail 表包含有 DATETIME 数据，那么它可以通过如下方式按照时钟排序：

```
mysql> SELECT * FROM mail ORDER BY HOUR(t), MINUTE(t), SECOND(t);
+-----+-----+-----+-----+-----+
| t    | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+-----+
| 2006-05-15 07:17:48 | gene    | mars   | gene    | saturn  | 3824 |
| 2006-05-15 08:50:57 | phil    | venus  | phil    | venus   | 978  |
| 2006-05-16 09:00:28 | gene    | venus  | barb   | mars    | 613  |
| 2006-05-14 09:31:37 | gene    | venus  | barb   | mars    | 2291 |
| 2006-05-11 10:15:08 | barb   | saturn | tricia | mars    | 58274 |
| 2006-05-15 10:25:52 | gene    | mars   | tricia | saturn  | 998532 |
| 2006-05-14 11:52:17 | phil    | mars   | tricia | saturn  | 5781 |
| 2006-05-12 12:48:13 | tricia | mars   | gene   | venus   | 194925 |
...

```

你也能使用 `TIME_TO_SEC()` 函数，它可以去掉日期部分信息并将时间部分信息转换为相应的秒数：

```
mysql> SELECT * FROM mail ORDER BY TIME_TO_SEC(t);
```

```
+-----+-----+-----+-----+-----+
| t           | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+
| 2006-05-15 07:17:48 | gene    | mars   | gene    | saturn  | 3824 |
| 2006-05-15 08:50:57 | phil    | venus  | phil    | venus   | 978  |
| 2006-05-16 09:00:28 | gene    | venus  | barb   | mars    | 613  |
| 2006-05-14 09:31:37 | gene    | venus  | barb   | mars    | 2291 |
| 2006-05-11 10:15:08 | barb   | saturn | tricia | mars    | 58274 |
| 2006-05-15 10:25:52 | gene    | mars   | tricia | saturn  | 998532 |
| 2006-05-14 11:52:17 | phil    | mars   | tricia | saturn  | 5781 |
| 2006-05-12 12:48:13 | tricia | mars   | gene   | venus   | 194925 |
...

```

7.9 按数据列的子串排序

Sorting Using Substrings of Column Values

问题

如果你希望使用每个数值的一个或多个子串排序某一组数值。

解决方案

提取你感兴趣部分的信息然后分别对它们排序。

讨论

在本章第 2 节有一个具体的按照表达式数值排序的应用。如果你希望仅使用数据列中的某一个特定部分来排序数据行，那么就提取你需要的子串并且在 `ORDER BY` 子句中使用它作为排序标准。最容易的情形就是子串在数据列中的位置和长度都是固定的。如果子串的长度或位置是可变的，当存在一些可靠的方法来确认相关信息时，你仍然可以使用它们来排序。接下来的几节将展示如何使用子串提取来生成特定的排序结果。

7.10 按固定长度的子串排序

Sorting by Fixed-Length Substrings

问题

如果你希望使用数据列中出现在指定位置的某一列的部分来排序。

解决方案

使用 LEFT()、MID() 或者 RIGHT() 函数提取你需要的部分，然后使用它们排序。

讨论

假设你有一张家庭用具的分类表，其中的项目由 3 个子类别组成 10 字符的识别值确认：一个 3 字符的缩写分类（比如 DIN 代表“dining room（食堂）”），KIT 代表“kitchen（厨房）”，一个 5 位数字的序列号，还有一个 2 字符的国家识别码指明该部分的产地：

```
mysql> SELECT * FROM housewares;
+-----+-----+
| id   | description |
+-----+-----+
| DIN40672US | dining table |
| KIT00372UK | garbage disposal |
| KIT01729JP | microwave oven |
| BED00038SG | bedside lamp |
| BTH00485US | shower stall |
| BTH00415JP | lavatory |
+-----+-----+
```

存储复杂的识别码并非一个好的方法，稍后我们将考虑如何使用分离的数据列来表示它们（见 11.11 节）。不过现在，我们先假设这些数值必须按照上面的方式存储。

如果你希望使用识别码对上表的数据行排序，只需要使用整个数据列值：

```
mysql> SELECT * FROM housewares ORDER BY id;
+-----+-----+
| id   | description |
+-----+-----+
| BED00038SG | bedside lamp |
| BTH00415JP | lavatory |
| BTH00485US | shower stall |
| DIN40672US | dining table |
| KIT00372UK | garbage disposal |
| KIT01729JP | microwave oven |
+-----+-----+
```

但是你可能有使用识别码的三个部分中任意部分排序的需要（例如，按照生产国家排序）。提取一个数据列中指定部分的函数 LEFT()、RIGHT()、MID() 可以实现该操作。这三个用来将识别码分解成为组成它的三个子部分：

```
mysql> SELECT id,
-> LEFT(id,3) AS category,
-> MID(id,4,5) AS serial,
-> RIGHT(id,2) AS country
-> FROM housewares;
+-----+-----+-----+
| id      | category | serial | country |
+-----+-----+-----+
| DIN40672US | DIN      | 40672  | US       |
| KIT00372UK | KIT      | 00372  | UK       |
| KIT01729JP | KIT      | 01729  | JP       |
| BED00038SG | BED      | 00038  | SG       |
| BTH00485US | BTH      | 00485  | US       |
| BTH00415JP | BTH      | 00415  | JP       |
+-----+-----+-----+
```

识别码的任意固定长度的字串都可以用来排序，无论是单独使用或联合使用。为了按照制造类别排序，可以提取分类数值并在 ORDER BY 字句中使用它：

```
mysql> SELECT * FROM housewares ORDER BY LEFT(id,3);
+-----+-----+
| id      | description |
+-----+-----+
| BED00038SG | bedside lamp |
| BTH00485US | shower stall |
| BTH00415JP | lavatory   |
| DIN40672US | dining table |
| KIT00372UK | garbage disposal |
| KIT01729JP | microwave oven |
+-----+-----+
```

为了按照制造序列号排序数据行，可以使用 MID() 函数从识别码中提取中间 5 位字符，从第 4 位开始：

```
mysql> SELECT * FROM housewares ORDER BY MID(id,4,5);
+-----+-----+
| id      | description |
+-----+-----+
| BED00038SG | bedside lamp |
| KIT00372UK | garbage disposal |
| BTH00415JP | lavatory   |
| BTH00485US | shower stall |
| KIT01729JP | microwave oven |
| DIN40672US | dining table |
+-----+-----+
```

这看起来似乎是数字排序，但实际上它是字符串排序，因为 MID() 函数返回的是字符串。之所以看起来相同是因为在这个例子中词汇和数字的排序结果是相同的，因为“数字”之前的 0 让它们具有相同的长度。

为了按国家识别码排序，可以使用识别码的最右边两个字符：

```
mysql> SELECT * FROM housewares ORDER BY RIGHT(id,2);
+-----+-----+
| id      | description |
+-----+-----+
| KIT01729JP | microwave oven   |
| BTH00415JP | lavatory        |
| BED00038SG | bedside lamp     |
| KIT00372UK | garbage disposal |
| DIN40672US | dining table    |
| BTH00485US | shower stall    |
+-----+-----+
```

你也可以使用联合字串排序。例如，为了按照国家识别码和串号排序，可以这样查询：

```
mysql> SELECT * FROM housewares ORDER BY RIGHT(id,2), MID(id,4,5);
+-----+-----+
| id      | description |
+-----+-----+
| BTH00415JP | lavatory        |
| KIT01729JP | microwave oven   |
| BED00038SG | bedside lamp     |
| KIT00372UK | garbage disposal |
| BTH00485US | shower stall    |
| DIN40672US | dining table    |
+-----+-----+
```

7.11 按可变长度的子串排序

Sorting by Variable-Length Substrings

问题

如果你希望使用一个串中没有给定位置的部分排序。

解决方案

找出一些确认你所需要的部分的方法，然后提取它们。否则，幸运不会降临到你身上。

讨论

如果你希望用来排序的字串长度是可变的，你需要一种可信赖的方法来提取你感兴趣的部分。为了显示它如何工作，我们首先创建一张类似 7.14 节的 housewares 表，叫做 housewares2 号表，只是它的识别码值中串号部分没有前置的 0：

```
mysql> SELECT * FROM housewares2;
+-----+-----+
| id      | description |
+-----+-----+
```

```

| DIN40672US | dining table      |
| KIT372UK   | garbage disposal   |
| KIT1729JP  | microwave oven    |
| BED38SG    | bedside lamp       |
| BTH485US   | shower stall       |
| BTH415JP   | lavatory          |
+-----+-----+

```

识别码中分类和国家部分可以被提取出并通过使用 LEFT() 和 RIGHT() 函数排序，与使用家庭用具表类似。但是现在数字段部分具有不同的长度不能通过使用简单的 MID() 函数调用来提取并排序了。取而代之的是使用 SUBSTRING() 函数来忽略头三个字符。这样，剩余部分将从第四个字符开始，提取除最右边两列的所有字符。下面是一种完成该任务的方法：

```

mysql> SELECT id, LEFT(SUBSTRING(id,4),CHAR_LENGTH(SUBSTRING(id,4)-2))
-> FROM housewares2;
+-----+-----+
| id      | LEFT(SUBSTRING(id,4),CHAR_LENGTH(SUBSTRING(id,4)-2)) |
+-----+-----+
| DIN40672US | 40672           |
| KIT372UK   | 372             |
| KIT1729JP  | 1729            |
| BED38SG    | 38              |
| BTH485US   | 485             |
| BTH415JP   | 415             |
+-----+-----+

```

但是这样比需要的麻烦太多了。SUBSTRING() 函数的第三个参数是可选参数，用来指明期望结果的长度，同时我们清楚中间部分的长度等于字符串长度减去 5（开始的 3 个字符和结束的 2 个字符）。接下来的查询演示了如何取得识别码中除开始部分和最右边后缀的中间数字部分：

```

mysql> SELECT id, SUBSTRING(id,4), SUBSTRING(id,4,CHAR_LENGTH(id)-5)
-> FROM housewares2;
+-----+-----+-----+
| id      | SUBSTRING(id,4) | SUBSTRING(id,4,CHAR_LENGTH(id)-5) |
+-----+-----+-----+
| DIN40672US | 40672US       | 40672           |
| KIT372UK   | 372UK         | 372             |
| KIT1729JP  | 1729JP        | 1729            |
| BED38SG    | 38SG          | 38              |
| BTH485US   | 485US         | 485             |
| BTH415JP   | 415JP         | 415             |
+-----+-----+-----+

```

不幸的是，尽管最终的表达式正确的从识别码中提取到了数字部分，但是返回的结果是字符串。因此，将它们按照词汇顺序排序比按数字顺序排序更好：

```

mysql> SELECT * FROM housewares2
-> ORDER BY SUBSTRING(id,4,CHAR_LENGTH(id)-5);

```

```
+-----+-----+
| id      | description   |
+-----+-----+
| KIT1729JP | microwave oven |
| KIT372UK  | garbage disposal |
| BED38SG   | bedside lamp    |
| DIN40672US | dining table   |
| BTH415JP  | lavatory       |
| BTH485US  | shower stall   |
+-----+-----+
```

那么怎样处理这个问题？一个方法是将结果加上 0，告诉 MySQL 执行一个字符串到数值的转换操作，这个操作得到的结果是一系列数字值的数字排序结果：

```
mysql> SELECT * FROM housewares2
-> ORDER BY SUBSTRING(id,4,CHAR_LENGTH(id)-5)+0;
+-----+-----+
| id      | description   |
+-----+-----+
| BED38SG | bedside lamp    |
| KIT372UK | garbage disposal |
| BTH415JP | lavatory       |
| BTH485US | shower stall   |
| KIT1729JP | microwave oven |
| DIN40672US | dining table   |
+-----+-----+
```

但是在这个特定的情形下，有一个可能的简单解决方法。计算字符串的数字部分长度是没有必要的，因为字符串到数字的转换操作将忽略跟随在后面的非数字后缀并且提供在识别码中可变长度的数字序列部分上排序所需要的数值。这也意味着 `SUBSTRING()` 函数的第三个参数实际上是不需要的：

```
mysql> SELECT * FROM housewares2
-> ORDER BY SUBSTRING(id,4)+0;
+-----+-----+
| id      | description   |
+-----+-----+
| BED38SG | bedside lamp    |
| KIT372UK | garbage disposal |
| BTH415JP | lavatory       |
| BTH485US | shower stall   |
| KIT1729JP | microwave oven |
| DIN40672US | dining table   |
+-----+-----+
```

在前面的示例中，提取可变长度子串的方法都是基于对比字符串末尾（也就是数字对比非数字），识别码的中部含有不同种类的字符。在其他情形下，你可能使用分隔符来分离数据列数值。下面的例子中，假设一张名为 `housewares3` 的表，其中的识别码如下所示：

```
mysql> SELECT * FROM housewares3;
+-----+-----+
| id      | description   |
+-----+-----+
```

```
+-----+-----+
| 13-478-92-2 | dining table      |
| 873-48-649-63 | garbage disposal   |
| 8-4-2-1       | microwave oven    |
| 97-681-37-66 | bedside lamp      |
| 27-48-534-2  | shower stall     |
| 5764-56-89-72 | lavatory         |
+-----+-----+
```

为了从这些数值中提取所需的数据片，使用 `SUBSTRING_INDEX(str, c, n)` 函数。它在字符串 `str` 中搜索给定字符 `c` 的第 `n` 个出现的位置并在那个字符的左边将所有的东西返回。例如，接下来的调用返回 13-478：

```
SUBSTRING_INDEX ('13-478-92-2', ' ', 2)
```

如果 `n` 是负数，那么对 `c` 的搜索从右边开始并返回最右边的字符串。下面的调用返回 478-92-2：

```
SUBSTRING_INDEX( '13-478-92-2' , ' ' , -3)
```

通过在 `SUBSTRING_INDEX()` 调用中集成正的和负的指数，从每个识别码中提取后继片是可能的。一个方法是提取该数值的第一个 `n` 片段部分，然后提取最右边的一个。通过把 `n` 从 1 变化到 4，我们得到了从左到右的后继片：

```
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',1),'-',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',2),'-',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',3),'-',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',4),'-',1)
```

这些表达式的第一个是可以被优化的，因为内部的 `SUBSTRING_INDEX()` 调用返回一个单独片段的串且通过它自己完全能够返回最左边的识别码片段：

```
SUBSTRING_INDEX(id,'-',1)
```

另外一个获得子串的方法是提取数值最右边的 `n` 个片段，然后提取第一个。这里我们将 `n` 从 -4 变化到 -1：

```
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',-4),'-',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',-3),'-',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',-2),'-',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',-1),'-',1)
```

这里再一次可以被优化。对第四个表达式，内部的 `SUBSTRING_INDEX()` 调用足以返回最后的子串：

```
SUBSTRING_INDEX(id,'-',1)
```

这些表达式可能阅读理解起来比较困难，因此你可以尝试使用它们中的一小部分来查看它们是如何工作的。这里有一个例子来显示如何从识别码数值中获取第二和第四个片段：

```
mysql> SELECT
-> id,
-> SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',2),'-',1) AS segment2,
```

```

-> SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',4),'-',1) AS segment4
-> FROM housewares3;
+-----+-----+-----+
| id      | segment2 | segment4 |
+-----+-----+-----+
| 13-478-92-2 | 478     | 2        |
| 873-48-649-63 | 48      | 63       |
| 8-4-2-1      | 4       | 1        |
| 97-681-37-66 | 681     | 66       |
| 27-48-534-2 | 48      | 2        |
| 5764-56-89-72 | 56      | 72       |
+-----+-----+-----+

```

为了使用子串进行排序，需要在 ORDER BY 子句中使用正确的表达式。（如果你希望使用数字方式而非词汇方式排序，那么记住通过加 0 实现的强制字符-数字转换方法。）接下来的两个查询通过第二个识别码片段来排序结果。第一个按照词汇方式排序，第二个是数字方式：

```

mysql> SELECT * FROM housewares3
    -> ORDER BY SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',2),'-',1);
+-----+-----+
| id      | description |
+-----+-----+
| 8-4-2-1      | microwave oven |
| 13-478-92-2 | dining table   |
| 873-48-649-63 | garbage disposal |
| 27-48-534-2 | shower stall   |
| 5764-56-89-72 | lavatory       |
| 97-681-37-66 | bedside lamp   |
+-----+-----+
mysql> SELECT * FROM housewares3
    -> ORDER BY SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',2),'-',1)+0;
+-----+-----+
| id      | description |
+-----+-----+
| 8-4-2-1      | microwave oven |
| 873-48-649-63 | garbage disposal |
| 27-48-534-2 | shower stall   |
| 5764-56-89-72 | lavatory       |
| 13-478-92-2 | dining table   |
| 97-681-37-66 | bedside lamp   |
+-----+-----+

```

这里的子串提取表达式显得混乱，但至少在我们应用表达式的数据列具有一致的数据分段。如果需要排序可变数据段的数值，工作可能更加复杂。下一节将通过一个例子说明。

7.12 按域名顺序排列主机名

Sorting Hostnames by Domain Order

问题

如果你希望按照域名顺序排序主机名，即主机名的最右边部分比最左边部分更有意义。

解决方案

将主机名拆分，且将拆分部分从右至左排序。

讨论

主机名是字符串，因此它们的自然排序方法是词汇顺序。然而，通常却期望按照域名顺序排序主机名，即主机名的最右边部分比最左边部分更有意义。假设你有一张主机名表，其中包括如下的主机名：

```
mysql> SELECT name FROM hostname ORDER BY name;
+-----+
| name |
+-----+
| cvs.php.net      |
| dbi.perl.org     |
| jakarta.apache.org |
| lists.mysql.com   |
| mysql.com        |
| www.kitebird.com  |
+-----+
```

前面的查询显示了主机名的词汇排序顺序。这与下表中所示的域名顺序是不同的：

词汇排序顺序	域名顺序
cvs.php.net	www.kitebird.com
dbi.perl.org	mysql.com
jakarta.apache.org	lists.mysql.com
lists.mysql.com	cvs.php.net
mysql.com	jakarta.apache.org
www.kitebird.com	dbi.perl.org

生成域名顺序的输出是一个子串排序问题，这里需要抽取主机名字的每一段以方便它们能够按照从右到左的方式排序。如果你的主机名数据是包括有不同数目的数据段，那么还有一点额外的麻烦，就像我们的示例主机名那样。（它们中的大多数包括有 3 段，但是 mysql.com 仅仅只有 2 段。）

为了抽取主机名的片段，可以类似前面在本章 11 节中描述的使用 SUBSTRING_INDEX() 来实现。主机名数值最大包括 3 段，我们可以按照如下的方法从左到右抽取所需要的片断：

```
SUBSTRING_INDEX(SUBSTRING_INDEX(name,'.',-3),'.',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(name,'.',-2),'.',1)
SUBSTRING_INDEX(name,'.',-1)
```

当所有的主机名都包括 3 个组成部分时，这些表达式都工作正常。但是如果主机名的组成部分少于 3 个，你可能就得不到正确的结果，如下面的查询所示：

```
mysql> SELECT name,
-> SUBSTRING_INDEX(SUBSTRING_INDEX(name,'.',-3),'.',1) AS leftmost,
-> SUBSTRING_INDEX(SUBSTRING_INDEX(name,'.',-2),'.',1) AS middle,
-> SUBSTRING_INDEX(name,'.',-1) AS rightmost
-> FROM hostname;
+-----+-----+-----+
| name | leftmost | middle | rightmost |
+-----+-----+-----+
| cvs.php.net | cvs | php | net |
| dbi.perl.org | dbi | perl | org |
| lists.mysql.com | lists | mysql | com |
| mysql.com | mysql | mysql | com |
| jakarta.apache.org | jakarta | apache | org |
| www.kitebird.com | www | kitebird | com |
+-----+-----+-----+
```

注意输出 mysql.com 的数据行，它在最左边的数据列中的数值是 mysql，实际上这里应该是空字符串。数据段抽取表达式的工作原理是抽掉最右边的 n 段数据然后返回结果中最左边的数据段。而 mysql.com 问题的根源是，如果原始数据不是 n 个数据段，那么表达式将简单返回最左边的数据段，无论原始数据有多少个数据段。为了修正这个问题，在主机名开始处添加一个周期的充分数以保证它们包括要求的数据段数：

```
mysql> SELECT name,
-> SUBSTRING_INDEX(SUBSTRING_INDEX(CONCAT('..',name),'.',-3),'.',1)
-> AS leftmost,
-> SUBSTRING_INDEX(SUBSTRING_INDEX(CONCAT('..',name),'.',-2),'.',1)
-> AS middle,
-> SUBSTRING_INDEX(name,'.',-1) AS rightmost
-> FROM hostname;
+-----+-----+-----+
| name | leftmost | middle | rightmost |
+-----+-----+-----+
| cvs.php.net | cvs | php | net |
| dbi.perl.org | dbi | perl | org |
| lists.mysql.com | lists | mysql | com |
| mysql..com | .. | mysql | com |
| jakarta.apache.org | jakarta | apache | org |
| www.kitebird.com | www | kitebird | com |
+-----+-----+-----+
```

这看起来相当难看。不过这些表达式确实是按照从右至左的形式提取了正确排序主机名所需要的子串：

```
mysql> SELECT name FROM hostname
-> ORDER BY
-> SUBSTRING_INDEX(name, '.', -1),
-> SUBSTRING_INDEX(SUBSTRING_INDEX(CONCAT('.',name), '.', -2), '.', 1),
-> SUBSTRING_INDEX(SUBSTRING_INDEX(CONCAT('..',name), '.', -3), '.', 1);
+-----+
| name      |
+-----+
| www.kitebird.com   |
| mysql.com        |
| lists.mysql.com    |
| cvs.php.net       |
| jakarta.apache.org |
| dbi.perl.org       |
+-----+
```

如果你的主机名具有最大四段而非三段数据，你需要在 ORDER BY 子句中增加另外一句 SUBSTRING_INDEX() 表达式，以在主机名开始处添加三个点。

7.13 按照数字顺序排序点分式 IP 地址

Sorting Dotted-Quadruplet IP Values in Numeric Order

问题

你希望按照数字顺序排序表示 IP 地址的字符串。

解决方案

将字符串拆分，并且按数字顺序排序各个数据片。或者只使用 INET_ATON() 函数。

讨论

如果一张表包含采用点段式表示的 IP 地址字符串，词汇顺序比数字顺序更适合它们的自然排序。为了采用数字方式替代原有排序方式，你可以按照 4 部分来排序它们，其中每一部分都按照数字方式排序。或者，为了提高效率，你可以采用 32 无符号整数来表示 IP 地址，这样可以更少的占用空间并且可以简单的采用数字方式排序。本节对两种方法都给出了演示。

为了排序字符串方式存储的点段式 IP 地址，可以采用排序主机名类似的技术，但是具有以下的不同点：

- 点段式 IP 地址总是具有四个分段，所以在提取子串之前在数值开始处添加点是不必要的。

- 点段式 IP 地址从左至右排序，所以在 ORDER BY 子句中使用的子串的顺序与在主机名排序中使用的是相反的。
- 点段式 IP 数值的每一段都是数字，所以在每一个子串之前加 0 来告知 MySQL 采用数字方式排序而非词汇方式。

假设你有一张 hostip 表，其中字符串数值表示的 ip 数据列包含 IP 数字：

```
mysql> SELECT ip FROM hostip ORDER BY ip;
+-----+
| ip   |
+-----+
| 127.0.0.1    |
| 192.168.0.10  |
| 192.168.0.2    |
| 192.168.1.10  |
| 192.168.1.2    |
| 21.0.0.1      |
| 255.255.255.255 |
+-----+
```

前面的查询得到的输出是按词汇方式排序的。为了采用数字方式排序 ip 数值，你可以提取每一段然后根据如下方式使用 ORDER BY 字句，通过加 0 将每一段转换为数字：

```
mysql> SELECT ip FROM hostip
   -> ORDER BY
   -> SUBSTRING_INDEX(ip,'.',1)+0,
   -> SUBSTRING_INDEX(SUBSTRING_INDEX(ip,'.',-3),'.',1)+0,
   -> SUBSTRING_INDEX(SUBSTRING_INDEX(ip,'.',-2),'.',1)+0,
   -> SUBSTRING_INDEX(ip,'.',-1)+0;
+-----+
| ip   |
+-----+
| 21.0.0.1    |
| 127.0.0.1    |
| 192.168.0.2    |
| 192.168.0.10   |
| 192.168.1.2    |
| 192.168.1.10   |
| 255.255.255.255 |
+-----+
```

然而，尽管 ORDER BY 得到了正确的结果，但是它也导致了大量的时间开销。这里有一个可能更简单的方法：使用 INET_ATON() 函数将字符串格式的网络地址形式直接转换为其相应的数字值然后排序这些数字：

```
mysql> SELECT ip FROM hostip ORDER BY INET_ATON(ip);
+-----+
| ip   |
+-----+
| 21.0.0.1    |
| 127.0.0.1    |
```

```
| 192.168.0.2      |
| 192.168.0.10    |
| 192.168.1.2      |
| 192.168.1.10    |
| 255.255.255.255 |
+-----+
```

如果你试图排序仅仅通过简单加 0 转换的 ip 地址并且在结果中使用 ORDER BY 子句，那么你需要考虑这种字符串到数据的转换实际得到的结果数值：

```
mysql> SELECT ip, ip+0 FROM hostip;
+-----+-----+
| ip      | ip+0   |
+-----+-----+
| 127.0.0.1 | 127   |
| 192.168.0.2 | 192.168 |
| 192.168.0.10 | 192.168 |
| 192.168.1.2 | 192.168 |
| 192.168.1.10 | 192.168 |
| 255.255.255.255 | 255.255 |
| 21.0.0.1 | 21    |
+-----+-----+
```

这种转换仅仅只保持每一个值能够被转换为有效数字的部分。剩余的部分对排序而言是无效的，尽管它对得到正确的顺序至关重要。

在 ORDER BY 子句中使用 INET_ATON() 比使用 6 次 SUBSTRING_INDEX() 调用有更高的效率。更进一步，如果你更乐于将 IP 地址视为数字方式排序而非字符串方式，那么在排序时你要完全避免执行任何转换。如果你对数据列做了索引那么你会获得额外的好处，查询优化程序可能能够对特定的查询使用这个索引。数字的 IP 地址具有 32 位，所以你能使用 INT UNSIGNED 数据列来存储它们。有些情形下你需要使用点段式格式来显示它们，可以通过 INET_NTOA() 函数转换它们。

7.14 将数值移动到排序结果的头部或尾部

Floating Values to the Head or Tail of the Sort Order

问题

你可能想要将一个数据列按通常方式排序，只是希望将一小部分数据放在排序结果的开始或结尾处。例如，假设你想要以词汇方式排序一个列表，但是确定的高优先级数据需要出现在排序结果的开始处而不管它们在普通排序结果中的位置。

解决方案

在 ORDER BY 字句中增加一个排序序列来放置这些你希望单独处理的少数数据。其余的排序数据列将对其余数据采用通常的方式处理。

讨论

如果你希望按照通常方式排序查询结果集合但是期望把特定数据放置到开始处，那么你需要创建一个额外的排序数据列，数据列 0 排序你希望单独处理的数据，数据列 1 排序其余数据。这允许你将数值放置到排序结果开始处。为了将数值放置到排序结果尾部，可以通过使用额外的数据列将数值映射为 1 而将别的数值映射为 0。

例如，当一个排序好的数据列包含 NULL 值时，MySQL 在排序结果中将它们放置在一起(升序时放置在开始处，降序时放置在结束处)。在聚类空值时看起来似乎有点奇怪，给定情形时（如接下来的查询显示的那样），它们在比较中不认为是相等的：

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

另一方面，相对于非空值 (non-NULL)，空值在概念上看起来彼此更相似，同时目前也没有好的方法将一个空值与其他空值分开。正常情形下，空值在排序的开始形成一类(或者是结束处，如果你指定 DESC 排序属性)。如果你希望将空值放置在排序结果结束的指定位置，你可以强制它们被放置在你指定的位置。假设你有一张含有如下内容的表：

```
mysql> SELECT val FROM t;
+-----+
| val |
+-----+
|    3 |
|  100 |
|  NULL |
|  NULL |
|     9 |
+-----+
```

正常情况下，升序排序时空值将被防止在排序结果的一开始：

```
mysql> SELECT val FROM t ORDER BY val;
+-----+
| val |
+-----+
|  NULL |
|  NULL |
|     3 |
|     9 |
+-----+
```

```
| 100 |  
+-----+
```

为了将它们放置在结束位置，引入一个额外的 ORDER BY 数据列将空值映射为比非空值更高的数值：

```
mysql> SELECT val FROM t ORDER BY IF(val IS NULL,1,0), val;  
+-----+  
| val |  
+-----+  
| 3 |  
| 9 |  
| 100 |  
| NULL |  
| NULL |  
+-----+
```

IF()表达式创建了一个新的数据列，它被用来作为排序的基本数值。

对于降序排序，空值在结束处聚类。为了将它们放置在开始处，使用同样的方法，只是将 IF()函数中的第 2 个参数和第 3 个参数交换，将空值映射为比非空值更低的数值：

```
IF(val IS NULL,0,1)
```

同样的方法对将空值以外的其余数值放置在排序的开始或结束也有效。假设你想将 mail 表信息按照发送者/接收者排序，但是你希望将特定发送者的信息放置到最开始。真实世界中，最有趣的发送者可能是邮局或者根用户。这些名字不会在 mail 表中出现，所以我们使用 phil 来替代有趣的姓名：

```
mysql> SELECT t, srcuser, dstuser, size  
-> FROM mail  
-> ORDER BY IF(srcuser='phil',0,1), srcuser, dstuser;  
+-----+-----+-----+-----+  
| t      | srcuser | dstuser | size |  
+-----+-----+-----+-----+  
| 2006-05-16 23:04:19 | phil    | barb    | 10294 |  
| 2006-05-12 15:02:49 | phil    | phil    | 1048  |  
| 2006-05-15 08:50:57 | phil    | phil    | 978   |  
| 2006-05-14 11:52:17 | phil    | tricia  | 5781  |  
| 2006-05-17 12:49:23 | phil    | tricia  | 873   |  
| 2006-05-14 14:42:21 | barb    | barb    | 98151 |  
| 2006-05-11 10:15:08 | barb    | tricia  | 58274 |  
| 2006-05-13 13:59:18 | barb    | tricia  | 271   |  
| 2006-05-14 09:31:37 | gene    | barb    | 2291  |  
| 2006-05-16 09:00:28 | gene    | barb    | 613   |  
| 2006-05-15 17:35:31 | gene    | gene    | 3856  |  
| 2006-05-15 07:17:48 | gene    | gene    | 3824  |  
| 2006-05-19 22:21:51 | gene    | gene    | 23992 |  
| 2006-05-15 10:25:52 | gene    | tricia  | 998532 |  
| 2006-05-12 12:48:13 | tricia  | gene    | 194925 |  
| 2006-05-14 17:03:01 | tricia  | phil    | 2394482 |  
+-----+-----+-----+-----+
```

如果数据行中 `srcuser` 项值为 `phil`, 那么其额外排序列中对应的值就为 0, 其他情形下其数值为 1。通过创建最重要的排序列, 可以将 `phil` 发送信息的数据行放置到输出的最上方。(如果希望将它们放置在最底部, 可以使用 `DESC` 来反向排序数据列, 或者将 `IF()` 函数的第 2 个和第 3 个参数位置互换。)

你也可以对特定条件使用这个方法, 而不只是指定数值。如果希望将人们发送给自己信息的数据行放置在开始处, 可以这样做:

```
mysql> SELECT t, srcuser, dstuser, size
-> FROM mail
-> ORDER BY IF(srcuser=dstuser,0,1), srcuser, dstuser;
+-----+-----+-----+-----+
| t      | srcuser | dstuser | size   |
+-----+-----+-----+-----+
| 2006-05-14 14:42:21 | barb   | barb   | 98151 |
| 2006-05-19 22:21:51 | gene   | gene   | 23992 |
| 2006-05-15 17:35:31 | gene   | gene   | 3856  |
| 2006-05-15 07:17:48 | gene   | gene   | 3824  |
| 2006-05-12 15:02:49 | phil   | phil   | 1048  |
| 2006-05-15 08:50:57 | phil   | phil   | 978   |
| 2006-05-11 10:15:08 | barb   | tricia | 58274 |
| 2006-05-13 13:59:18 | barb   | tricia | 271   |
| 2006-05-16 09:00:28 | gene   | barb   | 613   |
| 2006-05-14 09:31:37 | gene   | barb   | 2291  |
| 2006-05-15 10:25:52 | gene   | tricia | 998532|
| 2006-05-16 23:04:19 | phil   | barb   | 10294 |
| 2006-05-14 11:52:17 | phil   | tricia | 5781  |
| 2006-05-17 12:49:23 | phil   | tricia | 873   |
| 2006-05-12 12:48:13 | tricia | gene   | 194925|
| 2006-05-14 17:03:01 | tricia | phil   | 2394482|
+-----+-----+-----+-----+
```

如果你对你的表内容有相当好的想法, 有时你可以消除额外的排序数据列。例如, 在 `mail` 表中 `srcuser` 是不可能为空值, 因此以前的查询可以按照如下在 `ORDER BY` 子句中少使用一个数据列的方式重写(这依赖于将空值排序在非空值之前的属性):

```
mysql> SELECT t, srcuser, dstuser, size
-> FROM mail
-> ORDER BY IF(srcuser=dstuser,NULL,srcuser), dstuser;
+-----+-----+-----+-----+
| t      | srcuser | dstuser | size   |
+-----+-----+-----+-----+
| 2006-05-14 14:42:21 | barb   | barb   | 98151 |
| 2006-05-19 22:21:51 | gene   | gene   | 23992 |
| 2006-05-15 17:35:31 | gene   | gene   | 3856  |
| 2006-05-15 07:17:48 | gene   | gene   | 3824  |
| 2006-05-12 15:02:49 | phil   | phil   | 1048  |
| 2006-05-15 08:50:57 | phil   | phil   | 978   |
| 2006-05-11 10:15:08 | barb   | tricia | 58274 |
| 2006-05-13 13:59:18 | barb   | tricia | 271   |
| 2006-05-16 09:00:28 | gene   | barb   | 613   |
| 2006-05-14 09:31:37 | gene   | barb   | 2291  |
+-----+-----+-----+-----+
```

```
| 2006-05-15 10:25:52 | gene    | tricia   | 998532 |
| 2006-05-16 23:04:19 | phil    | barb     | 10294  |
| 2006-05-14 11:52:17 | phil    | tricia   | 5781   |
| 2006-05-17 12:49:23 | phil    | tricia   | 873    |
| 2006-05-12 12:48:13 | tricia | gene     | 194925 |
| 2006-05-14 17:03:01 | tricia | phil     | 2394482 |
+-----+-----+-----+-----+
```

参考

引入额外排序数据列的方法对生成多 SELECT 语句联合的 UNION 查询非常有效。你可以将 SELECT 的结果一个接一个的排列起来并且在每一个 SELECT 个体内部排序数据行。具体细节请查阅 12.12 节。

7.15 按照用户定义排序

Sorting in User-Defined Orders

问题

如果你希望对一个数据列值定义一个非标准的排序顺序。

解决方案

使用 FIELD() 函数将数据列数值映射为按照期望顺序排列的序列。

讨论

7.14 节展示了如何将特定的一组数据行放置到排序结果的头部。如果你希望将特定的顺序施加到整个数据列上，那么使用 FIELD() 函数将它们映射到一个数值列表并且使用数字方式排序。FIELD() 函数将它的第一个参数和后面的参数比较并返回一个数字指明与哪一个匹配。下面的 FIELD() 调用将目标数值与 str1、str2、str3 和 str4 比较，返回 1、2、3 或者 4，取决于与哪一个数值匹配：

```
FIELD(value,str1,str2,str3,str4)
```

比较使用的数值个数不一定是 4；FIELD() 函数的参数列表是变长的。如果数值是空值或者没有数值可以匹配，FILED() 函数返回 0。

FIELD() 函数可以按照任意你想要的顺序排序任一数据集。例如，按照 Henry、Suzi、Ben 的顺序显示 driver_log 表，可以这样做：

```
mysql> SELECT * FROM driver_log
-> ORDER BY FIELD(name,'Henry','Suzi','Ben');
+-----+-----+-----+-----+
| rec_id | name  | trav_date | miles |
+-----+-----+-----+-----+
```

10	Henry	2006-08-30	203
8	Henry	2006-09-01	197
6	Henry	2006-08-26	115
4	Henry	2006-08-27	96
3	Henry	2006-08-29	300
7	Suzi	2006-08-29	391
5	Ben	2006-08-29	131
9	Ben	2006-09-02	79
1	Ben	2006-08-30	152

你也可以对数据列子串使用 FIELD() 函数。为了按照制造商国标识码 US、UK、JP、SG 的顺序排序 housewares 表中的项目，可以这样做：

```
mysql> SELECT id, description FROM housewares
-> ORDER BY FIELD(RIGHT(id,2),'US','UK','JP','SG');
+-----+-----+
| id      | description |
+-----+-----+
| DIN40672US | dining table   |
| BTH00485US | shower stall    |
| KIT00372UK | garbage disposal |
| KIT01729JP | microwave oven  |
| BTH00415JP | lavatory        |
| BED00038SG | bedside lamp     |
+-----+-----+
```

更一般的，对任意不能按照自然顺序排序的数据分类，FILED() 函数可以用来将这些基于分类的数值按照给定顺序的排序。

7.16 排序枚举数值

Sorting ENUM Values

问题

ENUM（枚举）数值不能像其他字符串数据列一样排序。

解决方案

了解它们如何工作，然后利用这些属性作为你自己的优势。

讨论

ENUM 被认为是一个字符串数据类型，但是 ENUM 数值实际上是按照数字方式存储的，并且按照表定义中列举的顺序排序数值。这些数字值影响枚举值是怎样存储的，这一点非常有用。假设你有一样名为 weekday 的表，其中包含一个枚举数据列，数据列以 1 周 7 天的名字作为它的成员：

```
CREATE TABLE weekday
(
```

```
day ENUM('Sunday', 'Monday', 'Tuesday', 'Wednesday',
         'Thursday', 'Friday', 'Saturday')
);
```

在 MySQL 内部，已经定义了从 Sunday 到 Saturday 的枚举数值，在定义中具有从 1 到 7 的数值。为了让你自己看清楚这一点，使用刚才展示的定义创建一张表，然后对一周的每一天插入一个数据行。然而，为了让这个插入顺序与排序顺序不同（这样你就可以看到排序的效果），以随机的顺序增加那一天：

```
mysql> INSERT INTO weekday (day) VALUES('Monday'), ('Friday'),
-> ('Tuesday'), ('Sunday'), ('Thursday'), ('Saturday'), ('Wednesday');
```

然后，选择数值，同时包括字符串方式的数值和数字方式的数值（后者通过使用+0 的字符串到数字的转换获取）：

```
mysql> SELECT day, day+0 FROM weekday;
+-----+-----+
| day | day+0 |
+-----+-----+
| Monday | 2 |
| Friday | 6 |
| Tuesday | 3 |
| Sunday | 1 |
| Thursday | 5 |
| Saturday | 7 |
| Wednesday | 4 |
+-----+-----+
```

注意因为查询没有包含 ORDER BY 子句，返回的数据行是无序的。如果添加了一个 ORDER BY day 子句，显而易见，MySQL 使用内部的数字值进行排序：

```
mysql> SELECT day, day+0 FROM weekday ORDER BY day;
+-----+-----+
| day | day+0 |
+-----+-----+
| Sunday | 1 |
| Monday | 2 |
| Tuesday | 3 |
| Wednesday | 4 |
| Thursday | 5 |
| Friday | 6 |
| Saturday | 7 |
+-----+-----+
```

如果你偶尔希望按照词汇方式排序 ENUM 数值该如何做？通过使用 CAST() 函数强制它们被当作字符串来排序：

```
mysql> SELECT day, day+0 FROM weekday ORDER BY CAST(day AS CHAR);
+-----+-----+
| day | day+0 |
+-----+-----+
| Friday | 6 |
| Monday | 2 |
```

```

+-----+
| Saturday |      7 |
| Sunday   |      1 |
| Thursday |      5 |
| Tuesday  |      3 |
| Wednesday|      4 |
+-----+

```

如果你总是（或者几乎总是）按照指定的非词汇顺序排序非枚举数据列，考虑将数据类型变换为 ENUM，将它的数据按照期望的顺序列举出来。为了看明白它是怎么工作的，创建一张 color 表，其中包括一个字符串数据列，并且在其中放置一些示例数据行：

```

mysql> CREATE TABLE color (name CHAR(10));
mysql> INSERT INTO color (name) VALUES ('blue'),('green'),
-> ('indigo'),('orange'),('red'),('violet'),('yellow');

```

在这一点排序名字数据列得到了词汇顺序因为这个数据列包括 CHAR 数值：

```

mysql> SELECT name FROM color ORDER BY name;
+-----+
| name  |
+-----+
| blue  |
| green |
| indigo|
| orange|
| red   |
| violet|
| yellow|
+-----+

```

现在假设你希望按照彩虹顺序排序数据列。（这个顺序可以通过“Roy G. Biv”这个名字给出，这个名字中的相继的字符代表相应的颜色名字。）一个得到彩虹排序的方式是使用 FIELD() 函数：

```

mysql> SELECT name FROM color
-> ORDER BY
-> FIELD(name,'red','orange','yellow','green','blue','indigo','violet');
+-----+
| name  |
+-----+
| red   |
| orange|
| yellow|
| green  |
| blue   |
| indigo|
| violet|
+-----+

```

如果不想使用 FIELD() 函数还需要完成同样的任务，可以使用 ALTER TABLE 将名字数据列转换为一个 ENUM 类型，将色彩名字按照期望的排序顺序列举出：

```
mysql> ALTER TABLE color
-> MODIFY name
-> ENUM('red','orange','yellow','green','blue','indigo','violet')
```

在转换这张表之后，可以在名字数据列上得到自然的排序而无需特殊对待：

```
mysql> SELECT name FROM color ORDER BY name;
+-----+
| name |
+-----+
| red  |
| orange |
| yellow |
| green |
| blue |
| indigo |
| violet |
+-----+
```

生成摘要

Generating Summaries

8.0 引言

Introduction

数据库系统是用于存储和查询数据记录的，但是它们还能以更简洁的形式为你的数据生成摘要。当你想要所有的图片而不是图片中的细节时，摘要就非常有用了。另外摘要也比一长串的记录列表更容易理解。摘要技术让你能够回答类似于“有多少？”、“总数是多少？”或“数值范围是多少？”这样的问题。当你在处理商业事务时，你可能想要知道每个州有多少顾客，或者每个月你有多少销售额？你可以通过建立一张客户记录并自己动手计算来确定每个州的数目，但是当 MySQL 能够替你完成这些工作时，自己动手就显得毫无意义。同样，为了确定每月的销售额，如果需要自己动手根据原始订单的信息记录列表来得到总数没有任何特殊的用处，让 MySQL 来完成它吧。

刚才提及的范例说明了两个常见的摘要类型。第一个（每个州的客户数目记录）是一个计算摘要。每个记录的内容只有在被放置到正确的分组或分类中用于计算时才是重要的。这样的摘要本质是柱状图，在其中你将各个项目排序放置到一套分组中并在每个分组中计算项目的数目。第二个示例（每月销售额）是一个基于记录内容的摘要实例——销售总额是通过每一个单独订单记录的销售数额计算得到的。

也有其他种类的摘要既不生成计数也不得到总数，而是一张简单的数值唯一确定的列表。如果你不关心目前每个数值有多少实例而只关心目前每个数值是多少，它就非常有用。如果你想知道你在哪些州拥有客户，你需要一张记录了包含每个州的名字的列表，而非一张由每个记录中的州数值组成的列表。有时候，甚至将摘要技术应用在另一个摘要的结果上也非常有用。例如，为了确定你的客户居住在多少个州中，可以生成一张唯一的客户居住州列表，然后对它计数。

你生成摘要的类型可能依赖于你需要处理的数据种类。计数摘要可以从任何类型的数值中

得到，它们可能是数字、字符串或者日期。如果需要得到包括总数或平均数的摘要，那么只有数字型数值能被使用。你能够通过计数客户州名字的实例得到一张你的客户基数的人口统计分析，但是你不能对州名字求总数或者平均——那没有任何意义。

MySQL 中的摘要操作包含以下的 SQL 结构：

- 为了从一组单独数值中计算摘要数值，可以使用已知聚类函数中的一个。之所以这样称呼是因为它们都用于聚类（分组）数值的操作。聚类函数包括 COUNT() 函数，用于在查询结果中计算数据行或数值；MIN() 和 MAX() 函数，用于查找最小和最大数值；以及 SUM() 和 AVG() 函数，用于得到数值的总数和平均值。这些函数能够被用于计算整个结果集的数值，或者使用 GROUP BY 子句将数据行聚类到子集并且为每个子集得到一聚类数值。
- 为了获取唯一数值列表，使用 SELECT DISTINCT 比使用 SELECT 更好。
- 为了计算有多少不同的数值，使用 COUNT(DISTINCT) 比使用 COUNT() 更好。

本章各节一开始举例说明了基本摘要用法，并展示了如何执行更复杂的摘要操作。在后续的章节中你将看见更多的摘要方法的范例，特别是那些包含联合与统计操作的范例（详见 12 章和 13 章）。

摘要查询有时包含复杂表达式。对那些你经常执行的摘要操作，紧记视图可以让查询更容易。13.12 节展示了创建视图的基本技术。本章第 1 节显示了如何应用它来简化摘要，同时在本章后续部分中你将更容易的看见它是怎样被应用的。

本章中作为基本范例使用的表是 `drvier_log` 表和 `mail` 表。它们在第 7 章中被广泛使用，所以看起来应该比较熟悉。第三个贯穿本章的表是 `states`，其数据行包括很少的列，用以记录美国每个州的信息：

```
mysql> SELECT * FROM states ORDER BY name;
+-----+-----+-----+-----+
| name | abbrev | statehood | pop |
+-----+-----+-----+-----+
| Alabama | AL | 1819-12-14 | 4530182 |
| Alaska | AK | 1959-01-03 | 655435 |
| Arizona | AZ | 1912-02-14 | 5743834 |
| Arkansas | AR | 1836-06-15 | 2752629 |
| California | CA | 1850-09-09 | 35893799 |
| Colorado | CO | 1876-08-01 | 4601403 |
| Connecticut | CT | 1788-01-09 | 3503604 |
...

```

`name` 和 `abbrev` 列给出了完整的州名和相应的缩写。`statehood` 列指明了该州加入美利坚合众国的日期。`pop` 列是到 2004 年 7 月时相应州的人口，数据来自美国人口调查局。

本章偶尔也使用其他表。你可以通过在章节分布图表目录中找到的描述来创建其中的大部分表。第 5 章 15 节描述了 `kjv` 表。

8.1 使用 COUNT 函数生成摘要

Summarizing with COUNT()

问题

你想计算一张表中满足确定条件的数据行数目，或者特定数值发生的次数。

解决方案

使用 `COUNT()` 函数。

讨论

为了在整张表或在特定条件下计算数据行的数目，可以使用 `COUNT()` 函数。例如，为了显示一张表中的数据行内容，你可以使用 `SELECT *` 子句，但是如果需要对它们计数，可以使用 `SELECT COUNT(*)`。如果没有 `WHERE` 子句，那么该语句将计算整张表中的数据行，例如下面的语句显示 `driver_log` 表包含了多少数据行：

```
mysql> SELECT COUNT(*) FROM driver_log;
+-----+
| COUNT(*) |
+-----+
|      10 |
+-----+
```

如果你不知道美国有多少个州，这条语句可以告诉你：

```
mysql> SELECT COUNT(*) FROM states;
+-----+
| COUNT(*) |
+-----+
|      50 |
+-----+
```

没有 `WHERE` 子句的 `COUNT(*)` 对 MyISAM 表来说是非常快的。然而，对于 BDB 或 InnoDB 表而言，你可能想要避免使用它，因为这条语句要求执行完整的表扫描，对于巨大的表来说可能会非常缓慢。如果你所需要的的是一个近似的数据行计数，一个避免完整扫描存储引擎的工作方法是从 `INFORMATION_SCHEMA` 数据库中提取 `TABLE_ROWS` 数值：

```
mysql> SELECT TABLE_ROWS FROM INFORMATION_SCHEMA.TABLES
-> WHERE TABLE_SCHEMA = 'cookbook' AND TABLE_NAME = 'states';
+-----+
| TABLE_ROWS |
+-----+
|      50 |
+-----+
```

在 MySQL5.0 版本之前，INFORMATION_SCHEMA 是不可用的。取而代之的是，可以使用 SHOW TABLE STATUS 并提取 Rows 数据列的数值。

如果仅仅想计算满足确定条件的数据行数目，可以在 SELECT COUNT(*) 语句中包含正确的 WHERE 子句。这些条件可以使 COUNT(*) 能够用于回答许多类型的问题：

- 司机们有多少次一天行驶超过 200 英里？

```
mysql> SELECT COUNT(*) FROM driver_log WHERE miles > 200;
+-----+
| COUNT(*) |
+-----+
|      4 |
+-----+
```

- Suzi 行驶了多少天？

```
mysql> SELECT COUNT(*) FROM driver_log WHERE name = 'Suzi';
+-----+
| COUNT(*) |
+-----+
|      2 |
+-----+
```

- 在 20 世纪之初美国有多少个州？

```
mysql> SELECT COUNT(*) FROM states WHERE statehood < '1900-01-01';
+-----+
| COUNT(*) |
+-----+
|      45 |
+-----+
```

- 有多少个州是在 19 世纪加入美利坚合众国的？

```
mysql> SELECT COUNT(*) FROM states
-> WHERE statehood BETWEEN '1800-01-01' AND '1899-12-31';
+-----+
| COUNT(*) |
+-----+
|      29 |
+-----+
```

COUNT() 函数实际上有两种形式。我们使用过的形式 COUNT(*)，用以计数数据行。另一种形式 COUNT(expr)，接受一个数据列名称或者表达式参数并计算所有非空值数目。

下面的语句显示如何同时生成一张表的数据行计数和其中一个数据列的非空值计数：

```
SELECT COUNT(*), COUNT(mycol) FROM mytbl;
```

COUNT(expr)不计数空值的事实对于从同一个数据行集合中生成多重计数非常有用。仅使用一条语句从 driver_log 中计数 Saturday 和 Sunday 数目，可以这样做：

```
mysql> SELECT
-> COUNT(IF(DAYOFWEEK(trav_date)=7,1,NULL)) AS 'Saturday trips',
-> COUNT(IF(DAYOFWEEK(trav_date)=1,1,NULL)) AS 'Sunday trips'
-> FROM driver_log;
+-----+-----+
| Saturday trips | Sunday trips |
+-----+-----+
|            3 |          1 |
+-----+-----+
```

或者为了计算周末和周中行程的对比，这样做：

```
mysql> SELECT
-> COUNT(IF(DAYOFWEEK(trav_date) IN (1,7),1,NULL)) AS 'weekend trips',
-> COUNT(IF(DAYOFWEEK(trav_date) IN (1,7),NULL,1)) AS 'weekday trips'
-> FROM driver_log;
+-----+-----+
| weekend trips | weekday trips |
+-----+-----+
|           4 |          6 |
+-----+-----+
```

IF()表示决定了每一个数据列数值是否应该被计数。如果需要，表达式的计算结果为 1 且 COUNT()函数会对相应数据列计数。如果不需，表达式计算结果为空且 COUNT()函数会忽略该数据列。结果是计算满足 IF()函数第一个参数给定条件的数值数目。

创建视图来简化使用摘要

如果你经常需要一个给定的摘要，那么视图可以帮助你不必重复输入摘要表达式。例如，下面的视图实现了周中和周末旅程的摘要：

```
mysql> CREATE VIEW trip_summary_view AS
-> SELECT
-> COUNT(IF(DAYOFWEEK(trav_date) IN (1,7),1,NULL)) AS weekend trips,
-> COUNT(IF(DAYOFWEEK(trav_date) IN (1,7),NULL,1)) AS weekday trips
-> FROM driver_log;
```

从该视图中选取数值比直接从更低一层的表中选取数据容易得多：

```
mysql> SELECT * FROM trip_summary_view;
+-----+-----+
| weekend_trips | weekday_trips |
+-----+-----+
|           4 |          6 |
+-----+-----+
```

参考

本章第 8 节进一步讨论了 COUNT(*) 和 COUNT(expr) 的区别。

8.2 使用 MIN() 和 MAX() 函数生成摘要

Summarizing with MIN() and MAX()

问题

当你需要确定一个数值集中最小或最大值时。

解决方案

使用 MIN() 函数查找最小数值，MAX() 函数查找最大数值。

讨论

查找最小或者最大值与排序有点类似。与得到整个排序数值集不同的是，你仅仅需要选择排序范围内某端的一个单独的数值。这种类型的操作应用在类似查找最小、最大、最老、最新、最贵、最便宜等请求上。查找类似数值的一个方法是使用 MIN() 和 MAX() 函数。（另一种解决这些问题的方法是使用 LIMIT 函数，详细讨论参见 3.14 节和 3.16 节）

因为 MIN() 和 MAX() 函数确定了集合中的极值，所以它们对于确定数值范围非常有用：

- 在 mail 表中数据行用来表示什么样的日期范围？发送的最小和最大信息是什么？

```
mysql> SELECT
    -> MIN(t) AS earliest, MAX(t) AS latest,
    -> MIN(size) AS smallest, MAX(size) AS largest
    -> FROM mail;
+-----+-----+-----+-----+
| earliest | latest | smallest | largest |
+-----+-----+-----+-----+
| 2006-05-11 10:15:08 | 2006-05-19 22:21:51 | 271 | 2394482 |
+-----+-----+-----+-----+
```

- driver_log table 中最长和最短的行程分别是多少？

```
mysql> SELECT MIN(miles) AS shortest, MAX(miles) AS longest
    -> FROM driver_log;
+-----+-----+
| shortest | longest |
+-----+-----+
```

- 美国人口最多和最少的州分别是哪个州？

```
mysql> SELECT MIN(pop) AS 'fewest people', MAX(pop) AS 'most people'
-> FROM states;
+-----+-----+
| fewest people | most people |
+-----+-----+
|      506529 |     35893799 |
+-----+-----+
```

- 就词汇来说，第一个和最后一个州名分别是什么？

```
mysql> SELECT MIN(name), MAX(name) FROM states;
+-----+-----+
| MIN(name) | MAX(name) |
+-----+-----+
| Alabama   | Wyoming   |
+-----+-----+
```

MIN()函数和MAX()函数不需要被直接应用于数据列操作。它们也可以用于来自数据列的表达式或数值。例如，为了查找长度最短和最长的州名，可以这样做：

```
mysql> SELECT
-> MIN(CHAR_LENGTH(name)) AS shortest,
-> MAX(CHAR_LENGTH(name)) AS longest
-> FROM states;
+-----+-----+
| shortest | longest |
+-----+-----+
|      4 |     14 |
+-----+-----+
```

8.3 使用 SUM()和 AVG()函数生成摘要

Summarizing with SUM() and AVG()

问题

你需要得到一个集合的数目或者查找它们的平均值。

解决方案

使用SUM()函数或者AVG()函数。

讨论

SUM()函数和AVG()函数生成一个集合数值的总数和平均值：

- 信件传递的总数是多少？每个信息的平均大小是多少？

```
mysql> SELECT
-> SUM(size) AS 'total traffic',
```

```

-> AVG(size) AS 'average message size'
-> FROM mail;
+-----+-----+
| total traffic | average message size |
+-----+-----+
|      3798185 |          237386.5625 |
+-----+-----+

```

- driver_log 表中的司机行驶了多少英里？平均每天行驶了多少英里？

```

mysql> SELECT
-> SUM(miles) AS 'total miles',
-> AVG(miles) AS 'average miles/day'
-> FROM driver_log;
+-----+-----+
| total miles | average miles/day |
+-----+-----+
|      2166 |          216.6000 |
+-----+-----+

```

- 美国的总人口是多少？

```

mysql> SELECT SUM(pop) FROM states;
+-----+
| SUM(pop) |
+-----+
| 293101881 |
+-----+

```

数值表示的是 2004 年 7 月报告的人口数。这里显示的数字与美国人口调查局报告的美国人口不同，因为州表并不包含华盛顿特区的数字。

SUM() 函数和 AVG() 函数是很严格的数字函数，因此它们不能被用于字符串或者时间数据。另一方面，你有时能够将非数字数据转换为可用的数字形式。假设这里有一张存储表示已流逝时间的 TIME 类型数据：

```

mysql> SELECT t1 FROM time_val;
+-----+
| t1    |
+-----+
| 15:00:00 |
| 05:01:30 |
| 12:30:20 |
+-----+

```

为了计算已流逝时间的总数，在对它们求和之前使用 TIME_TO_SEC() 函数将 TIME 类型数据转换为以秒为单位的数据，求和返回的结果也将是以秒为单位的数据。然后将结果传递给 SEC_TO_TIME() 函数将其转换位 TIME 数据格式：

```

mysql> SELECT SUM(TIME_TO_SEC(t1)) AS 'total seconds',
-> SEC_TO_TIME(SUM(TIME_TO_SEC(t1))) AS 'total time'
-> FROM time_val;
+-----+-----+

```

```
| total seconds | total time |
+-----+-----+
|       117110 | 32:31:50  |
+-----+-----+
```

参考

SUM() 函数和 AVG() 函数在统计计算应用方面特别有用。13 章中，它们将与 STD() 函数一起进一步被探索，其中 STD() 是一个用于计算标准差的相关函数。

8.4 使用 DISTINCT 函数消除重复

SELECT DISTINCT name FROM driver_log;

问题

你想知道哪一个数值目前在数据集合中，但不希望重复数据被多次显示。或者你想要知道有多少个独立的数据。

解决方案

使用 DISTINCT 来选择唯一的数值或者使用 COUNT(DISTINCT) 来计数它们。

讨论

一个没有使用聚类函数的摘要操作是通过消除重复数值来决定哪些数据或者数据行是被包括在数据集合中的。使用 DISTINCT（或者 DISTINCTROW，它们是同义的）来完成这个任务。DISTINCT 对于摘要查询结果非常有用，并且常常被整合应用于 ORDER BY 子句中将数据以更加有意义的顺序放置。例如，为了确定 driver_log 表中列出的司机姓名，可以使用如下的语句：

```
mysql> SELECT DISTINCT name FROM driver_log ORDER BY name;
+-----+
| name  |
+-----+
| Ben   |
| Henry |
| Suzi  |
+-----+
```

没有 DISTINCT 的语句生成同样的姓名，但是即便是小的数据集合也不是很容易理解：

```
mysql> SELECT name FROM driver_log;
+-----+
| name  |
+-----+
| Ben   |
| Suzi  |
| Henry |
+-----+
```

```
| Henry |
| Ben   |
| Henry |
| Suzi  |
| Henry |
| Ben   |
| Henry |
+-----+
```

为了确定有多少不同的司机，使用 COUNT(DISTINCT)：

```
mysql> SELECT COUNT(DISTINCT name) FROM driver_log;
+-----+
| COUNT(DISTINCT name) |
+-----+
|            3 |
+-----+
```

COUNT(DISTINCT)忽略空值。如果你希望出现的空值也是集合中数据的一种，使用如下的表达：

```
COUNT(DISTINCT val) + IF(COUNT(IF(val IS NULL, 1, NULL))=0, 0, 1)
COUNT(DISTINCT val) + IF(SUM(ISNULL(val))=0, 0, 1)
COUNT(DISTINCT val) + (SUM(ISNULL(val))!=0)
```

DISTINCT 查询常常与聚类函数联合使用来获取你的数据中更加完备的特性。假设你有一张 customer 表，其中包含客户具体位于哪个州的 state 数据列信息。在这张 customer 表上应用 COUNT(*) 指出你有多少客户，在 state 的数据之上使用 DISTINCT 能获知你拥有客户的州的数目，在 state 数据上使用 COUNT(DISTINCT) 能获知你的客户基数表示多少个州。

当用于多数据列时，DISTINCT 表示了数据列中不同的数据联合，COUNT(DISTINCT) 对联合的数目计数。接下来的语句表示了 mail 表中不同的发送/接收对，同时也显示了有多少个这样的数据对：

```
mysql> SELECT DISTINCT srcuser, dstuser FROM mail
    -> ORDER BY srcuser, dstuser;
+-----+-----+
| srcuser | dstuser |
+-----+-----+
| barb   | barb   |
| barb   | tricia |
| gene   | barb   |
| gene   | gene   |
| gene   | tricia |
| phil   | barb   |
| phil   | phil   |
| phil   | tricia |
| tricia | gene   |
| tricia | phil   |
+-----+-----+
mysql> SELECT COUNT(DISTINCT srcuser, dstuser) FROM mail;
```

```
+-----+  
| COUNT(DISTINCT srcuser, dstuser) |  
+-----+  
| 10 |  
+-----+
```

DISTINCT 也可以与表达式一起工作，而不仅仅只是作用在数据列上。为了确定 mail 表中信息发送在一天中不同的小时数，可以计数不同的 HOUR() 函数得到的数据：

```
mysql> SELECT COUNT(DISTINCT HOUR(t)) FROM mail;  
+-----+  
| COUNT(DISTINCT HOUR(t)) |  
+-----+  
| 12 |  
+-----+
```

为了找出这些小时具体都是哪些，可以列出他们：

```
mysql> SELECT DISTINCT HOUR(t) AS hour FROM mail ORDER BY hour;  
+-----+  
| hour |  
+-----+  
| 7 |  
| 8 |  
| 9 |  
| 10 |  
| 11 |  
| 12 |  
| 13 |  
| 14 |  
| 15 |  
| 17 |  
| 22 |  
| 23 |  
+-----+
```

注意这条语句没有告诉你每个小时发送了多少信息。这将在本章第 15 节提及。

8.5 查找数值相关的最大值和最小值

Finding Values Associated with Minimum and Maximum Values

问题

当你想知道包括最小或者最大数值的数据行的其他数据列的数值。

解决方案

使用两条语句和一个用户定义变量，使用子查询或者使用连接。

讨论

MIN() 函数和 MAX() 函数查找数据范围的端点，但是当有时查找最小或者最大数值时，你可能也对数值出现的数据行中的其他数据感兴趣。例如，你可以这样查找最大人口数目的州：

```
mysql> SELECT MAX(pop) FROM states;
+-----+
| MAX(pop) |
+-----+
| 35893799 |
+-----+
```

但是这并不能告诉你哪一个州拥有这个数目的人口。可以通过下面的语句尝试显式地获取这个信息：

```
mysql> SELECT MAX(pop), name FROM states WHERE pop = MAX(pop);
error 1111(hy000): Invalid use of group function
```

也许每个人迟早都会像这样尝试，但是这样并不能正常工作。像 MIN() 和 MAX() 这样的聚类函数并不能在 WHERE 子句中使用，这里要求是能够应用于单个数据行的表达式。这条语句的意图是确定哪一个数据行具有最大人口的数值，同时显示相关的州的名字。问题在于尽管你我都很清楚的知道我们写下的语句意味着什么，但是对 MySQL 却毫无意义。这条语句失败的地方在于 MySQL 使用了 WHERE 子句来确定选择哪一个数据行，但是它所知道的聚类函数的数值却必需是从确定的函数值的数据行中选取的！所以，在某种意义上，这条语句是自相矛盾的。你可以通过将最大人口数值存储在用户定义的变量中，然后将数据行与该变量数值相比较来解决该问题：

```
mysql> SET @max = (SELECT MAX(pop) FROM states);
mysql> SELECT pop AS 'highest population', name FROM states WHERE pop = @max;
+-----+-----+
| highest population | name   |
+-----+-----+
|           35893799 | California |
+-----+-----+
```

对于单语句解决方案，可以在 WHERE 子句中使用子查询来返回最大人口数值：

```
mysql> SELECT pop AS 'highest population', name FROM states
      -> WHERE pop = (SELECT MAX(pop) FROM states);
+-----+-----+
| highest population | name   |
+-----+-----+
|           35893799 | California |
+-----+-----+
```

甚至最小或者最大数值本身并没有实际包含在数据行中时，该方法仍然也能运作，但是仅仅只是由它衍生而来的方法。如果你想知道国王詹姆斯译文中最短的诗，这是很容易找到的：

```
mysql> SELECT MIN(CHAR_LENGTH(vtext)) FROM kjv;
+-----+
| MIN(CHAR_LENGTH(vtext)) |
+-----+
|          11           |
+-----+
```

如果你想知道“那是什么诗”，可以这样做：

```
mysql> SELECT bname, cnum, vnum, vtext FROM kjv
-> WHERE CHAR_LENGTH(vtext) = (SELECT MIN(CHAR_LENGTH(vtext)) FROM kjv);
+-----+-----+-----+-----+
| bname | cnum | vnum | vtext      |
+-----+-----+-----+-----+
| John  |   11 |   35 | Jesus wept. |
+-----+-----+-----+-----+
```

还有另一种从包含最小或者最大数值的数据行中选取其他数据列的方法，这就是使用连接。将选择的数据放置到另一张表中，然后将该表与原始表连接来选择匹配该数值的数据行。为了查找具有最大人口数目的州，可以这样使用连接：

```
mysql> CREATE TABLE t SELECT MAX(pop) as maxpop FROM states;
mysql> SELECT states.* FROM states INNER JOIN t ON states.pop = t.maxpop;
+-----+-----+-----+-----+
| name    | abbrev | statehood | pop     |
+-----+-----+-----+-----+
| California | CA      | 1850-09-09 | 35893799 |
+-----+-----+-----+-----+
```

参考

关于连接的更多信息，请看 12 章。特别是 12.6 节，进一步讨论了查找包含分类最小或最大数值的数据行的问题。

8.6 控制 MIN() 函数和 MAX() 函数的字符串大小写区分

Controlling String Case Sensitivity for MIN() and MAX()

问题

可能 MIN() 函数和 MAX() 函数选择字符串时是按照大小写区分样式但你并不希望它们这样做，反之亦然。

解决方案

改变字符串的比较特性。

讨论

第 5 章讨论了字符串的比较属性是否依赖于字符串是二进制或者非二进制：

- 二进制字符串是字节序列。它们通过使用字节的数字值逐一比较。字符集和字母对于比较没有任何意义。
- 非二进制字符串是字母序列。它们有字符集和 Collation，通过定义的 Collation 顺序对字符进行逐一对比。

当你把字符串数据列作为 MIN() 函数或者 MAX() 函数的参数时，这些属性也能适用，因为它们是基于比较的。为了改变这些函数对字符串数据列工作的方式，你必须改变数据列比较的属性。第 5 章第 9 节讨论了如何控制这些属性，同时第 7 章第 4 节显示了它们是怎样应用于字符串排序的。同样的规则可以应用于查找最小和最大字符串数值，所以我在这儿只是简单总结，你可以通过阅读第 7 章第 5 节了解更多的细节。

- 为了在区分大小写的模式下比较大小写不区分的字符串，可以通过大小写区分的 Collation 来对数值定序：

```
SELECT
MIN(str_col COLLATE latin1_general_cs) AS min,
MAX(str_col COLLATE latin1-general_cs) AS max
FROM tbl;
```

- 为了在不区分大小写的模式下比较大小写区分的字符串，可以通过大小写不区分的 Collation 对数值定序：

```
SELECT
MIN(str_col COLLATE latin1_swedish_ci) AS min,
MAX(str_col COLLATE latin1_swedish_ci) AS max
FROM tbl;
```

另一种可能是比较全部转换为同样的字母的数值，这会让字母不相关联。然而，这也将改变重新获取后得到的数值：

```
SELECT
MIN(UPPER(str_col)) AS min,
MAX(upper(str_col)) AS max
FROM tbl;
```

- 二进制字符串通过使用字节的数字值比较，所以并没有字母的概念被引入。然而，不同形式的字母有不同的字节数值，二进制字符串的有效比较是大小写区分（也就是，a 和 A 是不同的）。如果使用大小写不区分的方式来比较二进制字符串，可以将它们转换为非二进制字符串，同时使用正确的 Collation：

```
SELECT
MIN(CONVERT(str_col USING latin1) COLLATE latin1_swedish_ci) AS min,
MAX(CONVERT(str_col USING latin1) COLLATE latin1_swedish_ci) AS max
FROM tbl;
```

如果默认的 Collation 是大小写不区分的 (latin1 就是这样)，你可以忽略 COLLATE 子句。

8.7 将摘要划分为子群

Dividing a Summary into Subgroups

问题

你可能想要对数据行集合的每个子群计算摘要，而不是计算一个整体的摘要值。

解决方案

使用一条 GROUP BY 子句将数据行划分为各个群。

讨论

到目前为止展示过的值摘要语句都是针对结果集的所有数据行计算摘要数值。例如，接下来的语句确定了 mail 表中的记录的数目，同时还包括已经发送的邮件信息的总数目：

```
mysql> SELECT COUNT(*) FROM mail;
+-----+
| COUNT(*) |
+-----+
|      16 |
+-----+
```

有时，需要将一个数据行集合分散成子群并且针对每个子群生成摘要。可以通过使用连接让 GROUP BY 子句与聚类函数共同工作来实现。为了确定每个发送者发送的信息数目，按照发送者名字划分群，计算每个名字出现了多少次，然后显示姓名和相应的计数次数：

```
mysql> SELECT srcuser, COUNT(*) FROM mail
-> GROUP BY srcuser;
+-----+-----+
| srcuser | COUNT(*) |
+-----+-----+
| barb   |      3 |
| gene   |      6 |
| phil   |      5 |
| tricia |      2 |
+-----+-----+
```

查询对用来划分群的同类数据列 (srcuser) 生成摘要，但是那并非总是必需的。假设你想要一个 mail 表的快速的描述，用来显示其中列举的每一个发送者的总通信流量（字节大小）和每条信息的平均字节数。在这种情形下，你仍然可以使用 srcuser 数据列来将数据列分群，但是摘要函数对 size 的数值进行操作：

```
mysql> SELECT srcuser,
-> SUM(size) AS 'total bytes',
-> AVG(size) AS 'bytes per message'
-> FROM mail GROUP BY srcuser;
+-----+-----+
```

```
+-----+-----+-----+
| srcuser | total bytes | bytes per message |
+-----+-----+-----+
| barb   |      156696 |      52232.0000 |
| gene   |     1033108 |     172184.6667 |
| phil   |      18974 |      3794.8000 |
| tricia |     2589407 |     1294703.5000 |
+-----+-----+-----+
```

使用划分子群需要的数据列数目来实现你所要求的完美摘要。先前的查询显示的每个发送者发送的信息数是个粗略的数字。为了更准确的找出每个发送者从每个主机发送了多少信息，应该使用两个数据列来划分子群。这将产生一个具有嵌套子群的结果（子群中还有子群）：

```
mysql> SELECT srcuser, srchost, COUNT(srcuser) FROM mail
      -> GROUP BY srcuser, srchost;
+-----+-----+-----+
| srcuser | srchost | COUNT(srcuser) |
+-----+-----+-----+
| barb   | saturn   |          2 |
| barb   | venus    |          1 |
| gene   | mars     |          2 |
| gene   | saturn   |          2 |
| gene   | venus    |          2 |
| phil   | mars     |          3 |
| phil   | venus    |          2 |
| tricia | mars     |          1 |
| tricia | saturn   |          1 |
+-----+-----+-----+
```

本节前面的例子使用 COUNT()、SUM() 和 AVG() 函数来为每个子群生成摘要。你也可以使用 MIN() 或者 MAX() 函数。通过 GROUP BY 子句，它们可以告诉你每个子群中最小或者最大的数值。接下来的查询通过消息发送者来对 mail 表中的数据行进行子群划分，显示每个子群中发送的最大的消息的大小和最近的消息的日期：

```
mysql> SELECT srcuser, MAX(size), MAX(t) FROM mail GROUP BY srcuser;
+-----+-----+-----+
| srcuser | MAX(size) | MAX(t)   |
+-----+-----+-----+
| barb   |    98151 | 2006-05-14 14:42:21 |
| gene   |   998532 | 2006-05-19 22:21:51 |
| phil   |   10294  | 2006-05-17 12:49:23 |
| tricia | 2394482 | 2006-05-14 17:03:01 |
+-----+-----+-----+
```

你可以使用很多数据列划分子群同时显示数据列中数值的每一个合并中的最大值。下面的查询将查找 mail 表中列举的每一对发送者和接收者之间发送的最大的信息：

```
mysql> SELECT srcuser, dstuser, MAX(size) FROM mail GROUP BY srcuser, dstuser;
+-----+-----+-----+
| srcuser | dstuser | MAX(size) |
+-----+-----+-----+
```

```

+-----+-----+-----+
| barb | barb | 98151 |
| barb | tricia | 58274 |
| gene | barb | 2291 |
| gene | gene | 23992 |
| gene | tricia | 998532 |
| phil | barb | 10294 |
| phil | phil | 1048 |
| phil | tricia | 5781 |
| tricia | gene | 194925 |
| tricia | phil | 2394482 |
+-----+-----+-----+

```

当使用聚类函数来生成预划分群的摘要数值时，注意下面的问题，它引入了选择与划分子群的数据列无关的无摘要表数据列。假设你想知道 driver_log 表中每个司机最长的旅程。可以通过这样的查询得到：

```

mysql> SELECT name, MAX(miles) AS 'longest trip'
-> FROM driver_log GROUP BY name;
+-----+-----+
| name | longest trip |
+-----+-----+
| Ben | 152 |
| Henry | 300 |
| Suzi | 502 |
+-----+-----+

```

但是如果你也想显示每个司机最长的旅程发生的日期，那么你能通过在输出数据列中增加 trav_date 来实现吗？对不起，这样是无法满足要求的：

```

mysql> SELECT name, trav_date, MAX(miles) AS 'longest trip'
-> FROM driver_log GROUP BY name;
+-----+-----+-----+
| name | trav_date | longest trip |
+-----+-----+-----+
| Ben | 2006-08-30 | 152 |
| Henry | 2006-08-29 | 300 |
| Suzi | 2006-08-29 | 502 |
+-----+-----+-----+

```

这个查询确实会得到一个结果，但是如果你把它与整个表（如下所示）相比较，你会发现尽管 Ben 和 Henry 的日期是正确，但是 Suzi 的日期是错误的：

```

+-----+-----+-----+-----+
| rec_id | name | trav_date | miles | ←Ben 的最远旅程
+-----+-----+-----+-----+
| 1 | Ben | 2006-08-30 | 152 |
| 2 | Suzi | 2006-08-29 | 391 |
| 3 | Henry | 2006-08-29 | 300 | ←Henry 的最远旅程
| 4 | Henry | 2006-08-27 | 96 |
| 5 | Ben | 2006-08-29 | 131 |
| 6 | Henry | 2006-08-26 | 115 |
| 7 | Suzi | 2006-09-02 | 502 | ←Suzi 的最远旅程
| 8 | Henry | 2006-09-01 | 197 |
+-----+-----+-----+-----+

```

```
|      9 | Ben    | 2006-09-02 |    79 |
|     10 | Henry  | 2006-08-30 |   203 |
+-----+-----+-----+-----+
```

那该如何处理？为什么摘要语句得到的是不正确的结果？之所以会产生这样的结果是因为当你在查询中使用 GROUP BY 子句的时候，你唯一能选择的子句就是划分子群的子句或者从子群中计算得到的摘要数值。如果你显示表中额外的数据列，它们将不会被用来划分数据列同时它们被显示的数值也是不确定的。（对刚刚显示的语句，MySQL 可能对每一个司机简单选取了第一个日期，而不考虑它是否是匹配司机最大的行程数值的日期。）

这个显示与最小或最大子群数值相关的问题的通常解决办法是引入连接。这个技术将在 12 章中介绍。对于当前的问题，所要求的结果可以这样得到：

```
mysql> CREATE TABLE t
-> SELECT name, MAX(miles) AS miles FROM driver_log GROUP BY name;
mysql> SELECT d.name, d.trav_date, d.miles AS 'longest trip'
-> FROM driver_log AS d INNER JOIN t USING (name, miles) ORDER BY name;
+-----+-----+-----+
| name | trav_date | longest trip |
+-----+-----+-----+
| Ben  | 2006-08-30 |      152 |
| Henry | 2006-08-29 |      300 |
| Suzi | 2006-09-02 |      502 |
+-----+-----+-----+
```

8.8 摘要与空值

Summaries and NULL Values

问题

你正在生成摘要的数值集合可能包括 NULL 值，你需要知道如何解释结果。

解决方案

明白聚类函数是如何处理 NULL 值的。

讨论

大部分的聚类函数忽略 NULL 值。假设你有一张记录实验结果的 expt 表，对其中的每一个项目给定四次测试并且对没有执行的测试给出 NULL 值：

```
mysql> SELECT subject, test, score FROM expt ORDER BY subject, test;
+-----+-----+-----+
| subject | test | score |
+-----+-----+-----+
| Jane   | A    |    47 |
+-----+-----+-----+
```

```

| Jane    | B      | 50 |
| Jane    | C      | NULL |
| Jane   | D      | NULL |
| Marvin | A      | 52 |
| Marvin | B      | 45 |
| Marvin | C      | 53 |
| Marvin | D      | NULL |
+-----+-----+-----+

```

通过使用 GROUP BY 子句来安排数据行，分类标准是项目名称。每个项目测试的数目、总分、平均分、最低分和最高分可以这样计算得到：

```

mysql> SELECT subject,
-> COUNT(score) AS n,
-> SUM(score) AS total,
-> AVG(score) AS average,
-> MIN(score) AS lowest,
-> MAX(score) AS highest
-> FROM expt GROUP BY subject;
+-----+-----+-----+-----+-----+
| subject | n | total | average | lowest | highest |
+-----+-----+-----+-----+-----+
| Jane    | 2 | 97   | 48.5000 | 47    | 50    |
| Marvin  | 3 | 150  | 50.0000 | 45    | 53    |
+-----+-----+-----+-----+-----+

```

你可以从结果中被标记为 n（测试数目）的数据列中看出，查询计数的值仅有 5，尽管整个表格包含有 8 次。为什么？因为该计数的数据列为相应的每个项目测试结果不为空的得分的数目。其他项目显示的结果也都仅仅是从非空得分中计算得到的。

聚类函数会忽略 NULL 值已经很明显。如果它们遵循通常的 SQL 算法规则，将 NULL 值与其他任何值相加会得到 NULL 值。这会让聚类函数变得真的很难被使用，因为当你每次执行摘要操作时不得不自己过滤 NULL 值以避免得到 NULL 值。通过忽略 NULL 值，聚类函数变得方便了许多。

然而，需要注意的是尽管聚类函数可能忽略 NULL 值，它们中的某些仍然会得到 NULL 值。这种情况发生在当没有任何东西可以被用来生成摘要时，也就是如果数据集为空或仅仅包含 NULL 值。接下来的查询与前一个相同，仅有一个小小的改变。它仅仅选择了 NULL 的测试成绩，所以对聚类函数而言没有任何可以操作的东西：

```

mysql> SELECT subject,
-> COUNT(score) AS n,
-> SUM(score) AS total,
-> AVG(score) AS average,
-> MIN(score) AS lowest,
-> MAX(score) AS highest
-> FROM expt WHERE score IS NULL GROUP BY subject;
+-----+-----+-----+-----+-----+

```

```

+-----+-----+-----+-----+-----+
| subject | n | total | average | lowest | highest |
+-----+-----+-----+-----+-----+
| Jane    | 0 | NULL  | NULL   | NULL   | NULL   |
| Marvin  | 0 | NULL  | NULL   | NULL   | NULL   |
+-----+-----+-----+-----+-----+

```

对 COUNT() 函数，每个项目的得分数目是 0 同时也是那样报告的。另一方面，当没有数值可以用来生成摘要时，SUM() 函数、AVG() 函数、MIN() 函数和 MAX() 函数都将返回空值。如果你不想这些函数在查询结果输出中产生 NULL 值，可以使用 IFNULL() 函数来正确映射它们的数值：

```

mysql> SELECT subject,
-> COUNT(score) AS n,
-> IFNULL(SUM(score),0) AS total,
-> IFNULL(AVG(score),0) AS average,
-> IFNULL(MIN(score),'Unknown') AS lowest,
-> IFNULL(MAX(score),'Unknown') AS highest
-> FROM expt WHERE score IS NULL GROUP BY subject;
+-----+-----+-----+-----+-----+
| subject | n | total | average | lowest | highest |
+-----+-----+-----+-----+-----+
| Jane    | 0 | 0    | 0.0000 | Unknown | Unknown |
| Marvin  | 0 | 0    | 0.0000 | Unknown | Unknown |
+-----+-----+-----+-----+-----+

```

COUNT() 函数对待空值与其他聚类函数略有不同。像其他聚类函数一样，COUNT(expr) 仅仅计算非 NULL 值，但是 COUNT(*) 在计算数据行时，是不会考虑其内容的。你可以通过下面的例子看出 COUNT() 函数的两种使用形式之间的差别：

```

mysql> SELECT COUNT(*), COUNT(score) FROM expt;
+-----+-----+
| COUNT(*) | COUNT(score) |
+-----+-----+
|     8    |      5   |
+-----+-----+

```

这告诉我们 expt 表中有 8 个数据行，但是仅有 5 个被填充了 score。COUNT() 函数的不同形式对于计数遗漏的数值非常有用。让我们来看看其中的不同：

```

mysql> SELECT COUNT(*) - COUNT(score) AS missing FROM expt;
+-----+
| missing |
+-----+
|      3  |
+-----+

```

遗漏和没遗漏的计数也能够通过子群来确定。接下来的查询对每一个项目都提供了一种简单的方法来评估实验完成的程度：

```

mysql> SELECT subject,
-> COUNT(*) AS total,
-> COUNT(score) AS 'nonmissing',

```

```

-> COUNT(*) - COUNT(score) AS missing
-> FROM expt GROUP BY subject;
+-----+-----+-----+-----+
| subject | total | nonmissing | missing |
+-----+-----+-----+-----+
| Jane   |    4 |          2 |      2 |
| Marvin |    4 |          3 |      1 |
+-----+-----+-----+-----+

```

8.9 使用确定的特性选择组群

Selecting Only Groups Matching Specific Conditions

问题

你希望计算各组的摘要，但是仅仅显示那些匹配确定标准的组群的结果。

解决方案

使用 HAVING 子句。

讨论

你可能对 WHERE 子句的使用比较熟悉，它用于指明每个单独数据行在选择语句中必须满足的条件。所以，很自然地就可以使用 WHERE 来书写包括摘要值的条件。唯一麻烦就是它实际上不能工作。如果你想确定 driver_log 中行驶超过 3 天的司机，你一开始可能打算这样书写查询语句：

```

mysql> SELECT COUNT(*), name
-> FROM dirver_log
-> WHERE COUNT(*) > 3
-> GROUP BY name;
ERROR 1111(HY000): INVALID use of group function

```

问题是 WHERE 指明的初始约束条件确定了哪些数据行应该被选择，但是 COUNT() 函数的数值仅仅只能在数据行被选中之后才确定。解决方法是将 COUNT() 表达式放置到 HAVING 子句中。HAVING 子句与 WHERE 类似，但是它通常是被应用到群组特性而非单个数据行之中。也就是说，HAVING 是操作在已经选定和划分好子群的数据行集上，是基于对聚类函数的结果的额外约束条件的应用，而聚类函数结果在初始选择过程中并不知道。因此前面的查询应该这样重写：

```

mysql> SELECT COUNT(*), name
-> FROM driver_log
-> GROUP BY name
-> HAVING COUNT(*) > 3;
+-----+-----+
| COUNT(*) | name  |
+-----+-----+

```

```
+-----+  
|      5 | Henry |  
+-----+
```

当你使用 HAVING 时，你仍然可以包含 WHERE 子句，但是仅仅只能选择数据行，而非测试摘要数值。

HAVING 可以使用别名，所以前面的查询可以像这样重写：

```
mysql> SELECT COUNT(*) AS count, name  
-> FROM driver_log  
-> GROUP BY name  
-> HAVING count > 3;  
+-----+-----+  
| count | name |  
+-----+-----+  
|      5 | Henry |  
+-----+-----+
```

8.10 使用计数确定数值是否唯一

Using Counts to Determine Whether Values Are Unique

问题

你想要知道表中的数值是否唯一。

解决方案

联合使用 HAVING 和 COUNT() 函数。

讨论

DISTINCT 消除了重复数据但是不能表明原始数据中的哪些数值实际上真的重复了。你可以使用 HAVING 在没有应用 DISTINCT 函数的条件下来查找唯一数值。HAVING 函数可以告诉你哪个数值是唯一或者不唯一的。

接下来的语句表明了哪一天仅有一个司机当班，哪一天不只一个司机上班。它们是使用 HAVING 和 COUNT() 来确定哪一个 trav_date 数值是唯一或者不唯一的：

```
mysql> SELECT trav_date, COUNT(trav_date)  
-> FROM driver_log  
-> GROUP BY trav_date  
-> HAVING COUNT(trav_date) = 1;  
+-----+-----+  
| trav_date | COUNT(trav_date) |  
+-----+-----+  
| 2006-08-26 |           1 |  
| 2006-08-27 |           1 |  
| 2006-09-01 |           1 |  
+-----+-----+  
mysql> SELECT trav_date, COUNT(trav_date)  
-> FROM driver_log  
-> GROUP BY trav_date
```

```

-> HAVING COUNT(trav_date) > 1;
+-----+-----+
| trav_date | COUNT(trav_date) |
+-----+-----+
| 2006-08-29 |            3 |
| 2006-08-30 |            2 |
| 2006-09-02 |            2 |
+-----+-----+

```

这项技术也能应用于联合数值中。例如，为了查找仅仅发送了一个信息的信息发送/接收对，可以在 mail 表中查找仅仅发生了一次的联合：

```

mysql> SELECT srcuser, dstuser
-> FROM mail
-> GROUP BY srcuser, dstuser
-> HAVING COUNT(*) = 1;
+-----+-----+
| srcuser | dstuser |
+-----+-----+
| barb   | barb    |
| gene   | tricia  |
| phil   | barb    |
| tricia | gene    |
| tricia | phil    |
+-----+-----+

```

注意这个查询并不给出计数结果。例如，一开始的两个例子给出了计数，表明计数函数被正常使用，但是你也可以在 HAVING 子句中引用聚类数值而不是输出数据列中使用它。

8.11 使用表达式结果分组

Grouping by expression results

问题

你可能想通过一个表达式计算得到的结果将数据行划分为若干子群。

解决方案

将表达式放到 GROUP BY 子句中。

讨论

像 ORDER BY 子句一样，GROUP BY 子句能够引用表达式。这意味着你可以使用计算作为分组的基础。例如，为了查找州名称长度的分布，使用这些长度作为组群划分的特征：

```

mysql> SELECT CHAR_LENGTH(name), COUNT(*)
-> FROM states GROUP BY CHAR_LENGTH(name);
+-----+-----+
| CHAR_LENGTH(name) | COUNT(*) |
+-----+-----+

```

```
+-----+-----+
|      | 4 |      | 3 |
|      | 5 |      | 3 |
|      | 6 |      | 5 |
|      | 7 |      | 8 |
|      | 8 |      | 12 |
|      | 9 |      | 4 |
|      | 10 |      | 4 |
|      | 11 |      | 2 |
|      | 12 |      | 4 |
|      | 13 |      | 3 |
|      | 14 |      | 2 |
+-----+-----+
```

就像使用 ORDER BY 子句一样，你能够在 GROUP BY 子句中直接书写分组表达式，或者对表达式（如果它出现在输出数据列中）使用别名，然后在 GROUP BY 中使用别名。

你还可以使用多个表达式分组。为了查找某年多个州加入合众国的日期，可以使用建州的月和日期分类数据，然后使用 HAVING 和 COUNT() 函数来查找非唯一的联合数据：

```
mysql> SELECT
-> MONTHNAME(statehood) AS month,
-> DAYOFMONTH(statehood) AS day,
-> COUNT(*) AS count
-> FROM states GROUP BY month, day HAVING count > 1;
+-----+-----+-----+
| month | day | count |
+-----+-----+-----+
| February | 14 | 2 |
| June | 1 | 2 |
| March | 1 | 2 |
| May | 29 | 2 |
| November | 2 | 2 |
+-----+-----+-----+
```

8.12 分类无类别数据

Categorizing Noncategorical Data

问题

你需要对没有自然类别的数据集合生成摘要。

解决方案

使用表达式来将这些数值分组。

讨论

本章 11 节显示了如何通过表达式的结果来对数据行进行划分。这样做的一个重要应用就是为没有特定类别的数值提供分类。这非常有用，因为 GROUP BY 最适合处理具有重复数

值的数据列。例如，你可能试图在 state 表中使用 POP 数据列分类数据行以实现人口分析。就像我们看见的一样，它不会工作得很好，因为在数据列中有太多的独立数值。实际上，它们是完全独立的，就像下面的查询表示的一样：

```
mysql> SELECT COUNT(pop), COUNT(DISTINCT pop) FROM states;
+-----+-----+
| COUNT(pop) | COUNT(DISTINCT pop) |
+-----+-----+
|      50 |          50 |
+-----+-----+
```

在这样的情形中，数值无法很好地分类成类别数目相对较少的集合。你可以使用一个转换来强制它们分类。例如：通过确定人口数值范围来完成这项工作：

```
mysql> SELECT MIN(pop), MAX(pop) FROM states;
+-----+-----+
| MIN(pop) | MAX(pop) |
+-----+-----+
| 506529 | 35893799 |
+-----+-----+
```

通过这个结果你可以看出，如果你将人口划分为 5 个百万量级，它们将划分为 6 个类别——一个合理的数字（分类范围是 1 到 5 000 000、5 000 001 到 10 000 000 等等）。为了将每个人口数值放置到正确的类别，划分为 5 个百万量级，同时使用整数结果：

```
mysql> SELECT FLOOR(pop/5000000) AS 'max population (millions)',
    -> COUNT(*) AS 'number of states'
    -> FROM states GROUP BY 'max population (millions)';
+-----+-----+
| max population (millions) | number of states |
+-----+-----+
| 0 | 29 |
| 1 | 13 |
| 2 | 4 |
| 3 | 2 |
| 4 | 1 |
| 7 | 1 |
+-----+-----+
```

然而这不是非常正确。表达式将人口数值分类到一个小数目的类别集合中很好，但是并不能正确的报告类别的数值。让我们尝试将 FLOOR() 函数的结果乘以 5：

```
mysql> SELECT FLOOR(pop/5000000)*5 AS 'max population (millions)',
    -> COUNT(*) AS 'number of states'
    -> FROM states GROUP BY 'max population (millions)';
+-----+-----+
| max population (millions) | number of states |
+-----+-----+
| 0 | 29 |
+-----+-----+
```

	5	13
	10	4
	15	2
	20	1
	35	1

这仍然不正确。人口最多的州的人口数为 35 893 799，应该被分类到 40 000 000 的类别中，而非 35 000 000 中。问题在于生成类别的表达式将划分群的数值按照每个群的下界设定。为了让分类子群的数值按照群组的上界分类，可以使用如下的方法。对于大小为 n 的类别，你可以使用如下的表达式将数值 x 放置到正确的类别中：

```
FLOOR( (x+(n-1))/n)
```

所以我们的查询最后看起来应该是这样的：

```
mysql> SELECT FLOOR((pop+4999999)/5000000)*5 AS 'max population (millions)',  
-> COUNT(*) AS 'number of states'  
-> FROM states GROUP BY 'max population (millions)';  
+-----+-----+  
| max population (millions) | number of states |  
+-----+-----+  
| 5 | 29 |  
| 10 | 13 |  
| 15 | 4 |  
| 20 | 2 |  
| 25 | 1 |  
| 40 | 1 |  
+-----+-----+
```

结果很清楚的表明美国多数的州拥有 5 000 000 或者更少的人口。

这个技术对所有数字类型的数值都有效。例如，你可以按照如下方式将 mail 表中的数据行按照 100 000 字节的量级分类：

```
mysql> SELECT FLOOR((size+99999)/100000) AS 'size (100KB)',  
-> COUNT(*) AS 'number of messages'  
-> FROM mail GROUP BY 'size (100KB)';  
+-----+-----+  
| size (100KB) | number of messages |  
+-----+-----+  
| 1 | 13 |  
| 2 | 1 |  
| 10 | 1 |  
| 24 | 1 |  
+-----+-----+
```

在一些情况下，按照对数标尺来分类群组可能更正确。例如，各个州的人口能够按照如下方式处理：

```
mysql> SELECT FLOOR(LOG10(pop)) AS 'log10(population)',  
-> COUNT(*) AS 'number of states'  
-> FROM states GROUP BY 'log10(population)';
```

log10(population)	number of states
5	7
6	35
7	8

这个查询结果是分别拥有十万量级、百万量级和千万量级人口的州的数目。

数值集合中重复的数值有多少？

为了估计目前的数值集合中有多少重复，可以使用 COUNT(DISTINCT) 和 COUNT() 的比值。如果所有的数值都是唯一的，那么两个计数方式的结果应该相等，则其比值应该为 1。这就是 mail 表中 t 数据和 state 表中 pop 数据的情况：

```
mysql> SELECT COUNT(DISTINCT t) / COUNT(t) FROM mail;
+-----+
| COUNT(DISTINCT t) / COUNT(t) |
+-----+
|           1.0000 |
+-----+
mysql> SELECT COUNT(DISTINCT pop) / COUNT(pop) FROM states;
+-----+
| COUNT(DISTINCT pop) / COUNT(pop) |
+-----+
|           1.0000 |
+-----+
```

对于重复更多的数值集，COUNT(DISTINCT) 将比 COUNT() 的数值小，所以比值将更小：

```
mysql> SELECT COUNT(DISTINCT name) / COUNT(name) FROM driver_log;
+-----+
| COUNT(DISTINCT name) / COUNT(name) |
+-----+
|           0.3000 |
+-----+
```

该比值的实际用处是什么？接近 0 的结果指明了极高的重复度，这意味着可以很自然的将群组划分为数目较小的类别。当结果为 1 或者接近 1 则说明具有很多唯一的数值，必然的结果就是 GROUP BY 子句不会很有效的将数值分类（也就是会有很多类，这与数值的数目相关。）这是告诉你如何去生成摘要，你将会发现使用人工的手段来分类数据是很必要的，本节的技术就是为此而准备的。

8.13 控制摘要显示顺序

Controlling Summary Display Order

问题

你想排序摘要语句运行得到的结果。

解决方案

使用 ORDER BY 子句——如果 GROUP BY 不能生成想要的排序顺序。

讨论

在 MySQL 中，GROUP BY 不是仅仅只划分群组，它也有排序的能力。因此，在摘要语句中常常是没有必要使用 ORDER BY 子句的。但是，如果你需要的排序顺序与 GROUP BY 默认的排序顺序不同，你仍然可以使用 ORDER BY 子句来实现你的排序要求。例如，为了为了确定 driver_log 表中每个人的行驶日期数和总行驶英里数，可以使用这样的语句：

```
mysql> SELECT name, COUNT(*) AS days, SUM(miles) AS mileage
      -> FROM driver_log GROUP BY name;
+-----+-----+-----+
| name | days | mileage |
+-----+-----+-----+
| Ben  |    3 |     362 |
| Henry |    5 |     911 |
| Suzi |    2 |     893 |
+-----+-----+-----+
```

但是这是按照姓名排序。如果你想根据行驶的里程数或天数来对司机进行排序，可以增加正确的 ORDER BY 子句：

```
mysql> SELECT name, COUNT(*) AS days, SUM(miles) AS mileage
      -> FROM driver_log GROUP BY name ORDER BY days DESC;
+-----+-----+-----+
| name | days | mileage |
+-----+-----+-----+
| Henry |    5 |     911 |
| Ben  |    3 |     362 |
| Suzi |    2 |     893 |
+-----+-----+-----+
mysql> SELECT name, COUNT(*) AS days, SUM(miles) AS mileage
      -> FROM driver_log GROUP BY name ORDER BY mileage DESC;
+-----+-----+-----+
| name | days | mileage |
+-----+-----+-----+
| Henry |    5 |     911 |
| Suzi |    2 |     893 |
| Ben  |    3 |     362 |
+-----+-----+-----+
```

这些语句中的 ORDER BY 子句通过使用别名引入了一个总计值。在 MySQL5.0 或更新的版本中，不是必须这样做，你能在 ORDER BY 子句中直接使用总计值。在 MySQL5.0 之前的版本，你必须对它们建立别名然后在 ORDER BY 子句中使用别名。

有时你能通过选择正确的 GROUP BY 表达式来重排序摘要而不是用 ORDER BY 子句。例如，如果你想计数一周的每一天有多少个州加入了合众国，按照周日划分群组，那么按照词汇顺序排序的结果如下：

```
mysql> SELECT DAYNAME(statehood), COUNT(*) FROM states
-> GROUP BY DAYNAME(statehood);
+-----+-----+
| DAYNAME(statehood) | COUNT(*) |
+-----+-----+
| Friday           |     8 |
| Monday            |     9 |
| Saturday          |    11 |
| Thursday          |     5 |
| Tuesday           |     6 |
| Wednesday         |    11 |
+-----+-----+
```

从结果中你可以看出没有州在星期日加入合众国，但是只有在你认真观察了结果之后才能发现这个显然的事实。如果结果按照周日的顺序排序，那么结果会更加容易理解。通过增加一个 ORDER BY 子句对周日按照数值排序可以完成这个任务，但是也有其他不使用 ORDER BY 来达到同样效果的方法。可以通过使用 DAYOFWEEK() 函数来代替 DAYNAME() 函数：

```
mysql> SELECT DAYNAME(statehood), COUNT(*)
-> FROM states GROUP BY DAYOFWEEK(statehood);
+-----+-----+
| DAYNAME(statehood) | COUNT(*) |
+-----+-----+
| Monday            |     9 |
| Tuesday           |     6 |
| Wednesday         |    11 |
| Thursday          |     5 |
| Friday            |     8 |
| Saturday           |    11 |
+-----+-----+
```

GROUP BY 隐含的排序能力会在查询过程中增加管理。如果你不关心输出数据行是否被排序好了，可以增加一个 ORDER BY NULL 子句来禁止对它的管理：

```
mysql> SELECT name, COUNT(*) AS days, SUM(miles) AS mileage
-> FROM driver_log GROUP BY name;
+-----+-----+-----+
| name  | days | mileage |
+-----+-----+-----+
| Ben   |   3 |    362 |
| Henry |   5 |    911 |
| Suzi  |   2 |    893 |
+-----+-----+-----+
```

```

mysql> SELECT name, COUNT(*) AS days, SUM(miles) AS mileage
-> FROM driver_log GROUP BY name ORDER BY NULL;
+-----+-----+-----+
| name | days | mileage |
+-----+-----+-----+
| Ben  |    3 |    362 |
| Suzi |    2 |    893 |
| Henry |    5 |    911 |
+-----+-----+-----+

```

通过 GROUP BY 完成排序是 MySQL 的扩展。为 MySQL 书写的语句比为其他数据库系统书写的语句具有更好的版本兼容性，你可能会发现在所有情况下增加一个额外的 ORDER BY 子句会更有好处。

8.14 查找最小或最大的摘要数值

Finding Smallest or Largest Summary Values

问题

你想要计算每个群组的摘要数值，但是仅仅显示其中最小或最大的数值。

解决方案

在语句中增加 LIMIT 子句。

讨论

MIN() 函数和 MAX() 函数能够查找到数值范围的端点数值，但是如果你想知道摘要数值集合的两端数值，那这些函数就无法正常工作了。MIN() 函数和 MAX() 函数的参数不能作为其他聚类函数的参数。例如，你能很容易查找到每个司机的总行驶英里数：

```

mysql> SELECT name, SUM(miles)
-> FROM driver_log
-> GROUP BY name;
+-----+-----+
| name | SUM(miles) |
+-----+-----+
| Ben  |      362 |
| Henry |      911 |
| Suzi |      893 |
+-----+-----+

```

但是如果你仅仅想选择行驶里程最长的司机所在的数据行，那么这样并不能正常工作：

```

mysql> SELECT name, SUM(miles)
-> FROM driver_log
-> GROUP BY name
-> HAVING SUM(miles) = MAX(SUM(miles));
ERROR 1111 (HY000): Invalid use of group function

```

然而可以首先按照最大数值的顺序对数据行排序，然后使用 LIMIT 选择第一个数据行：

```
mysql> SELECT name, SUM(miles) AS 'total miles'  
-> FROM driver_log  
-> GROUP BY name  
-> ORDER BY 'total miles' DESC LIMIT 1;  
+-----+  
| name | total miles |  
+-----+  
| Henry | 911 |  
+-----+
```

注意这里如果有不止一个数据行具有给定的摘要值，一个 LIMIT 1 查询是无法告知你的。例如，你可能试图像这样确认各州的名称中最常用的初始字母：

```
mysql> SELECT LEFT(name,1) AS letter, COUNT(*) AS count FROM states  
-> GROUP BY letter ORDER BY count DESC LIMIT 1;  
+-----+  
| letter | count |  
+-----+  
| M | 8 |  
+-----+
```

但是同样有 8 个州的名字是以 N 开头的。如果你需要知道所有最常见的数值特别是当它不止一个时，那么首先查找最大的计数，然后选择匹配该最大数值的所有数值：

```
mysql> SET @max = (SELECT COUNT(*) FROM states  
-> GROUP BY LEFT(name,1) ORDER BY COUNT(*) DESC LIMIT 1);  
mysql> SELECT LEFT(name,1) AS letter, COUNT(*) AS count FROM states  
-> GROUP BY letter HAVING count = @max;  
+-----+  
| letter | count |  
+-----+  
| M | 8 |  
| N | 8 |  
+-----+
```

可以将二者合二为一，将最大计数的计算放到子查询中，同时将它们集成到一个语句里去：

```
mysql> SELECT LEFT(name,1) AS letter, COUNT(*) AS count FROM states  
-> GROUP BY letter HAVING count =  
-> (SELECT COUNT(*) FROM states  
-> GROUP BY LEFT(name,1) ORDER BY COUNT(*) DESC LIMIT 1);  
+-----+  
| letter | count |  
+-----+  
| M | 8 |  
| N | 8 |  
+-----+
```

8.15 基于日期的摘要

Date-Based Summaries

问题

你希望基于日期或时间值来生成摘要。

解决方案

使用 GROUP BY 在正确的时机将临时数值放置到正确的类别里。通常这包括使用表达式提取日期或时间中的有效部分。

讨论

为了按照时间顺序放置数据行，可以使用 ORDER BY 子句排序含有临时类型数据的数据列。如果你希望基于时间片分类来生成数据行的摘要，你需要确定如何将每个数据行分类到正确的时间片中，同时使用 GROUP BY 将它们划分到正确的群组中。

例如，为了确认有多少司机在路上同时每天行驶了多少英里，可以在 driver_log 表中通过日期聚类数据行：

```
mysql> SELECT trav_date,
    -> COUNT(*) AS 'number of drivers', SUM(miles) AS 'miles logged'
    -> FROM driver_log GROUP BY trav_date;
+-----+-----+-----+
| trav_date | number of drivers | miles logged |
+-----+-----+-----+
| 2006-08-26 | 1 | 115
| 2006-08-27 | 1 | 96
| 2006-08-29 | 3 | 822
| 2006-08-30 | 2 | 355
| 2006-09-01 | 1 | 197
| 2006-09-02 | 2 | 581
+-----+-----+-----+
```

然而，如果你在表中增加更多的数据行，那么摘要会变得更长。从某些角度来说，不同日期的数目变得太大将导致摘要无法使用，所以你最好将分类的标准换为周或月。

当一个临时数据列包含如此多的不同数值时，它很难被很好的分类。典型的对分类数据行生成摘要的方法是使用表达式将日期或时间数值相关部分映射到一个较小的类别集中。例如，为了给 mail 表中的数据行生成一个当天时间的摘要，可以这样做（注 1）：

```
mysql> SELECT HOUR(t) AS hour,
    -> COUNT(*) AS 'number of messages',
```

注 1：注意结果仅包含在数据中有记录的那些小时。如果要生成一个包含所有小时的统计，需要使用 join 来填补“缺失的”值。参考第 12.8 节。

```

-> SUM(size) AS 'number of bytes sent'
-> FROM mail
-> GROUP BY hour;
+-----+-----+-----+
| hour | number of messages | number of bytes sent |
+-----+-----+-----+
| 7 | 1 | 3824 |
| 8 | 1 | 978 |
| 9 | 2 | 2904 |
| 10 | 2 | 1056806 |
| 11 | 1 | 5781 |
| 12 | 2 | 195798 |
| 13 | 1 | 271 |
| 14 | 1 | 98151 |
| 15 | 1 | 1048 |
| 17 | 2 | 2398338 |
| 22 | 1 | 23992 |
| 23 | 1 | 10294 |
+-----+-----+-----+

```

为了使用当周的摘要来代替之前的摘要，可以使用 DAYOFWEEK() 函数：

```

mysql> SELECT DAYOFWEEK(t) AS weekday,
-> COUNT(*) AS 'number of messages',
-> SUM(size) AS 'number of bytes sent'
-> FROM mail
-> GROUP BY weekday;
+-----+-----+-----+
| weekday | number of messages | number of bytes sent |
+-----+-----+-----+
| 1 | 1 | 271 |
| 2 | 4 | 2500705 |
| 3 | 4 | 1007190 |
| 4 | 2 | 10907 |
| 5 | 1 | 873 |
| 6 | 1 | 58274 |
| 7 | 3 | 219965 |
+-----+-----+-----+

```

为了使输出更有意义，你可能想使用 DAYNAME() 函数来显示每天的名称。然而，因为星期中每天的名称是按照词汇顺序排序的（例如，“星期二”排序在“星期五”之后），使用 DAYNAME() 函数仅仅只能实现显示的目的。继续使用数字日期数值分类可以让输出行按照那样排序：

```

mysql> SELECT DAYNAME(t) AS weekday,
-> COUNT(*) AS 'number of messages',
-> SUM(size) AS 'number of bytes sent'
-> FROM mail
-> GROUP BY DAYOFWEEK(t);
+-----+-----+-----+
| weekday | number of messages | number of bytes sent |
+-----+-----+-----+
| Sunday | 1 | 271 |
| Monday | 4 | 2500705 |
| Tuesday | 4 | 1007190 |
| Wednesday | 2 | 10907 |
+-----+-----+-----+

```

Thursday	1		873
Friday	1		58274
Saturday	3		219965

类似的技术可以用于生成当年的摘要分类，使用数字值排序但按照月的名称来显示。

临时分类有很多种：

- DATETIME 或者 TIMESTAMP 数据列具有包含许多唯一数值的能力。为了生成每天的摘要，将日期部分的时间去掉以将给定日期内发生的所有数值变成统一一个数值。下面的任意一个 GROUP BY 语句都可以完成这个工作，尽管最后一个可能是最慢的：

```
GROUP BY DATE(col_name)
GROUP BY FROM_DAYS(TO_DAYS(col_name))
GROUP BY YEAR(col_name), MONTH(col_name), DAYOFMONTH(col_name)
GROUP BY DATE_FORMAT(col_name, '%Y-%m-%e')
```

- 为了生成每月或者一个季度的销售报告，可以使用 MONTH(col_name) 或 QUARTER(col_name) 划分子群以将日期放置到正确的年度位置中。
- 为了给 Web 服务器的行为生成摘要，可以将你的服务器日志存储到 MySQL 中然后运行相关命令语句，以将数据行归类到不同的时间分类中。19 章第 14 节展示了如何在 Apache 系统中完成相关设置。

8.16 同时使用每一组的摘要和全体的摘要

Working with Per-Group and Overall Summary Values Simultaneously

问题

你想要生成一个需要不同级别的摘要细节的报告，或者你想将每一组的摘要数值与全体的摘要数值相比较。

解决方案

使用能够获得不同级别的两条语句，或者使用能够获取摘要的子查询同时在引用其他查询的更外层查询中引用它。如果它仅仅在显示多个摘要数值时才需要，WITH ROLLUP 函数可能已足够。

讨论

有时一个报告包括不同层次的摘要信息。例如，下面的报告显示 driver_log 表中每个司机的总行驶里程数，和每个司机的行驶里程占整个表中的总里程的百分比：

```
+-----+-----+-----+
| name | miles/driver | percent of total miles |
+-----+-----+-----+
| Ben  |          362 |           16.7128 |
| Henry |          911 |           42.0591 |
| Suzi |          893 |           41.2281 |
+-----+-----+-----+
```

百分比代表每个司机的里程与所有司机的总里程的比值。为了完成百分比的计算，你需要每个组群的摘要来获取每个司机的英里数，同时也需要一个整体的摘要来获取总英里数。首先运行一个查询来获取整体的总英里数：

```
mysql> SELECT @total := SUM(miles) AS 'total miles' FROM driver_log;
+-----+
| total miles |
+-----+
|      2166 |
+-----+
```

然后计算每个组群的数值，同时使用整体数值计算百分比：

```
mysql> SELECT name,
    -> SUM(miles) AS 'miles/driver',
    -> (SUM(miles)*100)/@total AS 'percent of total miles'
    -> FROM driver_log GROUP BY name;
+-----+-----+-----+
| name | miles/driver | percent of total miles |
+-----+-----+-----+
| Ben  |          362 |           16.7128 |
| Henry |          911 |           42.0591 |
| Suzi |          893 |           41.2281 |
+-----+-----+-----+
```

为了将两条语句合成为一句，使用子查询来计算整体英里数：

```
mysql> SELECT name,
    -> SUM(miles) AS 'miles/driver',
    -> (SUM(miles)*100)/(SELECT SUM(miles) FROM driver_log)
    -> AS 'percent of total miles'
    -> FROM driver_log GROUP BY name;
+-----+-----+-----+
| name | miles/driver | percent of total miles |
+-----+-----+-----+
| Ben  |          362 |           16.7128 |
| Henry |          911 |           42.0591 |
| Suzi |          893 |           41.2281 |
+-----+-----+-----+
```

另外一种类型的问题是，当你想将每个子群的摘要与相应整体的摘要对比时就需要使用不同层次的摘要。假设你想确定哪个司机每天的平均行驶里程数低于整体平均值。首先在子查询中计算整体平均值，然后使用 HAVING 子句将每个司机的平均行驶里程与整体的平均值相比较：

```

mysql> SELECT name, AVG(miles) AS driver_avg FROM driver_log
   -> GROUP BY name
   -> HAVING driver_avg < (SELECT AVG(miles) FROM driver_log);
+-----+-----+
| name | driver_avg |
+-----+-----+
| Ben  |    120.6667 |
| Henry |    182.2000 |
+-----+-----+

```

如果你只是想显示不同的摘要数值（并不想执行一个层次的摘要与另一个层次的摘要相对比的计算），可以通过在 GROUP BY 子句中增加 WITH ROLLUP 实现：

```

mysql> SELECT name, SUM(miles) AS 'miles/driver'
   -> FROM driver_log GROUP BY name WITH ROLLUP;
+-----+-----+
| name | miles/driver |
+-----+-----+
| Ben  |      362 |
| Henry |      911 |
| Suzi |      893 |
| NULL |     2166 |
+-----+-----+
mysql> SELECT name, AVG(miles) AS driver_avg FROM driver_log
   -> GROUP BY name WITH ROLLUP;
+-----+-----+
| name | driver_avg |
+-----+-----+
| Ben  |    120.6667 |
| Henry |    182.2000 |
| Suzi |    446.5000 |
| NULL |    216.6000 |
+-----+-----+

```

在上面的每一种情形中，输出 name 数据列中的 NULL 数据列代表计算得到的所有司机的整体总里程数或平均里程数。

如果你采用不只一个数据列划分群组，WITH ROLLUP 能给出多层次的摘要。接下来的语句表明每对用户之间发送信件信息的数目：

```

mysql> SELECT srcuser, dstuser, COUNT(*)
   -> FROM mail GROUP BY srcuser, dstuser;
+-----+-----+-----+
| srcuser | dstuser | COUNT(*) |
+-----+-----+-----+
| barb   | barb    |      1 |
| barb   | tricia  |      2 |
| gene   | barb    |      2 |
| gene   | gene    |      3 |
| gene   | tricia  |      1 |
| phil   | barb    |      1 |
| phil   | phil    |      2 |
| phil   | tricia  |      2 |
| tricia | gene    |      1 |
+-----+-----+-----+

```

```
| tricia | phil      |      1 |
```

增加 WITH ROLLUP 的原因是输出中包括一个对每个 srcuser 数值的中介计数(在 dstuser 数据列中具有 NULL 数值的行), 在最后结束时增加一个全体计数:

```
mysql> SELECT srcuser, dstuser, COUNT(*)
-> FROM mail GROUP BY srcuser, dstuser WITH ROLLUP;
+-----+-----+-----+
| srcuser | dstuser | COUNT(*) |
+-----+-----+-----+
| barb   | barb    |      1 |
| barb   | tricia  |      2 |
| barb   | NULL    |      3 |
| gene   | barb    |      2 |
| gene   | gene    |      3 |
| gene   | tricia  |      1 |
| gene   | NULL    |      6 |
| phil   | barb    |      1 |
| phil   | phil    |      2 |
| phil   | tricia  |      2 |
| phil   | NULL    |      5 |
| tricia | gene    |      1 |
| tricia | phil    |      1 |
| tricia | NULL    |      2 |
| NULL   | NULL    |     16 |
+-----+-----+-----+
```

8.17 生成包括摘要和列表的报告

Generating a Report That Includes a Summary and a List

问题

你想要创建一个报告, 在显示摘要的同时显示与每个摘要数值相关的数据行列表。

解决方案

使用两个语句获取不同层次的摘要信息。或者使用一种编程语言来完成工作的一部分, 这样你可以只使用一条语句。

讨论

假设你想生成一个看来起来象这样的报告:

```
Name: Ben; days on road: 3; miles driven: 362
date: 2006-08-29, trip length: 131
date: 2006-08-30, trip length: 152
date: 2006-09-02, trip length: 79
Name: Henry; days on road: 5; miles driven: 911
date: 2006-08-26, trip length: 115
date: 2006-08-27, trip length: 96
```

```
date: 2006-08-29, trip length: 300
date: 2006-08-30, trip length: 203
date: 2006-09-01, trip length: 197
Name: Suzi; days on road: 2; miles driven: 893
date: 2006-08-29, trip length: 391
date: 2006-09-02, trip length: 502
```

报告表明，对 driver_log 表中的每个司机都有以下信息：

- 一个摘要行显示了司机的姓名、行驶日期数以及行驶里程数。
- 从摘要中计算得到的每个单独形成的日期和里程列表。

这个场景是在上一节中讨论的“不同层次的摘要信息”问题的一个变化。可能一开始看起来不像，因为其中一个信息类型是列表而非摘要。但是那实际上只是一个“第 0 层”摘要。这种类型的问题有很多出现的形式：

- 你有一个列举你的政党中所有候选人的捐款的数据库。政党主席要求将结果打印出来以显示每个候选人的捐款数和总捐款数，包括每个捐款者的姓名和住址。
- 你想制作一个公司介绍的手册，包括每个销售区域的总销售额摘要以及显示每个销售区之下的各个州的销售情况。

这样的问题可以通过使用一类方法来解决：

- 运行分离的语句以获取你所需要的每个层次的信息。（一个单独的查询不可能同时生成所有群组的摘要和每个群组的各个数据行的列表。）
- 取出组成列表的数据行，同时你自己执行摘要计算，以去除摘要语句。

让我们使用每个方法来获取本节开头展示的司机报表。接下来的实现（在 Python 中）通过使用一个查询来对每个司机的日期和行程生成摘要，同时使用另外一个对每个司机提取每次形成的数据行：

```
# 选择每个司机的总行程数并一个词典
# 将每个司机的姓名映射到行驶天数和行驶里程数
name_map = {}
cursor = conn.cursor()
cursor.execute ('''
    SELECT name, COUNT(name), SUM(miles)
    FROM driver_log GROUP BY name
    ''')
for name, days, miles in cursor.fetchall():
    name_map[name] = (days,miles)
```

```

# 选择每个司机的行程数并打印报告，显示行程列表中每个司机的摘要汇总项
cursor.execute (" " "
    SELECT name, trav_date, miles
    FROM driver_log GROUP BY name, trav_date
    " " ")

cur_name = ""
for (name, trav_date, miles) in cursor.fetchall():
    if cur_name != name: #新的司机; 打印司机信息
        print "Name: %s; days on road: %d; miles driven: %d" \
            %(name, name_map[name][0], name_map[name][1])
        cur_name = name
    print " date:%s, trip length: %d " % (trav_date, miles)
cursor.close()

```

在这个程序中改变实现执行了摘要操作。这样做，你能减少所需查询的数目。如果你自己在行程列表中反复执行且计算每个司机的行驶天数和总行程数，一句查询已经足够：

```

#取得司机的里程列表
cursor = conn.cursor ()
cursor.execute (" " "
    SELECT name, trav_date, miles FROM driver_log
    ORDER BY name, trav_date
    " " ")

# 将数据行放置到数据结构中因为我们必须迭代执行它们多次
rows = cursor.fetchall ()
cursor.close ()

# 迭代执行数据行一次以创建字典将每个司机姓名映射到行驶天数和总行程数
# (字典以条目的形式列出而非元组的形式是因为我们需要可变的数值以在循环中被改变)
name_map = {}
for (name, trav_date, miles) in rows:
    if not name_map.has_key (name): # initialize entry if nonexistent
        name_map[name] = [0,0]
    name_map[name][0] = name_map[name][0] + 1      # count days
    name_map[name][1] = name_map[name][1] + miles # sum miles

# 再次迭代执行数据行以打印报告，显示行程列表中每个司机的摘要汇总项
cur_name = ""
for (name, trav_date, miles) in rows:
    if cur_name != name: #新的司机; 打印司机信息
        print "Name: %s; days on road: %d; miles driven: %d" \
            %(name, name_map[name][0], name_map[name][1])
        cur_name = name
    print " date:%s, trip length: %d " % (trav_date, miles)

```

如果你要求获取更多层次的摘要信息，那么这种类型的问题变得更加困难。例如，你可能想要这样一份报告，能够显示司机的摘要和行程的日志，并加上所有司机的总行程数的信息：

```
Total miles driven by all drivers combined: 2166  
Name: Ben; days on road: 3; miles driven: 362  
date: 2006-08-29, trip length: 131  
date: 2006-08-30, trip length: 152  
date: 2006-09-02, trip length: 79  
Name: Henry; days on road: 5; miles driven: 911  
date: 2006-08-26, trip length: 115  
date: 2006-08-27, trip length: 96  
date: 2006-08-29, trip length: 300  
date: 2006-08-30, trip length: 203  
date: 2006-09-01, trip length: 197  
Name: Suzi; days on road: 2; miles driven: 893  
date: 2006-08-29, trip length: 391  
date: 2006-09-02, trip length: 502
```

在这种情形下，你要么需要其他的查询语句来生成总行程数，要么在你的程序中计算全体行程数。

获取和使用元数据

Obtaining and Using Metadata

9.0 引言

Introduction

目前为止，使用的大多数 SQL 语句都是被用于与数据库协同工作的，毕竟这是数据库设计中保留的部分。但是，有时你所需要的不仅仅是数据值。你需要刻画或者描述这些数据的信息——那就是元数据语句。元数据信息经常用于处理结果集的关系，但也能用于你的其他与 MySQL 交互的问题。本节描述了如何获取和使用如下类型的元数据：

关于语句结果的信息

对于删除或者更新数据行的语句，你能确定有多少数据行被更新。对 SELECT 语句，你能在结果集中找出数据列的数目，也就是结果集中的每个数据列的信息，例如数据列的名称和它的显示宽度。这些信息通常对于处理结果是非常重要的。例如，如果你在格式化一个表格的显示，你能确定每个数据列的宽度以及数据的左对齐或右对齐是否正确。

关于表和数据库的信息

关于表和数据库的结构的信息对于需要列举出数据库或者服务器上数据库主机的表的列表非常有用（例如，为了给出允许用户从一组可选项中选择一项的显示）。你也能使用这些来确定表或者数据库的存在。对于表元数据的另一个用处是确定 ENUM 或者 SET 数据列的合法数值。

关于 MySQL 服务器的信息

一些 API 提供了关于数据库服务器或你当前与服务器连接状态的信息。清楚服务器的版本以便确定它是否支持给定的能帮助你创建合适的应用程序的特征。关于连接的信息包括当前用户和默认数据库的信息。

总之，元数据信息被用于数据库系统的实现，所以它倾向于数据库部分相关。这意味着如果应用程序使用了本节介绍的技术且你要将它应用到别的数据库，那么你需要对它做些修改。例如，在 MySQL 中列举表和数据库可以通过使用 SHOW 语句来实现。然而，SHOW 是一个 MySQL 专用的扩展，所以如果你使用了类似 Perl 或 Ruby DBI、PEAR DB、DB-API 或者允许你以数据库无关的方式发送语句的 JDBC，而 SQL 本身是数据库特定的，需要修改之后才能与其他引擎协同工作。

一个更方便的元数据的来源是 INFORMATION_SCHEMA 数据库，它在 MySQL5.0 中可用。这个元数据数据库包括的信息有：数据库、表、列、字符集等。INFORMATION_SCHEMA 具有某些比 SHOW 更好的特征：

- 其他数据库系统也支持 INFORMATION_SCHEMA，所以使用它的应用程序可能比那些使用 SHOW 语句的更方便。
- INFORMATION_SCHEMA 是使用标准的 SELECT 语法，所以与 SHOW 相比，它与其他数据获取操作更类似。

因为这些优势，本章尽可能倾向于使用 INFORMATION_SCHEMA。

INFORMATION_SCHEMA 的一个劣势就是访问它的语句相当于 SHOW 语句更冗长。当你编写程序使用时这并没有关系，但是对于交互式使用，SHOW 语句可能更有吸引力因为它需要的输入更少。同时，如果你还没升级到 MySQL5.0 或者更高的版本，SHOW 是你唯一的选择。

本章范例所使用的代码可以在相关章节的元数据目录下找到。（它们中的某些使用的工具函数在库目录中）为了创建任意你所需要尝试范例的表，使用表目录下的描述。

就像已经指明的一样，本章倾向于使用 INFORMATION_SCHEMA 而非 SHOW。如果你使用的 MySQL 版本比 5.0 更老，那么第 1 版的《MySQL Cookbook》可能更有帮助。那里使用 SHOW 语句的优势是因为在那个 MySQL 的版本中 INFORMATION_SCHEMA 还不存在。简洁地，提供与给定 INFORMATION_SCHEMA 表内容类似信息的 SHOW 语句如下表显示：

INFORMATION_SCHEMA 表	SHOW 语句
SHEMATA	SHOW DATABASES
TABLES	SHOW TABLES
COLUMNS	SHOW COLUMNS

9.1 获取受语句影响的数据行数目

Obtaining the Number of Rows Affected by a Statement

问题

你想知道有多少数据行被一条 SQL 语句改变。

解决方案

有时，发起语句的函数会返回行数。有时是在发起语句之后由你调用一个单独的函数来返回行数。

讨论

对于影响数据行的语句（UPDATE、DELETE、INSERT、REPLACE），每个 API 提供一个方法来确定所包含的数据行的数目。对于 MySQL 而言，“被影响”默认的含义是“被改变”而非“不匹配”。也就是说，没有被语句所改变的数据行是不计数的，尽管它们符合语句中指明的条件。例如，下面的 UPDATE 语句得到一个零的数值“影响”，因为它没有改变任何数据列当前的数值，而不管 WHERE 子句有多少数据行匹配：

```
UPDATE limbs SET arms = 0 WHERE arms = 0;
```

当连接指明它希望数据行匹配计数而非数据行改变计数时，MySQL 服务器允许客户端设置标志位。在这种情形下，前面语句的数据行计数将与 arms 值为 0 的数据行数目相等，即使这个语句没有改变表中的任何东西。然而，不是所有的 MySQL 的 API 都可以得到这个标志。下面的讨论指明哪个 API 可以让你选择你想要的计数类型，哪个是默认使用数据行匹配计数而非数据行改变计数。

Perl

在 Perl DBI 中，改变数据行的语句的数据行计数可以通过 do() 返回：

```
my $count = $dbh->do($stmt);
# 当有错误发生的时候报告 0 数据行
printf "Number of rows affected: %d\n", (defined ($count) ? $count : 0);
```

如果你首先准备了一条语句，然后执行它，execute() 返回行数：

```
my $sth = $dbh->prepare ($stmt);
my $count = $sth->execute ();
printf "Number of rows affected: %d\n", (defined ($count) ? $count : 0);
```

当你连接到 MySQL 服务器时，你可以告知 MySQL 是否通过指定 connect() 调用的数据源名称参数部分的 mysql_client_found_rows 选项来返回数据行改变计数或者数据行匹

配计数。选项设为 0 时数据行改变计数，设为 1 时数据行匹配计数。如下所示：

```
myh %conn_attrs = (PrintError => 0, RaiseError => 1, AutoCommit => 1);
my $dsn = "DBI:mysql:cookbook:localhost;mysql_client_found_rows=1";
my $dbh = DBI->connect ($dsn, "cbuser", "dbpass", \%conn_attrs);
```

`mysql_client_found_rows` 在连接期间改变报告数据行的行为。

尽管 MySQL 本身默认的方式是返回数据行改变的计数，但是 MySQL 上的 Perl DBI 近来的版本却是自动计数数据行匹配数，除非你指定其他方式。对于依靠特定行为的应用程序，最好是在 DSN 中明确的将 `mysql_client_found_row` 选项设置为正确的值。

Ruby

对于改变数据行的语句，Ruby DBI 对于 `do` 方法返回与 Perl DBI 类似的数据行计数。`do` 自身返回计数：

```
ocunt = dbh.do(stmt)
puts "Number of rows affected: #{count}"
```

如果你使用 `execute` 来执行一条语句，`execute` 不会返回数据行计数。但可以在执行语句之后使用语句处理数据行的方法来获取计数：

```
sth = dbh.execute(stmt)
puts "Number of rows affected: #{sth.rows}"
```

MySQL 的 Ruby DBI 驱动默认是返回数据行改变计数，但是该驱动同样支持 `mysql_client_found_rows` 选项，能够让你控制服务器返回数据行改变计数还是数据行匹配计数。它的用途类似于 Perl DBI。例如，要求返回数据行匹配计数，可以这样做：

```
dsn = "DBI:MySQL:database=cookbook;host=localhost;mysql_client_found_rows=1"
dbh = DBI.connect(dsn, "cbuser", "dbpass")
```

PHP

在 PHP 中，调用连接对象的 `affectedRows()` 方法可以查找出一条语句改变了多少数据行：

```
$result = & $conn->query ($stmt);
# 如果语句执行失败将返回 0 行数据行
$count = (PEAR::ISERROR ($result) ? 0 : $conn->affectedRows());
print ("Number of rows affected: $count\n");
```

Python

Python 的 DB-API 将数据行改变计数作为语句指针的 `rowcount` 特征的数值：

```
cursor = conn.cursor ()
cursor.execute (stmt)
print 'Number of rows affected: %d' % cursor.rowcount
```

为了得到数据行匹配计数，引入 MySQLdb 客户端常量同时在 connect 方法的 client_flag 参数中传递 FOUND_ROWS 标志：

```
import MySQLdb.constants.CLIENT

conn = MySQLdb.connect (db = "cookbook",
                       host = "localhost",
                       user = "cbuser",
                       passwd = "cbpass",
                       client_flag = MySQLdb.constants.CLIENT.FOUND_ROWS)
```

Java

对于改变数据行的语句，MySQL 连接子/J JDBC 驱动提供数据行匹配计数而不是数据行改变计数。这是由 JDBC 规范确定的。

Java 的 JDBC 接口有两种不同的方法提供数据行计数，具体由你所执行语句的调用方法来确定。如果你使用 executeUpdate() 函数，数据行计数是它本身返回的数值：

```
Statement s = conn.createStatement ();
int count = s.executeUpdate (stmt);
s.close ();
System.out.println ("Number of rows affected: " + count);
```

如果你使用 execute() 函数，那么它将返回 true 或 false 来指明调用语句是否返回了结果集。对于类似 UPDATE 或 DELETE 等不返回结果集的语句，execute() 函数将返回 false，同时数据行计数将通过调用 getUpdateCount() 方法来得到：

```
Statement s = conn.createStatement ();
if (!s.execute (stmt))
{
    // 没有结果集，打印数据行计数
    System.out.println ("Number of rows affected: " + s.getUpdateCount ());
}
s.close ();
```

9.2 获取设置元数据的结果

Obtaining Result Set Metadata

问题

你已经知道如何得到结果集的数据行（第 2 章第 4 节）。现在你想知道关于结果集的信息，诸如数据列的名称、数据类型，或者有多少数据行和数据列等。

解决方案

正确使用你的 API 提供的功能。

讨论

对于类似 SELECT 能产生结果集的语句，你能够得到元数据类型的数目。本节讨论每个 API 提供的信息，通过使用程序来展示在给出范例语句 (SELECT name、foods FROM profile) 之后如何显示可用的结果集元数据。对这类信息，一个最简单的方法是通过几个例子程序来举例说明：当你从结果集中得到数据行的数值同时你想在一个循环中处理它们时，数据列计数作为循环迭代的上界存储在元数据中。

Perl

使用 Perl DBI 接口，你有两种方法能获取结果集合。这些结果集范围的不同将你的语句中的元数据设为可用状态：

使用语句句柄来处理语句

在这种情形中，你调用 `prepare()` 函数来得到语句句柄。句柄有一个 `execute()` 方法，你可以调用它来生成结果集，同时你可以在循环中取出数据行。通过这种方法，当结果集是活动状态时访问元数据也成为可能——在调用 `execute()` 之后且在到达结果集末尾之前。当数据行提取方法发现没有数据行时，它将隐式调用 `finish()` 函数，让元数据不可用（如果你自己显示地调用 `finish()` 函数也会得到同样结果）。因此通常情况下，调用 `execute()` 函数之后最好立即访问元数据，复制你希望在提取循环之后还需要的所有数值。

使用在单个操作中返回结果集合的数据库句柄方法来处理语句

通过这种方法，处理语句过程中任意的元数据都将通过其返回的时间处理，尽管你仍然可以从结果集的大小确定数据行和数据列的数目。

当你使用语句句柄方法来处理语句时，在你调用句柄的 `execute()` 方法之后 DBI 将结果集的元数据设置为可使用状态。引用数组的形式中的这个信息从根本上是可用的。对每一个这样类型的元数据，结果集中的数组每一列都有一个元素。数组引用是作为语句句柄的特征访问的。例如，`$sth->{NAME}` 指向数据列名称数组，每一个列的名称都被作为数组的元素：

```
$name = $sth->{NAME}->[$i];
```

或者你可以像这样访问整个数组：

```
@names = @{$sth->{NAME}};
```

下面的表格列举了特征名称，通过它你可以访问基于数组的元数据和每个数组中数值的含义。以大写字母开头的名称是标准 DBI 特征，对大多数数据库引擎都可用。以 mysql_ 开头的特征名称是 MySQL 专用的，不可移植。他们提供的这些种类的信息在其他数据库系统中可能也可用，但是其特征名称不同。

属性名	数组元素含义
NAME	数据列名称
NAME_lc	小写形式的数据列名称
NAME_uc	大写形式的数据列名称
NULLABLE	0 或空字符串 = 数据列数值不能为空值 1 = 数据列数值可以为空值 2 = 未知情况
PRECISION	数据列宽度
SCALE	小数位数（对数字型数据列）
TYPE	数据类型（数字型 DBI 代码）
mysql_is_blob	如果数据列为 BLOB（或 TEXT）类型，其数值为真
mysql_is_key	如果数据列是关键字的一部分，其数值为真
mysql_is_num	如果数据列具有数字类型，其数值为真
mysql_is_pri_key	如果数据列是主关键字的一部份，其数值为真
mysql_max_length	结果集中的数据列数值的实际最大长度
mysql_table	数据列所在表的名称
mysql-type	数据类型（数字的 MySQL 内部代码）
mysql_type_name	数据类型名称

在下面的表格中列举的一些元数据类型在作为引用复述访问时比作为数组更好。这些复述在每个数据列数值具有一个元素。这个元素关键字是数据列名称同时其数值是结果集中的数据列位置。例如：

```
$col_pos = $sth-> {NAME_hash}->{col_name};
```

属性名	哈希表元素含义
NAME_hash	数据列名称
NAME_hash_lc	小写形式的数据列名称
NAME_hash_uc	大写形式的数据列名称

结果集中的数据列数目和编报表中的占位符数目是作为标量使用的：

```
$num_cols = $sth->{NUM_OF_FIELDS};  
$num_placeholders = $sth->{NUM_OF_PARAMS};
```

下面是一些示例代码，展示了如何执行语句和现实结果集元数据：

```
my $stmt = "SELECT name, foods FROM profile";  
printf "Statement: %s\n", $stmt;  
my $sth = $dbh->prepare ($stmt);  
$sth->execute();  
# 元数据信息在这一处开始可用.....  
printf "NUM_OF_FIELDS: %d\n", $sth->{NUM_OF_FIELDS};  
print "Note: statement has no result set\n" if $sth->{NUM_OF_FIELDS} == 0;  
for my $i (0 .. $sth->{NUM_OF_FIELDS}-1)  
{  
    printf "--- Column %d (%s) ---\n", $i, $sth->{NAME}->[$i];  
    printf "NAME_lc:           %s\n", $sth->{NAME_lc}->[$i];  
    printf "NAME_uc:           %s\n", $sth->{NAME_uc}->[$i];  
    printf "NULLABLE:          %s\n", $sth->{NULLABLE}->[$i];  
    printf "PRECISION:         %s\n", $sth->{PRECISION}->[$i];  
    printf "SCALE:              %s\n", $sth->{SCALE}->[$i];  
    printf "TYPE:               %s\n", $sth->{TYPE}->[$i];  
    printf "mysql_is_blob:     %s\n", $sth->{mysql_is_blob}->[$i];  
    printf "mysql_is_key:      %s\n", $sth->{mysql_is_key}->[$i];  
    printf "mysql_is_num:      %s\n", $sth->{mysql_is_num}->[$i];  
    printf "mysql_is_pri_key:  %s\n", $sth->{mysql_is_pri_key}->[$i];  
    printf "mysql_max_length:  %s\n", $sth->{mysql_max_length}->[$i];  
    printf "mysql_table:        %s\n", $sth->{mysql_table}->[$i];  
    printf "mysql_type:         %s\n", $sth->{mysql_type}->[$i];  
    printf "mysql_type_name:   %s\n", $sth->{mysql_type_name}->[$i];  
}  
$sth->finish (); #因为我们不能抽取它的数据行，所以释放结果集
```

如果你使用上面的代码来执行如下语句：SELECT name、foods FROM profiles，输出结果如下：

```
Statement: SELECT name, foods FROM profile  
NUM_OF_FIELDS: 2  
--- Column 0 (name) ---  
NAME_lc:           name  
NAME_uc:           NAME  
NULLABLE:  
PRECISION:         20  
SCALE:             0  
TYPE:              1  
mysql_is_blob:  
mysql_is_key:  
mysql_is_num:      0  
mysql_is_pri_key:  
mysql_max_length:  7  
mysql_table:        profile  
mysql_type:         254  
mysql_type_name:   char  
--- Column 1(foods)  
NAME_lc:           foods  
NAME_uc:           FOODS
```

```

NULLABLE:          1
PRECISION:        42
SCALE:            0
TYPE:             1
mysql_is_blob:
mysql_is_key:
mysql_is_num:     0
mysql_is_pri_key:
mysql_max_length: 21
mysql_table:      profile
mysql_type:       254
mysql_type_name:  char

```

为了从通过调用 `execute()` 函数生成的结果集中得到数据行计数，你必须自己抽取数据行并对它们进行计数。在 DBI 文档中，使用 `$sth->row()` 函数在 `SELECT` 语句中获取计数是被明确反对的。

你也能通过调用使用数据库句柄而非语句句柄的 DBI 方法来获得结果集，例如 `selectall_arrayref()` 函数或 `selectall_hashref()` 函数。这些方法不能访问数据列元数据。这些信息已经被方法所返回的时间所暴露，对于你的脚本来说并不可用。然而，你可以通过检查结果集本身来获取数据列和数据行计数。你完成它的方法依赖于方法本身生成的数据结构的类型。这些数据结构和你使用它们获取的结果集的数据行和数据列计数在第 2 章第 4 节讨论过。

Ruby

Ruby DBI 在你使用 `execute` 执行语句之后提供结果集元数据，并且在你调用语句句柄 `finish` 方法之前都可以访问元数据。`column_names` 方法返回数据列名称的数组（如果没有结果集那么该数组为空）。如果有数据集，`column_info` 返回 `ColumnInfo` 对象的数组，每个数据列对应其中一个元素。`ColumnInfo` 对象与 hash 类似都拥有下表中展示的元素。以下划线开头的元素名称是 MySQL 特定的，对于其他数据库可能不能实现。（这些元素中的大多数都不能实现，除非你使用 Ruby DBI0.1.1 或者更高版本。）

成员	成员含义
<code>name</code>	数据列名称
<code>sql_type</code>	XOPEN 类型的数目
<code>type_name</code>	XOPEN 类型的名称
<code>precision</code>	数据列宽度
<code>scale</code>	小数位数（对数字类型的数据列）
<code>nullable</code>	如果数据列允许空值，其值为真
<code>indexed</code>	如果数据列被索引，其值为真
<code>primary</code>	如果数据列是主关键字的一部分，其值为真

成员	成员含义
unique	如果数据列是唯一索引的部分，其值为真
mysql_type	数据类型（数字的 MySQL 内部代码）
mysql_type_name	数据类型名称
mysql_length	数据列宽度
mysql_max_length	结果集中数据列数值的实际最大长度
mysql_flags	数据类型标志

下面的示例代码展示了如何执行语句并显示数据列的结果集的元数据数值：

```

stmt = "SELECT name, foods FROM profile"
puts "Statement: " + stmt
sth = dbh.execute(stmt)
# 元数据信息在这一点开始可用……
puts "Number of columns: #{sth.column_name.size}"
puts "Note: statement has no result set" if sth.column_name.size == 0
sth.column_info.each-with_index do | info, i |
  printf "--- Column %d (%s) ---\n", i, info["name"]
  printf "sql_type: %s\n", info["sql_type"]
  printf "type_name: %s\n", info["type_name"]
  printf "precision: %s\n", info["precision"]
  printf "scale: %s\n", info["scale"]
  printf "nullable: %s\n", info["nullable"]
  printf "indexed: %s\n", info["indexed"]
  printf "primary: %s\n", info["primary"]
  printf "unique: %s\n", info["unique"]
  printf "mysql_type: %s\n", info["mysql_type"]
  printf "mysql_type_name: %s\n", info["mysql_type_name"]
  printf "mysql_length: %s\n", info["mysql_length"]
  printf "mysql_max-length: %s\n", info["mysql_max_length"]
  printf "mysql_flags: %s\n", info["mysql_flags"]
end
sth.finish

```

如果你使用上面的代码执行语句：SELECT name, foods FROM profile，输出结果如下：

```

Statement: SELECT name, foods FROM profile
Number of columns: 2
--- Column 0 (name) ---
sql_type: 12
type_name: VARCHAR
precision: 20
scale: 0
nullable: false
indexed: false
primary: false
unique: false
mysql_type: 254
mysql_type_name: VARCHAR

```

```
mysql_length:          20
mysql_max_length:      7
mysql_flags:           4097
--- Column 1 (foods) ---
sql_type:              12
type_name:             VARCHAR
precision:             42
scale:                 0
nullable:              true
indexed:               false
primary:               false
unique:                false
mysql-type:            254
mysql_type_name:       VARCHAR
mysql_length:          42
mysql_max_length:      21
mysql_flags:            2048
```

为了得到调用 `execute` 生成的结果集的数据行计数，你可以提取数据行并自己对其计数。`sth.rows` 方法不能工作在结果集上。

你也能通过调用使用数据库句柄而非语句句柄的 DBI 方法来获得结果集，例如 `select_one` 函数或 `select_all` 函数。这些方法不能访问数据列元数据。这些信息已经被方法所返回的时间所暴露，对于你的脚本来说并不可用。然而，你可以通过检查结果集本身来获取数据列和数据行计数。

PHP

在 PHP 中，`SELECT` 语句的元数据信息从 PEAR DB 成功调用 `query()` 函数之后变得可用，它将一直保持为可用状态直到你释放结果集为止。

为了检查元数据是否处于可用状态，需要确认查询结果是一个结果集，然后将它传递给连接对象的 `tableInfo()` 方法，这个方法会返回一个包含数据列信息数组的结构。每一个数组元素都包含下面表中展示的成员。

成员名称	成员意义
<code>name</code>	数据列名称
<code>type</code>	数据类型名称
<code>len</code>	数据长度
<code>flags</code>	数据类型标志

PEAR DB 也通过结果集的 `numRows()` 和 `numCols()` 方法将结果集的数据行和数据列的计数设为可用。

接下来的代码显示如何访问和显示结果集的元数据：

```
$stmt = "SELECT name, foods FROM profile";
print ("Statement: $stmt\n");
$result = & $conn->query ($stmt);
if (PEAR::isError ($result))
die ("Statement failed\n");
# 元数据信息从这里开始可用……
if (is_a ($result, "DB_result")) # 语句生成结果集
{
$ nrows = $result-> numRows ();
$ ncols = $result-> numCols ();
$info = & $conn-> tableInfo ($result);
}
else # 语句不生成结果集
{
$nrows = 0;
$ncols = 0;
}
print ("Number of rows: $nrows\n");
print ("Number of columns: $ncols\n");
if ($ncols == 0)
print ("Note: statement has no result set\n");
for ($i = 0; $i < $ncols; $i++)
{
$col_info = $info[$i];
printf ("--- Column %d(%s) ---\n", $i, $col_info["name"]);
printf ("type: %s\n", $col_info["type"]);
printf ("len: %s\n", $col_info["len"]);
printf ("flags: %s\n", $col_info["flags"]);
}
if ($ncols > 0) #如果结果集存在，那么对其进行处理
$result->free();
```

该程序运行的输出如下：

```
statement: SELECT name, foods FROM profile
Number of rows: 10
Number of columns: 2
--- Column 0 (name) ---
type: char
len: 7
flags: not_null
--- Column 1 (foods) ---
type: char
len: 21
flags: set
```

Python

对于语句产生的结果集，Python 的 DB-API 使数据行和数据列计数处于可用状态，与单个数据列的信息相同。

为了得到数据集的数据行计数，可以访问指针的 `rowcount` 特征。数据列计数并非直接可用，不过在调用 `fetchone()` 或者 `fetchall()` 方法之后，你可以确定任意结果集数据行元组的计数。同样也能通过使用 `cursor.description` 而无需提取任何数据行来确定数据

列计数。这是结果集中一个元组，其中每个数据列包含一个元素，因此它的长度能够告诉你结果集中有多少个数据列。(如果语句不能生成结果集，例如 UPDATE 这样的语句，description 的数值为 None。)description 元组的每一个元素都是描述相应结果集的数据列元数据的另一个元组，每一个数据列有 7 个元数据数值。下面的代码显示了如何访问它们以及它们的含义：

```
stmt = "SELECT name, foods FROM profile"
print "Statement: ", stmt
cursor = conn.cursor ()
cursor.execute (stmt)
# 元数据信息从这里开始可用……
print "Number of rows:", cursor.rowcount
if cursor.description == None: # 没有结果集
    ncols = 0
else:
    ncols = len (cursor.description)
print "Number of columns:", ncols
if ncols == 0:
    print "Note: statement has no result set"
for i in range (ncols):
    col_info = cursor.description[i]
    # 打印名称和其他信息
    print "--- Column %d(%s) ---" % (I, col_info[0])
    print "Type:           ", col_info[1]
    print "Display size:   ", col_info[2]
    print "Internal size:  ", col_info[3]
    print "Precision:      ", col_info[4]
    print "Scale:          ", col_info[5]
    print "Nullable:       ", col_info[6]
cursor.close
```

该程序运行的输出如下：

```
Statement: SELECT name, foods FROM rofile
Number of rows: 10
Number of columns: 2
--- Column 0 (name) ---
Type:           254
Display size:   7
Internal size:  20
Precision:      20
Scale:          0
Nullable:       0
--- Column 1 (foods) ---
Type:           254
Display size:   21
Internal size:  42
Precision:      42
Scale:          0
Nullable:       1
```

Java

JDBC 通过 `ResultSetMetaData` 对象使结果集元数据可用，你可以通过调用你的 `ResultSet` 对象的 `getMetaData()` 方法来得到 `ResultSetMetaData` 对象。该元数据对象提供几种不同种类的信息访问。它的 `getColumnCount()` 方法返回结果集中数据列的计数。下面代码中其他元数据的类型，提供关于单个数据列的信息并将数据列的索引作为它们的参数。对 JDBC，数据列的索引从 1 而非 0 开始，这与其他的 API 不同。

```
String stmt = "SELECT name, foods FROM profile";
System.out.println ("Statement: " + stmt);
Statement s = conn.createStatement ();
s.executeQuery (stmt);
ResultSet rs = s.getResultSet ();
ResultSetMetaData md = rs.getMetaData ();
// 元数据信息从这里开始可用……
int ncols = md.getColumnCount ();
System.out.println ("Number of columns: " + ncols);
if (ncols == 0)
    System.out.println ("Note: statement has no result set");
for (int i = 1; i<= ncols; i++) //数据列索引是从1开始的
{
    System.out.println ("--- Column" + i
        + "(" + md.getColumnName (i) + ")---")
    System.out.println ("getColumnDisplaySize: " + md.getColumnDisplaySize
(i));
    System.out.println ("getColumnName: " + md.getColumnName (i));
    System.out.println ("getColumnLabel: " + md.getColumnLabel (i));
    System.out.println ("getColumnType: " + md.getColumnType (i));
    System.out.println ("getColumnTypeName: " + md.getColumnTypeName (i));
    System.out.println ("getPrecision: " + md.getPrecision (i));
    System.out.println ("getScale: " + md.getScale (i));
    System.out.println ("getTableName: " + md.getTableName (i));
    System.out.println ("isAutoIncrement: " + md.isAutoIncrement (i));
    System.out.println ("isNullable: " + md.isNullable (i));
    System.out.println ("isCaseSensitive: " + md.isCaseSensitive (i));
    System.out.println ("isSigned: " + md.isSigned (i));
}
rs.close ();
s.close ();
```

该程序运行的输出如下：

```
Statement: SELECT name, foods FROM profile
Number of columns: 2
--- Column 1 (name) ---
getColumnDisplaySize: 20
getColumnName: name
getColumnLabel: name
getColumnType: 1
getColumnTypeName: CHAR
getPrecision: 20
getScale: 0
getTableName: profile
isAutoIncrement: false
```

```
isSigned: false
--- Column 2 (foods)
getColumnDisplaySize: 42
getColumnName: foods
getColumnType: 1
getColumnTypeName: CHAR
getPrecision: 42
getScale: 0
getTableName: profile
isAutoIncrement: false
isNullable: 1
isCaseSensitive: false
isSigned: false
```

结果集的数据行计数不能直接使用，你必须提取数据行并对它们进行计数。

还有几个其他的 JDBC 结果集元数据调用，但是其中许多都不能对 MySQL 提供有用的信息。如果你想尝试它们，可以查阅 JDBC 参考手册得到它们的信息并通过修改程序来查看它们的返回值是什么。

9.3 确定一条语句是否生成了结果集

Determining Whether a Statement Produced a Result Set

问题

你只是执行一条 SQL 语句，但是你不能确定它是否能生成结果集。

解决方案

检查元数据中的数据列计数。如果该计数值为 0，那么就没有结果集。

讨论

如果你编写了一个应用程序，它能读取文件中或者用户键盘输入的语句字符串，你可能不必知道类似 SELECT 这样的语句产生了结果集或者类似 UPDATE 这样的语句不能产生结果集。这是很重要的区别，因为你处理产生结果集语句的方式与处理不产生结果集语句的方式是不同的。假设没有错误发生，一个告知不同的方法是在执行语句之后检查指明数据列计数的元数据数值（如同本章第 2 节展示的一样）。零值数据列计数说明执行的语句是 INSERT、UPDATE 或者其他不返回结果集的语句。非零值说明结果集存在，同时你可以继续处理并提取数据行。这个技术将 SELECT 语句与非 SELECT 语句甚至是返回空结果集的 SELECT 语句区分开。（空结果集与无结果集是不同的。前者没有返回数据行，但是数据列计数仍然是正确的；而后者根本没有数据列。）

一些 API 提供与检查数据列计数不同的方法来区分语句类型：

- 在 JDBC 中，你可以使用 `execute()` 方法来执行任何语句，它将通过返回 `true` 或 `false` 指明是否存在结果集。
- 在 PHP 中，PEAR DB 程序应该通过 `statement-execution` 方法来检查结果以查看返回值是否为 `DB_result` 对象：

```
$result =& $conn->query ($stmt);
if (PEAR::isError ($result))
    die ("Statement failed\n");
if (is_a ($result, "DB_result"))
{
    # 语句生成结果集
}
else
{
    # 语句不生成结果集
}
```

采用这样的方法来代替检查数据列计数是因为在非 `DB_result` 对象结果上试图调用 `numCols()` 函数会导致一个错误。

- 在 Python 中，对于不能产生结果集的语句，`cursor.description` 的数值为 `None`。

9.4 使用元数据来格式化查询输出

Using Metadata to Format Query Output

问题

你想产生一个很好的格式化显示的结果集。

解决方案

使用结果集元数据来协助你。它能提供关于数据结构的重要信息和结果的内容。

讨论

元数据信息对于格式化查询结果确实是有用的，因为它能告知你关于数据列的几件重要事情（例如名称和显示宽度），在你甚至不知道查询是什么的情况下。例如，你能编写一个通用的函数以表的格式来显示结果而不用知道查询是什么。下面的 Java 代码展示了一个完成这件事的方法。它采用了一个结果集对象来获取结果的元数据。然后使用两个对象依次来获取并格式化结果中的数值。输出与 mysql 生成的结果类似：一行跟随若干结果数

据行的数据列表头，具有很精细的方格和垂直线。下面是一个该函数的显示范例，给出了查询产生的结果集：

```
+-----+-----+-----+
| id   | name    | birth   |
+-----+-----+-----+
| 1    | Fred    | 1970-04-13 |
| 2    | Mort    | 1969-09-30 |
| 3    | Brit    | 1957-12-01 |
| 4    | Carl    | 1973-11-02 |
| 5    | Sean    | 1963-07-04 |
| 6    | Alan    | 1965-02-14 |
| 7    | Mara    | 1968-09-17 |
| 8    | Shepard | 1975-09-02 |
| 9    | Dick    | 1952-08-20 |
| 10   | Tony    | 1960-05-01 |
| 11   | Juan    | NULL     |
+-----+-----+-----+
Number of rows selected: 11
```

类似这样的应用必须解决的一个基本问题是确定每个数据列正确的显示宽度。`getColumnDisplaySize()`方法返回数据列宽度，但是我们实际上需要考虑其他部分的信息：

- 数据列名称的长度需要考虑（可能比数据列宽度更长）。
- 对于 `NULL` 值我们将打印“`NULL`”，所以如果数据列包括 `NULL` 值，显示宽度至少为 4。

下面的 Java 函数 `displayResultSet()` 能在考虑上述因素的前提下格式化一个结果集。它也能通过提取数据行来确定数据行计数，因为 JDBC 不能直接使元数据处于可用状态。

```
public static void displayResultSet (ResultSet rs) throws SQLException
{
    ResultSetMetaData md = rs.getMetaData ();
    int ncols = md.getColumnCount ();
    int nrows = 0;
    int[] width = new int[ncols + 1];      // 存储数据列宽度的数组
    StringBuffer b = new StringBuffer (); // 保持分隔线的缓冲区

    // 计算数据列宽度
    for (int i = 1; i<=ncols; i++)
    {
        // 对于 getColumnDisplaySize()，一些驱动返回 -1;
        // 如果这样，我们将使用数据列名称长度来覆盖它
        width[i] = md.getColumnDisplaySize (i);
        if (width[i] < md.getColumnName (i).length())
            width[i] = md.getColumnName (i).length();
        // isNullable() 的返回值为 1/0，而非 true/false;
        if (width[i] < 4 && md.isNullable (i) !=0)
            width[i] = 4;
    }
```

```

// 构造 +---+---...线
b.append("+");
for (int i = 1; i <= ncols; i++)
{
    for (int j = 0; j < width[i]; j++)
        b.append ("~");
    b.append ("+");
}

// 打印分隔线, 数据列头, 分隔线
System.out.println (b.toString());
System.out.print ("|");
for (int i = 1; i <= ncols; i++)
{
    System.out.print (md.getColumnName (i));
    for (int j = md.getColumnName (i).length (); j < width[i]; j++)
        System.out.print ("");
    System.out.print ("|");
}
System.out.println ();
System.out.println (b.toString());

// 打印数据集的内容
while (rs.next())
{
    ++nrows;
    System.out.print ("|");
    for (int i = 1; i <= ncols; i++)
    {
        String s = rs.getString (i);
        if (rs.wasNull())
            s = "NULL";
        System.out.print (s);
        for (int j = s.length (); j < width[i]; j++)
            System.out.print ("");
        System.out.print ("|");
    }
    System.out.println ();
}
// 打印分隔线和数据行技术
System.out.println (b.toString());
System.out.println ("Number of rows selected: " + nrows);
}

```

如果你想更加认真, 你还能测试数据列是否包含数字型数值。如果包含, 那么可以将其按照右对齐格式化。在 Perl DBI 脚本中, 很容易完成检查, 因为你可以访问元数据的 mysql_is_num 特征。对于其他 API, 除非有等同于“数据列是数字型”的元数据数值可用, 否则检查不是那么容易的事情。如果不是, 你必须查看数据类型指示以查看它是否是几个可能的数字型之一。

`displayResultSet()` 函数的其他缺点是它使用表定义中指明的数据列宽度来打印数据列, 而非结果集中实际表示数值的最大宽度, 后者通常小一点。你可以从前面 `display-`

`ResultSet()`的示例输出中看出这点。`id` 和 `name` 数据列分别是 10 个和 20 个字符宽，尽管它们最长的字符分别仅为 2 个和 7 个字符。在 Perl、Ruby、PHP 和 DB-API 中，你可以得到结果集中表示的数值的最大宽度。为了在 JDBC 中确定这些宽度，你必须自己迭代结果集并检查数据列数值长度。这需要提供可滚动的结果集功能的 JDBC2.0 驱动。如果你有这样的驱动（MySQL Connector/J 就是一个），`displayResultSet()` 函数中的数据列宽度计算能够按照如下方式修改：

```
// 计算数列列宽度
for (int i = 1; i <= ncols; i++)
{
    width[i] = md.getColumnName(i).length();
    // isNullable() 返回 1/0, 而非 true/false
    if (width[i] < 4 && md.isNullable (i) !=0)
        width[i] = 4;
}
// 迭代结果集并按照需要调整显示
while (rs.next ())
{
    for (int i = 1; i <= ncols; i++)
    {
        byte[] bytes = rs.getBytes (i);
        if (!rs.wasNull())
        {
            int len = bytes.length;
            if (width[i] < len)
                width[i] = len;
        }
    }
}
rs.beforeFirst ();    //在显示结果集之前重置它
```

做了这些改动之后，更加简洁的查询结果如下所示：

```
+---+-----+
| id | name   | birth      |
+---+-----+
| 1  | Fred    | 1970-04-13 |
| 2  | Mort    | 1969-09-30 |
| 3  | Brit    | 1957-12-01 |
| 4  | Carl    | 1973-11-02 |
| 5  | Sean    | 1963-07-04 |
| 6  | Alan    | 1965-02-14 |
| 7  | Mara    | 1968-09-17 |
| 8  | Shepard | 1975-09-02 |
| 9  | Dick    | 1952-08-20 |
| 10 | Tony    | 1960-05-01 |
| 11 | Juan    | NULL       |
+---+-----+
Number of rows selected: 11
```

参考

Ruby DBI::Utils::TableFormatter 模块拥有一个 `ascii` 方法，能够生成类似本节描述的格式化显示。可以这样使用它：

```
dbh.execute(stmt) do |sth|
  DBI::Utils::TableFormatter.ascii(sth.column_name, sth.fetch_all)
end
```

9.5 列举或检查数据库或表的扩展

Listing or Checking Existence of Databases or Tables

问题

你想得到一张 MySQL 服务器管理的数据库列表或者数据库中表的列表，或者你想检查一个特定的数据库或者表是否存在。

解决方案

使用 `INFORMATION_SCHEMA` 来得到相应信息。`SCHEMATA` 表中的每一个数据行对应一个数据库，同时 `TABLES` 表中的每一个数据行对应每个数据库中的每张表。

讨论

为了获得服务器管理的数据库列表，可以使用如下语句：

```
SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA;
```

如果你需要排序后的结果，那么增加一条 `ORDER BY SCHEMA_NAME` 子句。

为了检查指定的数据库是否存在，可以使用以给数据库命名为条件的 `WHERE` 子句。如果收回了一个数据行，那么数据库存在。如果没有收回，那么数据库不存在。接下来的 Ruby 方法展示了如何执行数据库是否存在的测试：

```
def database_exists(dbh, db_name)
  return db.select_one("SELECT SCHEMA_NAME
                        FROM INFORMATION_SCHEMA.SCHEMATA
                        WHERE SCHEMA_NAME = ?", db_name) != nil
end
```

为了获取数据库中的表的列表，可以在从 `TABLES` 表中进行选择的语句的 `WHERE` 子句中对数据库命名：

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'cookbook';
```

如果你想得到一个排序的结果，那么可以增加一条 `ORDER BY TABLE_NAME` 子句。

为了获取默认数据库中的表列表，可以使用如下的语句：

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = DATABASE();
```

如果没有数据库被选择，DATABASE()返回 NULL 值，同时没有数据行匹配，这就是正确的结果。

为了检查指定的表是否存在，可以使用以给表命名为条件的 WHERE 子句。目前有一个 Ruby 的方法，能够在给定数据库中执行表是否存在的测试：

```
def table_exists(dbh, db_name, tbl_name)
    return dbh.select_one(
        "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
         WHERE TABLE_SCHEMA = ? AND TABLE_NAME = ?",
        db_name, tbl_name) != nil
end
```



提示：从 INFORMATION_SCHEMA 中获取结果依赖于你的权限。你只能看见那些你拥有某些权限的数据库或表的信息。这意味着如果给定的对象存在但是你无权访问它，那么存在测试将返回 false。

一些 API 提供与数据库无关的方法来获取数据库或表的列表。在 Perl DBI 中，数据库句柄 tables() 方法返回默认数据库中的表列表：

```
@tables = $dbh->tables();
```

Ruby 方法类似：

```
tables = dbh.tables
```

在 Java 中，你可以使用设计为返回数据库或表列表的 JDBC 方法。对每一个方法，调用你的连接对象的 getMetaData() 方法并使用返回的 DatabaseMetaData() 对象来获取你想要的信息。下面是如何产生数据库列表的方法：

```
// 获取数据库列表
DatabaseMetaData md = conn.getMetaData ();
ResultSet RS = MD.getTables ();
while (rs.next ())
    System.out.println (rs.getString (1)); // 数据列 1 = 数据库名称
rs.close ();
```

在给定数据库中得到表的列表的过程类似：

```
// 获取被命名为 dbName 的数据库中表的列表;
// 如果 dbName 为空字符串，那么使用默认的数据库
DatabaseMetaData md = conn.getMetaData ();
ResultSet rs = md.getTables (dbName, "", "%", null);
while (rs.next ())
    System.out.println (rs.getString (3)); // 数据列 3= 表名称
rs.close ();
```

9.6 访问表数据列定义

Accessing Table Column Definitions

问题

你想找出一张表有哪些数据列以及它们是如何定义的。

解决方案

有好几种方法可以解决这个问题。你可以从 INFORMATION_SCHEMA 获取数据列定义，也可以通过 SHOW 语句或者是从 mysqldump 中得到你需要的信息。

讨论

表结构的信息能够让你回答诸如“表里有哪些数据列以及它们的类型是什么？”或“ENUM 或 SET 数据列的合法数值是什么？”在 MySQL 中，有好几种方法来查找关于表的结构：

- 从 INFORMATION_SCHEMA 中获取信息。COLUMNS 表包括数据列定义。
- 使用 SHOW COLUMNS 语句。
- 使用 SHOW CREATE TABLE 语句或者 mysqldump 命令行程序来获取显示表结构的 CREATE TABLE 语句。

接下来的部分讨论了你如何使用这些方法向 MySQL 查询表信息。为了尝试这些范例，创建如下的列举项目 ID、名称以及每个项目可用颜色的 item 表：

```
CREATE TABLE item
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name    CHAR(20),
    colors SET('chartreuse','mauve','lime green','puce') DEFAULT 'puce',
    PRIMARY KEY(id)
);
```

使用 INFORMATION_SCHEMA 来获取表结构

为了通过检查 INFORMATION_SCHEMA 来获取表中关于数据列的信息，可以使用如下形式的语句：

```
mysql> SELET * FROM INFORMATION-SCHEMA.COLUMNS
      -> WHERE TABLE-SCHEMA = 'cookbook' AND TABLE_NAME = 'item'\G
***** 1. row *****
    TABLE_CATALOG: NULL
    TABLE_SCHEMA: cookbook
    TABLE_NAME: item
    COLUMN_NAME: id
    ORDINAL_POSITION: 1
    COLUMN_DEFAULT: NULL
```

```

        IS_NULLABLE: NO
        DATA_TYPE: int
CHARACTER_MAXIMUM_LENGTH: NULL
CHARACTER_OCTET_LENGTH: NULL
        NUMERIC_PRECISION: 10
        NUMERIC_SCALE: 0
CHARACTER_SET_NAME: NULL
        COLLATION_NAME: NULL
        COLUMN_TYPE: int(10) unsigned
        COLUMN_KEY: PRI
        EXTRA: auto_increment
        PRIVILEGES: select,insert,update,references
COLUMN_COMMENT:

***** 2. row *****
        TABLE_CATALOG: NULL
        TABLE_SCHEMA: cookbook
        TABLE_NAME: item
        COLUMN_NAME: name
        ORDINAL_POSITION: 2
        COLUMN_DEFAULT: NULL
        IS_NULLABLE: YES
        DATA_TYPE: char
CHARACTER_MAXIMUM_LENGTH: 20
CHARACTER_OCTET_LENGTH: 20
        NUMERIC_PRECISION: NULL
        NUMERIC_SCALE: NULL
CHARACTER_SET_NAME: latin1
        COLLATION_NAME: latin1_swedish_ci
        COLUMN_TYPE: char(20)
        COLUMN_KEY:
        EXTRA:
        PRIVILEGES: select,insert,update,references
COLUMN_COMMENT:

***** 3. row *****
        TABLE_CATALOG: NULL
        TABLE_SCHEMA: cookbook
        TABLE_NAME: item
        COLUMN_NAME: colors
        ORDINAL_POSITION: 3
        COLUMN_DEFAULT: puce
        IS_NULLABLE: YES
        DATA_TYPE: set
CHARACTER_MAXIMUM_LENGTH: 32
CHARACTER_OCTET_LENGTH: 32
        NUMERIC_PRECISION: NULL
        NUMERIC_SCALE: NULL
CHARACTER_SET_NAME: latin1
        COLLATION_NAME: latin1_swedish_ci
        COLUMN_TYPE: set('chartreuse','mauve','lime green','puce')
        COLUMN_KEY:
        EXTRA:
        PRIVILEGES: select,insert,update,references
COLUMN_COMMENT:

```

下面是一些最可能被使用的 COLUMNS 表数值：

- COLUMN_NAME 指明了数据列名称。
- ORDINAL_POSITION 是表中定义的数据列位置。
- COLUMN_DEFAULT 是数据列的默认值。
- IS_NULLABLE 通过 YES 或 NO 来指明数据列是否能包括空值。
- DATA_TYPE 和 COLUMN_TYPE 提供数据类型信息。DATA_TYPE 是数据类型的关键字，COLUMN_TYPE 包含类型特征的附加信息。
- CHARACTER_SET_NAME 和 COLLATION_NAME 指明字符串数据列的字符集和 Collation。当没有字符串数据列时其值为 NULL。

为了获取单独一个数据列的信息，可以在 WHERE 子句中增加为正确 COLUMN_NAME 数值命名的条件：

```
mysql> SELECT * FROM INFORMATION_SCHEMA.COLUMNS
-> WHERE TABLE_SCHEMA = 'cookbook' AND TABLE_NAME = 'item'
-> AND COLUMN_NAME = 'colors'\G
***** 1. row *****
    TABLE_CATALOG: NULL
    TABLE_SCHEMA: cookbook
    TABLE_NAME: item
    COLUMN_NAME: colors
    ORDINAL_POSITION: 3
    COLUMN_DEFAULT: puce
    IS_NULLABLE: YES
    DATA_TYPE: set
    CHARACTER_MAXIMUM_LENGTH: 32
    CHARACTER_OCTET_LENGTH: 32
    NUMERIC_PRECISION: NULL
    NUMERIC_SCALE: NULL
    CHARACTER_SET_NAME: latin1
    COLLATION_NAME: latin1_swedish_ci
    COLUMN_TYPE: set('chartreuse','mauve','lime green','puce')
    COLUMN_KEY:
    EXTRA:
    PRIVILEGES: select,insert,update,references
    COLUMN_COMMENT:
```

如果你仅仅想确定信息的类型，将 SELECT * 替换为感兴趣的数值的列表：

```
mysql> SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE
-> FROM INFORMATION_SCHEMA.COLUMNS
-> WHERE TABLE_SCHEMA = 'cookbook' AND TABLE_NAME = 'item';
+-----+-----+-----+
| COLUMN_NAME | DATA_TYPE | IS_NULLABLE |
+-----+-----+-----+
| id          | int       | NO        |
| name        | char      | YES       |
| colors      | set       | YES       |
+-----+-----+-----+
```

INFORMATION_SCHEMA 的内容很容易在程序中使用。下面是说明该过程的一个 PHP 函数。它将数据库和表的名称作为参数，从 INFORMATION_SCHEMA 中选择获取表的数据列名称的列表，同时将名称作为数组返回。ORDER BY ORDINAL_POSITION 确认数组中的名称是按照表中定义的顺序返回的。

```
function get_column_names ($conn, $db_name, $tbl_name)
{
    $stmt = "SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
        WHERE TABLE_SCHEMA = ? AND TABLE_NAME = ?
        ORDER BY ORDINAL_POSITION";
    $result =& $conn->query ($stmt, array ($db_name, $tbl_name));
    if (PEAR::isError ($result))
        return (FALSE);
    $names = array();
    while (list ($col_name) = $result->fetchRow ())
        $names[] = $col_name;
    $result->free ();
    return ($names);
}
```

完成同样功能的 Ruby DBI 程序如下：

```
def get_column_names(dbh, db_name, tbl_name)
    stmt = "SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
        WHERE TABLE_SCHEMA = ? AND TABLE_NAME = ?
        ORDER BY ORDINAL_POSITION"
    return dbh.select_all(stmt, db_name, tbl_name).collect { |row| row[0] }
end
```

而 Python 的代码则是这样：

```
def get_column_names (conn, db_name, tbl_name):
    stmt = """
        SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
        WHERE TABLE_SCHEMA = %s AND TABLE_NAME = %s
        ORDER BY ORDINAL_POSITION
    """
    cursor = conn.cursor ()
    cursor.execute (stmt, (db_name, tbl_name))
    names = []
    for row in cursor.fetchall ():
        names.append (row[0])
    cursor.close ()
    return (names)
```

在 Perl DBI 中，该操作是没什么价值的，因为 selectcol_arrayref() 函数直接返回查询结果的第一个数据列：

```
sub get_column_names
{
    my ($dbh, $db_name, $tbl_name) = @_;
    my $stmt = "SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
        WHERE TABLE_SCHEMA = ? AND TABLE_NAME = ?
        ORDER BY ORDINAL_POSITION";
```

```
my $ref = $dbh->selectcol_arrayref ($stmt, undef, $db_name, $tbl_name);
return defined ($ref) ? @{$ref} : ();
}
```

上面展示的程序返回的是仅仅只包含数据列名称的数组。如果你需要更多的数据列信息，你可以编写更多的通用程序以返回结构数组，每一个结构中包含关于给定数据列的信息。本书章节分布的程序库目录中包含一些实例程序。可以在库文件中查找名为 `get_column_info()` 的程序。

使用 SHOW COLUMNS 来获取表结构

`SHOW COLUMNS` 对表中的每个数据列生成一个输出的数据行，每个数据行提供关于相应数据列的不同方面的信息（注 1）。下面的范例演示了 `SHOW COLUMNS` 产生的 `item` 表的输出。

```
mysql> SHOW COLUMNS FROM item\G
***** 1. row ****
Field: id
Type: int(10) unsigned
Null: NO
Key: PRI
Default: NULL
Extra: auto_increment
***** 2. row ****
Field: name
Type: char(20)
Null: YES
Key:
Default: NULL
Extra:
***** 3. row ****
Field: colors
Type: set('chartreuse','mauve','lime green','puce')
Null: YES
Key:
Default: puce
Extra:
```

语句显示的信息如下：

- `Field` 指明了数据列名称
- `Type` 显示了数据类型
- `Null` 为 `YES` 说明了数据列包含 `NUL` 值，反之亦然
- `Key` 提供了数据列是否被索引的信息
- `Default` 指明了默认数值
- `Extra` 列举了其余的信息

注 1: `SHOW COLUMNS FROM tbl_name` 等价于 `SHOW FIELDS tbl_name` 或 `DESCRIBE tbl_name`。

SHOW COLUMNS 的格式偶尔会改变，但是刚才描述的区域必须是一直处于可用状态。SHOW FULL COLUMNS 显示更多的区域。

SHOW COLUMNS 支持采用 SQL 模式的 LIKE 语句：

```
SHOW COLUMNS FROM tbl_name LIKE 'pattern';
```

模式被按照 SELECT 语句的 WHERE 子句中 LIKE 操作符同样的方式解释。(关于模式匹配的信息，请查阅第 5 章第 10 节) SHOW COLUMNS 与 LIKE 子句一起显示匹配该模式的任何数据列的信息。如果你指明了一个文字的数据列名称，字符串仅匹配该名称，同时 SHOW COLUMNS 会显示信息。然而，这个不引人注意的地方存在一个陷阱。如果你的数据列名称包含 SQL 模式字符（%或_）同时你希望从文字上匹配它们，你必须在模式字符串中的这些字符之前增加一个反斜杠以避免匹配其他名称。%字符通常不会被用于数据列名称中，但是_（下划线）却常常被使用，所以你很可能陷入这种情形里。假设你有一张包含二氧化碳测试结果的表，该结果被放置在名为 co_2 的数据列中，同时三角函数的余弦和余切函数的数据列名称为 cos1、cos2、cot1 和 cot2。如果你仅仅想得到关于 co_2 的信息，你不能使用如下的语句：

```
SHOW COLUMNS FROM tbl_name LIKE 'co_2';
```

模式字符串中的_（下划线）字符意味着“匹配任意字符”，所以该语句会返回关于 co_2、cos2 以及 cot2 的数据行。为了只匹配 co_2 数据列，可以这样编写 SHOW 命令：

```
SHOW COLUMNS FROM tbl_name LIKE 'co\_\_2';
```

在程序中，你可以在将数据列名称输入到 SHOW 语句之前使用你的 API 语言的模式匹配能力来避免 SQL 模式字符的影响。例如，在 Perl、Ruby 和 PHP 中，你可以使用下面的表达式。

Perl:

```
$name =~ s/([%_])/\\\$1/g;
```

Ruby:

```
name.gsub!(/([%_])/, '\\\\\\1')
```

PHP:

```
$name = ereg_replace ("([%_])", "\\\\1", $name);
```

对于 Python，可以引入 re 模块，同时使用它的 sub() 方法：

```
name = re.sub(r'([%_])', r'\\\\1', name)
```

对于 Java，使用 java.util.regex 包：

```
import java.util.regex.*;  
Pattern p = pattern.compile("([_%])");
```

```
Matcher m = p.matcher(name);
name = m.replaceAll ("\\\\\\\$1");
```

如果觉得这些表达式看起来有太多的反斜杠，记住 API 语言处理器本身会处理反斜杠并且在执行模式匹配之前会去掉一层。为了在结果中得到语义上的反斜杠，必须在模式中输入双反斜杠。PHP 在这之上有另外一层，因为它描绘了一套，同时模式处理器描绘了一套。

需要去除%和_以从语义上匹配 LIKE 模式的也能应用到其他形式的在 LIKE 子句中允许名称模式的 SHOW 语句中，例如 SHOW TABLES 和 SHOW DATABASES。

使用 CREATE TABLE 来获取表结构

另一个从 MySQL 中获取表结构的方法是从定义表的 CREATE TABLE 语句中得到信息。你可以通过使用 SHOW CREATE TABLE 语句来获取该信息：

```
mysql> SHOW CREATE TABLE item\G
***** 1. row *****
    Table: item
Create Table: CREATE TABLE `item` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `name` char(20) DEFAULT NULL,
  `colors` set('chartreuse','mauve','lime green','puce') DEFAULT 'puce',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

从命令行中，如果你使用--no-data 选项同样可以从 mysqldump 中得到相同的信息，该选项将仅往 mysqldump 中填充表的结构而非数据：

```
% mysqldump --no-data cookbook item
-- MySQL dump 10.10
--
-- Host: localhost      Database: cookbook
-- -----
-- Server version      5.0.27-log
--

-- Table structure for table `item`
--

CREATE TABLE `item` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` char(20) default NULL,
  `colors` set('chartreuse','mauve','lime green','puce') default 'puce',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

该格式具有非常广的信息范围并且很容易读取，因为它显示数据列信息的格式与你习惯的在第一位置创建的表的格式类似。它也很清楚的显示了索引结构，而其他方法不能实现。

然而，你可能会发现这个方法用于可视化的结构检测比用于程序更加有效。该信息不是按照规则的数据行和数据列格式提供的，所以分析它很困难。同时，当 CREATE TABLE 语句的功能有增强时该格式也随之会有一定的变化，这种变化有时发生在 MySQL 的功能被扩展时。

9.7 取得 ENUM 和 SET 数据列信息

Getting ENUM and SET Column Information

问题

你想知道 ENUM 或 SET 数据列包含哪些成员。

解决方案

这个问题是获取表结构元数据的子集。从表元数据中获取数据列定义，然后从定义中提取成员列表。

讨论

知道 ENUM 或 SET 数据列的合法数值列表通常是很用的。假设你想发布一个带有弹出菜单的网页表格，其中含有对应 ENUM 数据列的每个合法数值的选项，例如可以被预订的衣服的尺码，或者是传递包裹可用的运送方式。你可以将这些选项写入到生成表格的脚本中，但是如果你之后改变了数据列（例如，增加了一个新的枚举数值），那么你就在数据列和使用它的脚本之间引入了矛盾。如果以使用表元数据来查找合法数值，那么脚本将一直生成包含正确数值集的弹出菜单。类似的方法可以使用在 SET 数据列上。

为了找出 ENUM 或者 SET 数据列可以包含什么数值，可以使用本章第 6 节描述的方法来获取数据列定义并在定义中查找数据类型。例如，如果你从 INFORMATION_SCHEMA.COLUMNS 表中进行选择操作，那么 item 表中颜色数据列的 COLUMN_TYPE 数值看起来是这样的：

```
set ('chartreuse', 'mauve', 'lime green', 'puce')
```

ENUM 数据列类似，除非被告知是 enum 而非 set。对这两种数据类型，允许的数值通过分离初始词和括号（采用逗号分隔），并去掉每个数值的引号被提取出来。让我们编写一个 get_enumorset_info() 过程以从数据类型定义中提取这些数值。当我们运行它时，我们可以让程序返回数据列的类型、默认数值以及该数值是否可以为 NULL 值。同时该过程也可以被那些不仅仅只需要数值列表的脚本使用。下面是用 Ruby 实现的一个版本。它的参数是数据库句柄、数据库名称、表名称以及数据列名称。它返回一个包含对应数据库定义的各种不同表现形式条目的混合表（当数据列不存在时返回 nil 值）：

```

def get_enumorset_info(dbh, db_name, tbl_name, col_name)
  row = dbh.select_one(
    "SELECT COLUMN_NAME, COLUMN_TYPE, IS_NULLABLE, COLUMN_DEFAULT
     FROM INFORMATION_SCHEMA.COLUMNS
    WHERE TABLE_SCHEMA = ? AND TABLE_NAME = ? AND COLUMN_NAME = ?",
    db_name, tbl_name, col_name)
  return nil if row.nil?
  info = {}
  info["name"] = row[0]
  return nil unless row[1] =~ /^ENUM|SET\b((.*))$/i
  info["type"] = $1
  # 根据逗号来划分值列表，去除每个单词尾部的引号
  info["values"] = $2.split(",").collect { |val| val.sub(/^'(.*)'$/, "\\\1") }
  # 确定列是否能包含NULL值
  info["nullable"] = (row[2].upcase == "YES")
  # 获取默认值 (nil代表NULL)
  info["default"] = row[3]
  return info
end

```

在检查数据类型和无效特征时该过程采用不区分大小写的匹配原则。这是为了预防将来元数据结果中的字符形式可能被改变。

下面的范例展示了一个访问和显示 `get_enumorset_info()` 返回的混合表的每个元素的方法：

```

info = get_enumorset_info(dbh, db_name, tbl_name, col_name)
puts "Information for " + db_name + "." + tbl_name + "." + col_name + ":"+
  if info.nil?
    puts "No information available (not an ENUM or SET column?)"
  else
    puts "Name: " + info["name"]
    puts "Type: " + info["type"]
    puts "Legal values: " + info["values"].join(",")
    puts "Nullable: " + (info["nullable"] ? "yes" : "no")
    puts "Default value: " + (info["default"].nil? ? "NULL" : info["default"])
  end

```

上面的代码对 `item` 表的 `colors` 数据列产生了如下的输出：

```

Information for cookbook.item.colors:
Name: colors
Type: set
Legal values: chartreuse,mauve,lime green,puce
Nullable: yes
Default value: puce

```

其他 API 的相同功能的过程是类似的。这些过程对于在网页表格中产生元素列表特别方便（参考第 19 章第 2 节和第 3 节）。

9.8 在应用程序中使用表结构信息

Using Table Structure Information in Applications

问题

获取表结构信息之后，你计划用它来做什么？

解决方案

有很多事情可以做：你可以显示表数据列的列表，创建网页形式的元素，生成修改 ENUM 或 SET 数据列的 ALTER TABLE 语句，等等。

讨论

本节描述了 MySQL 提供的表结构信息的一些用途。

显示数据列列表

表信息最简单的用途可能是显示表数据列的一列。这在基于网页或 GUI 的应用程序中很常见，这些程序允许用户创建语句来交互的实现从一列中选取表的数据列同时输入一个数值来与数据列中的数值相比较。本章第 6 节展示的 `get_column_names()` 过程就能够用于这样的列表显示。

交互的记录编辑

表结构的信息对于交互式记录编辑应用程序非常有用。假设你有一个应用程序，从数据库中获取记录，并以表格的形式显示记录的内容，这样用户可以编辑它，并且在用户修改表格并提交之后可以更新数据库中的记录。你可以使用表结构信息来确认数据列数值。例如，如果数据列是 ENUM 类型，你可以找出有效的枚举数值并通过用户检查来确认其数值是否合法并检查被提交的数值。如果数据列类型是整数，那么检查被提交的数值以确认它是由数字组成的，可能之前有+或-。如果数据列包含日期，那么检查合法的日期格式。

但是如果用户让整个区域为空，那么怎么办？如果该区域对应表中的 CHAR 数据列，那么你是将数据列数值设为 NULL 还是空字符串？这也是一个可以通过检查表的结构来回答的问题。确定数据列是否包括 NULL 数值，如果是的话，将数据列设为 NULL；否则，将其设为空字符串。

将数据列定义映射到网页元素

一些数据类型，如 ENUM 和 SET 类型，很自然的对应于网页表格的元素：

- 一个 ENUM 具有你从一个固定的数值集合中选取的数值。这与一组单选按钮、一个弹出菜单或者一个单选的列表类似。
- SET 数据列类似，除非你可以选择多个数值。这对应于一组检验钮或者一个多选的列表。

通过使用表元数据来访问数据列的这些类型的定义，你可以很容易的确定一个数据列的合法数值并自动的将它们正确的映射到正确的表格元素。这能够让你将可用的数据列举在用户面前，用户可以很容易的选择所需要的数值而不用键入任何信息。本章第 7 节讨论了如何获取数据列的这些类型的定义。这里提出的方法在第 19 章中使用，在那里讨论了如何生成表格的细节。

在 ENUM 或 SET 数据列定义中增加元素

当你需要修改一个数据列的定义时，你可以使用 ALTER TABLE。然而，在一个 ENUM 或 SET 数据列定义中增加一个新的元素是真的很不方便，因为你不仅需要列出新的元素还必须列出所有存在的元素、默认数值，如果数据列不能包含 NULL 数值那么还必须列出 NOT NULL 数值。假设你想在 item 表的 colors 数据列中增加“hot pink”，而 item 表具有如下的结构：

```
CREATE TABLE item
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name    CHAR(20),
    colors  SET('chartreuse', 'mauve', 'lime green', 'puce') DEFAULT 'puce',
    PRIMARY KEY (id)
);
```

为了改变数据列定义，可以按照如下方式使用 ALTER TABLE：

```
ALTER TABLE item
MODIFY colors
SET('chartreuse', 'mauve', 'lime green', 'puce', 'hot pink')
DEFAULT 'puce';
```

ENUM 的定义并不包含许多元素，所以手工输入那条语句并不困难。然而，如果一个数据列有更多的元素，那么像那样输入语句就会面临更多的困难并更容易出错。为了在增加一个新的元素时避免再次输入已经存在的定义，你有一个策略性的选择：

- 编写一个为你工作的脚本。它可以检查表定义并使用数据列元数据生成 ALTER TABLE 语句。
- 使用 mysqldump 来获取包含当前数据列定义的 CREATE TABLE 语句，并在文本编辑器中修改语句以生成正确的 ALTER TABLE 语句来更改定义。

作为第一个方法的实现，我们开发了一个 Python 脚本 add_element.py，当给定数据库和表的名称、一个 ENUM 或 SET 数据列名称以及新的元素数值时，它可以自动的产生正确的

ALTER TABLE 语句。add_element.py 将使用该信息来构造正确的 ALTER TABLE 语句并显示它：

```
% add_element.py cookbook item colors "hot pink"
ALTER TABLE `cookbook`.`item`
MODIFY `colors`
SET ('chartreuse','mauve','lime green','puce','hot pink')
DEFAULT 'puce';
```

通过使用 add_element.py 生成的语句，你可以选择将它放入 mysql 中立即执行或者将输出保存到的一个文件中：

```
% add_element.py cookbook item colors "hot pink" | mysql cookbook
% add_element.py cookbook item colors "hot pink" > stmt.sql
```

add_element.py 脚本的第一部分引入必要的模块并检查命令行参数。这一点非常直接：

```
# add_element.py - 生成 ALTER TABLE 语句以添加一个元素到一个 ENUM 或 SET 列中

import sys
import MySQLdb
import Cookbook

if len (sys.argv) != 5:
    print "Usage: add_element.py db_name tbl_name col_name new_element"
    sys.exit (1)
(db_name, tbl_name, col_name, new_elt) = (sys.argv[1:5])
```

在连接上 MySQL 服务器之后（这里没有列出相应的代码，但是脚本中确实存在），脚本会检查 INFORMATION_SCHEMA 来获取数据列定义，看它是否允许 NULL 值以及它的默认数值是多少。接下来的代码检查以确认表中的数据列确实存在：

```
stmt = """
SELECT COLUMN_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = %s AND TABLE_NAME = %s AND COLUMN_NAME = %s
"""

cursor = conn.cursor ()
cursor.execute (stmt, (db_name, tbl_name, col_name))
info = cursor.fetchone ()
cursor.close
if info == None:
    print "Could not retrieve information for table %s.%s, column %s" \
          % (db_name, tbl_name, col_name)
    sys.exit (1)
```

在这里，如果 SELECT 语句执行成功，它所产生的信息将作为一个有效的元组存储在 info 变量中。我们需要使用这个元组中的几个元素。最重要的是 COLUMN_TYPE 数值，它将提供包含数据列定义的 enum(...) 或 set(...) 字符串。我们可以使用该字符串以确认该数据

列确实是 ENUM 或 SET 类型，同时插入语句结束之前在该字符串中加入新的元素。对于 colors 数据列，我们希望将下面这点：

```
set('chartreuse', 'mauve', 'lime green', 'puce')
```

改变为：

```
set('chartreuse', 'mauve', 'lime green', 'puce', 'hot pink')
```

检查数据列数值能否为 NULL 以及默认数值是什么也非常必要，这样程序可以在 ALTER TABLE 语句中增加正确的信息。下面的代码可以完成这些工作：

```
# 获得数据类型字符串；确保它是以 ENUM 或 SET 开始
type = info[0]
if type[0:5].upper() != "ENUM(" and type[0:4].upper() != "SET(":
    print "table %s.%s, column %s is not an ENUM or SET" % \
        (db_name, tbl_name, col_name)
    sys.exit(1)
# 在结尾的括号前插入逗号和正确引用的新元素
type = type[0:len(type)-1] + "," + conn.literal (new_elt) + ")"
# 如果列不能包含 NULL 值，加上 "NOT NULL"
if info[1].upper() == "YES":
    nullable = ""
else:
    nullable = "NOT NULL ";
# 构建 DEFAULT 子句（引用值是必须的）
default = "DEFAULT " + conn.literal (info[2])
print "ALTER TABLE `%s`.`%s`\n MODIFY `%s`\n %s\n %s;" \
    % (db_name, tbl_name, col_name, type, nullable, default)
```

这就是我们需要的结果。现在你有了一个可以更改 ENUM- 或 SET- 的程序。当然，add_element.py 的功能还相当简单，有不同的方法可以进一步增强它的功能：

- 确认你正在往数据列中增加的数值原本并不存在。
- 修改 add_element.py 使它在数据列名称之后接纳不止一个参数，同时将它们都增加到数据列定义中。
- 增加一个选项以指明给定元素应该被删除而非加入。

另一个改变 ENMU 或 SET 数据列的方法是，将当前的定义放到一个文件中然后编辑这个文件以产生正确的 ALTER TABLE 语句。

1. 运行 mysqldump 已获取包含数据列定义的 CREATE TABLE 语句：

```
% mysqldump --no-data cookbook item > test.txt
```

--no-data 选项告诉 mysqldump 不要从表中导出数据，使用它的原因是因为这里只需要创建表的语句。结果文件 test.txt 应该包含如下语句：

```
CREATE TABLE `item` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `name` char(20) DEFAULT NULL,
  `colors` set('chartreuse','mauve','lime green','puce') DEFAULT 'puce',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

2. 编辑 test.txt 将除 colors 数据列定义以外的其他部分全部去掉：

```
colors` set('chartreuse','mauve','lime green','puce') DEFAULT 'puce',
```

3. 修改定义以生成含有新的元素并且结尾处有分号的 ALTER TABLE 语句：

```
ALTER TABLE item MODIFY
`colors` set('chartreuse','mauve','lime green','puce','hot pink')
DEFAULT 'puce';
```

4. 将 test.txt 写回并保存它，然后退出编辑器并将 test.txt 作为一个批处理文件放入到 mysql：

```
% mysql cookbook < test.txt
```

对于简单的数据列，这个过程比手工输入 ALTER TABLE 语句的工作量更大。然而，对于具有冗长定义的 ENUM 和 SET 数据列而言，使用编辑器从 mysqldump 的输出中创建 mysql 批处理文件则有意义得多。当你想删除或者重排序 ENUM 或 SET 数据列的成员时，或者你想从数据列定义中增加或删除成员时，这个技术也非常有用。

选择除确定数据列以外的所有信息

有时你希望从表中获取“几乎所有”的数据列。假设你有一张图片表，其中包含存储图片的可能非常大的名为 data 的 BLOB 数据列，以及其他用来指明 BLOB 数据列特征的数据列，如它的 ID、描述等等。很容易编写一条 SELECT * 语句来获取所有的数据列，但是如果所需要的所有信息只是描述图片的信息而非图片本身，那么将巨大的 BLOB 数值与其他数据列绑定在一起会降低效率。因此，你可能希望选择数据行中除 data 数据列以外的所有信息。

不幸的是，在 SQL 中并没有直接语句支持“选择除此之外的所有列”。你必须明确的给出除 data 以外的所有数据列的名称。另一方面，可以通过使用表结构信息来构建那样的语句。提取数据列名称列表，删除不需要的数据列，然后从那些保留的数据列中构造 SELECT 语句。下面的例子显示了如何在 PHP 中完成这样的工作，使用前面章节中开发的 get_column_names() 函数从表中获取数据列名称：

```
$names = get_column_names ($conn, $db_name, $tbl_name);
$stmt = "";
# 构建要查询的列的列表：除"data"外的所有列
foreach ($names as $index => $name)
{
```

```

if ($name == "data")
    continue;
if ($stmt != "") # 在列名间加入逗号
    $stmt .= ", ";
$stmt .= "`$name`";
}
$stmt = "SELECT $stmt FROM `$db_name`.`$tbl_name`";

```

功能相同的构建该语句的 Perl 代码简短一点（同时相应的更含混点）：

```

my @names = get_column_names ($dbh, $db_name, $tbl_name);
my $stmt = "SELECT `"
    . join ("`, `", grep (!/^data$/, @names))
    . "` FROM `$db_name`.`$tbl_name`";

```

不管你使用哪种语言，结构都是你可以用来选择除 data 以外的所有数据列的语句。它可能比 SELECT * 更有效，因为它不会通过网络获取 BLOB 数值。当然，这个过程引入了额外的与服务器之间的通信以执行获取数据列名称的语句，所以你应该考虑你计划在 SELECT 语句中使用的内容。如果你计划仅仅只获取一个数据行，可能简单的选择整个数据行比引入额外的通信开销更有效率。但是如果你将要选择许多数据行，通过忽略 BLOB 数据行而得到的网络通行流量的减少可能值得因为获取表结构而产生的额外查询的开销。

9.9 获取服务器元数据

Getting Server Metadata

问题

你想要 MySQL 服务器告诉你关于它本身的信息。

解决方案

有几个 SQL 函数和 SHOW 语句可以返回服务器的信息。

讨论

MySQL 提供几个 SQL 函数和语句，能给出关于服务器本身和当前客户端连接的信息。你可能在这里列举的信息中找出一点有用的信息。为了获取他们提供的信息，使用下列语句，然后处理它的结果集。两种 SHOW 语句都允许使用 LIKE ‘pattern’ 子句来限制结果中的数据行之始匹配给定模式的数据行。

语句	语句所生成的信息
SELECT VERSION()	服务器版本字符串
SELECT DATABASE()	默认的数据库名称，如果没有则为空

语句	语句所生成的信息
SELECT USER()	客户端连接时给出的当前用户
SELECT CURRENT_USER()	用户用来检查客户端权限的
SHOW GLOBAL STATUS	服务器的全局状态指示器
SHOW VARIABLES	服务器配置变量

一个给定的 API 可能提供任意一个方法来访问信息的这些类型。例如，JDBC 有几个数据库无关的方来获取服务器元数据。使用你的连接对象来获取数据库元数据，同时调用正确的方法来获取你感兴趣的信息。你应该查阅 JDBC 参考资料来获取一个完整的列表，不过这里有一点有代表性的例子：

```
DatabaseMetaData md = conn.getMetaData ();
//也可以使用 SELECT VERSION()得到这个信息
System.out.println ("Product version: " + md.getDatabaseProductVersion ());
// 这与 SELECT USER()类似，但是不包括主机名称。
System.out.println ("Username: " + md.getUserName ());
```

9.10 编写适合 MySQL 服务器版本的应用程序

Writing Applications That Adapt to the MySQL Server Version

问题

你想要使用一个给定的仅在特定 MySQL 版本中才有效的特征。

解决方案

向服务器询问它的版本号。如果服务器版本太老而不能支持给定的特征，你可能需要回退到一个存在的较老的工作区。

讨论

每一个版本的 MySQL 都增加新的特征。如果你在编写一个需要特定特征的应用程序，那么需要检查服务器版本以确定该特征是否被支持。如果不支持，你必须执行某种工作区（假设它存在）。

为了获取服务器版本，使用 `SELECT VERSION()` 语句。结果看起来有些像 `5.0.13-rc` 或 `4.1.10a` 的字符串。换句话说，它返回的字符串包括主版本号、副版本号以及最小的版本号，可能在最小版本号还有些字母，还可能有后缀。如果你想要生成用户状态的显示，版本字符串可以用来做演示的目的。然而，相比之下，使用字母工作比较简单一点——特别是给定的 `Mmmtt` 格式的 5 位数字，其中 `M`、`mm`、`tt` 分别代表主版本号、副版本号以及最

小版本号。转换可以通过将字符串周期的分割来实现，在第三部分处从第一个非数字的字符开始将后缀去除，然后连接该部分。例如，5.0.13-rc. 变成 50013，而 4.0.10a 变成 40110。

这里有一个 Perl DBI 的函数，它将数据库句柄作为参数并返回一个同时包括字符串形式和数字形式的两个服务器版本信息元素的列表。这段代码假设副版本号以及最小版本号部分小于 100 个数字并且大于 2 个数字。这是一个合理的假设，因为 MySQL 本身的源代码也使用了相同的格式。

```
sub get_server_version
{
    my $dbh = shift;
    my ($ver_str, $ver_num);
    my ($major, $minor, $teeny);

    # 将结果取到一个标量字符串里
    $ver_str = $dbh->selectrow_array ("SELECT VERSION()");
    return undef unless defined ($ver_str);
    ($major, $minor, $teeny) = split (/./, $ver_str);
    $teeny =~ s/\D*$/ /; # 如果存在的话去除任意非数字的后缀
    $ver_num = $major*10000 + $minor*100 + $teeny;
    return ($ver_str, $ver_num);
}
```

为了第一次获取两种形式的版本信息，可以这样调用函数：

```
my ($ver_str, $ver_num) = get_server_version ($dbh);
```

为了只获取其中一个数值，像下面这样调用函数：

```
my $ver_str = (get_server_version ($dbh))[0]; # 字符串形式
my $ver_num = (get_server_version ($dbh))[1]; # 数值形式
```

下面的例子演示了如何使用数字版本数值来检查服务器是否支持特定的特征：

```
my $ver_num = (get_server_version ($dbh))[1];
printf "Quoted identifiers: %s\n", ($ver_num >= 32306 ? "yes" : "no");
printf "UNION statement:   %s\n", ($ver_num >= 40000 ? "yes" : "no");
printf "Subqueries:        %s\n", ($ver_num >= 40100 ? "yes" : "no");
printf "Views:              %s\n", ($ver_num >= 50001 ? "yes" : "no");
printf "Strict SQL mode:   %s\n", ($ver_num >= 50002 ? "yes" : "no");
printf "Events:             %s\n", ($ver_num >= 50106 ? "yes" : "no");
```

9.11 确定默认数据库

Determining the Default Database

问题

任何数据库都可以被选取作为默认数据库吗？它的名称是什么？

解决方案

使用 DATABASE() 函数。

讨论

SELECT DATABASE() 返回默认数据库的名称，如果没有数据库被选取那么返回 NULL。下面的 Ruby 代码使用了显示包括当前连接的状态信息的语句：

```
db = dbh.select_one("SELECT DATABASE()")[0]
puts "Default database: " + (db.nil? ? "(no database selected)" : db)
```

注意在 MySQL4.1.1 之前，如果当前没有正在使用的数据库，DATABASE() 返回空字符串（而非 NULL）。

9.12 监测 MySQL 服务器

Monitoring the MySQL Server

问题

你想要找出服务器是如何配置的或者监控其状态

解决方案

SHOW VARIABLES 和 SHOW STATUS 对这一点非常有用。

讨论

SHOW VARIABLES 和 SHOW STATUS 语句提供了服务器配置和性能信息：

```
mysql> SHOW VARIABLES;
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| back_log           | 50      |
| basedir            | /usr/local/mysql/ |
| bdb_cache_size     | 8388600  |
| bdb_log_buffer_size | 0       |
| bdb_home            |         |
...
mysql> SHOW /*!50002 GLOBAL */ STATUS;
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| Aborted_clients    | 319     |
| Aborted_connects   | 22      |
| Bytes_received     | 32085033 |
| Bytes_sent          | 26379272 |
| Connections         | 65684   |
...
```

上述两条语句都允许使用 SQL 模式的 `LIKE 'pattern'` 子句。在这种情形下，只有匹配模式的变量名称所在的数据行被返回。

注释`/*!50002 GLOBAL */`出现在 `SHOW STATUS` 语句中是由于 MySQL5.0.2 所做的改动。在 MySQL5.0.2 之前，状态信息是全局的（服务器层次的数值）。在 5.0.2 中，状态变量具有全局的和会话的（每个连接）数值，同时 `SHOW STATUS` 被扩展为默认采用 `GLOBAL` 或 `SESSION` 修改器，如果两者都没有被给出，那么显示会话的数值。该注释导致 5.0.2 及更高版本的服务器显示全局数值。5.0.2 之前的服务器会忽略该注释，但是显示全局数值，因为在那些版本中仅有全局数值存在。这个习惯用法贯穿本章始终。

系统和状态变量信息对于编写管理应用程序非常有用。例如，MyISAM 关键字缓存的命中率是衡量关键字命中缓存而非从磁盘中读取数值的频率。下面的公式计算了点击率，其中 `Key_reads` 和 `Key_read_requests` 分别代表磁盘读取次数和请求次数：

```
1 - (Key_reads / Key_read_requests)
```

接近 1 的数值指明较高的命中率，意味着该关键字缓存非常有效。接近 0 的数值则说明低命中率（可能是一个你应该增加系统 `key_buffer_size` 变量的数值以使用更大的缓存的信号）。在程序中很容易计算命中率，下面的 Ruby 代码就是个例子：

```
# 执行 SHOW STATUS 以获取服务器相关状态变量
# 使用名称和数值来创建通过名称键入的数值的混合信息。
stat_hash = {}
stmt = "SHOW /*!50002 GLOBAL */ STATUS LIKE 'Key_read%'""
dbh.select_all(stmt).each do |name, value|
  stat_hash[name] = value
end

key_reads = stat_hash["Key_reads"].to_f
key_reqs = stat_hash["Key_read_requests"].to_f
hit_rate = key_reqs == 0 ? 0 : 1.0 - (key_reads / key_reqs)

puts "      Key_reads: #{key_reads}"
puts "Key_read_requests: #{key_reqs}"
puts "      Hit rate: #{hit_rate}"
```

另一个例子，你可能编写一个长时间运行的程序以周期性的探测服务器来监控其行为。一个这种类型的简单应用可能要求服务器报告它接收到的连接数目和它的运行时间，以确定平均连接行为的运行显示。获取该信息的语句是：

```
SHOW /*!50002 GLOBAL */ STATUS LIKE 'Connections';
SHOW /*!50002 GLOBAL */ STATUS LIKE 'Uptime';
```

如果你想避免每次都不得不重新连接你给出的语句，你可以要求服务器周期性的暂停其客户端同时在比该时间短的时间片探测它的信息。你可以使用下面的语句来获取暂停的数值（以秒为长度）：

```
SHOW VARIABLES LIKE 'wait_timeout' ;
```

默认数值是 28800(8 小时)，但是在你的系统上可能被配置为不同的数值。

对于系统变量，一个不同访问它们的数值的方法是通过`@@ var_name`的方式来引用它们。例如：

```
mysql> SELECT @@wait_timeout;
+-----+
| @@wait_timeout |
+-----+
| 28800          |
+-----+
```

这个方法提供了让你在一个单独的数据行中选择多个数值的便利，同时你可以直接在表达式中使用该数值。

“MySQL 不确定规则”

量子现象测量的海森堡测不准原理与 MySQL 有点类似。如果你想监控 MySQL 的状态以查看它在时间上是如何变迁的，你可能注意到一些奇怪现象，因为对某些指示器，在你每次测量时你都改变了你正在测量的数值！例如，你能通过如下语句确定服务器接收到的语句的数目：

```
SHOW /*!50002 GLOBAL */ STATUS LIKE 'Questions'
```

然而，语句本身也是一条服务器接收的语句，所以每次你提交它，你就引起了所请求问题的数值的变化。从效果上说，你的性能估测方法影响了测量本身，这些你可能需要纳入考虑范围。

9.13 确定服务器支持哪个存储引擎

Determining Which Storage Engines the Server Supports

问题

你想要知道你能否使用给定的存储引擎创建一张表。

解决方案

使用`SHOW ENGINES`语句询问服务器支持哪种存储引擎。

讨论

`SHOW ENGINES`语句提供关于服务器支持哪种存储引擎的信息。它的输出如下所示：

```
mysql> SHOW ENGINES\G
***** 1. row ****
```

```
Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
***** 2. row *****
Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
***** 3. row *****
Engine: InnoDB
Support: YES
Comment: Supports transactions, row-level locking, and foreign keys
***** 4. row *****
Engine: BerkeleyDB
Support: NO
Comment: Supports transactions and page-level locking
...

```

Engine 数值指明了存储引擎的名称，Support 数值指明了它的状态。如果 Support 是 YES 或 DEFAULT，那么该引擎是可用的。如果数值是 NO 或 DISABLED，该引擎是不可用的。下面的 Ruby 方法使用了这些规则来确定引擎状态并返回了可用引擎的列表：

```
def get_storage_engines(dbh)
  engines = []
  dbh.select_all("SHOW ENGINES").each do |engine, support|
    engines << engine if ["YES", "DEFAULT"].include?(support.upcase)
  end
  return engines
end
```

数据导入导出

Importing and Exporting Data

10.0 引言

Introduction

假设现有一个名为 `somedata.csv` 的文件，包含 12 列以逗号定界（CSV）形式存放的数据。需要用其中第 2、11、5 和第 9 列来填充 MySQL 数据库之中一个包含 `name`、`birth`、`height` 和 `weight` 等列的表。同时要求确保 `height` 和 `weight` 的内容是正整数，并要求将 `birth` 的内容从 `MM/DD/YY` 格式转换成 `CCYY-MM-DD` 格式。你将如何做呢？

从某种角度来看，这个问题好像很特别，但是换个角度看，一点也不。因为每当你需要将数据传入 MySQL 数据库的时候，就会频繁地遇到需要按照一定要求对数据进行转换的问题。如果数据文件已经被很好地格式化并且为载入 MySQL 做好了准备，那当然再好不过。但是常见的情况却不是这样。所以，在将数据导入 MySQL 时经常需要将其加工成 MySQL 可以接收的格式。反过来也一样，从 MySQL 中导出的数据也需要加工成对其他应用程序有用的形式。

尽管在数据传递中有一些难度很大的工作，以至于需要大量手工检查和重新格式化，但是在大多数情况下都会有部分工作可以自动进行。实际上，所有数据传递问题中都会包含一些共同的转换问题。本章将探讨这些问题是什么；如何随心所欲地利用现有工具来应对这些问题；以及在必要时如何自己编写需要的工具。目的是讲解有代表性的技术和工具，而不是覆盖数据导入/导出的所有情况（一个不可能完成的任务）。你可以照葫芦画瓢原封不动地使用它们，也可以举一反三做些变化和适当改编，以适用于它们原来不能解决的地方。

（市面上还有许多商业化数据转换工具可以来帮你做这个工作，不过，本章的目的是让你学会如何自己做。）

本章教你的第一个诀窍是 MySQL 原生工具集，也就是 MySQL 自己提供的用于导入数据（`LOAD DATA` 语句和 `mysqlimport` 命令行程序）和导出数据（`SELECT ... INTO OUTFILE` 语句和 `mysqldump` 命令行程序）的工具集。对于不要求进行数据校验和重新格式化的操作，这些工具已经足够了。甚至对一些要求重新格式化的操作，它们也已足够，因为 `LOAD`

DATA 语句能够对数据进行预先处理，而 SELECT 语句能够利用函数和表达式将列的值转换成其他格式。

当 MySQL 原生导入/导出工具力有不逮的情况出现时，本章将继续教你新的诀窍——使用外部工具或自己开发。某种程度上，你可以使用已有的工具以避免自己开发。例如，使用 cut 命令可以从文件中抽取列的内容，而 sed 和 tr 命令可以作为后处理器来将 SQL 查询输出结果从一种格式转换成其他格式。但是，你很可能终将会有决定自己编写工具的那一天，到时候，你将面临两大类问题：

- 如何处理数据文件结构。当一个文件的格式不适合导入时，你需要将其转换成其他格式。这可能包括诸如改变列分隔符、改变行结束序列以及在文件中去除或重新排列所有列等等问题。
- 如何处理数据文件内容。如果你不能确定文件中的内容是否合法时，你需要对其进行预处理以校验和重新格式化内容。数字值可能需要校验以确定是否处在某个特定范围内，日期可能需要转换成 ISO 格式或者相反，等等诸如此类。

本章用到的程序片断和脚本的源代码可以在 `recipes` 发行的 `transfer` 目录下找到，另外部分工具函数包含在 `lib` 目录下的库文件中。对于比较短的工具，本章将会列出完整的源代码。对于比较长的工具，本章集中讨论其工作原理和用法，如果你想研究程序实现的细节，可以自己查看源代码。

本章解决的问题涉及大量文本处理和模式匹配。这恰恰是 Perl 的长处所在，所以文中展示的程序片断和工具主要用 Perl 编写。而 Ruby、PHP、Python 和 Java 都提供有模式匹配能力，所以，它们也都能完成大多数同样的任务。

导入和导出概述

在不同的应用程序之间传递数据，不兼容的数据文件格式以及对各类数值的不同解释规则会带来令人头痛的问题。不过，其中有些问题会反复不断出现。如果熟悉这些问题，你将会更容易地解决特定的导入和导出问题。

以最基本的形态来看，一段输入流不过是没有意义的字节的集合罢了。但要想成功地导入 MySQL，就必须能识别出哪些字节代表结构信息，哪些字节被框在结构之中代表数据值。由于这一识别能力是将输入分解成全适单元的关键，所以最基本的导入问题是：

- 记录分隔符是什么？知道这个可以让你将输入流分割成记录。

- 域定界符是什么？知道这个可以让你将记录再分割成域值。识别数据值的过程一般还包括剥离数值周围引号以及识别数值内部的换码序列。

将输入分解成记录和域是从输入中抽出数据值的关键。但是，在完成这一步后，数据值可能处在不能被直接使用的形式，你还需要考虑以下问题：

- 列的顺序和数目是否同数据库中表的结构相匹配？不匹配可能就需要对列进行重新排序或丢弃。
- 数据值是否需要进行验证或重新格式化？如果值的格式符合 MySQL 的要求就不需要进一步处理。否则，就需要检验和改写。
- NULL 或空值是如何处理的？它们被允许吗？NULL 值能够被检测出来吗？（有些系统将 NULL 导出为空字符串，以至无法分别 NULL 值和空字符串。）

从 MySQL 中导出数据同上述情况正相反。你可以确认存放在数据库中的数据值内容都合法，但它们可能也需要重新格式化后才能被其他应用程序所使用，或者需要增加列或记录分隔符来构成可以被其他应用程序可识别格式的输出流。

本章主要是从批量传递整个文件的角度来讨论这些问题的，但其中许多技术也可以用在其他情况下。假设有一个基于 Web 的应用程序，它显示一个表单让用户填写，然后根据表单中的内容在数据库中添加新行。这也是一种数据导入情形。Web API 一般会将表单内容表示成已解析好的离散值的集合，所以这个应用程序可能不需要处理记录和列的分隔符。另一方面，验证问题仍然极为重要。你真的无法知道用户送进你代码的数据是什么样子的！所以，重要的事就是：检查它们。本章讨论了验证问题，并且我们还将在第 19.6 节再次讨论它。

文件格式

数据文件格式多种多样，本章讨论最常用的两种：

制表定界格式

这是最简单的文件结构之一：数据值存放在文件的多个行中，在行中由制表符来分隔每个值。一个小一点的制表定界格式文件可能看起来如下所示，列与列之间的空白表示制表符：

a	b	c
a,b,c	d e	f

逗号定界 (CSV) 格式

按 CSV 格式写的文件可能会有某些差别，因为显然没有描述这一格式的现实标准。无论如何，总的思想是用逗号来分隔存放在行内的数据值，而本身内部就包含有逗

号的值需要用引号括起来，以避免被误认为分隔符。将内库包含有空白的值用引号括起来也很常见。这里有个例子，其中每行都包含有 3 个值：

```
a,b,c  
"a,b,c","d e",f
```

CSV 文件比制表符定界的文件更难处理，因为诸如引号和逗号字符有双重含义：它们可以表示文件结构，或者被包含在数据值的内容中。

另一个重要的数据文件特性是行结束序列。最常用的序列是回车、换行和回车/换行对，在文中有时会表示为缩写形式：CR、LF 和 CRLF。

数据文件一般均以包含各列标题的行开头。对于某些导入操作来说标题行有点烦，需要防止标题被当作数据行导入数据库中。在其他情况下，标题行非常有用：

- 当导入已经存在的表时，标题可以在数据文件的列与表中的列顺序不同时帮助你匹配这些列。
- 当从数据文件自动或半自动地建立新表时，标题可以用作列名。例如：第 10.36 节讨论了一个工具，可以检查一个数据文件并猜出用来从该文件来建立表的 CREATE TABLE 语句。如果存在标题行，该工具就使用其中的标题作为列名，否则就使用诸如 c1、c2 等等的通用名字，也就基本起不到说明作用了。

制表定界、换行终止格式

尽管数据文件的格式多种多样，但你不太可能会在自己写的用来处理数据文件的工具中包含所有可能的格式处理逻辑，我也一样。所以，本章中的工具都统一假定输入的数据文件格式为制表定界、换行终止格式。(这也是 MySQL 的 LOAD DATA 语句的默认格式。)通过这一假定，编写读数据文件的程序就方便了。

另一方面，需要有东西能够从其他格式中读入数据。为解决这一问题，我们开发了一个名为 cvt_file.pl 的脚本，它可以读入多种类型的文件（第 10.18 节）。该脚本是基于 Perl 的 Text::CSV_XS 模块，该模块尽管名为 CSV，可以处理的却不止 CSV 格式。cvt_file.pl 可以在许多文件类型之间进行转换，从而让其他要求文件格式是制表定界的程序可以使用本来不支持的格式的文件。换句话说，你可以先用 cvt_file.pl 将文件转成制表定界和换行终止格式，然后所有能够处理制表定界和换行终止格式文件的程序都可以处理它了。

调用命令行命令的说明

本章展示了很多你可以在命令行下调用的命令，在 Unix 环境中你可以使用像 bash 或 tcsh 那样的命令行解释器来完成调用，在 Windows 环境中你可以使用 cmd.exe (“命令行提示”) 来完成调用。这些命令的选项大多都被用引号括起来，有时候一个选项本身就是引号。虽然引号规则在不同的命令行解释器中不尽相同，但下面的命令行解释器大多都支持下述规则（包括 Windows 环境下的 cmd.exe）：

- 对于含有空白的参数，用双引号括起来，以免命令行解释器将其误解为多个被空白分开的参数。命令行解释器在向命令传入该参数之前会将引号自动去除。
- 当需要在参数中包含一个双引号时，在该双引号前缀上一个反斜杠符。

在本章中，有些命令行命令由于长度太长，我们就使用分开多行方式输入，每行末尾使用反斜杠符作为行结束符，来告诉命令行解释器命令没有结束，下行还有内容：

```
% prog_name \
  argument1 \
  argument2 ...
```

不过这种方式只适用于 Unix，对 Windows 不起作用，在 Windows 环境下你只好连续键入命令的全部内容：

```
C:\> prog_name argument1 argument2 ...
```

10.1 使用 LOAD DATA 和 mysqlimport 导入数据

Importing Data with LOAD DATA and mysqlimport

问题

你想使用 MySQL 内置导入功能将一个数据文件中的数据加载到数据库的表中。

解决方案

使用 LOAD DATA 语句或者 mysqlimport 命令行程序。

讨论

MySQL 提供了 LOAD DATA 语句可以充当批量数据加载器。下面例子语句将当前路径下的 mytbl.txt 文件加载到默认数据库中的 mytbl 表中。

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl;
```

在某些 MySQL 安装中，LOCAL 加载选项被禁用。如果你的安装属于这种情况，可在语句中省略 LOCAL 并指定数据文件的完整路径。参见第 10.2 节可了解更多关于本地与非本地加载的知识。

MySQL 另外还提供了一个名为 mysqlimport 的工具程序，用来封装 LOAD DATA 功能，以便你可以直接从命令行加载数据文件。假设 mytbl 是 cookbook 数据库中表，那么与前面 LOAD DATA 语句作用相同的 mysqlimport 命令是：

```
% mysqlimport --local cookbook mytbl.txt
```

同 MySQL 的其他程序一样，你在使用 mysqlimport 时需要指定连接参数选项，如--user、--host 等等（第 1.3 节）。

下面的内容将说明 LOAD DATA 的大体特性和能力，mysqlimport 也共享这些特性和能力。大部分时候 LOAD DATA 同 mysqlimport 的参考说明可以相互通用，个别不一样的地方在我们遇到时会再指出来。

LOAD DATA 提供了许多选项用来解决本章简介中提到的导入问题，例如，用来将输入流分解成记录的行结束序列；用来将记录分解成单个值的列定界符；用来圈住列值的引号；用于列值内部的引号和转义方法以及 NULL 值的表示方法等：

- 默认情况下，LOAD DATA 假定数据文件中包含与需要加载的表一样数目的列，并且列出现的顺序也相同。如果文件中没有包含表中所有列的值，或者值的顺序不对，你可以指定有哪些列和这些列的顺序。如果数据文件包含的列比表中的少，MySQL 会在数据文件中没有对应值的列赋予默认值。
- LOAD DATA 假定数据值是用制表符分开，而行是用换行符（新行）结束的。你可以为数据文件显示指定与此不同的数据格式。
- 你可以指定去除有可能用来括住数据值的引号，以及是什么样的引号。
- 有些特殊转义序列会在导入过程中被识别并转换。默认的转义符是反斜杠 (\)，但是你可以按自己喜好来改变它。\\N 序列是用来表示 NULL 值的。\\b、\\n、\\r、\\t、\\\\ 和 \\0 序列分别是用来表示退格、换行、回车、制表、反斜杠和 ASCII 码的 NUL 值的。（NUL 是零值字节，不同于 SQL 中的 NULL 值。）
- LOAD DATA 还提供诊断信息用于分析是哪些输入值造成了问题。你可以在执行完 LOAD DATA 语句后，发出 SHOW WARNINGS 语句来显示这些信息。

下面将说明如何使用 LOAD DATA 或 mysqlimport 将数据文件导入到 MySQL 的表。

10.2 指定数据文件位置

Specifying the Datafile Location

问题

你不知道如何告诉 `LOAD DATA` 语句到哪去找你的数据文件，尤其是当文件位于其他目录的时候。

解决方案

需要知道决定 MySQL 查找文件位置的规则。

讨论

你加载的数据文件可以位于你发出 `LOAD DATA` 语句所在的客户端主机或服务器主机中。默认情况下，MySQL 服务器假设数据文件位于服务器主机上。不过，有些情况下这并不合适：

- 当你从远程客户端机器上访问 MySQL 数据库，并且没有能力向服务器所在的机器传送文件时（例如，在机器上没有账号），你就没法将数据文件放到服务器上
- 即使你在服务器主机上有账号，还需要你的 MySQL 具有 `FILE` 权限、需要待加载文件任意可读并位于默认数据库的数据目录中。许多 MySQL 用户并不具有 `FILE` 权限（因为 `FILE` 权限也会给他们做出有危害事情的可能），你也有可能不想让文件任意可读（出于安全考虑）或者不放在数据库目录中

幸运的是，你可以通过使用 `LOAD DATA LOCAL` 而不是 `LOAD DATA` 语句来加载位于客户机器上的本地文件。对于加载本地文件的唯一要求就是你自己要能够读出该文件。值得注意的是，`LOCAL` 关键字在默认时可能会被禁用，这个时候你可以尝试使用 `mysql` 的`--local-infile` 选项来打开它。如果这样仍然不行，那可能你的服务器已经被配置成根本不允许使用 `LOAD DATA LOCAL` 了。（本章的例子都假设 `LOCAL` 是可以使用的。如果你的系统实在不行，就需要对这些例子进行适当改动，在相关的语句中省略掉 `LOCAL` 选项，并确保数据文件均位于 MySQL 服务器所在机器上，且遵循下面的规则来指定文件路径名称。例如，使用全路径名。）

如果 `LOAD DATA` 语句中没有 `LOCAL` 选项，MySQL 读取数据文件时将按下列规则在服务器所在机器上定位文件的位置：

- 如果给定的是文件的绝对全路径名（从文件系统的根开始），MySQL 直接读取该文件。
- 如果给定的是文件的相对路径名，就按照路径名的两种不同形式来解释和处理。如果是形如 `mytbl.txt` 之类的单层文件名，MySQL 将在默认数据库的数据目录中查找该文件。（如果当前默认数据库尚未被选定，语句就会失败。）如果是形如 `xyz/mytbl.txt`

之类的多层次文件名，MySQL 将会在数据目录中的同名路径下搜寻，也就是说，在该目录中的 xyz 子目录下寻找 mytbl.txt 文件。

因为数据库目录是服务器数据目录的直接下级，所以，如果数据库名为 cookbook 的话，下面两个语句的效果是相同的：

```
mysql> LOAD DATA INFILE 'mytbl.txt' INTO TABLE mytbl;
mysql> LOAD DATA INFILE 'cookbook/mytbl.txt' INTO TABLE mytbl;
```

如果在 LOAD DATA 语句中指定了 LOCAL 选项，你的客户端程序会在客户端机器上查找文件并发往服务器。客户端程序解释文件路径名的方式同前面一样：

- 绝对路径指定文件从文件系统根开始的完整位置。
- 相对路径指定文件从当前目录开始的位置。

如果你的数据文件是在客户端机器上而你又忘了指定 LOCAL 选项，你将会得到相应的错误提示：

```
mysql> LOAD DATA 'mytbl.txt' INTO TABLE mytbl;
ERROR 1045 (28000): Access denied for user: 'user_name@host_name'
(Using password: YES)
```

提示中的拒绝访问信息容易引起误解：不过，既然你能够连接到服务器并发出 LOAD DATA 语句就说明你已经能够访问 MySQL，对不？所以，这条消息的准确含义是 MySQL 试图打开服务器机器上的 mytbl.txt 文件时被拒绝。

如果你的 MySQL 数据库就位于你发出 LOAD DATA 语句的同一台机器上，“remote”和“local”都指向了同一台机器，即使这样，前述定位数据文件位置的规则仍然有效：没有 LOCAL，服务器读数据文件；有 LOCAL，客户端程序读文件并发给服务器。

mysqldump 在查找数据文件时使用和 LOAD DATA 一样的规则。默认时假定数据文件位于服务器所在机器上。要指明文件位于客户端机器上需要在命令行中使用--local（或-L）选项。

除非你显示地指定数据库名称，LOAD DATA 假定表位于默认数据库中。而 mysqldump 则始终要求提供数据库参数：

```
% mysqldump --local cookbook mytbl.txt
```

使用 LOAD DATA 加载文件到非默认数据库中时，可于表名前面指定数据库名。下面的语句就指明了表 mytbl 位于数据库 other_db 中：

```
mysql> LOAD DATA LOCAL 'mytbl.txt' INTO TABLE other_db.mytbl;
```

LOAD DATA 在数据文件名称和要加载的表名称之间没有约定，而 mysqlimport 则在数据文件名与表名之间进行约定，它使用文件名的最后一部分来确定表名。例如，mysqlimport 将 mytbl、mytbl.txt、mytbl.dat、/tmp/mytbl.txt、/u/paul/data/mytbl.csv 或 C:\projects\mytbl.txt 都对应到表 mytbl 上。

在 Windows 下命名数据文件

Windows 操作系统使用反斜杠 (\) 作为文件名中的路径分隔符号，这可能会造成一些问题，因为 MySQL 也使用反斜杠 (\) 作为字符串内的转义符。所以，指定 Windows 下的路径名时，可使用双反斜杠或者斜杠来代替反斜杠，下面的两条语句就演示了这两种指定 Windows 路径名的方法：

```
mysql> LOAD DATA LOCAL INFILE 'C:\\\\projects\\\\mydata.txt' INTO mytbl;
mysql> LOAD DATA LOCAL INFILE 'C:/projects/mydata.txt' INTO mytbl;
```

如果 SQL 的 NO_BACKSLASH_ESCAPES 模式被打开的话，反斜杠就不再具有特殊意义，你也就不再需要使用双反斜杠了。

```
mysql> SET sql_mode = 'NO_BACKSLASH_ESCAPES';
mysql> LOAD DATA LOCAL INFILE 'C:\\projects\\mydata.txt' INTO mytbl;
```

10.3 指定数据文件的结构

Specifying the Structure of the Datafile

问题

你有一个数据文件但它不是 LOAD DATA 默认格式。

解决方案

使用 FIELDS 和 LINES 子句告诉 LOAD DATA 如何解释数据文件。

讨论

在默认情况下，LOAD DATA 会假定数据文件包含由换行符（新行）终止的多个行，行内是被制表符分开的数据的值。下面的语句没有指定数据文件的格式，所以 MySQL 就会假定其为默认格式：

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl;
```

有两个 LOAD DATA 子句提供数据文件格式定义的显示信息。一个是 FIELDS 子句，定义一行内的域属性，另一个是 LINES 子句，定义了行结束序列。下面的 LOAD DATA 语句指出输入文件包含的数据值由冒号分开，而行则由回车结束：

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl  
>   FIELDS TERMINATED BY ':'  
>   LINES TERMINATED BY '\r';
```

每个子句跟在表名后面。如果两个子句都出现，FIELDS 子句必须位于 LINES 子句前面。行或域终止符可以包含多个字符，例如：\r\n 指明行由回车/换行对终止。

LINES 子句之下还包含了一个 STARTING BY 子句，用来指定每行中间剥除部分内容的序列。类似于 TERMINATED BY，序列可以有多个字符。如果 TERMINATED BY 和 STARTING BY 子句都出现在 LINES 子句中，对它们出现的次序没有限制。

注意：对于 STARTING BY 来说，每一行在给定序列之前的所有内容都会被剥除。例如：你指定 STARTING BY 'x' 而输入行以 abcX 开头时，所有这 4 个字符都会被剥除。

如果你使用的是 mysqlimport，文件格式的指定者是命令选项，同上面两个 LOAD DATA 语句作用一致的 mysqlimport 命令如下：

```
% mysqlimport --local cookbook mytbl.txt  
% mysqlimport --local --fields-terminated-by=":" --lines-terminated-by="\r" \  
cookbook mytbl.txt
```

选项的次序对 mysqlimport 没有影响。

你还可以使用十六进制符号来为 FIELDS 和 LINES 子句指定任意字符作为格式字符，这对加载二进制格式数据文件特别有用。例如：有一个数据文件以 Ctrl-A 分隔域，以 Ctrl-B 结束行，Ctrl-A 和 Ctrl-B 的 ASCII 值是 1 和 2，可用 0x01 和 0x02 来表示，如下：

```
FIELDS TERMINATED BY 0x01 LINES TERMINATED BY 0x02
```

mysqlimport 同样也接受十六进制常量作为格式符号，这在你不想记住如何在命令行下键入转义符号的时候特别有用。制表是 0x09；换行是 0x0a；回车是 0x0d 等等。下面的例子指明了数据文件由制表符界和 CRLF 对终止行：

```
% mysqlimport --local --lines-terminated-by=0x0d0a \  
--fields-terminated-by=0x09 cookbook mytbl.txt
```

当你导入数据文件时，不要假定 LOAD DATA（或者 mysqlimport）知道很多，最重要的是要始终牢记 LOAD DATA 并不知道你的数据文件的格式；始终确保你自己知道文件的格式。如果文件从一台机器传到另一台，内容可能会在你察觉不到的情况下发生细微变化。

许多 LOAD DATA 失败之所以发生，是因为人们指望 MySQL 会知道它根本不可能知道的东西。通过对行与域结束符的默认设置，LOAD DATA 对输入文件的结构做了假定。LOAD DATA

对输入文件的结构做了假定，表现为行和域结束符的默认设置，以及引号和转义字符的设置。假如你的输入不符合这些假定，你就需要告诉 MySQL。

发生疑问时，可以使用十六进制导出程序或其他工具来检查你的数据文件，这些工具可以显示诸如制表符、回车换行符等空白字符的可视化表示。在 Unix 操作系统环境中，类似 `od` 和 `hexdump` 的程序可以用各种格式来显示文件的内容。如果你没有这些工具，`recipes` 发行中的 `transfer` 目录下包含有分别使用 Perl、Ruby 和 Python 开发出来的十六进制工具 (`hexdump.pl`、`hexdump.rb` 和 `hexdump.py`)，以及可以显示文件中所有可打印字符的工具 `see.pl`、`see.rb` 和 `see.py`。你会发现它们在你需要检查文件中究竟包含有什么内容时非常有用。有时你会很惊讶地发现文件的内容跟你想像的不一样，尤其是在文件从一台机器传到另一台机器上的时候：

- 使用 FTP 在不同操作系统之间以文本方式而不是二进制（图像）方式传递文件时会很典型地将行结束改成目标机器上的格式。假定你有一个制表符界和换行结束的数据，文件在 Unix 系统环境中使用 `LOAD DATA` 语句默认设置成功加载进 MySQL 数据库中，那么当你使用 FTP 的文本模式将其拷贝到 Windows 机器上时，里面的换行符会被置换成回车/换行对，由于内容发生了变化，使用同样的 `LOAD DATA` 语句来加载文件时就会出错。那么 MySQL 有没有可能知道这一切呢，没有可能！这就需要你通过增加一个 `LINES TERMINATED BY '\r\n'` 子句来亲自告诉它。在使用不同默认行结束序列的操作系统之间拷贝文件会引起内容变化。
- 当数据文件被贴入电子邮件中时很难不被改动，电子邮件软件可能会打断长一点的行或者转换行结束序列。如果你必须通过电子邮件发送数据文件，最好还是通过附件来传。

10.4 处理引号和特殊字符

Dealing with quotes and special characters

问题

数据文件包含引用值或者转义字符。

解决方案

告知 `LOAD DATA` 当心引用和转义字符，以便它不将数据值载入未解释的数据库中。

讨论

FIELDS 子句可以指定除 TERMINATED BY 之外的其他格式化选项。LOAD DATA 默认情况下会假定值未被引用，并且它会将反斜线 (\) 解释为一个转义字符。要显式地表示值引用字符，使用 ENCLOSED BY；MySQL 在输入处理期间会将该字符从数据值的尾部剥去。要改变默认的转义字符，使用 ESCAPED BY。

FIELDS 子句的三个子句 (ENCLOSED BY、ESCAPED BY 和 TERMINATED BY) 在你指定它们中的多于一个项时，可以以任意顺序出现。例如，这些 FIELDS 字符是等价的：

```
FIELDS TERMINATED BY ',' ENCLOSED BY '''
FIELDS ENCLOSED BY ''' TERMINATED BY ','
```

TERMINATED BY 序列可以由多个字符组成。如果数据值在输入行内被类似于 *@* 之类字符分开，你可以以如下方式表示：

```
FIELDS TERMINATED BY '*@*'
```

要完全禁止转义符处理，可指定一个空的转义序列：

```
FIELDS ESCAPED BY ''
```

当你指定 ENCLOSED BY 以表明哪个引用字符应从数据值中剥离时，也可能在数据值中通过对对其进行成双处理或者在前面加上转义字符来包含引用字符。例如，如果引用和转义字符分别是”和\，输入值”a” “b\"c“被解释为 a”b”c。

对于 mysqlimport，用于指定引用和转义值的相应命令选项是--fields-enclosed-by 和 --fields-escaped-by。(当使用包含引号或反斜线或其他的对于你的命令解释器来说是特殊的字符的 mysqlimport 选项时，记住你可能需要引用或转义引号或转义字符。)

10.5 导入 CSV 文件

Importing CSV Files

问题

你需要载入一个 CSV 格式的文件。

解决方案

在你的 LOAD DATA 语句中加入合适的格式指定子句。

讨论

CSV 格式的数据文件包含以逗号而不是制表符分隔的值，且该值可能以双引号加以引用。

例如，包含以回车换行字符对结尾的行的 CSV 文件 mytbl.txt 可以使用 LOAD DATA 载入 mytbl 中：

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl  
-> FIELDS TERMINATED BY ',' ENCLOSED BY '\"'  
-> LINES TERMINATED BY '\r\n';
```

或者如下使用 mysqlimport：

```
% mysqlimport --local --lines-terminated-by="\r\n" \  
--fields-terminated-by=',' --fields-enclosed-by="" \  
cookbook mytbl.txt
```

10.6 读取不同操作系统的文件

Reading Files from Different Operating Systems

问题

不同的操作系统使用不同的行结束序列。

解决方案

那正是为什么 LOAD DATA 有一个 LINES TERMINATED BY 子句的原因。就用它好了。

讨论

数据文件中实用的行结束字符序列通常是由文件所在的系统决定的。Unix 文件通常用换行符来结束行，这时你可以在 LOAD DATA 语句中以如下方式表示：

```
LINES TERMINATED BY '\n'
```

然而，因为 \n 凑巧是 LOAD DATA 的默认行结束符，所以在这种情况下你不需要指定 LINES TERMINATED BY 子句，除非你想显式地表明行结束字符序列是什么。

如果你的系统不使用 Unix 默认的（换行符），你需要显式地指定行结束符。在 Mac OS X 或 Windows 下创建的文件通常其行结束符分别为回车字符或回车换行字符对。要处理这些不同种类的行结束符，使用相应的 LINES TERMINATED BY 子句：

```
LINES TERMINATED BY '\r'  
LINES TERMINATED BY '\r\n'
```

例如，要载入包含制表符分隔的域且行结束符为回车换行字符对的 Windows 文件，使用这个 LOAD DATA 语句：

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl  
-> LINES TERMINATED BY '\r\n';
```

对应的 mysqlimport 命令是：

```
% mysqlimport --local --lines-terminated-by="\r\n" cookbook mytbl.txt
```

10.7 处理重复的键值

Handling Duplicate Key Values

问题

你的输入中包含一些记录，它们的键值与已有的表行键值一致。

解决方案

让 LOAD DATA 忽略新的记录，或者取代旧的记录。

讨论

默认情况下，如果你试图载入一条记录，而该记录与已有的行的形成 PRIMARY KEY 或 UNIQUE 索引的某列或某些列有重复就会发生错误。要控制此行为，在文件名后指定 IGNORE 或 REPLACE 以告知 MySQL 或者忽略重复行，或者用新值来取代旧的行。

假设你周期性地从不同的监测站接收关于当前天气情况的气象数据，并且你将从这些站点获得的不同类型的测量数据存入如下所示的表中：

```
CREATE TABLE weatherdata
(
    station INT UNSIGNED NOT NULL,
    type     ENUM('precip','temp','cloudiness','humidity','barometer') NOT NULL,
    value    FLOAT,
    PRIMARY KEY (station, type)
);
```

要确保每个站点每种类型的测量数据你仅有 1 行，表包含一个站点 ID 和测量值类型的组合关键字。该表只保存当前的条件，所以当一个给定站点的新的测量数据被装入表中时，他们应该取代该站点之前的测量数据。为实现这点，使用 REPLACE 关键字：

```
mysql> LOAD DATA LOCAL INFILE 'data.txt' REPLACE INTO TABLE weatherdata;
mysqlimport 有与 LOAD DATA 的 IGNORE 和 REPLACE 关键字同样效果的--ignore 和 --replace 选项。
```

10.8 获取关于错误输入数据的诊断信息

Obtaining Diagnostics About Bad Input Data

问题

当你发送一个 LOAD DATA 语句时，你想知道是否有错误的输入值以及它们错在哪儿。

解决方案

使用 LOAD DATA 显示的信息行来确定是否存在有问题的输入值。如果有，使用 SHOW WARNINGS 来查出他们在哪儿以及问题是什么。

讨论

当一条 LOAD DATA 语句完成时，它会返回一行信息告诉你有多少错误和数据转化问题发生。假设你将一个文件载入某个表中并且在 LOAD DATA 完成时看到下面的信息：

```
Records: 134 Deleted: 0 Skipped: 2 Warnings: 13
```

这些值提供了关于导入操作的一些常用信息：

- Records 表明文件中发现的记录数。
- Deleted 和 Skipped 和对于与已有的表行的唯一索引值重复的输入记录的处理有关。 Deleted 表示表中多少行被删除并被输入记录所取代，Skipped 表示相对已有的行的输入记录被忽略。
- Warnings 表示在向列中载入数据值时所出现的问题数目。或者值被正确存入一列，或者没有。在后者中，值在 MySQL 中会有一些不一样的结果，MySQL 将其作为一个 warning。(例如，将一个字符串 abc 存入一个 numeric 列会导致所存储的值为 0。)

通过这些值你能了解到什么？Records 值通常应该和输入文件中的行数一致。如果它和文件的行数不同，那说明 MySQL 以一种不同于它应有的格式来解释该文件。在这种情况下，你还可能看见一个较高的 Warnings 值，这意味着很多值不得不进行转化因为他们与所期望的数据类型不匹配。（此问题的解决方案常是指定合适的 FIELDS 和 LINES 子句。）

假定你的 FIELDS 和 LINES 格式指示符正确，非零的 Warnings 数目表示出现了错误的输入值。从 LOAD DATA 信息行中的数字中你不能获悉哪些输入记录有问题或者哪一列出错了。要获取信息，可发起 SHOW WARNINGS 语句。

假设表 t 结构如下：

```
CREATE TABLE t
(
    i INT,
    c CHAR(3),
    d DATE
);
```

并假设数据文件 data.txt 如下所示：

```
1           1           1
abc         abc         abc
2010-10-10  2010-10-10  2010-10-10
```

将文件载入表中会使一个数字，一个字符串和一个日期被装入三个列中的每一个。这么做会导致许多数据转换和警告：

```
mysql> LOAD DATA LOCAL INFILE 'data.txt' INTO TABLE t;
Query OK, 3 rows affected, 5 warnings (0.01 sec)
Records: 3 Deleted: 0 Skipped: 0 Warnings: 5
```

要查看警告信息，可在 LOAD DATA 语句后立即使用 SHOW MESSAGES：

```
mysql> SHOW WARNINGS;
+-----+-----+
| Level | Code | Message
+-----+-----+
| Warning | 1265 | Data truncated for column 'd' at row 1
| Warning | 1264 | Out of range value for column 'i' at row 2
| Warning | 1265 | Data truncated for column 'd' at row 2
| Warning | 1265 | Data truncated for column 'i' at row 3
| Warning | 1265 | Data truncated for column 'c' at row 3
+-----+-----+
5 rows in set (0.00 sec)
```

SHOW MESSAGES 输出帮你确定哪些值被转化以及原因所在。结果表如下所示：

```
mysql> SELECT * FROM t;
+---+---+---+
| i | c | d
+---+---+---+
| 1 | abc | 0000-00-00
| 0 | abc | 0000-00-00
| 2010 | 201 | 2010-10-10
+---+---+---+
```

10.9 跳过数据文件行

Skiping Datafile Lines

问题

你希望 LOAD DATA 在开始载入记录之前跳过你数据文件的第一行或头部的数行。

解决方案

告诉 LOAD DATA 要忽略多少行。

讨论

要跳过数据文件的起始 n 行，在 LOAD DATA 语句中加入一个 IGNORE n LINES 子句。例如，如果一个制表符定界的文件以由列头部组成的行开始，按如下方式跳过该行：

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl IGNORE 1 LINES;
```

mysqldump 支持有同样效果的--ignore-lines= n 选项。

IGNORE 对于由外部源所生成的文件比较有用。例如，某个程序可能以 CSV 格式起始行为列表前的形式导出数据。以下语句对于跳过以回车换行结束的该文件的标签是适用的：

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl  
-> FIELDS TERMINATED BY ',' ENCLOSED BY ''  
-> LINES TERMINATED BY '\r'  
-> IGNORE 1 LINES;
```

10.10 指定输入列顺序

Specifying Input Column Order

问题

你数据文件的列与你要载入文件的表的列顺序不同。

解决方案

通过表明数据文件列对应于哪些表列来告知 LOAD DATA 如何在表和文件间建立匹配关系。

讨论

LOAD DATA 假设数据文件中的列和表中的列顺序相同。如果不是这样，指定一个列表来表示数据文件列应该被载入到哪些表列。假设你的表有列 a、b 和 c，但是数据文件中的列对应于列 b、c 和 a。你可以如下载入文件：

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl (b, c, a);
```

相应的 mysqldump 语句使用--columns 选项指定列列表：

```
% mysqldump --local --columns=b,c,a cookbook mytbl.txt
```

10.11 在插入输入值之前对数据文件进行预处理

Preprocessing Input Values Before Inserting Them

问题

你数据文件中的值并非适于载入表中的格式。例如，值的单位有错，或者两个输入域应被组合到一起插入到某一个列中。

解决方案

LOAD DATA 可以在插入输入值之前对数据文件进行有限的预处理。这可以使你在将输入数据载入你的表中之前将其匹配到合适的值。

讨论

第 10.10 节说明了如何为 LOAD DATA 指定一个列列表来表示输入域如何对应于表列。列列表还可以命名用户自定义的变量，这样对于每个输入记录，输入域被赋给变量。接着你可以对那些变量进行计算，然后将结果插入到表中。这些计算由在 SET 子句中用逗号分开的一个或多个 `col_name = expr` 赋值表达式来进行指定。

假设你的数据文件包含如下列，且第一行为列标签：

Date	Time	Name	Weight	State
2006-09-01	12:00:00	Bill Wills	200	Nevada
2006-09-02	09:00:00	Jeff Deft	150	Oklahoma
2006-09-04	03:00:00	Bob Hobbs	225	Utah
2006-09-07	08:00:00	Hank Banks	175	Texas

还假设该文件必须被载入如下的表中：

```
CREATE TABLE t
(
    dt      DATETIME,
    last_name CHAR(10),
    first_name CHAR(10),
    weight_kg FLOAT,
    st_abbrv CHAR(2)
);
```

在数据文件域和表列之间存在几个不匹配之处，必须对其进行调整以使其能导入文件：

- 文件包含独立的日期和时间列，必须组合成日期和时间值以便插入 DATETIME 列中。

- 文件包含一个姓名域，但必须分成独立的姓和名值以便插入到 `first_name` 和 `last_name` 列。
- 文件包含以磅为单位的重量，但必须被转化为公斤以便插入到 `weight_kg` 列。（转化因子是 1 磅等于 0.454kg。）
- 文件包含州名，但表中包含的 2-字母的缩写。可以通过在 `states` 表中执行一次查找操作将名称匹配到缩写。

要处理这些转化，应将每个输入列赋给一个用户自定义的变量，然后编写一个 `SET` 子句来执行计算。（记住跳过包含列标签的第一行。）

```
mysql> LOAD DATA LOCAL INFILE 'data.txt' INTO TABLE t
-> IGNORE 1 LINES
-> (@date,@time,@name,@weight_lb,@state)
-> SET dt = CONCAT(@date, ' ', @time),
->     first_name = SUBSTRING_INDEX(@name, ' ', 1),
->     last_name = SUBSTRING_INDEX(@name, ' ', -1),
->     weight_kg = @weight_lb * .454,
->     st_abbrev = (SELECT abbrev FROM states WHERE name = @state);
```

作为导入操作的结果，表包含这些行：

```
mysql> SELECT * FROM t;
+-----+-----+-----+-----+-----+
| dt      | last_name | first_name | weight_kg | st_abbrev |
+-----+-----+-----+-----+-----+
| 2006-09-01 12:00:00 | Wills    | Bill       | 90.8     | NV        |
| 2006-09-02 09:00:00 | Deft      | Jeff       | 68.1     | OK        |
| 2006-09-04 03:00:00 | Hobbs    | Bob        | 102.15   | UT        |
| 2006-09-07 08:00:00 | Banks    | Hank      | 79.45    | TX        |
+-----+-----+-----+-----+-----+
```

如本节所示，`LOAD DATA` 可以执行数据值重新格式化。本章后面的部分将包含使用此功能的其他示例。例如，第 10.33 节用它来执行数据导入期间非 ISO 日期到 ISO 格式的转换。然而，尽管 `LOAD DATA` 可以将输入值匹配到其他值，它不能立刻拒绝被发现包含不适合的值的输入记录。要做到这点，你可以对输入文件进行预处理来移除这些记录，或者在载入文件后发起一个 `DELETE` 语句。

10.12 忽略数据文件列

Ignoring Data File Columns

问题

数据文件中包含应该忽略不计不用载入表中的列。

解决方案

如果列位于输入行的末尾，那并不是问题。否则，你可以在 LOAD DATA 中使用一个列列表将应被忽略的列赋给一个用户自定义的假变量。

讨论

出现在输入行尾部的多余列易于处理。如果一行比表包含更多的列，LOAD DATA 就会忽略它们（虽然这可能产生一个非零的警告数目）。

跳过行中间的列有些棘手。假设你想从一个 Unix 密码文件/etc/passwd 中载入信息，其中包含如下格式的行：

```
account:password:UID:GID:GECOS:directory:shell
```

还假设你不想载入 password 列。包含其他列信息的表结构如下所示：

```
CREATE TABLE passwd
(
    account    CHAR(8),      # login name
    uid        INT,          # user ID
    gid        INT,          # group ID
    gecos      CHAR(60),     # name, phone, office, etc.
    directory  CHAR(60),     # home directory
    shell      CHAR(60)      # command interpreter
);
```

要载入文件，我们需要指定列分隔符是冒号，这可以轻松地使用一个 FIELDS 子句来处理：

```
FIELDS TERMINATED BY ':'
```

然而，我们还必须告诉 LOAD DATA 跳过包含密码的第二个域。要做到这点，须在语句中添加一个列列表。列表应该包括要被载入到表中的每个列的名称，以及用于要被忽略的任意列的用户自定义的假变量：

```
mysql> LOAD DATA LOCAL INFILE '/etc/passwd' INTO TABLE passwd
-> FIELDS TERMINATED BY ':'
-> (account,@dummy,uid,gid,gecos,directory,shell);
```

相应的 mysqlimport 命令应该包括一个--columns 选项：

```
% mysqlimport --local \
--columns="account,@dummy,uid,gid,gecos,directory,shell" \
--fields-terminated-by= ":" cookbook /etc/passwd
```

参考

忽略列的另一个方法是对输入文件进行预处理来移除列。第 10.19 节讨论了一个能以任意顺序取出并显示数据文件列的工具。

10.13 从 MySQL 中导出查询结果

Exporting Query Results from MySQL

问题

你想将 MySQL 的查询结果导出到一个文件或另一个程序。

解决方案

使用 SELECT...INTO OUTFILE 语句，或者重定向 mysql 程序的输出。

讨论

MySQL 提供了一个 SELECT...INTO OUTFILE 语句用于将查询结果直接导出到服务器主机上的一个文件中。如果你想在客户端主机上获取结果，导出查询的另一种方法是重定向 mysql 程序的输出。这些方法的优缺点各不相同，所以你应该对它们都有所了解，并在给定场景下应用最合适的一个。

用 SELECT...INTO OUTFILE 语句导出

这条语句的语法将一个正常的 SELECT 语句和 INTO OUTFILE *filename* 组合在一起。默认的输出格式和 LOAD DATA 一致，所以下面的语句将 passwd 表导出到制表符分隔，换行符终结的文件/tmp/passwd.txt 中：

```
mysql> SELECT * FROM passwd INTO OUTFILE '/tmp/passwd.txt';
```

你可以使用类似于 LOAD DATA 中所用的选项来改变输出格式，这些选项说明了如何引用和定界列和记录。例如，要以具有 CRLF 终结的行的 CSV 格式导出 passwd 表，使用这条语句：

```
mysql> SELECT * FROM passwd INTO OUTFILE '/tmp/passwd.txt'
      -> FIELDS TERMINATED BY ',' ENCLOSED BY '"'
      -> LINES TERMINATED BY '\r\n';
```

SELECT ... INTO OUTFILE 有以下属性：

- 输出文件直接由 MySQL 服务器创建，所以文件名应该表明你希望文件在服务器主机上被写入的位置。文件的位置使用与第 10.2 节所描述的去掉 LOCAL 的 LOAD DATA 语句一样的规则。本语句没有类似于 LOAD DATA 的 LOCAL 版本的 LOCAL 版本。

- 你必须拥有 MySQL FILE 权限来执行 SELECT...INTO 语句。
- 输出文件一定不能已经存在。（这有效地防止了 MySQL 可能覆盖重要的文件。）
- 你应该拥有服务器主机上的登录账户或者有某种方法来存取该主机上的文件。如果你不能获取输出文件，SELECT...INTO OUTFILE 将对你毫无价值。
- 在 Unix 下，文件被创建为全局可读，且为执行 MySQL 服务器的账户拥有。这意味着尽管你可能能读取文件，但你或许不能删除它。

使用 mysql 客户端来导出数据

因为 SELECT...INTO OUTFILE 将数据文件写入服务器主机上，除非你的 MySQL 账户有 FILE 权限，否则你不能使用它。要将数据导出到一个本地文件中，你必须使用某些其他策略。如果你所需要的是一个制表符定界的输出，你可以通过用 mysql 程序来执行一个 SELECT 语句并将输出重定向到一个文件来做一个“穷人的导出”。用这个方法你可以无须 FILE 权限将查询结果写入你本地主机的某个文件中。下面的示例导出本章之前所创建的 passwd 表中的登录名和命令解释器列：

```
% mysql -e "SELECT account, shell FROM passwd" --skip-column-names \
cookbook > shells.txt
```

-e 选项指定要执行的语句，--skip-column-names 告诉 MySQL 不要写入通常位于语句输出前部的列名行（见第 1.14 节和第 1.21 节）。

注意 MySQL 将 NULL 值写为字符串“NULL”。根据你想用输出文件来做什么，可能要对其做些后期处理来进行转化。

通过将输出结果送到一个后期处理过滤器（该过滤器将制表符转化为其他字符），可以生成非制表符定界的其他格式的输出。例如，要用#字符来作为分界符，将所有的制表符转化为#字符（TAB 表示你在命令中输入一个制表符的位置）：

```
% mysql --skip-column-names -e " your statement here " db_name \
| sed -e "s/ TAB /#/g" > output_file
```

你也可以将 tr 用于此目的，虽然此工具的不同实现语法可能有所变化。对于 Mac OS X 或 Linux，命令如下所示：

```
% mysql --skip-column-names -e " your statement here " db_name \
| tr "\t" "#" > output_file
```

上面所示的 mysql 命令使用--skip-column-names 来防止列标签显示在输出中。在某些情况下，包含标签会比较有用（例如，当以后要导入文件时它们就会有用）。如果是这样，从命令中去掉--skip-column-names 选项。在此方面，用 mysql 来导出查询结果比 SELECT...INTO OUTFILE 更为灵活，因为后者不能生成包含列标签的输出。

参考

另一个将查询结果导出到客户端主机上的某个文件的方法是使用第 10.17 节所描述的 mysql_to_text.pl 工具，该程序存在供你显式指定输出格式的选项。但要将查询结果导出为一个 Excel 表，参考第 10.38 节。

10.14 将表导出为文本文件

Exporting Tables as Text Files

问题

你想将整个表导出到一个文件中。

解决方案

使用带有--tab 选项的 mysqldump 程序。

讨论

mysqldump 程序用于拷贝或备份表和数据库。它能够将表输出写成一个文本数据文件，或者一个用于重建表行的 INSERT 语句集。前者将在这里进行描述，或者见第 10.15 节和第 10.16 节。

为将表导出为一个数据文件，你必须指定一个--tab 选项，它表示你 MySQL 服务器主机上希望服务器写入文件的目录（目录必须已经存在；服务器不会自己创建它）。例如，要将 cookbook 数据库中的 states 表导出到/tmp 目录中的一个文件中，使用如下的命令：

```
% mysqldump --no-create-info --tab=/tmp cookbook states
```

Mysqldump 使用表名加上一个.txt 后缀来创建一个数据文件，所以此命令写入了一个名为 /tmp/states.txt 的文件。这种形式的 mysqldump 在某些方面等价于 SELECT...INTO OUTFILE。例如，它将表写入服务器主机上的一个数据文件中，并且你必须拥有 FILE 权限来使用它。参考第 10.13 节可以查阅 SELECT...INTO OUTFILE 的通用属性列表。

如果你去掉--no-create-info 选项，mysqldump 也在你的本地主机上创建一个包含 CREATE TABLE 语句的/tmp/states.sql 文件。（后一个文件由你自身拥有，不同于数据文件由服务器拥有。）

你可以在数据库名称后命名多个表，在这种情况下 mysqldump 会将每个表都写入输出文件。如果你不命名任何表，mysqldump 会为数据库中的每个表都生成输出。

mysqldump 默认会创建格式为制表符定界，换行符结束的数据文件。要控制输出格式，使用--fields-enclosed-by、--fields-terminated-by 和--lines-terminated-by 选项（也就是，和 mysqlimport 的格式化描述符一样的选项）。例如，要以具有 CRLF 行结束符的 CSV 格式来输出 states 表，使用此命令：

```
% mysqldump --no-create-info --tab=/tmp \
  --fields-enclosed-by="" --fields-terminated-by="," \
  --lines-terminated-by="\r\n" cookbook states
```

使用这种方式导出的数据文件可以使用 LOAD DATA 或 mysqlimport 导入。如果你没有使用默认格式来导出表，请确保在导入时使用相匹配的格式化描述符。

10.15 以 SQL 格式导出表内容或定义

Exporting Table Contents or Definitions in SQL Format

问题

你希望以 SQL 语句形式导出表或数据库以方便以后导入。

解决方案

使用不带--tab 选项的 mysqldump 程序。

讨论

正如第 10.14 节所讨论的，mysqldump 用--tab 选项来调用时会引起 MySQL 服务器在服务器主机上将表写入文本数据文件中。如果你略去--tab，服务器会将表行格式化为 INSERT 语句形式，并将它们返回到 mysqldump 中，然后 mysqldump 将输出写到客户端主机上。输出还能包含每个表的 CREATE TABLE 语句。这提供了你可以很方便地在文件中找到输出并在以后用它来重新创建数据表。使用这些导出文件作为备份或用于将表拷贝到另一个 MySQL 服务器是经常性的操作。本节讨论如何将导出的输出保存在一个文件中；第 10.16 节说明如何通过网络将其直接发送到另一个服务器。

要以 SQL 格式将表导出到一个文件中，使用如下命令：

```
% mysqldump cookbook states > states.txt
```

它创建了包含 CREATE TABLE 语句和一系列 INSERT 语句的输出文件 states.txt。

```
-- MySQL dump 10.10
--
-- Host: localhost      Database: cookbook
-- Server version      5.0.27-log
-
-- `states` 表的表结构
--

CREATE TABLE `states` (
  `name` varchar(30) NOT NULL,
  `abbrev` char(2) NOT NULL,
  `statehood` date default NULL,
  `pop` bigint(20) default NULL,
  PRIMARY KEY (`abbrev`)
);

--
-- 导出表 `states` 的数据
--

INSERT INTO `states` VALUES ('Alabama','AL','1819-12-14',4530182);
INSERT INTO `states` VALUES ('Alaska','AK','1959-01-03',655435);
INSERT INTO `states` VALUES ('Arizona','AZ','1912-02-14',5743834);
INSERT INTO `states` VALUES ('Arkansas','AR','1836-06-15',2752629);
INSERT INTO `states` VALUES ('California','CA','1850-09-09',35893799);
INSERT INTO `states` VALUES ('Colorado','CO','1876-08-01',4601403);
...

```



提示: 前面的 mysqldump 输出实际上是使用--skip-extended-insert 选项生成的，它会使每行都被写成一个独立的 INSERT 语句。如果你去掉--skip-extended-insert (这是一般性情形)，mysqldump 会写入多行 INSERT 语句。这对于你我来说更难以阅读，但对于 MySQL 服务器来说处理起来可以更高效。

要导出多个表，在数据库名参数后附上它们所有的名称。要导出整个数据库，在数据库后不要命名任何表名。这条语句会导出 cookbook 数据库中的所有表：

```
% mysqldump cookbook > cookbook.txt
```

如果你想导出所有数据库中的所有表，以如下方式调用 mysqldump：

```
% mysqldump --all-databases > dump.txt
```

在这种情况下，输出文件还会在合适位置包括 CREATE DATABASE 和 USE db_name 语句以便当你以后读取文件时，每个表被创建在相应的数据库中。

还有一些选项可用于控制输出格式：

```
--no-create-info
```

压缩 CREATE TABLE 语句。当你仅想导出表内容时使用此选项。

```
--no-data
```

压缩 INSERT 语句。当你仅想导出表定义时使用此选项。

```
--add-drop-table
```

为每个 CREATE TABLE 语句生成一个 DROP TABLE 语句。这对于生成一个你以后可以用于从头重建表的文件来说是很有用的。

```
--no-create-db
```

压缩--all-databases 选项通常所生成的 CREATE DATABASE 语句。

假设现在你已经使用 mysqldump 创建了一个 SQL 格式导出文件。你怎么将文件导回 MySQL 中？在这里一个常犯的错误是使用 mysqlimport。毕竟，从逻辑上来说可以假定如果 mysqldump 导出表，mysqlimport 一定可以导入他们。对吗？对不起，答案是否定的。逻辑上可行，但并非总是正确的。如果你使用带有--tab 选项的 mysqldump，你可以用 mysqlimport 来导入最终的数据文件。但如果你导出了一个 SQL 格式文件，mysqlimport 就不能正确处理它了。这时应换用 mysql 程序。你这么做的方式依赖于导出的文件中有什么。如果你使用--all-databases 导出了多个数据库，文件将包含合适的 USE db_name 语句用以选择每个表所属的数据库，在命令行中你不需要数据库参数：

```
% mysql < dump.txt
```

如果你从单个数据库中导出表，你需要告诉 mysql 要将它们导入到哪个数据库中：

```
% mysql db_name < cookbook.txt
```

注意，使用第二个导入命令时，可能会将表导入到不同于其来源的一个数据库中。例如，你可以使用此情况在一个 test 数据库中创建一个表或一些表的拷贝以帮助调试数据操作语句，而无需担心影响原始表。

10.16 将表或数据库拷贝到另一个服务器

Copying Tables or Databases to Another Server

问题

你想将表或数据库从一个 MySQL 服务器拷贝到另一个。

解决方案

同时使用 mysqldump 和 mysql，通过管道进行连接。

讨论

mysqldump 的 SQL 格式输出能用于将表或数据库从一个服务器拷贝到另一个。假设你想将本地主机上 `cookbook` 数据库中的 `states` 表拷贝到 `other-host.example.com` 主机上的 `cb` 数据库中。实现此功能的一个方法是将输出导出到一个文件中（如第 10.15 节所述）：

```
% mysqldump cookbook states > states.txt
```

现在将 `states.txt` 拷贝到 `other-host.example.com`，并在该处执行下面的命令将表导入到 MySQL 服务器的 `cb` 数据库中：

```
% mysql cb < states.txt
```

要不用中间文件来实现此功能，将 mysqldump 的输出直接通过网络发送到远程 MySQL 服务器。如果你可以从你的本地机器上连接到服务器，使用如下命令：

```
% mysqldump cookbook states | mysql -h other-host.example.com cb
```

命令的 mysqldump 部分连接到本地服务器并将输出写入到管道中。命令的 mysql 部分连接到 `other-host.example.com` 上的远程 MySQL 服务器。它读取管道以作为输入并将每条语句发送到 `other-host.example.com` 服务器上。

如果你不能使用 mysql 从你的本地主机上直接连接到远程服务器，将输出发送到管道中，使用 ssh 远程调用 `other-host.example.com` 上的 mysql：

```
% mysqldump cookbook states | ssh other-host.example.com mysql cb
```

ssh 连接到 `other-host.example.com` 并调用 mysql。它接着从管道中读取 mysqldump 输出，并将其传给远程 mysql 进程。ssh 在有些时候是很有用的，譬如当你想通过网络将输出发送到另一台机器，但 MySQL 端口却被防火墙禁止，而允许 SSH 端口上的连接时。

要通过网络拷贝多个表，在 mysqldump 命令的数据库参数后列出它们所有的名称。要拷贝整个数据库，在数据库名称后不要指定任何表名。mysqldump 将导出数据库中包含的所有表。

如果你正考虑用 `--all-databases` 选项调用 mysqldump 将你所有的数据库发送到另一个服务器，要考虑到输出将会包含 mysql 数据库的表，其中就有授权表。如果远程服务器有不同的用户组成，你可能不想替换服务器的授权表！

10.17 编写你自己的导出程序

Writing Your Own Export Programs

问题

MySQL 内建的导出功能并不足够。

解决方案

编写你自己的工具。

讨论

当现有的导出软件不能完成你所想要的功能时，你可以编写你自己的程序。本节描述一个 Perl 脚本，`mysql_to_text.pl`，它执行任意一条语句并以你所指定的格式导出。它将输出写到客户端主机上，且能包含一行列标签（这是 `SELECT ... INTO OUTFILE` 所做不到的两点）。它能比使用具有后期处理器的 `mysql` 更方便地生成多种输出格式，并且它会将输出写到客户端主机，不同于 `mysqldump`，仅能将 SQL 格式的输出写到客户端。你可以在 `recipes` 发行包的 `transfer` 目录中找到 `mysql_to_text.pl`。

`mysql_to_text.pl` 基于 `Text::CSV_XS` 模块，如果在你的系统上没有安装，你需要先获取它。一旦已安装好，你可以如下阅读它的文档：

```
% perldoc Text::CSV_XS
```

这个模块非常方便，因为它将查询输出转化成相对较普通的 CSV 格式。你所做的就是提供一个列值数组，然后模块会将它们打包成为一个适当格式化的输出行。这就使得将查询输出转换为 CSV 格式有些无关紧要。但是使用 `Text::CSV_XS` 的实际好处在于它是可配置的；你可以告诉它使用那种定界符和引用字符。这就意味着虽然模块默认生成 CSV 格式，但你可以对其进行配置使它输出多种格式。例如，如果你将定界符设为制表符，引用字符设为 `undef`，`Text::CSV_XS` 会生成制表符定界的输出。我们将在本节中利用这种灵活性来编写 `mysql_to_text.pl`，以及在第 10.18 节中编写用于转化文件格式的文件处理工具。

`mysql_to_text.pl` 接收几种命令行选项。它们中的一些用于指定 MySQL 连接参数（诸如 `--user`、`--password` 和 `--host`）。你已经很熟悉这些了，因为它们被诸如 `mysql` 等标准 MySQL 客户端所使用。如果你在选项文件中指定一个 `[client]` 组，脚本还能从其中获取连接参数。`mysql_to_text.pl` 还接收以下选项：

```
--execute = query, -e query
```

执行查询并导出其输出。

```
--table = tbl_name, -t tbl_name
```

导出所命名表的内容。这等价于使用--execute 来指定 SELECT * FROM *tbl_name* 的查询值。

```
--labels
```

在输出中包含一行初始的列标签。

```
--delim = str
```

将列定界符设为 *str*。选项值可由一个或多个字符组成。默认使用制表符。

```
--quote = c
```

将列值引用字符设为 *c*。默认不引用任何事物。

```
--eol = str
```

将行结束符设为 *str*。选项值可由一个或多个字符组成。默认使用换行符。

--delim、--quote 和--eol 选项的默认值与 LOAD DATA 和 SELECT ... INTO OUTFILE 所使用的一致。

命令行的最后一个参数应该是数据库名称，除非语句中已经内含。例如，下面两个命令是等价的；每个都将 cookbook 数据库的 passwd 表导出成以冒号分隔的格式：

```
% mysql_to_text.pl --delim=":" --table=passwd cookbook  
% mysql_to_text.pl --delim=":" --table=cookbook.passwd
```

要生成以 CRLF 行结束符的 CSV 格式，使用如下命令：

```
% mysql_to_text.pl --delim="," --quote="" --eol="\r\n" \  
--table=cookbook.passwd
```

以上是如何使用 mysql_to_text.pl 的一般性描述。现在让我们来讨论一下它是如何工作的。mysql_to_text.pl 脚本的开头部分声明一些变量，然后使用第 2.8 节所开发的选项处理技术处理命令行参数。随着其发生，脚本中的大多数代码都用于处理命令行参数并进行设置以执行查询。其中很少的篇幅包括与 MySQL 的交互。

```
#!/usr/bin/perl  
# mysql_to_text.pl - 以用户指定的文本格式来导出 MySQL 查询输出  
  
# 用法: mysql_to_text.pl [ 选项 ] [ 数据库名 ] > text_file  
  
use strict;  
use warnings;  
use DBI;  
use Text::CSV_XS;  
use Getopt::Long;  
$Getopt::Long::ignorecase = 0; # 选项是大小写敏感的  
$Getopt::Long::bundling = 1; # 允许绑定短选项  
  
# ...构建用法信息变量$usage (未显示) ...
```

```
# 命令行选项变量 - 除控制输出结构的选项外所有的初始都被设为未定义值,  
# 而控制输出结构的选项被设为制表符定界的, 换行符结束。  
my $help;  
my ($host_name, $password, $port_num, $socket_name, $user_name, $db_name);  
my ($stmt, $tbl_name);  
my $labels;  
my $delim = "\t";  
my $quote;  
my $eol = "\n";  
  
GetOptions (  
    # =i 意指在选项后必须带有一个整型参数  
    # =s 意指选项后必须带有一个字符串值  
    "help"          => \$help,           # 输出帮助信息  
    "host|h=s"      => \$host_name,       # 服务器主机  
    "password|p=s"   => \$password,        # 密码  
    "port|P=i"       => \$port_num,        # 端口号  
    "socket|S=s"     => \$socket_name,      # 套接字名称  
    "user|u=s"       => \$user_name,        # 用户名  
    "execute|e=s"    => \$stmt,            # 要执行的语句  
    "table|t=s"      => \$tbl_name,         # 所要导出的表  
    "labels|l"        => \$labels,           # 生成列标签行  
    "delim=s"        => \$delim,            # 列定界符  
    "quote=s"        => \$quote,             # 列引用字符  
    "eol=s"          => \$eol                # 行结束(记录)分隔符  
) or die "$usage\n";  
  
die "$usage\n" if defined $help;  
  
$db_name = shift (@ARGV) if @ARGV;  
  
#--execute 或--table 之一必须被指定, 但一定不要同时指定两者  
die "You must specify a query or a table name\n\n$usage\n"  
    if !defined ($stmt) && !defined ($tbl_name);  
die "You cannot specify both a query and a table name\n\n$usage\n"  
    if defined ($stmt) && defined ($tbl_name);  
  
# 如果给定表名, 用它来创建选择整个表的查询  
$stmt = "SELECT * FROM $tbl_name" if defined ($tbl_name);  
  
# 解释文件结构选项中的特殊字符  
$quote = interpret_option ($quote);  
$delim = interpret_option ($delim);  
$eol = interpret_option ($eol);
```

interpret_option() 函数 (未显示) 处理--delim、--quote 和--eol 选项的退格符和十六进制序列。它将\n、\r、\t 和\n0 解释为换行, 回车, 制表符和 ASCII NUL 字符。它还解释以 0xnn 形式给定的十六进制值 (例如, 0x0d 表示回车)。

在处理完命令行选项后, mysql_to_text.pl 构建数据源名称 (DSN) 并连接到服务器:

```

my $dsn = "DBI:mysql:";
$dsn .= ";database=$db_name" if $db_name;
$dsn .= ";host=$host_name" if $host_name;
$dsn .= ";port=$port_num" if $port_num;
$dsn .= ";mysql_socket=$socket_name" if $socket_name;
# 从标准选项文件中读取[client]组参数
$dsn .= ";mysql_read_default_group=client";

my %conn_attrs = (PrintError => 0, RaiseError => 1, AutoCommit => 1);
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs);

```

数据库名称来自于命令行。连接参数可来源于命令行或选项文件。(第 2.8 节中涵盖了 MySQL 选项文件的用法。)

建立了 MySQL 连接后，脚本准备执行查询并生成一些输出。这时 Text::CSV_XS 模块就能派上用场了。首先，通过调用 new() 来创建一个 CSV 对象，它接受一个可选的控制对象如何处理数据行的选项哈希表。脚本准备并执行查询，输出一行列标签（如果用户指定了--labels 选项），并输出结果集的行：

```

my $csv = Text::CSV_XS->new ({
    sep_char => $delim,
    quote_char => $quote,
    escape_char => $quote,
    eol => $eol,
    binary => 1
});

my $sth = $dbh->prepare ($stmt);
$sth->execute ();
if ($labels)          # 输出列标签行
{
    $csv->combine (@{$sth->{NAME}}) or die "cannot process column labels\n";
    print $csv->string ();
}

my $count = 0;
while (my @val = $sth->fetchrow_array ())
{
    ++$count;
    $csv->combine (@val) or die "cannot process column values, row $count\n";
    print $csv->string ();
}

```

name() 调用中的 sep_char 和 quote_char 选项设置列定界符和引用字符。escape_char 选项被设为与 quote_char 一样的值，以便出现在数据值中的引用字符实例在输出中成对出现。eol 选项表示行终结序列。通常，Text::CSV_XS 将它留给你用于输出行的终结符。

通过将一个非 `undef` 的 `eol` 值传给 `new()`，模块自动将该值加到每个输出行中。二进制选项对于处理包含二进制字符的数据值来说很有用。

调用 `execute()` 后，可用 `$sth->{NAME}` 来引用列标签。每行输出都使用 `combine()` 和 `string()` 来生成。`combine()` 方法接收一个值数组，并将其转化为合理格式化的字符串。`string()` 返回字符串，所以我们可以将其输出。

10.18 将数据文件从一种格式转化为另一种格式

Converting Datafiles from One Format to Another

问题

你想将文件转化为一种不同的格式，以使其便于使用，或者使另一个程序能够理解它。

解决方案

使用本节所描述的 `cvt_file.pl` 转化器脚本。

讨论

第 10.17 节所讨论的 `mysql_to_text.pl` 脚本使用 MySQL 作为一个数据源，并生成你通过 `--delim`、`--quote` 和 `--eol` 选项所指定的格式的输出。本节描述 `cvt_file.pl`，一个提供类似格式选项的工具，但该选项对于输入和输出两者皆适用。它从文件而不是 MySQL 中读取数据，并将其从一种格式转化为另一种。例如，为读取一个制表符定界的文件 `data.txt`，将其转化为以冒号定界的格式，并将结果写到 `tmp.txt` 中，以如下方式调用 `cvt_file.pl`：

```
% cvt_file.pl --idelim="\t" --odelim=":" data.txt > tmp.txt
```

`cvt_file.pl` 脚本对于输入和输出有不同的选项。这样一来，`mysql_to_text.pl` 仅有一个 `--delim` 选项用于指定列定界符，而 `cvt_file.pl` 有 `--idelim` 和 `--odelim` 选项分别用于设置输入和输出行列定界符。但作为一个快捷方式，`--delim` 也提供了支持；它可以为输入和输出两者设置定界符。`cvt_file.pl` 所能理解的完整的选项集如下所示：

`--idelim = str, --odelim = str, --delim = str`

为输入、输出或两者设置列定界符。选项值可由一个或多个字符组成。

`--iquote = c, --oquote = c, --quote = c`

为输入、输出或两者设置列引用字符。

`--ieol = str, --oeol = str, --eol = str`

为输入、输出或两者设置行结束序列。选项值可由一个或多个字符组成。

```
--iformat = format, --oformat = format, --format = format,
```

指定输入格式，输出格式，或两者都指定。这个选项是设置引用字符和定界符值的快速方式。例如，`--iformat=csv` 设置输入引用字符和定界符为双引号和逗号。`--iformat=tab` 将它们设为“无引号”和制表符。

```
--ilabels, --olabels, --labels
```

期望开始有一行列标签作为输入，输出起始行标签，或两者皆具。如果你请求标签作为输出，但并不从输入中读取标签，`cvt_files.pl` 使用 `c1`、`c2` 等等作为列标签。

`cvt_file.pl` 假定有与 `LOAD DATA` 和 `SELECT ... INTO OUTFILE` 一样的默认文件格式，也就是，制表符定界，换行符结束一行的格式。

`cvt_file.pl` 可以在 `recipes` 发行包的 `transfer` 目录中找到。如果你希望正常使用它，你应该将其安装在你的搜索路径下的某个目录中，以便你可以任意调用它。脚本源文件的大部分类似于 `mysql_to_text.pl`，所以与其显示代码并讨论它如何工作，我不如将给出一些示例说明怎么使用它：

- 读取以 CRLF 作为行结束符的 CSV 格式文件，并写下换行符结束，制表符定界的输出：

```
% cvt_file.pl --iformat=csv --ieol="\r\n" --oformat=tab --oeol="\n" \
    data.txt > tmp.txt
```

- 读写 CSV 格式，将 CRLF 行结束符转化为回车符：

```
% cvt_file.pl --format=csv --ieol="\r\n" --oeol="\r" data.txt > % tmp.txt
```

- 从以冒号分隔的`/etc/passwd`生成制表符分隔的文件：

```
% cvt_file.pl --idelim=":" /etc/passwd > tmp.txt
```

- 将从 `mysql` 而来的制表符定界的查询输出转化为 CSV 格式：

```
% mysql -e "SELECT * FROM profile" cookbook \
    | cvt_file.pl --oformat=csv > profile.csv
```

10.19 提取和重排数据文件列

Extracting and Rearranging Database Columns

问题

你想从某个数据文件中取出列或者以不同的顺序对其重新排列。

解决方案

使用一经请求就能从文件生成列的工具。

讨论

cvt_file.pl 是能将整个文件从一个格式转化为另一种的工具。另一种公共的数据文件操作是操作它的列。这是必要的，例如，当将一个文件导入某个程序时，而该程序并不了解如何提取或重排输入列。要解决此问题，你可以重新排列数据文件。

回忆一下本章开始所描述的场景，其中有一个 12 列的 CSV 文件 somedata.csv，而仅需要列 2、11、5 和 9。你可以以如下方式将文件转化为制表符定界的格式：

```
% cvt_file.pl --ifformat=csv somedata.csv > somedata.txt
```

但然后做什么呢？如果你仅仅想写一段脚本来提取特定的 4 列，那相当简单：编写一个循环读取输入行，并仅仅写入你希望出现在合适位置的列。但那将是一个特定目的的脚本，仅可用于一个高度受限的上下文中。仅需要多一点点的努力，就有可能编写一个更通用的工具 yank_col.pl，它能使你提取任意的列集合。用这样一个工具，你应该以如下方式在命令行中指定列列表：

```
% yank_col.pl --columns=2,11,5,9 somedata.txt > tmp.txt
```

因为脚本并不使用硬编码的列列表，所以它能用于以任意顺序去除任意的列集合。列可以以逗号分隔的列序号列表或列取值范围来指定。（例如，`--columns=1,10,4-7` 意指列 1、10、4、5、6 和 7。）yank_col.pl 如下所示：

```
#!/usr/bin/perl
# yank_col.pl - 从输入中提取列

# 示例: yank_col.pl --columns=2,11,5,9 filename

# 假定为制表符定界的，换行符结束的输入行

use strict;
use warnings;
use Getopt::Long;
$Getopt::Long::ignorecase = 0; # 选项是大小写敏感的

my $prog = "yank_col.pl";
my $usage = <<EOF;
Usage: $prog [options] [data_file]

Options:
--help
    Print this message
--columns=column-list
    Specify columns to extract, as a comma-separated list of column positions
EOF

my $help;
my $columns;
```

```

GetOptions (
    "help"      => \$help,          # 输出帮助信息
    "columns=s" => \$columns       # 指定列列表
) or die "$usage\n";
die "$usage\n" if defined $help;

my @col_list = split (/,/, $columns) if defined ($columns);
@col_list or die "$usage\n";        # 需要非空的列列表

# 确定列区分符为正整数，并将其从 1 开始计数转换为从 0 开始计数

my @tmp;
for (my $i = 0; $i < @col_list; $i++)
{
    if ($col_list[$i] =~ /^(\d+)/)           # 单一的列号
    {
        die "Column specifier $col_list[$i] is not a positive integer\n"
            unless $col_list[$i] > 0;
        push (@tmp, $col_list[$i] - 1);
    }
    elsif ($col_list[$i] =~ /^(\d+)-(\d+)/)   # 列范围 m-n
    {
        my ($begin, $end) = ($1, $2);
        die "$col_list[$i] is not a valid column specifier\n"
            unless $begin > 0 && $end > 0 && $begin <= $end;
        while ($begin <= $end)
        {
            push (@tmp, $begin - 1);
            ++$begin;
        }
    }
    else
    {
        die "$col_list[$i] is not a valid column specifier\n";
    }
}
@col_list = @tmp;

while (<>)                      # 读取输入
{
    chomp;
    my @val = split (/\\t/, $_, 10000); # 切分，保存所有域
    # 提取所需要的列，将 undef 匹配到空字符串（如果索引起超出行中列数范围时可能发生）
    @val = map { defined ($_) ? $ : "" } @val[@col_list];
    print join ("\t", @val) . "\n";
}

```

输入处理循环将每行转化为一个值数组，然后从值数组中取出对应于所请求的列的值。为了避免循环整个数组，它使用 Perl 的符号，允许一次性指定一个下标列表以请求多个数组元素。例如，如果@col_list 包含值 2、6 和 3，这两个表达式是等价的：

```
($val[2] , $val[6], $val[3])  
@val[@col_list]
```

如果你想从非制表符定界格式的文件中提取列，或生成另一种格式的输出时，要做什么呢？在这种情况下，将 `yank_col.pl` 和第 10.18 节所讨论的 `cvt_file.pl` 结合起来。假设你想从冒号分隔的/etc/passwd 文件中提取除 password 列外的其他列，并将结果输出为 CSV 格式。使用 `cvt_file.pl` 为 `yank_col.pl` 将/etc/passwd 预处理为制表符定界的格式，并对提取的列进行后期处理，使其输出为 CSV 格式：

```
% cvt_file.pl --idelim=":" /etc/passwd \  
| yank_col.pl --columns=1,3-7 \  
| cvt_file.pl --oformat=csv > passwd.csv
```

如果你不想键入这样一条长命令，使用临时文件作为中间步骤：

```
% cvt_file.pl --idelim=":" /etc/passwd > tmp1.txt  
% yank_col.pl --columns=1,3-7 tmp1.txt > tmp2.txt  
% cvt_file.pl --oformat=csv tmp2.txt > passwd.csv  
% rm tmp1.txt tmp2.txt
```

强制 `split()` 放回每个域

Perl `split()` 函数相当有用，但通常它并不返回后面的空域。这意味着如果你仅写入 `split()` 返回的那些域，输出行的域数可能和输入行不一样。为避免此问题，传入第 3 个参数来表示要返回的域数目的最大值。这就强制 `split()` 返回行中所实际出现的尽可能多的域，或者所请求的数目，而该数值无论如何都更小些。如果第三个参数的值足够大，实践的效果会引起返回所有的域，无论为空与否。本章所示的脚本使用域计数值 10 000：

```
# 以制表符分行，保存所有域  
my @val = split (/\\t/, $_, 10000);
```

在输入行有比其更多的域时（未必有的），它将会被截断。如果你认为那是一个问题，你可以将该数目设为更高。

10.20 使用 SQL 模式来控制错误的输入数据处理

Using the SQL Mode to Control Bad Input Data Handling

问题

在默认的情况下，MySQL 会接受不正确的、越界的或者与你要插入的列的数据类型不匹

配的数据值。(服务器接受这些值并视图将其强制转化为最接近的合法值。) 但你希望服务器更为严格, 不接受错误的数据。

解决方案

设置 SQL 模式。有几个模式值可用于控制服务器的严格程度。其中有些模式应用于所有的输入值。其他的则应用于特定的数据类型, 譬如日期。

讨论

通常, 如果输入不匹配, MySQL 会接受数据并将其强制转化为你表列的数据类型。试想一下下面的表, 它拥有整数、字符串和日期列:

```
mysql> CREATE TABLE t (i INT, c CHAR(6), d DATE);
```

将带有不匹配的数据值的一行数据插入表中会出现警告(你可以用 SHOW WARNINGS 查看), 但在被强制转化为最接近的合法值(或者至少为满足该列的某个值)后值会被装入表中:

```
mysql> INSERT INTO t (i,c,d) VALUES('-1x','too-long string!','1999-02-31');
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message
+-----+-----+-----+
| Warning | 1265 | Data truncated for column 'i' at row 1 |
| Warning | 1265 | Data truncated for column 'c' at row 1 |
| Warning | 1265 | Data truncated for column 'd' at row 1 |
+-----+-----+-----+
mysql> SELECT * FROM t;
+-----+-----+-----+
| i    | c     | d      |
+-----+-----+-----+
| -1   | too-lo | 0000-00-00 |
+-----+-----+-----+
```

在 MySQL5.0 之前的版本中, 阻止这些警告的方法是在客户端检查输入数据以确保其合法性。这在某些场景下是一个合理的策略(参见第 10.21 节中的“服务器端和客户端验证的比较”), 但 MySQL5.0 及其更高版本为你提供了一个可选的方案: 让服务器在服务器端检查数据值并在其不合法时以一个错误来拒绝它们。这样你就不需要检查它们了。

要使用此策略, 通过设置 `sql_mode` 系统变量来开启输入数据接收限制。你可以设置 SQL 模式使服务器比其默认情况更为严格。代之以合适的限制后, 先前会导致转化和警告的数据值现在会引起错误。在启用“严格”SQL 模式后再执行一次前面的 `INSERT` 语句:

```
mysql> SET sql_mode = 'STRICT_ALL_TABLES';
mysql> INSERT INTO t (i,c,d) VALUES ('-1x','too-long string!', '1999-02-31');
ERROR 1265 (01000): Data truncated for column 'i' at row 1
```

这里语句甚至不会处理第二和第三个数据值，因为对于一个整数列第一个值就是不合法的，服务器会抛出一个错误。

甚至当没有开启输入限制时，MySQL 5.0 服务器在日期检查上也比先前的版本更为严格。在 5.0 之前，服务器仅检查日期值的月和日部分是否分别在 1 到 12 和 1 到 31 的范围内。这就会允许一个诸如 '2005-02-31' 这样的日期被输入。而在 MySQL 5.0 中，月份必须是从 1 到 12（和之前一样），但天数必须是给定月的合法值。这就意味着 '2005-02-31' 默认会生成一个警告。

虽然在 MySQL 5.0 中日期检查更为严格，MySQL 仍然允许诸如 '1999-11-00' 或 '1999-00-00' 之类拥有 0 值部分的日期，或 "0" 日期 ('0000-00-00')，这甚至在严格模式下也是正确的。如果你想限制这些类型的日期值，启用 NO_ZERO_IN_DATE 和 NO_ZERO_DATE SQL 模式在严格模式下产生警告或错误。例如，为禁止具有 0 值部分或 "0" 日期的日期值，如下设置 SQL 模式：

```
mysql> SET sql_mode = 'STRICT_ALL_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE';
```

一个可以开启这些限制甚至更多的、更简单的方法，就是使用 TRADITIONAL SQL 模式。TRADITIONAL 模式实际上是一群模式的集合，这可以通过设置 sql_mode 值然后进行显示：

```
mysql> SET sql_mode = 'TRADITIONAL';
mysql> SELECT @@sql_mode\G
***** 1. row *****
@@sql_mode: STRICT_TRANS_TABLES,STRICT_ALL_TABLES,NO_ZERO_IN_DATE,
NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,TRADITIONAL,
NO_AUTO_CREATE_USER
```

你可以阅读 MySQL 参考手册中了解不同的 SQL 模式的更多内容。

这些示例设置了 sql_mode 系统变量的会话值，所以他们仅仅改变你当前连接的 SQL 模式。要为所有客户端设置全局的模式，设置全局 sql_mode 值（需要 SUPER 权限）：

```
mysql> SET GLOBAL sql_mode = ' mode_value ';
```

10.21 验证并转换数据

Validating and Transforming Data

问题

你需要确定文件中所包含的数据值是合法的。

解决方案

对它们进行检查，可能还需要将它们重写为一种更合适的格式。

讨论

本章前面的几节说明了如何通过读取行并将它们划分为不同的列来处理文件的结构性特征。能那么做很是重要，但有时你需要处理一个文件的数据内容，而不仅仅是它的结构：

- 验证数据值以确定它们对于你想将它们存储到的数据类型来说是合法的，通常，这是一个好主意。例如，你可以确定用于 INT、DATE 和 ENUM 列的值分别是整数，CCYY-MM-DD 格式的日期值，以及合法的枚举值。
- 数据值可能需要重新格式化。将日期值从一个格式重写为另一格式尤为普遍；例如，一个程序将 MM-DD-YY 格式的日期值写成 ISO 格式以便于导入到 MySQL 中。如果一个程序仅能理解日期和时间格式，而不能理解复合的日期和时间格式（例如 MySQL 的 DATETIME 和 TIMESTAMP 数据类型），你需要将日期和时间值划分为分离的日期和时间值。
- 识别文件中的特殊值是必要的。用一个不会出现在文件中的值，诸如 -1, Unknown，或 N/A，来表示 NULL 是很普遍的。如果你不想以字面含义导入这些值，你需要对其进行识别并进行特殊的处理。

这是一系列描述验证和格式化技术的章节中的第一篇，这些技术对于以上类型的场景非常有用。这里所涵盖的用于检测值的技术包括模式匹配和验证数据库中的信息。特定的验证操作会反复出现，在这种情况下你将会发现构建一个函数库很有用。通过将验证操作包装为库程序，基于它们编写工具更为方便，并且工具使得对所有文件执行命令行操作更为简单以至于你可以避免自己来编辑它们。

服务器端和客户端验证

如 10.20 节所描述的那样，你可以将 SQL 模式设为严格以避免接受错误的输入数据，从而在服务器端执行数据验证。在这种情况下，如果你要插入的值与目标列的数据类型不匹配，MySQL 服务器会抛出一个错误。

在接下来的几分钟里，焦点是客户端验证而非服务器端验证。当你需要更多的验证控制而非简单的从服务器接收一个错误时客户端验证很有用。（例如，如果你自己对值进行测试，你可以更为方便地提供更多关于值的问题的提示性信息。）而且，将验证和重新格式化结合在一起以转化复杂值以便它们兼容于 MySQL 数据类型可能是必要的。在服务器端完成这点你有更多的灵活性。

如果你想避免编写你自己的库程序，看看是否已经有人已经编写了合适的程序供你使用。例如，如果你检查 Perl CPAN (cpan.perl.org)，你会发现一个 Data::Validate 模块层次。该模块提供了标准化许多通用验证任务的库程序。Data::Validate::MySQL 处理 MySQL 数据类型。

编写一个输入处理循环

在新的几个段落中所示的许多验证 recipes 是你在程序上下文中读取一个文件并检查各自的列值。这样一个文件处理工具的通用框架可以如下方式编写：

```
#!/usr/bin/perl
# loop.pl - 典型的输入处理循环

# 假定为制表符定界的，换行符结束的输入行

use strict;
use warnings;

while (<>)                      # 读取每行
{
    chomp;
    # 以制表符分割行，保存所有域
    my @val = split (/^\t/, $_, 10000);
    for my $i (0 .. @val - 1) # 迭代行中的所有列
    {
        # ... 在此测试$val[$i] ...
    }
}
```

while() 循环读取每个输入行并将其划分为域。在循环中，每行被分为多个域。然后内部的 for() 循环对行中的域进行迭代，允许顺序处理每个域。如果你不想统一应用一个给定的测试到所有的域上，用特定的列相关测试替代 for() 循环。

这个循环假定输入是以制表符定界，换行符结束，这是本章余下部分所讨论的大多数工具所共享的一个假定。要对其他格式的数据文件使用这些程序，你可以使用第 10.18 节所讨论的 cvt_file.pl 脚本将文件转换为制表符定界的格式。

将普遍的测试放入库中

对于某个你常执行的测试，将其封装为一个库函数是很有用的。这使得操作易于执行，并且赋给其一个有意义的名称，这相对比较代码自身来说更为清楚明白。例如，下面的测试执行一次模式匹配来检查\$val 是否完全由数字组成（前面可跟一个+号，可选的），然后确定该值大于 0：

```
$valid = ($val =~ /^[+\d+]?$/ && $val > 0);
```

换句话说，测试寻找代表正数的字符串。为使测试更便于使用且其意图更为清晰，你可以将它放入如下所示的函数中：

```
$valid=is_positive_integer($val);
```

函数本身可以被定义如下：

```
sub is_positive_integer
{
my $s = $_[0];
return ($s =~ /^\\+?\\d+$/ && $s > 0);
}
```

现在将函数定义放入一个库文件中以便多个脚本可以方便的使用它。发行包的 lib 目录中的 Cookbook_Utils.pm 模块文件是一个库文件的示例，其中包含许多验证函数。可以通读它来看看哪些函数在你自己的程序中能用到（或者作为一个编写你自己的库文件的模型）。为在脚本中能访问这个模块，以如下方式包括一个 use 语句：

```
use Cookbook_Utils;
```

当然，你必须将模块文件安装在一个 Perl 能够找到的目录中。要获取库安装的细节，请参考第 2.3 节。

将一系列工具程序放入一个库文件的一个显著好处就是你可以在所有种类程序中使用它。某个数据操作问题很少是完全唯一的。如果你可以从一个库中挑选出一些验证程序，就可能缩减你必须编写的代码数量，甚或对于高度专业化的程序也是如此。

10.22 使用模式匹配来验证数据

Using Pattern Matching to Validate Data

问题

你想将一个值与一系列难于用文字表示的值进行比较，但又不想用太过难看的表达式。

解决方案

使用模式匹配。

讨论

模式匹配是一项用于验证的强大工具，因为它可以使你用一个简单的表达式来测试整个类型的值。你还可以使用模式测试来将匹配的值划分为几个部分以进行进一步的独立测试或

者执行替换操作来重写匹配的值。例如，你可以将一个匹配的日期分为多个部分以便你能验证月份的取值在 1 到 12 之间，且天数在当月的天数取值范围内。你可以使用一个替代操作来将 *MM-DD-YY* 或 *DD-MM-YY* 值重排成 *YY-MM-DD* 格式。

接下来的几个部分描述了如何使用模式来测试几种类型的值，但首先让我们快速的了解一下一些通用的模式匹配原则。下面的讨论重点是 Perl 的正则表达式能力。Ruby、PHP 和 Python 中的模式匹配是类似的，虽然你应该查阅相关的文档来了解它们的不同之处。对于 Java，使用 `java.util.regex` 包。

在 Perl 中，模式构建器是 `/pat/`:

```
$it_matched = ($val =~ /pat/);      # 模式匹配
```

将 `i` 放在 `/pat/` 构建器之后使模式匹配大小写无关:

```
$it_matched = ($val =~ /pat/i);    # 大小写无关的匹配
```

要使用除斜线外的字符，以 `m` 开始构建器。如果模式自身包含斜线时这很有用:

```
$it_matched = ($val =~ m|pat|);    # 替换构建器字符
```

要查找不匹配的，用 `!~` 操作符来替换 `=~` 操作符:

```
$no_match = ($val !~ /pat/);      # 反模式匹配
```

要在 `$val` 基于模式匹配执行替换操作，使用 `s/ pat / replacement /`。如果 `pat` 出现在 `$val` 中，就会被 `replacement` 替换。要执行大小写无关的匹配，将 `i` 放置于最后一个斜线之后。要执行全局的替换，替换掉 `pat` 的所有实例而非仅第一个，在最后一个斜线后加上 `g`:

```
$val =~ s/pat/replacement/;      # 替换
$val =~ s/pat/replacement/i;    # 大小写无关的替换
$val =~ s/pat/replacement/g;    # 全局替换
$val =~ s/pat/replacement/ig;   # 大小写无关和全局结合
```

下面是 Perl 正则表达式中可用的特殊模式元素的列表:

模式	模式所匹配的内容
<code>^</code>	字符串起始位置
<code>\$</code>	字符串结尾
<code>.</code>	任意字符
<code>\s, \S</code>	空格或非空格字符
<code>\d, \D</code>	数值，或非数值字符
<code>\w, \W</code>	任意 ASCII 单字字符，或任意 ASCII 非单字字符
<code>[...]</code>	位于括号中的任意字符
<code>[^...]</code>	不在括号内的任意
<code>p1 p2 p3</code>	可选项；匹配 <code>p1</code> 、 <code>p2</code> 或 <code>p3</code> 任一个

模式	模式所匹配的内容
*	匹配前一项 0 次或多次
+	匹配前一项 1 次或多次
{ n }	匹配前一项 n 次
{ m , n }	匹配前一项至少 m 次，但不能超过 n 次

这些模式元素中的多数和 MySQL 的 REGEXP 正则表达式操作符（参考 5.11 节）是一致的。

要匹配模式中所出现的一些特殊字符，诸如*、^或\$，在前面加上反斜线即可。同样的，要在一个字符类构建器中包含字符类中（[，]或-）的特殊字符，也要在前面加上反斜线。要在一个字符类中包含一个文本^，将其放置于括号中第一个字符以外的任何位置。

下面的段落所述说的多数验证模式是/*pat*\$/>形式。以^和\$作为模式的开始和结束起到了要求 *pat* 匹配你正在测试的整个字符串的效果。这在数据验证场景下是很普遍的，因为通常都要求一个模式匹配完整的输入值，而不是其部分。（例如如果你想确保某个值表示一个整数，仅知道它在某个位置包含一个整数对于你来说没有任何用处。）然而，这并非一个必须遵守的规则，有时在适当情况下通过略去^和\$字符执行不太严格的测试更有用。例如，如果你想去掉某个值头部和尾部的空格，使用仅定位到字符串头部的模式和另一个仅定位到尾部的模式：

```
$val =~ s/^[\s+]/;    # 去除头部的空格
$val =~ s/[\s+]\$/;    # 去除尾部的空格
```

事实上，这是如此普遍的一个操作，所以它是写成一个工具函数的很好的候选。Cookbook_Utils.pm 文件包含了一个 trim_whitespace() 函数，它执行了替换和返回结果两者：

```
$val = trim whitespace ($val);
```

要记住模式所匹配的子字符串，在模式的相应部分用括号括起来。在一次成功的匹配后，你可以使用变量 \$1、\$2 等引用所匹配的子字符串：

```
if ("abcdef" =~ /^(\w)(.*)$/)
{
    $first_part = $1; # 这将是 ab
    $the_rest = $2;   # 这将是 cdef
}
```

为表明模式中的某个元素是可选的，在其后紧跟一个?字符。要匹配由数字序列组成的值，以一个可选的负号开头，并以可选的句点结束，使用如下模式：

/^-?\d+\.\?\$/

你还可以使用括号对模式中的可选项进行分组。下面的模式匹配 *hh:mm* 格式的时间值，后面可选择跟 AM 或 PM：

/^\d{1,2}:\d{2}\s*(AM|PM)?\$/i

在该模式中括号的使用也有在 \$1 中记住可选择的部分的副作用。为阻止此副作用，使用 `(?: pat)` 取而代之：

/^\d{1,2}:\d{2}\s*(?:AM|PM)?\$/i

在 Perl 模式匹配中，这是足够充分的背景知识，可用于对多种类型的数据值进行有效的验证测试构建。后面的章节提供了能用于测试多种内容类型，数值，日期时间值，以及 E-mail 地址或 URL 的模式。

recipes 发行包的 transfer 目录中包含一个 test_pat.pl 脚本，它可读取输入值，将它们和多种模式进行匹配，然后报告每个值匹配何种模式。这个脚本易于扩展，所以你可以使用它作为一个测试套来尝试你自己的模式。

10.23 使用模式来匹配广泛的内容类型

Using Patterns to Match Broad Content Types

问题

你希望将值分为不同的种类。

解决方案

使用相应的模式。

讨论

如果你需要了解值是否为空，或者仅由特定类型的字符组成，下表列出的模式可以满足需要：

模式	模式所匹配的值类型
/^\$/	空值
/./	非空值
/^\s*\$/	空格，可能为空
/^\s+\$/	非空的空格
/\S/	非空，且非空格
/^\d+\$/	非空的数值
/^*[a-z]+\$/i	仅字母组成的字符（大小写无关），非空
/^*[w]+\$/	字母数字或下划线字符，非空

10.24 使用模式来匹配数值

Using Patterns to Match Numeric Values

问题

你必须确保一个字符串是一组数字。

解决方案

使用匹配你正寻找的数值类型的模式。

讨论

模式能用于将值划分为几种类型的数值。

模式	模式匹配的值类型
/^\d+\$/	无符号整数
/^-?\d+\$/	负数或无符号整数
/^[-+]? \d+\$/	有符号或无符号整数
/^[-+]? (\d+(\.\d*)? \.\d+)\$/	浮点数

模式`/^$\d+$/`通过要求从头到尾仅由数字组成的一个非空值来匹配无符号整数。如果你只关注值是由数字开头的，你可以匹配一个初始的数值部分并提取它。要做到这点，仅匹配字符串的起始部分（略去要求模式匹配到字符串尾部的\$）并在`\d+`部分加上括号。然后在一次成功的匹配后用`$1`来引用被匹配的数值：

```
if ($val =~ /^(\d+)/)
{
    $val = $1; # 将值重设为匹配的子串
}
```

你还可以为该值加0，这会导致Perl执行一次隐式的字符串到数值的转换，它会抛弃非数值的后缀：

```
if ($val =~ /^$\d+/)
{
    $val += 0;
}
```

然而，如果你用`-w`选项来运行Perl或者在你的脚本中包含了一个`use warnings`行（这是我所推荐的），这种形式的转换用于实际拥有非数值部分的值时会产生警告。它还会将

诸如 0013 这样的字符串值转化为数值 13，这在某些上下文中是不可接受的。

某些类型的数值有一些特殊的格式或其他不同寻常的约束。下面是一些示例以及如何去处理它们：

邮政编码

Zip 和 Zip+4 编码是美国用于邮寄信件的邮政编码。它们的值如 12345 或 12345-6789（也就是说，5 位数字，可能后跟一个折线和 4 位数字）。要匹配其中任一种，或两种都匹配，使用下面的模式：

模式	模式所匹配的值类型
/^\d{5}\$/	邮编，仅 5 位数字
/^\d{5}-\d{4}\$/	Zip+4 编码
/^\d{5}(-\d{4})?\$/	Zip 或 Zip+4 编码

信用卡编号

信用卡编号通常由数字组成，但在几组数字之间常用空格、破折号或其他字符分开。例如，下面的数字被认为是等价的：

```
0123456789012345  
0123 4567 8901 2345  
0123-4567-8901-2345
```

要匹配这些值，使用这个模式：

```
/^[- \d]+/
```

(Perl 允许在字符类中出现\d 数字区分符。) 然而，该模式并不能识别长度有错的值，并且它能用于移除多余的字符。如果你要求信用卡值包含 16 位数字，进行一次替换来移除所有的非数字字符，然后检查结果的长度：

```
$val =~ s/\D//g;  
$valid = (length ($val) == 16);
```

所投的局数

在棒球中，投手的一种记录统计是投出的击球数目。用三振来衡量（对应于所记录的出局数）这些值是数值形式，但必须满足一种特定的额外约束：允许小数部分，但如果有，必须仅包含一个数值 0、1 或 2。也就是说，合法值是 0、.1、.2、1、1.1、1.2、2，等形式。要匹配一个无符号的整数（后可跟小数点和可能的小数部分数值 0、1 或 2），或者没有整数部分的一个小数值，使用这个模式：

```
/^(\d+(\.\[012]\?)?|\.\[012])$/
```

模式中的可选项部分用括号括了起来，因为要不然^仅会定位到它们中的第一个，而\$仅会定位第二个到字符串的尾部。

10.25 使用模式来匹配日期或时间

Using Patterns to Match Dates or Times

问题

你需要确定一个字符串是一个日期或时间。

解决方案

使用模式来匹配你期望的日期值类型。但要确保考虑清楚各个部分之间的定界符以及各个部分的长度等相关问题。

讨论

日期是一个验证难题，因为它们有太多的格式。模式测试对于排除非法值很有用，但对于全面验证常不太充分：某个日期在你希望是月份的地方可能有一个数值，但如果数值是13，日期就是非法的。本节介绍一些匹配通用日期格式的模式。第10.30节更为详细的讨论了这个主题，并探讨如何将模式测试和内容验证结合在一起。

要求值是ISO (CCYY-MM-DD) 格式的日期，使用这个模式：

```
/^\d{4}-\d{2}-\d{2}$/
```

该模式要求字符-作为日期各个部分之间的分隔符。为允许-或/作为定界符，在数值部分之间使用一个字符类（斜线要用反斜线来转义以避免被解释为模式构建器的结尾）：

```
/^\d{4}[-/] \d{2}[-/] \d{2}$/
```

或者，你可以用其他的定界符包含模式来避免使用反斜线：

```
m|^\d{4}[-/] \d{2}[-/] \d{2}$/
```

为允许任意非数字定界符（这对应于当它将字符串解释为日期时MySQL如何操作），使用这个模式：

```
/^\d{4}\D\d{2}\D\d{2}$/
```

如果你不需要在每个部分都要求填满数值（例如，允许如03这样的值前面省去0），仅查找三个非空的数值序列即可：

```
/^\d+\D\d+\D\d+$/
```

当然，这个模式是如此通用，以致它还能匹配其他的诸如美国社保卡号（格式如 012-34-5678）等值。为限制各个子部分的长度，在年份中要求 2 到 4 位数字，在月份和天数中要求 1 到 2 位数字，使用这个模式：

```
/^\d{2,4}?\D\d{1,2}\D\d{1,2}$/
```

对于诸如 *MM-DD-YY* 或 *DD-MM-YY* 等其他格式的日期，同样的模式也能适用，只是各个部分的顺序有所不同。下面这个模式匹配这两者：

```
/^\d{2}-\d{2}-\d{2}$/
```

如果你需要检查日期不同部分的值，在模式中使用括号并在一次成功的匹配之后提取子字符串。如果你正期盼日期是 ISO 格式，例如，以如下方式来做：

```
if ($val =~ /(\d{2,4})\D(\d{1,2})\D(\d{1,2})$/)
{
    ($year, $month, $day) = ($1, $2, $3);
}
```

recipes 发行包中的 lib/Cookbook_Utils.pm 库文件包含这些模式测试中的多个，它们被封装为函数调用。如果日期与模式不相匹配，它们会返回 `undef`。否则，它们会返回一个包含年、月、日等值的数组引用。这对于日期各个组成部分上执行进一步检验是很有用的。例如，`is_iso_date()` 查找匹配 ISO 格式的日期。它被定义为如下形式：

```
sub is_iso_date
{
    my $s = $_[0];

    return undef unless $s =~ /(\d{2,4})\D(\d{1,2})\D(\d{1,2})$/;
    return [ $1, $2, $3 ]; # 返回年、月、日
}
```

为使用此函数，照下面的方式做：

```
my $ref = is_iso_date ($val);
if (defined ($ref))
{
    # $val 匹配 ISO 格式模式;
    # 使用 $ref->[0] 到 $ref->[2] 来检查它的各个部分
}
else
{
    # $val 不匹配 ISO 格式模式
}
```

你将常会看见需要对日期进行必要的额外处理，因为虽然日期匹配模式排除了句法上错误的值，但它们并不能评估不同的组成部分是否包含合法值。要达成此目的，某些范围检查是必须的。该主题在后面的第 10.30 节中有所说明。

如果你想跳过子元素测试，而只是想重写部分值，你可以执行一次替换。例如，为将 MM-DD-YY 格式的值转成 YY-MM-DD 格式，应这么做：

```
$val =~ s/^(\d+)\D(\d+)\D(\d+)/$3-$1-$2/;
```

时间值有时比日期值更为有序，通常开头是小时，结尾是秒数，每部分两个数字：

```
/^\d{2}:\d{2}:\d{2}$/
```

放宽来说，你可以允许小时部分仅为一个数字，或者略去秒数部分：

```
/^\d{1,2}:\d{2}(:\d{2})?$/
```

如果你想对不同的组成部分进行范围检查，或者在秒数部分不存在时重新格式化值以使其包含为 00 的秒数部分，你可以用括号对时间的各个部分做标记。然而，如果秒数是可选项时需要注意括号和模式中的?字符。你希望允许模式尾部的整个:\d{2} 是可选的，但不想在时间的第三个部分出现时在 \$3 中保存:字符。为达成此目的，使用(?: pat)，一个不保存匹配的子字符串的替代的分组符号。在该符号中，在数字周围使用括号以保存它们。如果秒数部分没有出现的话 \$3 将是 undef，否则将包含秒数数值：

```
if ($val =~ /^( (\d{1,2}):(\d{2})(?: (\d{2}))?)?$/)
{
    my ($hour, $min, $sec) = ($1, $2, $3);
    $sec = "00" if !defined($sec); # 秒数不存在，用 00
    $val = "$hour:$min:$sec";
}
```

要将时间从带有 AM 和 PM 后缀的 12 小时格式转化为 24 小时格式，你可以按照如下的方式来实现：

```
if ($val =~ /(^(\d{1,2}):(\d{2})(?: (\d{2}))?)?\s*(AM|PM)?$/i)
{
    my ($hour, $min, $sec) = ($1, $2, $3);
    # 提供缺少的秒数
    $sec = "00" unless defined($sec);
    if ($hour == 12 && (!defined($4) || uc($4) eq "AM"))
    {
        $hour = "00"; # 12:xx:xx AM 时间是 00:xx:xx
    }
    elsif ($hour < 12 && defined($4) && uc($4) eq "PM")
    {
        $hour += 12; # PM 时间要加上 12:xx:xx
    }
    $val = "$hour:$min:$sec";
}
```

时间的各个部分被放入 \$1、\$2 和 \$3 中，如果秒数部分不存在 \$3 被设为 undef。如果后缀

存在则对应 \$4。如果后缀是 AM 或没有 (undef)，值会被解释为一个 AM 时间。如果后缀是 PM，值被解释为 PM 时间。

参考

这段仅是处理日期用于数据转换时你可以所能做的事情的开始。日期和时间测试及转化可以很特殊的，并且要考虑的问题数目令人难以置信：

- 基本的日期格式什么？日期有多种通用样式，如 ISO (*CCYY-MM-DD*)，U.S. (*MM-DD-YY*)，和英国 (*DD-MM-YY*) 格式。而这些仅是更多标准格式中的一些。太多可能。例如，某个数据文件可以包含写成 June 17, 1959 或 17 Jun '59 的日期。
- 日期尾部是否允许加上时间值或者必须加上？当需要时间时，是需要完整的时间还是仅小时和分钟？
- 是否允许诸如 `now` 或 `today` 这样的特殊值？
- 日期部分是否要求用某个特定的字符如-或/分隔？或者允许使用其他的分隔符？
- 日期部分是否要求为特定的数字数目？或者允许略去月份或年份值的前导 0 值？
- 月份是被写成数值形式的，还是用月份名称如 `January` 或 `Jan` 来表示？
- 是否允许 2 位数值的年份值？它们是否应该被转化为 4 位数值形式？如果是，转换规则是什么？(00 到 99 之间哪个值是转换点，从而在此值从一个世纪变成另一个世纪。)
- 日期部分是否应该进行检查以确保它们的正确性？模式可以识别看上去如日期或时间的字符串，但如果它们用于检测有错误的值，可能就不够。像 1947-15-99 这个值可以匹配某个模式，但它不是一个合法的日期。因此模式测试结合对日期的不同部分进行范围检查是最有用的。

数据转换中存在的这些普遍问题，意味着你也许要编写自己的验证器在某些场合处理非常特殊的日期格式。本章后面的几节可以提供更多的帮助。例如，10.29 节就介绍了如何将 2 位数值形式的年份值转换为 4 位数值形式，10.30 节讨论了如何对日期或时间值的各个组成部分进行正确性检查。

10.26 使用模式来匹配 E-mail 地址或 URL

Using Patterns to Match Email Addresses or URLs

问题

你希望确定一个值是否是一个 E-mail 地址或一个 URL。

解决方案

使用一种模式，将严格级别调整到你希望的程度。

讨论

前面的几节使用模式来识别诸如数值和日期等类型的值，这是正则表达式相当典型的应用。但模式匹配有如此广泛的应用以致不可能列出你可以用它进行数据验证的所有方法。为给出模式匹配可用于的其他类型值的一些观感，本节展示了一些 E-mail 地址和 URL 的测试。

要检查期盼是 E-mail 地址的值，模式应该要求至少有一个@字符，且两端都有非空字符串：

```
/^.@./
```

这是一个非常小的测试。很难实现一个完全通用的模式来涵盖所有的合法值并拒绝所有的非法值（注 1），但写出一个至少更为严格的模式还是比较简单的。例如，除非空外，用户名和域名应该由@字符和空格外的字符组成：

```
/^[@\w\W]+\@[^\w\W]+$/
```

你还可以要求域名部分包含至少两个由逗号分隔的部分：

```
/^[@\w\W]+\@[^\w\W]+(\.\w\w)+$/
```

要查找以协议指定符譬如 `http://`、`ftp://` 或 `mailto://` 等开头的 URL 值，在字符串的起始部分使用一个可选项来匹配它们的任一个。这些值包含斜线，所以在模式外用一个差异字符以避免不得不使用反斜线来转义斜线更简便一些：

```
m#^(http://|ftp://|mailto:)#i
```

模式中的可选项用括号括了起来，因为要不然^仅会将它们中的第一个定位为字符串的头部。模式后面紧跟 i 修饰符是因为 URL 中的协议指定符是大小写无关的。这个模式不太

注 1：要明白对 E-mail 地址进行模式匹配有多么困难，请参考 Jeffrey E. F. Friedl 的《Mastering Regular Expressions》一书（O'Reilly）（《精通正则表达式 第 3 版》，电子工业出版社）。

严格，因为它允许在协议指定符后面跟任意字符。这个留给读者自己去进一步添加必要的限制。

10.27 使用表元数据来验证数据

Using Table Metadata to Validate Data

问题

你需要根据一个 ENUM 或者 SET 类型列的成员来检查输入值的合法性。

解决方案

获取列定义，从其中取出成员列表，然后根据该列表检查（输入）数据。

讨论

有一些形式的验证涉及根据数据库中保存的值来验证输入值。这包括将要保存到一个 ENUM 或者 SET 列中的值，可以根据该列定义中包含的有效成员对输入进行验证。当你要检测一些必须与一个查找表结果匹配的值的合法性时，也需要数据库端的验证。例如，输入含有 customer ID 的记录，可能要求与 customers 表中的一行相匹配，或者地址中（美国）州名的缩写能够根据记录每一个州名（缩写）的表来检验。本节讨论了基于 ENUM- 和 SET- 的验证，10.28 节讨论了如何使用查找表（来做验证）。

验证输入值对于 ENUM 或者 SET 列值合法性的一种方法，是使用 INFORMATION_SCHEMA 中的信息把合法的列值保存到一个数组中，然后执行一个是否该数组成员的检查。例如，profile 表中的 favorite-color 列是如下定义的一个 ENUM 列：

```
mysql> SELECT COLUMN_TYPE FROM INFORMATION_SCHEMA.COLUMNS
    -> WHERE TABLE_SCHEMA = 'cookbook' AND TABLE_NAME = 'profile'
    -> AND COLUMN_NAME = 'color';
+-----+-----+
| COLUMN_TYPE |           |
+-----+-----+
| enum('blue','red','green','brown','black','white') |           |
+-----+-----+
```

如果你从 COLUMN_TYPE 值中提取出枚举值列表，并将其保存在一个数组 @members 中，你就可以这样进行成员资格检测：

```
$valid = grep (/^$val$/i, @members);
```

这个模式构造器分别以 ^ 和 \$ 开始和结束，以此来要求 \$val 与枚举值完全匹配（而不是与子串匹配）。同时它以 i 结尾，定义了一个大小写敏感的模式匹配，因为默认字符集

`latin1_swedish_ci` 是大小写敏感的。(如果你正在处理使用其他字符集的列，根据实际进行调整)。

在 9.7 节中，我们写一个函数 `get_enumorset_info()`，它返回 ENUM 或者 SET 列的元数据。这包括 (ENUM 或者 SET 的) 成员列表，那么就很容易定义另外一个工具程序，`check_enum_value()`，它获取合法的枚举值，并进行成员资格检测。该函数有 4 个参数：数据库句柄、表名、ENUM 列的列名，已经需要检查的值。函数返回值是 `true` 或者 `false` 来表示被检测值是否合法：

```
sub check_enum_value
{
    my ($dbh, $db_name, $tbl_name, $col_name, $val) = @_;
    my $valid = 0;
    my $info = get_enumorset_info ($dbh, $db_name, $tbl_name, $col_name);
    if ($info && uc ($info->{type}) eq "ENUM")
    {
        # 使用大小写无关的比较，因为默认字符序(latin1_swedish_ci)
        # 是大小字无关的。(如果你使用一个不同的字符序，自己进行调整)
        $valid = grep (/^$val$/i, @{$info->{values}});
    }
    return $valid;
}
```

对单个值检测，例如验证 Web 表单中提交的一个值，上面的检测方法是有效的。然而，如果你需要检测很多值(例如一个数据文件中的所有值)，最好一次将枚举值读入内存中，然后逐个的检测输入值。更进一步，哈希查找比直接进行数组查找更高效(至少在 Perl 里面是这样的)。要达到这个目的，查询枚举值，并将它们作为哈希键值保存。然后对每一个输入值检查它是否作为一个哈希键值存在。这需要更多的功夫来构造哈希，这就是 `check_enum_value()` 不这样做的原因。但是对于大批量的验证，查找速度的提升优于构造哈希表带来的影响(注 2)。

从获取一列的元数据开始，然后将合法的枚举值列表转换为哈希：

```
my $ref = get_enumorset_info ($dbh, $db_name, $tbl_name, $col_name);
my %members;
foreach my $member (@{$ref->{values}})
{
    # 将哈希键值转换成一致的小写形式
    $members{lc ($member)} = 1;
}
```

注 2：如果你想自己来检测一下数组成员测试相对哈希表查找的相对效率，使用 `recipes` 发行包 `transfer` 目录中的 `lookup_time.pl`。

这个循环将每一个枚举值转换为一个哈希元素的键值。哈希键值就是这里的关键点，与之关联的值与我们的处理过程是无关的。(这个例子中将哈希值设为 1，但是你也可以选择 0，或者 `undef`，或者其他任意值)。注意程序在保存哈希键值之前先将它们都转换为小写。这样做是因为 perl 中的哈希键值查询是大小写敏感的。

如果你要检测的值也是大小写敏感的，那么这样做没问题，但是默认情况下 `ENUM` 列值不是这样的。在将枚举值保存到哈希之前将它们转换为给定的大小写，然后对你要检测的值做相同的大小写转换，这样你所做的就是大小写不敏感的键值存在性检测。

```
$valid = exists ($members{lc ($val)});
```

前面的例子将枚举值和输入值（待检测值）都转换为小写形式，你也可以采用大写——你这样做的时候需要保证所有值保持一致。

注意如果输入的是空字符串，键值存在性检测可能失败。你需要根据不同的列定义来处理这一问题。例如，如果该列允许 `NULL` 值，有可以将空串解释为 `NULL` 或者以某一合法值。

对于 `SET` 值的验证过程类似于 `ENUM` 值，不同之处仅在于输入值可能包含逗号分隔的任意个数 `SET` 成员。如果输入值合法，要求其中包含的每一个值都合法。另外，由于“任意个数”包含了“none”（空集）的情况，空串对于任何一个 `SET` 类型列都是合法值。

对单个输入值的一次测试，你可以使用工具程序 `check_set_value()`，它类似于 `check_enum_value()`：

```
sub check_set_value
{
    my ($dbh, $db_name, $tbl_name, $col_name, $val) = @_;
    my $valid = 0;
    my $info = get_enumorset_info ($dbh, $db_name, $tbl_name, $col_name);
    if ($info && uc ($info->{type}) eq "SET")
    {
        return 1 if $val eq "";      # 空字符串是合法的
        # 使用大小写无关的比较，因为默认字符序(latin1_swedish_ci)
        # 是大小字无关的。(如果你使用一个不同的字符序，自己进行调整)
        $valid = 1;      # 假设值是合法的，直到我们发现不同
        foreach my $v (split (/,/, $val))
        {
            if (!grep (/^$v$/i, @{$info->{values}}))
            {
                $valid = 0; # 值包含一个非法的元素
                last;
            }
        }
    }
}
```

```
    return $valid;
}
```

对于批量的数据检测，根据 SET 成员构造一个哈希。这个过程和根据 ENUM 成员构造哈希相同：

```
my $ref = get_enumorset_info ($dbh, $db_name, $tbl_name, $col_name);
my %members;
foreach my $member (@{$ref->{values}})
{
    # 将哈希键值转换为一致的小写形式
    $members{lc ($member)} = 1;
}
```

为了使用 SET 成员哈希来检测输入值，(需要)把输入值转换为与哈希键值相同的大小写，按照逗号分隔构造一个输入的单个值的列表，然后检测每一个值。如果其中的任意一个非法，那么整个输入值就是非法的。

```
$valid = 1;      # 假设值是合法的，直到我们发现不同
foreach my $elt (split (/,/, lc ($val)))
{
    if (!exists ($members{$elt}))
    {
        $valid = 0; # 值包含一个非法的元素
        last;
    }
}
```

循环结束以后，如果输入值对于 SET 列是合法的，那么\$valid 为 true，否则为 false。空串对 SET 来说总是合法的，但是上面的程序对空串没有做任何特殊处理。那样的特殊处理并不必要，因为对于空串 split() 操作返回一个空的列表，循环也就不会开始，结果 \$valid 保持为 true。

10.28 使用一个查找表来验证数据

Using a Lookup Table to Validate Data

问题

你需要检查值以确定它们在一个查找表中。

解决方案

发起语句来看看值是否位于表中。然而，你这么做的方法取决于输入值的数目和表的大小。

讨论

要验证输入值是否和某个查找表中的内容一致，你可以使用类似于 10.27 节所示检测

ENUM 和 SET 列的某些技术。然而， ENUM 和 SET 列分别被限制为最大 65 535 和 64 个成员值，而一个查找表可以有几乎无限数目的值。你可能不想将它们都读入内存中。

如后面的讨论中所述，根据查找表的内容对输入值进行验证可以通过几种方法来实现。范例中所示的测试将值与存储在查找表中的内容进行比较。要执行大小写无关的比较，记住将所有的值转化为一致的小写形式。

发起不同的查询

对于仅一次的操作，你可以通过检查它是否存在于查找表中来测试值。下面的查询在存在该值时返回 true（非零），否则返回 false：

```
$valid = $dbh->selectrow_array (
    "SELECT COUNT(*) FROM $tbl_name WHERE val = ?",
    undef, $val);
```

这种测试适用于诸如检查 Web 表单中提交的值等目的，但对于验证大数据集来说是不够恰当的。没有内存用于存储先前对那些已经见过的值所做的测试的结果；结果就是，你不得不对每个单独的输入值发起一次查询。

从整个查找表构建一个哈希表

如果你将对一个大值集执行批量验证，将查找值取出置于内存中，用某种数据结构来保存它们，并根据该结构的内容检查每个输入值，这种方式更高效。使用内存查找避免了为每个值运行一次查询的过度执行。

首先，执行一次查询以检索所有的查找表值，并为它们构建一个哈希表：

```
my %members; # 查找表值的哈希表
my $sth = $dbh->prepare ("SELECT val FROM $tbl_name");
$sth->execute ();
while (my ($val) = $sth->fetchrow_array ())
{
    $members{$val} = 1;
}
```

然后通过执行一次哈希表键存在性测试来检查每个值：

```
$valid = exists ($members{$val});
```

这将数据库通信减至一次简单的查询。然而，对于一个很大的查找表，那可能仍是很大的流量，你可能不想将整个表置于内存中。

用其他的语言来执行查找

这里所示的用于查找值的批量测试的示例使用 Perl 哈希表来确定某个给定值是否出现在值集中：

```
$valid = exists ($members{$val});
```

其他语言中也有类似的数据结构。在 Ruby 中，使用一个哈希表，然后使用 `has_key?` 方法来检查输入值：

```
valid = members.has_key?(val)
```

在 PHP 中，使用一个关联数组，并按如下方式执行一次键查找：

```
$valid = isset ($members[$val]);
```

在 Python 中，使用字典，并使用 `has_key()` 方法检查输入值：

```
valid = members.has_key (val)
```

对于 Java 中的查找，使用一个 `HashMap`，并使用 `containsKey()` 方法测试值：

```
valid = members.containsKey (val);
```

`recipes` 发行包的 `transfer` 目录中包含用这些语言执行查找操作的一些示例代码。

使用哈希表作为已经存在的查找值的缓存

另一种查找技术是混合使用独立的语句和存储查找值存在信息的哈希表。如果你拥有一个非常大的查找表时这个方法很有用。从一个空的哈希表开始：

```
my %members; # 查找值的哈希表
```

然而，对于每个被测的值，检查它是否存在于哈希表中。如果没有，发起一次查询以确定其是否存在于查找表中，并将查询的结果记录在哈希表中。输入值的正确性是由与键关联的值所确定的，而不是该键的存在与否：

```
if (!exists ($members{$val})) # 此值之前未出现过
{
    my $count = $dbh->selectrow_array (
        "SELECT COUNT(*) FROM $tbl_name WHERE val = ?",
        undef, $val);
    # 存储 true/false 以表示值是否发现过
    $members{$val} = ($count > 0);
}
$valid = $members{$val};
```

对于此方法，哈希表作为一个缓存而存在，所以对于每个给定值不论它在输入中出现了多少次，你仅执行了一次查找表查询。对于拥有合理数目重复值的数据集来说，这个方法避免了对每个单值测试发起一次查询，而且对于每个唯一值在哈希表中仅需要一个条目。因此它在另两种方法针对数据库通信和哈希表所需要的程序内存的开销之间进行了折衷。

注意这个方法中哈希表的使用与前面的方法在方式上有所不同。之前，输入值作为哈希表中的键，它的存在与否确定了值的正确性，哈希表键所关联的值是不相关的。对于哈希表作为缓存的方法，键存在于哈希表中的意义从“它是正确的”变成了“它之前被测试过。”对于每个键，与它关联的值表示输入值是否存在于查找表中。（如果如果仅将那些在查找表中被找到过的值存为键，对于输入数据集中一个不正确的值的每个出现你都将发起一次查询，这是不恰当的。）

10.29 将两个数字的年份值转化为四位形式

Converting Two-Digit Year Values to Four-Digit Form

问题

你需要将日期值中的年份从 2 位转化为 4 位。

解决方案

让 MySQL 来为你完成此任务，或者在 MySQL 的转换规则不适合的时候你自己来执行操作。

讨论

2 位的年份值是一个问题，因为在数据值中世纪值不是显式的。如果你知道你的输入的年份跨度，你可以明确的添加世纪数。否则，你只能进行猜测。例如，日期 2/10/69 可能被多数在美国的人解释成 1969 年 2 月 10 日。但如果它代表圣雄甘地的诞辰，真实的年份是 1869。

将年份转化成 4 位数值的一个方法是让 MySQL 来完成此任务。如果你存储一个包含 2 位数值年份的日期，MySQL 会自动将它转换为 4 位数值的形式。MySQL 用 1970 作为一个转换点；它会将从 00 到 69 的值解释为 2000 到 2069 年，从 70 到 99 的值解释为 1970 到 1999 年。这些规则对于从 1970 到 2069 的年份都适用。如果你的值在此范围之外，那么在将它们存入 MySQL 之前你应该自己加上适当的世纪数。

要使用一个不同的转换点，需要自己将年份转化为 4 位数值的形式。下面是一个通用的程序，它可以讲 2 位数值年份转化成 4 位年份并允许任意的转换点：

```
sub yy_to_ccyy
{
    my ($year, $transition_point) = @_;
    $transition_point = 70 unless defined ($transition_point);
```

```
$year += ($year >= $transition_point ? 1900 : 2000) if $year < 100;  
return ($year);  
}
```

函数默认使用 MySQL 的转换点 (70)。可选的第二个参数可用于提供一个不同的转换点。`yy_to_ccyy()` 还在修改年份之前对其是否需要转换（要小于 100）进行确定。用这个方法你就可以无须首先对年份值进行检查而直接传递，不论它是否包含世纪值。使用默认转换点的一些示例调用结果如下：

```
$val = yy_to_ccyy (60);          # 返回 2060  
$val = yy_to_ccyy (1960);        # 返回 1960 (未进行转换)
```

但假设你想使用 50 作为转换点来转化如下的年份值：

```
00 .. 49 -> 2000 .. 2049  
50 .. 99 -> 1950 .. 1999
```

要做到这点，将一个显式的转换点参数传给 `yy_to_ccyy()`：

```
$val = yy_to_ccyy (60, 50);      # 返回 1960  
$val = yy_to_ccyy (1960, 50);    # 返回 1960 (未进行转换)
```

`yy_to_ccyy()` 函数也包含在 `Cookbook_Utils.pm` 库文件之中。

10.30 验证日期和时间合法性

Performing Validity Checking on Date or Time Subparts

问题

一个字符串通过模式检测为一个日期或者时间，但是你想进一步检测，以确保其合法性。

解决方案

将该值分解为子串，并对每个部分实行适当的取值范围检测。

讨论

模式匹配对于时间和日期检测可能是不够的。例如，像 `1947-15-19` 这样的值符合日期类型值的模式，但是这不是一个合法的日期值。如果你想进行更严格的值检测，需要结合模式匹配和取值范围检测。分别取出年、月和日的值，然后看它们是否都在合法的取值范围内。年应该小于 9999 (MySQL 日期值的上限是 `9999-12-31`)，月份值应该在 1 到 12 的范围内，而日的值应该在从 1 到当月包含天数之间。后面的部分是最有意思的：这与每一



个具体的月份相关，而对于二月份，这也与具体的年份相关，因为（二月份的天数）会因闰年而变化。

假设你正在检测基于 ISO 格式的日期值。在第 10.25 节中我们使用过一个 `Cookbook_Utils.pm` 库中的 `is_iso_date()` 函数来对一个日期字符串进行模式匹配，并将它分解为部分值：

```
my $ref = is_iso_date ($val);
if (defined ($ref))
{
    # $val 匹配 ISO 格式
    # check its subparts using $ref->[0] through $ref->[2]
}
else
{
    # $val didn't match ISO format pattern
}
```

如果这个值与 ISO 日期格式不匹配，那么 `is_iso_date()` 函数返回 `undef`。否则，该函数返回一个字符串引用，其中包含年、月和日值（注 1）。为了进一步检测日期部分，将其（返回结果）传递给另一个库函数：

```
$valid = is_valid_date ($ref->[0], $ref->[1], $ref->[2]);
```

或者以更加简洁的方式：

```
$valid = is_valid_date (@{$ref});
```

`is_valid_date()` 以如下方式来检测日期值的每一部分：

```
sub is_valid_date
{
    my ($year, $month, $day) = @_;
    # 年必须是非负值，月和日必须是正值。
    return 0 if $year < 0 || $month < 1 || $day < 1;
    # 分别检查最大值限制
    return 0 if $year > 9999;
    return 0 if $month > 12;
    return 0 if $day > days_in_month ($year, $month);
    return 1;
}
```

`is_valid_date()` 函数需要分开的年、月和日值，而不是一个日期字符串。这迫使你在调用它之前首先将待处理值分解为部分值（年、月、日），但也使它适用于更多上下文中。例如，你可以在调用 `is_valid_date()` 前，将 12 February 2003 中的月份值转换为对应的数值表示来验证这个日期的合法性。`is_valid_date()` 函数以一个符合特定格式的

注 1: `Cookbook_Utils.pm` 文件还包含匹配美国或英国格式日期的程序 `is_mmddyy_date()` 和 `is_ddmmyy_date()`，它们返回 `undef` 或者一个包含日期各个部分的数组的引用。（这些部分总是以年、月、日的顺序返回，而不是在输入日期字符串中所出现的顺序。）

字符串作为参数并不常见。

`is_valid_date()` 使用一个辅助函数 `days_in_month()` 来判定所要处理的日期所处月份内有多少天。`days_in_month()` 函数同时需要年和月作为参数，因为如果是 2 月 (February)，其天数取决于当年是否是闰年。这就意味着你必须传递一个四位数的年份值：2 位数的年份值在考虑到世纪时并不明确，就不可能正确的判断闰年，如第 6.17 节讨论的。`days_in_month()` 和 `is_leap_year()` 函数基于直接取自那一节所用的技术：

```
sub is_leap_year
{
my $year = $_[0];

return (($year % 4 == 0) && (((($year % 100) != 0) || ($year % 400) == 0));
}

sub days_in_month
{
my ($year, $month) = @_;
my @day_tbl = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
my $days = $day_tbl[$month-1];

# 闰年的 2 月多 1 天
$days++ if $month == 2 && is_leap_year ($year);
return ($days);
}
```

为了对时间值进行有效性检测，可以适用类似的过程，尽管对每一个部分值的取值范围不一样：小时部分取值为 0 到 23，而分钟和秒则从 0 到 59。这里有一个 `is_24hr_time()` 函数来检测 24 小时格式的时间值：

```
sub is_24hr_time
{
my $s = $_[0];

return undef unless $s =~ /^(\d{1,2})\D(\d{2})\D(\d{2})$/;
return [ $1, $2, $3 ]; # 返回小时、分、秒
}
```

下面的 `is_ampm_time()` 函数接收加上了 AM 或者 PM 后缀的 12 小时制的时间值，将 PM 的时间转换为 24 小时制：

```
sub is_ampm_time
{
my $s = $_[0];

return undef unless $s =~ /^(\d{1,2})\D(\d{2})\D(\d{2})(?:\s*(AM|PM))?\$/i;
my ($hour, $min, $sec) = ($1, $2, $3);
if ($hour == 12 && (!defined ($4) || uc ($4) eq "AM"))
{
    $hour = "00"; # 12:xx:xx AM 是 00:xx:xx
}
elsif ($hour < 12 && defined ($4) && uc ($4) eq "PM")
{
```

```
    $hour += 12; # PM times other than 12:xx:xx
}
return [ $hour, $min, $sec ]; # 返回小时、分钟和秒
}
```

对于不符合模式要求的时间值，两个函数都返回 `undef`。否则，它们都返回一个含有小时、分钟和秒三个元素的数组的引用。



提示：它们并不执行边界检查。要达成此目的，将数组传给另一个工具程序 `is_valid_time()`。

10.31 编写时间处理工具

Writing Date-Processing Utilities

问题

有一个你需要频繁使用的时间处理操作，所以你需要编写一个工具来给你工作。

解决方案

本节的处理方法中的工具提供了一些例子，来说明如何做到这一点。

讨论

由于日期值的特性，你可能会发现需要一次又一次的编写日期转换（程序）。本节列举了一些适合不同目的的转换器的例子：

- `isoize_date.pl` 读取一个含有 U.S. 格式的日期值 (`MM-DD-YY`) 并将它们转换为 ISO 格式。
- `cvt_date.pl` 将日期值在 ISO、U.S. 和 British 格式之间转换。它比 `isoize_date.pl` 更加通用但是要求你告诉它使用什么格式的输入以及以什么格式输出。
- `monddccyy_to_iso.pl` 接收类似 `Feb. 6, 1788` 格式的日期值，并将其转换为 ISO 格式。它说明了如何将日期值中的数值部分转换到 MySQL 能够理解的格式。

三个脚本程序都在 `recipes` 发行中的 `transfer` 目录下。它们都假定数据文件都使用制表符定界，换行符结束的格式。（如果你需要处理含有其他格式的文件，那要先用 `cvt_file.pl`。）

我们的第一个日期处理工具，`isoize_date.pl`，接收 U.S. 格式日期并将它们转换为 ISO 格式。

你会发现它模仿了第 10.21 节中所示的输入处理循环过程，其中加入了其他内容来实现一个特殊类型的转化：

```
#!/usr/bin/perl
# isoize_date.pl - 读取输入数据，查找匹配日期模式的值，将它们转化成 ISO 格式。
# 同时使用 70 为转换点，将 2 位数值年份转换为 4 位数值年份。

# 本程序默认寻找 MM-DD-[CC]YY 格式的日期

# 假设输入行以制表符定界，换行符结束

# 并不检查日期是否合法（例如，不会对 13-49-1928 报错）

use strict;
use warnings;

# 转换点，2 位数值年份若大于它则被认为是 19XX（小于改转换点会被当成 20XX）
my $transition = 70;

while (<>)
{
    chomp;
    my @val = split (/^\t/, $_, 10000); # split, preserving all fields
    for my $i (0 .. @val - 1)
    {
        my $val = $val[$i];
        # 寻找 MM-DD-[CC]YY 格式的字符串
        next unless $val =~ /^(\d{1,2})\D(\d{1,2})\D(\d{2,4})$/;

        my ($month, $day, $year) = ($1, $2, $3);
        # 为将日期解释为 DD-MM-[CC]YY，用下面的来替换之前的行
        # line with the following one:
        #my ($day, $month, $year) = ($1, $2, $3);

        # 将 2 位数值年份转换为 4 位数值年份，然后更新数组中的值。
        $year += ($year >= $transition ? 1900 : 2000) if $year < 100;
        $val[$i] = sprintf ("%04d-%02d-%02d", $year, $month, $day);
    }
    print join ("\t", @val) . "\n";
}
```

如果你想给 isoize_date.pl 输入如下的一个输入文件：

Fred	04-13-70
Mort	09-30-69
Brit	12-01-57
Carl	11-02-73
Sean	07-04-63
Alan	02-14-65
Mara	09-17-68
Shepard	09-02-75
Dick	08-20-52
Tony	05-01-60

它会产生如下结果：

Fred	1970-04-13
Mort	2069-09-30
Brit	2057-12-01
Carl	1973-11-02
Sean	2063-07-04
Alan	2065-02-14
Mara	2068-09-17
Shepard	1975-09-02
Dick	2052-08-20
Tony	2060-05-01

isoize_date.pl 提供了一个特定的功能：它只是把 U.S. 格式转换为 ISO 格式。它不对日期的每个组成部分做有效性检测或者允许指定用来增加世纪值的转换点（the transition point）。一个更通用的工具也许更加有用。下面的脚本，cvt_date.pl，扩展了 isoize_date.pl 的功能；可以识别以 ISO、U.S. 或 British 格式输入的日期值，并在它们之间任意转换。它也能把两位数表示的年份转换为四位数表示，使你能够指定（世纪）转换点，并且能对错误日期作出警告。同样的，它能被其他程序用作 MySQL 导入的预处理输入，或者对 MySQL 导出数据进行处理。

cvt_date.pl 识别以下选项：

--ifORMAT = format, --oFORMAT = format, --FORMAT = format

指定日期输入、输出格式或者输入输出格式。默认格式值为 iso；cvt_date.pl 也将任意以 us 或者 br 开头的字符串识别为 U.S. 或者 British 日期格式。

--add-century

将 2 位数年份转换为 4 位数。

--columns = column_list

仅对指定列名中的日期进行转化。默认情况下，cvt_date.pl 处理所有列中的日期。如果使用了本选项，column_list 应为一列或者多列下标的列表，或者是以逗号分隔的列范围列表。（m-n 指定的范围是从第 m 到第 n 列。）列下标从 1 开始。

--transition = n

指定从 2 位数年份到 4 位数年份的转换点。默认转换点是 70。这个选项启动了 --add-century。

--warn

警告错误日期。（注意，如果是 2 位数年份的日期值而你没有使用 --add-century，这个选项可能产生假的警告信息，因为这种情况下闰年检测并不一定准确。）

我在这里不想展示 cvt_date.pl 的代码（代码的很大一部分用来处理命令行选项），但是如果你愿意你可以自己查看源码。作为 cvt_date.pl 如何工作的例子，假设你有一个含有以下内容的 newdata.txt 文件：

```
name1 01/01/99 38
name2 12/31/00 40
name3 02/28/01 42
name4 01/02/03 44
```

用 cvt_date.pl 处理这个文件，加上选项说明日期采用 U.S. 格式并加上世纪值，产生结果为：

```
% cvt_date.pl --ifformat=us --add-century newdata.txt
name1 1999-01-01 38
name2 2000-12-31 40
name3 2001-02-28 42
name4 2003-01-02 44
```

生成日期采用 British 格式，并且不对年份值做转换，如下处理：

```
% cvt_date.pl --ifformat=us --offormat=br newdata.txt
name1 01-01-99 38
name2 31-12-00 40
name3 28-02-01 42
name4 02-01-03 44
```

cvt_date.pl 显然不知道每一个数据列的含义。如果你有一个非日期数据列，其中的值符合日期值的模式，它(cvt_date.pl)也将重写这一列。为了应对这一情况，指定一个--columns 选项来限定 cvt_date.pl 将要处理的列范围。

isoize_date.pl 和 cvt_date.pl 都只处理纯数字格式的日期。但是数据文件中的日期值常常以不同格式记录，这就需要一个特殊用途的脚本来处理它们。假设一个输入文件含有以下格式的日期（这样表示的日期是 U.S. 各州被联邦承认的日期格式）：

```
Delaware      Dec. 7, 1787
Pennsylvania  Dec 12, 1787
New Jersey    Dec. 18, 1787
Georgia       Jan. 2, 1788
Connecticut   Jan. 9, 1788
Massachusetts Feb. 6, 1788
Maryland      Apr. 28, 1788
South Carolina May 23, 1788
New Hampshire Jun. 21, 1788
Virginia      Jun 25, 1788
...
...
```

这样的日期由一个三个字母的月份缩写（可能后面加了句号）、数字形式的在本月的第几天、一个逗号和数字形式的年份组成。为了将该文件导入 MySQL，你需要将日期转换为 ISO 格式，得到一个看起来像下面这样的文件：

```
Delaware      1787-12-07
Pennsylvania  1787-12-12
New Jersey    1787-12-18
Georgia       1788-01-02
Connecticut   1788-01-09
```

```

Massachusetts 1788-02-06
Maryland      1788-04-28
South Carolina 1788-05-23
New Hampshire 1788-06-21
Virginia      1788-06-25
...

```

这是一类专门的转换，尽管这种常见的问题（将一个特定日期转换为 ISO 格式）不是非常罕见。为了执行转换，通过与特定模式匹配的值来确定（要处理的）日期，将月份名称映射到对应的数值，并且对结果重新格式化。下面的脚本，monddccyy_to_iso.pl，说明了如何做到这一点：

```

#!/usr/bin/perl
# monddccyy_to_iso.pl - 将日期从 mon[.] dd,ccyy 格式转换为 ISO 格式

# 假设输入以制表符定界，换行符结束

use strict;
use warnings;

my %map =  # 将 3 字符月份缩写匹配到数值月份
(
    "jan" => 1, "feb" => 2, "mar" => 3, "apr" => 4, "may" => 5, "jun" => 6,
    "jul" => 7, "aug" => 8, "sep" => 9, "oct" => 10, "nov" => 11, "dec" => 12
);

while (<>)
{
    chomp;
    my @val = split (/\\t/, $_, 10000);    # 划分，保存所有域
    for my $i (0 .. @val - 1)
    {
        # 如果值匹配模式，则对其重新格式化；否则假设它非要求格式的日期，放置不动
        if ($val[$i] =~ /(^[^.]+)\.?(\\d+),\\s*(\\d+)/)
        {
            # 使用小写月份名称
            my ($month, $day, $year) = (lc($1), $2, $3);
            if (exists($map{$month}))
            {
                $val[$i] = sprintf ("%04d-%02d-%02d",
                                    $year, $map{$month}, $day);
            }
            else
            {
                # 警告，但并不重新格式化
                warn "$val[$i]: bad date?\n";
            }
        }
        print join ("\t", @val) . "\n";
    }
}

```

这个脚本只做了重新格式化，它没有验证日期合法性。为了实现那个功能，修改该脚本，通过在 `use warnings` 一行后加上下面的语句来使用 `Cookbook_Utils.pm` 模块：

```
use Cookbook_Utils;
```

这使脚本能够调用模块中的 `is_valid_date()` 函数。修改下面的语句来使用它 (`is_valid_date()` 函数)：

```
if (exists ($map{$month}))
```

修改为：

```
if (exists ($map{$month}) && is_valid_date ($year, $map{$month}, $day))
```

10.32 使用不完整的日期

Using Dates with Missing Components

问题

在你的数据中的日期值不完整；也就是说，它们缺少一些部分值（年、月或日）。

解决方案

MySQL 能够用 0 代替缺失的部分，以 ISO 格式表示它们。

讨论

一些应用程序使用不完整的日期值。例如，你可能需要处理形如 Mar/2001 的输入值，其中只含有月和年。在 MySQL 中，可以使用 ISO 格式日期来表示这样的输入值，其中缺失的部分用 0 代替。（值 Mar/2001 可以存储为 2001-03-00。）为了把 month/year 格式值转换为 ISO 格式以用于导入 MySQL，定义一个哈希表把月份名称映射为对应的数值：

```
my %map = # 将 3 字符的月份名称映射为对应的数值
(
    "jan" => 1, "feb" => 2, "mar" => 3, "apr" => 4, "may" => 5, "jun" => 6,
    "jul" => 7, "aug" => 8, "sep" => 9, "oct" => 10, "nov" => 11, "dec" => 12
);
```

现在可以这样转换每一个输入值：

```
if ($val =~ /^[a-z]{3})\//(\d{4})$/i)
{
    my ($m, $y) = (lc($1), $2); # 使用小写的月份名称
    $val = sprintf ("%04d-%02d-00", $y, $map{$m})
}
```

把结果保存到 MySQL 之后，你可以通过一个用 `DATE_FORMAT()` 重写日期值的 `SELECT` 语句查询保存的值，并用原有的 month/year 格式来显示它们：

```
DATE_FORMAT(date_val, '%b/%Y')
```

那些采用严格的 SQL 模式但是在日期值中又要求 0 值部分的应用程序要小心避免设置 NO_ZERO_IN_DATE SQL 模式，那（该模式）会导致服务器认为这样（含有 0 值部分）的日期值是无效的。

10.33 导入非 ISO 格式日期值

Importing Non-ISO Date Values

问题

要导入的数据不是 MySQL 期望的 ISO (*CCYY-MM-DD*) 格式。

解决方案

在导入 MySQL 之前使用外部工具将（待处理）日期值转换为 ISO 格式（这里使用 *cvt_date.pl*）。或者在数据导入数据库之前用 LOAD DATA 功能作为输入数据的预处理过程。

讨论

假设你有一张表其中包含三列，*name*、*date* 和 *value*，其中 *date* 列是 DATE 类型要求的 ISO 格式数据 (*CCYY-MM-DD*)。同时假设你接收到一个数据文件 *newdata.txt* 要导入到该表中，但是（数据文件）的内容如下：

name1	01/01/99	38
name2	12/31/00	40
name3	02/28/01	42
name4	01/02/03	44

这里日期值是 *MM/DD/YY* 格式，并且必须转换为 ISO 格式以便在 MySQL 中以 DATE 类型值保存。一种办法是是用本章前面说明过的 *cvt_date.pl* 文件处理这个（数据）文件：

```
* cvt_date.pl --ifformat=us --add-century newdata.txt > tmp.txt
```

然后你可以把 *tmp.txt* 文件装载到数据库表中。这个工作也可以通过使用 SQL 来执行再格式化操作，即完全由 MySQL 完成，而不需外部工具。如第 10.11 节中讨论的，LOAD DATA 可以在把输入数据插入之前预先对其进行处理。把那个功能应用于这里的问题，使用 STR_TO_DATE() 函数（第 6.2 节）来解释输入日期值，数据重写的 LOAD DATA 语句看起来像这样：

```
mysql> LOAD DATA LOCAL INFILE 'newdata.txt'
-> INTO TABLE t (name,@date,value)
-> SET date = STR_TO_DATE(@date,'%m/%d/%y');
```

通过 `STR_TO_DATE()` 函数中的`%y` 格式表示符, MySQL 自动将 2 位数年份转换为 4 位数年份, 结果是使得最初的 `MM/DD/YY` 值转换成最终为 `CCYY-MM-DD` 格式。导入之后的结果数据看起来像这样:

name	date	value
name1	1999-01-01	38
name2	2000-12-31	40
name3	2001-02-28	42
name4	2003-01-02	44

这个过程假设 MySQL 自动把 2 位数年份转换为 4 位数生成的是正确的世纪值。这就意味着日期值的年份部分必须处于从 1970 到 2069 的范围内。如果不是这样, 你需要用其他方法转换年份值。(参考第 10.30 节中其他方法)

如果日期值采用了 `STR_TO_DATE()` 不能处理的格式, 也许你可以写一个存储函数来处理它们并返回 ISO 格式日期值。那种情况下, `LOAD_DATA` 语句看起来像这样, 其中 `my_date_interp()` 是存储函数的名称:

```
mysql> LOAD DATA LOCAL INFILE 'newdata.txt'  
-> INTO TABLE t (name,@date,value)  
-> SET date = my_date_interp(@date);
```

10.34 使用非 ISO 格式导出日期值

Exporting Dates Using Non-ISO Formats

问题

你想导出 MySQL 默认使用的 ISO (`CCMM-MM-DD`) 格式之外的日期格式。这也许是在把日期从 MySQL 导出到不能处理 ISO 格式的应用程序时的一个需求。

解决方案

在把数据从 MySQL 导出之后, 使用一个外部工具把日期值重写为非 ISO 格式 (这里使用 `cvt_date.pl`)。或者使用 `DATE_FORMAT()` 函数在导出操作过程中重写 (日期) 值。

讨论

假设你想把数据从 MySQL 导出到一个不能处理 ISO 格式日期的应用程序中。一种方法是把数据导出到一个文件, 保留日期的 ISO 格式。然后使用类似 `cvt_date.pl` 的工具处理数据文件, 将日期按照需要的格式重写。

另一种方法是用 `DATE_FORMAT()` 函数重写的方式将日期直接导出为所需格式。如果你有如下数据表:

```
CREATE TABLE datetbl
(
    i      INT,
    c      CHAR(10),
    d      DATE,
    dt     DATETIME,
    ts     TIMESTAMP
);
```

同时设想你需要从表中导出数据，但要求任意 DATE、DATETIME 或者 TIMESTAMP 列日期值按照 U.S. 格式重写 (MM-DD-CCYY)。一个使用 DATE_FORMAT() 函数的 SELECT 语句根据需要如下重写日期值：

```
SELECT
    i,
    c,
    DATE_FORMAT(d, '%m-%d-%Y') AS d,
    DATE_FORMAT(dt, '%m-%d-%Y %T') AS dt,
    DATE_FORMAT(ts, '%m-%d-%Y %T') AS ts
FROM datetbl
```

这样，如果 datetbl 含有如下行：

3	abc	2005-12-31	2005-12-31 12:05:03	2005-12-31 12:05:03
4	xyz	2006-01-31	2006-01-31 12:05:03	2006-01-31 12:05:03

这个 SQL 语句产生下面这样的输出结果：

3	abc	12-31-2005	12-31-2005 12:05:03	12-31-2005 12:05:03
4	xyz	01-31-2006	01-31-2006 12:05:03	01-31-2006 12:05:03

10.35 导入和导出 NULL 值

Importing and Exporting NULL Values

问题

你不明确如何在数据文件中表示 NULL 值。

解决方案

尝试使用一个其他地方不会出现的值，这样你就可以把 NULL 和其他合法的非 NULL 值区别开。到你导入文件时，查找那个特殊值并将其转换为 NULL。

讨论

没有在数据文件中表示 NULL 值的标准方法，这使得 (NULL 值) 在导入和导出操作中带来了一点麻烦。带来麻烦的原因是 NULL 说明没有任何值，而没有任何值在数据文件中不容易从字面上来表示。最常见的做法是使用一个空列值，但是这对于字符串类型列并不明确，因为无法区分这样表示的 NULL 和一个真正的空字符串。空值给其他类型也可能带来问题。

例如，如果你想使用 LOAD DATA 把一个空值装载到一个数值类型列中，这个空值会被保存成 0 而不是 NULL，因此造成无法区分输入数据中真正的 0。

解决这个问题的常见策略是用一个在数据其他地方不会出现的值来表示 NULL。LOAD DATA 和 mysqlimport 采用同样策略处理这一问题：它们识别 \N 值约定代表 NULL。（当 \N 单独出现，而不是以一个较大值的一部分出现，如 x\N 或 \Nx 等时，该值被解释为 NULL。）例如，如果你用 LOAD DATA 装载如下数据文件，它将 \N 解释为 NULL：

```
str1    13      1997-10-14
str2    \N      2009-05-07
\N      15      \N
\N      \N      1973-07-14
```

但是你可能想把 \N 以外的值解释为 NULL，并且你可能在不同列有不同解释规则。考虑下面的数据文件：

```
str1    13  1997-10-14
str2    -1  2009-05-07
Unknown 15
Unknown -1  1973-07-15
```

第一列包含字符串，并且以 Unknown 表示 NULL。第二列包含整数，且以 -1 代表 NULL。第三列包含日期，且以空串代表 NULL。这样如何处理呢？

为了处理这种情形，使用 LOAD DATA 功能来处理输入值：定义一个数据库列的列表，使用输入值对用户自定义变量赋值，并使用 SET 子句将特殊值映射到真正的 NULL 值。如果数据文件名为 has_null.txt，下面的 LOAD DATA 语句正确解释其内容：

```
mysql> LOAD DATA LOCAL INFILE 'has_nulls.txt'
-> INTO TABLE t (@c1,@c2,@c3)
-> SET c1 = IF(@c1='Unknown',NULL,@c1),
->       c2 = IF(@c2=-1,NULL,@c2),
->       c3 = IF(@c3='',NULL,@c3);
```

导入之后的结果数据如下：

c1	c2	c3
str1	13	1997-10-14
str2	NULL	2009-05-07
NULL	15	NULL
NULL	NULL	1973-07-15

前面的讨论适合于处理导入 MySQL 的 NULL 值，但是数据在不同方向迁移时仍然需要考虑 NULL 值，例如从 MySQL 到其他程序。下面是一些例子：

- SELECT ... INTO OUTFILE 将 NULL 值记录为\n。其他程序能理解这样的转换么？
如果不行，你需要把\n 转换为程序能够理解的其他值。例如，SELECT 语句使用如下语句导出数据列：

```
IFNULL (col_name, 'Unknown')
```

- 你可以简单的以 batch 模式使用 mysql 来生成制表符定界的输出(如第 10.13 节所示)，但是这样做了一个问题是 NULL 值在输出中表现为“NULL”。如果这个单词在输出的其他内容中不出现，你可以再次处理输出结果把这个单词转换为其他适当值。例如，你能使用一行 sed 命令：

```
% sed -e "s/NULL/\\"N/g" data.txt > tmp
```

如果单词“NULL”出现在一个表示非 NULL 值的地方，那这就会很不明确，你应该换一种方式来导出不同的数据。例如，你的导出语句可以使用 IFNULL() 将 NULL 转换为其他值。

10.36 根据数据文件猜测表结构

Guessing Table Structure from a Datafile

问题

别人给你一个数据文件跟你说，“你好，帮我把这个输入到 MySQL 里”。但是保存数据的表还不存在。

解决方案

自己写 CREATE TABLE 语句。或者通过检测数据文件内容使用工具猜测表结构。

讨论

有时你需要向 MySQL 中导入数据，其对应的数据表不存在。你可以基于自己对数据文件内容的理解，来自己创建数据表。或者你可以用 guess_table.pl 来避免这些工作，(该文件)位于在 recipes 发行中的 transfer 目录下。guess_table.pl 读取数据文件来分析其内容，然后试图生成一个合适的 CREATE TABLE 语句来匹配数据文件内容。这个脚本肯定不是完美的，因为有些列内容有时候很模糊。(例如，一个含有几个特定字符串的列可能是 VARCHAR 或者 ENUM 类型)。尽管如此，处理 guess_table.pl 脚本生成的 CREATE TABLE 语句要比从

头开始写整个语句容易得多。这个工具也有一个诊断函数，但这不是首要用途。例如，你可能认为某一列只含有数字，但是如果 `guess_table.pl` 认为它应该使用 `VARCHAR` 类型创建列，这就告诉你这一列中至少含有一个非数值形式的值。

`guess_table.pl` 认为其输入是用制表符定界，换行符结束的格式。它同时也假设输入的合法性，因为任何试图基于可能有瑕疵的数据的猜测都注定失败。这就意味着，例如，如果一个 `date` 列也要同样地被识别，它应该符合 ISO 格式。否则，`guess_table.pl` 会将其识别为 `VARCHAR` 列。如果一个数据文件不满足这些假设，你能用第 10.18 节和第 10.31 节中讨论的 `cvt_file.pl` 和 `cvt_date.pl` 工具先对其重写格式。

`guess_table.pl` 识别以下选项：

`--labels`

将输入的第一行解释为一个包含列标签的数据表行，并将其用为表的列名。如果省略这个标签，`guess_table.pl` 使用默认列名 `c1, c2` 等。

注意，如果数据文件包含一行标签，同时你忘记了指定这个选项，`guess_table.pl` 将会把标签行当作数据值来处理。可能的结果是这个脚本把所有列识别为 `VARCHAR` 类型列（甚至其他行只含有数值或者日期值），因为列中出现了一个非日期或非数字的值。

`--lower, --upper`

将 `CREATE TABLE` 语句中的列名强制为大写或者小写。

`--quote-name, --skip-quote-name`

在 `CREATE TABLE` 语句中对表或者列标识符加上或者省略'（单引号）（例如，'mytb1'）。如果一个标识符是保留字的话这个选项就很有用。默认的是给标识符加上引号。

`--report`

生成一个报告而不是一个 `CREATE TABLE` 语句。这个脚本显示了它收集的每一列的信息。

`--table = tb1_name`

指定 `CREATE TABLE` 语句中的表名。默认表名是 `t`。

下面是 `guess_table.pl` 如何工作的一个例子。假设一个名为 `stockdat.csv` 的 CSV 格式文件，其内容如下：

```
commodity,trade_date,shares,price,change
sugar,12-14-2006,1000000,10.50,-.125
oil,12-14-2006,96000,60.25,.25
wheat,12-14-2006,2500000,8.75,0
gold,12-14-2006,13000,103.25,2.25
```

```
sugar,12-15-2006,970000,10.60,.1  
oil,12-15-2006,105000,60.5,.25  
wheat,12-15-2006,2370000,8.65,-.1  
gold,12-15-2006,11000,101,-2.25
```

第一行说明了列标签，其后的几行包含数据记录，每行一条记录。`trade_date` 列中的值为 `date` 类型，但它们是 `MM-DD-CCYY` 格式数据，而不是 MySQL 所期望的 ISO 格式。`cvt_date.pl` 能把这些日期值转换为 ISO 格式。然后 `cvt_date.pl` 和 `guess_table.pl` 都要求输入文件满足制表符定界，换行符结束的格式。因此，首先使用 `cvt_file.pl` 将输入转换为制表符定界，换行符结束的格式然后我们能使用 `cvt_date.pl` 对输入进行转换：

```
% cvt_file.pl --ifformat=csv stockdat.csv > tmp1.txt  
% cvt_date.pl --ifformat=us tmp1.txt > tmp2.txt
```

然后将输出文件 `tmp2.txt` 导入到 `guess_table.pl`:

```
% guess_table.pl --labels --table=stocks tmp2.txt > stocks.sql
```

`guess_table.pl` 写到 `stock.sql` 的 `CREATE TABLE` 语句如下：

```
CREATE TABLE `stocks`  
(  
  `commodity` VARCHAR(5) NOT NULL,  
  `trade_date` DATE NOT NULL,  
  `shares` INT UNSIGNED NOT NULL,  
  `price` DOUBLE UNSIGNED NOT NULL,  
  `change` DOUBLE NOT NULL  
) ;
```

`guess_table.pl` 生成的语句基于下面的推论：

- 如果一列只包含数字类型值，且没有值包含小数点，或者 `DOUBLE` 等其他值，那么它将被假定为 `INT` 类型。
- 如果一个数字类型列中只有非负数值，那么该列可能为 `UNSIGNED`。
- 如果一列不包含空值，`guess_table.pl` 就假设该列值 `NOT NULL`。
- 不能被归类为数字类型或者日期类型的列被处理为 `VARCHAR` 类型列，其长度定义为该列中出现的最长值长度。

你可能想编辑 `guess_table.pl` 生成的 `CREATE TABLE` 语句，来做一些修改，例如增加字符域长度，将 `VARCHAR` 改为 `CHAR`，或者添加索引。修改 SQL 语句的另一个原因是，如果某一列名为 MySQL 保留字，你可以重命名该列。

使用 `guess_table.pl` 生成的语句来创建表：

```
% mysql cookbook < stocks.sql
```

然后你可以把数据文件导入表中（忽略创建列标签的行）：

```
mysql> LOAD DATA LOCAL INFILE 'tmp2.txt' INTO TABLE stocks
-> IGNORE 1 LINES;
```

完成导入之后的结果数据如下：

```
mysql> SELECT * FROM stocks;
+-----+-----+-----+-----+-----+
| commodity | trade_date | shares | price | change |
+-----+-----+-----+-----+-----+
| sugar     | 2006-12-14 | 1000000 | 10.5  | -0.125 |
| oil       | 2006-12-14 | 96000   | 60.25 | 0.25   |
| wheat     | 2006-12-14 | 2500000 | 8.75  | 0        |
| gold      | 2006-12-14 | 13000   | 103.25 | 2.25   |
| sugar     | 2006-12-15 | 970000  | 10.6   | 0.1     |
| oil       | 2006-12-15 | 105000  | 60.5   | 0.25   |
| wheat     | 2006-12-15 | 2370000 | 8.65  | -0.1    |
| gold      | 2006-12-15 | 11000   | 101    | -2.25  |
+-----+-----+-----+-----+-----+
```

10.37 在 MySQL 和 Access 之间交换数据

Exchanging Data Between MySQL and Microsoft Access

问题

你想在 MySQL 和 Access 之间交换信息。

解决方案

为了使用保存在 MySQL 中的信息，可以从 Access 直接连接到 MySQL 服务器。为了把信息从 Access 迁移到 MySQL，可以使用能够直接完成数据迁移的工具，或者先从 Access 导出表到一个文件，然后再将这些文件导入到 MySQL。

讨论

MySQL 和 Access 都支持 ODBC，所以你可以直接从 Access 连接到 MySQL。通过建立一个 ODBC 连接，Access 成为了一个前端（客户端），用它你可以使用 MySQL 数据库。在 [mysql.com Connector/ODBC](http://www.mysql.com/Connector/ODBC) 中有很多有用的信息：

<http://www.mysql.com/products/connector/odbc/>

在 DevShed 站点，W.J.Gilmore 的文章里可以找到建立 ODBC 并从 Access 通过 ODBC 连接到 MySQL 这个过程的最佳描述：

<http://www.devshed.com/c/a/MySQL/MySQL-and-ODBC/>

如果目前你的数据表在 Access 中，并且你想把它们迁移到 MySQL，你需要在 MySQL 创建一个表来保存信息，然后把 Access 信息导入到这些表中。Gilmore 的文章说明了如何做到这一点。

你也可以选择把 Access 表导出到文件，然后在把这些文件导入到 MySQL。这也许是必要的，例如，如果你的 MySQL 服务器运行在另一台机器上，并且不允许从你的 Windows 机器建立连接。如果你选择这个解决方案，那你需要注意的是所使用的文件格式，日期格式转换，以及如果 MySQL 中对应的表不存在的话如何为这些数据创建表。本章前面讨论的脚本（例如 `cvt_file.pl`, `cvt_date.pl` 和 `guess_table.pl`）在解决这些问题时可以提供帮助。从 Access 导入表到 MySQL 的过程可以描述如下：

1. 以某种文本格式从 Access 导出表，可能包括列标签。你可能要为期望制表符定界，换行符结束的格式的工具转换导出文件的格式，(因此) 最好采用期望的的格式导出文件。
2. 如果表包含日期值，而你不想以 ISO 格式导出，你需要帮助 MySQL 转换它们。用 `cvt_date.pl` 完成。
3. 如果你需要导入 Access 数据的 MySQL 表不存在，要创建它们。`guess_table.pl` 工具在生成 `CREATE TABLE` 语句时也许有用。
4. 用 `LOAD DATA` 或者 `mysqlimport` 导入数据文件到 MySQL。

10.38 在 MySQL 和 Microsoft Excel 之间交换数据

Exchanging Data Between MySQL and Microsoft Excel

问题

你想在 MySQL 和 Excel 之间交换数据。

解决方案

你使用的编程语言可能提供模块来简化这一任务。例如，有 Perl 模块读写 spreadsheet 文件。你能使用它们来构造你自己的数据迁移工具。

讨论

如果你需要把 Excel 文件导入到 MySQL 中，首先查询在你的编程语言中能够帮你完成这一任务的模块。例如，你可以安装下面的模块以后在 Perl 脚本中读写 spreadsheet：

- `Spreadsheet::ParseExcel::Simple` 提供了一个简单易用的接口用于读取 Excel 接口。
- `Spreadsheet::WriteExcel::Simple` 使你能够创建 Excel spreadsheet 格式文件。

这些 Excel 模块可以从 Perl CPAN 获取。（它们实际上用作你需要先安装的其他模块的客户端）安装这些模块之后，使用如下命令阅读它们的文档：

```
% perldoc Spreadsheet::ParseExcel::Simple  
% perldoc Spreadsheet::WriteExcel::Simple
```

使用这些模块可以相当方便地写出一些简短的脚本在 spreadsheets 和制表符定界的文件格式之间进行转换。结合在 MySQL 中导入和导出数据的技术，这些脚本可以帮助你把 spreadsheets 的内容导入到 MySQL，反之亦然。直接使用它们，或者修改它们以适合你的需求。

下面的脚本，from_excel.pl，读取一个 Excel spreadsheet 并将其转换为制表符定界格式：

```
#!/usr/bin/perl  
# form_excel.pl - 读取 Excel 电子表格，将制表符定界，  
# 换行符结束的输出写入到标准输出中  
  
use strict;  
use warnings;  
use Spreadsheet::ParseExcel::Simple;  
  
@ARGV or die "Usage: $0 excel-file\n";  
  
my $xls = Spreadsheet::ParseExcel::Simple->read ($ARGV[0]);  
foreach my $sheet ($xls->sheets ())  
{  
    while ($sheet->has_data ())  
    {  
        my @data = $sheet->next_row ();  
        print join ("\t", @data) . "\n";  
    }  
}
```

to_excel.pl 脚本执行读取一个制表符定界文件并写入 Excel 格式的转换操作：

```
#!/usr/bin/perl  
# to_excel.pl - 读取制表符定界，换行符结束的输入，  
# 将 Excel 格式的输出写入到标准输出中  
  
use strict;  
use warnings;  
use Spreadsheet::WriteExcel::Simple;  
  
my $ss = Spreadsheet::WriteExcel::Simple->new ();  
  
while (<>) # 读取输入的每一行  
{  
    chomp;  
    my @data = split (/[\t]/, $_, 10000); # 划分，保存所有域  
    $ss->write_row (\@data); # 将行写入到电子表格中  
}
```

```
binmode (STDOUT);
print $ss->data (); # 写入电子表格
```

`to_excel.pl` 假设输入为制表符定界，换行符结束的格式。结合 `cvt_file.pl` 文件使用来处理其他格式的文件。

另一个 Excel 相关的 Perl 模块，`Spreadsheet::WriteExcel::FromDB`，使用一个 DBI 连接从数据库表中读取数据，并写入到 Excel 格式。下面是一个简单的脚本，它把 MySQL 表导出为一个 Excel spreadsheet：

```
#!/usr/bin/perl
# mysql_to_excel.pl - 给定一个数据库和表名称,
# 将表以 Excel 格式导入到标准输出中

use strict;
use warnings;
use DBI;
use Spreadsheet::ParseExcel::Simple;
use Spreadsheet::WriteExcel::FromDB;

# ...处理命令行选项 (略去)

@ARGV == 2 or die "$usage\n";
my $db_name = shift (@ARGV);
my $tbl_name = shift (@ARGV);

# ...连接数据库 (略去)

my $ss = Spreadsheet::WriteExcel::FromDB->read ($dbh, $tbl_name);
binmode (STDOUT);
print $ss->as_xls ();
```

三个工具中的每一个都输出到标准输出，你可以把结果重定向以在一个文件中获取输出内容：

```
% from_excel.pl data.xls > data.txt
% to_excel.pl data.txt > data.xls
% mysql_to_excel.pl cookbook profile > profile.xls
```

10.39 将输出结果导出为 XML

Exporting Query Results as XML

问题

你想把一个查询结果导出为一个 XML 文件。

解决方案

MySQL 自己可以做到这一点，或者你可以编写你自己的导出工具。

讨论

你可以使用 MySQL 根据一个查询结果生成 XML 格式输出（如第 1.20 节）。

你也可以编写你自己的 XML-export 程序。一种方法是执行查询操作，然后输出查询结果，你自己加上所有的 XML 标签。但是更简单的方法是安装一些 Perl 模块，让它们来完成这些操作：

- `XML::Generator::DBI` 通过 DBI 连接执行一个查询，并将结果传递给一个合适的输出 writer。
- `XML::Handler::YAWriter` 提供了一个输出 writer。

下面的脚本，`mysql_to_xml.pl`，类似于 `mysql_to_text.pl`（第 10.17 节），但是不接受其中的一些选项，如 quote 或者分隔符等。读取 XML 文件不需要这些（选项），因为标准的 XML 解析程序承担了这些工作。`mysql_to_xml.pl` 能处理的选项是：

```
--execute = query, -e query
```

执行 `query`（一个查询），然后输出结果。

```
--table = tbl_name, -t tbl_name
```

输出指定表的内容。这与使用`--execute`指定一个内容为 `SELECT * FROM tbl_name` 的查询是等价的。

如果需要，你也可以指定标准连接函数选项，如`--user` 或`--host` 等。命令行的最后一个参数可以是数据库名，除非查询中暗示了数据库名。

假设你想导出一个如下的实验数据表 `expt`：

```
mysql> SELECT * FROM expt;
+-----+-----+-----+
| subject | test | score |
+-----+-----+-----+
| Jane   | A    |    47 |
| Jane   | B    |    50 |
| Jane   | C    |    NULL |
| Jane   | D    |    NULL |
| Marvin | A    |    52 |
| Marvin | B    |    45 |
| Marvin | C    |    53 |
| Marvin | D    |    NULL |
+-----+-----+-----+
```

做到那一点，使用下面的任意一条命令调用 `mysql_to_xml.pl`：

```
% mysql_to_xml.pl --execute="SELECT * FROM expt" cookbook > expt.xml
% mysql_to_xml.pl --table=cookbook.expt > expt.xml
```

生成的 XML 文件，`expt.xml`，如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<rowset>
```

```
<select query="SELECT * FROM expt">
<row>
<subject>Jane</subject>
<test>A</test>
<score>47</score>
</row>
<row>
<subject>Jane</subject>
<test>B</test>
<score>50</score>
</row>
<row>
<subject>Jane</subject>
<test>C</test>
</row>
<row>
<subject>Jane</subject>
<test>D</test>
</row>
<row>
<subject>Marvin</subject>
<test>A</test>
<score>52</score>
</row>
<row>
<subject>Marvin</subject>
<test>B</test>
<score>45</score>
</row>
<row>
<subject>Marvin</subject>
<test>C</test>
<score>53</score>
</row>
<row>
<subject>Marvin</subject>
<test>D</test>
</row>
</select>
</rowset>
```

表的每一行被记录为一个`<row>`元素。每一行中，列名和列值被记录为元素名和元素值，每个元素代表一列。注意输出结果中省略了`NULL`值。

这个脚本在处理了命令行参数并连接到 MySQL 服务器之后，使用很少的代码生成了输出结果。`mysql_to_xml.pl` 中的 XML 相关部分就是绑定在所需模块中，以及创建和使用 XML 对象的代码。给定一个数据库句柄`$dbh` 和一个查询字符串`$query`，这个过程不需要太多代码。这部分代码控制`writer` 对象将其结果发送到标准输出，然后把对象连接到 DBI，并执行`sql` 语句：

```

#!/usr/bin/perl
# mysql_to_xml.pl - 给定一个数据库和表名称,
# 将表以 XML 格式导入到标准输出中

use strict;
use warnings;
use DBI;
use XML::Generator::DBI;
use XML::Handler::YAWriter;

# ... 处理命令行选项 (略去)

# ... 连接数据库 (略去)

# 创建输出写入器; “-”表示“标准输出”
my $out = XML::Handler::YAWriter->new (AsFile => "-");
# 在 DBI 和输出写入器之间建立连接
my $gen = XML::Generator::DBI->new (
    dbh => $dbh,                      # 数据库句柄
    Handler => $out,                  # 输出写入器
    RootElement => "rowset"          # 文档根元素
);
# 发起查询并写入 XML
$gen->execute ($stmt);

$dbh->disconnect ();

```

其他语言也可能有库模块执行类似 XML 导出的操作。例如，Ruby DBI::Utils::XMLFormatter 模块有一个 table 方法可以方便地将查询结果导出为 XML。下面是使用它的一个简单脚本：

```

#!/usr/bin/ruby -w
# xmlformatter.rb - 说明 DBI::Utils::XMLFormatter.table 方法

require "Cookbook"

stmt = "SELECT * FROM expt"
# 如果给定了命令行参数用它来过载语句
stmt = ARGV[0] if ARGV.length > 0

begin
    dbh = Cookbook.connect
rescue DBI::DatabaseError => e
    puts "Could not connect to server"
    puts "Error code: #{e.err}"
    puts "Error message: #{e.errstr}"
end

DBI::Utils::XMLFormatter.table(dbh.select_all(stmt))

dbh.disconnect

```

10.40 将 XML 导入 MySQL

Importing XML into MySQL

问题

你想把一个 XML 文档导入到一个 MySQL 表。

解决方案

创建一个 XML 解析器来读取文档。然后使用文档中的记录来构造并执行 INSERT 语句。

讨论

基于能够解析并从中提取出记录内容导入一个 XML。你的具体解决方法取决于该文档是如何记录的。例如，一种格式可以将列名和列值表示为`<column>`元素的属性：

```
<?xml version="1.0" encoding="UTF-8"?>
<rowset>
  <row>
    <column name="subject" value="Jane" />
    <column name="test" value="A" />
    <column name="score" value="47" />
  </row>
  <row>
    <column name="subject" value="Jane" />
    <column name="test" value="B" />
    <column name="score" value="50" />
  </row>
  ...
</rowset>
```

另一种格式使用列名作为元素名，同时列值作为这些元素的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<rowset>
  <row>
    <subject>Jane</subject>
    <test>A</test>
    <score>47</score>
  </row>
  <row>
    <subject>Jane</subject>
    <test>B</test>
    <score>50</score>
  </row>
  ...
</rowset>
```

由于文档结构的多种可能性，这就需要对你将要使用的 XML 文档格式做一些假设。对这里的例子，作者将假设使用上述第二种文件格式。处理这一类文档的一种方法是使用 XML::XPath 模块，它使你能够通过路径表达式引用文档中的元素。例如，路径`//row`选择了文档根目录下所有的`<row>`元素，而路径`*`选择了一个给定元素下所有的孩子元素。你可以结合这些路径和 XML::Path 首先获取所有`<row>`元素的一个列表，然后每行得到一个列列表。

下面的脚本，`xml_to_mysql.pl`，接收三个参数：

```
% xml_to_mysql.pl db_name tbl_name xml_file
```

参数`filename`说明了要导入哪个文档，`database` 和 `table name` 两个参数说明了文件要导入哪张表中。`xml_to_mysql.pl` 处理了命令行参数，连接到 MySQL，然后处理文档：

```
#!/usr/bin/perl
# xml_to_mysql.pl - 将 XML 文件读入 MySQL 中

use strict;
use warnings;
use DBI;
use XML::XPath;

# ...处理命令行选项 (略去)

# ...连接数据库 (略去)

# Open file for reading
my $xp = XML::XPath->new (filename => $file_name);
my $row_list = $xp->find ("//row"); # 查找<row>元素集合
print "Number of records: " . $row_list->size () . "\n";
foreach my $row ($row_list->get_nodelist ()) # 循环遍历行
{
    my @name; # 列名数组
    my @val; # 列值数组
    my $col_list = $row->find ("*"); # 行的子列
    foreach my $col ($col_list->get_nodelist ()) # 循环遍历各列
    {
        # 保存列名和列值
        push (@name, $col->getName ());
        push (@val, $col->string_value ());
    }
    # 构建 INSERT 语句，接着执行它
    my $stmt = "INSERT INTO $tbl_name (
        . join (", ", @name)
        . ") VALUES (
        . join (", ", ("?" x scalar (@val)))
        . ")";
    $dbh->do ($stmt, undef, @val);
}
$dbh->disconnect ();
```

这个脚本创建了一个 XML::XPath 对象，这个对象打开并解析 XML 文档。然后使用路径 `//row` 查询该对象得到`<row>`元素集合。该集合的大小说明了文档中包含的记录数量。

为了处理每行记录，这个脚本使用路径`*`查询每一个行对象的所有子元素。每一个子元素对应到行中的一列；把`*`用作 `get_nodelist()` 的路径，这样做比较方便，因为你无需事先知道有哪些列。`xml_to_mysql.pl` 从获取每一列的名称和值，并将其保存在`@name` 和 `@value` 列中。所有列都处理完之后，这些数组被用来构造一个 `INSERT` 语句，它指定了一行中将要出现的列名，并包含每一个值的占位符。（第 2.5 节讨论了占位符列表的构造）。然后这个脚本执行了 `sql` 语句，把列值传递给 `do()` 函数，将这些值绑定到占位符。

在前面一节中，我们使用 `mysql_to_xml.pl` 导出 `expt` 表的内容到一个 XML 文档。`xml_to_mysql.pl` 可以执行相反的操作，把文档导回 MySQL：

```
% xml_to_mysql.pl cookbook expt expt.xml
```

在它处理文档的过程中，这个脚本生成并执行了下面的语句集合：

```
INSERT INTO expt (subject,test,score) VALUES ('Jane','A','47')
INSERT INTO expt (subject,test,score) VALUES ('Jane','B','50')
INSERT INTO expt (subject,test) VALUES ('Jane','C')
INSERT INTO expt (subject,test) VALUES ('Jane','D')
INSERT INTO expt (subject,test,score) VALUES ('Marvin','A','52')
INSERT INTO expt (subject,test,score) VALUES ('Marvin','B','45')
INSERT INTO expt (subject,test,score) VALUES ('Marvin','C','53')
INSERT INTO expt (subject,test) VALUES ('Marvin','D')
```

注意这些语句并没有插入相同数量的列。MySQL 将缺失的列设为它们的默认值。

10.41 尾声

Epilogue

回忆本章开头设想的情节：

假设你有一个名为 `somedata.csv` 的文件，它包含 comma-separated value (CSV) 格式的 12 列数据。你只想从文件中抽出第 2、11、5 和 9 列，并用来自在一个含有 `name`、`birth`、`height` 和 `weight` 列的 MySQL 表中构造数据库记录行。你要保证 `height` 和 `weight` 是正整数值，并把生日从 `MM/DD/YY` 格式转换为 `CCYY-MM-DD` 格式。你要怎么做呢？

那么，基于本章讨论的技术，你要怎么做呢？

大部分的工作都可以用这里开发的工具完成。你可以用 `cvt_file.pl` 把那个文件转换为制表

符合界格式，用 `yank_col.pl` 按照需要的顺序解析列，然后用 `cvt_date.pl` 把 `date` 列重写为 ISO 格式：

```
% cvt_file.pl --ifformat=csv somedata.csv \
| yank_col.pl --columns=2,11,5,9 \
| cvt_date.pl --columns=2 --ifformat=us --add-century > tmp
```

结果文件，`tmp`，将包含 4 列按顺序表示 `name`、`birth`、`height` 和 `weight` 值。只需要检测其中的 `height` 和 `weight` 列来保证它们是正整数。在 `Cookbook_Utils.pm` 模块文件中的 `is_positive_integer()` 库函数的帮助下，这项工作只需要一个简短的，不会大于输入循环的特殊用途脚本就可以实现：

```
#!/usr/bin/perl
# validate_htwt.pl - 高度/重量验证示例

# 假设输入行以制表符定界，换行符结束

# 输入列及其上执行的动作如下所示：
# 1:name; 原样输出
# 2:birth; 原样输出
# 3:height; 验证是否为正整数
# 4:weight; 验证是否为正整数

use strict;
use warnings;
use Cookbook_Utils;

while (<>)
{
    chomp;
    my ($name, $birth, $height, $weight) = split (/\\t/, $_, 4);
    warn "line $.:height $height is not a positive integer\\n"
        if !is_positive_integer ($height);
    warn "line $.:weight $weight is not a positive integer\\n"
        if !is_positive_integer ($weight);
}

validate_htwt.pl 脚本不会生成任何输出（警告信息除外），因为它不需要重新格式化任何输入值。假设 tmp 正确地通过了确认，它能够通过一个 LOAD DATA 语句装载如 MySQL：
```

```
mysql> LOAD DATA LOCAL INFILE 'tmp' INTO TABLE tbl_name;
```


生成和使用序列

Generating and Using Sequences

11.0 引言

Introduction

一个序列 (sequence) 是在需要时按顺序生成的一组整数 (1, 2, 3, ...)。序列在数据库中经常会用到，因为很多程序要求数据库表中每一行记录含有一个唯一值，而序列提供了一种简单的方法来为每一行生成一个唯一值。本章讨论如何在 MySQL 中使用序列，包含如下主题：

使用 AUTO_INCREMENT 列创建序列

AUTO_INCREMENT 列是 MySQL 为一些行生成序列的一种机制。当你每一次在含有一个 AUTO_INCREMENT 列定义的表中生成一行时，MySQL 自动生成序列中的下一个值作为其列值 (AUTO_INCREMENT 列)。这个 (序列) 值可以充当 (每一行) 的唯一标识，这使得序列可以方便地生成用户 ID、航运包裹的货运单号、发票或者商品订购编号、bug 报告 ID、票据编号，或者是产品序列号等。

查询序列值

对很多程序而言，仅生成序列值是不够的。也需要能够判断刚刚插入数据库的一行记录的序列值。一个 Web 程序可能需要向用户回显刚刚提交的表单中的一行记录的内容。该数值也可能需要被查询，因此需要将其保存到相关表的某一个行。

再序列技术

这个主题讨论如何对由于删除了某些记录而造成空洞的序列重新编号，以及为什么要避免再序列。另一个主题讨论了以 1 以外的数字作为序列起点，以及给一张表添加一个序列类型列。

使用一个 AUTO_INCREMENT 列来创建复合序列

在许多情况下，表中的 AUTO_INCREMENT 列是独立于其他列的，并且列值按照一个序列单调递增。然而，如果你创建了一个组合主键，其中包含了一个 AUTO_INCREMENT 列，你就可以用来生成复合序列。例如，假设你在运营一个公告牌，把所有信息按

照主题分类，你可以通过把一个 AUTO_INCREMENT 列和主题指示器列符合，对每一个主题内的记录使用序列进行编号。

管理组合并发 AUTO_INCREMENT 值

当你需要处理组合并发序列值时需要特别小心。这种情况常常出现在你在一张表中执行一组语句，或者在多张都有 AUTO_INCREMENT 列的表中同时插入记录时。这个主题讨论了在这些情况下要怎么做。

使用单行序列生成器

序列也可以用作计数器。例如，如果你在你的网站上提供标题广告服务，你可能在每一次点击都增加一次计数器的值（也就是说，你每一次提供广告服务）。对广告的计数构成了一个序列，但是因为计数值本身是我们唯一感兴趣的数值，没有必要每一次计数都插入一个新行。MySQL 为这一类问题提供了一个解决机制，从而可以方便地在表中同一行内生成序列。为了在表中保存组合计数，可以增加一列来唯一标识每一个计数器。同样的机制可以使序列按照 1 以外的增量值，或者是使用变化的增量值，或者使用负增量值来增加。

对查询结果行进行序列化编号

这个主题中建议了几种为生成只读序列查询结果进行编号的方法。

大多数的数据库系统引擎都提供了生成序列的功能，尽管不同的实现都依赖于具体的数据引擎。对于 MySQL 也是这样的（依赖于 MySQL 数据库引擎），因此本节中的讨论几乎都是 MySQL 特有的。换句话来说，就算你使用提供了抽象层的 DBI 或者 JDBC 的 API，生成序列的 SQL 语句本身是不可移植的。抽象接口或许能帮助你方便地使用 SQL 语句，但它并不会使不可移植的 SQL 具有可移植性。

与本章实例相关的脚本在序列 (sequence) 目录内。关于这里使用的创建表的脚本，参见 table 目录。

11.1 创建一个序列列并生成序列值

Creating a Sequence Column and Generating Sequence Values

问题

你想在表中包含一个序列。

解决方案

使用一个 AUTO_INCREMENT 列。

讨论

本节提供了 AUTO_INCREMENT 列如何工作的背景知识，首先举一个简短的例子来说明序列生成机制。这个例子围绕一个昆虫收集的场景展开：学校给你的儿子（八岁大的 Junior）布置了一个收集虫子的课堂作业。对每一个昆虫，Junior 都要记录它的名字（“蚂蚁”，“蜜蜂”等）、收集的日期和地点。你早些时候已经对 Junior 说明了使用 MySQL 记录数据的好处，所以那天你下班后一回到家，他就向你声明完成这个作业的必要性，然后，看着你的眼睛，宣布这是一个用 MySQL 处理问题的绝好机会。你要向谁抱怨呢？结果你们俩就开始工作了。Junior 放学后在等你的时候已经收集了一些昆虫回家，并且在笔记本上记录了下面的数据：

Name	Date	Origin
millipede	2006-09-10	driveway
housefly	2006-09-10	kitchen
grasshopper	2006-09-10	front yard
stink bug	2006-09-10	front yard
cabbage butterfly	2006-09-10	garden
ant	2006-09-10	back yard
ant	2006-09-10	back yard
millbug	2006-09-10	under rock

看完了 Junior 的笔记，你觉得很高兴，在他这样稚嫩的年纪，他已经学会了用 ISO 格式来记录数据了。然而，你也发现了他收集了完全不是昆虫的两个动物，一个 millipede 和一个 millbug。你决定现在不管这个，因为 Junior 忘了把这个作业的书写指导带回家，你这个时候也不确定这两个小东西是否合意。

在你考虑如何建表来记录这些信息的时候，根据（老师）要求 Junior 记录的信息，你显然需要 name、date 和 origin 列：

```
CREATE TABLE insect
(
    name    VARCHAR(30) NOT NULL,      # 昆虫类型
    date    DATE NOT NULL,            # 采集时间
    origin  VARCHAR(30) NOT NULL,      # 采集地点
);
```

然而，这些列还不足以创建一张易于使用的表。注意，到此为止收集到的记录还不是唯一的，两只蚂蚁都是在相同的时间和地点收集到的。如果你把数据记录到如上结构的一个 insect 表中，因为两个记录蚂蚁的信息之间并没有进行区别的信息，两条蚂蚁信息不能单独被引用。唯一的 ID 列或许能够帮助区分不同的行，并且提供一个方便对每一行进行查询的值。一个 AUTO_INCREMENT 列正适合这个目的，因此一个更好的 insect 表应该是这样：

```
CREATE TABLE insect
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    name    VARCHAR(30) NOT NULL,      # 昆虫类型
    date    DATE NOT NULL,            # 采集时间
    origin  VARCHAR(30) NOT NULL     # 采集地点
);
```

紧接着根据 insert 表的第二个定义创建了数据表。在第 11.2 节中我们将讨论为什么这样定义 id 列。

现在已经有了一个 AUTO_INCREMENT 列，你想用它来生成新的序列值。AUTO_INCREMENT 列的一个有用的特性是你不需要自己给它赋值：MySQL 会帮你完成（给 AUTO_INCREMENT 列赋值）。有两种方法生成一个新的 AUTO_INCREMENT 值，这里使用 insert 表的 id 列举例说明。首先，你可以显式地将 id 列设置为 NULL。下面的语句用这样的方法把 Junior 的前四条记录插入 insect 表中：

```
mysql> INSERT INTO insect (id, name, date, origin) VALUES
-> (NULL, 'housefly', '2006-09-10', 'kitchen'),
-> (NULL, 'millipede', '2006-09-10', 'driveway'),
-> (NULL, 'grasshopper', '2006-09-10', 'front yard'),
-> (NULL, 'stink bug', '2006-09-10', 'front yard');
```

第二，你可以在 INSERT 语句中完全忽略 id 列。在 MySQL 中，对于具有默认值的列，你可以在插入新行时不指定对应的值。MySQL 自动将默认值赋给每一个没有明确给出值的列，并且 AUTO_INCREMENT 列的默认值就是序列中的下一个序列值。因此，你在向 insect 表中插入新行的时候可以完全不指定 id 列。下面的语句使用这样的方式把 Junior 的另外四条记录插入 insect 表中：

```
mysql> INSERT INTO insect (name, date, origin) VALUES
-> ('cabbage butterfly', '2006-09-10', 'garden'),
-> ('ant', '2006-09-10', 'back yard'),
-> ('ant', '2006-09-10', 'back yard'),
-> ('millbug', '2006-09-10', 'under rock');
```

不管你使用哪种方法，MySQL 都会确定下一个序列值，并将其赋给 id 列，你可以这样自己验证：

```
mysql> SELECT * FROM insect ORDER BY id;
+----+-----+-----+-----+
| id | name        | date       | origin      |
+----+-----+-----+-----+
```

	1	housefly	2006-09-10 kitchen
	2	millipede	2006-09-10 driveway
	3	grasshopper	2006-09-10 front yard
	4	stink bug	2006-09-10 front yard
	5	cabbage butterfly	2006-09-10 garden
	6	ant	2006-09-10 back yard
	7	ant	2006-09-10 back yard
	8	millbug	2006-09-10 under rock

当 Junior 收集到更多的昆虫，你可以向表中插入更多记录，并且它们会被赋予序列中紧接着的值 (9, 10, ...)。

AUTO_INCREMENT 列背后的机制大体上是很简单的：每一次你创建一个新行，MySQL 生成序列中的下一个值，并把它赋给这个新行。但是也有一些细节需要明确，也就是不同的存储引擎如何处理 AUTO_INCREMENT 序列。通过了解这些问题，你可以更加有效地使用序列并避免出现未知结果。例如，如果你显式地把 id 设置为一个非 NULL 值，可能出现下面两个结果之一：

- 如果这个值已经在表中出现过，并且数据库不能接受重复，那么就会出现一个错误。对于 insect 表，id 列是一个主键，因此重复是不允许的：

```
mysql> INSERT INTO insect (id,name,date,origin) VALUES
-> (3,'cricket','2006-09-11','basement');
ERROR 1062 (23000): Duplicate entry '3' for key 1
```

- 如果表中没有出现过该值，MySQL 用它插入新行。另外，如果这个值比现有序列计数器的下一个值大，这个表的计数器会被重设为该值加 1。这里 insect 表有序列值 1 到 8。如果你插入一个新行，将 id 列设为 20，那么 20 就成为新的最大序列值。自动产生的序列值就从 21 开始。从 9 到 19 的值变得不可用，导致了序列中的一个断层。

接下来的讨论更多关注如何定义 AUTO_INCREMENT 列，以及它们如何工作的细节。

11.2 为序列列选择数据类型

Choosing the Data Type for a Sequence Column

问题

想知道关于如何定义一个序列类型列的更多东西。

解决方案

使用这里给出的方针。

讨论

在创建一个 AUTO_INCREMENT 列时你最好遵守下面的方针。作为一个例子，考虑 insect 表中的 id 列是如何定义的：

```
id INT UNSIGNED NOT NULL AUTO_INCREMENT,  
PRIMARY KEY (id)
```

AUTO_INCREMENT 关键字告诉 MySQL 它应该为该列值产生连续的数字，但是其他信息也同样重要：

- INT 是这一列的基本数据类型。你并不一定使用 INT，但是该列必须是一种整数类型：TINYINT, SMALLINT, MEDIUMINT, INT 或者 BIGINT。非常重要的一点：AUTO_INCREMENT 是只能作用于整数类型列的属性。
- 该列被定义为 UNSIGNED 来避免出现负值。这并不是 AUTO_INCREMENT 列的一个必须要求。然而，因为序列仅由正数组成（通常从 1 开始），并不允许出现负值。此外，不把列定义为 UNSIGNED 就把序列的取值范围缩小了一半。例如，TINYINT 具有取值范围 -128 到 127。序列只包含正值，因此 TINYINT 类型序列的取值范围从 1 到 127。TINYINT UNSIGNED 类型列值可以从 1 到 255，这就把序列上限增加到了 255。序列的最大值由使用的整数类型决定，因此你需要选择一个能够接受你可能使用的最大值的整数类型。各种类型的无符号整数的最大值见下表，你可以用来选择合适的数据类型：

日期类型	无符号整数的最大值
TINYINT	255
SMALLINT	65 535
MEDIUMINT	16 777 215
INT	4 294 967 295
BIGINT	18 446 744 073 709 551 615

有些时候用户省略了 UNSIGNED，使得他们能够创建含有负的序列值的行（使用 -1 代表“没有 ID”就是这样的例子）。这是一个坏主意。MySQL 并不确定在一个 AUTO_INCREMENT 列中会如何处理负数，因此如果你使用负数，你就是在玩火。例如，如果你对该列再次进行序列化，你会发现所有的负数都被变成了正的序列值。

- AUTO_INCREMENT 列不能包含 NULL 值，因此 id 列声明为 NOT NULL。（在插入新行时可以使用 NULL 对 id 列赋值，但对于 AUTO_INCREMENT，这样做的实际意义是（告

诉 MySQL) 生成下一个序列值) 如果你忘记了 (自己动手) MySQL 自动将 AUTO_INCREMENT 列定义为 NOT NULL。

- AUTO_INCREMENT 列必须被索引化。通常,一个序列存在的目的就是提供一个唯一标识,你使用一个 PRIMARY KEY 或者 UNIQUE 索引来确保唯一性。一张表只能含有一个 PRIMARY KEY,因此如果一张表含有其他 PRIMARY KEY 列,你可以将 AUTO_INCREMENT 列声明为 UNIQUE 索引来代替:

```
id INT UNSIGNED NOT NULL AUTO_INCREMENT,  
UNIQUE (id)
```

如果 AUTO_INCREMENT 列是 PRIMARY KEY 或者 UNIQUE 索引中的唯一列,那么你可以在列定义中对它进行定义,而不需要使用一个单独的子句进行定义。例如,下面的定义是等价的:

```
id INT UNSIGNED NOT NULL AUTO_INCREMENT,  
PRIMARY KEY (id)  
id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
```

下面的定义也相同:

```
id INT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE  
id INT UNSIGNED NOT NULL AUTO_INCREMENT,  
UNIQUE (id)
```

使用一个单独的子句来定义索引有助于强调一点:从严格意义上来说索引并不是列定义的一部分。

当你创建了一张包含 AUTO_INCREMENT 列的表时,考虑所使用的存储引擎 (MyISAM, InnoDB 等) 也是非常重要的。所用的引擎会影响一些操作,例如从序列顶部删除的值的再重用,以及你是否能够设置序列的初始值。一般而言,MyISAM 是支持含有 AUTO_INCREMENT 列的表的最佳引擎,因为它提供了序列管理的最大灵活性。在本章剩余部分的讨论中,这些将更加显而易见。

11.3 序列生成的行删除的效果

The Effect of Row Deletions on Sequence Generation

问题

你想知道当你表中删除含有 AUTO_INCREMENT 列的行时对应的序列发生了什么。

解决方案

这取决于你删除了哪一行,以及所使用的存储引擎。

讨论

到此为止我们只考虑了向表中插入行的情况下，如何生成一个 AUTO_INCREMENT 列中的序列值。但是不能不切实际地认为插入的行不会被删除。如果那样会对序列有什么影响呢？

再回到收集昆虫的项目上来，现在你有一张包含如下内容的 insect 表：

```
mysql> SELECT * FROM insect ORDER BY id;
+----+-----+-----+-----+
| id | name | date | origin |
+----+-----+-----+-----+
| 1  | housefly | 2006-09-10 | kitchen |
| 2  | millipede | 2006-09-10 | driveway |
| 3  | grasshopper | 2006-09-10 | front yard |
| 4  | stink bug | 2006-09-10 | front yard |
| 5  | cabbage butterfly | 2006-09-10 | garden |
| 6  | ant | 2006-09-10 | back yard |
| 7  | ant | 2006-09-10 | back yard |
| 8  | millbug | 2006-09-10 | under rock |
+----+-----+-----+-----+
```

Junior 想起把作业的书写要求带回家之后，情况有了一些改变，你读了一下要求，发现有两点要求会影响 insect 表的内容：

- 标本只能是昆虫，而不能是其他长得像昆虫的动物，例如 millipedes 和 millbugs。
- 这个作业的目标是收集尽量多不同种类的昆虫，而不只是尽量多的种类。也就是说只能记录一条蚂蚁的信息。

这些作业指导要求从 insect 表中删除一些行——具体的就是 id 为 2 (millipedes)，8 (millbug) 和 7 (多余的蚂蚁) 的行。因此，尽管 Junior 对缩小收集的昆虫数量非常失望，你还是指导他使用 DELETE 语句把不需要的行删除了。

```
mysql> DELETE FROM insect WHERE id IN (2,8,7);
```

这个语句说明了为什么使用唯一 ID 值是有好处的：它们让你精确地指定每一行。两行蚂蚁的记录除了 ID 值之外都是相同的。没有 insect 表中的这一行 (id)，就更难删除它们中的一行：

从表中删除了不符合条件的行以后，表的内容如下：

```
mysql> SELECT * FROM insect ORDER BY id;
+----+-----+-----+-----+
| id | name | date | origin |
+----+-----+-----+-----+
| 1  | housefly | 2006-09-10 | kitchen |
| 3  | grasshopper | 2006-09-10 | front yard |
+----+-----+-----+-----+
```

	4	stink bug	2006-09-10	front yard	
	5	cabbage butterfly	2006-09-10	garden	
	6	ant	2006-09-10	back yard	

id 列中的序列值现在有一个空洞（第 2 行不见了），并且序列顶端的值 7 和 8 也不见了。删除这些行会对以后的插入操作带来什么影响？下面插入的新行会使用哪个序列值？

删除第 2 行带来了序列中间的一个缺口。这对新序列的插入没有影响，因为 MySQL 不会去尝试补充一个序列中的空洞。另一方面，删除第 7 和第 8 行，就删除了序列顶端的值，这样做带来的影响取决于所用的存储引擎：

- 对于 BDB 表，接下来的序列值通常就是本列中当前最大整数加 1。如果你删除的行含有序列最顶端的值，那么这些值将会被再次使用。（因此，在你删除了序列值 7 和 8 之后，下一个插入的行将被分配序列值 7）。
- 对于 MyISAM 或者 InnoDB 表，序列值不会被重用。将来使用的序列值就是过去没有使用过的最小正整数。（对于最大值为 8 的序列，紧接着的序列值就是 9，不管你是否先删除了 7 和 8）。如果你严格要求单调递增序列，你可以使用这样的储存引擎（MyISAM 或者 InnoDB）。

如果一张表所使用的引擎的序列值重用特性与你所需要的不一样，可以使用 ALTER TABLE 将表转变到一个合适的引擎。例如，你想把一张 BDB 表转变为一张 MyISAM 表（避免删除一些行之后出现序列值重用），可以这样：

```
ALTER TABLE tbl_name ENGINE = MyISAM;
```

如果你不知道一张表使用了什么存储引擎，参考 INFORMATION_SCHEMA 或者使用 SHOW TABLE STATUS 或者 SHOW CREATE TABLE 来查询。例如，下面的语句说明 insect 表是一张 MyISAM 表：

```
mysql> SELECT ENGINE FROM INFORMATION_SCHEMA.TABLES
-> WHERE TABLE_SCHEMA = 'cookbook' AND TABLE_NAME = 'insect';
+-----+
| ENGINE |
+-----+
| MyISAM |
+-----+
```



提示：在本章，你可以认为如果一张表的定义中没有指明存储引擎，它就是一张 MyISAM 表。

如果你想清除一张表的所有信息，并将序列计数器重新初始化，使用如下语句：

```
TRUNCATE TABLE:  
TRUNCATE TABLE tbl_name;
```

11.4 查询序列值

Retrieving Sequence Values

问题

在插入了含有序列值的新行之后，你想知道序列值是多少。

解决方案

执行一个 `SELECT LAST_INSERT_ID` 语句。如果你在写一个程序，你所使用的 MySQL API 也许提供一种不需要执行 sql 语句而直接获取（所插入）序列值的方法。

讨论

很多应用程序需要确定新创建行的 `AUTO_INCREMENT` 列值。例如，如果你充满激情并写了一个基于 Web 程序为 Junior 的 `insect` 表输入数据，你可能想让你的程序在每一次点击“Submit”提交信息之后，在另一个页面上以整洁的格式显示刚刚输入的信息。为此，你需要知道新产生的 `id` 值，用来查询你刚刚输入的行。需要使用 `AUTO_INCREMENT` 值的另外一种情形出现在你使用多个表时：在你向一张主表插入一行之后，典型情况下，你需要（知道）这个新行的 ID，用来在另外一个关联到主表的附表中插入行（第 11.13 节中讨论了如何使用序列值关联多张表）。

在你生成一个新的 `AUTO_INCREMENT` 值时，你可以调用 `LAST_INSERT_ID()` 函数从服务器获取该值。另外，很多 MySQL API 提供了客户端机制来获取新生成的 `AUTO_INCREMENT` 值，而不需要单独的查询语句。这里对两种方法都进行了讨论，并对它们的特性进行了比较。

使用 `LAST_INSERT_ID()` 获取 `AUTO_INCREMENT` 值

获取新行的 `AUTO_INCREMENT` 值的一种直观（但是不正确）的方法基于的一个实际情况是，当 MySQL 生成一个 `AUTO_INCREMENT` 值，该值也就是一列中最大的序列值。因此，你可能想使用 `MAX()` 函数来查询（新的 `AUTO_INCREMENT` 值）：

```
SELECT MAX(id) FROM insect;
```

这种方法是不可靠的，因为它没有考虑到 MySQL 服务器多线程的特性。`SELECT` 语句确实会返回表中最大的 `id` 值，但是这个值可能并不是你生成的。假设你插入一个新行，生成的 `id` 值是 9。如果另外一个客户端在你执行 `SELECT` 语句之前也插入了一行，那么 `MAX(id)` 的返回值就是 10，而不是 9。解决这个问题的方法包括将 `INSERT` 和 `SELECT` 语

句合并为一个事务或者锁定表，但是 MySQL 提供了 `LAST_INSERT_ID()` 函数可以更加简洁地获取正确的 `id` 值。它返回你本次连接到服务器之后，最新创建的 `AUTO_INCREMENT` 值，而不会受其他客户端操作的影响。例如，你可以像这样向 `insect` 表中插入一行，然后获取对应的 `id` 值：

```
mysql> INSERT INTO insect (name,date,origin)
-> VALUES('cricket','2006-09-11','basement');
mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|         9        |
+-----+
```

或者你可以这样查询新插入的行，而不必知道它对应的 `id` 值具体是什么：

```
mysql> INSERT INTO insect (name,date,origin)
-> VALUES('moth','2006-09-14','windowsill');
mysql> SELECT * FROM insect WHERE id = LAST_INSERT_ID();
+----+----+----+----+
| id | name | date       | origin      |
+----+----+----+----+
| 10 | moth | 2006-09-14 | windowsill |
+----+----+----+----+
```

其他客户端能改变 `LAST_INSERT_ID()` 函数的返回值么？

如果你担心在你插入新行的时候其他客户端也执行了生成 `AUTO_INCREMENT` 值的操作，导致你通过 `LAST_INSERT_ID()` 获得错误的 `id` 值，那么你也许就会问上面的问题。那并没有什么好担心的。`LAST_INSERT_ID()` 的返回值基于服务器的每一个客户端连接。这一特性非常重要，因为它避免了不同用户之间的相互干扰。当你生成一个 `AUTO_INCREMENT` 值，`LAST_INSERT_ID()` 返回的就是这个值，不管当前是不是有其他用户同时也在表中插入了新行。这一行为是由服务器设计好的。

使用 API 定义方法获取 `AUTO_INCREMENT` 值

`LAST_INSERT_ID()` 是一个 SQL 函数，因此你可以在所有能识别 SQL 语句的客户端上使用。另一方面，你必须执行一个单独的语句来获取 (`LAST_INSERT_ID()` 的返回值)。如果你正在编写应用程序，你也许有其他选择 (获取 `LAST_INSERT_ID()` 值)。很多 MySQL 接口含有 API 扩展，它可以返回 `AUTO_INCREMENT` 值，而不需要另外的 SQL 语句。很多 API 都有这样的功能。

Perl

使用 `mysql_insertid` 属性获取一个语句生成的 `AUTO_INCREMENT` 值。通过数据库句柄或者一个语句 (statement) 句柄获取这个属性，具体情况依赖于你如何执行这个语句。下面的例子使用数据库句柄：

```
$dbh->do ("INSERT INTO insect (name,date,origin)
            VALUES('moth','2006-09-14','windowsill')");
my $seq = $dbh->{mysql_insertid};
```

如果你在使用 `prepare()` 和 `execute()` 语句，那么以语句句柄属性的形式使用 `mysql_insertid`：

```
my $sth = $dbh->prepare ("INSERT INTO insect (name,date,origin)
                           VALUES('moth','2006-09-14','windowsill')");
$sth->execute ();
my $seq = $sth->{mysql_insertid};
```

如果你发现 `mysql_insertid` 属性总是未定义或者是 0，可能是因为你使用了不支持 `mysql_insertid` 属性的老版本的 DBD::mysql。可以试试 `insertid` 属性 (`insertid` 只能作为数据库句柄属性)。

Ruby

MySQL 的 Ruby DBI 驱动使用返回驱动特定值的数据库句柄 `func` 方法，获取客户端 `AUTO_INCREMENT` 值：

```
dbh.do("INSERT INTO insect (name,date,origin)
        VALUES('moth','2006-09-14','windowsill')")
seq = dbh.func(:insert_id)
```

PHP

本地 PHP 提供的 MySQL 接口包含返回最新 `AUTO_INCREMENT` 值的函数，但是 PEAR DB 没有这样的函数。另一方面，PEAR DB 有自己的序列值生成机制。具体细节请参考 PEAR 文档。

Python

DB-API 的 MySQLdb 驱动提供了一个 `insert_id()` 对象连接方法，用来在执行生成 `AUTO_INCREMENT` 的语句之后获取该序列值。

```
cursor = conn.cursor ()
cursor.execute ("""
    INSERT INTO insect (name,date,origin)
    VALUES('moth','2006-09-14','windowsill')
""")
seq = conn.insert_id ()
```

MySQL Connector/J JDBC 驱动提供了一个 `getLastInsertID()` 方法来获取 `AUTO_INCREMENT` 值。可以用于 `Statement` 或者 `PreparedStatement` 对象。下面的例子使用一个 `Statement` 对象：

```
Statement s = conn.createStatement ();
s.executeUpdate ("INSERT INTO insect (name,date,origin)"
    + " VALUES ('moth','2006-09-14','windowsill')");
long seq = ((com.mysql.jdbc.Statement) s).getLastInsertID ();
s.close ();
```

注意，因为 `getLastInsertID()` 方法是驱动定义的，你实际上通过把 `Statement` 对象转换为 `com.mysql.jdbc.Statement` 类型对象来使用。如果你在使用 `PreparedStatement` 对象，就把它转换为 `com.mysql.jdbc.PreparedStatement` 类型对象：

```
PreparedStatement s = conn.prepareStatement (
    "INSERT INTO insect (name,date,origin)"
    + " VALUES ('moth','2006-09-14','windowsill')");
s.executeUpdate ();
long seq = ((com.mysql.jdbc.PreparedStatement) s).getLastInsertID ();
s.close ();
```

服务器端和客户端获取序列值的比较

先前提到过，`LAST_INSERT_ID()` 的维护基于服务器端的每一个特定 MySQL 连接。与之相对应的获取 `AUTO_INCREMENT` 的 API 方法直接在客户端实现。服务器端和客户端的序列值获取方法有相同之处，但是也有区别。

服务器端和客户端的所有方法，都要求生成和获取 `AUTO_INCREMENT` 操作在同一个 MySQL 连接内。如果你生成了一个 `AUTO_INCREMENT` 值，然后从服务器断开，然后再重新建立连接来获取刚刚生成的值，你会得到 0。在一个特定连接内，`AUTO_INCREMENT` 值的保持时间会比该连接本身更长：

- 你执行一个生成 `AUTO_INCREMENT` 值的语句之后，这个值对于 `LAST_INSERT_ID()` 操作一直都是有效的，就算你执行了其他语句，一直到你执行了其他生成 `AUTO_INCREMENT` 值的语句。
- 客户端序列值的有效性与每一条语句相关，而不仅仅是生成 `AUTO_INCREMENT` 值的语句决定。如果你执行一个 `INSERT` 语句生成了一个 `AUTO_INCREMENT` 值，然后在获取客户端序列值之前又执行了一些其他语句，那么这个值有可能会被设为 0。各个 API 的具体行为不同，但是你如果使用如下的原则，可以避免错误：当你生成一个不会马上实用的序列值，把它保存到一个你稍后可以实用的变量中。否则，你可能发现当你需要使用这个序列值的时候，它已经被归 0 或者覆盖了。

11.5 对一个已有的序列进行重新计数

Renumbering an Existing Sequence

问题

在序列中有断层，你想再次对其进行序列化。

解决方案

别麻烦了。或者你至少有一个罕见的还不错的理由。

讨论

如果你插入含有 AUTO_INCREMENT 列的行，并且从来不删除其中的任何一行，AUTO_INCREMENT 列中的值就组成一个连续的序列。但是，如果你删除其中的一些行，序列中就会出现空洞。例如，Junior 的 insect 表现在看起来就是这样，在序列中有空洞存在（假设你已经在“11.4 查询序列值”一节中的 insect 表中输入了蟋蟀(cricket)和蛾子(moth)）：

```
mysql> SELECT * FROM insect ORDER BY id;
+---+-----+-----+-----+
| id | name           | date       | origin    |
+---+-----+-----+-----+
| 1  | housefly        | 2006-09-10 | kitchen   |
| 3  | grasshopper     | 2006-09-10 | front yard|
| 4  | stink bug       | 2006-09-10 | front yard|
| 5  | cabbage butterfly| 2006-09-10 | garden    |
| 6  | ant             | 2006-09-10 | back yard |
| 9  | cricket          | 2006-09-11 | basement  |
| 10 | moth            | 2006-09-14 | windowsill|
+---+-----+-----+-----+
```

当你插入新行时，MySQL 不会使用未使用的序列值来弥补序列断层。不喜欢（数据库）这一行为的用户希望通过周期性地对 AUTO_INCREMENT 列进行再次序列化，来消除空洞。下面的窍门教你如何做到这一点。也可以扩展一个现有序列的范围，在原来没有序列的表上添加序列，强制重用在序列顶端删除的序列值，或者在新建和再序列化一张表时指定序列值。

在你决定对一个 AUTO_INCREMENT 列再次序列化之前，需要想好是否确实想要，或者说真的需要这样做。在很多情况下都不需要这样做。事实上，有些时候再序列化会给你带来麻烦。例如，如果某一列的值关联了其他表，你就不应该对其进行再序列化。对这些值再序列化破坏了它们与其他表的对应关系，不可能再将两张表中的行正确地关联起来。

这里是我见过的一些进行再序列的理由：

美化

有时候对一列再序列化是出于美观。人们似乎愿意选择中间没有空洞的序列。如果这就是你进行再序列化的原因，我大概没理由阻止你。然而，这并不是一个好的理由。

性能

再序列化的诱惑可能是，能清除空洞使序列紧凑，并且使得 MySQL 能够更快地执行语句。这并不正确。MySQL 并不在乎（序列中）是否有空洞，并且对一个 AUTO_INCREMENT 在序列化不会获得（更好的）性能。实际上，MySQL 在执行再序列化操作的时候会锁住表，对性能带来负面影响。

空间用完

一个序列的上限取决与它的数据类型。如果一个 AUTO_INCREMENT 序列到达了所用数据类型的上限，（对其做）在序列化可以压缩序列并在顶部释放一些值空间。这也可能是对一列进行再序列化的一个合理原因，但是在很多情况下仍然不需要这样做。你也许能在不改变列内容的前提下增大其取值范围（参见第 11.6 节）。

如果你不听我的劝告，决定要对一列进行再序列化，这也是很容易的：从表中删除这一列，然后再添加回去，MySQL 会将列值重新序列化为一个连续序列。下面的例子说明了如何使用这个技巧对 insect 表中的 id 列再序列化：

```
mysql> ALTER TABLE insect DROP id;
mysql> ALTER TABLE insect
-> ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
-> ADD PRIMARY KEY (id);
```

第一个 ALTER TABLE 语句删除了 id 列（结果也会删除主键，因为主键列已经不存在了）。第二个语句又将这一列添加回去，并将其设为主键（关键字 FIRST 将该列设为表中第一列，也就是这一列原来的位置。通常 ADD 会把所添加的列放在表尾）。当你把一个 AUTO_INCREMENT 列加入表，MySQL 自动对所有行进行连续序列化，因此 insect 表结果看起来像这样：

```
mysql> SELECT * FROM insect ORDER BY id;
+----+-----+-----+
| id | name | date | origin |
+----+-----+-----+
| 1 | housefly | 2006-09-10 | kitchen |
| 2 | grasshopper | 2006-09-10 | front yard |
```

	3		stink bug		2006-09-10		front yard	
	4		cabbage butterfly		2006-09-10		garden	
	5		ant		2006-09-10		back yard	
	6		cricket		2006-09-11		basement	
	7		moth		2006-09-14		windowsill	

使用单独的 ALTER TABLE 语句来对一列进行再序列化的一个问题是，在两个操作间隔的时间内，所操作列会从表里消失。这可能会给在这段时间内访问该表的其他客户端带来问题。为了防止这个问题，将两个操作放在同一个 ALTER TABLE 语句内：

```
mysql> ALTER TABLE insect
-> DROP id,
-> ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST;
```

MySQL 允许在一个 ALTER TABLE 进行多个操作（这不是所有数据库系统都支持的）。然而，需要注意的是这“多个操作”并不是两个 ALTER TABLE 语句独立操作的简单结合。区别在于这样（使用一个 ALTER TABLE 语句）不需要重新设定主键：除非在 ALTER TABLE 语句中所有语句执行结束之后，源表中的主键列消失了，否则 MySQL 不会删除主键。

11.6 扩展序列列的取值范围

Extending the Range of a Sequence Column

问题

你不想对一列进行再次序列化，但是你又用完了该序列中的所有值。

解决方案

检查你是否能够将列值类型设为 UNSIGNED，或者将列值类型改为（取值范围）更大的整数类型。

讨论

对一个 AUTO_INCREMENT 类型列进行再序列化，潜在的可能会改变表中每一行的值。这个问题通常可以通过增大列取值范围来避免，这样改变了表的结构，而非表内容：

- 如果列值类型是有符号的，改为 UNSIGNED，这样能够使可用值范围加大一倍。假设你现在有如下定义的一个 id 列：

```
id MEDIUMINT NOT NULL AUTO_INCREMENT
```

有符号 MEDIUMINT 类型列取值上限是 8 388 607。通过 ALTER TABLE 将该列变为 UNSIGNED 能够把上限增加到 16 777 215。

```
ALTER TABLE tbl_name MODIFY id MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT;
```

- 如果某一列已经是 UNSIGNED 并且不是最大的整数类型 (BIGINT)，把列类型变为更大的整数类型可以扩大取值范围。这里你一样可以使用 ALTER TABLE。例如，上例中的 *id* 列可以这样从 MEDIUMINT 类型转变为 BIGINT 类型：

```
ALTER TABLE tbl_name MODIFY id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT;
```

第 11.2 节中有一张表列出了所有整数类型取值范围。如果你在查询可用的整数类型，那张表可能有用。

11.7 序列顶部数值的再使用

Reusing Values at the Top of a Sequence

问题

你从一个序列的顶部删除了一些行。你能够再次使用被删除的值，而不需要对该列做再序列化么？

解决方案

是的，使用 ALTER TABLE 重置序列计数器。MySQL 将从当前表中最大值加 1 开始生成新的序列值。

讨论

如果你仅从序列顶端删除了一些行，剩下的部分仍然是连续有序的。（例如，如果你有编号从 1 到 100 的行，并且你删除了编号从 91 到 100 的行，剩下的行从 1 到 90 还是连续的）。在这种特殊的情况下，不需要对一列进行再序列化。代替的方法是，告诉数据库以现有的最大序列值加 1 开始生成序列值。这是 BDB 类型表的默认行为，因此不需要你的额外操作就可以重用被删除的值。对于 MyISAM 和 InnoDB 表，执行下面的语句：

```
ALTER TABLE tbl_name AUTO_INCREMENT = 1;
```

这使得 MySQL 把序列计数器重置为为了能够利用的最小值。

如果一个序列的中间有断层，你可以使用 ALTER TABLE 重置序列计数器，但是这样做的结果你还是只能重用在序列顶部删除的值。这样并不会除去（序列中的）断层。假设你有一张含有从 1 到 10 序列值的表，并且你从中删除了值为 3、4、5、9 和 10 的行，剩下的最大值是 8，因此如果你使用 ALTER TABLE 重置序列计数器，下面（新插入）的行的序列值从 9 开始，而不是 3。重新序列化一个表并去除其中的断层，参见第 11.5 节。

11.8 确保各行按照给定顺序重编号

Ensuring That Rows Are Renumbered in a Particular Order

问题

你对一列再次进行序列化，但是 MySQL 没有按照你希望的方式进行排列各行。

解决方案

查询这些行，保存到另外一个表中，使用 ORDER BY 语句使各行按照自己的意图排序，并在进行转存操作时让 MySQL 给每一行分配序列值。这样各行就按照你的目的进行序列编号了。

讨论

当你对一个 AUTO_INCREMENT 列再次进行序列化时，MySQL 可以从表中按照任意顺序取出行，因此它不需要按照你所希望的顺序再次进行序列化。如果你只是要求每一行有一个唯一编号，那（数据库的这种行为）没有什么影响。但是你也许有一个应用要求各行按照特定顺序指定序列值。例如，你可能希望序列值对应到每一行的创建顺序，就像一个 TIMESTAMP 列包含的（创建顺序）信息。使用下面的程序来按照特定顺序指定序列值：

1. 创建表的空克隆（见第 4.1 节）。
2. 使用 INSERT INTO...SELECT 从源表中复制行。复制除了序列之外的所有列，使用 ORDER BY 子句指定各行拷贝的顺序（同时指定一个序列值）。
3. 删除源表，并将克隆表重命名为源表表名。
4. 如果是一个很大的 MyISAM 表，并含有多个索引，创建新表时不定义除了 AUTO_INCREMENT 列之外的索引，会使整个过程更高效。然后再将数据从源表复制到新表，然后定义索引。

另外一个可选的程序：

1. 创建一个包含源表中除了 AUTO_INCREMENT 列之外所有的列的新表。
2. 使用 INSERT INTO ... SELECT 语句把非 AUTO_INCREMENT 列复制到新表中。
3. 在源表中删除所有行，如果需要的话将序列计数器重置为 1。

4. 把信息从新表中复制回源表，使用一个 ORDER BY 语句把按照你希望的序列值顺序对各行进行排序。MyISAM 类型表自动将序列值赋给 AUTO_INCREMENT 列。

11.9 从某个特定值开始一个序列

Starting a Sequence at a Particular Value

问题

序列从 1 开始，但是你想使用其他起始值。

解决方案

在创建表时，在 CREATE TABLE 语句中加上一个 AUTO_INCREMENT 子句。如果是已经创建好的表，使用 ALTER TABLE 语句来设定（序列）起始值。

讨论

默认情况下，AUTO_INCREMENT 序列从 1 开始：

```
mysql> CREATE TABLE t
    -> (id INT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY (id));
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> SELECT id FROM t ORDER BY id;
+---+
| id |
+---+
| 1 |
| 2 |
| 3 |
+---+
```

对于 MyISAM 或者 InnoDB 类型表，你可以在 CREATE TABLE 语句的末尾使用 AUTO_INCREMENT=*n* 子句指定一个序列从特定值 *n* 开始。

```
mysql> CREATE TABLE t
    -> (id INT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY (id))
    -> AUTO_INCREMENT = 100;
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> SELECT id FROM t ORDER BY id;
+---+
| id |
+---+
| 100 |
| 101 |
| 102 |
+---+
```

另外，你也可以先创建表，然后使用 ALTER TABLE 来指定序列起始值：

```
mysql> CREATE TABLE t
-> (id INT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY (id));
mysql> ALTER TABLE t AUTO_INCREMENT = 100;
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> SELECT id FROM t ORDER BY id;
+----+
| id |
+----+
| 100 |
| 101 |
| 102 |
+----+
```

为了使 MyISAM 或者 InnoDB 之外的存储引擎从 n 开始一个序列，你可以使用一个窍门：插入具有序列值 $n-1$ 的“假”行，然后在插入了一行或者多行“真”数据之后删除这个“假”行。下面的例子说明了对一个 BDB 表，如何从 100 开始一个序列：

```
mysql> CREATE TABLE t
-> (id INT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY (id))
-> ENGINE = BDB;
mysql> INSERT INTO t (id) VALUES(99);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> DELETE FROM t WHERE id = 99;
mysql> SELECT * FROM t ORDER BY id;
+----+
| id |
+----+
| 100 |
| 101 |
| 102 |
+----+
```

记住一点，如果你使用 TRUNCATE TABLE 清空一张表，那么序列起点可能重置为 1，甚至对那些不重用序列值的存储引擎也是如此（见第 11.3 节）。这种情况下，如果你不希望序列从 1 开始，你需要在清空表之后重新指定序列起始值。

11.10 序列化一个未序列的表

Sequencing an Unsequenced Table

问题

你在创建表时忘了包含一个序列。如果再要想序列化表中的行是不是为时过晚了？

解决方案

还不晚，只需要用 ALTER TABLE 语句加上一个 AUTO_INCREMENT 列。MySQL 会自动创建一列并对所有行进行序列化。

讨论

为了给原来没有序列的表加上序列，使用 ALTER TABLE 创建一个 AUTO_INCREMENT 列。假设你有一张表包含 name 和 age 两列，但是没有序列：

name	age
boris	47
clarence	62
abner	53

你可以像下面这样给表添加一个序列 id：

```
mysql> ALTER TABLE t
      -> ADD id INT NOT NULL AUTO_INCREMENT,
      -> ADD PRIMARY KEY (id);
mysql> SELECT * FROM t ORDER BY id;
+-----+-----+----+
| name | age | id |
+-----+-----+----+
| boris | 47 | 1 |
| clarence | 62 | 2 |
| abner | 53 | 3 |
+-----+-----+----+
```

MySQL 自动为每行编号。你不需要自己为每一行分配一个序列值。非常便捷。

默认的，ALTER TABLE 在表尾插入新列。在 ADD 子句的末尾使用 FIRST 或者 AFTER。再在特定位置插入新列。下面的 ALTER TABLE 语句看起来跟上面的很相似，但是把 id 列作为第一列插入，或者在 name 列之后插入，分别是：

```
ALTER TABLE t
  ADD id INT NOT NULL AUTO_INCREMENT FIRST,
  ADD PRIMARY KEY (id);
ALTER TABLE t
  ADD id INT NOT NULL AUTO_INCREMENT AFTER name,
  ADD PRIMARY KEY (id);
```

对 MyISAM 或者 InnoDB 表，通过在 ALTER TABLE 语句中加入 AUTO_INCREMENT=n 子句，你能够指定新加入序列的起始值。

```
mysql> ALTER TABLE t
      -> ADD id INT NOT NULL AUTO_INCREMENT FIRST,
      -> ADD PRIMARY KEY (id),
```

```
-> AUTO_INCREMENT = 100;
mysql> SELECT * FROM t ORDER BY id;
+----+-----+-----+
| id | name | age |
+----+-----+-----+
| 100 | boris | 47 |
| 101 | clarence | 62 |
| 102 | abner | 53 |
+----+-----+-----+
```

11.11 使用 AUTO_INCREMENT 栏来创建多重序列

Using an AUTO_INCREMENT Column to Create Multiple Sequences

问题

你需要使用比单个序列更复杂的序列行为。你不要把不同的序列绑定到表中其他的列。

解决方案

把 AUTO_INCREMENT 列和其他列链接起来，使它们都是同一个索引的一部分。

讨论

如果 PRIMARY KEY 或者 UNIQUE 只包含一列 AUTO_INCREMENT 类型列，每次你添加一行，(AUTO_INCREMENT 列) 它都产生每次递增 1 的单个连续序列 1、2、3... 而不管一行中其他列的内容。对于 MyISAM 或者 BDB 表，可以创建由 AUTO_INCREMENT 列和其他列组合而成的索引，在单张表内产生多个序列。

下面是具体的方法：让我们设想 Junior 对昆虫收集变得非常有热情，因此他决定在学校的作业结束之后，继续这项工作——除了不再收到老师要求的约束之外，他更愿意收集类似昆虫的虫子，例如千足虫 (millipedes)，并且甚至收集同一物种的多个标本。Junior 很高兴的出门去了，并在以后的几天收集到了更多标本：

Name	Date	Origin
ant	2006-10-07	kitchen
millipede	2006-10-07	basement
beetle	2006-10-07	basement
ant	2006-10-07	front yard
ant	2006-10-07	front yard

Name	Date	Origin
honeybee	2006-10-08	back yard
cricket	2006-10-08	garage
beetle	2006-10-08	front yard
termite	2006-10-09	kitchen woodwork
cricket	2006-10-10	basement
termite	2006-10-11	bathroom woodwork
honeybee	2006-10-11	garden
cricket	2006-10-11	garden
ant	2006-10-11	garden

记录好这些信息之后，他已经准备好把它们录入数据库，但是想单独给每一种虫子编号（蚂蚁 1、蚂蚁 2...，甲虫 1、甲虫 2、...，蟋蟀 1、蟋蟀 2 等等）。最后，你考查了这些信息（Junior 在屋子里发现的白蚁给了你一些警告，然后你慌张的给除蚁人打了电话），然后给 Junior 定义了下面这个 bug 表：

```
CREATE TABLE bug
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name    VARCHAR(30) NOT NULL, # 昆虫类型
    date    DATE NOT NULL,       # 采集日期
    origin  VARCHAR(30) NOT NULL, # 采集地点
    PRIMARY KEY (name, id)
);
```

这看起来和 insect 表很像，但是有一个非常明显的区别：PRIMARY KEY 由两列组成，而不是一列。结果就是，id 列的某些行为与 insect 表中（的 id 列）不一样。如果新的标本信息按照 Junior 写下的顺序输入 bug 表中，下面就是表最后的样子：

```
mysql> SELECT * FROM bug;
+----+-----+-----+-----+
| id | name | date | origin |
+----+-----+-----+-----+
| 1  | ant  | 2006-10-07 | kitchen |
| 1  | millipede | 2006-10-07 | basement |
| 1  | beetle | 2006-10-07 | basement |
| 2  | ant  | 2006-10-07 | front yard |
| 3  | ant  | 2006-10-07 | front yard |
| 1  | honeybee | 2006-10-08 | back yard |
| 1  | cricket | 2006-10-08 | garage |
| 2  | beetle | 2006-10-08 | front yard |
| 1  | termite | 2006-10-09 | kitchen woodwork |
| 2  | cricket | 2006-10-10 | basement |
| 2  | termite | 2006-10-11 | bathroom woodwork |
| 2  | honeybee | 2006-10-11 | garden |
```

	3	cricket	2006-10-11	garden
	4	ant	2006-10-11	garden

从某个角度来看这张表，看起来 id 值被随机赋值——但实际上并不是。用 name 和 id 来对表排序，MySQL 如何（对 id）赋值就很明显了。特别的，MySQL 为每一个唯一的 name 值创建了一个独立的序列：

```
mysql> SELECT * FROM bug ORDER BY name, id;
+----+-----+-----+-----+
| id | name | date | origin |
+----+-----+-----+-----+
| 1  | ant  | 2006-10-07 | kitchen |
| 2  | ant  | 2006-10-07 | front yard |
| 3  | ant  | 2006-10-07 | front yard |
| 4  | ant  | 2006-10-11 | garden |
| 1  | beetle | 2006-10-07 | basement |
| 2  | beetle | 2006-10-08 | front yard |
| 1  | cricket | 2006-10-08 | garage |
| 2  | cricket | 2006-10-10 | basement |
| 3  | cricket | 2006-10-11 | garden |
| 1  | honeybee | 2006-10-08 | back yard |
| 2  | honeybee | 2006-10-11 | garden |
| 1  | millipede | 2006-10-07 | basement |
| 1  | termite  | 2006-10-09 | kitchen woodwork |
| 2  | termite  | 2006-10-11 | bathroom woodwork |
+----+-----+-----+-----+
```

当你创建一个由多列组合而成的 AUTO_INCREMENT 索引，注意一下几点：

- CREATE TABLE 语句定义的索引列的顺序是无关紧要的。真正有关系的是索引定义中指定每一列的顺序。AUTO_INCREMENT 列必须在最后指定，否则组合序列的机制就不能正常工作。
- 一个 PRIMARY KEY 不能含有 NULL 值，但是一个 UNIQUE 索引可以。如果作为索引的非 AUTO_INCREMENT 列中可能含有 NULL，那么你应该创建一个 UNIQUE 索引，而非 PRIMARY KEY。

对于 bug 表，AUTO_INCREMENT 索引包含两列。同样的技术可以延伸到包含多于两列的情况，但是基本的概念是一样的：对一个 n 列索引，最后一列是 AUTO_INCREMENT 列，MySQL 为每一个由非 AUTO_INCREMENT 列组成的唯一组合生成一个独立的序列。

MySQL 的多列序列（multiple-column sequences）机制能够比单列值的逻辑相比更容易使用。回忆第 7.10 节中，我们使用一个 housewares 表，其中每一行包含一个由 3 个字母的种类缩写，5 个数字的序列号和 2 个字母的生产国家编号组成的 ID 值：

id	description
DIN40672US	dining table
KIT00372UK	garbage disposal
KIT01729JP	microwave oven
BED00038SG	bedside lamp
BTH00485US	shower stall
BTH00415JP	lavatory

这个表在第 7 章中用来说明如何使用 LEFT()、MID() 和 RIGHT() 把 id 值分解为几个要素，并单独对每一个要素进行排序。那样会带来一些很丑陋的 ORDER BY 子句，那一章中我没有提出的一个问题是如何得到 id 值中间的序列号部分。

有时你可以把这样的多列组合替换为多个独立的列，这些独立的列被绑定为一个 AUTO_INCREMENT 索引。例如，管理 houseware 表 id 值的方法是，用 category、serial 和 country 列来描述 id 列，并通过定义 serial 为 AUTO_INCREMENT 列将 3 列绑定为一个 PRIMARY KEY。这样做的结果是 serial 值为每一个 category 和 country 值的组合独立增长。你写了下面的 CREATE TABLE 语句来创建表：

```
CREATE TABLE housewares
(
    category      VARCHAR(3) NOT NULL,
    serial        INT UNSIGNED NOT NULL AUTO_INCREMENT,
    country       VARCHAR(2) NOT NULL,
    description   VARCHAR(255),
    PRIMARY KEY (category, country, serial)
);
```

作为选择，假设你有在前一章中用到的已经创建好的原始 housewares 表，你可以使用下面的语句把它转换为“合适的”新结构：

```
mysql> ALTER TABLE housewares
-> ADD category VARCHAR(3) NOT NULL FIRST,
-> ADD serial INT UNSIGNED NOT NULL AUTO_INCREMENT AFTER category,
-> ADD country VARCHAR(2) NOT NULL AFTER serial,
-> ADD PRIMARY KEY (category, country, serial);
mysql> UPDATE housewares SET category = LEFT(id,3);
mysql> UPDATE housewares SET serial = MID(id,4,5);
mysql> UPDATE housewares SET country = RIGHT(id,2);
mysql> ALTER TABLE housewares DROP id;
mysql> SELECT * FROM housewares;
+-----+-----+-----+
| category | serial | country | description |
+-----+-----+-----+
| DIN     | 40672 | US      | dining table |
| KIT     | 372   | UK      | garbage disposal |
```

```

| KIT      | 1729 | JP      | microwave oven   |
| BED      | 38   | SG      | bedside lamp     |
| BTH      | 485  | US      | shower stall    |
| BTH      | 415   | JP      | lavatory        |
+-----+-----+-----+-----+

```

对于分解为几个独立部分的 `id` 值，排序变得简单，因为你可以直接单独引用每一列，而不需要从原来的 `id` 值中解析出各个子串（再进行比较）。你还可以通过给 `serial` 和 `country` 加上使排序更加高效。但还有一个问题是如何以一个字符串的方式来显示产品 ID，而不是使用三个独立的字符串？用 `CONCAT()` 解决这个问题：

```

mysql> SELECT category, serial, country,
-> CONCAT(category,LPAD(serial,5,'0'),country) AS id
-> FROM housewares ORDER BY category, country, serial;
+-----+-----+-----+-----+
| category | serial | country | id       |
+-----+-----+-----+-----+
| BED      | 38   | SG      | BED00038SG |
| BTH      | 415  | JP      | BTH00415JP |
| BTH      | 485  | US      | BTH00485US |
| DIN      | 40672 | US      | DIN40672US |
| KIT      | 1729 | JP      | KIT01729JP |
| KIT      | 372   | UK      | KIT00372UK |
+-----+-----+-----+-----+

```

你甚至可以把 `serial` 列定义为一个以 5 位数字显示的 zero-filled 列，这样可以避免使用 `LPAD()` 函数：

```

mysql> ALTER TABLE housewares
-> MODIFY serial INT(5) UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT;

```

MySQL 提供开头自动为 0 的支持，这样一来 `CONCAT()` 表达式就简化了：

```

mysql> SELECT category, serial, country,
-> CONCAT(category,serial,country) AS id
-> FROM housewares ORDER BY category, country, serial;
+-----+-----+-----+-----+
| category | serial | country | id       |
+-----+-----+-----+-----+
| BED      | 00038 | SG      | BED00038SG |
| BTH      | 00415 | JP      | BTH00415JP |
| BTH      | 00485 | US      | BTH00485US |
| DIN      | 40672 | US      | DIN40672US |
| KIT      | 01729 | JP      | KIT01729JP |
| KIT      | 00372 | UK      | KIT00372UK |
+-----+-----+-----+-----+

```

这个例子说明了一个重要的原则：你可能以某种方式来考虑（处理）某些值（以单个字符串的方式处理 `id`），但是这并不意味着在数据库中你要以同样的方式来处理（这些值）。如果另外的处理方式（独立的列）更加高效或者更容易处理，也许就值得考虑（其他处理方式）——就算你最终为了方便人阅读而必须对这些列的显示进行格式化处理。

如果将多列格式化为一列牵涉到复杂的运算，或者你只是想对应用程序隐藏处理的细节，你可以定义一个存储函数，以相关的各列作为参数，返回合并好的 id 值。例如：

```
CREATE FUNCTION houseware_id(category VARCHAR(3),
                               serial INT UNSIGNED,
                               country VARCHAR(2))
RETURNS VARCHAR(10) DETERMINISTIC
RETURN CONCAT(category,LPAD(serial,5,'0'),country);
```

使用下面的函数。结果和前面是一样的，但是使用者不需要知道 id 是（使用 3 列）构造出来的：

```
mysql> SELECT category, serial, country,
-> houseware_id(category,serial,country) AS id
-> FROM housewares;
+-----+-----+-----+-----+
| category | serial | country | id      |
+-----+-----+-----+-----+
| BED     | 38    | SG     | BED00038SG |
| BTH     | 415   | JP     | BTH00415JP |
| BTH     | 485   | US     | BTH00485US |
| DIN     | 40672 | US     | DIN40672US |
| KIT     | 1729  | JP     | KIT01729JP |
| KIT     | 372   | UK     | KIT00372UK |
+-----+-----+-----+-----+
```

关于存储函数的更多信息，请参考第 16 章。

11.12 管理多重并发 AUTO_INCREMENT 数值

Managing Multiple Simultaneous
AUTO_INCREMENT Values

问题

你正在处理两张或者更多含有 AUTO_INCREMENT 列的表，并且你很难弄清楚（管理）为每一张表所生成的序列值。

解决方案

把序列值保存在用户定义变量中，以后使用。如果你正在一个应用程序内使用（sql）语句，把序列值保存在程序变量中。另外，你也许能够使用独立的连接或者语句实例来执行这些语句，从而避免混淆。

讨论

如同第 11.4 节中描述过的，`LAST_INSERT_ID()`返回的服务器端序列值，在每一次执行生成 `AUTO_INCREMENT` 值的语句时都会重置，同时客户端序列值在每一个语句执行时都会被重置。

如果你执行了一个生成 `AUTO_INCREMENT` 值的语句，但是你并不会马上引用这个新的序列值，而要等到执行另外一个生成 `AUTO_INCREMENT` 值的语句之后（才会引用生成的第一个序列值），会发生什么？

这时不管是使用 `LAST_INSERT_ID()` 还是客户端序列值，都无法在引用生成的第一个序列值。为了保持第一个新序列值的可用性，你可以在执行第二个语句前先保存生成的序列值。有几种方法来处理：

- 在 SQL 层面上，你可以把一个语句生成的 `AUTO_INCREMENT` 值保存在一个用户定义变量内：

```
INSERT INTO tbl_name (id,...) VALUES (NULL,...);
SET @saved_id = LAST_INSERT_ID();
```

然后你就可以执行另外的语句，而不用考虑对 `LAST_INSERT_ID()` 的影响。通过变量 `@saved_id`，可以在后续的语句中引用最早产生的 `AUTO_INCREMENT` 值。

- 在 API 的层面，你可以把 `AUTO_INCREMENT` 值保存在一个编程语言变量中。可以通过保存 `LAST_INSERT_ID()` 函数返回值，或者保存可用的 API 扩展的返回值来实现。
- 第三种可用的技术是使用（某些）API 能够使你保持独立的客户端 `AUTO_INCREMENT` 值。例如，Perl 中的语句句柄包含一个 `mysql_insertid` 属性，这个属性在各个不同的句柄之间是不会互相影响的。在 Java 中，使用独立的 `Statement` 或者 `PreparedStatement` 对象。

11.13 使用 `AUTO_INCREMENT` 值将表进行关联

Using `AUTO_INCREMENT` Values to Relate Tables

问题

你把一张表里的序列值当作另外一张表的键值，这样你就可以把两张表中的行关联起来。但是这种关联的设置不是很恰当。

解决方案

你或许没有按照正确的顺序插入行，或者你没有管理（跟踪）好序列值。改变插入的顺序，或者保存好序列值以便在需要的时候使用。

讨论

如果为了将一个明细表中的行关联到主表中的某一行，而将主表中用于生成 ID 值的 AUTO_INCREMENT 值也保存在明细表的行中，那么需要对这些值特别小心（地处理）。这种情况很常见。设想你有一张 invoice 表记录了客户订单的货物清单，和一张 inv_item 表记录了每一个客户订单的包括的每一种货物。在这里，invoice 是主表，inv_item 是明细表。为了唯一地表示每一个订单，invoice 表包含一个 AUTO_INCREMENT 列 inv_id。你也会把每一个订单的 id 号保存在 inv_item 的每一行，使你能够标记每一行属于哪个订单。这两张表看起来像这样：

```
CREATE TABLE invoice
(
    inv_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (inv_id),
    date DATE NOT NULL
    # 其他列（用户 ID、购物地址，等）
);
CREATE TABLE inv_item
(
    inv_id INT UNSIGNED NOT NULL, # invoice 表 ID（由 invoice 表获取）
    INDEX (inv_id),
    qty INT,                      # 数量
    description VARCHAR(40)       # 描述
);
```

对于具有这样关系的表，典型的使用方法是首先在主表中插入一行（生成标识该记录的 AUTO_INCREMENT 值），然后使用 LAST_INSERT_ID() 获取主表行 ID，用于在明细表中插入行。例如，如果一个客户买了一个锤子、三箱钉子和一打绷带（为了应付手指头被锤子砸到），与这个订单相关的行以如下方式插入两张表中：

```
INSERT INTO invoice (inv_id,date)
VALUES(NULL,CURDATE());
INSERT INTO inv_item (inv_id,qty,description)
VALUES(LAST_INSERT_ID(),1,'hammer');
INSERT INTO inv_item (inv_id,qty,description)
VALUES(LAST_INSERT_ID(),3,'nails, box');
INSERT INTO inv_item (inv_id,qty,description)
VALUES(LAST_INSERT_ID(),12,'bandage');
```

第一个 INSERT 在 invoice 主表中插入一行，并为 inv_id 列生成一个新的 AUTO_INCREMENT 值。之后的每一个 INSERT 在 inv_item 表中插入一行，并使用 LAST_INSERT_ID() 获取相关的订单 ID 号。这个 ID 号把明细表中的行与主表中的行正确地关联起来。

如果你需要处理多个订单会怎样呢？这里有一种正确的方法和一种错误的方法来插入信息。正确的方法是输入第一个订单的所有信息，然后继续后面的订单（每次都输入一个完

整的订单信息)。错误的方法是先在 `invoice` 表中插入所有的主表行，然后再将所有的明细信息插入 `inv_item` 表。如果你这样做，`inv_item` 表中的所有明细行都将包含最后插入 `invoice` 表中的行所生成的 `AUTO_INCREMENT` 值，两张表中的行不会具有正确的对应关系。

如果明细表也有它自己的 `AUTO_INCREMENT` 列，对于如何向两张表中插入行，你就要更加小心了。假设你想对 `inv_item` 表中的属于一个订单的所有行进行编号。解决的办法是创建一个多列联合的 `AUTO_INCREMENT` 索引，它会为每一个订单的所有明细行生成一个独立的序列(第 11.11 节中讨论了这一类索引)。像下面这样创建 `inv_item` 表，使用 `inv_id` 列和定义为 `AUTO_INCREMENT` 的 `seq` 列作为联合主键：

```
CREATE TABLE inv_item
(
    inv_id INT UNSIGNED NOT NULL, # invoice 表 ID (由 invoice 表获取)
    seq    INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (inv_id, seq),
    qty    INT,                  # 数量
    description VARCHAR(40)      # 描述
);
```

和原来的表结构一样，`inv_id` 列将 `inv_item` 中的每一列关联到所属订单在 `invoice` 表中的行。另外，定义的索引使得 `inv_item` 表中属于同一订单的所有行的 `seq` 值从 1 开始赋予序列值。但是，现在两张表中都有了 `AUTO_INCREMENT` 列，你不能像以前那样输入订单信息了。来看看为什么不能那样做了，试试下面的：

```
INSERT INTO invoice (inv_id, date)
VALUES (NULL, CURDATE());
INSERT INTO inv_item (inv_id, qty, description)
VALUES (LAST_INSERT_ID(), 1, 'hammer');
INSERT INTO inv_item (inv_id, qty, description)
VALUES (LAST_INSERT_ID(), 3, 'nails, box');
INSERT INTO inv_item (inv_id, qty, description)
VALUES (LAST_INSERT_ID(), 12, 'bandage');
```

这些语句和以前的一样，但是由于 `inv_item` 表结构的改变，执行的结果也不一样了。向 `invoice` 表插入行的 `INSERT` 语句正确执行。第一个向 `inv_item` 表插入行的语句也正确执行；`LAST_INSERT_ID()` 返回 `invoice` 表中的 `inv_id` 值。然而，这个语句也生成了自己的 `AUTO_INCREMENT` 值 (`seq` 列)，这会改变 `LAST_INSERT_ID()` 的返回值，并导致主表的 `inv_id` 值不再可用。结果导致后续 `INSERT` 语句使用错误的 `inv_id` 值向 `inv_item` 表中插入行。

为了避免这个错误，把向主表中插入行时生成的序列值保存起来，用于明细表插入。为了保存主表生成的序列值，你可以使用 SQL 中的用户自定义变量，或者应用程序变量。

使用用户定义变量

将主表中的 AUTO_INCREMENT 值保存在一个用户定义变量中，用于明细表行插入：

```
INSERT INTO invoice (inv_id, date)
VALUES(NULL, CURDATE());
SET @inv_id = LAST_INSERT_ID();
INSERT INTO inv_item (inv_id, qty, description)
VALUES(@inv_id, 1, 'hammer');
INSERT INTO inv_item (inv_id, qty, description)
VALUES(@inv_id, 3, 'nails, box');
INSERT INTO inv_item (inv_id, qty, description)
VALUES(@inv_id, 12, 'bandage');
```

使用一个 API 变量

这个方法与前面的方法类似，但是只要在 API 内部应用。在主表中插入行，然后把 AUTO_INCREMENT 值保存到一个 API 变量，用于明细表插入。例如，在 Ruby 中，又可以通过数据库句柄 `insert_id` 属性获取 AUTO_INCREMENT 值，使得订单录以如下过程输入：

```
dbh.do("INSERT INTO invoice (inv_id, date) VALUES(NULL, CURDATE())")
inv_id = dbh.func(:insert_id)
sth = dbh.prepare("INSERT INTO inv_item (inv_id, qty, description)
VALUES(?, ?, ?)")
sth.execute(inv_id, 1, "hammer")
sth.execute(inv_id, 3, "nails, box")
sth.execute(inv_id, 12, "bandage")
```

11.14 将序列生成器用作计数器

Using Sequence Generators as Counters

问题

你只想计数，因此不必为每一个序列值创建一行。

解决方案

采用一个计数器占用一行的序列生成机制。

讨论

AUTO_INCREMENT 列对于给一些独立的行创建序列值是有用的。但是对于某些应用，你只想对一个事件发生的次数进行计数，这样为每一个序列值都创建一行就没有意义了。这样的例子包括网页或者标题广告的点击次数计数器、商品销售计数或者选举的投票统计等。对于这一类应用，你只需要一行来记录并更改计数值。MySQL 针对这一类应用提供了一

种机制，采用和 AUTO_INCREMENT 类似的机制处理计数值，你不但能够增加计数值，而且可以方便的获取该值。

为了对单个事件计数，你可以使用一个仅包含一行一列的表。例如，如果你在卖某本书，你可以创建一个表来记录销售情况：

```
CREATE TABLE booksales (copies INT UNSIGNED);
```

然而，如果你要统计不同的书，上面的方法就不奏效了。你显然不想为每一本书都单独创建一张只含一行的表。可选的方法是，如果你在一张表内包含了一列，该列为每一本书提供唯一标识，你就能够在单表内统计所有书。下面的 booksales 表，除了 copies 列统计销售数量外，还有 title 列保存每一本书的书名：

```
CREATE TABLE booksales
(
    title  VARCHAR(60) NOT NULL,      # 书名
    copies INT UNSIGNED NOT NULL,   # 销量
    PRIMARY KEY (title)
);
```

有不同的方法用于记录每一本书的销售情况：

- 把一本书的 copies 列初始化为 0：

```
INSERT INTO booksales (title,copies) VALUES('The Greater Trumps',0);
```

然后每售出一本书 copies 值递增 1：

```
UPDATE booksales SET copies = copies+1 WHERE title = 'The Greater Trumps';
```

这种方法要求你记住（首先）初始化每一本书的记录行，否则 UPDATE 将会失败。

- 在 INSERT 语句中加上 ON DUPLICATE KEY UPDATE，它将售出的第一本书的 copies 值初始化为 1，并在以后的销售中递增计数值：

```
INSERT INTO booksales (title,copies)
VALUES('The Greater Trumps',1)
ON DUPLICATE KEY UPDATE copies = copies+1;
```

这种方法更简单，因为你能够用相同的语句完成销售计数的初始化和更新。

执行下面的 SELECT 语句来查询销售计数（这样你就可以向客户显示一些消息，例如“你刚刚购买了 n 本书”）：

```
SELECT copies FROM booksales WHERE title = 'The Greater Trumps';
```

不幸的是，这样并不是非常正确。假设在你更新和查询计数值之间，有人又购买了同样的书（这样就增加了 copies 值）。这样 SELECT 语句不能查询到你刚刚更新过的计数值，而是最后更新过的值。换句话说，其他用户可能在你查询计数值之前对它做了修改。这个问题

题类似于之前讨论过的，你用 `MAX(col_name)` 而不是 `LAST_INSERT_ID()` 从一列中查询最新的 `AUTO_INCREMENT` 值（可能会出现的问题）。

对此有多种解决方法（例如将两个语句放入一个事务，或者锁定表），但是 MySQL 提供了一种基于 `LAST_INSERT_ID()` 的解决方法。如果你把 `LAST_INSERT_ID()` 作为表达式参数来调用，MySQL 会把它当作 `AUTO_INCREMENT` 值来处理。为了在 `booksales` 中应用这一特性，稍微修改更新计数值的语句：

```
INSERT INTO booksales (title,copies)
VALUES('The Greater Trumps',LAST_INSERT_ID(1))
ON DUPLICATE KEY UPDATE copies = LAST_INSERT_ID(copies+1);
```

这个语句在计数值初始化和更新中都使用了 `LAST_INSERT_ID(expr)` 子句。使用 `LAST_INSERT_ID()` 作为表达式参数，MySQL 就将刚表达式作为一个 `AUTO_INCREMENT` 值处理。然后你就可以使用无参数的 `LAST_INSERT_ID()` 函数来获取计数值：

```
SELECT LAST_INSERT_ID();
```

用这样的方法设置和查询 `copies` 列值，就算其他客户端同时对其进行了更新，你也能够获取你所设定的 `copies` 值。如果你通过程序语言 API 执行 `INSERT` 语句，而该 API 提供了最新 `AUTO_INCREMENT` 值的直接访问机制的话，你甚至不需要执行 `SELECT` 查询。例如，在 Python 中，你能使用 `insert_id()` 函数更新计数值，并获取最新的计数值：

```
cursor = conn.cursor ()
cursor.execute """
    INSERT INTO booksales (title,copies)
    VALUES('The Greater Trumps',LAST_INSERT_ID(1))
    ON DUPLICATE KEY UPDATE copies = LAST_INSERT_ID(copies+1)
"""
count = conn.insert_id ()
```

在 Java 中，可以这样操作：

```
Statement s = conn.createStatement ();
s.executeUpdate (
    "INSERT INTO booksales (title,copies)"
    + " VALUES('The Greater Trumps',LAST_INSERT_ID(1))"
    + " ON DUPLICATE KEY UPDATE copies = LAST_INSERT_ID(copies+1)");
long count = ((com.mysql.jdbc.Statement) s).getLastInsertID ();
s.close ();
```

用 `LAST_INSERT_ID(expr)` 来生成序列与 `AUTO_INCREMENT` 序列相比，有一些不同的属性：

- `AUTO_INCREMENT` 值每次递增 1，然而 `LAST_INSERT_ID(expr)` 生成的计数值可以增加你指定的任意值。例如，为了生成序列 10、20、30...每次使计数器的值增加 10。

你甚至不需要每次都使计数器增加相同的值。如果你售出了一打相同的书，而不是一本，你可以这样更新计数值：

```
INSERT INTO booksales (title, copies)
VALUES ('The Greater Trumps', LAST_INSERT_ID(12))
ON DUPLICATE KEY UPDATE copies = LAST_INSERT_ID(copies+12);
```

- 序列起始值可以是任意数值，甚至是负数。也可以使用一个负数作为增量生成一个递减序列（对于用来生成包含负数值的序列的一列，你需要从列定义中去除 `UNSIGNED` 定义）。
- 只需要简单地把计数器值设置为你所想要的值，就可以重置计数器。假设你想告诉消费者当月的销售情况，而不是所有销售记录（例如，你想显示一条消息“您是本月第 `n` 个客户”）。运行下面的语句，在每月开始的时候将计数器设为 0：

```
UPDATE booksales SET copies = 0;
```

- 你不想看到的一个特性是：不是所有情况下，用 `LAST_INSERT_ID(expr)` 生成的值都能用客户端查询方法获取。你可以在 `UPDATE` 或者 `INSERT` 语句之后获取它，但 `SET` 语句之后不能（获取该序列值）。如果你像下面这样产生一个值（在 Ruby 中，`insert_id` 返回的对应客户端值可能是 0，而不是 48）：

```
dbh.do("SET @x = LAST_INSERT_ID(48)")
seq = dbh.func(:insert_id)
```

查询客户端获取这种情况下的值：

```
seq = dbh.select_one("SELECT LAST_INSERT_ID()")[0]
```

参考

第 19.12 节中再次讨论了单行的序列值生成机制，（这一机制）是实现网页点击计数器的基本方法。

11.15 创建循环序列

Generating Repeating Sequences

问题

你需要创建含有循环的序列。

解决方案

生成一个序列，使用 `division` 和 `modulo` 操作符生成循环元素。

讨论

有些序列生成问题要求序列值能够循环使用。假设你正在生产药品或者汽车零件，你必须通过批号跟踪所有商品，如果以后发现了产品问题，要求召回售出的某一批产品。假设你把 12 个产品包装为 1 盒，6 盒包装为 1 箱。这种情况下，产品编号为 3 个部分：单品编号（1 到 12）、盒编号（1 到 6）和一个批号（从 1 到任意值）。

这个产品跟踪问题要求你维护三个计数器，因此你也许会考虑用下面的算法生成下一个产品标识编号：获取最新使用过的箱、盒和单品编号：

```
unit = unit + 1      # 增加产品数
if (unit > 12)        # 是否需要一个新包装盒?
{
    unit = 1          # 记为下一个盒的第一个产品
    box = box + 1
}
if (box > 6)          # 是否需要一个新的包装箱?
{
    box = 1           # 记为下一个箱的第一盒
    case = case + 1
}
```

保存新的箱、盒和产品编号。你确实可以这样实现一个算法。但是，也有可以简单的赋予每一个产品一个序列编号，然后由此生成对应的箱、盒和单品编号。序列编号可以由 AUTO_INCREMENT 列生成，或者由单行的序列生成器生成。根据序列编号生成箱、盒和单品编号的公式如下：

```
unit_num = ((seq - 1) % 12) + 1
box_num = int ((seq - 1) / 12) % 6) + 1
case_num = int ((seq - 1)/(6 * 12)) + 1
```

下表说明了序列值与对应的箱、盒和单品编号之间的关系：

序列值	箱	盒	单品
1	1	1	1
12	1	1	12
13	1	2	1
72	1	6	12
73	2	1	1
144	2	6	12

11.16 按行顺序输出数列查询

Numbering Query Output Rows Sequentially

问题

你想对查询结果中的行编号。

解决方案

如果你在写自己的应用程序，可以自己给每行加上编号。

讨论

与数据库内容无关的一种序列，是对一个查询结果输出行进行编号。当你使用程序语言 API（访问数据库），你可以维护一个计数器来对结果行编号，并将计数值与每一行内容一起显示出来。下面的 Python 程序使用 insects 表。它显示了 origin 列中非重复值的一个简单编号列表：

```
cursor = conn.cursor ()
cursor.execute ("SELECT DISTINCT origin FROM insect")
count = 1
for row in cursor.fetchall ():
    print count, row[0]
    count = count + 1
cursor.close ()
```

参考

mysql 提供了非显式的行编号方法，尽管你能使用一个用户定义变量在查询输出中加入一个行号列。生成满足你要求的查询结果的另一种方法是，用另外的程序过滤 mysql 输出，并加上行编号。第 1.27 节中讨论了这种技术。

使用多重表

Using Multiple Tables

12.0 引言

Introduction

前面章节讨论的例子中大部分使用的是单一表，但是对于任何具有普通复杂性的应用来说，你很可能会需要使用多重表。一些问题不能够简单地使用单一表来解决，并且关系数据库的真正能力只有在你从复合数据来源中整合信息时才能表现出来。使用多重表有如下几个理由：

- 将来自多个表中的行结合起来，获取的信息比从单个表中所能获取的更为全面。
- 保持多步操作的中间结果。
- 根据其他表的信息修改某个表中的记录行。

一个使用多重表的语句可能是多个表的连接 (join)，也可能是嵌套在另一个语句中的子查询，或者是多重 SELECT 语句查询结果的联合 (union)。前面的章节中，我们已经或多或少地接触过子查询了，因此本章主要的焦点将放在连接和联合上面，当然在必要时也会提到子查询。本章覆盖的主题如下：

连接不同的表以发现它们中相匹配或不匹配的行

为了解决这些问题，你必须知道连接操作的类型。内部连接 (inner joins) 可以展示某个表中哪些行与另一个表中的行相匹配。而外部连接 (outer joins) 既可以显示匹配行，也可以用于在某个表中找到与另一个表中的行不匹配的那些行。

将一个表与其自身相比较

解决某些问题时你需要将某些表与其自身相比较，这与在不同表间执行连接操作相类似，不同点在于你必须使用表别名以消除表引用的歧义。

使用联合以将多个结果集结合起来

对于某些查询，需要的信息包含了多个结果集，这些结果集可能是从不同的表中获取的，也可能是在同一个表中以不同的查询方式得到的。为了产生这样的结果，可以使用 UNION 将多重 SELECT 语句所获得的结果集联合起来。

删除不匹配的行

如果两个相关的数据集在关系分配上有缺陷，你可以确定哪些行是不匹配的。并且如果这些行是不需要的，你可以删除它们。

在不同数据库的表之间执行连接操作

当你使用多重表时，它们可能来源于同一个数据库，也可能来自不同的数据库。有时候，你甚至可能会需要使用来源于不同 MySQL 服务器上数据库中的表。在前两种情况下，你需要知道如何查阅不同的表中的列，这可能包括使用表的别名或者在表面前加上数据库名修饰。在第 3 种情况下，你可以建立跨类联合存储式数据库表（FEDERATED table）以使一个 MySQL 服务器能够自动访问位于另一个服务器上的表，或者为每个服务器打开一个连接并自己手动将其中的信息组合起来。

本章所使用的建表语句可以在目录表中找到。对于一些此处所讨论到的技术的实现脚本，可以参看连接与联合目录。

12.1 在表中找到与另一个表中的行相匹配的行

Finding Rows in One Table That Match Rows in Another

问题

你需要编写语句以查询多个表中的信息。

解决方案

使用连接（join）——即一个查询，它在 FROM 子句列出多重表并告诉 MySQL 如何匹配其中的信息。

讨论

连接的本质概念是将一个表中的行与另一个或更多的表中的行相联合。对于多重表来说，如果它们中的每个表只含有你所感兴趣的信息的一部分，那么连接使你能够将这些表中的信息结合起来。连接的输出行所包含的信息要多于从其中任何一个表中单独取出的行信息。

一个完全的连接将产生所有可能的行联合，即笛卡尔积。举例来说，将一个具有 100 行的表与另一个包含 200 行的表进行连接，产生的结果将会包含 100×200 即 20 000 个行。对于更大的表，或者对两个以上的表进行连接，由笛卡尔积产生的结果集将会膨胀得很大。正因如此，同时也因为你很少需要得到所有的联合，连接通常会包含 ON 或 USING 子句以指定如何在表间进行连接。（这需要每个表含有一个或多个包含共同信息的列，以便能

够从逻辑上将它们连接在一起。) 你还可以在连接中包含 WHERE 子句以限制选择哪些连接行。这些子句都能够缩小查询的范围。

本节将介绍基本的 join 语法，并展示当你查找表间的匹配时，连接是如何帮助你解决某些类型问题的。后面几节将介绍如何识别表间的不匹配，以及如何将一个表与其自身相比较。

下例假设你进行美术收藏，并且用下面两个表来记录你的藏品。artist 表列出了你想要收藏其作品的画家，painting 表列出了你已经购买的作品：

```
CREATE TABLE artist
(
    a_id INT UNSIGNED NOT NULL AUTO_INCREMENT, # 艺术家 ID
    name VARCHAR(30) NOT NULL,                  # 艺术家名字
    PRIMARY KEY (a_id),
    UNIQUE (name)
);
CREATE TABLE painting
(
    a_id INT UNSIGNED NOT NULL,                # 艺术家 ID
    p_id INT UNSIGNED NOT NULL AUTO_INCREMENT, # 油画 ID
    title VARCHAR(100) NOT NULL,                 # 油画名称
    state VARCHAR(2) NOT NULL,                  # 购买地
    price INT UNSIGNED,                        # 购买价格(美元)
    INDEX (a_id),
    PRIMARY KEY (p_id)
);
```

由于你刚开始收藏，因此表中只有如下几行：

```
mysql> SELECT * FROM artist ORDER BY a_id;
+-----+-----+
| a_id | name   |
+-----+-----+
|    1 | Da Vinci |
|    2 | Monet    |
|    3 | Van Gogh |
|    4 | Picasso  |
|    5 | Renoir   |
+-----+
mysql> SELECT * FROM painting ORDER BY a_id, p_id;
+-----+-----+-----+-----+
| a_id | p_id | title        | state | price |
+-----+-----+-----+-----+
|    1 |    1 | The Last Supper | IN    |  34   |
|    1 |    2 | The Mona Lisa  | MI    |  87   |
|    3 |    3 | Starry Night   | KY    |  48   |
|    3 |    4 | The Potato Eaters | KY    |  67   |
|    3 |    5 | The Rocks     | IA    |  33   |
|    5 |    6 | Les Deux Soeurs | NE    |  64   |
+-----+-----+-----+-----+
```

painting 表的 price 列值都很小，这暴露了你的收藏都是些廉价摹本，而不是真迹的事实。嗯，这很正常，谁能买得起原作呢？

每张表都只包含了你的藏品的部分信息。比如说，artist 表没有指出所收藏的是哪个画家的哪幅作品，而 painting 表列出了画家的 ID 而不是他们的名字。为了使用两个表的信息，你可以通过写一个执行连接的查询语句，以要求 MySQL 为你显示画家和作品的不同组合。连接语句在 FROM 关键字后面列出两个或更多个表的名字。在输出列的列表中，你可以列出从部分或所有已连接的表中所要取的列的名字，或者使用建立在这些列基础上的表达式，如 *tbl_name.** 选择给定表的所有列，或者 * 选择所有表的所有列。

最简单的连接包括两个表，并选择它们中的所有列。由于没有任何限制，连接产生的结果包括所有行的联合（即笛卡尔积）。下面为在 artist 和 painting 表之间的完全连接：

```
mysql> SELECT * FROM artist, painting;
+-----+-----+-----+-----+-----+-----+-----+
| a_id | name   | a_id | p_id | title      | state | price |
+-----+-----+-----+-----+-----+-----+-----+
| 1   | Da Vinci | 1   | 1   | The Last Supper | IN    | 34   |
| 2   | Monet    | 1   | 1   | The Last Supper | IN    | 34   |
| 3   | Van Gogh | 1   | 1   | The Last Supper | IN    | 34   |
| 4   | Picasso   | 1   | 1   | The Last Supper | IN    | 34   |
| 5   | Renoir   | 1   | 1   | The Last Supper | IN    | 34   |
| 1   | Da Vinci | 1   | 2   | The Mona Lisa  | MI    | 87   |
| 2   | Monet    | 1   | 2   | The Mona Lisa  | MI    | 87   |
| 3   | Van Gogh | 1   | 2   | The Mona Lisa  | MI    | 87   |
| 4   | Picasso   | 1   | 2   | The Mona Lisa  | MI    | 87   |
| 5   | Renoir   | 1   | 2   | The Mona Lisa  | MI    | 87   |
| 1   | Da Vinci | 3   | 3   | Starry Night   | KY    | 48   |
| 2   | Monet    | 3   | 3   | Starry Night   | KY    | 48   |
| 3   | Van Gogh | 3   | 3   | Starry Night   | KY    | 48   |
| 4   | Picasso   | 3   | 3   | Starry Night   | KY    | 48   |
| 5   | Renoir   | 3   | 3   | Starry Night   | KY    | 48   |
| 1   | Da Vinci | 3   | 4   | The Potato Eaters | KY   | 67   |
| 2   | Monet    | 3   | 4   | The Potato Eaters | KY   | 67   |
| 3   | Van Gogh | 3   | 4   | The Potato Eaters | KY   | 67   |
| 4   | Picasso   | 3   | 4   | The Potato Eaters | KY   | 67   |
| 5   | Renoir   | 3   | 4   | The Potato Eaters | KY   | 67   |
| 1   | Da Vinci | 3   | 5   | The Rocks     | IA    | 33   |
| 2   | Monet    | 3   | 5   | The Rocks     | IA    | 33   |
| 3   | Van Gogh | 3   | 5   | The Rocks     | IA    | 33   |
| 4   | Picasso   | 3   | 5   | The Rocks     | IA    | 33   |
| 5   | Renoir   | 3   | 5   | The Rocks     | IA    | 33   |
| 1   | Da Vinci | 5   | 6   | Les Deux Soeurs | NE   | 64   |
| 2   | Monet    | 5   | 6   | Les Deux Soeurs | NE   | 64   |
| 3   | Van Gogh | 5   | 6   | Les Deux Soeurs | NE   | 64   |
| 4   | Picasso   | 5   | 6   | Les Deux Soeurs | NE   | 64   |
| 5   | Renoir   | 5   | 6   | Les Deux Soeurs | NE   | 64   |
+-----+-----+-----+-----+-----+-----+-----+
```

该语句的输出说明了为什么完全的连接通常是无用的，因为它产生的输出项太多了，以至于结果没有意义。显然，你维护这些表不是为了像前面语句所做的那样，将每个画家和每幅作品都匹配起来。此例中毫无限制的连接没有任何价值。

为了解决一些有意义的问题，你必须在连接两张表时，只产生相关的匹配，而通过包含适当的连接条件你可以做到这一点。举例来说，为了产生一幅画作及其画家的列表，你可以使用简单的 WHERE 子句将两个表的行关联起来，该子句通过两张表都有的画家的 ID 值进行匹配，作为它们之间的链接：

```
mysql> SELECT * FROM artist, painting
-> WHERE artist.a_id = painting.a_id;
+-----+-----+-----+-----+-----+-----+-----+
| a_id | name   | a_id | p_id | title      | state | price |
+-----+-----+-----+-----+-----+-----+-----+
| 1   | Da Vinci | 1   | 1   | The Last Supper | IN    | 34   |
| 1   | Da Vinci | 1   | 2   | The Mona Lisa  | MI    | 87   |
| 3   | Van Gogh | 3   | 3   | Starry Night    | KY    | 48   |
| 3   | Van Gogh | 3   | 4   | The Potato Eaters | KY    | 67   |
| 3   | Van Gogh | 3   | 5   | The Rocks      | IA    | 33   |
| 5   | Renoir   | 5   | 6   | Les Deux Soeurs | NE    | 64   |
+-----+-----+-----+-----+-----+-----+-----+
```

WHERE 子句中的列名包含了表名的修饰符，这清楚地指出了比较的是哪个 a_id。输出指明了在你的收藏中，每幅画作的作者是谁，以及每个画家有哪些作品。

达到同样连接目的的另一种方式是使用 INNER JOIN 而不是逗号操作符，并在 ON 子句中指明匹配条件：

```
mysql> SELECT * FROM artist INNER JOIN painting
-> ON artist.a_id = painting.a_id;
+-----+-----+-----+-----+-----+-----+
| a_id | name   | a_id | p_id | title      | state | price |
+-----+-----+-----+-----+-----+-----+
| 1   | Da Vinci | 1   | 1   | The Last Supper | IN    | 34   |
| 1   | Da Vinci | 1   | 2   | The Mona Lisa  | MI    | 87   |
| 3   | Van Gogh | 3   | 3   | Starry Night    | KY    | 48   |
| 3   | Van Gogh | 3   | 4   | The Potato Eaters | KY    | 67   |
| 3   | Van Gogh | 3   | 5   | The Rocks      | IA    | 33   |
| 5   | Renoir   | 5   | 6   | Les Deux Soeurs | NE    | 64   |
+-----+-----+-----+-----+-----+-----+
```

在特殊情况下，即两个表的匹配列名字相同，并且是使用=操作符进行比较的话，你可以使用 INNER JOIN 并搭配 USING 子句作为替代，从而不需要表修饰符，而且连接列只需要被列出一次：

```
mysql> SELECT * FROM artist INNER JOIN painting
-> USING(a_id);
+-----+-----+-----+-----+-----+
| a_id | name   | p_id | title      | state | price |
+-----+-----+-----+-----+-----+
```

```

| 1 | Da Vinci | 1 | The Last Supper | IN | 34 |
| 1 | Da Vinci | 2 | The Mona Lisa | MI | 87 |
| 3 | Van Gogh | 3 | Starry Night | KY | 48 |
| 3 | Van Gogh | 4 | The Potato Eaters | KY | 67 |
| 3 | Van Gogh | 5 | The Rocks | IA | 33 |
| 5 | Renoir | 6 | Les Deux Soeurs | NE | 64 |
+-----+-----+-----+-----+-----+

```

注意当你在查询中使用 USING 子句时, SELECT * 只返回所有连接列中的一个实例 (a_id)。

ON、USING 或 WHERE 中的任何一个都可以包含比较操作, 那么你怎么知道每个子句中该放入什么样的连接条件呢? 根据经验规则, 通常使用 ON 或 USING 来指定如何连接表, 而使用 WHERE 子句限制选择哪些已连接的行。举例来说, 如果根据 a_id 列连接表, 但是只选择在肯塔基州 (Kentucky) 购买的画作, 那么使用 ON (或 USING) 子句匹配两个表中的行, 使用 WHERE 子句来检验 state 列:

```

mysql> SELECT * FROM artist INNER JOIN painting
   -> ON artist.a_id = painting.a_id
   -> WHERE painting.state = 'KY';
+-----+-----+-----+-----+-----+
| a_id | name | a_id | p_id | title | state | price |
+-----+-----+-----+-----+-----+
| 3 | Van Gogh | 3 | 3 | Starry Night | KY | 48 |
| 3 | Van Gogh | 3 | 4 | The Potato Eaters | KY | 67 |
+-----+-----+-----+-----+-----+

```

前面的查询使用了 SELECT * 以选择所有的列。为了更好地选择一个语句应该显示哪些列, 你可以提供所感兴趣的那些列的名字列表:

```

mysql> SELECT artist.name, painting.title, painting.state, painting.price
   -> FROM artist INNER JOIN painting
   -> ON artist.a_id = painting.a_id
   -> WHERE painting.state = 'KY';
+-----+-----+-----+-----+
| name | title | state | price |
+-----+-----+-----+-----+
| Van Gogh | Starry Night | KY | 48 |
| Van Gogh | The Potato Eaters | KY | 67 |
+-----+-----+-----+

```

编写连接查询时不一定只限于两个表。假设在前面的查询结果中, 你需要查看的不是州名简写而是完整的州名, 前面章节中使用的 states 表将州的简写与名称映射到一起, 因此你可以将之加入到前面的查询中以显示完整的州名:

```

mysql> SELECT artist.name, painting.title, states.name, painting.price
   -> FROM artist INNER JOIN painting INNER JOIN states
   -> ON artist.a_id = painting.a_id AND painting.state = states.abbrev;
+-----+-----+-----+-----+
| name | title | name | price |
+-----+-----+-----+

```

Da Vinci	The Last Supper	Indiana	34
Da Vinci	The Mona Lisa	Michigan	87
Van Gogh	Starry Night	Kentucky	48
Van Gogh	The Potato Eaters	Kentucky	67
Van Gogh	The Rocks	Iowa	33
Renoir	Les Deux Soeurs	Nebraska	64

三种方式的连接的另一种常见用途是枚举多对多的关系，参见第 12.5 节中的例子。

通过在你的连接中包含适当的条件，你可以解决一些非常特殊的问题，比如下面的例子：

- 哪幅画是梵高 (Van Gogh) 的作品？为了回答这个问题，可以使用 `a_id` 值查找匹配行；再用 `WHERE` 子句限制那些包含该画家名字的行的输出，只从中选择作品名称 (`title`)：

```
mysql> SELECT painting.title
    -> FROM artist INNER JOIN painting ON artist.a_id = painting.a_id
    -> WHERE artist.name = 'Van Gogh';
+-----+
| title |
+-----+
| Starry Night |
| The Potato Eaters |
| The Rocks |
+-----+
```

- 蒙娜丽莎 (Mona Lisa) 是谁画的？你又一次使用 `a_id` 列以连接行，但是这一次 `WHERE` 子句限制那些包含画名的行的输出，以从这些行中选择画家的名字：

```
mysql> SELECT artist.name
    -> FROM artist INNER JOIN painting ON artist.a_id = painting.a_id
    -> WHERE painting.title = 'The Mona Lisa';
+-----+
| name |
+-----+
| Da Vinci |
+-----+
```

- 你在肯塔基 (Kentucky) 或印第安纳 (Indiana) 州购买了哪些画家的画作？这与前一个语句有些类似，通过检验 `painting` 表中的 `a_id` 列以确定将哪些行与 `artist` 表进行连接：

```
mysql> SELECT DISTINCT artist.name
    -> FROM artist INNER JOIN painting ON artist.a_id = painting.a_id
    -> WHERE painting.state IN ('KY','IN');
+-----+
| name |
+-----+
| Da Vinci |
| Van Gogh |
+-----+
```

该语句使用了 DISTINCT 以将每个画家的名字只显示一次。如果没有 DISTINCT，你将会发现 Van Gogh 被列出了两次，这是因为你在肯塔基获得了两幅 Van Gogh 的作品。

- 连接还可以与聚集函数一起使用以产生表摘要（summaries）。举个例子，为了找到对于每个画家你各收藏了多少作品，可以使用这条语句：

```
mysql> SELECT artist.name, COUNT(*) AS 'number of paintings'
-> FROM artist INNER JOIN painting ON artist.a_id = painting.a_id
-> GROUP BY artist.name;
+-----+-----+
| name | number of paintings |
+-----+-----+
| Da Vinci | 2 |
| Renoir | 1 |
| Van Gogh | 3 |
+-----+-----+
```

更详细的语句可以显示你为每个画家的作品花费了多少钱，包括总和以及每幅作品平均的花费：

```
mysql> SELECT artist.name,
-> COUNT(*) AS 'number of paintings',
-> SUM(painting.price) AS 'total price',
-> AVG(painting.price) AS 'average price'
-> FROM artist INNER JOIN painting ON artist.a_id = painting.a_id
-> GROUP BY artist.name;
+-----+-----+-----+-----+
| name | number of paintings | total price | average price |
+-----+-----+-----+-----+
| Da Vinci | 2 | 121 | 60.5000 |
| Renoir | 1 | 64 | 64.0000 |
| Van Gogh | 3 | 148 | 49.3333 |
+-----+-----+-----+-----+
```

注意该语句只为在 artist 表中你实际购买过其作品的画家产生输出。（举例来说，artist 表中的 Monet 没有在概要中出现，因为你还没有他的任何画作。）如果你想要表概要包括所有的画家，即使你还没有收藏他们的作品，你必须明确地使用另一种连接——外部连接（outer join）：

- 通过逗号操作符或 INNER JOIN 所编写的连接为内部连接，这意味着它们只为某个表中与其他表的值相匹配的值产生结果。
- 外部连接同样可以产生这些匹配，此外还可以为你显示某个表中的哪些值与另外一个表中的值是不相符的。第 12.2 节介绍了外部连接。

连接与索引

因为连接会很容易导致 MySQL 产生大量的行组合，所以确认你所要比较的列已经被索引是个好主意。否则，在表规模增长时性能会下降得很快。对于 artist 和 painting 表，连接是基于每个表中的 a_id 列值产生的。如果你向前查看这些表的建表语句，你会发现 a_id 在每张表中都已被索引。

`tbl_name.col_name` 记号在列名前使用表名进行修饰，这在连接中总是允许的，但是如果名字只出现在要连接表中的一个表内，那么可以被缩短为 `col_name`。在那种情况下，MySQL 可以清楚地确定该列是来自于哪个表，而表名修饰符不是必须的。我们不能在下面的连接中这样做，因为两个表都含有 a_id 列，因此列引用会有二义性：

```
mysql> SELECT * FROM artist INNER JOIN painting ON a_id = a_id;
ERROR 1052 (23000): Column 'a_id' in on clause is ambiguous
```

作为对比，下面的查询是非二义的。每个 a_id 的实例都使用适当的表名修饰，只有 artist 含有 name 列，只有 painting 含有 title 和 state 列：

```
mysql> SELECT name, title, state FROM artist INNER JOIN painting
-> ON artist.a_id = painting.a_id;
+-----+-----+-----+
| name | title | state |
+-----+-----+-----+
| Da Vinci | The Last Supper | IN |
| Da Vinci | The Mona Lisa | MI |
| Van Gogh | Starry Night | KY |
| Van Gogh | The Potato Eaters | KY |
| Van Gogh | The Rocks | IA |
| Renoir | Les Deux Soeurs | NE |
+-----+-----+-----+
```

为了使阅读者更清楚地掌握语句的含义，即使在不是必须的情况下，修饰列名也是有好处的，因此我倾向于在连接例子中使用修饰名。

如果你不希望在修饰列引用时需要写完整的表名，可以给每个表一个别名，并使用此别名指向它的列。下面两个语句是等价的：

```
SELECT artist.name, painting.title, states.name, painting.price
  FROM artist INNER JOIN painting INNER JOIN states
    ON artist.a_id = painting.a_id AND painting.state = states.abbrev;

SELECT a.name, p.title, s.name, p.price
  FROM artist AS a INNER JOIN painting AS p INNER JOIN states AS s
    ON a.a_id = p.a_id AND p.state = s.abbrev;
```

在 AS *alias_name* 子句中，AS 是可选的。

对于选择多行的复杂语句，别名可以节省很多打字输入。此外，对于某类语句，别名不仅更方便而且也是必须的，当我们进入自连接（self-joins）主题（第 12.3 节）时，这会变得很明显。

12.2 查找与其他表不匹配的行

Finding Rows with No Match in Another Table

问题

你需要在某个表中找到与另一个表不匹配的那些行，或者你需要产生一个基于表间连接的列表，在该列表中，对第一个表的每一行都有对应的条目，甚至当该行与第二个表并不匹配时也不例外。

解决方案

使用外部连接——LEFT JOIN 或者 RIGHT JOIN。

讨论

第 12.1 节关注的是内部连接，及在表之间发现匹配的连接。然而，解答某些问题需要确定哪些行并没有匹配（或者，换个说法，哪些行含有另一个表中所缺少的那些值）。举个例子，你或许需要知道在 artist 表中哪些画家的作品你还没有拥有。同样类型的问题在其他环境中也会出现，下面为一些例子：

- 你从事销售工作，并掌握了一个潜在客户的列表，以及已经订购你产品的客户列表。你需要在第一个列表中找出那些没有出现在第二个列表的客户。
- 你有一个棒球运动员的列表，和另一个打出过全垒打的运动员列表，并且你想知道第一个表中的哪些运动员没有打出过全垒打。解决方法是在第一个列表中找出那些没有出现在第二个表的运动员。

对于这些类型的问题，使用外部连接是必要的。和内部连接一样，外部连接也可以发现表间的匹配。但是与内部连接不同的是，外部连接还可以确定一个表中的哪些行与另一个表不相匹配。两种类型的外部连接为 LEFT JOIN 和 RIGHT JOIN。

为了了解为什么外部连接是有用的，让我们考虑一下确定 artist 表中哪些画家没有出现在 painting 表中的问题。到目前为止，这些表都比较小，因此通过观察很容易知道结果：

```
mysql> SELECT * FROM artist ORDER BY a_id;
+-----+
```

```

| a_id | name      |
+-----+-----+
| 1   | Da Vinci |
| 2   | Monet     |
| 3   | Van Gogh  |
| 4   | Picasso   |
| 5   | Renoir    |
+-----+-----+
mysql> SELECT * FROM painting ORDER BY a_id, p_id;
+-----+-----+-----+-----+
| a_id | p_id | title          | state | price |
+-----+-----+-----+-----+
| 1   | 1   | The Last Supper | IN    | 34   |
| 1   | 2   | The Mona Lisa   | MI    | 87   |
| 3   | 3   | Starry Night    | KY    | 48   |
| 3   | 4   | The Potato Eaters | KY    | 67   |
| 3   | 5   | The Rocks       | IA    | 33   |
| 5   | 6   | Les Deux Soeurs | NE    | 64   |
+-----+-----+-----+-----+

```

通过查看这些表，你发现了你没有 Monet 和 Picasso 的作品（在 painting 表中没有 a_id 值为 2 或 4 的行）。但是当你获得的画作越来越多，painting 表也越来越大，通过肉眼观察来回答这个问题将不再是件容易的事。那么你可以用 SQL 来回答它吗？当然可以，通常解决这个问题的第一个尝试是采用类似下面的语句，该语句使用了条件不等式以在两表间寻找不匹配项：

```

mysql> SELECT * FROM artist INNER JOIN painting
-> ON artist.a_id != painting.a_id;
+-----+-----+-----+-----+-----+-----+-----+
| a_id | name      | a_id | p_id | title          | state | price |
+-----+-----+-----+-----+-----+-----+-----+
| 2   | Monet     | 1   | 1   | The Last Supper | IN    | 34   |
| 3   | Van Gogh  | 1   | 1   | The Last Supper | IN    | 34   |
| 4   | Picasso   | 1   | 1   | The Last Supper | IN    | 34   |
| 5   | Renoir    | 1   | 1   | The Last Supper | IN    | 34   |
| 2   | Monet     | 1   | 2   | The Mona Lisa   | MI    | 87   |
| 3   | Van Gogh  | 1   | 2   | The Mona Lisa   | MI    | 87   |
| 4   | Picasso   | 1   | 2   | The Mona Lisa   | MI    | 87   |
| 5   | Renoir    | 1   | 2   | The Mona Lisa   | MI    | 87   |
| 1   | Da Vinci | 3   | 3   | Starry Night    | KY    | 48   |
| 2   | Monet     | 3   | 3   | Starry Night    | KY    | 48   |
| 4   | Picasso   | 3   | 3   | Starry Night    | KY    | 48   |
| 5   | Renoir    | 3   | 3   | Starry Night    | KY    | 48   |
| 1   | Da Vinci | 3   | 4   | The Potato Eaters | KY    | 67   |
| 2   | Monet     | 3   | 4   | The Potato Eaters | KY    | 67   |
| 4   | Picasso   | 3   | 4   | The Potato Eaters | KY    | 67   |
| 5   | Renoir    | 3   | 4   | The Potato Eaters | KY    | 67   |
| 1   | Da Vinci | 3   | 5   | The Rocks       | IA    | 33   |
| 2   | Monet     | 3   | 5   | The Rocks       | IA    | 33   |
| 4   | Picasso   | 3   | 5   | The Rocks       | IA    | 33   |
| 5   | Renoir    | 3   | 5   | The Rocks       | IA    | 33   |
| 1   | Da Vinci | 5   | 6   | Les Deux Soeurs | NE    | 64   |
| 2   | Monet     | 5   | 6   | Les Deux Soeurs | NE    | 64   |
| 3   | Van Gogh | 5   | 6   | Les Deux Soeurs | NE    | 64   |
+-----+-----+-----+-----+-----+-----+-----+

```

	4	Picasso		5		6	Les Deux Soeurs	NE		64	
--	---	---------	--	---	--	---	-----------------	----	--	----	--

输出显然是不正确的。(比如，它错误地指出每幅作品是由几个不同的画家画的。) 问题在于该语句产生了两个表间所有画家的 ID 不同的值组合列表，而实际上你所需要的是一个 artist 表的值列表，且该列表中的所有值从未在 painting 表中出现过。此处的麻烦是内部连接只能基于两个表中都出现的值组合产生结果，却不能告诉你任何一个只在两表之一中缺少的值。

当面对在某个表中寻找与另一个表不匹配(即另一个表所缺少)的值的问题时，你应该养成如此思维的习惯，“啊，这是一个 LEFT JOIN 问题。” LEFT JOIN 是外部连接的一种：它与内部连接类似，即试图将第一个(left)表和第二个(right)的表中的行相匹配。但是另一点，如果 left 表中没有与 right 表相匹配的行，LEFT JOIN 仍将产生一行——该行中所有来自 right 表的列被设为 NULL。这意味着你可以通过查找 NULL 来发现 right 表中所缺少的值。通过循序渐进的工作，更容易理解这是如何发生的，因此我们一开始以内部连接方式显示匹配行：

```
mysql> SELECT * FROM artist INNER JOIN painting
-> ON artist.a_id = painting.a_id;
+-----+-----+-----+-----+-----+-----+-----+
| a_id | name   | a_id | p_id | title          | state | price |
+-----+-----+-----+-----+-----+-----+-----+
| 1    | Da Vinci | 1    | 1    | The Last Supper | IN    | 34   |
| 1    | Da Vinci | 1    | 2    | The Mona Lisa  | MI    | 87   |
| 3    | Van Gogh | 3    | 3    | Starry Night    | KY    | 48   |
| 3    | Van Gogh | 3    | 4    | The Potato Eaters | KY    | 67   |
| 3    | Van Gogh | 3    | 5    | The Rocks       | IA    | 33   |
| 5    | Renoir   | 5    | 6    | Les Deux Soeurs | NE    | 64   |
+-----+-----+-----+-----+-----+-----+-----+
```

在这个输出中，第一个 a_id 列来自于 artist 表，而第二个则来自于 painting 表。现在将该结果与 LEFT JOIN 的输出进行对比，LEFT JOIN 语句的写法非常类似 INNER JOIN：

```
mysql> SELECT * FROM artist LEFT JOIN painting
-> ON artist.a_id = painting.a_id;
+-----+-----+-----+-----+-----+-----+-----+
| a_id | name   | a_id | p_id | title          | state | price |
+-----+-----+-----+-----+-----+-----+-----+
| 1    | Da Vinci | 1    | 1    | The Last Supper | IN    | 34   |
| 1    | Da Vinci | 1    | 2    | The Mona Lisa  | MI    | 87   |
| 2    | Monet    | NULL | NULL | NULL           | NULL  | NULL |
| 3    | Van Gogh | 3    | 3    | Starry Night    | KY    | 48   |
| 3    | Van Gogh | 3    | 4    | The Potato Eaters | KY    | 67   |
| 3    | Van Gogh | 3    | 5    | The Rocks       | IA    | 33   |
| 4    | Picasso  | NULL | NULL | NULL           | NULL  | NULL |
+-----+-----+-----+-----+-----+-----+-----+
```

| 5 | Renoir | 5 | 6 | Les Deux Soeurs | NE | 64 |



输出与内部连接的结果相近，但是 LEFT JOIN 还为 artist 表中的每一行产生了至少一个输出行，包括那些在 painting 表中没有匹配项的行。对于那些输出行，所有来自 painting 表的列都被设为 NULL，而在内部连接中则不会产生这样的行。

下一步，为了将输出限制为只包含不匹配的 artist 行，需要增加在任意 painting 列的值中查找 NULL 的 WHERE 子句，因为输出在与 painting 表匹配的情况下不可能包含 NULL。这种做法过滤掉了内部连接所产生的行，只剩下由外部连接所产生的那些行。

```
mysql> SELECT * FROM artist LEFT JOIN painting
    -> ON artist.a_id = painting.a_id
    -> WHERE painting.a_id IS NULL;
+-----+-----+-----+-----+-----+-----+
| a_id | name | a_id | p_id | title | state | price |
+-----+-----+-----+-----+-----+-----+
|   2 | Monet | NULL | NULL | NULL | NULL | NULL |
|   4 | Picasso | NULL | NULL | NULL | NULL | NULL |
+-----+-----+-----+-----+-----+-----+
```

最后，为了只显示在 artist 表中却不被 painting 表所拥有的值，需要缩短输出列的列表，以使其只包含来自于 artist 表的列。LEFT JOIN 列出了包含了的左表中的行，而该行所包含的 a_id 值在右表中没有出现。

```
mysql> SELECT artist.* FROM artist LEFT JOIN painting
    -> ON artist.a_id = painting.a_id
    -> WHERE painting.a_id IS NULL;
+-----+-----+
| a_id | name |
+-----+-----+
|   2 | Monet |
|   4 | Picasso |
+-----+-----+
```

一种类似的操作可以用来报告左表中的每个值并指示其是否位于右表。为了实现这个目的，需要执行 LEFT JOIN，其中对同时出现在右表中的每个左表值进行计数。下面的语句列出了 artist 表中的每个画家，并且显示你是否已经拥有了这些画家的作品：

```
mysql> SELECT artist.name,
    -> IF(COUNT(painting.a_id)>0,'yes','no') AS 'in collection'
    -> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
    -> GROUP BY artist.name;
+-----+-----+
| name | in collection |
+-----+-----+
| Da Vinci | yes |
| Monet | no |
| Picasso | no |
+-----+-----+
```

```

| Renoir | yes |
| Van Gogh | yes |
+-----+-----+

```

RIGHT JOIN 是另一种外部连接，它与 LEFT JOIN 类似，只不过将左表和右表的角色调换过来。从语义上说，RIGHT JOIN 强制匹配过程为右表的每一条记录产生一行，即使左表中不存在相应的行。在语法上，tbl1 LEFT JOIN tbl2 和 tbl2 RIGHT JOIN tbl1 是等价的，这意味着你可以像下面这样重写前面的 LEFT JOIN 语句，将之转换为 RIGHT JOIN，并产生同样的结果：

```

mysql> SELECT artist.name,
    -> IF(COUNT(painting.a_id)>0,'yes','no') AS 'in collection'
    -> FROM painting RIGHT JOIN artist ON artist.a_id = painting.a_id
    -> GROUP BY artist.name;
+-----+-----+
| name | in collection |
+-----+-----+
| Da Vinci | yes |
| Monet | no |
| Picasso | no |
| Renoir | yes |
| Van Gogh | yes |
+-----+-----+

```

编写 LEFT JOIN 和 RIGHT JOIN 查询的其他方法

在本书的其他地方，出于简便考虑通常只使用 LEFT JOIN，但是如果你将左右表的角色对调，它们都可以被转化为 RIGHT JOIN。与 INNER JOIN 一样，在外部连接中，如果两个表中待匹配的列名相同，并且是使用=操作符进行比较的话，你可以使用 USING 子句取代 ON 子句。举例来说，下面两条语句是等价的：

```

SELECT * FROM t1 LEFT JOIN t2 ON t1.n = t2.n;
SELECT * FROM t1 LEFT JOIN t2 USING (n);

```

下面两条也是：

```

SELECT * FROM t1 RIGHT JOIN t2 ON t1.n = t2.n;
SELECT * FROM t1 RIGHT JOIN t2 USING (n);

```

在特殊情况下，你希望根据两个表中的所有列进行比较，那么可以使用 NATURAL LEFT JOIN 或者 NATURAL RIGHT JOIN 并省去 ON 或 USING 子句：

```

SELECT * FROM t1 NATURAL LEFT JOIN t2;
SELECT * FROM t1 NATURAL RIGHT JOIN t2;

```

参考

正如本节中所展示的那样，LEFT JOIN 对于查找在另一个表中无匹配的值，或者显示每个值是否有匹配是很有用的。LEFT JOIN 还可能被用于产生包括表中所有条目的概要，即使

其中某些条目无内容可被概要，这在刻画主表（master table）和从表（detail table）的关系时是很常见的。举个例子，LEFT JOIN 可以产生“每个顾客的总销售额”列表，其中包含了所有的顾客，虽然有部分顾客在表概要时还没有购买任何产品。（参考第 12.4 节以获取关于主-从表的更多信息。）

当你收到两个可能具有相关性的数据文件时，你需要确定它们是否真的相关，那么你可以使用 LEFT JOIN 执行一致性检查。（也就是说，你需要检查它们关系的完整性。）将每个文件导入到 MySQL 表中，并运行两个 LEFT JOIN 语句以确定是否在其中任何一个表中存在独立的行——在另一个表中无匹配项的行。第 12.13 节讨论了如何识别（并选择删除）这些独立行。

12.3 将表与自身进行比较

Comparing a Table to Itself

问题

你希望将表中的行与同一个表中其他行进行比较。举例来说，你需要在你的收藏中找到创作“吃土豆的人（The Potato Eaters）”的画家的所有作品，或者你想要知道在 states 表中的哪些州与纽约是同一年加入联邦的，再或者你需要了解与其他任一州不在同一年加入联邦的州列表。

解决方案

需要与表自身进行比较的问题牵涉到一个被称之为自连接（self-join）的操作。它的执行方式与其他的连接非常相似，除了一点，即你必须始终使用表别名，以便能够在一个语句中以不同方式引用同一个表。

讨论

将一个表与另一个表进行连接的特殊情况是两个表为同一个，这称之为自连接。尽管不少人在一开始会对这种想法感到迷惑和奇怪，但这是完全合理的。你很可能将发现你经常使用自连接，因为它们是如此重要。

当你希望知道表中哪对元素满足某些条件时，你需要使用自连接。举个例子，假设你最喜欢的画作是“吃土豆的人”（The Potato Eaters），并且你想要在你的收藏中找到该画的作者的所有作品。那么可以如下操作：

1. 在 painting 表中找到包含了画名为 The Potato Eaters 的行，以获得它的 a_id 值。

2. 使用 a_id 值匹配表中具有相同 a_id 值的其他行。

3. 显示所有匹配行的画名。

开始时的画家的 ID 和画作的 titles 如下所示：

```
mysql> SELECT a_id, title FROM painting ORDER BY a_id;
+-----+-----+
| a_id | title
+-----+-----+
| 1    | The Last Supper
| 1    | The Mona Lisa
| 3    | Starry Night
| 3    | The Potato Eaters
| 3    | The Rocks
| 5    | Les Deux Soeurs
+-----+-----+
```

一个不使用连接而找到正确画名的 2-步方法为先用一个语句查找画家的 ID，再在第一个语句中使用该 ID 选择匹配的行：

```
mysql> SELECT @id := a_id FROM painting WHERE title = 'The Potato Eaters';
+-----+
| @id := a_id |
+-----+
|      3      |
+-----+
mysql> SELECT title FROM painting WHERE a_id = @id;
+-----+
| title
+-----+
| Starry Night
| The Potato Eaters
| The Rocks
+-----+
```

另一种不同的解决方式只需要一个语句，即使用自连接，其诀窍在于如何选择使用适当的符号。编写连接表自身的语句的第一次尝试通常与下类似：

```
mysql> SELECT title
   -> FROM painting INNER JOIN painting
   -> ON a_id = a_id;
   -> WHERE title = 'The Potato Eaters';
ERROR 1066 (42000): Not unique table/alias: 'painting'
```

该语句的问题在于列引用具有二义性，MySQL 不能确定任一给定的列名所引用的 painting 表的实例。解决方法是至少给其中一个表实例起个别名，以便能够使用不同的表修饰符区分列引用。下面的语句展示具体做法，即使用别名 p1 和 p2 以两种方式引用 painting 表：

```
mysql> SELECT p2.title
   -> FROM painting AS p1 INNER JOIN painting AS p2
   -> ON p1.a_id = p2.a_id
```

```
-> WHERE p1.title = 'The Potato Eaters';
+-----+
| title      |
+-----+
| Starry Night   |
| The Potato Eaters |
| The Rocks    |
+-----+
```

该语句的输出表明了自连接的一些典型特征：当你开始在一个表实例中使用引用值（The Potato Eaters）以在第二个表实例中查找相匹配的行（同一画家的画作）时，其输出会包含所引用的值。这是有意义的：毕竟，该引用与其自身也是匹配的。如果你只需要找出同一家画家的其他作品，那么可以显式地从输出中排除该引用值：

```
mysql> SELECT p2.title
-> FROM painting AS p1 INNER JOIN painting AS p2
-> ON p1.a_id = p2.a_id
-> WHERE p1.title = 'The Potato Eaters' AND p2.title != 'The Potato Eaters';
+-----+
| title      |
+-----+
| Starry Night   |
| The Rocks    |
+-----+
```

排除引用值的一个更常用不需要逐字地列出该值的方法，是指明你不需要在输出行中包含与引用相同的画名，无论这个画名是什么：

```
mysql> SELECT p2.title
-> FROM painting AS p1 INNER JOIN painting AS p2
-> ON p1.a_id = p2.a_id
-> WHERE p1.title = 'The Potato Eaters' AND p2.title != p1.title
+-----+
| title      |
+-----+
| Starry Night   |
| The Rocks    |
+-----+
```

前面的语句在两个表实例中通过对 ID 值的比较来匹配行，实际上任何一种值都可以被用作此用途。举例来说，使用 states 表来回答“哪些州与纽约是同一年加入联邦的？”这个问题，即根据位于表中 statehood 列的日期的年部分进行两两对比：

```
mysql> SELECT s2.name, s2.statehood
-> FROM states AS s1 INNER JOIN states AS s2
-> ON YEAR(s1.statehood) = YEAR(s2.statehood)
-> WHERE s1.name = 'New York'
-> ORDER BY s2.name;
+-----+-----+
```

```
+-----+-----+
| name | statehood |
+-----+-----+
| Connecticut | 1788-01-09 |
| Georgia | 1788-01-02 |
| Maryland | 1788-04-28 |
| Massachusetts | 1788-02-06 |
| New Hampshire | 1788-06-21 |
| New York | 1788-07-26 |
| South Carolina | 1788-05-23 |
| Virginia | 1788-06-25 |
+-----+-----+
```

这里同样地，引用值（New York）在输出中出现。如果你需要防止这种情况，可以在 ON 子句中增加一个条件表达式，以显式地排除引用：

```
mysql> SELECT s2.name, s2.statehood
-> FROM states AS s1 INNER JOIN states AS s2
-> ON YEAR(s1.statehood) = YEAR(s2.statehood) AND s1.name != s2.name
-> WHERE s1.name = 'New York'
-> ORDER BY s2.name;
+-----+-----+
| name | statehood |
+-----+-----+
| Connecticut | 1788-01-09 |
| Georgia | 1788-01-02 |
| Maryland | 1788-04-28 |
| Massachusetts | 1788-02-06 |
| New Hampshire | 1788-06-21 |
| South Carolina | 1788-05-23 |
| Virginia | 1788-06-25 |
+-----+-----+
```

与找到创作“吃土豆的人”的画家的其他作品之问题类似，statehood 问题同样可以借助用户定义的变量，并使用两个语句解决。当你在表中寻找单个精确的行匹配时，这种方式总是成立的。但是另外一些问题需要找到多行的匹配，此时 2-语句方法就无法奏效了。假设你想要找到每一对在同一年加入联邦的州，该输出可能会包含 states 表中任意州的两两组合。这里没有固定的引用值，因此你不能通过变量来保存引用。对这个问题来说，自连接是完美的解决方案：

```
mysql> SELECT YEAR(s1.statehood) AS year,
-> s1.name AS name1, s1.statehood AS statehood1,
-> s2.name AS name2, s2.statehood AS statehood2
-> FROM states AS s1 INNER JOIN states AS s2
-> ON YEAR(s1.statehood) = YEAR(s2.statehood) AND s1.name != s2.name
-> ORDER BY year, s1.name, s2.name;
+-----+-----+-----+-----+
| year | name1 | statehood1 | name2 | statehood2 |
+-----+-----+-----+-----+
| 1787 | Delaware | 1787-12-07 | New Jersey | 1787-12-18 |
| 1787 | Delaware | 1787-12-07 | Pennsylvania | 1787-12-12 |
| 1787 | New Jersey | 1787-12-18 | Delaware | 1787-12-07 |
| 1787 | New Jersey | 1787-12-18 | Pennsylvania | 1787-12-12 |
+-----+-----+-----+-----+
```

1787	Pennsylvania	1787-12-12	Delaware	1787-12-07
1787	Pennsylvania	1787-12-12	New Jersey	1787-12-18
...				
1912	Arizona	1912-02-14	New Mexico	1912-01-06
1912	New Mexico	1912-01-06	Arizona	1912-02-14
1959	Alaska	1959-01-03	Hawaii	1959-08-21
1959	Hawaii	1959-08-21	Alaska	1959-01-03

ON 子句中的条件要求两个州的名字不能相同，如此消除多余的重复行，这些重复行由于每个州与它自己也是同一年加入联邦而造成的。但是你会发现剩下的每个州的两两组合都出现了两次。举个例子，其中有一行列出了 Delaware 和 New Jersey，但是另外一行列出了 New Jersey 和 Delaware。这是自连接常常导致的情况：即产生包含同样值的行对，只不过值出现的顺序不同。关于从查询结果集中消除这些“疑似重复项”的技术，参见第 14.5 节。

某些自连接问题属于“哪些值与表中其他行不匹配？”的类型。一个例子是“哪些州与其他任何州都不是在同一年加入联邦的？”问题。找到这些州属于“非匹配”问题，这是典型的涉及 LEFT JOIN 的一类问题。在此情况下，解决方案是对 states 表与其自身使用 LEFT JOIN：

```
mysql> SELECT s1.name, s1.statehood
-> FROM states AS s1 LEFT JOIN states AS s2
-> ON YEAR(s1.statehood) = YEAR(s2.statehood) AND s1.name != s2.name
-> WHERE s2.name IS NULL
-> ORDER BY s1.name;
+-----+-----+
| name      | statehood |
+-----+-----+
| Alabama   | 1819-12-14 |
| Arkansas  | 1836-06-15 |
| California| 1850-09-09 |
| Colorado   | 1876-08-01 |
| Illinois  | 1818-12-03 |
| Indiana   | 1816-12-11 |
| Iowa      | 1846-12-28 |
| Kansas    | 1861-01-29 |
| Kentucky  | 1792-06-01 |
| Louisiana | 1812-04-30 |
| Maine     | 1820-03-15 |
| Michigan  | 1837-01-26 |
| Minnesota | 1858-05-11 |
| Mississippi| 1817-12-10 |
| Missouri  | 1821-08-10 |
| Nebraska  | 1867-03-01 |
| Nevada   | 1864-10-31 |
| North Carolina | 1789-11-21 |
| Ohio     | 1803-03-01 |
| Oklahoma | 1907-11-16 |
| Oregon   | 1859-02-14 |
```

```

| Rhode Island | 1790-05-29 |
| Tennessee   | 1796-06-01 |
| Utah         | 1896-01-04 |
| Vermont      | 1791-03-04 |
| West Virginia | 1863-06-20 |
| Wisconsin    | 1848-05-29 |
+-----+

```

该语句为 `states` 表的每一行选择与其 `statehood` 列中年相同的州，并排除了该州自身。对于那些没有这项匹配的行，`LEFT JOIN` 仍然强制输出包含一个连接行，其中 `s2` 的所有列被设为 `NULL`，而正是这些行标识了与其他州不是在同一年加入联邦的州（译注 1）。

12.4 产生主从列表和摘要

Producing Master-Detail Lists and Summaries

问题

两个相关表具有主从关系，并且你希望产生一个列表以同时显示每一个主行与对应的从行，或者为每一个主行产生其从行的摘要。

解决方案

这是一对多的关系。对该问题的解决方案会包含连接操作，但是连接的类型与你要回答的问题有关。如果产生一个只包含一些存在从行的主行列表，可以使用基于主表的主关键字的内部连接。如果产生的列表包含所有主行，包括那些没有从行的（主行），可以使用外部连接。

讨论

从两个相关表中产生一个列表经常是有用的。对于那些具有主从关系或父子关系的表，其中一个表的给定行或许和另一表中的几行相匹配。本节提出了一些你可以提问（并回答）的此类问题，与前一节一样，同样使用了 `artist` 和 `painting` 表。

对于这些表的一种主从问题的形式是，“每幅画作的画者是谁？”这是一个简单的内部连接，即根据画家的 ID 值将每个 `painting` 行与对应的 `artist` 行进行匹配：

```

mysql> SELECT artist.name, painting.title
-> FROM artist INNER JOIN painting ON artist.a_id = painting.a_id
-> ORDER BY name, title;
+-----+-----+
| name   | title        |
+-----+-----+
| Da Vinci | The Last Supper |
| Da Vinci | The Mona Lisa |

```

译注 1：最后简单地使用 `Where` 子句可将这些行选择出来。

```

| Renoir | Les Deux Soeurs |
| Van Gogh | Starry Night |
| Van Gogh | The Potato Eaters |
| Van Gogh | The Rocks |
+-----+

```

在你希望只列出具有从行的主行时，内部连接可以满足要求。然而，另一种你可以提出的主从问题的形式是，“每个画家画了哪些作品？”这个问题与前一个相似，但不完全一样。如果在 artist 表中存在 painting 表中没有出现过的画家，那么此问题将有不同的答案，并且需要一个不同的语句以产生正确的回答。在此情况下，连接的输出应当包括一个表中与另一个表无匹配的那些行，这是需要使用外部连接（第 12.2 节）的“查找非匹配行”问题的一种形式。因此，为了显示每个 artist 行，无论其是否具有对应的 painting 行，应使用 LEFT JOIN：

```

mysql> SELECT artist.name, painting.title
-> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
-> ORDER BY name, title;
+-----+-----+
| name | title |
+-----+-----+
| Da Vinci | The Last Supper |
| Da Vinci | The Mona Lisa |
| Monet | NULL |
| Picasso | NULL |
| Renoir | Les Deux Soeurs |
| Van Gogh | Starry Night |
| Van Gogh | The Potato Eaters |
| Van Gogh | The Rocks |
+-----+-----+

```

结果中 title 列为 NULL 的行对应了存在于 artist 表中但是你没有收藏其作品的那些画家。

同样的准则也适用于为主从表产生摘要的情况。举例来说，为了根据每个画家的作品数来概述你的艺术收藏，你或许会提出，“在 painting 表中每个画家的作品有多少？”的问题。为了根据画家的 ID 找到答案，你可以很容易地使用下面语句对画作进行计数：

```

mysql> SELECT a_id, COUNT(a_id) AS count FROM painting GROUP BY a_id;
+-----+-----+
| a_id | count |
+-----+-----+
| 1 | 2 |
| 3 | 3 |
| 5 | 1 |
+-----+-----+

```

当然，上面的输出实际上没有任何意义，除非你能够记住所有画家的 ID 号。为了显示画家们的姓名而不是 ID，需要连接 painting 表与 artist 表：

```

mysql> SELECT artist.name AS painter, COUNT(painting.a_id) AS count
-> FROM artist INNER JOIN painting ON artist.a_id = painting.a_id
-> GROUP BY artist.name;
+-----+-----+
| painter | count |
+-----+-----+
| Da Vinci |    2 |
| Renoir   |    1 |
| Van Gogh |    3 |
+-----+-----+

```

另一方面，你可能会提问，“每个画家画了多少幅作品？”只要每个 `artist` 表中的画家都至少具有一个相应的 `painting` 表的行，那么此问题与前一个是完全等同的（并且可以使用同样的语句来回答）。但是如果在 `artist` 表中含有你没有收藏其作品的画家，他们将不会出现在该语句的输出中。为了产生每个画家的计数摘要，甚至包括那些在 `painting` 表中没有画作的画家，应该使用 `LEFT JOIN`：

```

mysql> SELECT artist.name AS painter, COUNT(painting.a_id) AS count
-> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
-> GROUP BY artist.name;
+-----+-----+
| painter | count |
+-----+-----+
| Da Vinci |    2 |
| Monet    |    0 |
| Picasso  |    0 |
| Renoir   |    1 |
| Van Gogh |    3 |
+-----+-----+

```

在编写这种语句时要注意一个容易产生的细小错误。假设你在写 `COUNT()` 函数时有些细微的改变，如下所示：

```

mysql> SELECT artist.name AS painter, COUNT(*) AS count
-> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
-> GROUP BY artist.name;
+-----+-----+
| painter | count |
+-----+-----+
| Da Vinci |    2 |
| Monet    |    1 |
| Picasso  |    1 |
| Renoir   |    1 |
| Van Gogh |    3 |
+-----+-----+

```

现在每个画家看起来都至少含有一幅作品。为什么结果不同了呢？产生此问题的原因在于使用了 `COUNT(*)` 而不是 `COUNT(painting.a_id)`。`LEFT JOIN` 的工作方式是为左表中（与右表）不匹配的行同样产生连接行，而该连接行中所有来自右表的列被设为 `NULL`。在本例中，右表为 `painting` 表，因此使用 `COUNT(painting.a_id)` 的语句是正确的，因为 `COUNT(expr)` 仅对非空值计数。而使用 `COUNT(*)` 的语句错在对所有的连接行计数，甚至

那些包含 NULL 值即对应无画作的画家的行。LEFT JOIN 对其他类型的摘要同样是合适的，为了产生显示 artist 表中每个画家的作品数总计和平均值，可以使用下面的语句：

```
mysql> SELECT artist.name AS painter,
-> COUNT(painting.a_id) AS 'number of paintings',
-> SUM(painting.price) AS 'total price',
-> AVG(painting.price) AS 'average price'
-> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
-> GROUP BY artist.name;
+-----+-----+-----+-----+
| painter | number of paintings | total price | average price |
+-----+-----+-----+-----+
| Da Vinci | 2 | 121 | 60.5000 |
| Monet | 0 | NULL | NULL |
| Picasso | 0 | NULL | NULL |
| Renoir | 1 | 64 | 64.0000 |
| Van Gogh | 3 | 148 | 49.3333 |
+-----+-----+-----+-----+
```

注意对那些你没有收藏其作品的画家，COUNT() 函数的结果是零，而 SUM() 和 AVG() 的结果是 NULL，这是因为后两个函数在应用到一个不包含非空值的值集合时，会返回 NULL。为了在此情况下将 sum 或 average 值显示为零，应当修改语句，即首先使用 IFNULL() 测试 SUM() 或 AVG() 的返回值：

```
mysql> SELECT artist.name AS painter,
-> COUNT(painting.a_id) AS 'number of paintings',
-> IFNULL(SUM(painting.price),0) AS 'total price',
-> IFNULL(AVG(painting.price),0) AS 'average price'
-> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
-> GROUP BY artist.name;
+-----+-----+-----+-----+
| painter | number of paintings | total price | average price |
+-----+-----+-----+-----+
| Da Vinci | 2 | 121 | 60.5000 |
| Monet | 0 | 0 | 0.0000 |
| Picasso | 0 | 0 | 0.0000 |
| Renoir | 1 | 64 | 64.0000 |
| Van Gogh | 3 | 148 | 49.3333 |
+-----+-----+-----+-----+
```

12.5 枚举多对多的关系

Enumerating a Many-to-Many Relationship

问题

你希望显示这样的表间关系，即一个表中的多行可能和另一个表中的多个行相匹配。

解决方案

这是多对多的关系，它需要第 3 个表以关联两个原来的表并使用 3-向连接 (three-way join) 来列出它们之间的关系。

讨论

前面章节中是使用的 `artist` 和 `painting` 表是一对多的关系：一个给定的画家可能创作了许多作品，但是每幅作品只能被唯一一位画家所创作。一对多关系相对比较简单，两个具有此关系的表可以使用它们共有的关键字进行连接。

更简单的是一对一关系，此关系经常被用来执行查找一个值集到另一个值集的映射。举个例子，`states` 表包含了州的名字与简写列，可以列出完整的州名以及它们对应的简写：

```
mysql> SELECT name, abbrev FROM states;
+-----+-----+
| name | abbrev |
+-----+-----+
| Alabama | AL |
| Alaska | AK |
| Arizona | AZ |
| Arkansas | AR |
...

```

这种一一对应的关系可以用来映射 `painting` 表中州名称的缩写，其中 `painting` 表中包含一列数据以指明每一幅作品是在哪一个州购买的。如果没有映射，那么 `painting` 表的条目显示出来是这样的：

```
mysql> SELECT title, state FROM painting ORDER BY state;
+-----+-----+
| title | state |
+-----+-----+
| The Rocks | IA |
| The Last Supper | IN |
| Starry Night | KY |
| The Potato Eaters | KY |
| The Mona Lisa | MI |
| Les Deux Soeurs | NE |
+-----+-----+

```

如果你想知道完整的州名而不是简写，可以利用 `states` 表中所列举的它们间的一对一关系。下面的方法将该表与 `painting` 表相连接，通过使用两表共有的州名简写值：

```
mysql> SELECT painting.title, states.name AS state
    -> FROM painting INNER JOIN states ON painting.state = states.abbrev
    -> ORDER BY state;
+-----+-----+
| title | state |
+-----+-----+
| The Last Supper | Indiana |

```

The Rocks	Iowa
Starry Night	Kentucky
The Potato Eaters	Kentucky
The Mona Lisa	Michigan
Les Deux Soeurs	Nebraska

表间更复杂的关系是多对多关系，此关系发生在当一个表中的行与另一个表具有多个匹配，并且反过来也一样时。数据库书籍常常会使用“零件与供应商”问题来说明这种关系。

(一个给定的零件可能有好几家供应商供货，那么你如何才能产生显示哪些零件是由哪些供应商所供应的列表呢？)但是，由于我们已经太多次看到这个例子，我宁愿使用一个不同的例子。因此让我们考虑下面的场景，虽然从概念上来说它实际上是同一回事：你和你的一帮朋友是狂热的尤克牌 (euchre) 爱好者，并且举行 4 人 1 组的纸牌比赛，其中每两人为一对。每年，你们都聚集在一起，结成对家，并开始友好的比赛。自然地，不同的玩家对每次比赛结果的记忆不同，为了避免口舌之争，你必须在数据库中记录结对情况与结果。记录比赛结果的一种方式是建立如下的一个表，对每一次比赛的年份内，你需要记录下团队的名字、输赢记录、玩家以及玩家居住的城市：

```
mysql> SELECT * FROM euchre ORDER BY year, wins DESC, player;
+-----+-----+-----+-----+-----+-----+
| team | year | wins | losses | player | player_city |
+-----+-----+-----+-----+-----+-----+
| Kings | 2005 | 10 | 2 | Ben | Cork |
| Kings | 2005 | 10 | 2 | Billy | York |
| Crowns | 2005 | 7 | 5 | Melvin | Dublin |
| Crowns | 2005 | 7 | 5 | Tony | Derry |
| Stars | 2005 | 4 | 8 | Franklin | Bath |
| Stars | 2005 | 4 | 8 | Wallace | Cardiff |
| Sceptres | 2005 | 3 | 9 | Maurice | Leeds |
| Sceptres | 2005 | 3 | 9 | Nigel | London |
| Crowns | 2006 | 9 | 3 | Ben | Cork |
| Crowns | 2006 | 9 | 3 | Tony | Derry |
| Kings | 2006 | 8 | 4 | Franklin | Bath |
| Kings | 2006 | 8 | 4 | Nigel | London |
| Stars | 2006 | 5 | 7 | Maurice | Leeds |
| Stars | 2006 | 5 | 7 | Melvin | Dublin |
| Sceptres | 2006 | 2 | 10 | Billy | York |
| Sceptres | 2006 | 2 | 10 | Wallace | Cardiff |
+-----+-----+-----+-----+-----+-----+
```

如上表所显示的那样，每个队有多个玩家，且每个玩家可以参加多个队。该表获取了这种自然的多对多关系，但是是以一个非正规的形式，因为每行不必要的保存了不少重复信息。(每支队伍的信息同时作为每个玩家的信息被记录了多次。)一个更好地表现此多对多关系的方法是使用多重表：

- 只在 `euchre_team` 表中保存一次每个团队的名称、年度和记录。
- 只在 `euchre_player` 表中保存一次每个玩家的姓名、居住地。
- 建立第 3 个表 `euchre_link`, 该表作为两个基本表的连接或者说桥梁, 保存团队-玩家间的联系。为了将此表保存的信息最小化, 为每个团队和玩家在它们各自的表中赋予一个唯一的 ID, 并且在 `euchre_link` 表中只保存这些 ID。

因此团队和玩家表如下所示:

```
mysql> SELECT * FROM euchre_team;
+----+-----+-----+-----+
| id | name | year | wins | losses |
+----+-----+-----+-----+
| 1 | Kings | 2005 | 10 | 2 |
| 2 | Crowns | 2005 | 7 | 5 |
| 3 | Stars | 2005 | 4 | 8 |
| 4 | Sceptres | 2005 | 3 | 9 |
| 5 | Kings | 2006 | 8 | 4 |
| 6 | Crowns | 2006 | 9 | 3 |
| 7 | Stars | 2006 | 5 | 7 |
| 8 | Sceptres | 2006 | 2 | 10 |
+----+-----+-----+-----+
mysql> SELECT * FROM euchre_player;
+----+-----+-----+
| id | name | city |
+----+-----+-----+
| 1 | Ben | Cork |
| 2 | Billy | York |
| 3 | Tony | Derry |
| 4 | Melvin | Dublin |
| 5 | Franklin | Bath |
| 6 | Wallace | Cardiff |
| 7 | Nigel | London |
| 8 | Maurice | Leeds |
+----+-----+-----+
```

`euchre_link` 表以如下方式关联团队和玩家信息:

```
mysql> SELECT * FROM euchre_link;
+-----+-----+
| team_id | player_id |
+-----+-----+
| 1 | 1 |
| 1 | 2 |
| 2 | 3 |
| 2 | 4 |
| 3 | 5 |
| 3 | 6 |
| 4 | 7 |
| 4 | 8 |
| 5 | 5 |
| 5 | 7 |
| 6 | 1 |
```

	6	3
	7	4
	7	8
	8	2
	8	6

为了使用这些表来回答关于团队与玩家的问题，你需要执行一个 3-向连接，即利用连接表来使两个基本表相互关联。下面为一些例子：

- 列出所有的配对以显示每个团队及参加此队的玩家。此语句列举了所有 euchre_team 和 euchre_player 表的对应关系，并且重现了原来位于非正规的 euchre 表中的信息：

```
mysql> SELECT t.name, t.year, t.wins, t.losses, p.name, p.city
      -> FROM euchre_team AS t INNER JOIN euchre_link AS l
      -> INNER JOIN euchre_player AS p
      -> ON t.id = l.team_id AND p.id = l.player_id
      -> ORDER BY t.year, t.wins DESC, p.name;
+-----+-----+-----+-----+-----+-----+
| name | year | wins | losses | name   | city   |
+-----+-----+-----+-----+-----+-----+
| Kings | 2005 | 10 | 2 | Ben    | Cork   |
| Kings | 2005 | 10 | 2 | Billy  | York   |
| Crowns | 2005 | 7 | 5 | Melvin | Dublin |
| Crowns | 2005 | 7 | 5 | Tony   | Derry  |
| Stars | 2005 | 4 | 8 | Franklin | Bath   |
| Stars | 2005 | 4 | 8 | Wallace  | Cardiff|
| Sceptres | 2005 | 3 | 9 | Maurice | Leeds  |
| Sceptres | 2005 | 3 | 9 | Nigel   | London |
| Crowns | 2006 | 9 | 3 | Ben    | Cork   |
| Crowns | 2006 | 9 | 3 | Tony   | Derry  |
| Kings | 2006 | 8 | 4 | Franklin | Bath   |
| Kings | 2006 | 8 | 4 | Nigel  | London |
| Stars | 2006 | 5 | 7 | Maurice | Leeds  |
| Stars | 2006 | 5 | 7 | Melvin | Dublin |
| Sceptres | 2006 | 2 | 10 | Billy  | York   |
| Sceptres | 2006 | 2 | 10 | Wallace | Cardiff|
+-----+-----+-----+-----+-----+-----+
```

- 列出特定团队的所有成员（2005 年的 Crowns 队）：

```
mysql> SELECT p.name, p.city
      -> FROM euchre_team AS t INNER JOIN euchre_link AS l
      -> INNER JOIN euchre_player AS p
      -> ON t.id = l.team_id AND p.id = l.player_id
      -> AND t.name = 'Crowns' AND t.year = 2005;
+-----+-----+
| name | city  |
+-----+-----+
| Tony | Derry |
| Melvin | Dublin |
+-----+-----+
```

- 列出给定玩家 (Billy) 曾经参与的所有团队：

```
mysql> SELECT t.name, t.year, t.wins, t.losses
-> FROM euchre_team AS t INNER JOIN euchre_link AS l
-> INNER JOIN euchre_player AS p
-> ON t.id = l.team_id AND p.id = l.player_id
-> WHERE p.name = 'Billy';
+-----+-----+-----+-----+
| name | year | wins | losses |
+-----+-----+-----+-----+
| Kings | 2005 | 10 | 2 |
| Sceptres | 2006 | 2 | 10 |
+-----+-----+-----+
```

12.6 查找每组行中含有最大或最小值的行

Finding Rows Containing Per-Group Minimum or Maximum Values

问题

你希望找到表中每组行中的哪一行包含了给定列的最大或最小值。举个例子，你想知道在你的收藏中每个画家最贵的作品。

解决方案

建立一个临时表以保存每组的最大或最小值，并将临时表和原始表进行连接以得到每组中匹配的行。如果你喜欢用单个查询语句来解决，可以在 FROM 子句中使用子查询，而不是临时表。

讨论

很多问题涉及在特定的表列中查找最大或最小值，并且你希望知道包含此值的行中其他列的值也是非常普遍的。举例来说，当你正使用 artist 和 painting 表的时候，有可能被问到像这样的问题：“哪幅画作在你的收藏中是最贵的，并且其作者是谁？”一种方法是将最高的价格保存在用户定义变量中，再使用该变量确定含有此价格的行，以便能够从中获取其他列的值：

```
mysql> SET @max_price = (SELECT MAX(price) FROM painting);
mysql> SELECT artist.name, painting.title, painting.price
-> FROM artist INNER JOIN painting
-> ON painting.a_id = artist.a_id
-> WHERE painting.price = @max_price;
+-----+-----+-----+
| name | title | price |
+-----+-----+-----+
```

```
| Da Vinci | The Mona Lisa | 87 |  
+-----+-----+-----+
```

同样的问题也可以通过创建临时表来保存最高价格，然后将之与另一个表相连接来解决：

```
mysql> CREATE TABLE tmp SELECT MAX(price) AS max_price FROM painting;  
mysql> SELECT artist.name, painting.title, painting.price  
-> FROM artist INNER JOIN painting INNER JOIN tmp  
-> ON painting.a_id = artist.a_id  
-> AND painting.price = tmp.max_price;  
+-----+-----+-----+  
| name      | title        | price |  
+-----+-----+-----+  
| Da Vinci | The Mona Lisa | 87 |  
+-----+-----+-----+
```

上面展示的使用用户定义变量或临时表的技术在第 8.5 节中有所阐述。此处它们的用法与其类似，只不过现在我们将它们用在了多重表上。

从表面上看，使用临时表和连接解决此问题比使用用户定义变量要更复杂一点，但这项技术有实际价值吗？是的，当然有，因为它引申了一项更通用的技术，以解决更复杂的问题。前面的语句只显示了在整个表中最贵的单幅画作的信息，如果你的问题是，“每个画家的最贵作品是什么？”，那么你将无法使用用户定义变量去回答这个，因为此解答需要为每个画家找到最高价格，而一个变量一次只能保存一个值。然而使用临时表的技术仍然可以工作得很好，因为表可以包含多个行，并且通过连接可以找到它们的所有匹配。

为了回答此问题，选择每个画家的 ID 和相应的最高作品价格并存入一个临时表。该表保存的不是所有作品最高价格而是每组的最大值，“组”在此被定义为“给定画家的作品”。然后使用保存在临时表中的画家 ID 和价格与 painting 表中的行相匹配，并将结果与 artist 表连接以得到画家的姓名：

```
mysql> CREATE TABLE tmp  
-> SELECT a_id, MAX(price) AS max_price FROM painting GROUP BY a_id;  
mysql> SELECT artist.name, painting.title, painting.price  
-> FROM artist INNER JOIN painting INNER JOIN tmp  
-> ON painting.a_id = artist.a_id  
-> AND painting.a_id = tmp.a_id  
-> AND painting.price = tmp.max_price;  
+-----+-----+-----+  
| name      | title        | price |  
+-----+-----+-----+  
| Da Vinci | The Mona Lisa | 87 |  
| Van Gogh | The Potato Eaters | 67 |  
| Renoir    | Les Deux Soeurs | 64 |  
+-----+-----+-----+
```

为了在单个语句中获得同样的结果，可以在 FROM 子句中使用子查询在临时表中获取相同的行：

```
mysql> SELECT artist.name, painting.title, painting.price
-> FROM artist INNER JOIN painting INNER JOIN
-> (SELECT a_id, MAX(price) AS max_price FROM painting GROUP BY a_id)
-> AS tmp
-> ON painting.a_id = artist.a_id
-> AND painting.a_id = tmp.a_id
-> AND painting.price = tmp.max_price;
+-----+-----+-----+
| name | title | price |
+-----+-----+-----+
| Da Vinci | The Mona Lisa | 87 |
| Van Gogh | The Potato Eaters | 67 |
| Renoir | Les Deux Soeurs | 64 |
+-----+-----+-----+
```

另一种回答每组最大值问题的方法是使用 LEFT JOIN 以将表与其自身相连接。下面的语句确定了每个画家 ID 的最高价格的作品（我们使用 IS NULL 来选择属于 p1 的并且在 p2 中没有比其价格更高的那些行）：

```
mysql> SELECT p1.a_id, p1.title, p1.price
-> FROM painting AS p1 LEFT JOIN painting AS p2
-> ON p1.a_id = p2.a_id AND p1.price < p2.price
-> WHERE p2.a_id IS NULL;
+-----+-----+-----+
| a_id | title | price |
+-----+-----+-----+
| 1 | The Mona Lisa | 87 |
| 3 | The Potato Eaters | 67 |
| 5 | Les Deux Soeurs | 64 |
+-----+-----+-----+
```

为了显示画家的姓名而不是 ID 值，将 LEFT JOIN 的结果与 artist 表相连接：

```
mysql> SELECT artist.name, p1.title, p1.price
-> FROM painting AS p1 LEFT JOIN painting AS p2
-> ON p1.a_id = p2.a_id AND p1.price < p2.price
-> INNER JOIN artist ON p1.a_id = artist.a_id
-> WHERE p2.a_id IS NULL;
+-----+-----+-----+
| name | title | price |
+-----+-----+-----+
| Da Vinci | The Mona Lisa | 87 |
| Van Gogh | The Potato Eaters | 67 |
| Renoir | Les Deux Soeurs | 64 |
+-----+-----+-----+
```

与自身左连接的方法或许比使用临时表或自查询稍显得更不直观一些。

上面的技术同样可用于其他类型的值，比如临时值。假设 `driver_log` 表列出了司机及其行驶的里程：

```
mysql> SELECT name, trav_date, miles
   -> FROM driver_log
   -> ORDER BY name, trav_date;
+-----+-----+-----+
| name | trav_date | miles |
+-----+-----+-----+
| Ben  | 2006-08-29 | 131  |
| Ben  | 2006-08-30 | 152  |
| Ben  | 2006-09-02 | 79   |
| Henry | 2006-08-26 | 115  |
| Henry | 2006-08-27 | 96   |
| Henry | 2006-08-29 | 300  |
| Henry | 2006-08-30 | 203  |
| Henry | 2006-09-01 | 197  |
| Suzi  | 2006-08-29 | 391  |
| Suzi  | 2006-09-02 | 502  |
+-----+-----+-----+
```

对于此表，一种分组最大值的问题是“显示每个司机最近的行驶里程”。它可以使用如下的临时表来解决：

```
mysql> CREATE TABLE tmp
   -> SELECT name, MAX(trav_date) AS trav_date
   -> FROM driver_log GROUP BY name;
mysql> SELECT driver_log.name, driver_log.trav_date, driver_log.miles
   -> FROM driver_log INNER JOIN tmp
   -> ON driver_log.name = tmp.name AND driver_log.trav_date = tmp.trav_date
   -> ORDER BY driver_log.name;
+-----+-----+-----+
| name | trav_date | miles |
+-----+-----+-----+
| Ben  | 2006-09-02 | 79   |
| Henry | 2006-09-01 | 197  |
| Suzi  | 2006-09-02 | 502  |
+-----+-----+-----+
```

同样，你可以在 `FROM` 子句中使用子查询来解决，如下所示：

```
mysql> SELECT driver_log.name, driver_log.trav_date, driver_log.miles
   -> FROM driver_log INNER JOIN
   -> (SELECT name, MAX(trav_date) AS trav_date
   -> FROM driver_log GROUP BY name) AS tmp
   -> ON driver_log.name = tmp.name AND driver_log.trav_date = tmp.trav_date
   -> ORDER BY driver_log.name;
+-----+-----+-----+
| name | trav_date | miles |
+-----+-----+-----+
| Ben  | 2006-09-02 | 79   |
| Henry | 2006-09-01 | 197  |
| Suzi  | 2006-09-02 | 502  |
+-----+-----+-----+
```

哪种技术更好：临时表还是在 FROM 子句中的子查询？对于较小的表，两种方式或许没有多大的区别。但如果临时表或子查询的结果很大，使用临时表一般具有一个优点，即你可以在创建它之后，使用它进行连接之前，先对其做索引。

参考

本节展示了如何解答分组最大值的，即通过选择摘要信息存入临时表并将之与原始表连接，或者在 FROM 子句中使用子查询。这些技术在很多情况下有用，比如计算团队等级，每个团队的等级是根据组中每个团队与最高纪录的团队相比较的结果而决定的。第 12.7 节讨论了具体的做法。

12.7 计算队伍排名

Computing Team Standings

问题

你需要根据球队的输赢记录来计算球队的排名，包括比赛落后（games-behind, GB）值。

解决方案

先确定哪支球队处于第一位，然后将结果与原来的行相连接。

讨论

体育团队相互竞争的排名一般根据如下方法：先找到胜负差最大的队（作为第一名），而所有不是第一名的队都被赋予一个“比赛落后”值，该值指示了它们距离第一名的差距，本节展现了如何计算这些值的方法。第一个例子使用包含了队伍记录的单个集合的表，以说明计算的逻辑性。第二个例子使用了包含记录的多个集合的表（即所有在两个联盟分区的队伍记录，包括赛季的两个阶段）。在此情况下，使用连接来为每组队伍独立地执行计算是必需的。

考虑到下面的表 `standings1`，该表包含了棒球比赛记录（代表了 1902 年北部联赛的最终排名）的单个集合：

```
mysql> SELECT team, wins, losses FROM standings1
-> ORDER BY wins-losses DESC;
+-----+-----+-----+
| team      | wins | losses |
+-----+-----+-----+
| Winnipeg   | 37  |    20  |
| Crookston  | 31  |    25  |
| Fargo      | 30  |    26  |
```

Grand Forks	28		26	
Devils Lake	19		31	
Cavalier	15		32	

输出行根据 win-loss 的差值来排序，这就是将队伍从第一位按顺序排到最后一位的方法。但队伍排名的显示一般包括队伍的胜率和一个指示了所有非第一名的队伍比领先者落后场次的数字，因此我们需要将这些信息添加到输出中去。计算胜率是很简单的，胜率即获胜场次与已赛总场次的比率，它可以使用此表达式来确定：

```
wins / (wins + losses)
```

当某个队伍还没有比赛时，该表达式可能会包含除零操作。出于简单性考虑，我将假设比赛数不为零。但是如果你想要处理这种情况，可以将表达式归纳为：

```
IF(wins=0,0,wins/(wins+losses))
```

此表达式利用了下面的事实，即如果队伍还没有赢过一场比赛时，根本不需要除法运算（直接将结果赋零）。

确定比赛-落后值有一点麻烦，它基于两支队伍的胜负差记录的关系，即计算下面两个值的平均数：

- 第二名队伍如果达到与第一名同样多的胜场数还需要赢得的比赛数。
- 第一名队伍如果与第二名具有同样多的负场数还需要输掉的比赛数。

举例来说，假设两支队伍 A 和 B 具有下面的胜负记录：

team	wins	losses
A	17	11
B	14	12

这里，队伍 B 需要再赢 3 场比赛，而队伍 A 需要输掉 1 场比赛，二者战绩才能相同。3 和 1 的平均值是 2，因此 B 落后 A 两场比赛。从数学上，两个队的比赛落后值的计算可以如下表示：

$$((winsA - winsB) + (lossesB - lossesA)) / 2$$

通过对表达式项的稍许整理，该式变为：

$$((winsA - lossesA) - (winsB - lossesB)) / 2$$

第二个表达式与第一个是等价的，但是它的每个因子为单支队伍胜负差，而不是两支队伍的比较。这使其更为有用，因为每个因子可以独立地根据单支队伍的记录确定下来。第一

一个因子代表了处于第一名队伍的胜负差，因此我们可以先计算此值，而所有其他队伍的 GB 值的确定都涉及到它。

第一名队伍具有最大的胜负差值。使用下面语句查找该值并保存到变量中：

```
mysql> SET @wl_diff = (SELECT MAX(wins-losses) FROM standings1);
```

接着如下所示，此差值产生包括胜率与 GB 值的队伍排名：

```
mysql> SELECT team, wins AS W, losses AS L,
-> wins/(wins+losses) AS PCT,
-> (@wl_diff - (wins-losses)) / 2 AS GB
-> FROM standings1
-> ORDER BY wins-losses DESC, PCT DESC;
```

team	W	L	PCT	GB
Winnipeg	37	20	0.6491	0.0000
Crookston	31	25	0.5536	5.5000
Fargo	30	26	0.5357	6.5000
Grand Forks	28	26	0.5185	7.5000
Devils Lake	19	31	0.3800	14.5000
Cavalier	15	32	0.3191	17.0000

此处应当关注一下两个格式方面的小问题。通常，排名列表将胜率值保留为小数点后三位，而 GB 值保留到小数点后一位（第一名的队伍例外，它的 GB 显示为-）。可以使用 TRUNCATE(expr,n) 来显示 n 位小数，而为了适当地显示第一名队伍的 GB 值，在计算 GB 列的表达式中嵌入 IF ()，将 0 映射为分隔号：

```
mysql> SELECT team, wins AS W, losses AS L,
-> TRUNCATE(wins/(wins+losses),3) AS PCT,
-> IF(@wl_diff = wins-losses,
->     '--',TRUNCATE(@wl_diff - (wins-losses))/2,1)) AS GB
-> FROM standings1
-> ORDER BY wins-losses DESC, PCT DESC;
```

team	W	L	PCT	GB
Winnipeg	37	20	0.649	-
Crookston	31	25	0.553	5.5
Fargo	30	26	0.535	6.5
Grand Forks	28	26	0.518	7.5
Devils Lake	19	31	0.380	14.5
Cavalier	15	32	0.319	17.0

这些语句将队伍以 win-loss 的差值进行排序，在差值相同的情况下使用胜率进一步细分。

当然，如果只使用胜率来排序更为简单，但是你不一定总能得到正确的顺序。一个令人惊奇的事实是胜率低的队伍的排名可以比胜率高的队伍高。（这一般发生在赛季早期，相对来说，此时队伍的比赛数具有比较大的差距。）两支队伍 A 和 B 的情况如下所示：

team	wins	losses
A	4	1
B	2	0

根据队伍的比赛记录计算 GB 和胜率值得到下面的结果，即排第一的队伍具有比排第二的队伍较低的胜率。

team	W	L	PCT	GB
A	4	1	0.800	-
B	2	0	1.000	0.5

排名计算显示了不使用连接所能做到的程度，它们只包含一个队伍记录的简单集合，因此第一名队伍的胜负差可以存放在变量里。而当一个数据集包含几个记录集合时，会出现更复杂的情况。举个例子，1997 年北部联赛具有两个分区（东部和西部）。此外，由于每半个赛季的胜利者相互比赛，以角逐竞争联赛冠军的资格，因此一个赛季的前半部分和后半部分支持独立的排名。下面的表 `standings2` 显示了这些行，根据半赛季、分区、胜负差值排序：

```
mysql> SELECT half, division, team, wins, losses FROM standings2
    -> ORDER BY half, division, wins-losses DESC;
```

half	division	team	wins	losses
1	Eastern	St. Paul	24	18
1	Eastern	Thunder Bay	18	24
1	Eastern	Duluth-Superior	17	24
1	Eastern	Madison	15	27
1	Western	Winnipeg	29	12
1	Western	Sioux City	28	14
1	Western	Fargo-Moorhead	21	21
1	Western	Sioux Falls	15	27
2	Eastern	Duluth-Superior	22	20
2	Eastern	St. Paul	21	21
2	Eastern	Madison	19	23
2	Eastern	Thunder Bay	18	24
2	Western	Fargo-Moorhead	26	16

	2	Western	Winnipeg	24	18	
	2	Western	Sioux City	22	20	
	2	Western	Sioux Falls	16	26	

为了产生这些行的排名，需要为赛程和分区的组合计算 GB 值，在半赛季与分区的四种组合中。先计算与每组中第一名队伍的胜负差，并将该值保存到隔离的 firstplace 表：

```
mysql> CREATE TABLE firstplace
-> SELECT half, division, MAX(wins-losses) AS wl_diff
-> FROM standings2
-> GROUP BY half, division;
```

然后将 firstplace 表与原始的排名相连接，同时对每支队伍的记录使用相应的胜负差计算其 GB 值：

```
mysql> SELECT wl.half, wl.division, wl.team, wl.wins AS W, wl.losses AS L,
-> TRUNCATE(wl.wins/(wl.wins+wl.losses),3) AS PCT,
-> IF(fp.wl_diff = wl.wins-wl.losses,
->      '-',TRUNCATE((fp.wl_diff - (wl.wins-wl.losses)) / 2,1)) AS GB
-> FROM standings2 AS wl INNER JOIN firstplace AS fp
-> ON wl.half = fp.half AND wl.division = fp.division
-> ORDER BY wl.half, wl.division, wl.wins-wl.losses DESC, PCT DESC;
+-----+-----+-----+-----+-----+-----+
| half | division | team           | W   | L    | PCT   | GB    |
+-----+-----+-----+-----+-----+-----+
| 1    | Eastern   | St. Paul        | 24  | 18   | 0.571 | -    |
| 1    | Eastern   | Thunder Bay     | 18  | 24   | 0.428 | 6.0  |
| 1    | Eastern   | Duluth-Superior | 17  | 24   | 0.414 | 6.5  |
| 1    | Eastern   | Madison         | 15  | 27   | 0.357 | 9.0  |
| 1    | Western   | Winnipeg         | 29  | 12   | 0.707 | -    |
| 1    | Western   | Sioux City       | 28  | 14   | 0.666 | 1.5  |
| 1    | Western   | Fargo-Moorhead  | 21  | 21   | 0.500 | 8.5  |
| 1    | Western   | Sioux Falls      | 15  | 27   | 0.357 | 14.5 |
| 2    | Eastern   | Duluth-Superior | 22  | 20   | 0.523 | -    |
| 2    | Eastern   | St. Paul         | 21  | 21   | 0.500 | 1.0  |
| 2    | Eastern   | Madison          | 19  | 23   | 0.452 | 3.0  |
| 2    | Eastern   | Thunder Bay      | 18  | 24   | 0.428 | 4.0  |
| 2    | Western   | Fargo-Moorhead  | 26  | 16   | 0.619 | -    |
| 2    | Western   | Winnipeg          | 24  | 18   | 0.571 | 2.0  |
| 2    | Western   | Sioux City        | 22  | 20   | 0.523 | 4.0  |
| 2    | Western   | Sioux Falls       | 16  | 26   | 0.380 | 10.0 |
+-----+-----+-----+-----+-----+-----+
```

输出稍微显得难以阅读，不过，为了让其更易理解，你或许可以在程序中执行该语句并重新格式化所得到的结果，以分离地显示每个队伍记录集合。这里是一些对应操作的 Perl 代码，在每次遇到一个新的排名组时，开始一组新的输出。代码假设连接语句已被执行，并且其结果可以使用语句句柄 \$sth 得到：

```
my ($cur_half, $cur_div) = ("", "");
while (my ($half, $div, $team, $wins, $losses, $pct, $gb)
```

```

    = $sth->fetchrow_array ())
{
    if ($cur_half ne $half || $cur_div ne $div) # new group of standings?
    {
        # print standings header and remember new half/division values
        print "\n$div Division, season half $half\n";
        printf "%-20s %3s %3s %5s %s\n", "Team", "W", "L", "PCT", "GB";
        $cur_half = $half;
        $cur_div = $div;
    }
    printf "%-20s %3d %3d %5s %s\n", $team, $wins, $losses, $pct, $gb;
}

```

重新格式化的输出如下所示：

Eastern Division, season half 1				
Team	W	L	PCT	GB
St. Paul	24	18	0.571	-
Thunder Bay	18	24	0.428	6.0
Duluth-Superior	17	24	0.414	6.5
Madison	15	27	0.357	9.0

Western Division, season half 1				
Team	W	L	PCT	GB
Winnipeg	29	12	0.707	-
Sioux City	28	14	0.666	1.5
Fargo-Moorhead	21	21	0.500	8.5
Sioux Falls	15	27	0.357	14.5

Eastern Division, season half 2				
Team	W	L	PCT	GB
Duluth-Superior	22	20	0.523	-
St. Paul	21	21	0.500	1.0
Madison	19	23	0.452	3.0
Thunder Bay	18	24	0.428	4.0

Western Division, season half 2				
Team	W	L	PCT	GB
Fargo-Moorhead	26	16	0.619	-
Winnipeg	24	18	0.571	2.0
Sioux City	22	20	0.523	4.0
Sioux Falls	16	26	0.380	10.0

此代码来源于本节子目录的 calc_standings.pl 脚本文件。该目录同时包含了 PHP 脚本——calc_standings.php，此脚本创建了 HTML 表格式的输出，因为你可能更需要在 Web 环境下产生排名。

12.8 使用连接补全或识别列表的缺口

Using a Join to Fill or Identify Holes in a List

问题

你需要为每个类别产生摘要，但是其中的一些并没有要被摘要的数据。因此，摘要就缺失了一些类别。

解决方案

创建一个引用表列出每个种类，并且根据在此列表与包含数据的表之间的 LEFT JOIN 产生摘要，从而引用表中的每个种类都会出现在结果中，甚至包括那些数据中没有的类别也会在摘要中列出。

讨论

当你运行一个摘要查询，一般来说它只为数据中实际存在的值产生条目。假设你需要为 mail 表中的行产生每天的时间段的摘要。该表如下所示：

```
mysql> SELECT * FROM mail;
+-----+-----+-----+-----+-----+-----+
| t    | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+-----+
| 2006-05-11 10:15:08 | barb   | saturn  | tricia  | mars    | 58274 |
| 2006-05-12 12:48:13 | tricia | mars    | gene    | venus   | 194925 |
| 2006-05-12 15:02:49 | phil   | mars    | phil    | saturn  | 1048  |
| 2006-05-13 13:59:18 | barb   | saturn  | tricia  | venus   | 271   |
| 2006-05-14 09:31:37 | gene   | venus  | barb    | mars    | 2291  |
| 2006-05-14 11:52:17 | phil   | mars    | tricia  | saturn  | 5781  |
...

```

为了确定一天中的每个小时有多少条被发送的消息，可以使用下面的语句：

```
mysql> SELECT HOUR(t) AS hour, COUNT(HOUR(t)) AS count
      -> FROM mail GROUP BY hour;
+-----+-----+
| hour | count |
+-----+-----+
| 7    | 1    |
| 8    | 1    |
| 9    | 2    |
| 10   | 2    |
| 11   | 1    |
| 12   | 2    |
| 13   | 1    |
| 14   | 1    |
| 15   | 1    |
| 17   | 2    |

```

```
| 22 | 1 |
| 23 | 1 |
+-----+-----+
```

此处，摘要类别为一天中的各个小时，然而，该摘要是“不完全的”，因为其中只包含在 mail 表中存在的那些小时的条目。为了产生包含一天中所有小时的摘要，甚至包括那些在此期间没有发送消息的小时，需要创建一个列出每个类别（即每小时）的引用表：

```
mysql> CREATE TABLE ref (h INT);
mysql> INSERT INTO ref (h)
-> VALUES(0),(1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),
-> (12),(13),(14),(15),(16),(17),(18),(19),(20),(21),(22),(23);
```

然后使用 LEFT JOIN 连接引用表和 mail 表：

```
mysql> SELECT ref.h AS hour, COUNT(mail.t) AS count
-> FROM ref LEFT JOIN mail ON ref.h = HOUR(mail.t)
-> GROUP BY hour;
+-----+-----+
| hour | count |
+-----+-----+
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 1 |
| 8 | 1 |
| 9 | 2 |
| 10 | 2 |
| 11 | 1 |
| 12 | 2 |
| 13 | 1 |
| 14 | 1 |
| 15 | 1 |
| 16 | 0 |
| 17 | 2 |
| 18 | 0 |
| 19 | 0 |
| 20 | 0 |
| 21 | 0 |
| 22 | 1 |
| 23 | 1 |
+-----+-----+
```

现在摘要为一天中的个小时都包含了一个条目，因为 LEFT JOIN 强制输出必须包含引用表中的每个行，无论在 mail 表中是否有相关内容。

上面的例子使用引用表和 LEFT JOIN 来填补类别表中的缺口。同样，使用引用表来发现数据集中的缺口也是可能的，也就是说，确定哪些类别在被摘要的数据中没有出现。下面

的语句通过查找在 mail 表中没有与其类别值相匹配的引用行，显示了一天中那些没有发送消息的小时：

```
mysql> SELECT ref.h AS hour
-> FROM ref LEFT JOIN mail ON ref.h = HOUR(mail.t)
-> WHERE mail.t IS NULL;
+-----+
| hour |
+-----+
|    0 |
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|   16 |
|   18 |
|   19 |
|   20 |
|   21 |
+-----+
```

对于摘要语句来说，包含了类别列表的引用表是很有用处的，但是手动创建这样的表是费力且容易出错的。你或许已经发现编写使用类别值范围端点的脚本，以产生引用表的方法更为可取。本质上，这种类型的脚本担负起迭代器的角色，为该范围中的每个值产生一行。

下面的 Perl 脚本 make_date_list.pl 展示了这种方法的一个例子。它产生一个包含某个特定日期范围内的每个日期的引用表，并且对该表使用了索引以提高大量连接操作的速度。

```
#!/usr/bin/perl
# make_date_list.pl - 创建一张每个日期条目都是在给定范围内的表。该表可以在生成摘要时
# 被用于 LEFT JOIN，以保证每个日期都出现在摘要中，而不管被摘要的数据实际上是否包含
# 给定某天的任何数值
# 使用方法: make_date_list.pl db_name tbl_name col_name min_date max_date

use strict;
use warnings;
use DBI;

# ... 处理命令行选项 (未展示) ...

# 检查参数数目

@ARGV == 5 or die "$usage\n";
my ($db_name, $tbl_name, $col_name, $min_date, $max_date) = @ARGV;
```

```

# ... 连接数据库(未展示) ...

# 确定日期范围内的天数

my $days = $dbh->selectrow_array (qq{ SELECT DATEDIFF(?,?) + 1 },
                                  undef, $max_date, $min_date);

print "Minimum date: $min_date\n";
print "Maximum date: $max_date\n";
print "Number of days spanned by range: $days\n";
die "Date range is too small\n" if $days < 1;

# 如果表存在则将其删除，然后重新建立

$dbh->do ("DROP TABLE IF EXISTS $db_name.$tbl_name");
$dbh->do (qq{
    CREATE TABLE $db_name.$tbl_name
    ($col_name DATE NOT NULL, PRIMARY KEY ($col_name))
});

# 将日期范围内的每一个日期添加到表中

my $sth = $dbh->prepare (qq{
    INSERT INTO $db_name.$tbl_name ($col_name) VALUES(? + INTERVAL ? DAY)
});
foreach my $i (0 .. $days-1)
{
    $sth->execute ($min_date, $i);
}

```

由 make_date_list.pl 产生的引用表可以用来产生每个日期的摘要或者查找在表中没有的日期。假设你需要对 driver_log 表进行摘要以确定每天在公路上有多少个司机。该表具有这些行：

```

mysql> SELECT * FROM driver_log ORDER BY rec_id;
+-----+-----+-----+
| rec_id | name  | trav_date | miles |
+-----+-----+-----+
|    1   | Ben   | 2006-08-30 | 152  |
|    2   | Suzi  | 2006-08-29 | 391  |
|    3   | Henry | 2006-08-29 | 300  |
|    4   | Henry | 2006-08-27 | 96   |
|    5   | Ben   | 2006-08-29 | 131  |
|    6   | Henry | 2006-08-26 | 115  |
|    7   | Suzi  | 2006-09-02 | 502  |
|    8   | Henry | 2006-09-01 | 197  |
|    9   | Ben   | 2006-09-02 | 79   |
|   10   | Henry | 2006-08-30 | 203  |
+-----+-----+-----+

```

一个简单的摘要如下所示：

```

mysql> SELECT trav_date, COUNT(trav_date) AS drivers
-> FROM driver_log GROUP BY trav_date;

```

```
+-----+-----+
| trav_date | drivers |
+-----+-----+
| 2006-08-26 |      1 |
| 2006-08-27 |      1 |
| 2006-08-29 |      3 |
| 2006-08-30 |      2 |
| 2006-09-01 |      1 |
| 2006-09-02 |      2 |
+-----+-----+
```

但是，该摘要没有显示没有司机活动的日期。为了产生包含每个缺失日期的摘要，可以利用 make_date_list.pl。从刚展现的简单摘要来看，我们能够得知其最小和最大的日期，因此产生一个名为 ref 的引用表，包含日期的列 d 跨越了这些日期：

```
% make_date_list.pl cookbook ref d 2006-08-26 2006-09-02
Minimum date: 2006-08-26
Maximum date: 2006-09-02
Number of days spanned by range: 8
```

在创建引用表之后，使用下面的语句产生完整的摘要：

```
mysql> SELECT ref.d, COUNT(driver_log.trav_date) AS drivers
-> FROM ref LEFT JOIN driver_log ON ref.d = driver_log.trav_date
-> GROUP BY d;
+-----+-----+
| d      | drivers |
+-----+-----+
| 2006-08-26 |      1 |
| 2006-08-27 |      1 |
| 2006-08-28 |      0 |
| 2006-08-29 |      3 |
| 2006-08-30 |      2 |
| 2006-08-31 |      0 |
| 2006-09-01 |      1 |
| 2006-09-02 |      2 |
+-----+-----+
```

这里的第二个摘要包含了额外的行，以显示没有司机活动的日期。为了只列出没有司机的日期，可以使用这条语句：

```
mysql> SELECT ref.d
-> FROM ref LEFT JOIN driver_log ON ref.d = driver_log.trav_date
-> WHERE driver_log.trav_date IS NULL
-> ORDER BY d;
+-----+
| d      |
+-----+
| 2006-08-28 |
| 2006-08-31 |
+-----+
```

12.9 计算连续行的差值

Calculating Successive-Row Differences

问题

你有一个表，在它的行中包含了连续累积值，并且你需要计算每对连续行的差值。

解决方案

使用自连接以匹配相邻的每对行，并计算每个行对成员的差值。

讨论

如果你有一个绝对的（或者累积的）值集合，并且需要将之转换为表示每对连续行差值的相对值时，自连接是很有用的。举例来说，如果你开车远行，并记录了每个在休息点前走过的路程，你可以使用相邻点的差值来确定从其中一站到另一站的距离。下表显示了从 Texas 州的 San Antonio 到 Wisconsin 州的 Madison 的旅程中所有的站点。其中每一行显示了到该站点前走过的总路程：

```
mysql> SELECT seq, city, miles FROM trip_log ORDER BY seq;
+-----+-----+-----+
| seq | city      | miles |
+-----+-----+-----+
| 1   | San Antonio, TX | 0    |
| 2   | Dallas, TX     | 263  |
| 3   | Benton, AR    | 566  |
| 4   | Memphis, TN   | 745  |
| 5   | Portageville, MO | 878  |
| 6   | Champaign, IL  | 1164 |
| 7   | Madison, WI   | 1412 |
+-----+-----+-----+
```

自连接能够将这些累积值转换为连续的差值，以表示每个城市到下一城市的距离。下面的语句显示了如何使用行序号来匹配相邻的行对并计算每个行对之间的路程值：

```
mysql> SELECT t1.seq AS seq1, t2.seq AS seq2,
-> t1.city AS city1, t2.city AS city2,
-> t1.miles AS miles1, t2.miles AS miles2,
-> t2.miles-t1.miles AS dist
-> FROM trip_log AS t1 INNER JOIN trip_log AS t2
-> ON t1.seq+1 = t2.seq
-> ORDER BY t1.seq;
+-----+-----+-----+-----+-----+-----+
| seq1|seq2|city1      |city2      | miles1 | miles2 | dist  |
+-----+-----+-----+-----+-----+-----+
| 1   | 2   | San Antonio, TX | Dallas, TX | 0    | 263  | 263  |
+-----+-----+-----+-----+-----+-----+
```

	2		3	Dallas, TX	Benton, AR		263		566		303	
	3		4	Benton, AR	Memphis, TN		566		745		179	
	4		5	Memphis, TN	Portageville, MO		745		878		133	
	5		6	Portageville, MO	Champaign, IL		878		1164		286	
	6		7	Champaign, IL	Madison, WI		1164		1412		248	

`trip_log`表中 `seq`列的存在对于计算连续差值是非常重要的，它需要被用来确定哪个行位于另一行之前，以及将行 n 与行 $n+1$ 匹配。也即是说，如果你需要对绝对或累积值执行相对差值计算，那么该表应当包含序号列。如果某表包含了序号列，但是其中存在断档，那么需要对其进行重编号。如果表中没有这样的列，就增加一个。第 11.5 节和第 11.10 节描述了如何执行这些操作。

一个更复杂的情况发生在需要为多个列计算连续差值并使用计算的结果时。下表 `player_stats` 显示了棒球运动员在赛季中每个月末的累计得分。`ab` 表示球员在给定日期前的击球次数 (at-bats) 总计，`h` 表示其中的击中次数 (hits)。（第一行指示了球员赛季的开始日期，因此 `ab` 和 `h` 值为 0。）

```
mysql> SELECT id, date, ab, h, TRUNCATE(IFNULL(h/ab,0),3) AS ba
-> FROM player_stats ORDER BY id;
+----+-----+-----+-----+
| id | date      | ab   | h    | ba      |
+----+-----+-----+-----+
| 1  | 2006-04-30 | 0    | 0    | 0.000   |
| 2  | 2006-05-31 | 38   | 13   | 0.342   |
| 3  | 2006-06-30 | 109  | 31   | 0.284   |
| 4  | 2006-07-31 | 196  | 49   | 0.250   |
| 5  | 2006-08-31 | 304  | 98   | 0.322   |
+----+-----+-----+-----+
```

查询结果的最后一行同样显示了球员在每个日期的命中率。此列没有在表中存储，但是可以容易地使用命中数与击球数的比值计算出来。结果大体提供了球员在赛季过程中击球效率的变化，但是并没有给出球员在每个月中详细情况。为了确定这点，必须计算每个行对的相对差值。这可以使用 self-join 匹配行 n 和行 $n+1$ ，并计算击球数与命中数间的差值。这些差值可以用来计算每个月的击球效率：

```
mysql> SELECT
-> t1.id AS id1, t2.id AS id2,
-> t2.date,
-> t1.ab AS ab1, t2.ab AS ab2,
-> t1.h AS h1, t2.h AS h2,
-> t2.ab-t1.ab AS abdiff,
-> t2.h-t1.h AS hdiff,
```

```

-> TRUNCATE(IFNULL((t2.h-t1.h)/(t2.ab-t1.ab),0),3) AS ba
-> FROM player_stats AS t1 INNER JOIN player_stats AS t2
-> ON t1.id+1 = t2.id
-> ORDER BY t1.id;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id1 | id2 | date      | ab1 | ab2 | h1 | h2 | abdiff | hdifff | ba   |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1   | 2   | 2006-05-31 | 0   | 38  | 0  | 13  | 38    | 13    | 0.342 |
| 2   | 3   | 2006-06-30 | 38  | 109 | 13  | 31  | 71    | 18    | 0.253 |
| 3   | 4   | 2006-07-31 | 109 | 196 | 31  | 49  | 87    | 18    | 0.206 |
| 4   | 5   | 2006-08-31 | 196 | 304 | 49  | 98  | 108   | 49    | 0.453 |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

这结果显示的信息比原始表清楚得多，该球员在赛季初表现良好，但是在赛季中期，特别是7月份有所退步，但同样也表明了他在8月份的效率多么强大。

12.10 发现累积和与动态均值

Finding Cumulative Sums and Running Averages

问题

你有一个一段时间内测量的观察数据集合，并且希望计算每个度量点的累积和或者动态均值。

解决方案

使用自连接产生每个度量点的连续观测数据集合，然后使用聚集函数对每个值集合计算其和或平均值。

讨论

第12.9节阐述了自连接是如何根据绝对值产生相对值。同样，自连接也可以做相反的事情，即为一个观测数据集合的每个连续阶段产生累积值。下表显示了一段日期的降雨量测量数据，其中每一行的值显示了观测日期和降雨量的英寸值：

```

mysql> SELECT date, precip FROM rainfall ORDER BY date;
+-----+-----+
| date       | precip |
+-----+-----+
| 2006-06-01 | 1.50  |
| 2006-06-02 | 0.00  |
| 2006-06-03 | 0.50  |
| 2006-06-04 | 0.00  |
| 2006-06-05 | 1.00  |
+-----+-----+

```

为了计算至给定日期前的累计的降雨量，需要将当天的降雨量与该日期之前的所有降雨量值相加。举个例子，确定至 2006-06-03 累积降雨量的方法如下：

```
mysql> SELECT SUM(precip) FROM rainfall WHERE date <= '2006-06-03';
+-----+
| SUM(precip) |
+-----+
|      2.00 |
+-----+
```

如果你需要为表中的每个日期都计算其累积降雨量，那么单独计算这些值是乏味而又缓慢的，而自连接可以运用一个语句计算所有日期的累积值。使用 rainfall 表的一个实例作为引用，并为另一个表实例中的每行的日期，确定在其之前的所有行的 precip 值的和。

下面的语句显示了每一天的日降雨量和累积降雨量：

```
mysql> SELECT t1.date, t1.precip AS 'daily precip',
-> SUM(t2.precip) AS 'cum. precip'
-> FROM rainfall AS t1 INNER JOIN rainfall AS t2
-> ON t1.date >= t2.date
-> GROUP BY t1.date;
+-----+-----+-----+
| date | daily precip | cum. Precip |
+-----+-----+-----+
| 2006-06-01 |      1.50 |      1.50 |
| 2006-06-02 |      0.00 |      1.50 |
| 2006-06-03 |      0.50 |      2.00 |
| 2006-06-04 |      0.00 |      2.00 |
| 2006-06-05 |      1.00 |      3.00 |
+-----+-----+-----+
```

自连接可以被扩展用来显示每个日期之前的天数，以及这段时间降雨量总计的平均值：

```
mysql> SELECT t1.date, t1.precip AS 'daily precip',
-> SUM(t2.precip) AS 'cum. precip',
-> COUNT(t2.precip) AS 'days elapsed',
-> AVG(t2.precip) AS 'avg. precip'
-> FROM rainfall AS t1 INNER JOIN rainfall AS t2
-> ON t1.date >= t2.date
-> GROUP BY t1.date;
+-----+-----+-----+-----+
| date | daily precip | cum. Precip | days elapsed | avg. precip |
+-----+-----+-----+-----+
| 2006-06-01 |      1.50 |      1.50 |          1 |      1.500000 |
| 2006-06-02 |      0.00 |      1.50 |          2 |      0.750000 |
| 2006-06-03 |      0.50 |      2.00 |          3 |      0.666667 |
| 2006-06-04 |      0.00 |      2.00 |          4 |      0.500000 |
| 2006-06-05 |      1.00 |      3.00 |          5 |      0.600000 |
+-----+-----+-----+-----+
```

在前面的语句中，由于表中没有缺失的时期，每个日期之前的天数及动态平均降雨量可以容易地使用 COUNT() 和 AVG() 计算出来。但如果允许存在缺失的日期，计算将会变得较为复杂，因为每次计算中已过去的天数不再等于行数了。你可以通过删除没有发生降雨的那几天的行发现此时表中产生了两个“缺口”：

```
mysql> DELETE FROM rainfall WHERE precip = 0;
mysql> SELECT date, precip FROM rainfall ORDER BY date;
+-----+-----+
| date | precip |
+-----+-----+
| 2006-06-01 | 1.50 |
| 2006-06-03 | 0.50 |
| 2006-06-05 | 1.00 |
+-----+-----+
```

删除这些行并没有改变剩余行日期的累积和与动态均值，但是它们的计算方法必须改变。如果你再次尝试使用自连接，将会在天数和平均降雨量列中产生错误的结果：

```
mysql> SELECT t1.date, t1.precip AS 'daily precip',
-> SUM(t2.precip) AS 'cum. precip',
-> COUNT(t2.precip) AS 'days elapsed',
-> AVG(t2.precip) AS 'avg. precip'
-> FROM rainfall AS t1 INNER JOIN rainfall AS t2
-> ON t1.date >= t2.date
-> GROUP BY t1.date;
+-----+-----+-----+-----+
| date | daily precip | cum. Precip | days elapsed | avg. precip |
+-----+-----+-----+-----+
| 2006-06-01 | 1.50 | 1.50 | 1 | 1.500000 |
| 2006-06-03 | 0.50 | 2.00 | 2 | 1.000000 |
| 2006-06-05 | 1.00 | 3.00 | 3 | 1.000000 |
+-----+-----+-----+-----+
```

为了修正此问题，必须以不同的方式确定已过去的天数。可以在 sum 函数中使用下面的表达式根据最大和最小日期计算天数：

```
DATEDIFF(MAX(t2.date),MIN(t2.date)) + 1
```

该值可以被用来计算已过去天数列及动态均值，因此最终的语句如下：

```
mysql> SELECT t1.date, t1.precip AS 'daily precip',
-> SUM(t2.precip) AS 'cum. precip',
-> DATEDIFF(MAX(t2.date),MIN(t2.date)) + 1 AS 'days elapsed',
-> SUM(t2.precip) / (DATEDIFF(MAX(t2.date),MIN(t2.date)) + 1)
-> AS 'avg. precip'
-> FROM rainfall AS t1 INNER JOIN rainfall AS t2
-> ON t1.date >= t2.date
-> GROUP BY t1.date;
+-----+-----+-----+-----+
| date | daily precip | cum. Precip | days elapsed | avg. precip |
+-----+-----+-----+-----+
```

2006-06-01	1.50	1.50	1	1.500000
2006-06-03	0.50	2.00	3	0.666667
2006-06-05	1.00	3.00	5	0.600000

正如本例所示，根据相对值计算累积值只需要具备可以将行适当排序的列。（对于 rainfall 表来说，即 date 列。）该列的值不需要是连续的，甚至也不需要是数字，这一点与根据累积值产生差值的计算不同（第 12.9 节），该计算需要表中存在包含完整序号的列。

rainfall 例子中的动态均值是将每天的累积降雨量之和除以天数得到的。当表中没有缺失的日期，该天数与所求和的值的数目是相等的，这使得计算连续均值较容易。而当表中存在缺失的行时，计算变得更为复杂。这证明了根据数据的性质来选择适当的方式计算均值是十分必要的。下一个例子与前一个类似，即同样计算累积和与动态均值，但是它使用了另一种执行计算的方式。

下表显示了一个马拉松选手在一场比赛中各个阶段的成绩。每行的值包括该阶段的公里数和选手跑完赛程所耗费的时间。换句话说，这些值附属于马拉松的各阶段，因而与整个比赛是相关的：

```
mysql> SELECT stage, km, t FROM marathon ORDER BY stage;
+-----+-----+-----+
| stage | km  | t      |
+-----+-----+-----+
| 1     | 5   | 00:15:00 |
| 2     | 7   | 00:19:30 |
| 3     | 9   | 00:29:20 |
| 4     | 5   | 00:17:50 |
+-----+-----+-----+
```

可以用自连接计算每个阶段的累积距离，方法如下：

```
mysql> SELECT t1.stage, t1.km, SUM(t2.km) AS 'cum. km'
-> FROM marathon AS t1 INNER JOIN marathon AS t2
-> ON t1.stage >= t2.stage
-> GROUP BY t1.stage;
+-----+-----+-----+
| stage | km  | cum. Km |
+-----+-----+-----+
| 1     | 5   | 5       |
| 2     | 7   | 12      |
| 3     | 9   | 21      |
| 4     | 5   | 26      |
+-----+-----+-----+
```

可以很容易地通过直接求和计算出累积距离，而计算累积时间稍麻烦一点。必须先将时间转换成秒数，再对其求和，最后将所求和转换为时间值。为了计算选手到每个阶段终点为止的平均速度，需要将累积的距离除以累积的时间。将所有的操作整合起来，产生下面的语句：

```
mysql> SELECT t1.stage, t1.km, t1.t,
-> SUM(t2.km) AS 'cum. km',
-> SEC_TO_TIME(SUM(TIME_TO_SEC(t2.t))) AS 'cum. t',
-> SUM(t2.km)/(SUM(TIME_TO_SEC(t2.t))/(60*60)) AS 'avg. km/hour'
-> FROM marathon AS t1 INNER JOIN marathon AS t2
-> ON t1.stage >= t2.stage
-> GROUP BY t1.stage;
+-----+-----+-----+-----+-----+
| stage | km | t | cum. km | cum. t | avg. km/hour |
+-----+-----+-----+-----+-----+
| 1 | 5 | 00:15:00 | 5 | 00:15:00 | 20.0000 |
| 2 | 7 | 00:19:30 | 12 | 00:34:30 | 20.8696 |
| 3 | 9 | 00:29:20 | 21 | 01:03:50 | 19.7389 |
| 4 | 5 | 00:17:50 | 26 | 01:21:40 | 19.1020 |
+-----+-----+-----+-----+-----+
```

从中我们可以发现该选手的平均速度在比赛的第二阶段稍有增加，但之后又开始降低（大概是由于体力原因）。

12.11 使用连接控制查询输出的顺序

Using a Join to Control Query Output Order

问题

你需要对某个语句的输出进行排序，但所要排序的特征无法使用 ORDER BY 来指定。举个例子，你需要根据分组对行集合排序，将包含最多行的组排在第一位，而将包含最少行的组排在最后。但是“每组中的行数”并不是每个行的属性，因此你不能使用它来排序。

解决方案

取出排序所需的信息并将之存储在辅助表中，然后连接原始表与辅助表，并使用辅助表来控制排列顺序。

讨论

当排序一个查询结果时，大多数情况下，你可以使用 ORDER BY 子句来指定哪一列或者哪几列被用来排序。但是有些时候你需要进行排序的值并没有出现在行中，当你想要根据组属性来排列行时就属于后一种情况。下面的例子使用 driver_log 表中的行来说明这点，该表包含这些行：

```
mysql> SELECT * FROM driver_log ORDER BY rec_id;
+-----+-----+-----+-----+
| rec_id | name  | trav_date | miles |
+-----+-----+-----+-----+
| 1     | Ben   | 2006-08-30 | 152  |
| 2     | Suzi  | 2006-08-29 | 391  |
| 3     | Henry | 2006-08-29 | 300  |
| 4     | Henry | 2006-08-27 | 96   |
| 5     | Ben   | 2006-08-29 | 131  |
| 6     | Henry | 2006-08-26 | 115  |
| 7     | Suzi  | 2006-09-02 | 502  |
| 8     | Henry | 2006-09-01 | 197  |
| 9     | Ben   | 2006-09-02 | 79   |
| 10    | Henry | 2006-08-30 | 203  |
+-----+-----+-----+-----+
```

前面的语句使用表中存在的 ID 列对行进行排序，但是如果你需要显示一个列表，并根据未出现在表中的摘要值进行排序该怎么做呢？解决此问题需要一点窍门。假设你需要根据日期显示每个司机所在行，你不能使用摘要查询，因为那样你将无法恢复独立的司机行。但是，你也不能不使用摘要查询，因为摘要值需要被用来排序。走出困境的方法是创建另外一个包含每个司机摘要值的表，然后将之与原始表相连接。通过该方法你可以在产生独立行的同时，使用摘要值进行排序。

为了将行驶的路程总和摘要至另一个表中，可以做如下工作：

```
mysql> CREATE TABLE tmp
-> SELECT name, SUM(miles) AS driver_miles FROM driver_log GROUP BY name;
```

这产生了我们需要的值以用来根据总里程的值对 name 进行排序：

```
mysql> SELECT * FROM tmp ORDER BY driver_miles DESC;
+-----+-----+
| name | driver_miles |
+-----+-----+
| Henry |      911 |
| Suzi |      893 |
| Ben  |      362 |
+-----+-----+
```

然后使用 name 值将摘要表与 driver_log 表相连接，并利用 driver_miles 值排序所得结果。下面语句在结果中显示了英里数总计，但只是为了阐明这些行是如何排序的，实际上不一定要显示它们（英里数），它们只是在 ORDER BY 子句中被用到。

```
mysql> SELECT tmp.driver_miles, driver_log.*
-> FROM driver_log INNER JOIN tmp
-> ON driver_log.name = tmp.name
-> ORDER BY tmp.driver_miles DESC, driver_log.trav_date;
+-----+-----+-----+-----+
| driver_miles | rec_id | name  | trav_date | miles |
+-----+-----+-----+-----+
```

```

| 911 | 6 | Henry | 2006-08-26 | 115 |
| 911 | 4 | Henry | 2006-08-27 | 96 |
| 911 | 3 | Henry | 2006-08-29 | 300 |
| 911 | 10 | Henry | 2006-08-30 | 203 |
| 911 | 8 | Henry | 2006-09-01 | 197 |
| 893 | 2 | Suzi | 2006-08-29 | 391 |
| 893 | 7 | Suzi | 2006-09-02 | 502 |
| 362 | 5 | Ben | 2006-08-29 | 131 |
| 362 | 1 | Ben | 2006-08-30 | 152 |
| 362 | 9 | Ben | 2006-09-02 | 79 |
+-----+-----+-----+-----+-----+

```

为了避免使用临时表，可以在 FROM 子句中使用子查询选择同样的行：

```

mysql> SELECT tmp.driver_miles, driver_log.*
-> FROM driver_log INNER JOIN
-> (SELECT name, SUM(miles) AS driver_miles
-> FROM driver_log GROUP BY name) AS tmp
-> ON driver_log.name = tmp.name
-> ORDER BY tmp.driver_miles DESC, driver_log.trav_date;
+-----+-----+-----+-----+-----+
| driver_miles | rec_id | name | trav_date | miles |
+-----+-----+-----+-----+-----+
| 911 | 6 | Henry | 2006-08-26 | 115 |
| 911 | 4 | Henry | 2006-08-27 | 96 |
| 911 | 3 | Henry | 2006-08-29 | 300 |
| 911 | 10 | Henry | 2006-08-30 | 203 |
| 911 | 8 | Henry | 2006-09-01 | 197 |
| 893 | 2 | Suzi | 2006-08-29 | 391 |
| 893 | 7 | Suzi | 2006-09-02 | 502 |
| 362 | 5 | Ben | 2006-08-29 | 131 |
| 362 | 1 | Ben | 2006-08-30 | 152 |
| 362 | 9 | Ben | 2006-09-02 | 79 |
+-----+-----+-----+-----+-----+

```

12.12 在单个查询中整合几个结果集

Combining Several Result Sets in a Single Query

问题

你需要从几个表中选择行，或者从一个表中选择几个行的集合——并且把它们作为单个结果集。

解决方案

使用 UNION 操作符将多个查询结果整合为一个。

讨论

连接对于并排地整合从不同的表中的列是很有用的，但是当你需要得到一个结果集，其中

包含了来源于几个表的行集合，或者同一个表的多个行集合时，连接就不起作用了，而这是 UNION 运用的典型操作实例。UNION 使你能够运行几个 SELECT 语句并将它们的结果整合在一起。也即是说，不是运行多个查询并接收多个结果集，而是接收到单个结果集。

假设你有两个表，分别列出潜在客户和实际客户，并且还有第三个表列出了你的供货商。现在你需要创建一个邮件列表，将所有三个表的姓名和地址合并到一起，UNION 提供了一个实现方式。假定三个表具有下面的内容：

```
mysql> SELECT * FROM prospect;
+-----+-----+-----+
| fname | lname | addr
+-----+-----+-----+
| Peter | Jones | 482 Rush St., Apt. 402 |
| Bernice | Smith | 916 Maple Dr.
+-----+
mysql> SELECT * FROM customer;
+-----+-----+-----+
| last_name | first_name | address
+-----+-----+-----+
| Peterson | Grace | 16055 Seminole Ave. |
| Smith | Bernice | 916 Maple Dr.
| Brown | Walter | 8602 1st St.
+-----+
mysql> SELECT * FROM vendor;
+-----+-----+
| company | street
+-----+-----+
| ReddyParts, Inc. | 38 Industrial Blvd.
| Parts-to-go, Ltd. | 213B Commerce Park.
+-----+
```

这些表中的列是相似而又不完全一样的。prospect 和 customer 表对于姓名两个列采用了不同的名称，而 vendor 表只有一个名字列。这对于 UNION 来说无关紧要，你所要做的就是确定从每个表中选择同样数目的列，并遵循同样的顺序。下面的语句阐明了如何一次性从三个表选择名称和地址：

```
mysql> SELECT fname, lname, addr FROM prospect
      -> UNION
      -> SELECT first_name, last_name, address FROM customer
      -> UNION
      -> SELECT company, '', street FROM vendor;
+-----+-----+-----+
| fname | lname | addr
+-----+-----+-----+
| Peter | Jones | 482 Rush St., Apt. 402 |
| Bernice | Smith | 916 Maple Dr.
| Grace | Peterson | 16055 Seminole Ave.
| Walter | Brown | 8602 1st St.
| ReddyParts, Inc. | | 38 Industrial Blvd.
```

结果集的列名取决于第一个 SELECT 语句所获得的列名。注意默认情况下，UNION 会消除重复项，比如 Bernice Smith 在 prospect 和 customer 两个表中都出现过，但在最后的结果中只出现一次。如果你要选择所有的行，包括发生重复的，可以在每个 UNION 关键字后面加上 ALL：

```
mysql> SELECT fname, lname, addr FROM prospect
-> UNION ALL
-> SELECT first_name, last_name, address FROM customer
-> UNION ALL
-> SELECT company, '', street FROM vendor;
+-----+-----+-----+
| fname | lname | addr
+-----+-----+-----+
| Peter | Jones | 482 Rush St., Apt. 402 |
| Bernice | Smith | 916 Maple Dr. |
| Grace | Peterson | 16055 Seminole Ave. |
| Bernice | Smith | 916 Maple Dr. |
| Walter | Brown | 8602 1st St. |
| ReddyParts, Inc. | | 38 Industrial Blvd. |
| Parts-to-go, Ltd. | | 213B Commerce Park.
+-----+-----+-----+
```

由于必须在每个表中选择同样数目的列，因此对于 vendor 表（该表只有一个名字列）的查询提取了一个空的姓列。另一种保证同样数目列的方式是将 prospect 和 customer 表中的姓、名列合并为一个列：

```
mysql> SELECT CONCAT(lname, ', ', fname) AS name, addr FROM prospect
-> UNION
-> SELECT CONCAT(last_name, ', ', first_name), address FROM customer
-> UNION
-> SELECT company, street FROM vendor;
+-----+-----+
| name | addr
+-----+-----+
| Jones, Peter | 482 Rush St., Apt. 402 |
| Smith, Bernice | 916 Maple Dr. |
| Peterson, Grace | 16055 Seminole Ave. |
| Brown, Walter | 8602 1st St. |
| ReddyParts, Inc. | 38 Industrial Blvd. |
| Parts-to-go, Ltd. | 213B Commerce Park.
+-----+-----+
```

为了对结果集排序，可以将每个 SELECT 语句置于一对圆括号中，并在最后增加 ORDER BY 句。任何在 ORDER BY 中列举的列名都应该在第一个 SELECT 中出现过，因为这些是被用来作为结果集的列名。举个例子，如果根据 name 排序，应该做如下操作：

```

mysql> (SELECT CONCAT(lname, ', ', fname) AS name, addr FROM prospect)
-> UNION
-> (SELECT CONCAT(last_name, ', ', first_name), address FROM customer)
-> UNION
-> (SELECT company, street FROM vendor)
-> ORDER BY name;
+-----+-----+
| name | addr |
+-----+-----+
| Brown, Walter | 8602 1st St. |
| Jones, Peter | 482 Rush St., Apt. 402 |
| Parts-to-go, Ltd. | 213B Commerce Park. |
| Peterson, Grace | 16055 Seminole Ave. |
| ReddyParts, Inc. | 38 Industrial Blvd. |
| Smith, Bernice | 916 Maple Dr. |
+-----+-----+

```

确保连续地从每个 SELECT 中获取结果是可能的，尽管你需要产生一个额外的列以用来排序。首先将 SELECT 置入括号中，再为其中每一个增加排序值列，最后在末尾增加 ORDER BY 以使用该列进行排序：

```

mysql> (SELECT 1 AS sortval, CONCAT(lname, ', ', fname) AS name, addr
-> FROM prospect)
-> UNION
-> (SELECT 2 AS sortval, CONCAT(last_name, ', ', first_name) AS name, address
-> FROM customer)
-> UNION
-> (SELECT 3 AS sortval, company, street FROM vendor)
-> ORDER BY sortval;
+-----+-----+
| sortval | name | addr |
+-----+-----+
| 1 | Jones, Peter | 482 Rush St., Apt. 402 |
| 1 | Smith, Bernice | 916 Maple Dr. |
| 2 | Peterson, Grace | 16055 Seminole Ave. |
| 2 | Smith, Bernice | 916 Maple Dr. |
| 2 | Brown, Walter | 8602 1st St. |
| 3 | ReddyParts, Inc. | 38 Industrial Blvd. |
| 3 | Parts-to-go, Ltd. | 213B Commerce Park. |
+-----+-----+

```

如果你还需要对每个 SELECT 内部所获取的行进行排序，可以在 ORDER BY 子句中再增加第二个排序列。下面的查询在每个 SELECT 中根据 name 排序：

```

mysql> (SELECT 1 AS sortval, CONCAT(lname, ', ', fname) AS name, addr
-> FROM prospect)
-> UNION
-> (SELECT 2 AS sortval, CONCAT(last_name, ', ', first_name) AS name, address
-> FROM customer)
-> UNION
-> (SELECT 3 AS sortval, company, street FROM vendor)
-> ORDER BY sortval, name;
+-----+-----+
| sortval | name | addr |
+-----+-----+

```

```
+-----+-----+-----+
| 1 | Jones, Peter | 482 Rush St., Apt. 402 |
| 1 | Smith, Bernice | 916 Maple Dr. |
| 2 | Brown, Walter | 8602 1st St. |
| 2 | Peterson, Grace | 16055 Seminole Ave. |
| 2 | Smith, Bernice | 916 Maple Dr. |
| 3 | Parts-to-go, Ltd. | 213B Commerce Park. |
| 3 | ReddyParts, Inc. | 38 Industrial Blvd. |
+-----+-----+-----+
```

相似的语法同样可以用于 LIMIT。也即是说，你可以在末尾使用 LIMIT 子句限制整个结果集，或者限制每个单独的 SELECT 语句。典型地，LIMIT 会与 ORDER BY 结合使用。假设你需要选择一个幸运中奖者，向其派发奖励品。为了在结合三个表的结果中随机选择一个获奖者，可以如下操作：

```
mysql> (SELECT CONCAT(lname, ', ', fname) AS name, addr FROM prospect)
-> UNION
-> (SELECT CONCAT(last_name, ', ', first_name), address FROM customer)
-> UNION
-> (SELECT company, street FROM vendor)
-> ORDER BY RAND() LIMIT 1;
+-----+-----+
| name | addr |
+-----+-----+
| Peterson, Grace | 16055 Seminole Ave. |
+-----+-----+
```

相对地，如果要从每个表中选择一个获奖者并整合结果，需要如下操作：

```
mysql> (SELECT CONCAT(lname, ', ', fname) AS name, addr
-> FROM prospect ORDER BY RAND() LIMIT 1)
-> UNION
-> (SELECT CONCAT(last_name, ', ', first_name), address
-> FROM customer ORDER BY RAND() LIMIT 1)
-> UNION
-> (SELECT company, street
-> FROM vendor ORDER BY RAND() LIMIT 1);
+-----+-----+
| name | addr |
+-----+-----+
| Smith, Bernice | 916 Maple Dr. |
| ReddyParts, Inc. | 38 Industrial Blvd. |
+-----+-----+
```

也许你对结果感到奇怪（“为什么没有选出三个行呢？”），记住 Bernice 在两个表中都被列出，而 UNION 消除了此重复项。如果第一和第二个 SELECT 语句都碰巧选择了 Bernice，其中一个实例会被删除，从而最后的结果只包含了两行。（如果三个表中没有重复项，该语句将始终返回三行。）当然你可以使用 UNION ALL 确保在任何情况下都出现三行结果。

12.13 识别并删除失配或独立行

Identifying and Removing Mismatched or Unattached Rows

问题

你具有两个相关的关系集，但它们的关系可能并不完全。你需要确定其中一个关系集中是否存在独立的记录（与另一个关系集中的任何一个记录都不匹配），并且如果存在的话，则将之删除。举个例子，这可能会发生在你从一个外部数据源中接收数据并且必须对其进行检查以验证其完整性时。

解决方案

使用 LEFT JOIN 以便在每个表中发现独立的值，如果存在并且你希望删除它们，可以使用针对多表的 DELETE 语句。另一种可能的方法是使用 NOT IN 子查询来识别和删除独立行。

讨论

内部连接可以用于识别关联的发生，而外部连接可以用于识别关联的缺失。当你具有相关的关系集但是其中的关系可能并不完全时，外部连接的这种属性是很有利用价值的。

数据集间的失配可能发生在你从外部源中接收两个数据文件时，它们被假定为相关的，但实际上它们关系的完整性具有缺陷。失配还可能作为一个有意动作的预期结果而发生。假设一个在线讨论板使用父表列出讨论的主题，使用子表列出每个主题所提交的文章。如果你删除了子表中旧的文章行，可能会导致父表中某一主题行不再具有任何子项。如果这样，该主题相关帖子的缺失暗示了此问题可能已经无人关注了，因而主题表中的父行也可以被删除。在此情况下，你可以显式地删除一个子行集合，该操作或许会影响父行，使它们同样满足被删除的条件。

当你遇到关联表具有失配行的时候，你可以使用 SQL 语句进行分析和修改。明确地说，即识别其中的独立行并删除它们，以修复表间的关系：

- 可以使用 LEFT JOIN 识别独立行，因为这属于“查找不到行”问题。（参见第 12.2 节以获得关于 LEFT JOIN 的信息。）
- 为了删除不匹配行，使用多表 DELETE 语句以类似于 LEFT JOIN 的方式指明哪些行需要被删除。

知道不匹配数据的存在是很有用的，因为你可以警示给你发送数据的一方。这在数据采集方法中通常是一个必须被改正的错误信号。举例来说，对于销售数据，一个区域数据的缺失可能说明某些区域经理没有报告，并且这个疏忽没有被注意到。

下面的例子使用两个描述销售地区和每个地区销量的数据集显示了如何识别并删除不匹配行。一个数据集包含了每个销售区域的 ID 和位置：

```
mysql> SELECT * FROM sales_region ORDER BY region_id;
+-----+-----+
| region_id | name
+-----+-----+
| 1 | London, United Kingdom |
| 2 | Madrid, Spain |
| 3 | Berlin, Germany |
| 4 | Athens, Greece |
+-----+-----+
```

而另一个数据集包含了销售量数据。每一行含有给定季度的销售总量，并且指明了该行的销售区域：

```
mysql> SELECT * FROM sales_volume ORDER BY region_id, year, quarter;
+-----+-----+-----+-----+
| region_id | year | quarter | volume |
+-----+-----+-----+-----+
| 1 | 2006 | 1 | 100400 |
| 1 | 2006 | 2 | 120000 |
| 3 | 2006 | 1 | 280000 |
| 3 | 2006 | 2 | 250000 |
| 5 | 2006 | 1 | 18000 |
| 5 | 2006 | 2 | 32000 |
+-----+-----+-----+-----+
```

通过简单地观察就可以发现任何一表都不是和另一个完全匹配的。销售区域 2 和 4 没有出现在销售量表中，而销售量表包含了区域 5，该区域并没有出现在销售区域表中。但是我们不需要通过观察来检查这些表，我们希望使用 SQL 语句可以做同样的工作，即找到不匹配行。

不匹配性的识别大多利用外部连接，举个例子，为了发现没有销量行的销售地区，使用下面的 LEFT JOIN：

```
mysql> SELECT sales_region.region_id AS 'unmatched region row IDs'
    -> FROM sales_region LEFT JOIN sales_volume
    -> ON sales_region.region_id = sales_volume.region_id
    -> WHERE sales_volume.region_id IS NULL;
+-----+
| unmatched region row IDs |
+-----+
| 2 |
| 4 |
+-----+
```

相反地，为了查找与已知地区不相关的销售量行，需要将两个表的角色对调：

```
mysql> SELECT sales_volume.region_id AS 'unmatched volume row IDs'
-> FROM sales_volume LEFT JOIN sales_region
-> ON sales_volume.region_id = sales_region.region_id
-> WHERE sales_region.region_id IS NULL;
+-----+
| unmatched volume row IDs |
+-----+
|      5 |
|      5 |
+-----+
```

这种情况下，如果对于缺失的地区具有多个销量行，那么 ID 会出现多次。可以使用 SELECT DISTINCT 只列出不匹配 ID 一次：

```
mysql> SELECT DISTINCT sales_volume.region_id AS 'unmatched volume row IDs'
-> FROM sales_volume LEFT JOIN sales_region
-> ON sales_volume.region_id = sales_region.region_id
-> WHERE sales_region.region_id IS NULL
+-----+
| unmatched volume row IDs |
+-----+
|      5 |
+-----+
```

要删除不匹配行，你可以在多表删除语句中使用它们的 ID。为了构建正确的多表 DELETE 语句以删除表中的不匹配行，只需要利用 SELECT 语句识别这些行，然后将该语句 FROM 关键字后面的内容取代 DELETE table_name 中的对应部分。举例来说，找出无子项的父项的查询如下所示：

```
SELECT sales_region.region_id AS 'unmatched region row IDs'
FROM sales_region LEFT JOIN sales_volume
    ON sales_region.region_id = sales_volume.region_id
WHERE sales_volume.region_id IS NULL;
```

相应的 DELETE 如下所示：

```
DELETE sales_region
FROM sales_region LEFT JOIN sales_volume
    ON sales_region.region_id = sales_volume.region_id
WHERE sales_volume.region_id IS NULL;
```

相反地，识别无父项的子项的语句如下所示：

```
SELECT sales_volume.region_id AS 'unmatched volume row IDs'
FROM sales_volume LEFT JOIN sales_region
    ON sales_volume.region_id = sales_region.region_id
WHERE sales_region.region_id IS NULL;
```

删除它们的相应 DELETE 语句如下：

```
DELETE sales_volume
FROM sales_volume LEFT JOIN sales_region
```

```
ON sales_volume.region_id = sales_region.region_id  
WHERE sales_region.region_id IS NULL;
```

你还可以使用 NOT IN 子查询来识别和删除不匹配行。显示或删除与销售量行无匹配的销售区域行的语句如下所示：

```
SELECT region_id AS 'unmatched region row IDs'  
FROM sales_region  
WHERE region_id NOT IN (SELECT region_id FROM sales_volume);  
  
DELETE FROM sales_region  
WHERE region_id NOT IN (SELECT region_id FROM sales_volume);
```

识别及删除不匹配的销售量行的语句是类似的，但是这些表的角色发生了颠倒：

```
SELECT region_id AS 'unmatched volume row IDs'  
FROM sales_volume  
WHERE region_id NOT IN (SELECT region_id FROM sales_region);  
  
DELETE FROM sales_volume  
WHERE region_id NOT IN (SELECT region_id FROM sales_region);
```

使用外键来增强引用完整性

数据库系统提供的帮助你在表间维持一致性的一项功能是允许定义外关键字关联。这意味着你可以显式地在表定义中指明父表的主键（比如 sales_region 表的 region_id 列）是另一个表键值的父键（sales_volume 表中的 region_id 列）。通过将子表中的 ID 列定义为指向父表 ID 列的外键，数据库系统可以增强对非法操作的限制。举例来说，可以阻止你使用在父表中未出现的 ID 创建子表中的行，或者在删除父表中的行时，没有首先删掉对应的子表行。外键的实现可能提供了级联删除与更新：如果你删除或更新了一个父行，数据库引擎将删除或更新的影响级联至子表并自动为你删除或更新子表行。MySQL 中的 InnoDB 存储引擎支持外关键字及级联删除与更新。

12.14 为不同数据库间的表执行连接

Performing a Join Between Tables in Different Databases

问题

你需要对一些表执行连接操作，但是它们并不处于同一个数据库。

解决方案

使用数据库名称限定符以告知 MySQL 到何处查找这些表。

讨论

有时必须对处于不同数据库的两个表之间执行连接操作。为了达到此目的，需要充分修饰表名和列名，以使 MySQL 了解你引用的是什么表。迄今为止，我们使用的 artist 和 painting 表暗含着它们处于同一个 cookbook 数据库，这意味着当 cookbook 是默认数据库时，我们可以简单地引用这些表而无需指定数据库名称。举个例子，下面的语句使用了两个表以关联画家和他们的作品：

```
SELECT artist.name, painting.title  
  FROM artist INNER JOIN painting  
    ON artist.a_id = painting.a_id;
```

但是假设 artist 表在 db1 数据库而 painting 在 db2 数据库中，为了适应此情况，需要在每个表名前面使用前缀进行修饰以指明其属于哪个数据库。完整的修饰后的连接格式如下所示：

```
SELECT db1.artist.name, db2.painting.title  
  FROM db1.artist INNER JOIN db2.painting  
    ON db1.artist.a_id = db2.painting.a_id;
```

如果没有默认数据库，或者默认数据库既不是 db1 也不是 db2，那么必须使用这种完整的修饰格式。如果默认数据库是 db1 或 db2 的一种，你可以省去相应的修饰符。举例来说，如果默认数据库是 db1，你可以略去 db1 修饰符：

```
SELECT artist.name, db2.painting.title  
  FROM artist INNER JOIN db2.painting  
    ON artist.a_id = db2.painting.a_id;
```

相反，如果默认数据库是 db2，则 db2 修饰符是不必要的：

```
SELECT db1.artist.name, painting.title  
  FROM db1.artist INNER JOIN painting  
    ON db1.artist.a_id = painting.a_id;
```

12.15 同时使用不同的 MySQL 服务器

Using Different MySQL Servers Simultaneously

问题

你需要执行语句，其中使用了位于不同 MySQL 服务器的数据库中的表。

解决方案

建立 FEDERATED 表，该表能够使 MySQL 服务器访问位于另一个 MySQL 服务器的表。另一种方法是为每个服务器打开一个独立的连接，然后手动地将两个表中的信息结合起来，

或者将其中的一个表拷贝到另外一个服务器上，以使你能够在单个服务器上处理两个表。

讨论

在本章中，我们假设所有多表操作相关的表是由同一个 MySQL 服务器管理的。如果这个假设不成立，这些表将难以使用因为连向 MySQL 服务器的连接只能使你直接访问位于该服务器上的表。但是，MySQL 支持 FEDERATED 存储引擎，以使你能够远程访问另一个 MySQL 服务器上的表。对于一个 FEDERATED 表，本地的 MySQL 服务器充当客户端的角色，连接到另一个 MySQL 服务器，以使其能够根据你的需要访问远程表，使远程表的内容如同在本地一样。

下面使用 artist 和 painting 表举例阐述该问题。假设你需要查找达芬奇画作的名字，这需要在 artist 表中获得达芬奇的 ID，然后将它与 painting 表中的行相匹配。如果两个表位于同一个数据库，你可以使用下面的语句执行一个连接以识别这些画作：

```
mysql> SELECT painting.title
    -> FROM artist INNER JOIN painting
    -> ON artist.a_id = painting.a_id
    -> WHERE artist.name = 'Da Vinci';
+-----+
| title      |
+-----+
| The Last Supper |
| The Mona Lisa  |
+-----+
```

现在假设 painting 表不在我们平常连接的 MySQL 服务器中，而是位于另一个远程 MySQL 服务器。我们可以通过创建 FEDERATED 表访问远程表，如同访问本地表一样。FEDERATED 表被定义为与远程表同样的结构。创建 FEDERATED 表的 CREATE TABLE 语句必须包括表选项，以指定 FEDERATED 存储引擎，并使用 connection 字符串告诉本地服务器如何连接远程服务器并定位该表。第 12.1 节显示了 painting 表的原始结构。建立相应的 FEDERATED 表，并如下定义：

```
CREATE TABLE fed_painting
(
    a_id INT UNSIGNED NOT NULL,          # 艺术家 ID
    p_id INT UNSIGNED NOT NULL AUTO_INCREMENT, # 油画 ID
    title VARCHAR(100) NOT NULL,           # 油画名称
    state VARCHAR(2) NOT NULL,            # 购买地
    price INT UNSIGNED,                  # 购买价格(美元)
    INDEX (a_id),
    PRIMARY KEY (p_id)
)
```

```
FENGINE = FEDERATED  
CONNECTION = 'mysql://cbuser:cbpass@remote.example.com/cookbook/painting';
```

这里使用 CONNECTION 的字符串具有下面的格式：

```
mysql://user_name:pass_val@host_name/db_name/tbl_name
```

换句话说，远程服务器位于 remote.example.com，MySQL 用户名和密码为 cbuser 和 cbpass，位于 cookbook 数据库的表名称为 painting。对于你所在的网络，应该根据情况对参数进行调整。创建 FEDERATED 表后，你可以使用它访问远程表，如同其在本地一样。举例来说，要执行本节前面描述的连接操作，其编写方式如下：

```
mysql> SELECT fed_painting.title  
      -> FROM artist INNER JOIN fed_painting  
      -> ON artist.a_id = fed_painting.a_id  
      -> WHERE artist.name = 'Da Vinci';  
+-----+  
| title |  
+-----+  
| The Last Supper |  
| The Mona Lisa |  
+-----+
```

到目前为止，FEDERATED 表只能用来访问其他的 MySQL 服务器，而不能访问其他数据库引擎的服务器。



提示：在 MySQL 5.0 的二进制版本中，除非你使用 MySQL-Max 服务器，FEDERATED 存储引擎将不会被激活。而在 MySQL 5.1 的二进制版本中，FEDERATED 被默认激活。如果你是从源码（任一个版本）中编译 MySQL，使用--with-federated-storage-engine 配置选项以激活 FEDERATED 的支持。

另一种连接位于不同服务器表的方法是编写程序模拟一个连接：

1. 为每个数据库服务器打开一个单独的连接。
2. 运行一个循环以获取 artist 的 ID 和名字。
3. 每次循环中，使用当前的画家 ID 来构建语句，以查找 painting 表中与画家 ID 值相匹配的行，然后向管理 painting 表的服务器发送该语句。在接收画作名称时，将它们与当前画家的名字一起显示。

此技术能够模拟位于两个服务器中的表的连接。顺便提一下，当你需要处理不同类型的数据库引擎上的表时，它也可以使用。（举个例子，你可以用该方法模拟一个 MySQL 表与 PostgreSQL 表的连接。）

第三种方式是将其中一个服务器上的表拷贝到另一个服务器上，然后你可以使用同一个服务器处理两个表，这使你能够在它们之间直接执行标准的连接。参见第 10.16 节以获取更多在服务器直接拷贝表的信息。

12.16 在程序中引用连接的输出列名称

Referring to Join Output Column Names in Programs

问题

你需要在程序中处理连接的结果，但是结果集中的列名不唯一。

解决方案

使用列别名修改查询语句，以使每列具有唯一的名字，或者根据位置引用这些列。

讨论

连接通常从相关表中获取数据列，因此从不同的表中选出的列具有同样的名字也是常见的。下面的连接显示了你的艺术品收藏中的条目（参见 12.1 节）。对于每个作品，它显示了画家姓名、画作名称、你就在哪个州获得的以及它的价值：

```
mysql> SELECT artist.name, painting.title, states.name, painting.price
-> FROM artist INNER JOIN painting INNER JOIN states
-> ON artist.a_id = painting.a_id AND painting.state = states.abbrev;
+-----+-----+-----+-----+
| name | title | name | price |
+-----+-----+-----+-----+
| Da Vinci | The Last Supper | Indiana | 34 |
| Da Vinci | The Mona Lisa | Michigan | 87 |
| Van Gogh | Starry Night | Kentucky | 48 |
| Van Gogh | The Potato Eaters | Kentucky | 67 |
| Van Gogh | The Rocks | Iowa | 33 |
| Renoir | Les Deux Soeurs | Nebraska | 64 |
+-----+-----+-----+-----+
```

该语句对每个输出列使用了表限定符。然而，MySQL 并没有在列头中包含表名，因此输出的列名不能被区分。如果你在处理程序中得到的连接结果，并将这些行存储到一个根据列名引用列值的数据结构中，不唯一的列名会导致无法访问某些值。下面的 Perl 脚本片段说明了难点所在：

```
$stmt = qq{
  SELECT artist.name, painting.title, states.name, painting.price
  FROM artist INNER JOIN painting INNER JOIN states
  ON artist.a_id = painting.a_id AND painting.state = states.abbrev}
```

```

};

$sth = $dbh->prepare ($stmt);
$sth->execute ();
# 确定结果行集合中列数目的两种方法:
# - 检查 NUM_OF_FIELDS 语句的句柄特征
# - 取出一行放到 hash 表中并查看该 hash 表中含有多少关键字
$count1 = $sth->{NUM_OF_FIELDS};
$ref = $sth->fetchrow_hashref ();
$count2 = keys (%$ref);
print "The statement is: $stmt\n";
print "According to NUM_OF_FIELDS, the result set has $count1 columns\n";
print "The column names are: " . join sort ("", @{$sth->{NAME}}) . "\n";
print "According to the row hash size, the result set has $count2 columns\n";
print "The column names are: " . join sort ("", @{$sth->{NAME}}) . "\n";

```

该脚本提交语句，然后确定结果中的列数，方法为首先检查 NUM_OF_FIELDS 属性，然后将行插入 hash 表中，再对 hash 表的键值计数。下面输出了执行这段脚本的结果：

```

According to NUM_OF_FIELDS, the result set has 4 columns
The column names are: name,name,title,price
According to the row hash size, the result set has 3 columns
The column names are: name,price,title

```

这里有个问题：列数并不匹配。第二个 count 为 3（而不是 4），这是因为非唯一的列名会产生被映射为同一个 hash 元素的多个列值。由于 hash 键值冲突的结果，一部分值被丢失了。为了解决这个问题，需要提供别名以保证列名唯一。举个例子，语句可以按如下方式重写：

```

SELECT
  artist.name AS painter, painting.title,
  states.name AS state, painting.price
FROM artist INNER JOIN painting INNER JOIN states
  ON artist.a_id = painting.a_id AND painting.state = states.abbrev

```

如果你重新 make 并运行这段脚本，它的输出变为：

```

According to NUM_OF_FIELDS, the result set has 4 columns
The column names are: painter,price,state,title
According to the row hash size, the result set has 4 columns
The column names are: painter,price,state,title

```

现在两个列计数是相等的了，这暗示了在插入 hash 表时没有值被丢失。

另一种无需重命名列的解决问题的方式是将行置入非 hash 表的数据结构中。比如说，你可以将行存入一个数组中，并在数组中根据列的位置顺序引用它们：

```

while (my @val = $sth->fetchrow_array ())
{
  print "painter: $val[0], title: $val[1], "

```

```
    . "state: $val[2], price: $val[3]\n";
}
```

在其他语言中，上面描述的名字冲突问题或许有不同的解决方案。举例来说，在使用 MySQLdb 模块的 Python 脚本中，该问题发生的方式稍有不同。假设你使用字典（Python 中与 Perl 的 hash 相对应的数据结构）来提取一个行（第 2.4 节），在此情况下，MySQLdb 会注意到冲突的列名，并使用包含表名和列名的键值将它们放入字典中。因而，对于下面的语句，字典键值将为 name、title、states.name 和 price：

```
SELECT artist.name, painting.title, states.name, painting.price
FROM artist INNER JOIN painting INNER JOIN states
    ON artist.a_id = painting.a_id AND painting.state = states.abbrev
```

这表示列名不会丢失。然而遗憾的是，这里同样需要关注非唯一的列名。如果你试着只使用列名来引用列值，你将得不到你所期望的结果，因为所报告的名字包含了表名前缀。而如果你使用别名来保证列名的唯一性，此问题就不会发生，因为字典条目将保存你所指定的名字。

统计技术

Statistical Techniques

13.0 引言

Introduction

本章包括几个与基本统计技术相关的主题。其中的大部分内容以前面章节讨论过的技术为基础，比如第8章中讨论的摘要技术。本章的例子展示了适用于前面章节内容的更多方法，概括而言，本章讨论的主题包括：

- 数据描述技术，比如计算描述统计、产生频率分布、计数缺失值、以及计算最小平方回归或相关系数等。
- 随机方法，比如如何产生随机数并利用它们随机化一个行集合或者从行中随机地选择独立的条目。
- 等级分配。

统计学涵盖了广泛而又多样的主题，本章只是涉及其中的皮毛以简单地说明一些在MySQL中可能被用于统计分析的潜在领域。这里要注意，一些统计方法可以用不同方式定义（举例来说，你是根据自由度 n 还是 $n-1$ 来计算标准差）。由于这个原因，如果我对给定术语所使用的定义与你所喜欢的不同，在某种程度上你需要改编本章所展示的查询或算法。

你可以在本章的目录中找到所讨论的相关例子脚本，而例子中所用表的创建脚本可以在表目录中找到。

13.1 计算描述统计

Calculating Descriptive Statistics

问题

你需要通过计算一般的描述统计或概要统计特征以描述数据集。

解决方案

许多常见的描述统计，如均值和标准差，可以对你的数据使用聚集函数来获得。其他的，如中值或众数，可以根据计数查询来计算。

讨论

假设你有一个名为testscore的表，其中所含有的数据包括对象ID、年龄、性别和测试得分：

```
mysql> SELECT subject, age, sex, score FROM testscore ORDER BY subject;
+-----+-----+-----+-----+
| subject | age | sex | score |
+-----+-----+-----+-----+
|      1 |   5 |   M |     5 |
|      2 |   5 |   M |     4 |
|      3 |   5 |   F |     6 |
|      4 |   5 |   F |     7 |
|      5 |   6 |   M |     8 |
|      6 |   6 |   M |     9 |
|      7 |   6 |   F |     4 |
|      8 |   6 |   F |     6 |
|      9 |   7 |   M |     8 |
|     10 |   7 |   M |     6 |
|     11 |   7 |   F |     9 |
|     12 |   7 |   F |     7 |
|     13 |   8 |   M |     9 |
|     14 |   8 |   M |     6 |
|     15 |   8 |   F |     7 |
|     16 |   8 |   F |    10 |
|     17 |   9 |   M |     9 |
|     18 |   9 |   M |     7 |
|     19 |   9 |   F |    10 |
|    20 |   9 |   F |     9 |
+-----+-----+-----+-----+
```

分析观测数据集合的良好的第一步是产生能够整体上概括它们特征的统计描述值。常用的此类统计值包括：

- 数据数目、总和、范围（最大和最小值）。
- 集中趋势的度量，如平均数、中间数和众数。
- 变化的度量，如标准差和方差。

除了中间数和众数，所有这些可以通过调用聚集函数很容易地计算出来：

```
mysql> SELECT COUNT(score) AS n,
-> SUM(score) AS sum,
-> MIN(score) AS minimum,
-> MAX(score) AS maximum,
-> AVG(score) AS mean,
-> STDDEV_SAMP(score) AS 'std. dev.',
-> VAR_SAMP(score) AS 'variance'
-> FROM testscore;
+-----+-----+-----+-----+-----+-----+
| n | sum | minimum | maximum | mean | std. dev. | variance |
+-----+-----+-----+-----+-----+-----+
| 20 | 146 | 4 | 10 | 7.3000 | 1.8382 | 3.3789 |
+-----+-----+-----+-----+-----+-----+
```

`STDDEV_SAMP()` 和 `VAR_SAMP()` 函数产生的是采样方法而不是总体方法。也即是说，对于一个有 n 个值的集合，它们根据 $n-1$ 的自由度产生结果。如果你使用总体方法，如 `STDDEV_POP()` 和 `VAR_POP()` 函数，则基于 n 自由度。`STDDEV()` 和 `VARIANCE()` 与 `STDDEV_POP()` 和 `VAR_POP()` 是同义的。

标准差可以用来确定外围值，即那些远离均值的值。举个例子，选择与均值的差值大于 3 倍标准差的值，你可以进行如下操作：

```
SELECT @mean := AVG(score), @std := STDDEV_SAMP(score) FROM testscore;
SELECT score FROM testscore WHERE ABS(score-@mean) > @std * 3;
```

MySQL 没有计算值集合的众数或中间数的内建函数，但是你可以自己计算。众数是出现频率最高的值，为了找到它，需要对每个值计数以发现哪一个最为常见：

```
mysql> SELECT score, COUNT(score) AS frequency
-> FROM testscore GROUP BY score ORDER BY frequency DESC;
+-----+-----+
| score | frequency |
+-----+-----+
| 9 | 5 |
| 6 | 4 |
| 7 | 4 |
| 4 | 2 |
| 8 | 2 |
| 10 | 2 |
| 5 | 1 |
+-----+-----+
```

此例中 `score` 值的众数是 9。

一个有序集合的中间数可以如下计算（注1）：

- 如果值的数目是奇数，中间数就是顺序位于最中间的值。
- 如果值的数目是偶数，中间数是位于中间顺序的两个值的平均数。

根据此定义，可以使用下面的过程为存储在数据库中的观测数据确定中间数：

1. 产生计数观测值数目的查询。根据数目，你可以确定需要一个还是两个值来计算中间数，以及它们在有序值集合中的位置索引是多少。
2. 产生一个包含ORDER BY子句的查询以排序数据，并使用LIMIT子句以取得中间值（1个或2个）。
3. 如果中间值只有一个，那么它就是中间数。否则，中间数等于中间值的平均值。

举例来说，如果表中包含37（为奇数）个值，你需要使用下面的语句选择一个值以获取中间数：

```
SELECT score FROM t ORDER BY score LIMIT 18,1
```

如果该列包含了38（为偶数）个值，语句将变为：

```
SELECT score FROM t ORDER BY score LIMIT 18,2
```

然后你可以选择语句返回的值，计算它们的平均值获取中间数。

下面的Perl函数实现了中间数计算。它获取数据库句柄以及数据库名、表名和包含观测值集合的列名。然后产生语句以获取相关值并返回平均值：

```
sub median
{
    my ($dbh, $db_name, $tbl_name, $col_name) = @_;
    my ($count, $limit);

    $count = $dbh->selectrow_array ("SELECT COUNT($col_name)
                                    FROM $db_name.$tbl_name");
    return undef unless $count > 0;
    if ($count % 2 == 1) # 奇数；取中间值
    {
        $limit = sprintf ("LIMIT %d,1", ($count-1)/2);
    }
    else          # 偶数；取两个中间值的平均值
    {
        $limit = sprintf ("LIMIT %d,2", $count/2 - 1);
    }
}
```

注1：注意此处中间数的定义并不十分标准；它没有考虑到数据库中的中间值发生重复时该怎么办。

```

my $sth = $dbh->prepare ("SELECT $col_name
                           FROM $db_name.$tbl_name
                           ORDER BY $col_name $limit");
$sth->execute ();
my ($n, $sum) = (0, 0);
while (my $ref = $sth->fetchrow_arrayref())
{
    ++$n;
    $sum += $ref->[0];
}
return ($sum / $n);
}

```

前面的技术用于处理存储在数据库中的值集合。如果你正好已经将一个有序值集合存入数组@val中，你可以如下计算中间数：

```

if (@val == 0)           # 如果数组为空，则中间数未定义
{
    $median = undef;
}
elsif (@val % 2 == 1)    # 如果数组为奇数，则中间数为中间值
{
    $median = $val[(@val-1)/2];
}
else                     # 如果数组为偶数，则中间数为中间两个值的平均数
{
    $median = ($val[@val/2 - 1] + $val[@val/2]) / 2;
}

```

代码所处理的数组首元素下标被初始化为0；对于使用基数为1的语言，可以相应地调整算法。

13.2 分组描述统计

Per-Group Descriptive Statistics

问题

你需要为观测数据中的每个子组生成描述统计。

解决方案

使用聚集函数，同时利用GROUP BY子句将数据排列到合适的组中去。

讨论

13.1节显示了如何对testscore表中的整个scores集合进行描述统计计算。为了做到更加精确，你可以使用GROUP BY将数据分组并计算每组的统计量。举个例子，testscore表中的

对象是根据年龄和性别列出的，因此可以通过适当地应用GROUP BY子句根据年龄或性别（或两者）计算相似统计量。

下面显示了如何根据年龄计算：

```
mysql> SELECT age, COUNT(score) AS n,
-> SUM(score) AS sum,
-> MIN(score) AS minimum,
-> MAX(score) AS maximum,
-> AVG(score) AS mean,
-> STDDEV_SAMP(score) AS 'std. dev.',
-> VAR_SAMP(score) AS 'variance'
-> FROM testscore
-> GROUP BY age;
+-----+-----+-----+-----+-----+-----+-----+
| age | n   | sum  | minimum | maximum | mean    | std.dev. | variance |
+-----+-----+-----+-----+-----+-----+-----+
| 5   | 4   | 22   |        4 |        7 | 5.5000 | 1.2910  | 1.6667  |
| 6   | 4   | 27   |        4 |        9 | 6.7500 | 2.2174  | 4.9167  |
| 7   | 4   | 30   |        6 |        9 | 7.5000 | 1.2910  | 1.6667  |
| 8   | 4   | 32   |        6 |       10 | 8.0000 | 1.8257  | 3.3333  |
| 9   | 4   | 35   |        7 |       10 | 8.7500 | 1.2583  | 1.5833  |
+-----+-----+-----+-----+-----+-----+-----+
```

根据性别：

```
mysql> SELECT sex, COUNT(score) AS n,
-> SUM(score) AS sum,
-> MIN(score) AS minimum,
-> MAX(score) AS maximum,
-> AVG(score) AS mean,
-> STDDEV_SAMP(score) AS 'std. dev.',
-> VAR_SAMP(score) AS 'variance'
-> FROM testscore
-> GROUP BY sex;
+-----+-----+-----+-----+-----+-----+-----+
| sex | n   | sum  | minimum | maximum | mean    | std.dev. | variance |
+-----+-----+-----+-----+-----+-----+-----+
| M   | 10  | 71   |        4 |        9 | 7.1000 | 1.7920  | 3.2111  |
| F   | 10  | 75   |        4 |       10 | 7.5000 | 1.9579  | 3.8333  |
+-----+-----+-----+-----+-----+-----+-----+
```

根据性别和年龄：

```
mysql> SELECT age, sex, COUNT(score) AS n,
-> SUM(score) AS sum,
-> MIN(score) AS minimum,
-> MAX(score) AS maximum,
-> AVG(score) AS mean,
-> STDDEV_SAMP(score) AS 'std. dev.',
-> VAR_SAMP(score) AS 'variance'
-> FROM testscore
-> GROUP BY age, sex;
+-----+-----+-----+-----+-----+-----+-----+
| age | sex | n   | sum  | minimum | maximum | mean    | std.dev. | variance |
+-----+-----+-----+-----+-----+-----+-----+
| 5   | M   | 2   | 9   |        4 |        5 | 4.5000 | 0.7071 | 0.5000 |
+-----+-----+-----+-----+-----+-----+-----+
```

	5		F		2		13		6		7		6.5000		0.7071		0.5000
	6		M		2		17		8		9		8.5000		0.7071		0.5000
	6		F		2		10		4		6		5.0000		1.4142		2.0000
	7		M		2		14		6		8		7.0000		1.4142		2.0000
	7		F		2		16		7		9		8.0000		1.4142		2.0000
	8		M		2		15		6		9		7.5000		2.1213		4.5000
	8		F		2		17		7		10		8.5000		2.1213		4.5000
	9		M		2		16		7		9		8.0000		1.4142		2.0000
	9		F		2		19		9		10		9.5000		0.7071		0.5000

13.3 产生频率分布

Generating Frequency Distributions

问题

你需要知道表中每个值的出现频率。

解决方案

推导出频率分布以概括数据集的内容。

讨论

一个分组摘要技术的常见应用为产生每个值出现次数的分析，称之为频率分布。对于 testscore 表，频率分布如下：

```
mysql> SELECT score, COUNT(score) AS occurrence
-> FROM testscore GROUP BY score;
+-----+-----+
| score | occurrence |
+-----+-----+
| 4     | 2      |
| 5     | 1      |
| 6     | 4      |
| 7     | 4      |
| 8     | 2      |
| 9     | 5      |
| 10    | 2      |
+-----+-----+
```

如果你将结果表示为百分数而不是原始计数，则产生了相对频率分布。为了分解一个数据集并显示每个计数占总数的百分比，可以使用一个查询获得数据的总数以计算每组的百分比：

```
mysql> SELECT @n := COUNT(score) FROM testscore;
mysql> SELECT score, (COUNT(score)*100)/@n AS percent
-> FROM testscore GROUP BY score;
+-----+-----+
| score | percent |
+-----+-----+
```

	4		10	
	5		5	
	6		20	
	7		20	
	8		10	
	9		25	
	10		10	
+	-	-	-	+

上面显示的分布概括了不同score值出现的次数。但是，如果数据集合包含的不同值数目巨大，并且你需要频率分布只显示一小部分分类，你可能需要将值分类，并对每个类产生计数。“分类”技术见8.12节中的讨论。

频率分布的典型用途是输出结果以供图表程序使用。如果没有此程序，你可以使用MySQL来产生一个简单的图作为分布的可视化表现。举个例子，为了显示测试得分计数的ASCII条状图，可以将计数转换为*字符串：

```
mysql> SELECT score, REPEAT('*',COUNT(score)) AS occurrences
-> FROM testscore GROUP BY score;
+-----+-----+
| score | occurrences |
+-----+-----+
|     4 |   **
|     5 |   *
|     6 | *****
|     7 | *****
|     8 |   **
|     9 | ***** 
|    10 |   **
+-----+-----+
```

如果要将相对频率分布以图示方式显示，则使用百分率值：

```
mysql> SELECT @n := COUNT(score) FROM testscore;
mysql> SELECT score, REPEAT('*',(COUNT(score)*100)/@n) AS percent
-> FROM testscore GROUP BY score;
+-----+-----+
| score | percent |
+-----+-----+
|     4 | ***** 
|     5 | **** 
|     6 | ***** 
|     7 | ***** 
|     8 | **** 
|     9 | ***** 
|    10 | **** 
+-----+-----+
```

显然，ASCII图的方法相当简略，但它能够快速得到数据分布图，并且不需要其他的工具。

如果你是一个种类范围产生频率分布，其中一部分类别并没有被包含在你的数据中，那么缺失的类别将不会出现在输出中。为了强制显示每个类别，需要使用引用表和LEFT JOIN（12.8节中讨论了该技术）。对于testscore表，可能的score范围为0到10，因此引用表应该包含其中的每个值：

```
mysql> CREATE TABLE ref (score INT);
mysql> INSERT INTO ref (score)
-> VALUES(0),(1),(2),(3),(4),(5),(6),(7),(8),(9),(10);
```

然后将引用表与testscore表相连接以产生频率分布：

```
mysql> SELECT ref.score, COUNT(testscore.score) AS occurrences
-> FROM ref LEFT JOIN testscore ON ref.score = testscore.score
-> GROUP BY ref.score;
+-----+-----+
| score | occurrences |
+-----+-----+
| 0     | 0          |
| 1     | 0          |
| 2     | 0          |
| 3     | 0          |
| 4     | 2          |
| 5     | 1          |
| 6     | 4          |
| 7     | 4          |
| 8     | 2          |
| 9     | 5          |
| 10    | 2          |
+-----+-----+
```

该分布包括了score为0到3的行，而它们没有在前面显示的频率分布中出现。

同样的原则适用于相对频率分布：

```
mysql> SELECT @n := COUNT(score) FROM testscore;
mysql> SELECT ref.score, (COUNT(testscore.score)*100)/@n AS percent
-> FROM ref LEFT JOIN testscore ON ref.score = testscore.score
-> GROUP BY ref.score;
+-----+-----+
| score | percent |
+-----+-----+
| 0     | 0        |
| 1     | 0        |
| 2     | 0        |
| 3     | 0        |
| 4     | 10       |
| 5     | 5        |
| 6     | 20       |
| 7     | 20       |
| 8     | 10       |
| 9     | 25       |
| 10    | 10       |
+-----+-----+
```

13.4 计数缺失值

Counting Missing Values

问题

数据集合并不完整，你需要找出缺失了多少值。

解决方案

计数集合中空值出现的次数。

讨论

数据集合中造成值的缺失可能有多种原因：如一个测试没有被有效执行，测试期间发生某些故障导致无效数据等。你可以在数据集合中使用NULL值来表示它们以提示它们为丢失的或者无效的值，并使用摘要语句来刻画数据集合的完整性。

如果表t包含的值可以用一维方式摘要，则可以使用简单的查询来表示缺失值。假设t如下所示：

```
mysql> SELECT subject, score FROM t ORDER BY subject;
+-----+-----+
| subject | score |
+-----+-----+
|      1 |    38 |
|      2 |   NULL |
|      3 |    47 |
|      4 |   NULL |
|      5 |    37 |
|      6 |    45 |
|      7 |    54 |
|      8 |   NULL |
|      9 |    40 |
|     10 |    49 |
+-----+-----+
```

COUNT(*)对所有行计数，COUNT(score)仅仅计数没有缺少的成绩。这两个数值之间的差别就是缺少的成绩的数目，而且与总数相关的差别还可以提供缺少的成绩的百分比。计算过程如下所示：

```
mysql> SELECT COUNT(*) AS 'n (total)',
-> COUNT(score) AS 'n (nonmissing)',
-> COUNT(*) - COUNT(score) AS 'n (missing)',
-> ((COUNT(*) - COUNT(score)) * 100) / COUNT(*) AS '% missing'
-> FROM t;
+-----+-----+-----+-----+
| n (total) | n (nonmissing) | n (missing) | % missing |
+-----+-----+-----+-----+
|      10 |            7 |          3 |    30.00 |
+-----+-----+-----+-----+
```

与其他计数的方法不同，计数NULL值的替代方法是直接使用SUM(ISNULL(score))。ISNULL()函数在参数为NULL时返回1，否则返回0：

```
mysql> SELECT COUNT(*) AS 'n (total)',  
-> COUNT(score) AS 'n (nonmissing)',  
-> SUM(ISNULL(score)) AS 'n (missing)',  
-> (SUM(ISNULL(score)) * 100) / COUNT(*) AS '% missing'  
-> FROM t;  
+-----+-----+-----+-----+  
| n (total) | n (nonmissing) | n (missing) | % missing |  
+-----+-----+-----+-----+  
|      10 |          7 |          3 |     30.00 |  
+-----+-----+-----+-----+
```

如果值已经被分组，NULL值的出现次数可以根据每组基数来估计。假设t所包含的对象得分按照两个因子A和B的条件来分布，则其中的每个条件具有两个级别：

```
mysql> SELECT subject, A, B, score FROM t ORDER BY subject;  
+-----+-----+-----+-----+  
| subject | A    | B    | score |  
+-----+-----+-----+-----+  
|      1 | 1   | 1   |    18 |  
|      2 | 1   | 1   |   NULL |  
|      3 | 1   | 1   |    23 |  
|      4 | 1   | 1   |    24 |  
|      5 | 1   | 2   |    17 |  
|      6 | 1   | 2   |    23 |  
|      7 | 1   | 2   |    29 |  
|      8 | 1   | 2   |    32 |  
|      9 | 2   | 1   |    17 |  
|     10 | 2   | 1   |   NULL |  
|     11 | 2   | 1   |   NULL |  
|     12 | 2   | 1   |    25 |  
|     13 | 2   | 2   |   NULL |  
|     14 | 2   | 2   |    33 |  
|     15 | 2   | 2   |    34 |  
|     16 | 2   | 2   |    37 |  
+-----+-----+-----+-----+
```

在此情况下，查询使用GROUP BY子句来为每个条件组合产生摘要：

```
mysql> SELECT A, B, COUNT(*) AS 'n (total)',  
-> COUNT(score) AS 'n (nonmissing)',  
-> COUNT(*) - COUNT(score) AS 'n (missing)',  
-> ((COUNT(*) - COUNT(score)) * 100) / COUNT(*) AS '% missing'  
-> FROM t  
-> GROUP BY A, B;  
+-----+-----+-----+-----+-----+-----+  
| A    | B    | n(total) | n (nonmissing) | n(missing) | % missing |  
+-----+-----+-----+-----+-----+-----+  
| 1    | 1    |      4 |            3 |          1 |    25.00 |  
| 1    | 2    |      4 |            4 |          0 |    0.00 |  
+-----+-----+-----+-----+-----+-----+
```

	2		1		4	
	2		2		4	

2		2		50.00
1		1		25.00



13.5 计算线性回归和相关系数

Calculating Linear Regressions or Correlation Coefficients

问题

你需要为两个变量计算最小平方回归线或表示它们之间关系强弱的相关系数。

解决方案

运用摘要函数计算所需要的项。

讨论

当变量X和Y的数据值被存储在数据库中，它们的最小平方回归可以很容易地使用聚集函数计算，同样对于相关系数也是成立的。两种计算实际上相当类似，许多执行计算的项在两个过程中是相同的。

假设你需要使用在testscore表中的观测数据，为age和testscore值计算最小平方回归：

```
mysql> SELECT age, score FROM testscore;
+-----+-----+
| age | score |
+-----+-----+
| 5   | 5    |
| 5   | 4    |
| 5   | 6    |
| 5   | 7    |
| 6   | 8    |
| 6   | 9    |
| 6   | 4    |
| 6   | 6    |
| 7   | 8    |
| 7   | 6    |
| 7   | 9    |
| 7   | 7    |
| 8   | 9    |
| 8   | 6    |
| 8   | 7    |
| 8   | 10   |
| 9   | 9    |
| 9   | 7    |
| 9   | 10   |
| 9   | 9    |
+-----+-----+
```

下面的等式表示了回归线，其中a和b分别是截距和斜率：

$$Y = bX + a$$

使用 x 表示age, y 表示score, 先计算回归方程式所需要的项, 包括数据的均值、总和、每个变量的平方和以及每个变量乘积的和 (注2):

```

mysql> SELECT
-> @n := COUNT(score) AS N,
-> @meanX := AVG(age) AS 'X mean',
-> @sumX := SUM(age) AS 'X sum',
-> @sumXX := SUM(age*age) AS 'X sum of squares',
-> @meanY := AVG(score) AS 'Y mean',
-> @sumY := SUM(score) AS 'Y sum',
-> @sumYY := SUM(score*score) AS 'Y sum of squares',
-> @sumXY := SUM(age*score) AS 'X*Y sum'
-> FROM testscore\G
***** 1. row *****
          N: 20
          X mean: 7.000000000
          X sum: 140
X sum of squares: 1020
          Y mean: 7.300000000
          Y sum: 146
Y sum of squares: 1130
          X*Y sum: 1053

```

利用这些项来计算回归的斜率与截距，方法如下：

然后回归等式为：

注2：通过查询任何的标准静态文本来查看术语出自何处。

为了计算相关系数，许多同样的项也被用到：

```
mysql> SELECT
-> (@n*@sumXY - @sumX*@sumY)
-> / SQRT((@n*@sumXX - @sumX*@sumX) * (@n*@sumYY - @sumY*@sumY))
-> AS correlation;
+-----+
| correlation |
+-----+
| 0.61173620442199 |
+-----+
```

你可以在任何翻译的标准统计课本里查到这些术语。

13.6 生成随机数

Generating Random Numbers

问题

你需要产生一些随机数。

解决方案

调用MySQL的RAND()函数

讨论

MySQL具有RAND()函数，能够产生0到1之间的随机数：

```
mysql> SELECT RAND(), RAND(), RAND();
+-----+-----+-----+
| RAND() | RAND() | RAND() |
+-----+-----+-----+
| 0.18768198246852 | 0.0052350517411111 | 0.46312934203365 |
+-----+-----+-----+
```

当提供一个整数参数调用时，RAND()使用该值作为随机数产生器的种子。你可以使用这种方式为查询结果列产生可重复的数字系列。下面的例子显示了不带参数的RAND()每次查询产生了不同的值列，而RAND(N)则生成可重复的列：

```
mysql> SELECT i, RAND(), RAND(10), RAND(20) FROM t;
+-----+-----+-----+-----+
| i | RAND() | RAND(10) | RAND(20) |
+-----+-----+-----+
| 1 | 0.60170396939079 | 0.65705152196535 | 0.15888261251047 |
| 2 | 0.10435410784963 | 0.12820613023658 | 0.63553050033332 |
| 3 | 0.71665866180943 | 0.66987611602049 | 0.70100469486881 |
| 4 | 0.27023101623192 | 0.96476222012636 | 0.59843200407776 |
+-----+-----+-----+-----+
mysql> SELECT i, RAND(), RAND(10), RAND(20) FROM t;
+-----+-----+-----+-----+
| i | RAND() | RAND(10) | RAND(20) |
+-----+-----+-----+-----+
```

		1 0.55794027034001 0.65705152196535 0.15888261251047	
		2 0.22995210460383 0.12820613023658 0.63553050033332	
		3 0.47593974273274 0.66987611602049 0.70100469486881	
		4 0.68984243058585 0.96476222012636 0.59843200407776	

如果你需要随机产生RAND()函数的种子，可以从一个无序源选取，比如即时时间戳或者连接标志符，单独使用或联合使用都可以：

```
RAND(UNIX_TIMESTAMP())
RAND(CONNECTION_ID())
RAND(UNIX_TIMESTAMP() + CONNECTION_ID())
```

RAND()产生的结果随机程度怎样呢

RAND()函数有没有产生平均分布的数字？你可以使用下面的Python脚本(rand_test.py,位于本节stats目录下)进行检查，它使用了RAND()生成随机数，然后以1为单位分类，从中建立频率分布，这提供了一种评估这些值是否平均分布的方法：

```
#!/usr/bin/python
# rand_test.pl - 创建一个随机的频数分布
# 这可以提供一个RAND()函数的随机测试
# 方法：产生一个0到1.0之间随机数,
# 同时计数在间隔为1的时间片里发生了多少次

import MySQLdb
import Cookbook

npicks = 1000 # 提取次数的值
bucket = [0] * 10 # 每个时间间隔片计数的存储桶

conn = Cookbook.connect ()
cursor = conn.cursor ()

for i in range (0, npicks):
    cursor.execute ("SELECT RAND()")
    (val,) = cursor.fetchone ()
    slot = int (val * 10)
    if slot > 9:
        slot = 9 # 将1.0放入最后一个slot
    bucket[slot] = bucket[slot] + 1

cursor.close ()
conn.close ()

# 打印得到的频数分布

for slot in range (0, 9):
    print "%2d %d" % (slot+1, bucket[slot])
```

stats目录还包含其他语言的相应脚本。

当然，如果条件允许，使用其他的种子源可能更好。举个例子，如果你的系统中具有`/dev/random`或`/dev/urandom`设备，你可以读入该设备并使用它来产生一个作为`RAND()`函数种子的值。

13.7 随机化行集合

Randomizing a Set of Rows

问题

你需要随机化一个行或值的集合。

解决方案

使用`ORDER BY RAND()`。

讨论

MySQL的`RAND()`函数可以被用来对查询所返回的行进行随机排序。稍显荒谬的是，这种随机化是通过在查询中增加`ORDER BY`实现的。此技术大致上与电子数据表的随机化方法相同。假设你在电子数据表中有一个值集合，如下所示：

Patrick
Penelope
Pertinax
Polly

为了以随机顺序放置这些值，应首先增加一个包含随机选择数的列：

Patrick	.73
Penelope	.37
Pertinax	.16
Polly	.48

然后根据随机数的值排序这些行：

Pertinax	.16
Penelope	.37
Polly	.48
Patrick	.73

通过这种方式，由于对随机数排序的效果等同于随机化与它们相关的值，因此原始值以随机次序被排列。如果要重新随机化这些值，可以选择另一个随机数集合，并据此对行再次进行排序。

在MySQL中，类似地，你也可以为查询结果关联一个随机数集合，然后根据这些数排列结果。执行此项工作需要增加`ORDER BY RAND()`子句：

```
mysql> SELECT name FROM t ORDER BY RAND();
+-----+
| name |
+-----+
| Pertinax |
| Penelope |
| Patrick |
| Polly    |
+-----+
mysql> SELECT name FROM t ORDER BY RAND();
+-----+
| name |
+-----+
| Patrick |
| Pertinax |
| Penelope |
| Polly    |
+-----+
```

随机化一个行集合可以应用于任何选择而非置换（从一个条目集合中选择每一个条目，直到选完为止）的场景。下面是一些应用实例：

- 确定某个活动参与者的开始顺序。先在表中列出所有的参与者，然后以随机顺序选取他们。
- 先在表中列出所有的赛道，然后为参加赛跑的选手以随机顺序选择分配赛道。
- 为一个测试问题集合选择提问的顺序。
- 要打乱一副牌。可将每张牌对应到表中的一行，并通过随机选择这些行来打乱这幅牌。这样一个接一个地处理它们，直到这幅牌被整理完毕。

为了使用最后一个例子作为说明，让我们实现洗牌算法。洗牌和发牌属于随机选择且不归还：每张牌只被处理一次，直到整副牌被处理完毕时，该牌被重新洗好并使新的发牌顺序随机化。在程序中，此任务可以在MySQL中使用具有52行的表deck来执行，假设一个牌集合包括13种面值与4种花色：

1. 选择整个表，并将之存储到数组中。
2. 每当需要一张牌时，从数组中取出下一个元素。
3. 当数组被取空时，所有的牌都已被处理，则该表被“重新洗牌”并产生了新的顺序。

如果你是通过手动编写INSERT语句插入52个牌记录的话，那么建立纸牌表真是一项很繁琐的任务。实际上，牌的内容可以在程序中通过生成每个面值-花色对，以组合方式更容易地

生成。下面是一些PHP代码，产生了包含花色和面值列的deck表，然后使用嵌套循环生成值对，并提供给INSERT语句填充该表：

```
$result = & $conn->query ("CREATE TABLE deck
(
    face ENUM('A', 'K', 'Q', 'J', '10', '9', '8',
             '7', '6', '5', '4', '3', '2') NOT NULL,
    suit ENUM('hearts', 'diamonds', 'clubs', 'spades') NOT NULL
)");
if (PEAR::isError ($result))
    die ("Cannot issue CREATE TABLE statement\n");

$face_array = array ("A", "K", "Q", "J", "10", "9", "8",
                     "7", "6", "5", "4", "3", "2");
$suit_array = array ("hearts", "diamonds", "clubs", "spades");

# 为每一个suit和face的组合将一张"牌"插入扑克中

$stmt = & $conn->prepare ("INSERT INTO deck (face,suit) VALUES(?,?)");
if (PEAR::isError ($stmt))
    die ("Cannot insert card into deck\n");
foreach ($face_array as $index => $face)
{
    foreach ($suit_array as $index2 => $suit)
    {
        $result = & $conn->execute ($stmt, array ($face, $suit));
        if (PEAR::isError ($result))
            die ("Cannot insert card into deck\n");
    }
}
```

洗牌操作由下面语句完成：

```
SELECT face, suit FROM deck ORDER BY RAND();
```

为了在脚本中实现此操作并将结果存储到数组中，可以通过编写shuffle_deck()函数以提交查询并以数组方式返回结果值（再次显示在PHP中）：

```
function shuffle_deck ($conn)
{
    $result = & $conn->query ("SELECT face, suit FROM deck ORDER BY RAND()");
    if (PEAR::isError ($result))
        die ("Cannot retrieve cards from deck\n");
    $card = array ();
    while ($obj = & $result->fetchRow (DB_FETCHMODE_OBJECT))
        $card[] = $obj;    # 将牌的记录加入到$card列表的最后
    $result->free ();
    return ($card);
}
```

通过保持一个范围为0到51的计数器来指示应该选择哪张牌。当计数到达52时，该牌已被用完，并需要重新洗牌。

13.8 从行集合中随机选择条目

Selecting Random Items from a Set of Rows

问题

你需要从值集合中随机地选取一个或多个条目。

解决方案

随机排列这些值，然后选择第一个（或者如果你需要多个，可以选择前几个）。

讨论

当一个条目集合被存储在MySQL中时，你可以以下列方式随机选出一个：

1. 使用如13.7节所述的ORDER BY RAND()从处于随机顺序的集合中选择条目。
2. 在查询中增加LIMIT 1以获取第一个条目。

举例来说，一个掷骰子的简单模拟可以通过创建骰子表，其中行的值为1到6，与一个六面体骰子相对应，然后从中随机地选择行：

```
mysql> SELECT n FROM die ORDER BY RAND() LIMIT 1;
+---+
| n |
+---+
|   6 |
+---+
mysql> SELECT n FROM die ORDER BY RAND() LIMIT 1;
+---+
| n |
+---+
|   4 |
+---+
mysql> SELECT n FROM die ORDER BY RAND() LIMIT 1;
+---+
| n |
+---+
|   5 |
+---+
mysql> SELECT n FROM die ORDER BY RAND() LIMIT 1;
+---+
| n |
+---+
|   4 |
+---+
```

当你重复这项操作时，你从集合中以随机顺序选取了这些条目。这是选择并归还的一种形式：即一个条目从条目池中被选出后，又归还到池中以供下一次挑选。因为这些条目被放

回原处，当使用此方式连续地进行选择时，可能会多次选择同一个条目。另一些选择并归还的例子包括：

- 在Web页面上选择显示一个广告条；
- 在“当日报价”应用中选取一行；
- 在“任意选择一张牌”的魔术表演中，其中每次选择需要从满副牌开始。

如果你想要选择多于一个的条目，可以修改LIMIT的参数。举个例子，为了从包含竞赛选手信息的drawing表中随机地选择5个获胜选手，可以与LIMIT一起使用RAND()：

```
SELECT * FROM drawing ORDER BY RAND() LIMIT 5;
```

如果某个表包含一个从1至n并且顺序完整的列，那么当你从中选取单个行时会产生特殊情况。由此，可以选择位于该范围内的随机数，然后再选择与之相匹配的行，如此避免了在整个表中执行ORDER BY操作：

```
SET @id = FLOOR(RAND()*n)+1;
SELECT ... FROM tbl_name WHERE id = @id;
```

当表的容量增加时，这比使用ORDER BY RAND() LIMIT 1要快得多。

13.9 分配等级

Assigning Ranks

问题

你需要为集合中的值分配等级。

解决方案

确定一种分级方法，然后使用该方法将值放入所期望的顺序。

讨论

一些统计测试需要为值分配等级。本节描述三种分级方法并显示如何通过用户定义变量来实现每一种方法。例子假设表t包含下面需要被分级的分数，其降序排列如下：

```
mysql> SELECT score FROM t ORDER BY score DESC;
+-----+
| score |
+-----+
|      5 |
|      4 |
|      4 |
|      3 |
|      2 |
```

```
|      2 |
|      2 |
|      1 |
+-----+
```

一种简单的分级方法是在数值的有序集中为每个值赋予它的行号，为了产生此种分级，需要知道行号并使用它来作为当前的级别：

```
mysql> SET @rownum := 0;
mysql> SELECT @rownum := @rownum + 1 AS rank, score
-> FROM t ORDER BY score DESC;
+-----+-----+
| rank | score |
+-----+-----+
|    1 |     5 |
|    2 |     4 |
|    3 |     4 |
|    4 |     3 |
|    5 |     2 |
|    6 |     2 |
|    7 |     2 |
|    8 |     1 |
+-----+-----+
```

这种分级方式并没有考虑平分（即实例的值相同）的可能性。第二种分级方法重视了此点，只有当值发生变化时，级别才会增加：

```
mysql> SET @rank = 0, @prev_val = NULL;
mysql> SELECT @rank := IF(@prev_val=score,@rank,@rank+1) AS rank,
-> @prev_val := score AS score
-> FROM t ORDER BY score DESC;
+-----+-----+
| rank | score |
+-----+-----+
|    1 |     5 |
|    2 |     4 |
|    2 |     4 |
|    3 |     3 |
|    4 |     2 |
|    4 |     2 |
|    4 |     2 |
|    5 |     1 |
+-----+-----+
```

第三种分级方法为上面两种方法的结合。它根据行号来对值进行分级，除了平分发生时以外。在此情况下，平分值每次获得与第一个值的行号相等的级别。为了实现此方法，需要了解行号和前面的值，当值发生变化时，将等级定为当前行号：

```
mysql> SET @rownum = 0, @rank = 0, @prev_val = NULL;
mysql> SELECT @rownum := @rownum + 1 AS row,
-> @rank := IF(@prev_val!=score,@rownum,@rank) AS rank,
```

```

-> @prev_val := score AS score
-> FROM t ORDER BY score DESC;
+-----+-----+
| row | rank | score |
+-----+-----+
| 1   | 1    | 5   |
| 2   | 2    | 4   |
| 3   | 2    | 4   |
| 4   | 4    | 3   |
| 5   | 5    | 2   |
| 6   | 5    | 2   |
| 7   | 5    | 2   |
| 8   | 8    | 1   |
+-----+-----+

```

通过程序也可以容易地分配等级。举个例子，下面的Ruby代码片段使用了第三种分级方式对t表中的分值进行了分级：

```

dbh.execute("SELECT score FROM t ORDER BY score DESC") do |sth|
  rownum = 0
  rank = 0
  prev_score = nil
  puts "Row\tRank\tScore\n"
  sth.fetch do |row|
    score = row[0]
    rownum += 1
    rank = rownum if rownum == 1 || prev_score != score
    prev_score = score
    puts "#{rownum}\t#{rank}\t#{score}"
  end
end

```

第三种分级方式在统计方法领域之外也得到广泛使用，回忆3.16节中的内容，我们使用的表al_winner包含了美国棒球联盟中在2001赛季赢得的比赛超过15场的投手：

```

mysql> SELECT name, wins FROM al_winner ORDER BY wins DESC, name;
+-----+-----+
| name          | wins |
+-----+-----+
| Mulder, Mark | 21  |
| Clemens, Roger | 20  |
| Moyer, Jamie | 20  |
| Garcia, Freddy | 18  |
| Hudson, Tim | 18  |
| Abbott, Paul | 17  |
| Mays, Joe | 17  |
| Mussina, Mike | 17  |
| Sabathia, C.C. | 17  |
| Zito, Barry | 17  |
| Buehrle, Mark | 16  |
| Milton, Eric | 15  |
| Pettitte, Andy | 15  |
| Radke, Brad | 15  |
| Sele, Aaron | 15  |
+-----+-----+

```

这些投手可以根据第三种方法进行分级，如下所示：

```
mysql> SET @rownum = 0, @rank = 0, @prev_val = NULL;
mysql> SELECT @rownum := @rownum + 1 AS row,
   -> @rank := IF(@prev_val!=wins,@rownum,@rank) AS rank,
   -> name,
   -> @prev_val := wins AS wins
   -> FROM al_winner ORDER BY wins DESC;
+-----+-----+-----+-----+
| row | rank | name          | wins |
+-----+-----+-----+-----+
| 1   | 1   | Mulder, Mark | 21  |
| 2   | 2   | Clemens, Roger | 20  |
| 3   | 2   | Moyer, Jamie | 20  |
| 4   | 4   | Garcia, Freddy | 18  |
| 5   | 4   | Hudson, Tim | 18  |
| 6   | 6   | Zito, Barry | 17  |
| 7   | 6   | Sabathia, C.C. | 17  |
| 8   | 6   | Mussina, Mike | 17  |
| 9   | 6   | Mays, Joe | 17  |
| 10  | 6   | Abbott, Paul | 17  |
| 11  | 11  | Buehrle, Mark | 16  |
| 12  | 12  | Milton, Eric | 15  |
| 13  | 12  | Pettitte, Andy | 15  |
| 14  | 12  | Radke, Brad | 15  |
| 15  | 12  | Sele, Aaron | 15  |
+-----+-----+-----+-----+
```


处理重复项

Handling Duplicates

14.0 引言

Introduction

有时候表或结果集会含有重复行，在某些情况下这是可以接受的。举例来说，如果你发起了一个Web投票，需要为每张选票记录下日期和客户端IP，那么在这里重复行是允许存在的，因为许多Internet服务的客户可能是通过同一个代理主机进行通信的，这样可能会造成大量选票来源于同一个IP地址。而在另一些情况下，重复项是不可接受的，你需要采取措施避免其存在。处理重复行的操作包括以下几种方式：

- 在一开始就禁止在数据表中产生重复项。如果表中的每一行都试图代表一个独立的实体（如人、目录表项或者一个实验中的某次观测数据等），那么重复项会在使用上带来巨大困难。如果重复项存在，那么从逻辑意义上明白地区分每一行变成不可能，因此最好确保永远不会出现重复。
- 对重复项计数以确定它们是否存在以及严重到何种程度。
- 通过鉴别重复值（或包含它们的行）来发现哪些是重复项以及它们是在何处出现的。
- 消除重复项以确保每一行都是唯一的。这包括从表中删除多余的行，只留下唯一一行。或者从表中选择一个不含重复项的结果集。举例来说，如果需要显示你的顾客所在州的列表，你大概不会将所有顾客记录中的州名都列在表中（包含了大量的重复项）。该列表将每个州名只显示一次就足够了并且更容易理解。

有几种方式可以帮你处理重复行，你可以根据你要实现的目标来选择它们：

- 创建包含主键或唯一索引的表以防止重复项被添加至表中。MySQL使用索引作为约束以强制表中每一行在索引列或列联合上都包含一个唯一键值。

- 与唯一索引相结合，`INSERT IGNORE`和`REPLACE`语句使你能够从容地处理重复行插入而不会产生错误。对于桶载入操作，同样可以在`LOAD DATA`语句中使用`IGNORE`或`REPLACE`修饰符达到此目的。
- 如果你需要确定表中是否含有重复行，使用`GROUP BY`将行分类划分到组中，再使用`COUNT()`查看每组中含有多少行。这些技术在第8章中生成摘要的相关部分阐述，但是它们对于重复计数和鉴别也是有用的。计数摘要本质上是一种根据类别将值聚合以确定每类发生频率的操作。
- `SELECT DISTINCT` 用于在结果集中删除重复行（可参见3.8节获得更多信息）。对于已存在重复行的表，可以通过增加唯一索引来删除它们。如果你确定表中含有 n 个同样的行，你可以使用`DELETE...LIMIT`从这个特殊的行集合中删除 $n-1$ 个实例。

本章示例的相关脚本在`recipes`发行包的`dups`目录下，而对于创建所使用的表的脚本，可以参见`tables`目录。

14.1 防止在表中发生重复

Preventing Duplicates from Occurring in a Table

问题

你需要保证某个表从不会包含重复项。

解决方案

使用`PRIMARY KEY`或者`UNIQUE`索引。

讨论

为了确保表中每一行是唯一的，那么每行中某些列或者列的联合必须包含唯一值。当此要求被满足时，你可以通过唯一性标志清楚地识别表中任意一行。而为了保证表具有此性质，你在创建表时，必须在表结构中包含`PRIMARY KEY`或者`UNIQUE`索引。下表不含有这些限制，因此它是允许有重复行的：

```
CREATE TABLE person
(
    last_name  CHAR(20),
    first_name CHAR(20),
```

```
address    CHAR(40)
);
```

为了防止在表中产生多个姓和名都一样的行，可以在表定义中加上PRIMARY KEY。当你这么做时，被PRIMARY KEY标记的列必须是非空的，因为PRIMARY KEY不允许有空值：

```
CREATE TABLE person
(
    last_name    CHAR(20) NOT NULL,
    first_name   CHAR(20) NOT NULL,
    address      CHAR(40),
    PRIMARY KEY (last_name, first_name)
);
```

如果一个表具有唯一索引，当你向该表中插入一个与已存在行的索引列发生重复的记录时，通常会产生一个错误。14.2节讨论了如何处理这些错误或修改MySQL的重复处理机制。

另一种强制唯一性的方法是为表添加UNIQUE索引而不是PRIMARY KEY。这两种索引方式是类似的，有一点不同在于建立UNIQUE索引的列可以允许NULL值。对于这个person表来说，很可能姓和名都必须填写，如果这样，你可以将这些列声明为NOT NULL，下面的表定义与前一个是等价的：

```
CREATE TABLE person
(
    last_name    CHAR(20) NOT NULL,
    first_name   CHAR(20) NOT NULL,
    address      CHAR(40),
    UNIQUE (last_name, first_name)
);
```

如果一个UNIQUE索引可以允许NULL值，则NULL值具有特殊性是因为它可以出现多次。这样做的理由在于确定一个未知值是否与另一个未知值相等是不可能的，因此多个未知值是允许共存的。（一个例外是BDB表只允许具有UNIQUE索引的列至多有一个NULL值。）

当然你或许会希望person表能够反映现实世界，即有时候不同人会具有相同的名字。在这种情况下，你不能为姓名列建立唯一性索引，因为重复的名字是允许出现的。作为替代，必须赋予每个person记录唯一性标志符，即区分行与行的值。在MySQL中，一般通过使用AUTO_INCREMENT列来实现。

```
CREATE TABLE person
(
    id          INT UNSIGNED NOT NULL AUTO_INCREMENT,
    last_name   CHAR(20),
```

```
    first_name  CHAR(20),
    address     CHAR(40),
    PRIMARY KEY (id)
);
```

在此情况下，当你创建一个id值为NULL的行时，MySQL将自动为该列赋予一个唯一的ID。另一种可能的做法是指派外部标志符并使用它们作为唯一的键值。举例来说，某个国家的公民可能具有唯一的纳税ID号码，那么这些号码就可以用来作为唯一性索引：

```
CREATE TABLE person
(
    tax_id       INT UNSIGNED NOT NULL,
    last_name    CHAR(20),
    first_name   CHAR(20),
    address      CHAR(40),
    PRIMARY KEY  (tax_id)
);
```

参考

如果已有的表已经包含你想去掉的行的副本，请参考第14.4节及第11章中对AUTO_INCREMENT列的进一步讨论。

14.2 处理向表中装载行时出现的重复错误

Dealing with Duplicates When Loading Rows into a Table

问题

你已经创建了一个表，并且在需要索引的列或列的联合上建立了唯一索引以防止重复值的出现。但是这将导致在你试图向表中插入一个重复行时产生错误，并且你希望可以不用强制处理这些错误。

解决方案

一种简单的方法就是忽略掉这些错误。另一种方式是使用INSERT IGNORE、REPLACE或INSERT ... ON DUPLICATE KEY UPDATE语句，这些语句修改了MySQL的重复处理机制。对于桶载入操作，载入数据的修饰语使你能够指定如何处理重复。

讨论

在默认方式下，当你向表中插入一个与已存在的唯一键值重复的行时，MySQL会产生一个报错。假设person表具有如下结构，即在last_name和first_name列上具有唯一索引：

```
CREATE TABLE person
(
    last_name   CHAR(20) NOT NULL,
    first_name  CHAR(20) NOT NULL,
```

```
address      CHAR(40),
PRIMARY KEY (last_name, first_name)
);
```

如果你试图向表中插入一行，而该行在索引列上含有重复值的话，则会导致一个错误：

```
mysql> INSERT INTO person (last_name, first_name)
-> VALUES('X1','Y1');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO person (last_name, first_name)
-> VALUES('X1','Y1');
ERROR 1062 (23000): Duplicate entry 'X1-Y1' for key 1
```

如果你是通过mysql的前端程序执行这些语句的，那么你可以轻松地说：“好吧，这样做不行”，然后忽略这个错误并继续你的工作。但是如果你是在自己的程序中插入这样的行，那么这个错误可能会终止整个程序。一种可以采取的措施是修改程序的错误处理机制，以捕获这个错误并忽略它。关于错误处理机制请参考2.2节。

如果你希望从一开始就阻止这个错误的发生，你或许会考虑使用二次查询方法以解决重复行的问题：

- 先使用SELECT以检查该行是否已经存在了。
- 如果该行不存在，再使用INSERT语句。

但是这实际上并不起作用：别的客户或许会在你SELECT之后INSERT之前插入一行记录，而该行与你要插入的行相重复。此情况下，在你INSERT后仍然会发生错误。为了确保错误不会发生，你可以使用一个事务或者锁住该表，但这样你的语句将会由2条变为4条，从而执行效率大为下降。MySQL对于处理重复行错误提供了三个简单查询的方案，你可以根据你所需要的重复处理行为从中选择一个：

- 如果当重复发生时你希望保持原有的行，那么应该使用INSERT IGNORE来取代INSERT。如果待插入行与已存在行未发生重复，MySQL将按通常方式插入。如果待插入行是重复行，那么IGNORE关键字将通知MySQL悄悄丢弃该行并且不会产生一个错误：

```
mysql> INSERT IGNORE INTO person (last_name, first_name)
-> VALUES('X2','Y2');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT IGNORE INTO person (last_name, first_name)
-> VALUES('X2','Y2');
Query OK, 0 rows affected (0.00 sec)
```

行计数值指示了该行是被成功插入还是被忽略掉了。在程序中，你可以在你的API中定义一个检查受影响行数的函数，并使用它来检查此计数值。（参见2.4节和9.1节）

- 如果在重复发生时，你希望用新行来取代原有行，那么应该使用REPLACE而非INSERT。如果待插入行未发生重复，那么与INSERT方式一样被正常插入。而在发生重复时，新行将取代旧行：

```
mysql> REPLACE INTO person (last_name, first_name)
-> VALUES('X3','Y3');
Query OK, 1 row affected (0.00 sec)
mysql> REPLACE INTO person (last_name, first_name)
-> VALUES('X3','Y3');
Query OK, 2 rows affected (0.00 sec)
```

此情况下受影响行的计数值为2，包括删除的旧行和插入的新行。

- 如果当重复发生时，你希望修改已存在行的相关列，那么可以使用`INSERT...ON DUPLICATE KEY UPDATE`语句。如果待插入行未重复则被正常插入。如果是重复行，`ON DUPLICATE KEY UPDATE`子句将指示如何修改表中已存在的行。换句话说，此语句可以根据需要插入或者更新一行。可以通过受影响行的计数值了解具体情况：1代表插入，2代表更新。

`INSERT IGNORE`比`REPLACE`效率更高，因为它并不实际地插入重复行。因此，当你只需要证实表中是否存在一个给定行的拷贝时，最好使用前者。另一方面，`REPLACE`更适用于表中其他非键列需要被替换的情况。而如果你要求对于表中未出现的新记录能够正常插入，而在索引列发生重复时更新其中的某些列的话，使用`INSERT...ON DUPLICATE KEY UPDATE`更为合适。

假设你在维护某个Web应用的一个名为`passtbl`的表，该表包含了`email`地址和密码哈希值，其中`email`地址作为键值：

```
CREATE TABLE passtbl
(
    email      VARCHAR(60) NOT NULL,
    password   VARBINARY(60) NOT NULL,
    PRIMARY KEY (email)
);
```

那么你如何为新用户创建新行，而为已存在用户修改他们的密码呢？一个典型的行维护处理算法如下：

- 根据给定的`email`值使用`SELECT`确定该行是否已存在。
- 如果该行不存在，则使用`INSERT`插入一个新行。
- 如果该行已存在，则使用`UPDATE`进行更新。

这些步骤必须在一个事务中或者锁住该表的情况下执行，以阻止其他用户在你执行它们的期间修改该表。在MySQL中，你可以简单地使用`REPLACE`做到在一个简单语句操作中同时实现这两种方式：

```
REPLACE INTO passtbl (email,password) VALUES(address,hash_value);
```

如果没有与给定`email`地址相同的行的出现，MySQL将创建一个新行。如果该行已存在，将会更新该地址所在行的密码列。

当你试图插入一行时，如果你确实知道什么值需要被存储在表中时，`INSERT IGNORE`和`REPLACE`是非常有用的，但这样的情况并非总能出现。举个例子，你或许需要在待插入行

不存在时插入它，否则只更新其中的某些部分，这通常会在你使用表来计数时发生。假设你记录了选举中候选人的得票数，使用的表如下：

```
CREATE TABLE poll_vote
(
    poll_id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    candidate_id INT UNSIGNED,
    vote_count   INT UNSIGNED,
    PRIMARY KEY  (poll_id, candidate_id)
);
```

主键是poll_id和candidate_id的联合，该表的使用方式如下：

- 对于给定选举候选人的第一张得票，将为之建立一条新记录并插入表中，其中vote_count设为1。
- 对于后来该候选人的每一张得票，将已存在记录的vote_count值加1。

此处INSERT IGNORE和REPLACE都不合适，因为除了第一张之外，你不知道已有的得票数是多少。而INSERT ... ON DUPLICATE KEY UPDATE用于此处会更好些。下面的例子显示了它在以一张空表作为起始的情况下是如何工作的：

```
mysql> SELECT * FROM poll_vote;
Empty set (0.01 sec)
mysql> INSERT INTO poll_vote (poll_id,candidate_id,vote_count) VALUES(14,2,1)
-> ON DUPLICATE KEY UPDATE vote_count = vote_count + 1;
Query OK, 1 row affected (0.01 sec)
mysql> SELECT * FROM poll_vote;
+-----+-----+-----+
| poll_id | candidate_id | vote_count |
+-----+-----+-----+
| 14      | 2            | 1          |
+-----+-----+-----+
1 row in set (0.01 sec)
mysql> INSERT INTO poll_vote (poll_id,candidate_id,vote_count) VALUES(14,2,1)
-> ON DUPLICATE KEY UPDATE vote_count = vote_count + 1;
Query OK, 2 rows affected (0.00 sec)
mysql> SELECT * FROM poll_vote;
+-----+-----+-----+
| poll_id | candidate_id | vote_count |
+-----+-----+-----+
| 14      | 2            | 2          |
+-----+-----+-----+
1 row in set (0.00 sec)
```

对于第一个INSERT，由于表中不存在与该候选人相关的行，因此该行被顺利插入。对于第二个INSERT，由于此时该行已存在，因此MySQL仅仅更新vote_count。使用INSERT ... ON DUPLICATE KEY UPDATE时，你不需要检查该行是否已存在，MySQL自动为你做了此事。返回的行计数指示了执行INSERT语句的实际方式：1为新增一行，2为更新了一个已存在行。

以上所描述技术的优点在于消除了使用事务带来的费用，但是这是以牺牲可移植性为代价的，因为它们都含有MySQL所特有的语法。如果在你的应用中可移植性具有较高的优先级，那么你应该坚持使用事务的方法。

参考

对于大多数你通过使用LOAD DATA语句将行集合从文件装载到表中的记录装载操作而言，重复行的处理可以通过使用IGNORE和REPLACE这两个限定语来控制。这样所产生的行为类似于INSERT IGNORE and REPLACE语句类似。更多的信息请参考10.7节。INSERT... ON DUPLICATE KEY UPDATE用于初始化和更新计数的使用将在11.14节和19.12节进一步的演示。

14.3 计数和识别重复项

Counting and Identifying Duplicates

问题

你需要确定表中是否含有重复项，它们出现的频率如何，或者你希望能够发现包含重复值的行。

解决方案

使用计数摘要以查找并显示重复值。为了查看重复值发生的行，将该摘要与原始表相连接以显示匹配的行。

讨论

假设你的Web主页包含注册页面，使访问者能够将他们添加到你的邮件列表中以定期接收产品目录邮件。但是当你创建表时，忘记在表中包含一个唯一性索引，并且现在你怀疑一些人已经重复注册。可能是因为他们忘记了他们已经位于列表中，或者可能人们将他们的朋友也加入列表中，而实际上他们已经注册过了。无论什么原因，造成重复行的结果都使你需要发出重复目录，这对于你来说是额外的花费，并且对接收者造成了困扰。本节讨论如何发现表中是否存在重复项，它们出现的频繁程度怎样，以及如何显示这些重复项。（对于包含重复项的表，14.4节描述了如何消除它们。）

为了确定表中是否发生了重复，可以使用计数摘要（第8章所涵盖的主题）。摘要技术可以通过使用GROUP BY将行分组，并对每个组使用COUNT()函数进行计数，以确定并计算出重

复项个数。本节的例子中，假设目录的接收人被列在表catalog_list中，并包括下面的内容：

```
mysql> SELECT * FROM catalog_list;
+-----+-----+-----+
| last_name | first_name | street
+-----+-----+-----+
| Isaacson | Jim | 515 Fordam St., Apt. 917 |
| Baxter | Wallace | 57 3rd Ave.
| McTavish | Taylor | 432 River Run
| Pinter | Marlene | 9 Sunset Trail
| BAXTER | WALLACE | 57 3rd Ave.
| Brown | Bartholomew | 432 River Run
| Pinter | Marlene | 9 Sunset Trail
| Baxter | Wallace | 57 3rd Ave., Apt 102
+-----+-----+-----+
```

假设你希望根据last_name和first_name列来定义“重复”，也即是说，具有同样名字的接收者会被认为是同一个人。下面的语句典型地描述了表内容并估计重复值的存在与范围：

- 表的总行数：

```
mysql> SELECT COUNT(*) AS rows FROM catalog_list;
+-----+
| rows |
+-----+
|     8 |
+-----+
```

- 不同名字的行数：

```
mysql> SELECT COUNT(DISTINCT last_name, first_name) AS 'distinct names'
-> FROM catalog_list;
+-----+
| distinct names |
+-----+
|          5 |
+-----+
```

- 包含重复名字的行数：

```
mysql> SELECT COUNT(*) - COUNT(DISTINCT last_name, first_name)
-> AS 'duplicate names'
-> FROM catalog_list;
+-----+
| duplicate names |
+-----+
|          3 |
+-----+
```

- 包含唯一或不唯一名字的行的比例：

```
mysql> SELECT COUNT(DISTINCT last_name, first_name) / COUNT(*)
-> AS 'unique',
-> 1 - (COUNT(DISTINCT last_name, first_name) / COUNT(*))
```

```

-> AS 'nonunique'
-> FROM catalog_list;
+-----+
| unique | nonunique |
+-----+
| 0.6250 |      0.3750 |
+-----+

```

这些语句帮助你描述了重复项的范围，但它们没有显示哪些值是重复的。为了发现catalog_list表的重复名字，使用摘要语句显示不唯一的值及其数目：

```

mysql> SELECT COUNT(*) AS repetitions, last_name, first_name
-> FROM catalog_list
-> GROUP BY last_name, first_name
-> HAVING repetitions > 1;
+-----+-----+-----+
| repetitions | last_name | first_name |
+-----+-----+-----+
|          3 | Baxter    | Wallace   |
|          2 | Pinter    | Marlene  |
+-----+-----+-----+

```

语句包含了HAVING子句，限制输出中只包含那些出现次数多于一次的名字。（如果你省略了该子句，摘要也显示了只出现一次的名字，这当你只对重复项感兴趣时是毫无用处的。）通常确定重复值的集合，需要如下操作：

1. 确定哪些列可能包含重复值。
2. 连同COUNT(*)，在列选择列表中列出那些列。
3. 在GROUP BY子句中同样列出这些列。
4. 增加HAVING子句，限制组的数目必须多于一个以消除唯一值。

以这种方式构建的查询具有下面的形式：

```

SELECT COUNT(*), column_list
FROM tbl_name
GROUP BY column_list
HAVING COUNT(*) > 1

```

给定数据库和表名，以及一个非空的列名集合，在程序中很容易生成如上的查找重复项的查询。举例来说，下面的Perl函数make_dup_count_query()生成了在给定列发现并计数重复值的正确查询：

```

sub make_dup_count_query
{
my ($db_name, $tbl_name, @col_name) = @_;
return (
    "SELECT COUNT(*), " . join (", ", @col_name)
. "\nFROM $db_name.$tbl_name"

```

```
. "\nGROUP BY " . join (",", @col_name)
. "\nHAVING COUNT(*) > 1"
);
}
```

make_dup_count_query() 返回查询结果作为字符串，如果你像这样调用它：

```
$str = make_dup_count_query ("cookbook", "catalog_list",
                             "last_name", "first_name");
```

\$str的返回值是：

```
SELECT COUNT(*),last_name,first_name
FROM cookbook.catalog_list
GROUP BY last_name,first_name
HAVING COUNT(*) > 1
```

怎样使用该查询字符串取决于你，你可以在创建它的脚本中执行它，也可以将之传给另一个程序，或者将之写入文件中以便以后执行。本节的dups目录下包含了名为dup_count.pl的脚本，你可以使用它来试验该函数（其中还包括向其他语言的移植）。14.4节讨论了使用make_dup_count_query()函数来实现删除重复项的技术。

摘要技术对于估计重复项是否存在、发生的频率以及显示重复值都是有用处的。但是如果重复项被确定只使用了表的所有列的一个子集，那么摘要本身不能够显示包含重复值的行的完整内容。（举个例子，上面的摘要显示了在catalog_list表中重复名字的个数，但是没有显示与这些名字相关的地址。）为了发现包含重复名的原始行，需要将摘要信息与产生它的表相连接。下面的例子显示了如何进行操作以显示包含重复名的catalog_list行。摘要被写入到临时表中，然后与catalog_list相连接，以产生与那些名字匹配的行：

```
mysql> CREATE TABLE tmp
    -> SELECT COUNT(*) AS count, last_name, first_name FROM catalog_list
    -> GROUP BY last_name, first_name HAVING count > 1;
mysql> SELECT catalog_list.*
    -> FROM tmp INNER JOIN catalog_list USING(last_name, first_name)
    -> ORDER BY last_name, first_name;
+-----+-----+-----+
| last_name | first_name | street          |
+-----+-----+-----+
| Baxter   | Wallace   | 57 3rd Ave.      |
| BAXTER   | WALLACE   | 57 3rd Ave.      |
| Baxter   | Wallace   | 57 3rd Ave., Apt 102 |
| Pinter   | Marlene   | 9 Sunset Trail    |
| Pinter   | Marlene   | 9 Sunset Trail    |
+-----+-----+-----+
```

重复性鉴别与字符串大小写问题

对于那些含有大小写敏感字序集、仅仅由大小写来区分的数值的串，处于比较的目的将被视为相同。为了将它们视为独立值，可以使用区分大小写或者二进制的方法加以比较。5.9节显示了如何进行此项操作。

14.4 从表中消除重复项

Eliminating Duplicates from a Table

问题

你需要从表中删除重复的行，以使之只含有具有唯一性的行。

解决方案

从表中选择具有唯一性的行并放入第二张表中，你可以使用它来取代原始的表，或者使用ALTER TABLE增加一个唯一性索引，这些方法都消除了重复项并创建了索引，或者使用DELETE ... LIMIT n来删除给定重复行集合中多余的行。

讨论

14.1节讨论了如何通过在创建表时增加唯一性索引来阻止重复项被添加到该表中。但是，如果你在创建表时忘记了增加唯一性索引，之后你或许会发现它包含了重复项，并且必须使用某种删除重复项的技术。早先使用的catalog_list就是一个例子，因为它包含了几个实例，其中同一个人被列出了多次：

```
mysql> SELECT * FROM catalog_list ORDER BY last_name, first_name;
+-----+-----+-----+
| last_name | first_name | street
+-----+-----+-----+
| Baxter    | Wallace   | 57 3rd Ave.
| BAXTER    | WALLACE   | 57 3rd Ave.
| Baxter    | Wallace   | 57 3rd Ave., Apt 102
| Brown     | Bartholomew | 432 River Run
| Isaacsion | Jim        | 515 Fordam St., Apt. 917
| McTavish  | Taylor     | 432 River Run
| Pinter    | Marlene   | 9 Sunset Trail
| Pinter    | Marlene   | 9 Sunset Trail
+-----+-----+-----+
```

表中包含了多余的条目，为了消除重复的邮件并减少邮费，那么删除它们是一个好主意。为了实现此目标，你有下面几个选择：

- 将表中具有唯一性的行选入另一个表中，然后使用该表来取代原来的表。这样的结果就是去除了表的重复项。这种方法在“重复”意味着“整个行与另一个相同”时起作用。
- 使用ALTER TABLE向表中增加唯一性索引。此操作将重复行转化为唯一行，此处“重复”的含义为“索引值相同”。
- 你可以在某个重复行集合中通过DELETE ... LIMIT n来删除重复项，只保留其中的一行。

本节讨论了每一种消除重复的方法。当你在不同的环境下考虑选择其中的哪一个时，对特定问题的给定方法的适用性通常根据下面几个因素确定：

- 该方法是否要求表含有唯一性索引？
- 如果发生重复值的列中可能包含了NULL，那么该方法会删除重复的NULL值吗？
- 该方法能够阻止将来不再出现重复项吗？

使用表替换来删除重复项

如果只有一个行与另一个行完全相同时，才会认为该行是重复的，那么从表中消除重复项的一种方法是将具有唯一性的行移入一个具有同样结构的新表中，并使用新表来取代原始表。执行表替换方法，可以使用下面的过程：

- 创建一个与原始表具有同样结构的新表。对此，CREATE TABLE ... LIKE是非常有效的（参见4.1节）：

```
mysql> CREATE TABLE tmp LIKE catalog_list;
```

- 使用INSERT INTO ... SELECT DISTINCT从原始表中选择具有唯一性的行，并将之转移到新表中去：

```
mysql> INSERT INTO tmp SELECT DISTINCT * FROM catalog_list;
```

从tmp表中选择行，以验证新表中不再含有重复项：

```
mysql> SELECT * FROM tmp ORDER BY last_name, first_name;
+-----+-----+-----+
| last_name | first_name | street
+-----+-----+-----+
| Baxter   | Wallace    | 57 3rd Ave.
| Baxter   | Wallace    | 57 3rd Ave., Apt 102
| Brown    | Bartholomew | 432 River Run
| Isaacson | Jim         | 515 Fordam St., Apt. 917
| McTavish  | Taylor      | 432 River Run
| Pinter    | Marlene     | 9 Sunset Trail
+-----+-----+-----+
```

- 在创建包含唯一性索引的新的tmp表之后，可以使用它来取代原始表：

```
mysql> DROP TABLE catalog_list;
mysql> RENAME TABLE tmp TO catalog_list;
```

此过程的有效结果为catalog_list不再含有重复项。

表替换方法可以工作在缺乏索引的情况下（尽管可能对于大型表来说速度较慢）。对于包含重复NULL值的表，它将删除那些重复项，但并不能阻止将来出现重复项。

此方法要求被认为是重复项的行必须完全相同。因而，它将名字都为Wallace Baxter，但街道号码稍有不同的那些行视为不同的行。

如果重复项被定义为只与表中列的子集相关，可以为那些列创建一个具有唯一性索引的新表，并使用INSERT IGNORE将这些行选入其中，然后将原始表替换为新表：

```
mysql> CREATE TABLE tmp LIKE catalog_list;
mysql> ALTER TABLE tmp ADD PRIMARY KEY (last_name, first_name);
mysql> INSERT IGNORE INTO tmp SELECT * FROM catalog_list;
mysql> SELECT * FROM tmp ORDER BY last_name, first_name;
+-----+-----+-----+
| last_name | first_name | street
+-----+-----+-----+
| Baxter    | Wallace    | 57 3rd Ave.
| Brown     | Bartholomew | 432 River Run
| Isaacson  | Jim         | 515 Fordam St., Apt. 917
| McTavish   | Taylor      | 432 River Run
| Pinter     | Marlene    | 9 Sunset Trail
+-----+-----+-----+
mysql> DROP TABLE catalog_list;
mysql> RENAME TABLE tmp TO catalog_list;
```

唯一性索引阻止了在向tmp表插入行时出现重复的键值，IGNORE提示MySQL当重复发生时不要因为报错停止。这种方法的缺点是如果索引列需要包含NULL值，那么你必须使用UNIQUE索引而不是PRIMARY KEY，此时索引将不会删除多余的NULL键值。（即UNIQUE索引允许多个NULL值。）此方法阻止了以后重复项的产生。

通过增加索引来删除重复项

为了适当地从表中删除重复项，需要使用ALTER TABLE向表中增加唯一性索引，并使用IGNORE关键字来提示在索引构建过程中丢弃带有重复键值的行。没有索引的原始catalog_list表如下所示：

```
mysql> SELECT * FROM catalog_list ORDER BY last_name, first_name;
+-----+-----+-----+
| last_name | first_name | street
+-----+-----+-----+
| Baxter    | Wallace    | 57 3rd Ave.
| BAXTER    | WALLACE    | 57 3rd Ave.
```

Baxter	Wallace	57 3rd Ave., Apt 102
Brown	Bartholomew	432 River Run
Isaacson	Jim	515 Fordam St., Apt. 917
McTavish	Taylor	432 River Run
Pinter	Marlene	9 Sunset Trail
Pinter	Marlene	9 Sunset Trail

增加一个唯一性索引，然后检查此操作给表的内容带来了什么影响：

```
mysql> ALTER IGNORE TABLE catalog_list
    -> ADD PRIMARY KEY (last_name, first_name);
mysql> SELECT * FROM catalog_list ORDER BY last_name, first_name;
+-----+-----+-----+
| last_name | first_name | street      |
+-----+-----+-----+
| Baxter   | Wallace    | 57 3rd Ave. |
| Brown    | Bartholomew | 432 River Run |
| Isaacson | Jim        | 515 Fordam St., Apt. 917 |
| McTavish  | Taylor     | 432 River Run |
| Pinter    | Marlene    | 9 Sunset Trail |
+-----+-----+-----+
```

如果索引列可以包含NULL值，你必须使用UNIQUE索引而不是PRIMARY KEY。在此情况下，索引不会删除重复的NULL键值。除了能够删除已存在的重复项，此方法还能防止未来重复项的出现。

删除特定行的重复项

你可以使用LIMIT来限制DELETE语句对待删除的行子集的影响，这使此语句适用于删除重复行。假设你有一个具有下面内容的表t：

color
blue
green
blue
blue
red
green
red

该表列出了blue三次，green和red各两次。为了删除每种颜色的多余实例，可以如下操作：

```
mysql> DELETE FROM t WHERE color = 'blue' LIMIT 2;
mysql> DELETE FROM t WHERE color = 'green' LIMIT 1;
mysql> DELETE FROM t WHERE color = 'red' LIMIT 1;
mysql> SELECT * FROM t;
+-----+
| color |
+-----+
```

```
+-----+
| blue   |
| green  |
| red    |
+-----+
```

此技术可以在缺少唯一性索引时工作，并且消除了重复的NULL值。如果你只想对表中特定的行集合删除重复项，该方法是十分快捷的。不过，如果你想要删除的不同的重复项子集有很多，你肯定不愿手动地完成这个过程，此过程可以使用前面在14.3节中讨论的如何确定哪些是重复值的技术来自动实现。在那里，我们编写了make_dup_count_query()函数来产生语句，以在表中给定的列集合中计数重复值发生的次数。该语句的结果可以用来产生一个DELETE ... LIMIT n语句的集合以删除重复行并只保留唯一的行。本节的dups目录包含了如何产生那些语句的代码。

一般而言，使用DELETE ... LIMIT n可能会比通过使用第二张表或增加唯一性索引的方法更慢。那些方法将数据保存在服务器端并让服务器做所有的工作。DELETE ... LIMIT n包含了许多客户端——服务器交互，因为它使用SELECT语句来得到关于重复项的信息，之后才是几个删除重复行实例的DELETE语句。同样，此技术没有阻止将来表中重复项的发生。

14.5 从自连接的结果中消除重复

Eliminating Duplicates from a Self-Join Result

问题

自连接经常会产生“近似”重复的行——也即是说，这些行包含了同样的值，但是是以不同的顺序存储的。因此，SELECT DISTINCT将无法删除重复项。

解决方案

在行中以特定顺序选择列值以将行归类为不同值的重复项集合，然后可以使用SELECT DISTINCT来删除重复项。而另一种可选的方法甚至不需要选择那些以此种方式提取的具有近似重复项的行。

讨论

自连接可以产生意义上重复的行，因为它们包含了相同的值，但是又不是完全一样的。考虑下面的语句（参见12.3节），该语句使用自连接来查找所有的在同一年加入联邦的州对：

```

mysql> SELECT YEAR(s1.statehood) AS year,
-> s1.name AS name1, s1.statehood AS statehood1,
-> s2.name AS name2, s2.statehood AS statehood2
-> FROM states AS s1 INNER JOIN states AS s2
-> ON YEAR(s1.statehood) = YEAR(s2.statehood) AND s1.name != s2.name
-> ORDER BY year, s1.name, s2.name;
+-----+-----+-----+-----+
| year | name1      | statehood1 | name2      | statehood2 |
+-----+-----+-----+-----+
| 1787 | Delaware    | 1787-12-07 | New Jersey  | 1787-12-18 |
| 1787 | Delaware    | 1787-12-07 | Pennsylvania | 1787-12-12 |
| 1787 | New Jersey   | 1787-12-18 | Delaware    | 1787-12-07 |
| 1787 | New Jersey   | 1787-12-18 | Pennsylvania | 1787-12-12 |
| 1787 | Pennsylvania | 1787-12-12 | Delaware    | 1787-12-07 |
| 1787 | Pennsylvania | 1787-12-12 | New Jersey  | 1787-12-18 |
...
| 1912 | Arizona     | 1912-02-14 | New Mexico   | 1912-01-06 |
| 1912 | New Mexico   | 1912-01-06 | Arizona     | 1912-02-14 |
| 1959 | Alaska       | 1959-01-03 | Hawaii      | 1959-08-21 |
| 1959 | Hawaii       | 1959-08-21 | Alaska      | 1959-01-03 |
+-----+-----+-----+-----+

```

ON子句中的条件要求州对的名字不能相同以消除琐碎的重复行，即那些显示每个州与它自己同一年加入联邦的行。但是每个剩余的州对仍然出现了两次。举例来说，有一行列出了New Jersey和Delaware，同时另一行列出了Delaware和New Jersey。每个此类的行对都可以被认为是实质上的重复行，因为它们包含了同样的值。然而，由于这些值不是在行中以相同的顺序被列出，因此它们不是同一的，从而你无法通过在语句中增加DISTINCT来删除重复行。

解决此问题的一种方法是确保州名在行中始终以特定的顺序被列出。这可以通过使用一对表达式选择州名来实现，即在输出列的列表中将较小值放在首位：

```

IF(val1<val2, val1, val2) AS lesser_value,
IF(val1>val2, val1, val2) AS greater_value

```

将此技术应用到州对查询中会产生下面的结果，其中的表达式在每行中以词典顺序显示州名：

```

mysql> SELECT YEAR(s1.statehood) AS year,
-> IF(s1.name< s2.name, s1.name, s2.name) AS name1,
-> IF(s1.name< s2.name, s1.statehood, s2.statehood) AS statehood1,
-> IF(s1.name> s2.name, s2.name) AS name2,
-> IF(s1.name> s2.name, s2.statehood, s1.statehood) AS statehood2
-> FROM states AS s1 INNER JOIN states AS s2
-> ON YEAR(s1.statehood) = YEAR(s2.statehood) AND s1.name != s2.name
-> ORDER BY year, name1, name2;
+-----+-----+-----+-----+
| year | name1      | statehood1 | name2      | statehood2 |
+-----+-----+-----+-----+
| 1787 | Delaware    | 1787-12-07 | New Jersey  | 1787-12-18 |
| 1787 | Delaware    | 1787-12-07 | New Jersey  | 1787-12-18 |
+-----+-----+-----+-----+

```

1787 Delaware	1787-12-07 Pennsylvania	1787-12-12
1787 Delaware	1787-12-07 Pennsylvania	1787-12-12
1787 New Jersey	1787-12-18 Pennsylvania	1787-12-12
1787 New Jersey	1787-12-18 Pennsylvania	1787-12-12
...		
1912 Arizona	1912-02-14 New Mexico	1912-01-06
1912 Arizona	1912-02-14 New Mexico	1912-01-06
1959 Alaska	1959-01-03 Hawaii	1959-08-21
1959 Alaska	1959-01-03 Hawaii	1959-08-21

重复行仍然出现在输出中，但是现在重复对是同一的了，你可以通过向语句增加DISTINCT来消除多余的拷贝：

```
mysql> SELECT DISTINCT YEAR(s1.statehood) AS year,
-> IF(s1.name < s2.name, s1.name, s2.name) AS name1,
-> IF(s1.name < s2.name, s1.statehood, s2.statehood) AS statehood1,
-> IF(s1.name < s2.name, s2.name, s1.name) AS name2,
-> IF(s1.name < s2.name, s2.statehood, s1.statehood) AS statehood2
-> FROM states AS s1 INNER JOIN states AS s2
-> ON YEAR(s1.statehood) = YEAR(s2.statehood) AND s1.name != s2.name
-> ORDER BY year, name1, name2;
+-----+-----+-----+-----+
| year | name1      | statehood1 | name2      | statehood2 |
+-----+-----+-----+-----+
| 1787 | Delaware   | 1787-12-07 | New Jersey | 1787-12-18 |
| 1787 | Delaware   | 1787-12-07 | Pennsylvania | 1787-12-12 |
| 1787 | New Jersey | 1787-12-18 | Pennsylvania | 1787-12-12 |
...
| 1912 | Arizona    | 1912-02-14 | New Mexico | 1912-01-06 |
| 1959 | Alaska     | 1959-01-03 | Hawaii     | 1959-08-21 |
+-----+-----+-----+-----+
```

另一种删除非同一的重复项的技术不依赖于在选择的行中检查并消除它们，而是在查询结果中只显示每对中的一行。这使得在输出行中重新排序值并使用DISTINCT不再是必须的了。对于州对查询，只选择那些第一个州名在词典中小于第二个州名的那些行，从而自动删除了以另一个顺序出现的那些州名（注1）：

注1：这一约束也删除了州名相同的那些行。

```
mysql> SELECT YEAR(s1.statehood) AS year,
-> IF(s1.name < s2.name, s1.name, s2.name) AS name1,
-> IF(s1.name < s2.name, s1.statehood, s2.statehood) AS statehood1,
-> IF(s1.name < s2.name, s2.name, s1.name) AS name2,
-> IF(s1.name < s2.name, s2.statehood, s1.statehood) AS statehood2
-> FROM states AS s1 INNER JOIN states AS s2
-> ON YEAR(s1.statehood) = YEAR(s2.statehood) AND s1.name < s2.name
-> ORDER BY year, name1, name2;
+-----+-----+-----+-----+
| year | name1      | statehood1 | name2      | statehood2 |
+-----+-----+-----+-----+
| 1787 | Delaware    | 1787-12-07 | New Jersey   | 1787-12-18 |
| 1787 | Delaware    | 1787-12-07 | Pennsylvania | 1787-12-12 |
| 1787 | New Jersey   | 1787-12-18 | Pennsylvania | 1787-12-12 |
...
| 1912 | Arizona     | 1912-02-14 | New Mexico   | 1912-01-06 |
| 1959 | Alaska       | 1959-01-03 | Hawaii       | 1959-08-21 |
+-----+-----+-----+-----+
```


执行事务

Performing Transactions

15.0 引言

Introduction

因为采取多线程方式，MySQL服务器可以同时处理多个客户端。为了解决客户端冲突的问题，服务器执行必要的锁定操作以确保两个客户端不能同时修改同样的数据。但是，当服务器执行SQL语句时，很可能会出现这样的情况：在接收给定客户端的连续语句时，与其他客户端的语句产生交叉。如果客户端所提交的多重语句是相互独立的，并且其他客户端在这些语句中对表进行了更新，那么可能会产生问题。如果一个多语句操作没有被完整地运行，也会造成语句执行失败。假设你有一个flight表，其中包含了关于航班时间表的信息，并且你需要更新Flight 578所在的行，即为其从备用名单中选出一个飞行员。你可以使用如下三个语句来实现：

```
SELECT @p_val := pilot_id FROM pilot WHERE available = 'yes' LIMIT 1;
UPDATE pilot SET available = 'no' WHERE pilot_id = @p_val;
UPDATE flight SET pilot_id = @p_val WHERE flight_id = 578;
```

第一个语句从备用飞行员中选择一个，第二个语句将该飞行员标记为已用，第三个语句将这个飞行员赋给此航班。在实际生活中这是直接了当的做法，但是理论上，此过程中存在两个重大的问题：

并发性问题

如果两个客户端需要调度飞行员，他们可能都会运行第一个SELECT查询语句并获得同样的飞行员ID，并且都没来得及修改飞行员的状态为已用。如果发生了这种情况，同一飞行员会被两个航班同时调度。

完整性问题

这三个语句必须作为一个单元完整地被执行。举个例子，如果SELECT和第一个UPDATE成功执行完毕，但第二个UPDATE失败了，那么该飞行员的状态会被设置为已用，而实际上他并没有被调度给航班，从而数据库处于一个不一致状态。

在这类情况下防止并发性和完整性问题的发生，使用事务是非常有效的。一个事务将语句集组合在一起并保证其具有下面的属性：

- 在事务执行过程中，别的客户端不能更新事务中所使用的数据，这好像是你独自拥有服务器一样。举个例子，当你为一个航班预订飞行员的时候，其他客户不能够修改pilot或者flight记录。通过阻止其他客户端干扰你正在执行的操作，事务解决了MySQL服务器多用户特性带来的并发性问题。实际上，事务将在多语句操作中对共享资源的访问串行化了。
- 只有在事务中的语句都执行成功的情况下，它们才作为一个单元被聚合并提交（发生作用），任何在发生错误前的操作都会被回滚，以保证相关表没有受影响，就如同语句中任何一个都没有被执行一样，这防止了数据库产生不一致。举例来说，如果一个更新flight表的操作失败，回滚将会取消pilots表的变化，使该飞行员仍然为可用。回滚使你不再需要手动地指出如何撤销部分完成的操作。

本章显示了MySQL语句中开始和结束事务的记号，描述了如何在程序中实现事务操作，并使用错误探测来决定是提交还是回滚。最后一节讨论了你可以在应用中使用非事务化的存储引擎来仿真事务。有时候通过使用LOCK TABLE和UNLOCK TABLE在多重语句中锁住你的表就已经够用了，这会阻止其他客户的干扰，尽管如果其中有语句失败时，不会产生回滚。另一种可选的方法是重写语句以使它们不再需要事务。

这里所显示例子的脚本位于本节中transactions目录下。

15.1 使用事务存储引擎

Choosing a Transactional Storage Engine

问题

你需要使用事务。

解决方案

检查你的MySQL服务器以确定它支持哪些事务存储引擎。

讨论

MySQL支持几种存储引擎，但是并不是每一种都支持事务。为了能够使用事务，你必须使用事务安全存储引擎。目前，事务引擎包括InnoDB、NDB和BDB等。要检查你的MySQL服务器支持哪些引擎，可以通过查看SHOW ENGINES语句的输出：

```
mysql> SHOW ENGINES\G
***** 1. row *****
Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
***** 2. row *****
Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
***** 3. row *****
Engine: InnoDB
Support: YES
Comment: Supports transactions, row-level locking, and foreign keys
***** 4. row *****
Engine: BerkeleyDB
Support: YES
Comment: Supports transactions and page-level locking
...
...
```

上面显示了MySQL 5.0的输出。事务引擎可以根据注释值来确定，即那些Support值为YES或DEFAULT的引擎为可用的。在MySQL5.1中，SHOW ENGINES的输出包括一个事务列，其中，明确说明了哪些引擎支持事务。

在确定哪些事务存储引擎可用之后，你可以通过在你的CREATE TABLE语句中增加ENGINE = *tbl_engine*子句来创建使用给定引擎的表：

```
CREATE TABLE t1 (i INT) ENGINE = InnoDB;
CREATE TABLE t2 (i INT) ENGINE = BDB;
```

如果你的应用中已有还没有使用事务的表，但是你需要修改它使之能够执行事务，你可以更改此表来使用事务存储引擎。举例来说，MyISAM表是非事务的，如果试图将他们用于事务会导致错误的结果，因为它们并不支持事务回滚。在此情况下，你可以使用ALTER TABLE将这类表转化为事务类型的。假设t是一个MyISAM表，可以通过下面操作将之变为InnoDB表：

```
ALTER TABLE t ENGINE = InnoDB;
```

在改变一个表之前，需要考虑到将其转变为使用事务存储引擎的表可能会影响其他方面影响其行为。例如，相对于其他存储引擎而言，MyISAM引擎提供更灵活的AUTO_INCREMENT处

理。如果你的应用依赖于只适用于MyISAM的序号特征，改变其存储引擎会产生问题。参见11章以获得更多信息。

15.2 使用SQL执行事务

Performing Transactions Using SQL

问题

你需要保证一个语句集合必须作为一个单元执行，要么成功，要么失败，也就是说，你需要执行事务。

解决方案

使用MySQL的自动提交模式以激活多语句事务，然后根据语句执行成功与否决定提交或回滚。

讨论

本节描述了在MySQL中控制事务行为的SQL语句。下一节讨论如何在程序中实现事务。一些API需要你通过使用本章讨论的SQL语句来实现事务，而其他则提供了一个无需直接编写SQL来激活事务管理的特殊机制。但是，即使是后一种情况，API机制也是将程序操作映射到事务SQL语句，因此阅读本章将会帮助你理解API是如何为你工作的。

MySQL通常以自动提交模式工作，该方式在语句一执行时就提交。（实际上，每个语句都是它自己的事务。）为了执行事务，你必须关闭自动提交模式，组织组成事务的语句，并将之提交或回滚。在MySQL中，你可以做下面两个方面的工作：

- 使用START TRANSACTION（或BEGIN）语句以挂起自动提交模式，然后发出组成事务的语句。如果语句执行成功，将结果记录入数据库并通过发出COMMIT来结束事务：

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
mysql> START TRANSACTION;
mysql> INSERT INTO t (i) VALUES(1);
mysql> INSERT INTO t (i) VALUES(2);
mysql> COMMIT;
mysql> SELECT * FROM t;
+----+
| i |
+----+
```

```
+----+  
| 1 |  
| 2 |  
+----+
```

如果错误发生，则不能使用COMMIT。相反地，应通过ROLLBACK语句取消事务。下面的例子中，t在事务结束后为空，因为INSERT语句已经被回滚：

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;  
mysql> START TRANSACTION;  
mysql> INSERT INTO t (i) VALUES(1);  
mysql> INSERT INTO t (x) VALUES(2);  
ERROR 1054 (42S22): Unknown column 'x' in 'field list'  
mysql> ROLLBACK;  
mysql> SELECT * FROM t;  
Empty set (0.00 sec)
```

- 另一种聚合语句的方法是通过将自动提交会话变量设置为0来显式地关闭自动提交模式。之后，你编写的每个语句都成为当前事务的一部分。为了结束事务并开始下一个，需要发布COMMIT或者ROLLBACK语句：

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;  
mysql> SET autocommit = 0;  
mysql> INSERT INTO t (i) VALUES(1);  
mysql> INSERT INTO t (i) VALUES(2);  
mysql> COMMIT;  
mysql> SELECT * FROM t;  
+----+  
| i |  
+----+  
| 1 |  
| 2 |  
+----+
```

可以使用下面语句将自动提交模式设回原来的状态：

```
mysql> SET autocommit = 1;
```

不是一切都可以取消的

事务具有其局限性，因为不是所有的语句都可以作为事务的一部分。举个例子，如果你发起了一个DROP DATABASE语句，不可能通过执行ROLLBACK来恢复数据库。

15.3 在程序中执行事务

Performing Transactions from Within Programs

问题

你正在编写一个需要实现事务操作的程序。

解决方案

如果你所用的程序语言API提供事务术语的话，则可以直接使用它。如果没有，则可以使用API的常用语句执行机制来发布事务SQL语句，以直接利用普通数据库调用的API。

讨论

当你在mysql程序中编写交互式的语句时（如上一节所示的例子），你通过检查语句是否执行成功来决定提交或回滚。而在存储于文件中的非交互式的SQL脚本中，这种方式则不能很好地工作。你不能根据语句成功与否来确定应该提交还是回滚，因为MySQL没有包含IF/THEN/ELSE结构以控制脚本的流程。（有IF()函数，但不是一回事。）出于这个原因，一般都是在程序中执行事务处理，因为你可以使用API语言来探测错误并选择合适的操作。本节讨论了一些关于如何处理这种情形的一般背景。下一节将提供特定语言关于MySQL API的细节，如Perl、Ruby、PHP、Python和Java。

每个MySQL API都支持事务，即使表面上看来你只能显式地使用事务相关的SQL语句，如START TRANSACTION 和COMMIT。不过，一些API同样提供事务术语以使你能够不直接使用SQL来控制事务行为。这种方式隐藏了实现细节，并提供了对其他具有不同的底层事务SQL语法的数据库引擎更好的迁移性。对于我们在这本书中使用的每种语言，事务API术语都是可用的。

下面几节将分别实现同一个例子来说明如何执行基于程序的事务。它们使用的表t包含了下面的初始行，其中显示了两个人各有多少钱：

name	amt
Eve	10
Ida	0

事务用例是简单的资金转移，即使用两个UPDATE语句从Eve的钱中取出6元给Ida：

```
UPDATE money SET amt = amt - 6 WHERE name = 'Eve';
UPDATE money SET amt = amt + 6 WHERE name = 'Ida';
```

所期望的结果如下所示：

name	amt
Eve	4

必须在一个事务中执行两个语句以保证它们同时产生作用。如果没有事务，假设第二个语句失败，Eve的钱将会丢失而没有计入Ida的户头。通过使用事务，如果语句产生错误，该表会保持原状态不变。

每种语言的示例程序位于本章的transactions目录下。如果对它们进行比较，你会发现它们都使用了类似的框架来执行事务处理：

- 事务中的语句连同commit操作被聚合在一个控制结构中。
- 如果控制结构的状态显示了没有成功执行完毕，那么事务会被回滚。

程序逻辑可以如下表示，其中block代表用于聚合语句的控制结构：

```
block:  
    statement 1  
    statement 2  
    ...  
    statement n  
    commit  
    if the block failed:  
        roll back
```

如果block中的语句成功执行，程序到达了block的末尾并提交。否则，如果产生错误则会导致一个异常，并触发了错误处理代码以执行事务回滚。

如上所示，将代码结构化的益处在于可以将确定是否回滚的测试次数减到最少。而另一种方式——检查事务中每个语句的结果并根据单个语句错误产生回滚，这会很快地使你的代码变得杂乱而难以阅读。

一个需要了解的细节在于程序中回滚也会产生异常，因为回滚操作本身也可能失败从而导致了另一个异常。如果你不对其进行处理，则程序将会终止。为了处理这种情况，需要将回滚操作置于另一个包括异常处理器的程序块中，示例程序中包含了此项操作。

这些示例程序都在执行事务时显式地关闭了自动提交模式，并在执行完毕后重新激活该模式。因为在应用中不必要以事务方式执行所有的数据库处理，因此在你连接数据库服务器以后，一旦关闭了自动提交模式，需要将它再设回来。

检查API事务术语是如何映射到SQL语句的

对于提供事务术语的API，你可以查看它们的接口是如何映射到底层SQL语句的：先激活MySQL服务器的日常查询日志，再观察日志文件来查看在你运行事务程序时API执行了哪些语句。（参见MySQL参考手册以获取激活日志的用法说明。）

15.4 在Perl程序中使用事务

Using Transactions in Perl Programs

问题

你需要在Perl DBI脚本中执行事务。

解决方案

使用标准DBI事务支持机制。

讨论

Perl DBI事务机制基于显式的自动提交模式操作：

1. 如果RaiseError属性没有被激活则打开它，如果PrintError属性被打开则关闭它。你需要的是错误产生异常而不用打印任何信息，如果PrintError被打开在某些情况下会干扰错误检测。
2. 关闭AutoCommit属性以使提交仅在你发出命令后进行。
3. 在eval块中执行组成事务的语句，以保证错误产生异常并结束程序块。在块的末尾应当调用commit()，以在所有语句都成功完成的情况下提交事务。
4. 在eval执行完毕以后，检查\$@变量，如果\$@包含空字符串，则事务执行成功。否则，eval由于产生错误而造成失败，此时\$@会包含错误消息，然后调用rollback()来取消事务。如果你需要显示错误消息，在调用rollback()之前可以将\$@打印出来。

下面的代码显示了如何实现这些步骤以执行此范例事务：

```
#正确的设置错误处理和自动提交属性  
$dbh->{RaiseError} = 1; # 如果错误发生则抛出异常  
$dbh->{PrintError} = 0; # 不打印出错信息  
$dbh->{AutoCommit} = 0; # 取消自动提交
```

```

eval
{
    # 把一些钱从一个人的移动到另一个人
    $dbh->do ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'");
    $dbh->do ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'");
    # 所有语句执行成功; 提交事务
    $dbh->commit ();
};

if ($@) # 错误发生
{
    print "Transaction failed, rolling back. Error was:\n$@\n";
    # 在eval中处理以阻止回滚
    # 终止运行失败的脚本
    eval { $dbh->rollback (); };
}

```

所示的代码没有在执行事务之前保存error-handling和auto-commit的属性值，也没有在执行完毕后恢复。如果你将它们保存并还原，你的事务处理代码将会更具有通用性，因为它没有对程序中使用不同属性值的其他部分造成影响，但是这种做法需要更多行的代码。为了让事务处理更为简单（在你需要执行多个事务时避免额外代码重复），可以创建一些工具函数以处理在执行eval之前和之后的过程：

```

sub transaction_init
{
my $dbh = shift;
my $attr_ref = {}; # 创建混合表以保存属性

$attr_ref->{RaiseError} = $dbh->{RaiseError};
$attr_ref->{PrintError} = $dbh->{PrintError};
$attr_ref->{AutoCommit} = $dbh->{AutoCommit};
$dbh->{RaiseError} = 1; # 如果错误发生则抛出异常
$dbh->{PrintError} = 0; # 不打印出错信息
$dbh->{AutoCommit} = 0; # 取消自动提交
return ($attr_ref); # 返回属性给调用者
}

sub transaction_finish
{
my ($dbh, $attr_ref, $error) = @_;

if ($error) # 错误发生
{
    print "Transaction failed, rolling back. Error was:\n$error\n";
    # 在eval中回滚以防止回滚失败而停止脚本运行
    eval { $dbh->rollback (); };
}
# 恢复错误处理和自动提交属性
$dbh->{AutoCommit} = $attr_ref->{AutoCommit};
$dbh->{PrintError} = $attr_ref->{PrintError};

```

```
$dbh->{RaiseError} = $attr_ref->{RaiseError};  
}
```

通过使用这两个函数，我们的示例事务可以在相当程度上被简化：

```
$ref = transaction_init ($dbh);  
eval  
{  
    # 把一些钱从一个人的移动给另一个人  
    $dbh->do ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'");  
    $dbh->do ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'");  
    # 所有语句执行成功；提交事务  
    $dbh->commit ();  
};  
transaction_finish ($dbh, $ref, $@);
```

在Perl DBI中，手动处理AutoCommit属性的另一种替代方法为通过调用begin_work()来开始事务。该方法关闭了AutoCommit并在你调用commit()或rollback()后自动再次激活该属性。

15.5 在Ruby程序中使用事务

Using Transactions in Ruby Programs

问题

你需要在Ruby DBI脚本中执行事务。

解决方案

使用标准DBI事务支持机制，实际上，Ruby提供了两种机制。

讨论

Ruby DBI模块提供了两种执行事务的方式，尽管它们都依赖于自动提交模式的操作。一种方法是使用begin/rescue块，并显式地调用commit和rollback方法：

```
begin  
    dbh['AutoCommit'] = false  
    dbh.do("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'")  
    dbh.do("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'")  
    dbh.commit  
    dbh['AutoCommit'] = true  
rescue DBI::DatabaseError => e  
    puts "Transaction failed"  
    puts "#{e.err}: #{e.errstr}"  
    begin          # 在回滚失败的情形下清空异常处理句柄  
        dbh.rollback  
        dbh['AutoCommit'] = true  
    rescue  
    end  
end
```

Ruby还支持另一种事务方法，该方法根据代码块的执行成功与否自动地提交或回滚事务：

```
begin
  dbh['AutoCommit'] = false
  dbh.transaction do |dbh|
    dbh.do("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'")
    dbh.do("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'")
  end
  dbh['AutoCommit'] = true
rescue DBI::DatabaseError => e
  puts "Transaction failed"
  puts "#{e.err}: #{e.errstr}"
  dbh['AutoCommit'] = true
end
```

使用这种事务方法，不需要显式地调用commit或rollback，如果事务回滚则会产生一个异常，因此例子中仍使用了begin/rescue块来检测错误。

15.6 在PHP程序中使用事务

Using Transactions in PHP Programs

问题

你需要在PHP脚本中执行事务。

解决方案

使用标准的PEAR DB事务支持机制。

讨论

PEAR DB模块支持事务术语，可以被用于执行事务。使用autoCommit()方法可以关闭auto-commit模式。然后，在执行语句之后，调用commit()或rollback()以提交或回滚事务。

下面的代码使用异常来标记事务错误，该方法需要PHP 5的支持。（早先的PHP版本不支持异常。）PEAR DB事务处理方法本身在发生错误时不产生异常，因此示例程序在try块中使用状态检查来确定何时产生自己的异常：

```
try
{
  $result =& $conn->autoCommit (FALSE);
  if (PEAR::isError ($result))
    throw new Exception ($result->getMessage ());
  $result =& $conn->query (
```

```

        "UPDATE money SET amt = amt - 6 WHERE name = 'Eve'");
if (PEAR::isError ($result))
    throw new Exception ($result->getMessage ());
$result =& $conn->query (
        "UPDATE money SET amt = amt + 6 WHERE name = 'Ida'");
if (PEAR::isError ($result))
    throw new Exception ($result->getMessage ());
$result =& $conn->commit ();
if (PEAR::isError ($result))
    throw new Exception ($result->getMessage ());
$result =& $conn->autoCommit (TRUE);
if (PEAR::isError ($result))
    throw new Exception ($result->getMessage ());
}
catch (Exception $e)
{
    print ("Transaction failed: " . $e->getMessage () . ".\n");
    # 在回滚失败的情况下清空异常处理
    try
    {
        $conn->rollback ();
        $conn->autoCommit (TRUE);
    }
    catch (Exception $e2) { }
}

```

15.7 在Python程序中使用事务

Using Transactions in Python Programs

问题

你需要在DB-API脚本中执行事务。

解决方案

使用标准的DB-API事务支持机制。

讨论

Python的DB-API术语在连接对象方法中提供了事务处理控制。DB-API规范说明数据库连接开始时需要关闭自动提交模式。因此，当你打开通向数据库服务器的连接时，MySQLdb会关闭自动提交模式，即意味着开始了一个事务。每个事务以commit()或rollback()结束，commit()调用发生在try语句中， rollback()产生于except子句中以在错误发生时取消事务：

```

try:
    cursor = conn.cursor ()
    # 把一些钱从一个人的移动给另一个人
    cursor.execute ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'");
    cursor.execute ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'");

```

```
cursor.close()
conn.commit()
except MySQLdb.Error, e:
    print "Transaction failed, rolling back. Error was:"
    print e.args
try:    # 在回滚失败的情况下清空异常处理
    conn.rollback()
except:
    pass
```

15.8 在Java程序中使用事务

Using Transactions in Java Programs

问题

你需要在JDBC应用中执行事务。

解决方案

使用标准的JDBC事务支持机制。

讨论

在Java中执行事务，可以使用连接对象来关闭自动提交模式。然后，在执行语句之后，使用对象的commit()方法以提交事务或rollback()方法来取消它。典型情况下，在try块中执行事务语句，其中在块的末尾使用commit()。为了处理错误，在相应的异常处理块中调用rollback()：

```
try
{
    conn.setAutoCommit (false);
    Statement s = conn.createStatement ();
    // 把一些钱从一个人的移动给另一个人
    s.executeUpdate ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'");
    s.executeUpdate ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'");
    s.close ();
    conn.commit ();
    conn.setAutoCommit (true);
}
catch (SQLException e)
{
    System.err.println ("Transaction failed, rolling back.");
    Cookbook.printErrorMessage (e);
    // 在回滚失败的情况下清空异常处理
    try
    {
        conn.rollback ();
        conn.setAutoCommit (true);
    }
}
```

```
        catch (Exception e2) { }  
    }
```

15.9 使用事务的替代方法

Using Alternatives to Transactions

问题

你需要执行事务处理，但是你的应用中使用了无事务的存储引擎。

解决方案

一些类似事务的操作，如显式地锁定表可以起到弥补无事务环境的作用。在某些情况下，你甚至或许并不需要事务，可以通过重写你的语句，来彻底地消除对事务的需求。

讨论

事务是有价值的，但是有时候它们不能或者不必要使用：

- 你的应用中使用的存储引擎可能不支持事务。举个例子，如果你使用MyISAM表，你不能使用事务因为MyISAM存储引擎是非事务化的。每个MyISAM表上的更新无需提交就能够立即完成，并且不能被回滚。在这种情况下，你没有选择，只能使用类似事务的方法。一种在一些条件下起作用的策略是直接使用外部表锁来防止并发性问题。
- 有时候应用中使用的事务实际上是不必要的。你能够通过改写语句来消除对事务的需要，这可能会导致应用执行速度变快。

使用锁来聚合语句

如果你正使用无事务的存储引擎，但是你又需要执行一组语句，并且不被其他客户干扰，你可以通过使用LOCK TABLE和UNLOCK TABLE来实现（注1）：

1. 使用LOCK TABLE来获得对所有将要使用的表的锁。（对于要被修改的表需要写锁，而其他的需要读锁。）这阻止了其他客户在使用它们的同时修改这些表。
2. 所发布的语句必须在组中执行。
3. 使用UNLOCK TABLE来释放锁，其他客户端将重新获得对这些表的访问权。

注1：LOCK TABLES和UNLOCK TABLES与LOCK TABLE和UNLOCK TABLE的意思相同。

使用LOCK TABLE获得的锁一直到释放它们之前都保持作用，因而可以适用于多个语句执行的整个过程。这使你获得与事务相同的并发性益处。不过，如果错误发生，将不会有回滚操作，因此表锁定不是对于所有的应用都适合。举个例子，你或许尝试执行下面的操作，即资金从Eve转给Ida：

```
LOCK TABLE money WRITE;
UPDATE money SET amt = amt - 6 WHERE name = 'Eve';
UPDATE money SET amt = amt + 6 WHERE name = 'Ida';
UNLOCK TABLE;
```

不幸的是，如果第二个更新操作失败，第一个更新的影响没有被回滚。尽管有这个限制，在特定类型的情况下，表锁定对于应用的目标来说也已经是足够了：

- 一个只包含SELECT查询的语句集合。如果你需要执行几个SELECT语句并且需要在你查询这些表时阻止其他客户修改它们，那么锁住这些表就可以达到目的了。举个例子，如果你需要对一个表集合运行几个概要查询，如果在你查询期间允许其他客户端修改行的话，你的概要结果将会基于不同的数据集合，从而产生不一致。为了防止此事发生，需要锁定你要使用的表。
- 锁定对于只有最后一个语句是更新操作的语句集合也是有用的。在此情况下，前面的语句不对表产生任何改变。因而在更新失败时，不需要任何回滚操作。

重写语句以避免事务

有时候应用使用事务是不必要的。假设你有一个表meeting，其中记录了会议信息（包括每个会议剩余的门票数），并且你正在编写一个Ruby应用，其中包含分配门票的get_ticket()方法。实现此函数的一种方法是检查剩余票数，如果为正数则减一，并返回一个指示是否有可用票的状态。为了防止多个客户同时获取最后一张票，需要在事务中执行语句：

```
def get_ticket(dbh, meeting_id)
  count = 0
  begin
    dbh['AutoCommit'] = false
    # 检查当前的票计数
    row = dbh.select_one("SELECT tix_left FROM meeting
                          WHERE meeting_id = ?",
                          meeting_id)
    count = row[0]
    # 如果有票剩余，则减少票的计数
    if count > 0
      dbh.do("UPDATE meeting SET tix_left = tix_left-1
              WHERE meeting_id = ?",
              meeting_id)
```

```

    end
    dbh.commit
    dbh['AutoCommit'] = true
rescue DBI::DatabaseError => e
    count = 0          # 如果有错误发生，则没有tix可用
begin           # 在回滚失败的情况下清空异常处理
    dbh.rollback
    dbh['AutoCommit'] = true
rescue
end
end

return count > 0
end

```

该方法正确地分配了门票，但是包含了一些不必要的多余工作。事实上如果自动提交模式被激活，可以不使用事务而完成同样的事情。即只是在票数大于0时将之减一，然后检查语句是否影响了一行：

```

def get_ticket(dbh, meeting_id)
    count = dbh.do("UPDATE meeting SET tix_left = tix_left-1
                    WHERE meeting_id = ? AND tix_left > 0",
                    meeting_id)
    return count > 0
end

```

在MySQL中，UPDATE语句返回的行计数指示了有多少行发生了变化。这表示如果某个会议中没有剩票了，那么UPDATE将不会改变该行，此时返回的计数值为0。这使得通过单个语句来确定是否有票可用变得很简单，而不需要使用基于事务的多个语句。这个例子告诉我们尽管事务是重要并有其存在位置的，但是你仍然可能避免使用它们，结果可以获得更快的应用速度。单语句的解决方式是MySQL参考手册中所指的“原子操作”的一个例子，该手册将它们作为事务的有效替代物进行了讨论。

使用存储例程、触发器和事件

Using Stored Routines, Triggers, and Events

16.0 引言

Introduction

本章讨论下面几种数据库对象：

存储例程（函数和过程）

存储函数执行计算并返回值，比返回值可以用于表达式中，用法与内建函数如RAND()、NOW()或LEFT()类似。而存储过程只执行不需要返回值的计算。过程不能用于表达式，而是在CALL语句中被调用。一个过程可以被执行来更新表中的行或产生发送到客户端程序的结果集。一个使用存储例程的理由是将执行计算的代码封装起来，这使你能够通过调用例程很容易地执行计算，而不用每次都重写相应代码。

触发器

触发器被定义为当表被修改时产生动作的对象。触发器可以用于INSERT、UPDATE和DELETE语句。举个例子，你可以在将值插入表中之前进行检查，或者指定哪些行在被从表中删除时应当被记入另一个作为数据变化日志的表中。触发器对于自动化这些行为是很有用处的，因为你不再需要记住在每次修改表时进行这些操作。

事件

一个事件是在预定时间执行SQL语句的对象。你可以将事件理解为类似于unix系统工作的时间单位的东西，当然后者不是在MySQL中运行的。举例来说，事件可以帮你执行管理任务，如周期性地删除旧表或创建每夜摘要。

存储例程和触发器在MySQL 5.0中得到支持，而从MySQL 5.1开始支持事件。

这些不同种类的对象具有共同的属性，即它们都是用户定义，但是存储在服务端以供日后执行。这与从客户端向服务端发送SQL语句并立即执行不同。这些对象还具有下面的属性：它们都是被定义为在对象被调用时执行其他SQL语句。对象的执行体为单个SQL语句，但是该语句可以使用包含多重语句的复合语句符号（如BEGIN...END块）。这表示执行体的范围从非常简单到特别复杂都是允许的。下面的存储过程是一个简单的例程，其中执行体只是由单个SET语句组成，以设置一个用户定义变量：

```
CREATE PROCEDURE get_time()
SET @current_time = CURTIME();
```

对于更复杂的操作，需要一个复合语句来实现：

```
CREATE PROCEDURE part_of_day()
BEGIN
    CALL get_time();
    IF @current_time < '12:00:00' THEN
        SET @day_part = 'morning';
    ELSEIF @current_time = '12:00:00' THEN
        SET @day_part = 'noon';
    ELSE
        SET @day_part = 'afternoon or night';
    END IF;
END;
```

这里，BEGIN ... END块包含多个语句，但是其本身被视为单语句。组合语句使你能够声明本地变量并使用条件逻辑和循环结构。还需要注意的是，一个存储过程可以调用另一个：part_of_day()调用了get_time()。与在非组合语句如SELECT或UPDATE中自己编写内联表达式相比较，这些能力为你提供了更高的可适应性，特别是对于算术表达式来说。

在复合语句中的每一个语句都需要以;来结束。如果你使用mysql客户端来定义使用复合语句的对象，那么这个限制会产生问题。因为mysql本身将;解释为确定语句的边界。解决此问题的方法是在定义复合语句对象时重新定义mysql的语句分隔符。16.1节中介绍了如何实现此操作，你必须保证先阅读过那一节后，才能进入其后的内容。

由于篇幅限制，本章通过例子来阐述，但是没有深入描述关于存储例程、触发器和事件的众多符号的细节。对于完整的符号描述，请参见《MySQL用户手册》。

本章例子所使用的脚本可以在routines、triggers和events目录中找到，创建表的语句位于tables目录。

除了本章所示的存储例程，还有一些可以在本书的其他部分找到。举例来说，如5.7、6.2和11.11小节。

存储例程、触发器和事件的权限

当你创建存储例程时，下面权限要求必须被满足，否则将会产生问题

- 要创建存储例程，必须具有CREATE ROUTINE权限。
- 如果MySQL服务器的二进制日志被激活（在实践中通常如此），那么在创建存储函数（不包括存储过程）时需要额外的要求。如果你使用二进制日志来复制或恢复备份，这些要求对于保证函数调用在初次执行和重新执行时产生同样的作用是必需的：
 - 你必须具有SUPER权限，并且必须使用DETERMINISTIC、NO SQL、或READS SQL DATA等特性将函数声明为确定性的或只读的。（也有可能创建非确定或者对数据进行修改的函数，但是它们在复制和备份时可能不是安全的。）
 - 作为另一种选择，你可以激活系统变量log_bin_trust_function_creators，如此服务器就会放弃前面的限制要求。

要在MySQL 5.0中创建触发器，需要SUPER权限。而在MySQL 5.1中，必须具有相关链表的TRIGGER权限。

要创建事件，必须具有相应数据库的EVENT权限。

16.1 创建复合语句对象

Creating Compound-Statement Objects

问题

你需要定义一个存储过程、触发器或者事件，其中使用了；作为语句结束符。该符号与mysql默认使用的结束符相同，因此mysql将会曲解此定义并产生一个错误。

解决方案

使用delimiter命令重新定义mysql语句结束符。

讨论

每个存储过程、触发器或事件是必须放入单个SQL语句中的对象。但是，这些对象经常会执

行需要几个语句的复杂操作。为了处理这种情况，你需要使用BEGIN...END块将语句封装起来以形成一个复合语句。也就是说，这个块本身是单语句，但其中包含了多个语句，每个语句以;结束。BEGIN...END块中可以包含SELECT或INSERT语句，还可以允许条件语句如IF或CASE，循环结构如WHILE或REPEAT，甚至可以再包含其他的BEGIN...END块。

复合语句语法提供了许多的灵活性，但如果你需要在mysql中定义复合语句，你将很快遇到一个小问题：复合语句中的每条语句都是以;字符结束的，而mysql本身将;解释为每个SQL语句结束的标识，从而一次性将该符号之前的语句发送至服务端执行。因此，mysql在看到第一个;字符时，就会停止读入复合语句，而这显然太早了。解决此问题的方法是让mysql使用另一个语句分界符，如此mysql将会忽略在对象体内遇到的;字符。你可以使用一个新的分界符来结束此对象，mysql在识别该分界符后，将整个对象定义发送至服务端。在定义复合语句对象之后，你还可以将mysql的分界符还原至原始值。

假设mail表中包含邮件消息，你需要定义一个计算并返回其字节数平均值的存储函数。该函数可以使用由单个SQL语句组成的语句体来定义，如下所示：

```
CREATE FUNCTION avg_mail_size()
RETURNS FLOAT READS SQL DATA
RETURN (SELECT AVG(size) FROM mail);
```

RETURNS FLOAT语句显示了该函数的返回类型，READS SQL DATA表明该函数指示读取而非修改数据。这些语句之后为函数体，其中包含了单个RETURN语句，该语句执行一个子查询并产生提供给调用者的返回值。（每个存储函数必须至少含有一个RETURN语句。）

在mysql中，你可以键入上面的语句而不会产生任何问题。此定义只需要在末尾标注一个结束符，而在定义体中间不需要，因此不会产生二义性。但是假设你要定义包含参数user的函数，该参数的含义如下：

- 如果参数值为NULL，函数将返回所有消息的平均大小（与前面的一样）。
- 如果参数不为NULL，函数将返回user所发送消息的平均大小。

为了完成此任务，存储函数需要定义一个使用BEGIN...END块的复杂执行体：

```
CREATE FUNCTION avg_mail_size(user VARCHAR(8))
RETURNS FLOAT READS SQL DATA
BEGIN
    IF user IS NULL THEN
```

```

# 返回所有用户消息的平均大小
RETURN (SELECT AVG(size) FROM mail);
ELSE
    # 返回给定用户所发生消息的平均大小
    RETURN (SELECT AVG(size) FROM mail WHERE srcuser = user);
END IF;
END;

```

如果你试着通过输入上面的定义在mysql中定义此函数,mysql将会错误地将函数体中的第一个分号解释为定义结束标识。为了处理这种情况,需要使用delimiter命令来暂时性地修改mysql的分界符。下面的例子显示了如何实现此操作并在之后将分界符还原为默认值:

```

mysql> delimiter $$ 
mysql> CREATE FUNCTION avg_mail_size (user VARCHAR(8))
-> RETURNS FLOAT READS SQL DATA
-> BEGIN
->     IF user IS NULL THEN
->         # 返回所有用户消息的平均大小
->         RETURN (SELECT AVG(size) FROM mail);
->     ELSE
->         # 返回给定用户所发生消息的平均大小
->         RETURN (SELECT AVG(size) FROM mail WHERE srcuser = user);
->     END IF;
-> END;
-> $$ 
Query OK, 0 rows affected (0.02 sec)
mysql> delimiter ;

```

在定义该存储函数之后,你可以使用与内建函数同样的方式来调用它:

```

mysql> SELECT avg_mail_size(NULL), avg_mail_size('barb');
+-----+-----+
| avg_mail_size(NULL) | avg_mail_size('barb') |
+-----+-----+
|      237386.5625 |          52232 |
+-----+-----+

```

上述原则在定义使用复合语句的其他对象(存储过程、触发器和事件)时也适用。

16.2 使用存储函数封装计算

Using a Stored Function to Encapsulate a Calculation

问题

一个产生特定值的计算需要在不同的应用中多次执行,但是你不想在每次需要的时候都写出该表达式。或者,计算十分复杂,需要条件或循环逻辑,以致于不能在表达式中内联执行。

解决方案

使用存储函数来隐藏讨厌的计算细节并使之更易执行。

讨论

存储函数能够简化你的应用，因为它只需要在函数定义中编写一次产生计算结果的代码，然后可以在任何需要执行计算的时候简单地调用该函数。存储函数还能够使用比在表达式中的内联计算更为复杂的算术结构。本节显示的例子说明了存储函数在这些方面是如何使用的。当然，这里的例子实际上并不复杂，但是你可以以同样的原理来编写更精细的函数。

美国不同的州具有不同的销售税率。如果你向不同州的客户销售货物，那么必须按照客户所在地的税率缴税，你需要为每笔销售计算税费。你可以利用一个列出各州销售税率的表和一个根据给定销售额和所在州计算应纳税总数的存储函数来处理这个问题。

要创建该表，可以使用本节tables目录中的sales_tax_rate.sql脚本。sales_tax_rate表具有两列：state（二字字母简写）和tax_rate（值为DECIMAL而不是FLOAT，以保持精确度）。

存储函数sales_tax()如下定义：

```
CREATE FUNCTION sales_tax(state_param CHAR(2), amount_param DECIMAL(10,2))
RETURNS DECIMAL(10,2) READS SQL DATA
BEGIN
    DECLARE rate_var DECIMAL(3,2);
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET rate_var = 0;
    SELECT tax_rate INTO rate_var
        FROM sales_tax_rate WHERE state = state_param;
    RETURN amount_param * rate_var;
END;
```

函数检查给定州的税率，并返回应缴税费，即销售额与税率的乘积。

假设Vermont和New York的税率分别为1%和9%，使用此函数查看对于销售额为100美元时，税费是否得到正确计算：

```
mysql> SELECT sales_tax('VT',100.00), sales_tax('NY',100.00);
+-----+-----+
| sales_tax('VT',100.00) | sales_tax('NY',100.00) |
+-----+-----+
|           1.00 |             6.00 |
+-----+-----+
```

对于没有在税率表中列出的地点，函数将会失败并将税率和所计算的税费都设为0：

```
mysql> SELECT sales_tax('ZZ',100.00);
+-----+
| sales_tax('ZZ',100.00) |
+-----+
|           0.00 |
+-----+
```

显然地，如果你的销售地没有在该表中列出，那么函数不能确定相应的税率。在此情况下，函数假设税率为0%，这通过CONTINUE处理器来实现，如果No Data条件发生（SQLSTATE值为02000），那么CONTINUE处理器会跳过。也即是说，如果对于给定的state_param值不存在相应的行，那么查找销售税率的SELECT语句将会失败。这种情况下，CONTINUE处理器将税率设为0并继续执行SELECT后的下一条语句。（此处理器的用例又一次说明了你可以在存储例程中实现某些逻辑，而在内联表达式中是无法使用的。）

16.3 使用存储过程来“返回”多个值

Using a Stored Procedure to "Return" Multiple Values

问题

你需要执行产生两个或更多值的操作，但是存储函数只能返回一个值。

解决方案

使用带有OUT或INOUT参数的存储过程，并在调用过程时将用户定义变量传入这些参数。过程并不像函数那样“返回”值，但是它可以将值赋给这些参数，从而在过程返回时，对应的变量中会含有这些值。

讨论

存储函数的参数只是用于输入值，与之不同的是，存储过程的参数可以有下面三种类型：

- IN参数是仅作为输入使用的，如果你没有指定类型，那么它将作为默认参数类型。
- INOUT参数即可以传入一个值，也可以传回一个值。
- OUT参数只被用于传回值。

这表示如果你需要在一个操作中产生多个值，你可以使用INOUT或OUT参数。下面的例子说明了此点，即使用IN参数作为输入，并通过OUT参数来传回3个值。

16.1节显示了一个`avg_mail_size()`函数返回了给定发送者的信件消息的平均长度。该函数只返回一个值。如果你需要更多的信息，比如消息的数量和消息的长度总和，那么单个函数无法起作用。你可以编写3个独立的函数，但是使用单个过程来获取给定发送者的多个值也是可能的。下面的过程`mail_sender_stats()`运行了一个在`mail`表中的查询以获取给定用户名（作为输入值）的邮件发送统计信息。此过程确定了用户发送了多少消息，以及消息的总和与平均长度（以字节为单位），这些通过3个OUT参数来返回：

```
CREATE PROCEDURE mail_sender_stats(IN user VARCHAR(8),
                                    OUT messages INT,
                                    OUT total_size FLOAT,
                                    OUT avg_size FLOAT)
BEGIN
    # 在用户没有数据行的情况下使用IFNULL()来为SUM()和AVG()返回0
    # (在该情形下函数返回NULL)
    SELECT COUNT(*), IFNULL(SUM(size), 0), IFNULL(AVG(size), 0)
    INTO messages, total_size, avg_size
    FROM mail WHERE srcuser = user;
END;
```

为了使用过程，需要传入包含用户名的字符串，以及3个接收OUT值的用户定义变量。在过程返回后，检查变量值：

```
mysql> CALL mail_sender_stats('barb',@messages,@total_size,@avg_size);
mysql> SELECT @messages, @total_size, @avg_size;
+-----+-----+-----+
| @messages | @total_size | @avg_size |
+-----+-----+-----+
| 3         | 156696      | 52232      |
+-----+-----+-----+
```

16.4 用触发器来定义动态的默认列值

Using a Trigger to Define Dynamic Default Column Values

问题

表中的一列需要被初始为一个非常数的值，但是MySQL只运行常数的默认值。

解决方案

使用`BEFORE INSERT`触发器，它使你能够以一个任意表达式产生的值初始化列。换句话说，触发器通过计算默认值执行动态的列初始化。

讨论

除了可以根据当前日期和时间初始化的TIMESTAMP列，MySQL中列的默认值必须为常数值。你不能使用DEFAULT子句定义指向函数调用（或其他任意表达式）的列，并且也不能定义根据其他列赋初值的列。这表示下面的每个列定义都是不合法的：

```
d      DATE DEFAULT NOW()
i      INT DEFAULT (... some subquery ...)
hashval CHAR(32), DEFAULT MD5(blob_col)
```

但是，你可以通过创建合适的触发器来绕开此限制，这使你能够根据需要初始化列。触发器能够有效地定义动态（或者说通过计算得出）的默认列值。

适用此需求的触发器类型为BEFORE INSERT，它使你能够在向表中插入数据之前设置列值。（AFTER INSERT触发器可以检查一个新行的列值，但是到此触发器激活时，改变值已经太迟了。）

假设你需要使用表来存储大型数据值如PDF或XML文档、图片或声音等，并且你还需要以后能够快速查找它们。如此TEXT或者BLOB数据类型不太适合存储这些值，因为它们并不适于对其进行查找。（对大型数据查找中的比较操作将会十分缓慢。）为了解决此问题，可以使用下面的策略：

1. 为每个数据值计算某种hash值并与数据一起存入表中。
2. 查找行中是否包含了特定的数据值，可以计算该值的hash值，然后在表中搜索相应的hash值。为了更好的性能，可以对hash列进行索引。

为了实现此策略，BEFORE INSERT触发器是很有帮助的，因为你在表中存储新的数据值之前使用它来计算hash值。（如果你将来可能会改变数据值，同样需要建立BEFORE UPDATE触发器来重新计算hash值。）

下面的例子假设你需要在表中存储文档，包括文档作者和标题。此外，表中还包含一个存储根据文档内容计算的hash值的列。本例使用MD5()函数来生成hash值，该函数返回一个32位16进制字符串，这对用于比较的值来说显得仍有些长。但是，相对于基于存储整个文档内容的列进行比较，它已经短得多了。

首先，创建表来放置文档信息，然后根据文档内容计算hash值。下表使用MEDIUMBLOB列以允许存储的文档最大为16MB：

```
CREATE TABLE doc_table
(
    author  VARCHAR(100) NOT NULL,
    title   VARCHAR(100) NOT NULL,
    document MEDIUMBLOB NOT NULL,
    doc_hash CHAR(32) NOT NULL,
    PRIMARY KEY (doc_hash)
);
```

下一步为处理插入，先创建BEFORE INSERT触发器，使用要被插入的文档来计算hash值并将其存储在表中：

```
CREATE TRIGGER bi_doc_table BEFORE INSERT ON doc_table
FOR EACH ROW SET NEW.doc_hash = MD5(NEW.document);
```

此触发器很简单，它的执行体只包含单个SQL语句。对于在执行体中需要执行多个语句的触发器，可以使用BEGIN ... END复合语句符号。在此情况下，如果你使用mysql来创建事件，你需要在定义触发器时改变语句分割符，如16.1节中讨论的那样。

在触发器中，`NEW. col_name`指向一个将被插入到指定列的新值。通过在触发器中将值赋给`NEW. col_name`，使得新行中的对应列具有该值。

最后，插入新行并检查触发器是否正确地初始化该文档的hash值：

```
mysql> INSERT INTO doc_table (author,title,document)
-> VALUES ('Mr. Famous Writer','My Life as a Writer',
->           'This is the document');
mysql> SELECT * FROM doc_table\G;
***** 1. row *****
author: Mr. Famous Writer
title: My Life as a Writer
document: This is the document
doc_hash: 5282317909724f9f1e65318be129539c
mysql> SELECT MD5('This is the document');
+-----+
| MD5('This is the document') |
+-----+
| 5282317909724f9f1e65318be129539c |
+-----+
```

第一个SELECT显示，即使INSERT没有提供值，`doc_hash`列仍会被初始化。第二个SELECT显示了触发器正确执行后该行存储的hash值。

这个例子示范了触发器如何初始化一个行中的列，这种方式超过了在列定义中使用`DEFAULT`子句所能实现的功能。同样的思路可以应用于更新操作，将此方法应用到下面的情景是个好主意：当一个列的初始化值为另一个列值的函数时（比如本例中的`doc_hash`），即表示它依赖于该列。因此，在它所依赖的列发生任何变化的时候，都应该更新该列。举例来说，

如果你更新了document列的值，你同样应该更新相应的doc_hash值，这也可以通过触发器来实现。创建一个BEFORE UPDATE触发器，其中的操作与INSERT触发器一样：

```
CREATE TRIGGER bu_doc_table BEFORE UPDATE ON doc_table  
FOR EACH ROW SET NEW.doc_hash = MD5(NEW.document);
```

可以通过更新文档值并检查hash值是否被正确更新来测试UPDATE触发器：

```
mysql> UPDATE doc_table SET document = 'A new document'  
      -> WHERE document = 'This is the document';  
mysql> SELECT * FROM doc_table\G;  
***** 1. row *****  
author: Mr. Famous Writer  
title: My Life as a Writer  
document: A new document  
doc_hash: 21c03f63d2f01b598665d4d960f3a4f2  
mysql> SELECT MD5('A new document');  
+-----+  
| MD5('A new document') |  
+-----+  
| 21c03f63d2f01b598665d4d960f3a4f2 |  
+-----+
```

16.5 为其他日期和时间类型模拟TIMESTAMP属性

Simulating TIMESTAMP Properties for Other Date and Time Types

问题

TIMESTAMP数据类型提供了自动初始化和自动更新的属性，你需要为其他的时间数据类型使用这些属性，但是其他类型的初始化只允许常量值，并且它们不是自动更新的。

解决方案

使用INSERT触发器在记录创建时提供合适的当前日期或时间值。当行发生变化时，使用UPDATE触发器将该列更新到当前日期或时间。

讨论

6.5节描述了TIMESTAMP 数据类型特有的初始化和更新属性，即能够在记录行被创建时自动修改时间。这些属性对于其他时间类型是不可用的，尽管你可能出于某些原因希望它们能够具有这些属性。举个例子，如果你使用分开的DATE和TIME列来存储记录修改时间，则可以对DATE列进行索引以提高日期查询的效率。（使用TIMESTAMP，你不能仅对列的日期部分进行索引。）

一种为其他日期数据类型模拟TIMESTAMP属性的方法是使用下面的策略：

- 当创建一行时，将DATE列初始化为当天的日期，同时将TIME列设为当前时间。
- 当更新一行时，将DATE和TIME列设为新的日期和时间。

但是，此策略需要所有使用此表的应用都采用同样的策略，即使只有一个应用漏做了此工作也会导致失败。为了将记忆设置列属性的负担交给MySQL服务器而不是应用开发者，需要对表使用触发器。实际上，这是16.4节中讨论的通用策略的一个特殊应用，即使用触发器来为初始化（或更新）行中的列提供计算得到的值。

下面的例子显示了如何使用触发器为每个DATE、TIME或DATETIME数据类型模拟TIMESTAMP属性。首先创建下表，该表含有一个存储数据的非时间列以及DATE、TIME和DATETIME等时间类型的列：

```
CREATE TABLE ts_emulate
(
    data CHAR(10),
    d   DATE,
    t   TIME,
    dt  DATETIME
);
```

这里的目的是当应用插入或更新数据列中的值时，MySQL应该将适当地设置时间列以反应修改发生的时间。要完成此目标，需要创建使用当前日期和时间来为新行初始化时间列的触发器，并且当行发生改变时更新它们。BEFORE INSERT触发器在新行创建时，通过调用CURDATE()、CURTIME()和NOW()函数来获取当前的时间、日期或时间与日期值，并使用这些值来设置时间列：

```
CREATE TRIGGER bi_ts_emulate BEFORE INSERT ON ts_emulate
FOR EACH ROW SET NEW.d = CURDATE(), NEW.t = CURTIME(), NEW.dt = NOW();
```

当数据列的值发生变化时，BEFORE UPDATE触发器处理对时间列的更新。这里需要IF语句来仿效TIMESTAMP属性，即更新只会在行中的值从当前值发生实际变化时产生：

```
CREATE TRIGGER bu_ts_emulate BEFORE UPDATE ON ts_emulate
FOR EACH ROW
BEGIN
    # 当且仅当非临时列有变化时才更新临时列
    IF NEW.data <> OLD.data THEN
        SET NEW.d = CURDATE(), NEW.t = CURTIME(), NEW.dt = NOW();
    END IF;
END;
```

为了测试INSERT触发器，可以创建两个行，但是只为数据列提供值，然后检查MySQL是否为每个时间列提供了合适的默认值：

```
mysql> INSERT INTO ts_emulate (data) VALUES('cat');
mysql> INSERT INTO ts_emulate (data) VALUES('dog');
mysql> SELECT * FROM ts_emulate;
+-----+-----+-----+-----+
| data | d   | t       | dt      |
+-----+-----+-----+-----+
| cat  | 2006-06-23 | 13:29:44 | 2006-06-23 13:29:44 |
| dog  | 2006-06-23 | 13:29:49 | 2006-06-23 13:29:49 |
+-----+-----+-----+-----+
```

改变一行的数据值以检验BEFORE UPDATE触发器是否更新了产生变化行的时间列：

```
mysql> UPDATE ts_emulate SET data = 'axolotl' WHERE data = 'cat';
mysql> SELECT * FROM ts_emulate;
+-----+-----+-----+-----+
| data | d   | t       | dt      |
+-----+-----+-----+-----+
| axolotl | 2006-06-23 | 13:30:12 | 2006-06-23 13:30:12 |
| dog   | 2006-06-23 | 13:29:49 | 2006-06-23 13:29:49 |
+-----+-----+-----+-----+
```

再执行另一个UPDATE，但是这一次更新不改变数据列的值。在此情况下，BEFORE UPDATE触发器注意到没有值发生变化，并保持时间列不变：

```
mysql> UPDATE ts_emulate SET data = data;
mysql> SELECT * FROM ts_emulate;
+-----+-----+-----+-----+
| data | d   | t       | dt      |
+-----+-----+-----+-----+
| axolotl | 2006-06-23 | 13:30:12 | 2006-06-23 13:30:12 |
| dog   | 2006-06-23 | 13:29:49 | 2006-06-23 13:29:49 |
+-----+-----+-----+-----+
```

前面的例子显示了如何模拟TIMESTAMP列提供的自动初始化和自动更新属性。如果你只需要使用这些属性中的一个，可以只创建一个触发器而忽略其他的。

16.6 使用触发器记录表的变化

Using a Trigger to Log Changes to a Table

问题

你有一个表，其中包含你要跟踪物品的当前价值（比如被竞价的拍卖品），并且你还需要记录该表的变化过程（历史日志）。

解决方案

使用触发器来“捕获”表的变化并将它们写入另一个日志表。

讨论

假设你在管理一个在线拍卖行，并且需要在表中维持每个拍卖品的信息，如下所示：

```
CREATE TABLE auction
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    ts      TIMESTAMP,
    item   VARCHAR(30) NOT NULL,
    bid    DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (id)
);
```

auction表包含当前活动拍卖品的信息（包括要拍卖的物品和每个拍卖品的当前竞价）。当一个拍卖开始时，需要向表中键入一行。表的bid列在每次该物品产生新的出价时更新。当拍卖结束后，bid值为最后的成交价并且该行将从表中删除。在拍卖过程中，ts列不断更新以反映最近一次出价的时间。

如果你需要维护排名过程中该表所有的创建和删除行的变化，你可以修改auction表，允许每个物品有多条记录并且增加一个状态列以显示每行所代表的拍卖类别。或者你可以保持auction不变并创建另一个表来记录拍卖的变化历史信息。第二种策略可以使用触发器来实现。

为了保持每个拍卖过程的历史，使用包含下面列的auction_log表：

```
CREATE TABLE auction_log
(
    action ENUM('create','update','delete'),
    id      INT UNSIGNED NOT NULL,
    ts      TIMESTAMP,
    item   VARCHAR(30) NOT NULL,
    bid    DECIMAL(10,2) NOT NULL,
    INDEX  (id)
);
```

auction_log表与auction表相比，存在两方面的不同：

- 它包含action列以指明每行记录的是哪一种变化。

- `id`列具有不唯一的索引（与限制值唯一的primary key不同）。这允许每个`id`值对应多个行，因为一个给定的拍卖品可能在日志表中产生许多行。

为了保证表的变化被记录到

- 对于插入，记录row-creation操作并在新行中显示该值。
- 对于更新，记录row-update操作并在被更新的行中显示新值。
- 对于删除，记录row-removal操作显示被删除行的值。

对于这个应用，使用了AFTER触发器，因为它们只是在表变化成功之后被激活。（BEFORE触发器即使在行变化操作由于某些原因失败时也可能被激活。）触发器定义如下：

```
CREATE TRIGGER ai_auction AFTER INSERT ON auction
FOR EACH ROW
BEGIN
    INSERT INTO auction_log (action,id,ts,item,bid)
    VALUES('create',NEW.id,NOW(),NEW.item,NEW.bid);
END;

CREATE TRIGGER au_auction AFTER UPDATE ON auction
FOR EACH ROW
BEGIN
    INSERT INTO auction_log (action,id,ts,item,bid)
    VALUES('update',NEW.id,NOW(),NEW.item,NEW.bid);
END;

CREATE TRIGGER ad_auction AFTER DELETE ON auction
FOR EACH ROW
BEGIN
    INSERT INTO auction_log (action,id,ts,item,bid)
    VALUES('delete',OLD.id,OLD.ts,OLD.item,OLD.bid);
END;
```

`INSERT`和`UPDATE`触发器使用`NEW.col_name`来访问将被存储在行中的新值。`DELETE`触发器使用`OLD.col_name`来访问被删除行中的旧值。`INSERT`和`UPDATE`触发器使用`NOW()`来获取行被修改的时间，虽然`ts`列使用当前的日期和时间自动初始化，但是`New.ts`不包含这个值。

假设一个拍卖行被创建，并且初始的出价为5美元：

```
mysql> INSERT INTO auction (item,bid) VALUES ('chintz pillows',5.00);
mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|      792      |
+-----+
```

SELECT语句获取auction ID值，并在随后的拍卖操作中使用，然后该物品在拍卖结束并被移除之前，接收到3个更高的报价：

```
mysql> UPDATE auction SET bid = 7.50 WHERE id = 792;
... time passes ...
mysql> UPDATE auction SET bid = 9.00 WHERE id = 792;
... time passes ...
mysql> UPDATE auction SET bid = 10.00 WHERE id = 792;
... time passes ...
mysql> DELETE FROM auction WHERE id = 792;
```

此时，在auction表中没有留下任何拍卖的痕迹，但是如果你查询auction_log表，就可以获得所发生的完整的记录：

```
mysql> SELECT * FROM auction_log WHERE id = 792 ORDER BY ts;
+-----+-----+-----+-----+-----+
| action | id   | ts           | item      | bid   |
+-----+-----+-----+-----+-----+
| create | 792  | 2006-06-22 21:24:14 | chintz pillows | 5.00  |
| update | 792  | 2006-06-22 21:27:37 | chintz pillows | 7.50  |
| update | 792  | 2006-06-22 21:39:46 | chintz pillows | 9.00  |
| update | 792  | 2006-06-22 21:55:11 | chintz pillows | 10.00 |
| delete | 792  | 2006-06-22 22:01:54 | chintz pillows | 10.00 |
+-----+-----+-----+-----+-----+
```

使用上面列出的策略，排名表会保持相对较小，但是我们总可以根据需要通过查找auction_log表找到拍卖品的历史信息。

16.7 使用事件调度数据库动作

Using Events to Schedule Database Actions

问题

你需要创建周期性运行的数据库操作，而不需要用户交互。

解决方案

创建根据计划执行的事件。

讨论

在MySQL 5.1中，一种可用的能力为事件调度，它能够帮助你建立根据自定义时间运行的数据库操作。本节描述了可以使用事件来完成的任务，并从一个定期向表中写入一行的简单事件开始。为什么需要创建这个事件？一个原因是表中的行作为连续工作的服务器操作的

日志，与某些Unixsyslogd服务中的MARK line类似，即周期性地写入系统日志以表示它们仍然处于活动状态。

首先创建包含mark记录的表。该表包含一个TIMESTAMP列（MySQL将自动对其初始化）和一个存储消息的列：

```
mysql> CREATE TABLE mark_log (ts TIMESTAMP, message VARCHAR(100));
```

我们的日志事件将向新行中写入字符串，可以使用CREATE EVENT语句来建立此事件：

```
mysql> CREATE EVENT mark_insert
    -> ON SCHEDULE EVERY 5 MINUTE
    -> DO INSERT INTO mark_log (message) VALUES ('-- MARK --');
```

mark_insert事件将使消息'-- MARK --'每隔5分钟被记录到mark_log表一次，可以使用不同的时间间隔来提高或降低日志的频率。

此事件很简单，其执行体只包含单个SQL语句。对于需要执行多个语句的事件执行体，可以使用BEGIN ... END复合语句符号。在那种情况下，如果你使用mysql来创建事件，则需要在定义事件时改变语句分隔符，如16.1节中讨论的那样。

这个例子中，你需要等待几分钟，然后选取mark_log表的内容来校验调度中刚被写入的新行。不过，如果这是你创建的第一个事件，你或许会发现该表始终为空，无论你等待了多长时间：

```
mysql> SELECT * FROM mark_log;
Empty set (0.00 sec)
```

如果发生了那种情况，很可能事件调度器并没有运行（除非你激活它，否则这是默认状态）。可以通过检查系统变量event_scheduler的值来检测调度器的状态：

```
mysql> SHOW VARIABLES LIKE 'event_scheduler';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| event_scheduler | 0      |
+-----+-----+
```

如果调度器没有运行，可以手动激活它，即执行下面的语句（这需要SUPER权限）：

```
mysql> SET GLOBAL event_scheduler = 1;
```

该语句激活了调度器，但是只是到服务器宕机为止。为了确保每次服务器启动时，调度器都会运行，需要在my.cnf选项文件中将此系统变量设置为1：

```
[mysqld]
event_scheduler=1
```

当事件调度器被激活时，`mark_insert`事件最终会在表中创建过多的行。为了防止表一直膨胀下去，可以使用几种方法来影响事件的执行：

- 删除事件：

```
mysql> DROP EVENT mark_insert;
```

这是停止事件发生的最简单的方法。但是如果你以后还想继续执行，则必须重新创建它了。

- 悬挂事件的执行：

```
mysql> ALTER EVENT mark_insert DISABLE;
```

仍然保留事件，但是使其失效而不再运行，直到你重新激活它为止：

```
mysql> ALTER EVENT mark_insert ENABLE;
```

- 让事件继续执行，但是建立另一个事件来“终止”旧的`mark_log`行。此第二个事件不需要经常运行（可能一天运行一次）。它的执行体应当包含`DELETE`语句以删除比给定时间阈值更旧的行。下面的定义创建的事件将删除超过两天的行：

```
mysql> CREATE EVENT mark_expire
      -> ON SCHEDULE EVERY 1 DAY
      -> DO DELETE FROM mark_log WHERE ts < NOW() - INTERVAL 2 DAY;
```

如果你采用这种策略，你将持有两个互相协作的事件，其中一个向`mark_log`表增加行而另一个删除它们。这两个事件共同维护包含近期记录的日志，并保证它所占空间不会太大。

要检查事件是否处于活动状态，可以查看服务器的错误日志，其中记录了哪个事件被被执行及其执行的时间等信息。

关于Web应用中MySQL的介绍

Introduction to MySQL on the Web

17.0 引言

Introduction

本章以及下一章的部分内容讨论了MySQL帮助你构建一个较好的Web站点的一些方法。其中一个重要的好处是MySQL能够使你创建一个交互性更强的站点，因为它提供动态内容比提供静态内容更加简单。静态内容在Web服务器的文档树中作为页面存在。访问者只能访问放置在树上的文档，并且只有当你增加、修改或删除这些文档时才会发生变化。与此对比，动态内容是根据需要来创建的。与打开一个文件并将其内容直接提供给客户端相比，这里，Web服务器是执行一个产生页面的脚本并发送结果输出。例如，一个脚本能够处理关键字请求，并返回一个在一览表中列出和关键字匹配的项目的页面。每当提交一个关键字，脚本为请求产生合适的结果。而且这些都是对于起始者而言的，Web脚本具有和书写它们的编程语言近似的能力，所以它们执行的用来产生页面的动作具有相当的广泛性。例如，Web脚本对于表单处理非常重要，一个单独的脚本可能负责产生一个表单并将其发送给用户，在稍后用户提交表单时处理表单的内容以及在数据库中存储内容。通过这种方法，Web脚本和你的Web站点的用户进行交互，并根据用户需要裁剪所提供的信息。

本章覆盖了在Web环境中使用MySQL编写脚本的介绍性内容。有一些材料不是MySQL相关的，但是对于建立在Web编程的内容中使用数据库的初步基础来说是必不可少的。这里覆盖的主题包括：

- Web脚本与编写静态HTML文档或与通过命令行执行的脚本相比有何不同。
- 执行Web脚本的先决条件。特别地，你必须安装一个Web服务器，并且将它设置成能够把你的脚本作为将要执行的程序，而不是作为在网络上照字面意义提供的文件。

- 如何使用我们的每一个API语言来编写简短的Web脚本，这些脚本用以查询MySQL服务器并将结果显示在Web页面上。
- 如何合适地进行编码输出。HTML由将要显示的文本以及散布的专用标签构成。然而，如果文本中包含特殊的字符，你必须对它们进行编码来避免产生异常的Web页面。每个API为此提供一种机制。

余下的章节将对一些主题详细讲述，比如如何通过不同的格式（段落、列表、表格等）显示查询结果、使用图像、表单处理以及通过作为一个单一用户回话的多个页面请求过程来跟踪用户。

本书使用Apache Web服务器来处理Perl、Ruby、PHP以及Python脚本，使用Tomcat服务器处理使用JSP标记编写的Java脚本。Apache和Tomcat可以从Apache团体获得：

<http://httpd.apache.org>

<http://tomcat.apache.org>

因为Apache的安装相当流行，我继续假设你的系统中已经安装过以及你只要配置它就可以了。本章第2节讨论了如何针对Perl、Ruby、PHP和Python配置Apache，以及如何使用每种语言编写一个简短的脚本。第3节讨论了使用Tomcat编写JSP脚本。Tomcat没有像Apache那么被广泛地配置，所以在附录C中给出了一些额外的安装信息。你可以选择使用Apache和Tomcat之外的一些服务器，但是你需要修改这里给出的指令。

本章中的例子脚本可以在由运行它们的服务器命名的章节发布目录中找到。对于Perl、Ruby、PHP和Python的例子，查看apache目录。对于Java（JSP）的例子，可以在tomcat目录下找到。

我在这里假设你基本上精通HTML。对于Tomcat，了解一点XML的知识是很有用的，因为Tomcat的配置文件是使用XML文档编写的，而且JSP页面也包含XML语法的元素。如果不了解XML，你可以在下面的“XML and XHTML in a Nutshell”中看到一个快速概要。一般来说，本书中的Web脚本产生的输出，不仅作为HTML来说是有效的，而且作为XHTML（一种HTML和XML之间的过渡格式）也是有效的。（这也是要熟悉XML的另外一个原因。）例如，XHTML需要结束标签，所以段落在段落体后要有一个结束标签</p>。这种输出风格的使用对于使用像PHP编写的脚本中是非常明显的，在PHP中，HTML标签在脚本中按字面意义被包含。对于为你产生HTML的接口来说，与XHTML的一致性就成为是否模块本身产生XHTML的原因。例如，Perl的CGI.pm模块产生XHTML，但是Ruby的cgi模块却不产生。

Nutshell中的XML和XHTML

XML在某些方面和HTML很相似，而且因为很多人都了解HTML，所以可以很简单地使用和HTML的对比来刻画XML：

- 在HTML中，标签的字符大小写和属性名称并不重要。在XML中，名字是区分大小写的。
- 在HTML中，标签属性可以通过一个带引号的或不带引号的值来指定，或者有时根本就没有值。在XML中，每个标签属性必须有一个值，而且这个值必须带引号。
- XML中每一个开始标签必须有一个相应的结束标签。即使没有元素体，也必须这样，当然这种情况下可以使用一个简写标签形式。例如，在HTML中，你可以写`
`，但是在XML中，就必须有一个结束标签了。你可以这样写`
</br>`，但是元素没有体，所以可以写成简写形式`
`，它将开始标签和结束标签结合起来。然而，当编写将会转换为HTML的XML时，编写斜杠前有空白的`
`标签是安全的。这个空白使得老的浏览器不会认为标签名是`br/`，那样它就会将其作为一个未识别的元素丢弃掉。

XHTML是一个从HTML到XML的Web方法迁移的过渡形式。它没有XML那么严格，但是比HTML严格。例如，XHTML的标签和属性名必须是严格的小写字母，以及属性必须有一个双引号括起的值。

在HTML中，你或许会写一个像这样的单选按钮元素：

```
<INPUT TYPE=RADIO NAME="my button" VALUE=3 CHECKED>
```

在XHTML中，标签名字必须是小写的，属性值必须由引号包含，`checked`的属性必须给定一个值，并且必须有一个结束标签：

```
<input type="radio" name="my button" value="3" checked="checked"></input>
```

在元素没有体的情况下，可以使用单标签简写形式：

```
<input type="radio" name="my button" value="3" checked="checked" />
```

对于HTML、XHTML或XML的其他知识，请参考附录D。

17.1 Web页面产生的基本原则

Basic Principles of Web Page Generation

问题

你想从脚本中而不是手工编写产生Web页面。

解决方案

编写执行时产生页面的程序，比编写静态页面更能控制发往客户端的内容，尽管它也许需

讨论

HTML是一个标记语言——这也是“ML”的含义。HTML由被显示的无格式文本的混合和特殊的标记指示器或构造器组成，这些标记控制了无格式文本如何显示。这里有一个非常简单的HTML页面的例子，它在页面的头部里指定了标题，页面体是包含了一个单一段落的白色背景：

```
<html>
<head>
<title>Web Page Title</title>
</head>
<body bgcolor="white">
<p>Web page body.</p>
</body>
</html>
```

可以编写产生同样页面的脚本，但是这和编写静态页面又有不同。首先，你需要同时使用两种语言编写。（脚本使用你的编程语言编写，而脚本本身编写出HTML。）另外一个不同是，你也许必须产生更多的发往客户端的响应。当Web服务器向客户端发送静态页面时，它实际上发送了一组一个或多个头部行，首先提供了关于页面的附加信息。例如，一个HTML文档前包含一个Content-Type:头部，它使客户端了解到期望信息的种类；包含一个空白行表示将头部和页面体分隔开：

```
Content-Type: text/html

<html>
<head>
<title>Web Page Title</title>
</head>
<body bgcolor="white">
<p>Web page body.</p>
</body>
</html>
```

对于静态HTML页面，Web服务器自动创建头部消息。当你编写Web脚本时，你或许需要提供头部消息。一些API（例如PHP）可以自动发送一个content-type头部，但是也允许你覆盖默认类型。例如，如果你的脚本向客户端发送一张JPEG图片，而不是发送一个HTML页面，你应该使用脚本将内容类型从text/html改为image/jpeg。

对于输入和输出来说，编写Web脚本都和编写命令行脚本不同。在输入端，给Web脚本的信

息由Web服务器提供，而不是由命令行参数或你键入的输入提供。这意味着你的脚本不能使用输入语句获得数据。取而代之，Web服务器将信息放置在脚本的执行环境，脚本可以从它的环境中提取信息并使用信息。

在输出端，命令行脚本典型地产生无格式文本输出，然而Web脚本产生HTML、图像或者你需要发往客户端的其他内容类型。在Web环境中产生的输出通常必须高度的结构化来确保接收输出的客户端程序能够理解。

任何编程语言都可以允许你通过打印语句产生输出，但是一些编程语言也提供了产生Web页面的专门工具助手。这种支持典型地通过以下模块提供：

- 对于Perl脚本，一个流行的模块是CGI.pm。它提供产生HTML标记、格式处理等特征。
- 在Ruby中，cgi模块提供类似于CGI.pm的功能。
- PHP脚本的编写使用HTML和内嵌PHP代码的混合。这就是说，你将HTML逐字地写入脚本，然后当你需要通过执行代码来产生输出的时候进入程序模式。PHP在发往客户端的结果页面中通过它的输出来替代代码。
- Python包括cgi和urllib模块，能够帮助执行Web编程任务。
- 对于Java，我们将根据JSP规范编写脚本，JSP允许在Web页面中嵌入脚本指令和代码。这有点类似于PHP的工作机制。

除本书中用到的以外，还可以获得其他产生页面的包——其中的一些在你使用语言的方式上可以有显著的效果。例如，Mason、embPerl、ePerl以及Axkit可以使你把Perl作为一个嵌入式语言对待，有点类似于PHP的工作方式。同样地，mod_snake Apache模块使得Python代码可以嵌入到HTML模板中。

另外的动态页面产生方法将页面设计和脚本编写分开。将一个Web页面作为一个模板来设计，包含用简单的标记来指示替换哪里可以改变每次请求的值，以及脚本获得将要显示的数据。通过这种方法，脚本将获取数据看作页面产生的一个阶段。脚本调用模板引擎作为页面产生的另外一个阶段，将数据放入模板中并产生最终的输出页面。这将页面设计从脚本编写中分离出来。第18章第10节通过使用Ruby页面模板和PHP Smarty模板包，更进一步探究了用这种方法来产生页面。

在你可以在Web环境中运行脚本之前，你的Web服务器必须合适地配置。Apache和Tomcat的配置信息在本章的第2节和第3节中给出，但从理论上讲，一个Web服务器可以通过两种方法中的一种来典型地运行脚本。第一，Web服务器可以使用外部程序来执行脚本。例如，它

可以调用Python解释器的一个实例来运行Python脚本。第二，如果服务器激活了适当的语言处理能力，它就可以自己执行脚本。使用外部程序执行脚本不需要Web服务器的特殊能力，但是因为它包括启动单独的程序，以及有将请求信息写入脚本并从中读取结果的额外开销，因此比较慢。如果你在服务器中嵌入语言处理器，就能够直接执行脚本，可以获得更好的性能。

像很多Web服务器一样，Apache可以运行外部脚本。它也支持扩展模块的概念，即成为Apache进程的一部分（或者编译进去，或者在运行时载入）。这个特征的一个通常应用是将语言处理器嵌入到服务器中以加速脚本执行。可以使用两种方法中的任一个来执行Perl、Ruby、PHP和Python脚本。像命令行脚本一样，外部执行的Web脚本编写可执行文件，此文件以指定合适语言解释器的路径名的#!行开始。Apache使用路径名来决定哪一个解释器来运行脚本。另外，你可以使用例如Perl的mod_perl，Ruby的mod_ruby，PHP的mod_php以及Python的mod_python和mod_snake模块来扩展Apache。这使Apache有能力直接执行通过这些语言编写的脚本。

对于Java JSP脚本，脚本被编译成Java Servlets并在Servlet容器中运行。脚本由管理脚本的服务器进程来运行，而不是为每个脚本调用外部进程，从这个意义上说，JSP的运行机制有点类似于内嵌解释器的方法。客户端第一次请求某个JSP页面的时候，容器首先将页面编译成一个可执行的Java字节码形式的Servlet，然后载入并运行。容器缓存字节码Servlet，所以紧接着对该脚本的请求可以不经过编译阶段直接运行。如果你修改了脚本，那么当下一次请求到达的时候容器会注意到并重新编译脚本成一个新的Servlet，然后重载它。相对于直接编写Servlet，JSP的方法有一个重要的优点，就是你不用自己编译代码以及处理Servlet的载入和载出。Tomcat可以作为Servlet的容器，也可以作为和容器通信的Web服务器。

如果你在一个主机上运行多个服务器，它们要在不同的端口上监听请求。在一个典型配置中，Apache在默认HTTP端口80上监听请求，Tomcat在另外一个端口8080上监听请求。这里的例子使用本地主机的服务器主机名来表示通过Apache或Tomcat处理的脚本的URL。对Tomcat脚本，例子使用8080端口。在本书中你将会看到的URL的典型形式如下：

```
http://localhost/cgi-bin/my_perl_script.pl  
http://localhost/cgi-bin/my_ruby_script.rb  
http://localhost/cgi-bin/my_python_script.py  
http://localhost/mcb/my_php_script.php  
http://localhost:8080/mcb/my_jsp_script.jsp
```

对于你自己的服务器包含的页面，如果需要的话，要改变主机名和端口号。

17.2 使用Apache运行Web脚本

Using Apache to Run Web Scripts

问题

你想要在Web环境中运行Perl、Ruby、PHP或者Python程序。

解决方案

使用Apache服务器执行它们。

讨论

本节描述了如何配置Apache来运行Perl、Ruby、PHP和Python脚本。本节也举例说明了如何使用每种语言编写基于Web的脚本。

在Apache的根目录下有几个典型的目录。这里，我们假设根目录是/usr/local/apache，或许和你的系统上的有所不同。例如，在Windows系统下，你可能在C:\Program Files目录下发现Apache。Apache的根目录下包括bin（包含httpd——即Apache自身——以及其他Apache相关的可执行程序）、conf（放置配置文件，特别是httpd.conf，Apache使用的主要文件）、htdocs（文档树的根）以及logs（日志文件目录）。目录分布也许和你的系统上有些不同。例如，你或许在/etc/httpd目录中发现配置文件，在/var/log/httpd中发现日志文件。那么就要相应地修改下面的说明。

为了脚本执行而配置Apache，在conf目录下编辑httpd.conf文件。典型地，可执行脚本通过位置或文件后缀名来标识。位置可以是语言无关或语言指定的。

Apache配置通常在Apache的根目录下有一个cgi-bin的目录，你可以将应该作为外部程序运行的脚本安装在那里。使用ScriptAlias指令来进行配置：

```
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

第二个参数是你的文件系统中脚本目录的实际位置，第一个参数是在URL中和此目录相对应的路径名。这样，指令的含义就是把位于/usr/local/apache/cgi-bin中的脚本和cig-bin后跟随主机名的URL之间联系起来。例如，你在本地主机上将Ruby脚本myscript.rb安装在/usr/local/apache/cgi-bin目录下，你应该用这样的URL来请求此脚本：

```
http://localhost/cgi-bin/myscript.rb
```

当用这种方法配置时，cgi-bin目录可以包含任何语言编写的脚本。基于这个原因，目录是语言无关的，所以Apache需要能够指出使用哪种语言处理器来执行安装在这里的脚本。为了提供这个信息，脚本的第一行应该是#!跟随执行此脚本程序的路径名。例如，一个脚本具有以下的第一行，应该由Perl来运行：

```
#!/usr/bin/perl
```

在Unix下，你必须也要使脚本是可执行的（使用chmod +x），不然它就不会正确运行。刚才展示的#!行对于一个在/usr/bin/perl安装了Perl的系统来说是合适的。如果你的Perl解释器安装在其他位置，那么相应的修改此行。如果你位于在C:\Perl\bin\perl.exe安装Perl的Windows系统下，#!行应该是：

```
#!C:\Perl\bin\perl
```

对于Windows系统，另外一个更简单的选项是在以.pl后缀结尾的脚本名和Perl解释器之间建立文件名扩展联系。这样，脚本就可以具有如下的第一行：

```
#!perl
```

只是针对脚本的目录一般放置在你的Apache文档树之外。像使用为脚本指定的目录一样，你可以通过文件扩展名来标识脚本，这样有特定后缀名的文件就和指定的语言处理器联系在一起了。这种情况下，你可以将脚本放置在文档树的任何地方。这是使用PHP时最通常的办法。例如，你有一个支持内嵌使用mod_php模块的PHP配置的Apache，你可以告诉它以.php结尾的脚本应该解释成PHP脚本。为了做到这一点，在httpd.conf文件中增加一行：

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

你或许需要为php增加（或者去注释）一个LoadModule指令。

当使用PHP时，你可以将一个PHP脚本安装在htdocs（Apache文档树目录）下。这样调用脚本的URL就是：

```
http://localhost/myscript.php
```

如果PHP作为一个单独外部程序运行，你要告诉Apache在哪里能找到它。例如，如果你在Windows下并且PHP安装在C:\Php\php.exe，就要将下面几行写到httpd.conf文件中（注意在路径名中使用斜杠而不是反斜杠）：

```
ScriptAlias /php/ "C:/Php/"
AddType application/x-httpd-php .php
Action application/x-httpd-php /php/php.exe
```

为了用例子展示URL，我将假设Perl、Ruby和Python脚本在你的cgi-bin目录下，并且PHP脚

本在你的文档树的mcb目录下，通过.php标识。这意味着在这些语言中，脚本的URL会像下面这样：

```
http://localhost/cgi-bin/myscript.pl  
http://localhost/cgi-bin/myscript.rb  
http://localhost/cgi-bin/myscript.py  
http://localhost/mcb/myscript.php
```

如果你计划使用类似的设置，要确保你有一个cgi-bin目录并且Apache知道它，以及在你的Apache文档根下有一个mcb目录。然后，为了部署Perl、Ruby或Python Web脚本，应将它们安装在cgi-bin目录下。为了部署PHP脚本，应将它们安装在mcb目录下。

一些脚本使用特定的模块或库文件。如果你将这些文件安装在一个你的语言处理器默认搜索的库目录下，它们能够自动地被找到。否则，你必须指明文件在哪里。一个简单的方法是在你的httpd.conf文件中使用SetEnv指令，通过它设置环境变量，当Apache调用脚本时用到。（使用SetEnv指令需要使用Apache的mod_env模块。）例如，如果你将库文件安装在/usr/local/lib/mcb下，下面的指令使Perl、Ruby和Python脚本能够找到它们：

```
SetEnv PERL5LIB /usr/local/lib/mcb  
SetEnv RUBYLIB /usr/local/lib/mcb  
SetEnv PYTHONPATH /usr/local/lib/mcb
```

对于PHP，在你的php.ini配置文件中向include_path值中增加/usr/local/lib/mcb。

至于库相关环境变量和php.ini文件的信息，请参考第2章第3节。

在将Apache配置成支持脚本执行后，重启Apache。然后你可以编写脚本产生Web页面了。本节的余下部分讲述了对Perl、Ruby、PHP和Python如何操作。每种语言连接MySQL服务器的例子，都执行一个简单的查询，然后在Web页面上显示出来。这里的脚本指明了脚本是否需要包含额外的模块或库。（稍后，我将会一般地假设适当的模块已经包含进来了并会展示一些脚本片段。）

在继续进行之前，我需要提及一些调试技巧：

- 如果你请求一个Web脚本，并且在响应中得到一个错误页面，查看一下Apache的错误日志，当你试图指出为什么脚本不能正常工作时，这里有很多有用的诊断信息。在日志目录下，这个日志的通常名字叫做error_log。如果你没有找到相应文件，那么检查httpd.conf中的ErrorLog指令，看看Apache在哪里记录错误。
- 有时直接检查Web脚本产生的输出是非常有用的。你可以从命令行调用脚本来实现这一点。你将会看到脚本产生的HTML，和任何它可能打印出的错误消息。一些Web模块

期望知道参数字符串，当你在命令行调用脚本时甚至可能提示你输入。碰到这种情况，你或许能够在命令行中以参数形式提供这些参数值来避免命令提示。例如，Ruby的cgi模块期望看到这些参数，如果没有它们，将会提示你：

```
% myscript.rb  
(离线模式：在标准的输入上键入名字=值对)
```

在命令提示上，键入参数值，然后键入Ctrl-D (EOF)。为了避免命令提示，在命令行上提供参数值：

```
% myscript.rb "param1=val1;param2=val2;param3=val3"
```

为了清晰地指定“无参数”，提供一个空的参数：

```
% myscript.rb ""
```

Web安全注解

在Unix下，脚本在它们执行时是和特定的用户ID和组ID相关的。你通过命令行执行的脚本通过你自己的用户ID以及组ID运行，拥有和你账户相关的文件系统权限。通过Web服务器执行的脚本不由你的用户和组ID运行，也不拥有你的权限。它们是在运行Web服务器的账户的用户和组ID下运行，拥有该账户的权限。(为了决定是哪一个账户，在httpd.conf配置文件中查找User和Group指令。)这意味着如果你期望由Web脚本来读写文件，那些文件必须是运行Web服务器的账户可访问的。例如，如果你的服务器在nobody账户下运行，你想要脚本能够存储上传的图片到文档树下一个叫uploads的目录里，你必须确保nobody用户可以读写这个目录。

另外一个建议是，如果其他用户能够编写由你的Web服务器执行的脚本，这些脚本也将作为nobody运行，它们可以像你自己的脚本一样读写相同的文件。也就是说，你的脚本用到的文件不能看成你的脚本私有的。这个问题的一个解决方案是使用Apache的suEXEC机制。(如果为一个Web主机使用ISP，你或许会发现suEXEC已经能使用了。)

Perl

下面的列表显示了我们第一个基于Web的Perl脚本，show_tables.pl。它产生合适的Content-Type：头部，分割头部和页面内容的空白行，页面的初始部分。然后它从cookbook数据库中检索和显示一个表格。一个HTML标签紧跟这个表格列表来结束该页面。

```
#!/usr/bin/perl  
# show_tables.pl -通过直接生成HTML来显示手册数据库中的表名称  
# by generating HTML directly
```

```

use strict;
use warnings;
use Cookbook;

# 打印头、空白行、以及页面初始部分

print <<EOF;
Content-Type: text/html

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>
EOF

# 连接数据库，显示表列表，然后断开连接

my $dbh = Cookbook::connect ();
my $stmt = "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
            WHERE TABLE_SCHEMA = 'cookbook' ORDER BY TABLE_NAME";
my $sth = $dbh->prepare ($stmt);
$sth->execute ();
while (my @row = $sth->fetchrow_array ())
{
    print "$row[0]<br />\n";
}
$dbh->disconnect ();

# 打印页面挂接

print <<EOF;
</body>
</html>
EOF

```

为了测试该脚本，将它安装在你的cgi-bin目录下，在你的浏览器中用下面的URL请求：

```
http://localhost/cgi-bin/show_tables.pl
```

show_tables.pl产生Content-Type:清晰的头部，它通过在打印语句中包含标签来产生HTML元素。另外一种Web页面的生成方法是使用CGI.pm模块，它可以很方便地编写Web脚本而无需写HTML标签。CGI.pm提供了一个面向对象的接口，函数叫做接口，这样你可以用两种风格中的任一个来使用它编写Web页面。这里有一个脚本，show_tables_oo.pl，它使用CGI.pm面向对象接口来产生和show_tables.pl相同的结果：

```

#!/usr/bin/perl
# show_tables_oo.pl -通过使用CGI.pm中的面向对象接口显示手册数据库中的表名称
# using the CGI.pm object-oriented interface

```

```

use strict;
use warnings;
use CGI;
use Cookbook;

# 创建访问CGI.pm方法的对象

my $cgi = new CGI;

# 打印头、空白行、以及页面初始部分

print $cgi->header ();
print $cgi->start_html (-title => "Tables in cookbook Database",
                        -bgcolor => "white");

print $cgi->p ("Tables in cookbook database:");

# 连接数据库，显示表列表，然后断开连接

my $dbh = Cookbook::connect ();
my $stmt = "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
            WHERE TABLE_SCHEMA = 'cookbook' ORDER BY TABLE_NAME";
my $sth = $dbh->prepare ($stmt);
$sth->execute ();
while (my @row = $sth->fetchrow_array ())
{
    print $row[0], $cgi->br ();
}
$dbh->disconnect ();

# 打印页面挂接

print $cgi->end_html ();

```

脚本通过使用一个CGI语句来包含CGI.pm模块，然后产生一个CGI对象，\$cgi，通过这个对象，脚本调用不同的生成HTML的调用。header()产生Content-Type:头部；start_html()产生开始页面标签<body>。在产生页面的第一部分以后，show_table_oo.pl从服务器中检索和显示信息。每一个表名紧跟一个通过调用br()方法产生的
标签。end_html()产生结束标签</body>和</html>。当你将脚本安装在cgi-bin目录下，并从浏览器中调用它时，你可以看到它和show_tables.pl产生相同类型的页面。

CGI.pm 调用经常需要多个参数，这些参数多数都是可选参数。为了可以让你指定你所需要的那些参数，CGI.pm可以解析在参数列表中 -name=>value的标记。例如，在start_html()的调用中，title参数设置了标题bgcolor参数设置了背景颜色。这种-name=>value标记还允许参数以任意的顺序被指定，于是这两句语句是等价的：

```
print $cgi->start_html (-title => "My Page Title", -bgcolor => "white");
print $cgi->start_html (-bgcolor => "white", -title => "My Page Title");
```

为了使用CGI.pm这个函数调用而不使用基于对象的接口，你的脚本必须写的有点稍微不同。为了使用引用了CGI.pm的语句，要将这些方法的名字引入到你脚本代码的命名空间中去。这样你就可以像使用函数一样直接调用他们，而不需要创建一个CGI的对象。例如，为了引入最经常使用的方法，程序脚本应该包含这个语句：

```
use CGI qw(:standard);
```

下面的这个脚本，show_tables_fc.pl，是和刚才show_tables_oo.pl脚本作用一样的函数调用。它使用了同样的CGI.pm调用，但是把它当作单独的函数直接来调用的，而不是通过一个\$cgi对象。

```
#!/usr/bin/perl
# show_tables_fc.pl - 显示cookbook数据库中表的名字
# 使用 CGI.pm 函数调用接口

use strict;
use warnings;
use CGI qw(:standard); # 引入标准方法名到脚本的命名空间中
use Cookbook;

# 打印头(header)、空行和页面的初始部分

print header ();
print start_html (-title => "Tables in cookbook Database",
                  -bgcolor => "white");

print p ("Tables in cookbook database:");

# 连接数据库，显示表的列表，然后断开连接

my $dbh = Cookbook::connect ();
my $stmt = "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
            WHERE TABLE_SCHEMA = 'cookbook' ORDER BY TABLE_NAME";
my $sth = $dbh->prepare ($stmt);
$sth->execute ();
while (my @row = $sth->fetchrow_array ())
{
    print $row[0], br ();
}
$dbh->disconnect ();

# 打印页尾(trailer)

print end_html ();
```

将show_tables_fc.pl脚本安装到你的cgi-bin目录并且通过你的浏览器向它发送请求，看看它是否返回了和show_tables_oo.pl同样的输出。

本书在此之后将对于基于perl的Web脚本使用CGI.pm函数调用接口。你可以在命令行方式下通过下列的命令阅读已经安装的文档来获取CGI.pm更多的信息：

```
% perldoc CGI  
% perldoc CGI::Carp
```

附录D列出了在线的和已经印刷出来的关于这个模块信息的其他资源。

Ruby

Ruby的cgi模块提供了用来生成HTML的方法接口。要使用它，就要创建一个CGI的对象并且调用它的方法来生成HTML页面元素。这些方法以他们对应的HTML元素名为后缀来命名。他们的调用语法遵循下列规则：

- 如果一个（HTML）元素含有属性，那么将它们当作参数传递给该方法。
- 如果一个（HTML）元素有body内容，和方法调用一起将内容在代码块中指定。

例如，下面的方法调用产生一个<P>元素，这个元素包含一个align属性并且它的内容是“*This is a sentence*”：

```
cgi.p("align" => "left") { "This is a sentence" }
```

输出如下：

```
<P align="left">This is a sentence.</P>
```

要将生成的HTML内容展现出来，将它在代码段中当作参数传递给cgi.out方法即可。下列的Ruby脚本，show_tables.rb，重新得到cookbook数据库中表格的列表并将他们用HTML文档的形式展现出来：

```
#!/usr/bin/ruby -w  
# show_tables.rb - 显示cookbook数据库中的表名  
  
require "cgi"  
require "Cookbook"  
  
# 连接数据库，显示数据库表的列表，然后断开连接  
  
dbh = Cookbook.connect  
stmt = "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES  
       WHERE TABLE_SCHEMA = 'cookbook' ORDER BY TABLE_NAME"  
rows = dbh.select_all(stmt)  
dbh.disconnect  
  
cgi = CGI.new("html4")  
  
cgi.out {  
  cgi.html {  
    cgi.head {  
      cgi.title { "Tables in cookbook Database" }  
    }  
  }  
}
```

```
    } +
    cgi.body("bgcolor" => "white") {
      cgi.p { "Tables in cookbook Database:" } +
      rows.collect { |row| row[0] + cgi.br }.join
    }
  }
}

collect方法遍历了包含所有数据库表明的行数组并且创建了一个新的数组，这个新数组包含每一个数据库表名并且附加了一个<br>在表名之上。join方法将结果数组中的字符串连接起来。
```

这个脚本并没有引入外部的代码来生成Content-Type: header因为cgi.out生成了一个。

将这个脚本安装到你的cgi-bin目录下面，并且通过你的浏览器发送一个类似下面的请求：

```
http://localhost/cgi-bin/show_tables.rb
```

如果你从命令行来调用Ruby的Web脚本并且查看产生的HTML，你会发现所有的HTML都显示在同一行中，很难阅读。为了使输出易于阅读，可以通过CGI.pretty工具方法来处理它，CGI.pretty工具方法会给输出添加换行符和缩进。假设你的页面输出调用类似如下所示语句：

```
cgi.out {
  page content here
}
```

要改为使用CGI.pretty，那么改为这样来写：

```
cgi.out {
  CGI.pretty(page content here)
}
```

PHP

PHP并没有提供很多便捷的标签，这对源于Web的PHP来说很让人惊讶。从另外一方面来说，因为PHP是一个内嵌语言，你可以将你的HTML直接写到你的脚本里面去而不需要print语句。这是一个show_tables.php脚本在HTML模式和PHP模式之间来回切换的例子：

```
<?php
# show_tables.php - 显示cookbook数据库的表名

require_once "Cookbook.php";

?>

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
```

```

<body bgcolor="white">
<p>Tables in cookbook database:</p>
<?php

# 连接数据库，显示表名，断开数据库连接
$conn = & Cookbook::connect ();
$stmt = "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
        WHERE TABLE_SCHEMA = 'cookbook' ORDER BY TABLE_NAME";
$result = & $conn->query ($stmt);
while (list ($tbl_name) = $result->fetchRow ())
    print ($tbl_name . "<br />\n");
$result->free ();
$conn->disconnect ();

?>

</body>
</html>

```

如果要尝试运行这个脚本，将它放到你Web服务器文档树的mcb目录下并且通过你的浏览器发送如下请求：

```
http://localhost/mcb/show_tables.php
```

这个PHP脚本并没有包含用来产生Content-Type: header的代码，因为PHP自动生成它。（如果你想重写这个动作行为来创建你自己的headers，参考PHP手册header()方法部分）

除了换行标签，show_tables.php通过将HTML内容写在<?php和?>标签对之外来包含HTML内容，这样PHP就能不需解释器解释而直接将HTML内容更容易的输出。下面是一个通过print语句来生成所有HTML内容的不同版本：

```

<?php
# show_tables_print.php - 显示cookbook数据库中的表名
# 使用 print() 来生成所有的HTML

require_once "Cookbook.php";

print ("<html>\n");
print ("<head>\n");
print ("<title>Tables in cookbook Database</title>\n");
print ("</head>\n");
print ("<body bgcolor=\"white\">\n");
print ("<p>Tables in cookbook database:</p>\n");

# 连接数据库，显示表名，断开数据库连接
$conn = & Cookbook::connect ();
$stmt = "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
        WHERE TABLE_SCHEMA = 'cookbook' ORDER BY TABLE_NAME";
$result = & $conn->query ($stmt);
while (list ($tbl_name) = $result->fetchRow ())
    print ($tbl_name . "<br />\n");

```

```
$result->free ();
$conn->disconnect ();

print ("</body>\n");
print ("</html>\n");
?>
```

有时候使用某一种方式更合适，有时候使用另一种方式，有时候在同一个脚本中同时使用它们两者。如果一个HTML块不参考到任何的变量或者表达式的值，将它写成HTML模式将更加清晰。否则将它写成用print或者echo语句的方式将更清晰，避免在HTML模式和PHP模式之间过于频繁的切换。

Python

一个标准的Python安装包括cgi和urllib模块，这两个模块对Web开发非常有用。然而，我们现在还并不真的需要他们，因为我们第一个Python Web脚本做的和Web相关的事情仅仅是生成一些简单的HTML。这是一个显示MySQL表格的Python版本脚本程序：

```
#!/usr/bin/python
# show_tables.py - 显示cookbook数据库的表名

import MySQLdb
import Cookbook

# 打印header、空行和页面初始部分

print """Content-Type: text/html

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>
"""

# 连接数据库，显示表名，断开数据库连接

conn = Cookbook.connect ()
cursor = conn.cursor ()
stmt = """
    SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_SCHEMA = 'cookbook' ORDER BY TABLE_NAME
"""
cursor.execute (stmt)
for (tbl_name, ) in cursor.fetchall ():
    print tbl_name + "<br />"
cursor.close ()
conn.close ()

# 打印页尾(trailer)
```

```
print """
</body>
</html>
"""
```

将这个脚本放到Apache的cgi-bin目录下面并且用你的浏览器发送如下请求：

```
http://localhost/cgi-bin/show_tables.py
```

17.3 使用Tomcat运行Web脚本

Using Tomcat to Run Web Scripts

问题

你想在Web环境中运行基于Java的程序。

解决方案

使用JSP符号来编写程序，在一个例如Tomcat的servlet容器中执行程序。

讨论

像上一节中描述的，Apache可以用来运行Perl、Ruby、PHP和Python脚本。对于Java，需要一种不同的方法，因为Apache不能执行JSP页面。作为替代，我们使用Tomcat，一个为处理Web环境中的Java程序而设计的服务器。Apache和Tomcat是非常不同的服务器，但是有一个亲缘关系——像Apache一样，Tomcat也是由Apache软件基金会开发的。

本节提供了使用Tomcat进行JSP编程的概述，但是要做一些假设：

- 你已经精通JavaServer Pages的相关概念，比如什么是servlet容器，什么是应用程序上下文，什么是基本的JSP脚本元素。
- 已经安装了Tomcat服务器，这样你可以执行JSP页面，并且你知道如何启动和停止Tomcat。
- 你熟悉Tomcat的Webapps目录，知道一个Tomcat应用程序的结构。特别地，你了解WEB-INF目录和Web.xml文件的用途。
- 你了解标签库是什么并且知道如何使用它。

我承认假设的东西太多了。如果你对JSP不熟悉或者需要安装Tomcat的说明，附录C提供了必需的背景知识。

一旦你在适当位置安装了Tomcat，你必须安装下面的组件，这样你才能运行本书中的例子：

- 位于发行版Tomcat目录中的mcb例子程序。
- 一个MySQL的JDBC驱动。你可能为了使用前面的章节中的脚本已经安装过了，但是Tomcat也需要一个拷贝。本书使用的是MySQL Connector/J。
- JSP标准标签库（JSTL），它包含在JSP页面中执行数据库操作的标签、条件测试标签、迭代操作标签。

本节讨论了如何安装这些组件，提供了一些JSTL标签的简短概述，并且描述了如何编写JSP脚本，编写的JSP脚本和在上一节中使用Perl、Ruby、PHP和Python实现的MySQL表显示脚本等价。

安装mcb应用程序

Tomcat的Web应用程序典型地作为WAR（Web档案文件）文件打包，并安装在Tomcat的Webapps目录下，该目录大概类似于Apache的htdocs文档树目录。发行版包含了一个名为mcb的例子程序，你可以使用这个例子程序来测试这里描述的JSP例子。查看发行版的tomcat目录，在那里你会找到一个名为mcb.war的文件。将它拷贝到Tomcat的Webapps目录下。

这里是 Unix 下例子安装程序，假设发行版和 Tomcat 分别位于 /u/paul/recipes 和 /usr/local/Jakarta-tomcat 下。如果有必要在你的系统里调整路径名。安装mcb.war的命令如下：

```
% cp /u/paul/recipes/tomcat/mcb.war /usr/local/jakarta-tomcat/webapps
```

对于Windows，如果相应目录是C:\recipes和C:\jakarta-tomcat，命令为：

```
C:\> copy C:\recipes\tomcat\mcb.war C:\jakarta-tomcat\webapps
```

在将mcb.war文件拷贝到Webapps目录下以后，重启Tomcat。按照发布说明，Tomcat配置成当启动时默认在Webapps目录下寻找WAR文件，并自动解压任何还未解压的包。这意味着在将mcb.war拷贝到Webapps目录后重启Tomcat应该足够可以解压mcb应用程序。当Tomcat完成启动序列，查看Webapps，你应该看到一个新的mcb目录，在这个目录下有mcb.war里包含的所有文件。（如果Tomcat没有自动解压mcb.war，参考“手动解压WAR文件”。）如果你愿意，查看一下mcb目录，里面应该包含一些客户端能够通过浏览器请求的文件。还应该有一个WEB-INF子目录，这里包含私有信息——也就是说，由mcb目录下的脚本使用的信息，客户端不能直接访问。

接下来，验证Tomcat能够从mcb应用程序上下文中通过从你的浏览器请求页面来运行这些页面。下面的URL依次请求静态HTML、一个servlet和一个简单JSP页面：

```
http://localhost:8080/mcb/test.html  
http://localhost:8080/mcb/servlet/SimpleServlet  
http://localhost:8080/mcb/simple.jsp
```

根据你的安装适当地调整主机名和端口号。

手动解压WAR文件

WAR文件实际上是ZIP格式的存档文件，可以使用jar，WinZip或者任意其他的ZIP文件解压工具来解包。然而，当手动解压WAR文件时，你必须首先创建一个顶级目录。下面的序列步显示了手动解压的一种方法，使用jar工具来解压一个名为mcb.war的WAR文件，mcb.war假设位于Tomcat的Webapps目录下。对于Unix，更改一下Webapps目录的位置，然后考虑下面的命令：

```
% mkdir mcb  
% cd mcb  
% jar xf ../mcb.war
```

对于Windows，命令稍有不同：

```
C:\> mkdir mcb  
C:\> cd mcb  
C:\> jar xf ..\mcb.war
```

在Webapps目录下解压WAR文件会创建一个新的应用程序上下文，这样你就需要在通知新的应用程序之前重启Tomcat。

安装JDBC驱动

在mcb应用程序中的JSP页面需要一个JDBC驱动来连接cookbook数据库。下面的指示描述了如何安装MySQL Connector/J驱动；其他驱动的安装过程类似。

为了使用Tomcat应用程序来安装MySQL Connector/J，需要将它的一个拷贝放置在Tomcat的目录树中。假设驱动打包成一个JAR文件（是MySQL Connector/J的一种），在Tomcat的根目录下，你可以安装的合适地方有三处，这依赖于你想要的驱动的可见度：

- 为了使驱动只能由mcb应用程序使用，可将其放置在Tomcat的Webapps目录下的mcb/WEB-INF/lib目录里。
- 为了使所有的Tomcat应用程序都能使用驱动，但是Tomcat本身不可以，需要将其放置在Tomcat根目录下的shared/lib目录下。
- 为了使所有的Tomcat应用程序包括Tomcat本身都可以使用驱动，应将其放置在Tomcat根目录下的common/lib目录下。

我建议在common/lib目录下安装驱动的一个拷贝。这使其是全局可见的（即Tomcat和应用程序都可以使用），并且你只需要安装一次。如果你通过将拷贝放置在mcb/WEB-INF/lib下，使驱动只能由mcb应用程序使用，但是紧接着就要开发其他的使用MySQL的应用程序，那么你就需要将其拷贝到每个应用程序里或者将其移到一个全局位置。

如果你想在某些地方选择使用基于JDBC的会话管理或领域验证，使驱动具有更全局的可见性是很有用的。这些动作由Tomcat在应用程序级别之上处理，这样Tomcat就需要能够访问驱动来执行这些动作。

这里有一个Unix下安装程序的例子，假设MySQL Connector/J驱动和Tomcat分别位于/tmp/mysql-connector-java-bin.jar和/usr/local/Jakarta-tomcat下。如果需要根据你的系统调整路径名。安装驱动的命令如下：

```
% cp /tmp/mysql-connector-java-bin.jar /usr/local/jakarta-tomcat/common/lib
```

对于Windows，如果组件安装在C:\mysql-connector-java-bin.jar和C:\jakarta-tomcat下，使用如下命令：

```
C:\> copy C:\mysql-connector-java-bin.jar C:\jakarta-tomcat\common\lib
```

安装完驱动之后，重启Tomcat，然后请求如下的mcb应用程序页面来验证Tomcat能够正确地找到驱动：

```
http://localhost:8080/mcb/jdbc_test.jsp
```

你可能需要首先编辑jdbc_test.jsp页面来更改连接参数。

安装JSTL发行版

mcb例子应用程序中的大多数脚本都使用JSTL，所以需要安装JSTL，不然脚本不能正常工作。为了在应用程序上下文中安装标签库，将标签库文件拷贝到应用程序WEB-INF目录下合适的位置。一般来说，这意味着至少要安装一个JAR文件和一个标签库描述符（TLD）文件，并且将一些标签库信息添加到应用程序的Web.xml文件中。JSTL实际上由一些标签集合组成，所以包括几个JAR文件和TLD文件。下面描述了安装JSTL供mcb应用程序使用的步骤：

1. 确定mcb.war已经解压，并在Tomcat的Webapps目录下创建了一个mcb应用程序目录层次。（参考本节的“安装mcb应用程序”。）这是必须的，因为JSTL文件必须安装在mcb/WEB-INF目录下，这个目录直到mcb.war解压后才出现。
2. 从Jakarta工程Web站点上下载JSTL发布。通过<http://jakarta.apache.org/taglibs/>访问Jakarta

标签库工程页面，根据标准标签库链接得到JSTL信息页面；后者有一个下载区，从那里你可以得到JSTL的二进制发布。确保你下载的是1.1.2版本或者更高。

3. 解压JSTL发布到一个方便的位置，最好在Tomcat目录层次之外。解压命令类似于解压Tomcat的命令（参考附录C中的“安装Tomcat发行包”）。例如，为了解压一个ZIP格式的发布，使用下面的命令，需要时调整一下文件名：

```
* jar xf jakarta-taglibs-standard.zip
```

4. 解压发布会创建一个包含多个文件的目录。将JAR文件（jstl.jar、standard.jar等）拷贝到mcb/WEB-INF/lib目录下。这些文件包含实现JSTL标签动作的类库。将标签库描述符文件（c.tld、sql.tld等）拷贝到mcb/WEB-INF目录下。这些文件为由JAR文件中的类实现的动作定义了接口。
5. mcb/WEB-INF目录包含了一个名为Web.xml的文件，这是Web应用程序的部署描述符文件（即“配置文件”）。更改Web.xml，为每个JSTL TLD文件增加<taglib>条目。条目的形式如下：

```
<taglib>
  <taglib-uri>http://java.sun.com/jsp/jstl/core</taglib-uri>
  <taglib-location>/WEB-INF/c.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>http://java.sun.com/jsp/jstl/sql</taglib-uri>
  <taglib-location>/WEB-INF/sql.tld</taglib-location>
</taglib>
```

每一个<taglib>包含一个<taglib-uri>元素，它描述了mcb的JSP页面引用相应TLD文件的符号名；一个<taglib-location>元素，指明了在mcb应用程序下TLD文件的位置。

（你会发现已经发布的Web.xml文件中早已包含这些条目。然而，你应该检查一下确保它们和你刚才已经安装的TLD文件名相匹配。）

6. mcb/WEB-INF目录也包含一个名为jstl-mcb-setup.inc的文件。这个文件不是JSTL的一部分，但是它包含一个JSTL标签<sql:setDataSource>，许多mcb的JSP页面使用这个标签设置数据源来连接cookbook数据库。这个文件内容大致如下：

```
<sql:setDataSource
  var="conn"
  driver="com.mysql.jdbc.Driver"
  url="jdbc:mysql://localhost/cookbook"
  user="cbuser"
  password="cbpass"
/>
```

如果需要改变你访问cookbook数据库时的连接参数，可编辑driver、url、user和password标签属性。不要改变var属性。

7. JSTL发布也包含WAR文件，这些文件里包含了文档和例子（standard-doc.war和standard-examples.war）。如果你想安装这些，将它们拷贝到Tomcat的Webapps目录下。（我建议你安装文档，这样你就可以从你自己的服务器上本地访问文档了。安装例子也非常有用，因为它们提供了有利的示范，展示了如何在JSP页面中使用JSTL标签。）
8. 重启Tomcat，它会注意到你对mcb应用程序刚做的这些变动，并会解压包含JSTL文档和例子的WAR文件。如果Tomcat不能自动地解压WAR文件，请参考前面的“手动解压WAR文件”。

在安装完JSTL并重启Tomcat后，应请求下面的mcb应用程序页面来验证Tomcat能够找到JSTL标签：

```
http://localhost:8080/mcb/jstl_test.jsp
```

使用JSTL编写JSP页面

本节讨论了一些JSTL标签的语法，这些标签在mcb的JSP页面中频繁用到。描述是非常简单的，这些标签中的许多都有另外的属性，允许以和这里不同的方式使用它们。要了解更多的信息，请参考附录D中的JSTL描述。

一个使用JSTL的JSP页面必须包含页面使用的每个标签集合自己的标签库指令。本书的例子使用核心标签和数据库标签，通过下面的标签库指令标识：

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
```

uri的值应该和在Web.xml文件<taglib>条目中列出来的符号名相匹配。（参看前面的“安装JSTL发布”一节）。prefix的值指示了用在标签名中的初始字符串，用来标识标签作为给定标签库的一部分。

JSTL标签是使用XML格式书写，对标签属性使用特殊的语法来包含表达式。在标签属性中，逐字解释文本，除非文本被\${...}包围，这种情况下，文本被解释成一个表达式并进行求值。下面的段落总结了一些常用的核心标签和数据库标签。

JSTL核心标签集。下面的标签是JSTL的部分核心标签集：

```
<c:out>
```

这个标签计算它的值属性并使用结果替代标签。这个标签的一个通常用法是为输出页面提供内容。下面的这个标签计算的结果是3：

```
<c:out value="${1+2}" />  
<c:set>
```

这个标签给变量赋值。例如，为了将一个字符串赋给名为title的变量并在后面的输出页面的<title>元素中使用变量，应该如下编写：

```
<c:set var="title" value="JSTL Example Page"/>  
  
<html>  
<head>  
<title><c:out value="${title}" /></title>  
</head>  
...
```

这个例子阐明了一个针对JSTL标签的通用原则：为了指定一个将存储某个值的变量，不要使用\${...}命名。为了在后面引用这个变量的值，在\${...}中使用它，这样它将会作为一个表达式进行求值。

```
<c:if>
```

这个标签对它的test属性进行条件测试。如果测试结果为真，评估标签体并将评估结果作为标签输出；如果测试结果为假，将忽略标签体：

```
<c:if test="${1 != 0}">  
1 is not equal to 0  
</c:if>
```

关系操作符是==、!=、<、>、<=和>=。替换操作符eq、ne、lt、gt、le和ge，使用起来更容易，能避免在表达式中使用特殊的HTML字符。算术操作符是+、-、*、/（或div）和%（或mod）。逻辑操作符是&&（或and）、||（或or）和!（或not）。如果值为空或为null，那么特殊的empty操作符为真：

```
<c:set var="x" value="" />  
<c:if test="${empty x}">  
x is empty  
</c:if>  
<c:set var="y" value="hello" />  
<c:if test="${!empty y}">  
y is not empty  
</c:if>
```

<c:if>标签不提供“else”子句。为了实现if/then/else测试，可以使用<c:choose>标签。

```
<c:choose>
```

这是另外一个条件标签，但是它允许多重条件测试。为每个你想要清晰测试的条件包含一个<c:when>标签，如果有一个默认子句，还要包含一个<c:otherwise>标签：

```
<c:choose>
    <c:when test="${count == 0}">
        Please choose an item
    </c:when>
    <c:when test="${count gt 1}">
        Please choose only one item
    </c:when>
    <c:otherwise>
        Thank you for choosing exactly one item
    </c:otherwise>
</c:choose>

<c:forEach>
```

这是迭代标签，能让你对一个集合的值进行循环。下面的例子使用<c:forEach>标签来对查询结果集合中的rows集合进行循环：

```
<c:forEach items="${rs.rows}" var="row">
    id = <c:out value="${row.id}" />,
    name = <c:out value="${row.name}" />
    <br />
</c:forEach>
```

每一次循环迭代都将当前行复制给变量行。假设查询结果包含有名为id和name的列，它们的值可以通过row.id和row.name来访问。

JSTL数据库标签集。 JSTL数据库标签集允许你执行SQL语句并访问结果：

```
<sql:setDataSource>
```

这个标签配置连接参数，当JSTL和数据库服务器连接时使用。例如，为了给使用MySQL Connector/J驱动访问cookbook数据库，标签如下所示：

```
<sql:setDataSource
    var="conn"
    driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/cookbook"
    user="cbuser"
    password="cbpass"
/>
```

driver、url、user和password属性指定了连接参数，var属性命名连接相关变量。根据约定，本书中的mcb JSP页面使用变量conn，所以在稍后页面中出现的需要数据源的标签能够使用表达式\${conn}。

为了避免在使用MySQL的JSP页面中重复地列出连接参数，你可以将<sql:setDataSource>标签放在一个文件中，在需要数据库连接的每个页面中包含这个文件。对于发

行版，这个包含文件是WEB-INF/jstl-mcb-setup.inc。JSP页面能够像下面这样访问这个文件来建立数据库连接：

```
<%@ include file="/WEB-INF/jstl-mcb-setup.inc" %>
```

为了改变mcb页面使用的连接参数，编辑jstl-mcb-setup.inc文件。

```
<sql:update>
```

对于像UPDATE、DELETE或者INSERT这样不返回任何行的语句，应使用`<sql:update>`标签。`dataSource`标签属性指定数据源，受影响的语句执行结果行计数在名为var的属性变量中返回，语句本身应该描述为标签体：

```
<sql:update dataSource="\${conn}" var="count">
    DELETE FROM profile WHERE id > 100
</sql:update>
Number of rows deleted: <c:out value="\${count}" />
```

```
<sql:query>
```

为了处理返回一个结果集合的语句，使用`<sql:query>`。像`<sql:update>`一样，`dataSource`属性指示数据源，标签体给出了语句文本。`<sql:query>`标签也有一个var属性，它命名和结果集合相关的变量，这样你可以访问结果行：

```
<sql:query dataSource="\${conn}" var="rs">
    SELECT id, name FROM profile ORDER BY id
</sql:query>
```

根据约定，mcb页面使用rs作为结果集变量的名字。后面会大致介绍访问结果集内容的策略。

```
<sql:param>
```

你可以将数据值逐字地写入语句字符串中，但是JSTL也允许使用占位符，这对包含SQL语句中特殊字符的值是很有帮助的。在语句字符串中，为每一个占位符使用一个?字符，同时要在和语句相关的标签体中使用`<sql:param>`标签的占位符，提供一个与其绑定的值。一个数据值可以在`<sql:param>`标签体内指定，也可以在标签的值属性中指定：

```
<sql:update dataSource="\${conn}" var="count">
    DELETE FROM profile WHERE id > ?
    <sql:param>100</sql:param>
</sql:update>

<sql:query dataSource="\${conn}" var="rs">
    SELECT id, name FROM profile WHERE cats = ? AND color = ?
    <sql:param value="1"/>
    <sql:param value="green" />
</sql:query>
```

有多种方法可以访问`<sql:query>`返回的结果集内容。假设你已经将一个名为`rs`的变量和结果集联系起来，你可以使用`rs.rows[i]`或`rs.rowsByIndex[i]`来访问结果集中的第`i`行，其中行序从0开始。第一种形式产生一行，此行有多个可以通过名字访问的列。第二种形式产生的行中各列可以通过列序号（从0开始）访问。例如，如果结果集中有名为`id`和`name`的列，你可以使用如下列名形式来访问第三行的值：

```
<c:out value="${rs.rows[2].id}" />
<c:out value="${rs.rows[2].name}" />
```

使用列序号的话，应该像下面这样：

```
<c:out value="${rs.rowsByIndex[2][0]}" />
<c:out value="${rs.rowsByIndex[2][1]}" />
```

你也可使用`<c:forEach>`作为迭代器来循环结果集中的行。为了通过名字来访问列值，使用`rs.rows`进行迭代：

```
<c:forEach items="${rs.rows}" var="row">
  id = <c:out value="${row.id}" />,
  name = <c:out value="${row.name}" />
  <br />
</c:forEach>
```

为了通过列序号访问列值，使用`rs.rowsByIndex`来迭代：

```
<c:forEach items="${rs.rowsByIndex}" var="row">
  id = <c:out value="${row[0]}" />,
  name = <c:out value="${row[1]}" />
  <br />
</c:forEach>
```

结果集中的行数可以通过`rs.rowCount`获得：

选择的列的数目：`<c:out value="${rs.rowCount}" />`

结果集中列的名字可以使用`rs.columnNames`获得：

```
<c:forEach items="${rs.columnNames}" var="name">
  <c:out value="${name}" />
  <br />
</c:forEach>
```

使用JSP和JSTL编写MySQL脚本

17.2节中说明了如何编写Perl、Ruby、PHP和Python版本的脚本来显示`cookbook`数据库中的表名。使用JSTL标签，我们可以编写相应的提供那些信息的JSP脚本，如下：

```
<%-- show_tables.jsp - Display names of tables in cookbook database --%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@ include file="/WEB-INF/jstl-mcb-setup.inc" %>
```

```
<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>

<sql:query dataSource="${conn}" var="rs">
    SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_SCHEMA = 'cookbook' ORDER BY TABLE_NAME
</sql:query>

<c:forEach items="${rs.rowsByIndex}" var="row">
    <c:out value="${row[0]}"/><br />
</c:forEach>

</body>
</html>
```

taglib指令确定脚本需要的标签库，include指令导入建立数据源的代码，用以访问 cookbook数据库。脚本余下的部分产生页面内容。

假设你已经如前所述，在你的Tomcat服务器Webapps目录下安装了mcb.war文件，你应该会在mcb子目录找到show_tables.jsp脚本，在你的浏览器中按如下地址请求：

http://localhost:8080/mcb/show_tables.jsp

JSP脚本不明确地产生任何Content-Type: header。JSP引擎自动地产生一个有text/html 内容类型的默认头部。

17.4 在Web输出中编码特殊字符

Encoding Special Characters in Web Output

问题

某些字符在Web页面中是特殊的，如果你想逐字显示它们，必须对其进行编码。因为数据库内容经常包括这些字符的实例，在Web页面中包括查询结果的脚本应该对这些结果进行编码，以防止浏览器曲解这些信息。

解决方案

使用由你的API提供的方法来执行HTML编码和URL编码。

讨论

HTML是一个标记语言：它使用特定的字符作为有特殊含义的标签。为了在页面中包含这些字符实例的原意，你必须对它们进行编码才能避免使它们有特殊含义。例如，<应该编码为<，才能避免浏览器将其解释为一个标签的开始。此外，实际上有两种编码，取决于你使用字符的上下文。一种编码对HTML文本是合适的，另外一种用于在超链接中作为URL一部分的文本。

在17.2和17.3节中的MySQL表格显示脚本，是如何使用程序产生Web页面的简单示例。但是有一个例外，脚本有一个弱点：它们没有考虑到对从MySQL服务器上检索出的信息中的特殊字符编码。(这种例外是脚本的JSP版本。那里使用的<c:out>标签自动地处理编码，我们稍后将作简单的讨论。)

碰巧，我选择用于显示的信息未包含任何特殊字符；即使没有任何编码，脚本也应该能正确地工作。然而，在通常情况下，假设查询结果不包含特殊字符是危险的，所以你必须准备为在Web页面中显示而进行编码。忽视这些将导致在产生页面的脚本中包含畸形的HTML时，显示就会出错。

本节描述了如何处理特殊字符，从一些一般的原则开始，然后讨论每个API如何实现编码支持。特定API的例子展示了如何处理从数据库表中得到的数据，但是这些例子适用于你在Web页面中包含的任何内容，不管内容的来源。

一般编码原则

一种编码形式应用于在编写HTML结构时使用的字符上；另外一种应用于包含在URL中的文本上。明白这个区别是很重要的，这样你不会不适当当地编码。



提示：为Web页面内容编码文本，为SQL语句内容在数据值中编码特殊字符，这两者是完全不同的。2.5节讨论了后者。

编码HTML中的特殊字符。HTML标记使用<and>字符来作为标签的开始和结束，&表示特殊实体名的开始（比如&nbsp表示不换行间隔），使用”将标签中的属性值括起来（比如<p align="left">）。因此，为了显示这些字符本身，你应该将它们编码为HTML实体，这样浏览器或者其他客户端能够明白你的意思。为了做到这一点，应按照下表所示将特殊字符<、>、&和”转换为相应HTML实体。

特殊字符	HTML实体
<	<
>	>
&	&
"	"

假设你想在Web页面中照字符本身显示下面的字符串：

```
Paragraphs begin and end with <p> &lt;/p> tags.
```

如果你将这段文本按上述发送到客户端浏览器，浏览器会曲解它。`<p>`和`</p>`标签会被当作段落标记，`&`又有可能被当作HTML实体指定的开始。为了按你的意图显示这个字符串，必须将特殊字符编码为`<`、`>`和`&`实体：

```
Paragraphs begin and end with &lt;p&gt; &amp;lt;/p&gt; tags.
```

这种方法编码文本的原则在标签内也是有用的。例如，HTML标签属性值通常包含在双引号里，所以在属性值上执行HTML编码是非常重要的。假设你想在一个表单中包含输入文本框，并且你想在文本框中提供一个Rich "Goose" Gossage的初始值来显示。你不能像下面这样在标签里直接写入值本身：

```
<input type="text" name="player_name" value="Rich "Goose" Gossage" />
```

这里的问题是双引号括起来的值属性里还包含双引号，这使得`<input>`标签畸形。合适的编写方式是对双引号进行编码：

```
<input type="text" name="player_name" value="Rich &quot;Goose&quot; Gossage" />
```

当浏览器收到这些文本后，它解码`"`实体为"字符，并正确地解释值属性的值。

在URL中编码特殊字符。出现在HTML页面中的超链接URL有它们自己的语法和编码。这种编码在多个标签里应用到属性上：

```
<a href="URL">

<form action="URL">
<frame src="URL">
```

在URL中的许多字符有特殊含义，比如`:`、`/`、`?`、`=`、`&`和`;`。下面的URL包含其中的一些字符：

```
http://localhost/myscript.php?id=428&name=Gandalf
```

这里`:`和`/`字符将URL分割为几部分，`?`字符表示参数的开始，`&`字符间隔参数，每一个参数都可以描述为`name=pair`对。（在刚才显示的URL中并没有出现`;`字符，但是通常是代替`&`符

来间隔参数。)如果你想在一个URL中按字面意义包含这些字符,你必须对它们进行编码以防浏览器按照它们通常的特殊意义对其进行解释。其他字符,比如空白,也需要特殊处理。空白不允许出现在URL中,所以如果你在本地主机上想引用一个名为my home page.html的页面,像下面这样超链接中的URL就不正确:

```
<a href="http://localhost/my home page.html">My Home Page</a>
```

特殊字符和保留字符的URL编码是将每一个这样的字符转换成%后跟两个该字符ASCII码的十六进制数来表示。例如,空白字符的ASCII码是十进制数32,或十六进制数20,所以你可以像下面这样编写刚才的超链接:

```
<a href="http://localhost/my%20home%20page.html">My Home Page</a>
```

有时你会看到在URL中空白编码成+。这也是合法的。

对内容使用适当的编码方法。确保为你使用的内容正确地进行编码。假设你想创建一个能触发对一个搜索项进行搜索的超链接,并且你想此搜索项本身作为显示在页面里的链接标签出现。这种情况下,搜索项在URL中以一个参数的形式出现,并且作为HTML文本出现在<a>和标签中。如果搜索项是“cats & dogs”,未编码的超链接像下面这样:

```
<a href="/cgi-bin/myscript?term=cats & dogs">cats & dogs</a>
```

这是不正确的,因为&在上下文和空白中都是特殊的。所以链接应该写成下面这样:

```
<a href="/cgi-bin/myscript?term=cats%20%26%20dogs">cats & dogs</a>
```

这里,对于超链接标签&字符使用HTML编码,为&。对于URL则进行URL编码,为%26,其中还包含编码为%20的空白。

当然,在向Web页面里写入文本之前对其进行编码是很麻烦的,有时你要足够了解你能够不进行编码的值。(参见“你需要总是对Web页面输出进行编码吗?”)但是更多时候进行编码是可靠的。幸运的是,很多API为你提供了进行编码的函数。这意味着你需要知道在一个给定的上下文中每一个特殊的字符。你只需要知道执行哪一种编码,这样你就可以调用合适的函数产生想要的结果。

你需要总是对Web页面输出进行编码吗

如果你知道在Web页面内一个特定上下文中的某个值是合法的,你不需要对其进行编码。例如,如果你从一个不能为空的数据库表的整型值列中获取一个值,它就必须是一个整数。在Web页面中包含这个值不需要HTML编码和URL编码,因为数字在HTML文本或在URL中都不是特殊的。另一方面,假设你在Web表单中使用一个域来请求一个整型数。你或许希望用户提供一个整数,但是用户可能不清楚并输入了一个非法值。你可以通过显示一个错误页面来处理这种情况,该错误页面显示该值并解释其不是整数。但是如果这个值中包含特殊字符并且你没有进行编码,那么页面就不能正确显示此值,会更加困扰用户。

使用WebAPI编码特殊字符

下面编码的例子说明了如何从MySQL中获取值并对其执行HTML编码和URL编码来产生链接。每个例子读取包含短语的名为phrase的表，然后使用读取的内容构造超链接，指向一个（假设的）脚本，该脚本在其他一些表中搜索短语的实例。此表包含下面的行：

```
mysql> SELECT phrase_val FROM phrase ORDER BY phrase_val;
+-----+
| phrase_val |
+-----+
| are we "there" yet? |
| cats & dogs |
| rhinoceros |
| the whole > sum of parts |
+-----+
```

这里的目标是产生一个使用每个短语的超链接列表，短语作为超链接标签（这需要HTML编码）以及在URL中作为每个搜索脚本的参数（这需要URL编码）。产生的超链接像下面这样：

```
<a href="/cgi-bin/mysearch.pl?phrase=are%20we%22there%22%20yet%3F">
are we &quot;there&quot; yet?</a>
<a href="/cgi-bin/mysearch.pl?phrase=cats%20%26%20dogs">
cats &amp; dogs</a>
<a href="/cgi-bin/mysearch.pl?phrase=rhinoceros">
rhinoceros</a>
<a href="/cgi-bin/mysearch.pl?phrase=the%20whole%20%3E%20sum%20of%20parts">
the whole &gt; sum of parts</a>
```

href属性值的开始部分每个API会有不同。同样，由一些API产生的链接看上去也会稍有不同，因为它们会将空白编码为+而不是%20。

Perl. Perl的CGI.pm模块提供两个方法，escapeHTML()和escape()，分别处理HTML编码和URL编码。使用它们，有三种方法对字符串\$str进行编码：

- 作为CGI类方法，使用CGI::前缀调用escapeHTML()和escape()：

```
use CGI;
printf "%s\n%s\n", CGI::escape ($str), CGI::escapeHTML ($str);
```

- 创建CGI对象，作为对象的方法调用escapeHTML()和escape()：

```
use CGI;
my $cgi = new CGI;
printf "%s\n%s\n", $cgi->escape ($str), $cgi->escapeHTML ($str);
```

- 在你的脚本的名字空间中明确地导入名字。这种情况下，CGI对象或CGI::前缀都不需要，你可以将它们作为独立函数调用。下面的例子除了标准名字之外，还将两个方法的名字导入：

```
use CGI qw(:standard escape escapeHTML);
printf "%s\n%s\n", escape ($str), escapeHTML ($str);
```

我建议使用最后一种方法，因为它和CGI.pm函数调用接口一致，对其他导入的方法名你也使用这些接口。不过要记住为需要这些方法的Perl脚本中的CGI语句中包含编码方法名，不然当脚本执行时就会碰到“未定义子例程”的错误。

下面的代码读取phrase表中的内容，并使用escapeHTML()和escape()为每个短语产生超链接：

```
my $stmt = "SELECT phrase_val FROM phrase ORDER BY phrase_val";
my $sth = $dbh->prepare ($stmt);
$sth->execute ();
while (my ($phrase) = $sth->fetchrow_array ())
{
    # 对URL中使用的短语进行使用URL编码
    my $url = "/cgi-bin/mysearch.pl?phrase=" . escape ($phrase);
    # 对链接标签中的短语使用HTML编码
    my $label = escapeHTML ($phrase);
    print a ({-href => $url}, $label), br (), "\n";
}
```

Ruby. Ruby的cgi模块包含两个方法，CGI.escapeHTML()和CGI.escape()，分别执行HTML编码和URL编码。然而，如果参数不是字符串，这两个方法都会出现异常。处理这种情况的一种方法是为任何可能不是字符串的参数应用to_s函数，将其强制转换成字符串并将nil转换为空字符串。例如：

```
stmt = "SELECT phrase_val FROM phrase ORDER BY phrase_val"
dbh.execute(stmt) do |sth|
  sth.fetch do |row|
    # 确认该值是字符串
    phrase = row[0].to_s
    # 对URL中使用的短语进行使用URL编码
    url = "/cgi-bin/mysearch.rb?phrase=" + CGI.escape(phrase)
    # 对链接标签中的短语使用HTML编码
    label = CGI.escapeHTML(phrase)
    page << cgi.a("href" => url) { label } + cgi.br + "\n"
  end
end
```

这里使用的page是作为一个变量，它累加页面内容，并最终传递给cgi.out用于显示页面。

PHP. 在PHP中，`htmlspecialchars()`和`urlencode()`函数执行HTML编码和URL编码。它们的使用方式如下：

```
$stmt = "SELECT phrase_val FROM phrase ORDER BY phrase_val";
$result =& $conn->query ($stmt);
if (!PEAR::isError ($result))
{
    while (list ($phrase) = $result->fetchRow ())
    {
        # 对URL中使用的短语进行使用URL编码
        $url = "/mcb/mysearch.php?phrase=" . urlencode ($phrase);
        # 对链接标签中的短语使用HTML编码
        $label = htmlspecialchars ($phrase);
        printf ("<a href=\"%s\"%s</a><br />\n", $url, $label);
    }
    $result->free ();
}
```

Python. 在Python中，`cgi`和`urllib`模块包含相关的编码函数。`cgi.escape()`执行HTML编码，`urllib.quote()`执行URL编码。然而除非它们的参数是字符串，不然这两个函数都会出现异常。一种解决方法是对任何可能不是字符串的参数使用`str()`函数，强制转换成`string`并将`None`转换为字符串“`None`”。（如果你想将`None`转换成空字符串，你需要为此明确地作测试。）例如：

```
import cgi
import urllib

stmt = "SELECT phrase_val FROM phrase ORDER BY phrase_val"
cursor = conn.cursor ()
cursor.execute (stmt)
for (phrase,) in cursor.fetchall ():
    # 确认该值是字符串
    phrase = str (phrase)
    # 对URL中使用的短语进行使用URL编码
    url = "/cgi-bin/mysearch.py?phrase=" + urllib.quote (phrase)
    # 对链接标签中的短语使用HTML编码
    label = cgi.escape (phrase, 1)
    print "<a href=\"%s\"%s</a><br />" % (url, label)
cursor.close ()
```

`cgi.escape()`的第一个参数是将要进行HTML编码的字符串。默认地，这个函数将`<`、`>`和`&`字符转换成HTML实体。为了告诉`cgi.escape()`要将双引号转换成`"`实体，需要传递`1`给第二个参数，就像例子中那样。如果你在对要替换到双引号括起的属性的值进行编码时，这是很重要的。

Java. <c:out>JSTL标签为JSP页面自动地执行HTML编码。（严格地说，它执行XML编码，但是受影响的字符集是`<`、`>`、`&`、`"`和`'`，这些包含所有需要进行HTML编码的字符。）通过使用<c:out>在Web页面中显示文本，你无须考虑特殊字符到HTML实体之间的转换。如果由于某些原因，你想要禁止编码，调用带有值为`false`的`encodeXML`属性的<c:out>：

```
<c:out value="value to display" encodeXML="false"/>
```

对包含在URL中的参数进行URL编码，要使用<c:url>标签。在标签的值属性中指定URL字符串，并在<c:url>标签体中使用<c:param>标签，一些参数值和名字都包含在<c:param>标签中。一个参数值可以在<c:param>标签的属性值中给出，也可以在标签体中给出。这里有两种使用方法的例子：

```
<c:url var="urlStr" value="myscript.jsp">
  <c:param name="id" value = "47"/>
  <c:param name="color">sky blue</c:param>
</c:url>
```

这将会对id的值和color参数进行URL编码，并将它们增加到URL的尾部。结果放置在名为urlStr的对象里，你可以如下显示：

```
<c:out value="${urlStr}"/>
```



提示：<c:url>标签不会对特殊字符进行编码，比如在它的值属性中给出的字符串中包含的空白。你必须自己对其进行编码，所以最好避免创建在名字中带有空白的页面，消除你需要引用它们的可能性。

为了显示phrase表中的条目，如下使用<c:out>和<c:url>标签：

```
<sql:query dataSource="${conn}" var="rs">
  SELECT phrase_val FROM phrase ORDER BY phrase_val
</sql:query>

<c:forEach items="${rs.rows}" var="row">
  <%-- URL-encode the phrase value for use in the URL --%>
  <c:url var="urlStr" value="/mcb/mysearch.jsp">
    <c:param name="phrase" value ="${row.phrase_val}" />
  </c:url>
  <a href=<c:out value="${urlStr}" />>
    <%-- HTML-encode the phrase value for use in the link label --%>
    <c:out value ="${row.phrase_val}" />
  </a>
  <br />
</c:forEach>
```


在Web页面中混合查询结果

Incorporating Query Results into Web Pages

18.0 引言

Introduction

当你把信息存储在数据库中时，你可以通过Web以各种方式轻松地获取这些信息。查询的结果能够以非结构化的段落、或者结构化的元素如列表或表格显示出来。此外，查询结构还能被显示为静态文本或超链接。查询元数据在格式化查询结果时是非常有用的，比如当我们生成一个基于HTML的表格来显示一个结构集时，常常需要使用元数据来获知表格的列标题。类似的任务结合了处理Web脚本的语句，一般来讲需要考虑很多问题，比如对查询结果做恰当的编码处理（如&或<），为你想产生的元素添加合适的HTML标签等。

本章将介绍如何从查询结果中生成以下几种类型的Web输出：

- 段落
- 列表
- 表格
- 超链接
- 导航索引(单页面和多页面)

同时，本章还会介绍如何将二进制数据插入你的数据库中，以及如何读取二进制数据并将它传输到客户端。数据库操作中最简单和最常见的是对文本信息的操作，但MySQL数据库同样也能让二进制信息（如图片、声音或者PDF文件）的处理变得简单。同样地，通过MySQL数据库，你还能够提供查询结果下载而非仅仅将其显示在页面上。最后，本章将讨论如何使用模板包来生成Web页面。

本章是建立在第17章中的技术之上的，这些技术主要用于从脚本中生成Web页面，以及编码输出和显示。如果你需要相关的背景知识，请参见第17章的内容。

本章中用来生成表格的脚本存放在tables目录中。以Perl、Ruby、PHP和Python脚本为例，请查看Apache目录（请阅读17.3节以了解如何配置Apache服务器）。示例脚本中常用的程序存放在lib目录中。如果想阅读Java（JSP）相关的示例，请查看tomcat目录，我们假设你在建立mcb应用环境的过程中，已经安装了tomcat。

须注意的是，虽然本章的脚本一般由您的浏览器调用，但其中大部分脚本（除JSP页面外）还可以通过命令行的方式直接调用。

每一小节的讨论并不包含所有的语言，如果有一小节正好没有您想要的语言示例，请查看安装光盘。

18.1 以段落文本显示查询结果

Displaying Query Results as Paragraph Text

问题

你想以自由文本的方式显示一个查询的结果。

解决方案

在段落标签中显示查询结果。

讨论

当显示没有特定结构的自由文本时，段落是非常有用的方式。在这种情况下，你要做的就是获取需要显示的文本，把它编码成HTML对应的文本，然后把编码后的文本用段落标签<p>和</p>封装起来。下面的例子显示了如何为一个状态信息（包含当前的日期、时间、服务器的版本、客户姓名以及数据库名字）生成段落的过程。上述这些值可以通过以下SQL查询获得：

```
mysql> SELECT NOW(), VERSION(), USER(), DATABASE();
+-----+-----+-----+-----+
| NOW() | VERSION() | USER() | DATABASE() |
+-----+-----+-----+-----+
| 2006-10-17 15:47:33 | 5.0.27-log | cbuser@localhost | cookbook |
+-----+-----+-----+-----+
```

在Perl中，CGI.pm模块提供了一个p()函数，这个函数能给你传送给它的文本加上段落标签。

由于p()不使用HTML编码它的参数，所以当函数调用escapeHTML()时必须特别小心：

```
($now, $version, $user, $db) =
  $dbh->selectrow_array ("SELECT NOW(), VERSION(), USER(), DATABASE()");
$db = "NONE" unless defined ($db);
print p (escapeHTML ("Local time on the MySQL server is $now."));
print p (escapeHTML ("The server version is $version."));
print p (escapeHTML ("The current user is $user."));
print p (escapeHTML ("The default database is $db."));
```

在Ruby中，使用CGI模块的escapeHTML()函数去编码段落文本，然后传送给它p函数以生成段落标签：

```
(now, version, user, db) =
  dbh.select_one("SELECT NOW(), VERSION(), USER(), DATABASE()")
db = "NONE" if db.nil?
cgi = CGI.new("html4")
cgi.out =
  cgi.p { CGI.escapeHTML("Local time on the MySQL server is #{now}.") } +
  cgi.p { CGI.escapeHTML("The server version is #{version}.") } +
  cgi.p { CGI.escapeHTML("The current user is #{user}.") } +
  cgi.p { CGI.escapeHTML("The default database is #{db}.") }
```

在PHP中，把<p>和</p>标签直接放在编码后的段落文本前后：

```
$result =& $conn->query ("SELECT NOW(), VERSION(), USER(), DATABASE()");
if (!PEAR::isError ($result))
{
  list ($now, $version, $user, $db) = $result->fetchRow ();
  $result->free ();
  if (!isset ($db))
    $db = "NONE";
  $para = "Local time on the MySQL server is $now.";
  print ("<p>" . htmlspecialchars ($para) . "</p>\n");
  $para = "The server version is $version.";
  print ("<p>" . htmlspecialchars ($para) . "</p>\n");
  $para = "The current user is $user.";
  print ("<p>" . htmlspecialchars ($para) . "</p>\n");
  $para = "The default database is $db.";
  print ("<p>" . htmlspecialchars ($para) . "</p>\n");
}
```

或者，在获取查询结果后，您还可以通过开始一个新的HTML模块，然后在模块间切换的方法打印一段文本：

```
<p>Local time on the MySQL server is
<?php print (htmlspecialchars ($now)); ?>.</p>
<p>The server version is
<?php print (htmlspecialchars ($version)); ?>.</p>
<p>The current user is
<?php print (htmlspecialchars ($user)); ?>.</p>
<p>The default database is
<?php print (htmlspecialchars ($db)); ?>.</p>
```

在Python中显示段落，可以这么做：

```
cursor = conn.cursor ()
cursor.execute ("SELECT NOW(), VERSION(), USER(), DATABASE()")
(now, version, user, db) = cursor.fetchone ()
cursor.close ()
if db is None: # check database name
    db = "NONE"
para = ("Local time on the MySQL server is %s.") % now
print "<p>" + cgi.escape (para, 1) + "</p>"
para = ("The server version is %s.") % version
print "<p>" + cgi.escape (para, 1) + "</p>"
para = ("The current user is %s.") % user
print "<p>" + cgi.escape (para, 1) + "</p>"
para = ("The default database is %s.") % db
print "<p>" + cgi.escape (para, 1) + "</p>"
```

在JSP中，段落可以用如下方式显示：用rowsByIndex方法通过数据索引获取数据集中每一行的所有列，然后用<c:out>方法编码并输出文本：

```
<sql:query dataSource="${conn}" var="rs">
    SELECT NOW(), VERSION(), USER(), DATABASE()
</sql:query>
<c:set var="row" value="${rs.rowsByIndex[0]}"/>
<c:set var="db" value="${row[3]}"/>
<c:if test="${empty db}">
    <c:set var="db" value="NONE"/>
</c:if>

<p>Local time on the server is <c:out value="${row[0]}"/>.</p>
<p>The server version is <c:out value="${row[1]}"/>.</p>
<p>The current user is <c:out value="${row[2]}"/>.</p>
<p>The default database is <c:out value="${db}"/>.</p>
```

参见

18.10节讨论了如何使用模板产生段落。

18.2 以列表形式显示查询结果

Displaying Query Results as Lists

问题

查询结果包含一组应显示为结构化列表的项。

解决方案

HTML列表一共有几种，把这些列表项写在对应列表的标签之间。

讨论

列表是位于表格和段落之间的一种显示方式，它有助于我们显示一组独立的项。HTML提供了几种风格的列表，比如有序列表、无序列表、以及定义列表。当然，你或许还希望使用嵌套列表，那要求你在列表中使用列表格式。

一般来讲，列表由开放及关闭两类标签组成，这两类标签中又包含一系列的数据项，每一个数据项都由它自己的标签所限定。列表数据项很自然地对应到一个查询所返回的行，所以在程序中生成一个HTML列表结构的过程，其实就是编码你的查询结果，并把每行都放入对应的标签对，并最终用列表的开放和关闭标签包含起来的过程。

两种列表产生方法是常用的。为了能在处理查询结果集时打印标签，你可以：

1. 打印列表的开放标签。
2. 获取结果集的每一行，并把它当作一个列表项来打印，包含每个项的标签。
3. 打印列表的关闭标签。

此外，你还可以在内存中处理列表：

1. 将所有的数据项存在一个数组中。
2. 把数组传给列表生成函数，该函数负责添加合适的标签对。
3. 输出结果。

下面通过示例来说明上述这两种方法的使用情况。

有序列表

一个有序列表中的数据项，是按照一个特定的顺序排列的。典型地，浏览器将有序列表显示为一组带数字下标的项：

1. First item
2. Second item
3. Third item

你并不需要规定数据项的下表数字，因为浏览器可以自动添加。在HTML语法中，有序列表通常由标签``和``限定，列表中的数据项由``和``标签对限定：

```
<ol>
  <li>First item</li>
  <li>Second item</li>
  <li>Third item</li>
</ol>
```

假设你现在有一个成分表格，这个表格中包含一个菜单的各个成分：

id	item
----	------

	1		3	cups	flour		
	2		1/2	cup	raw ("unrefined")	sugar	
	3		3	eggs			
	4		pinch	(< 1/16	teaspoon)	salt	

这张表格中包含一个id列，但你只需要以合适的顺序取出文本值，以有序列表形式来显示它们即可。这是因为浏览器会自动为你添加对应的数字。由于数据项中包含特殊字符，如"和<，所以在使用将项转换为HTML列表的标签之前，你应该对它们进行HTML编码。代码如下：

```
<ol>
  <li>3 cups flour</li>
  <li>1/2 cup raw ("unrefined") sugar</li>
  <li>3 eggs</li>
  <li>pinch (< 1/16 teaspoon) salt</li>
</ol>
```

一种从脚本中生成上述列表的方法，是当你从结果集中获取行时打印HTML。这里展示了你如何通过在JSP页面中使用JSTL标签来做到这一点：

```
<sql:query dataSource="${conn}" var="rs">
  SELECT item FROM ingredient ORDER BY id
</sql:query>
<ol>
  <c:forEach items="${rs.rows}" var="row">
    <li><c:out value="${row.item}" /></li>
  </c:forEach>
</ol>
```

同样的工作，在PHP中我们可以这么做：

```
$stmt = "SELECT item FROM ingredient ORDER BY id";
$result =& $conn->query ($stmt);
if (PEAR::isError ($result))
  die (htmlspecialchars ($result->getMessage ()));
print ("<ol>\n");
while (list ($item) = $result->fetchRow ())
  print ("<li>" . htmlspecialchars ($item) . "</li>\n");
$result->free ();
print ("</ol>\n");
```

实际上可以不必像这个例子中那样，在关闭标签后加上换行符；Web浏览器是不理睬这个换行符是否存在的。我喜欢加上换行符，是因为由一个脚本生成的HTML更易于直接检查，如果不完全在单独的一个行上，并简化调试。

上面的例子使用了一种HTML代码生成的方法，这种方法是在获取和输出阶段插入行。我们完全可以将这两个阶段分开（或者拆分）：先获取数据，然后输出。每一个列表所对应的查询语句都是不同的，但生成列表的过程却几乎是一样的。如果你将列表生成代码放入一个

工具函数中，你将可以在多个查询中复用这个工具函数。使用这个工具函数有两个问题需要处理：HTML编码（如果你还未编码）和添加合适的HTML标签。例如，一个名为 make_ordered_list() 的函数能在PHP中如下编写，它以列表项作为数组参数，并返回一个字符串形式的列表：

```
function make_ordered_list ($items, $encode = TRUE)
{
    $str = "<ol>\n";
    foreach ($items as $k => $v)
    {
        if ($encode)
            $v = htmlspecialchars ($v);
        $str .= "<li>$v</li>\n";
    }
    $str .= "</ol>\n";
    return ($str);
}
```

在编写完这个工具函数后，你可以先读取数据，然后再输出HTML：

```
# 获取用于列表的项
$stmt = "SELECT item FROM ingredient ORDER BY id";
$result =& $conn->query ($stmt);
if (PEAR::isError ($result))
    die (htmlspecialchars ($result->getMessage ()));
$item = array ();
while (list ($item) = $result->fetchRow ())
    $item[] = $item;
$result->free ();

# 生成HTML列表
print (make_ordered_list ($item));
```

这个工具函数，在Python中可以定义为：

```
def make_ordered_list (items, encode = True):
    list = "<ol>\n"
    for item in items:
        if item is None: # 处理NULL项的可能性
            item = ""
        # 确定该项是一个字符串，然后进行必要的编码
        item = str (item)
        if encode:
            item = cgi.escape (item, 1)
        list = list + "<li>" + item + "</li>\n"
    list = list + "</ol>\n"
    return list.
```

然后以如下方式使用：

```
# 获取用于列表的项
stmt = "SELECT item FROM ingredient ORDER BY id"
cursor = conn.cursor ()
cursor.execute (stmt)
items = []
```

```
for (item,) in cursor.fetchall():
    items.append (item)
cursor.close ()

# 生成HTML列表
print make_ordered_list (items)
```

函数make_ordered_list()的第二个参数表示它是否应该对列表项进行HTML编码。最简单的方法当然是让函数为你做这个工作（所以默认值是true）。然而，如果你从一组已经包含了HTML标签的数据项中生成列表，那么你就不需要该函数为你在那些标签里进行特殊字符编码了。举个例子，假设你正在生成一组超链接的列表，每个列表项都会包含标签。为了防止这些标签被转化为<lt;a>，你需要向make_ordered_list()函数传递的第二个参数就必须为false。

如果你的API提供了函数，以让你生成HTML结构，当然你就不需要写上述的函数了。例如，当使用Perl CGI.pm和Ruby CGI模块时，使用Perl，通过调用它的li()函数为每个数据项添加开放和关闭标签，这样可以节省往组中写入的标签项。然后把输出传输给ol()以添加列表的形式开放和关闭标签：

```
my $stmt = "SELECT item FROM ingredient ORDER BY id";
my $sth = $dbh->prepare ($stmt);
$sth->execute ();
my @items = ();
while (my $ref = $sth->fetchrow_arrayref ())
{
    # 处理NULL (undef) 项的可能性
    my $item = (defined ($ref->[0]) ? escapeHTML ($ref->[0]) : "");
    push (@items, li ($item));
}
print ol (@items);
```

之所以将NULL（以`undef`表示）转化为空字符串，这是为了在使用警告功能时，避免Perl提出未初始化变量的警告。（上例中，`ingredient`表并不包含NULL，但这是一项在实际编程中非常有用的技术。）

上述的例子，将行获取和HTML生成捆绑在一起了。其实，我们可以将这两个步骤分开。先把获取的数据项存入一个数组，然后把数组作为参数传送给li()，并将结果传送给ol()：

```
# 获取用于列表的项
my $stmt = "SELECT item FROM ingredient ORDER BY id";
my $item_ref = $dbh->selectcol_arrayref ($stmt);

# 生成HTML列表，处理NULL (undef) 项的可能性
$item_ref = [ map { defined ($_) ? escapeHTML ($_) : "" } @{$item_ref} ];
print ol (li ($item_ref));
```

关于li()函数，有两个需要注意的事项：

- 它不执行任何的HTML编码操作，你必须亲自做。
- 它可以处理单一的值或一个数组。然而，如果你传递一个数组，你应该使用数组引用。当你这么做时，`li()`会对每个数组元素增加``和``标签，然后将它们连在一起，返回结果字符串。如果你传递一个数组本身而不是引用，`li()`函数首先连接每个项，然后随结果增加一个标签对，这通常不是你所期望的。许多其他的能够对单一值或多个值进行操作的CGI.pm函数也具有这一行为。例如，当你传递给表数据函数`td()`一个标量或列表时，它也会增加一对`<td>`和`</td>`标签。如果你传递一个列表引用，那么它为列表中的每一项增加一对标签。

你是否应该将行获取和HTML生成分开？

如果你正在急匆匆地写一个脚本，那么把这两个操作混在一起编写能让你很快就把系统运行起来。但将两者分开无疑是有好处的，最明显的就是通过使用一个工具函数来生成HTML代码，该函数只需编写一遍，然后就可以在多个脚本中复用。此外，还有其他的一些好处：

- 生成HTML结构的函数能使用来自其他数据源的数据，而不仅仅是来自数据库的。
- 你不必直接处理输出。你可以在内存中构建一个页面元素，然后在你准备好之后再输出。这在构建包含多个部分的页面时，是非常有用的，因为它让你在按一定顺序生成多个部分时，能享有很大的自由。另一方面，如果你要处理的数据集非常大时，这个方法将占用非常多的内存空间。
- 在你生成的输出类型上，将行获取和输出生成分开会给你更多的灵活性。如果你决定生成一个无序列表，而不是有序的，你只需调用不同的输出函数便可，而数据收集的过程并不需要改变。这点总是有效的，哪怕你是采用不同的输出格式（如XML或WML，而不是HTML）。这种情况下，你仍然只需要不同的输出函数，而数据收集保持不变。
- 通过提前获取列表的数据项，你可以非常灵活地决定生成哪种列表。尽管我们现在还不到讨论Web Form的时候，但它们往往严重依赖于对应的列表类型。在那种情况下，先获取数据项，然后再生成HTML结构是非常有用的，因为这样我们可以根据列表的大小来决定它的类型。比如说，当数据项的数目很小时，你可以输出一组radio按钮，或者下拉式菜单；如果数目很大时，你可以生成带滚动条的列表。

前面同样的示例，用Ruby编写代码如下：

```
# 获取用于列表的项  
stmt = "SELECT item FROM ingredient ORDER BY id"
```

```
items = dbh.select_all(stmt)

list = ""
items.each do |item|
  list << cgi.li { CGI.escapeHTML(item.to_s) }
end
list = cgi.ol { list }
```

无序列表

无序列表与有序列表非常相似，区别在于浏览器使用同样的标记符号显示所有数据项，比如一个bullet：

- First item
- Second item
- Third item

所谓无序，指的是标记字符提供无序信息。当然你还是可以把数据项以任意你想要的顺序显示出来。HTML的无序列表标签跟有序列表是一样的，区别是开放标签和关闭标签变为和，而不是和：

```
<ul>
  <li>First item</li>
  <li>Second item</li>
  <li>Third item</li>
</ul>
```

对于那些你直接打印标签的API而言，它采用的操作过程和有序列表是一样的，但是用和代替了和。以JSP为例：

```
<sql:query dataSource="${conn}" var="rs">
  SELECT item FROM ingredient ORDER BY id
</sql:query>
<ul>
  <c:forEach items="${rs.rows}" var="row">
    <li><c:out value="${row.item}" /></li>
  </c:forEach>
</ul>
```

在Perl中，使用CGI.pm模块的ul()函数生成一个无序列表，而不是在有序列表时的ol()：

```
# 获取用于列表的项
my $stmt = "SELECT item FROM ingredient ORDER BY id";
my $item_ref = $dbh->selectcol_arrayref ($stmt);

# 生成HTML列表，处理NULL (undef) 项的可能性
$item_ref = [ map { defined ($_) ? escapeHTML ($_) : "" } @{$item_ref} ];
print ul (li ($item_ref));
```

如果你正在为无序列表写自己的工具函数，那么从一个生成有序列表的函数中直接派生出新的函数是非常简单的。例如，可以很简单地更改make_ordered_list()函数生成make_unordered_list()函数，因为两者的区别仅在于开放标签和关闭标签。

定义列表

一个定义列表包含两部分内容，每个都包含有一个术语和一个定义。术语和定义有着并不严格的含义，因为HTML允许你显示任何类型的信息。例如，下面的表格将音乐中的音阶和记忆短句连在了一起，以方便人们的记忆，但记忆短句在这里可不是音阶的解释：

id	note	mnemonic
1	do	A deer, a female deer
2	re	A drop of golden sun
3	mi	A name I call myself
4	fa	A long, long way to run
5	so	A needle pulling thread
6	la	A note to follow so
7	ti	I drink with jam and bread

不管怎样，在这里音符和记忆短句被显示为定义列表：

```
do
A deer, a female deer
re
A drop of golden sun
mi
A name I call myself
fa
A long, long way to run
so
A needle pulling thread
la
A note to follow so
ti
I drink with jam and bread
```

在HTML中，定义列表以`<dl>`开始，以`</dl>`结束。每个数据项都包含一个术语，被`<dt>`和`</dt>`标签包围，以及一个定义，被`<dd>`和`</dd>`标签所包围：

```
<dl>
<dt>do</dt> <dd>A deer, a female deer</dd>
<dt>re</dt> <dd>A drop of golden sun</dd>
<dt>mi</dt> <dd>A name I call myself</dd>
<dt>fa</dt> <dd>A long, long way to run</dd>
<dt>so</dt> <dd>A needle pulling thread</dd>
<dt>la</dt> <dd>A note to follow so</dd>
<dt>ti</dt> <dd>I drink with jam and bread</dd>
</dl>
```

在JSP中，你能够以下方式生成定义列表：

```
<sql:query dataSource="${conn}" var="rs">
    SELECT note, mnemonic FROM doremi ORDER BY note
</sql:query>
```

```

<dl>
<c:forEach items="${rs.rows}" var="row">
  <dt><c:out value="${row.note}" /></dt>
  <dd><c:out value="${row.mnemonic}" /></dd>
</c:forEach>
</dl>

```

在PHP中，这个过程如下：

```

$stmt = "SELECT note, mnemonic FROM doremi ORDER BY id";
$result =& $conn->query ($stmt);
if (PEAR::isError ($result))
  die (htmlspecialchars ($result->getMessage ()));
print ("<dl>\n");
while (list ($note, $mnemonic) = $result->fetchRow ())
{
  print ("<dt>" . htmlspecialchars ($note) . "</dt>\n");
  print ("<dd>" . htmlspecialchars ($mnemonic) . "</dd>\n");
}
$result->free ();
print ("</dl>\n");

```

当然，我们可以先获取数据，然后把它传送给一个带有术语和定义的数组的工具函数，此函数返回一个字符串形式的列表：

```

# 获取用于列表的项
$stmt = "SELECT note, mnemonic FROM doremi ORDER BY id";
$result =& $conn->query ($stmt);
if (PEAR::isError ($result))
  die (htmlspecialchars ($result->getMessage ()));
$terms = array ();
$defs = array ();
while (list ($note, $mnemonic) = $result->fetchRow ())
{
  $terms[] = $note;
  $defs[] = $mnemonic;
}
$result->free ();

# 生成HTML列表
print (make_definition_list ($terms, $defs));

```

make_definition_list() 函数可以写成这样：

```

function make_definition_list ($terms, $definitions, $encode = TRUE)
{
  $str = "<dl>\n";
  $n = count ($terms);
  for ($i = 0; $i < $n; $i++)
  {
    $term = $terms[$i];
    $definition = $definitions[$i];
    if ($encode)
    {
      $term = htmlspecialchars ($term);
      $definition = htmlspecialchars ($definition);
    }
  }
  $str .= "</dl>";
  return $str;
}

```

```
    $str .= "<dt>$term</dt>\n<dd>$definition</dd>\n";
}
$str .= "</dl>\n";
return ($str);
}
```

在Ruby中，使用dt和dd方法来生成列表项内容，然后将其传递给dl方法来增加最外面的列表标签：

```
stmt = "SELECT note, mnemonic FROM doremi ORDER BY id"
list = ""
dbh.execute(stmt) do |sth|
  sth.fetch do |row|
    list << cgi.dt { CGI.escapeHTML(row["note"].to_s) }
    list << cgi.dd { CGI.escapeHTML(row["mnemonic"].to_s) }
  end
end
list = cgi.dl { list }
```

这里我们再给出另一个例子（在Perl中）。每个术语都是一个数据库名字，对应的定义表明数据库中有多少个表。我们使用一个查询从INFORMATION_SCHEMA中获取对应的数字。你可以调用dt()和dd()函数生成术语及定义，然后将其存入数组，最后把数组传给dl()函数：

```
# 对每个数据库的表数目进行统计
my $sth = $dbh->prepare ("SELECT TABLE_SCHEMA, COUNT(TABLE_NAME)
                           FROM INFORMATION_SCHEMA.TABLES
                           GROUP BY TABLE_SCHEMA");
$sth->execute ();
my @items = ();
while (my ($db_name, $tbl_count) = $sth->fetchrow_array ())
{
  push (@items, dt (escapeHTML ($db_name)));
  push (@items, dd (escapeHTML ($tbl_count . " tables")));
}
print dl (@items);
```

counts指的是，当连接到MySQL服务器时，该脚本使用的MySQL账号所能够访问的表的数目。如果存在数据库或表格是该账号所不能访问的，那么你就无法获取对应的计数信息。

非标记列表

这是一个完全没有标记的列表，它的类型不能像一般讨论那样探讨。它是一组数据项，每个数据项对应单独的一行。生成一个非标记列表是非常容易的：获取每一行，并在其后添加一个分隔标签。以JSP为例：

```
<c:forEach items="${rs.rows}" var="row">
  <c:out value="${row.item}" /><br />
</c:forEach>
```

如果你已经把数据项存入数组，那么循环读取便可。例如，在Ruby中，如果你有一组项集合在一个名为`items`的数组中，可以按如下生成列表：

```
list = items.collect { |item| CGI.escapeHTML(item.to_s) + cgi.br }.join
```

嵌套列表

当作为一个列表的列表表示时，一些应用程序显示的信息更容易理解。以下例子将姓名显示为定义列表，并根据首字符将这些定义列表分组。对于列表中的每一个数据项，术语就是首字符，而定义就是一个无序列表，该列表的项是以那个字符开始的状态名字：

```
A
  o Alabama
  o Alaska
  o Arizona
  o Arkansas
C
  o California
  o Colorado
  o Connecticut
D
  o Delaware
...
```

一种使用Perl生成这种列表的方式是：

```
# 获取起始字符列表
my $ltr_ref = $dbh->selectcol_arrayref (
    "SELECT DISTINCT UPPER(LEFT(name,1)) AS letter
     FROM states ORDER BY letter");
my @items = ();
# 获得每个字符的州列表
foreach my $ltr (@{$ltr_ref})
{
    my $item_ref = $dbh->selectcol_arrayref (
        "SELECT name FROM states WHERE LEFT(name,1) = ?
         ORDER BY name", undef, $ltr);
    $item_ref = [ map { escapeHTML($_) } @{$item_ref} ];
    # 将州列表转换为未排序的列表
    my $item_list = ul (li @{$item_ref});
    # 对于每个定义列表项，起始字符是条目，州列表是定义
    push (@items, dt ($ltr));
    push (@items, dd ($item_list));
}
print dl (@items);
```

这个例子使用一个查询获取不同的字符，然后使用另一个查询为每个字符寻找对应的名字。你也可以使用单独一个查询获取所有的信息，然后在结果集中遍历，每到一个新的字符时开始一个新的列表项：

```
my $sth = $dbh->prepare ("SELECT name FROM states ORDER BY name");
$sth->execute ();
```

```

my @items = ();
my @names = ();
my $cur_ltr = "";
while (my ($name) = $sth->fetchrow_array ())
{
    my $ltr = uc (substr ($name, 0, 1)); # 名称的起始字符
    if ($cur_ltr ne $ltr)           # 开始一个新的字母?
    {
        if (@names)      # 前一字符的任意存储的名称
        {
            # 对于每个定义列表项，起始字符是条目，州列表是定义
            push (@items, dt ($cur_ltr));
            push (@items, dd (ul (li (\@names))));
        }
        @names = ();
        $cur_ltr = $ltr;
    }
    push (@names, escapeHTML ($name));
}
if (@names)          # 最后的字母的任意保留的名称
{
    push (@items, dt ($cur_ltr));
    push (@items, dd (ul (li (\@names))));
}
print dl (@items);

```

第三种方法使用单独一个查询，但却将数据收集和HTML生成分成了两个阶段：

```

# 收集州名，并将其和合适的起始字符列表关联起来
my $sth = $dbh->prepare ("SELECT name FROM states ORDER BY name");
$sth->execute ();
my %litr = ();
while (my ($name) = $sth->fetchrow_array ())
{
    my $ltr = uc (substr ($name, 0, 1)); # 名称的起始字母
    # 如果这是该字母的第一个州，将字母表初始化为一个空数组，然后将州添加到数组中
    $litr{$ltr} = [] unless exists ($litr{$ltr});
    push (@{$litr{$ltr}}, $name);
}

# 现在生成输出列表
my @items = ();
foreach my $ltr (sort (keys (%litr)))
{
    # 编码该字母的州名列表，生成未排序的列表
    my $ul_str = ul (li (map { escapeHTML ($_) } @{$litr{$ltr}}));
    push (@items, dt ($ltr), dd ($ul_str));
}
print dl (@items);

```

参考

18.10节讨论了两种使用模板生成列表的方法。

18.3 以表格形式显示查询结果

Displaying Query Results as Tables

问题

你希望把一个查询结果显示为HTML表格。

解决方案

把查询结果集当中的每一行当作表格的每一行。如果你需要一个初始的行以显示列标题，请自己提供这一行，或者是通过查询元数据来获得结果集中各列的名字。

讨论

HTML表格对于呈现高度结构化输出是非常有用的。表格之所以流行的一个重要原因，是它们包含了行和列，因此在表格和查询结果集之间存在一个非常自然的对应关系。此外，你还可以通过访问查询元数据来获取列标题。一个HTML表格的基本结构如下：

- 表格以`<table>`和`</table>`标签对为始终，且包含一组行。
- 每行都以`<tr>`和`</tr>`标签对为始终，且每行包含一组单元。
- 每个单元都以`<td>`和`</td>`标签对为始终，头单元的限定标签对是`<td>`和`</th>`。（典型地，浏览器会以黑体或其他的突出形式来显示头单元。）
- 标签中可以包含属性。例如，在单元四周加上边框只需要将`border="1"`放入`<table>`标签中即可。又例如，让一个单元中的内容右对齐，只需在`<td>`标签中写`align="right"`。

需要注意的是，你必须为每个表格元素提供结束标签。这对任一HTML元素而言通常是个好习惯，这点对表格尤其重要。如果你省略了某个结束标签，后果将是不可预测的浏览器行为。

假设现在你想显示CD收藏中的内容：

```
mysql> SELECT year, artist, title FROM cd ORDER BY artist, year;
+-----+-----+-----+
| year | artist          | title        |
+-----+-----+-----+
| 1999 | Adrian Snell    | City of Peace |
| 1999 | Charlie Peacock  | Kingdom Come   |
| 2004 | Dave Bainbridge  | Veil of Gossamer |
| 1990 | Iona              | Iona          |
| 2001 | Iona              | Open Sky       |
```

1998 jaci velasquez	jaci velasquez	
1989 Richard Souther	Cross Currents	
1987 The 77s	The 77s	
1982 Undercover	Undercover	

要用一个有边框的HTML表格来显示这个查询结果集，你需要生成这样的HTML代码：

```
<table border="1">
<tr>
  <th>Year</th>
  <th>Artist</th>
  <th>Title</th>
</tr>
<tr>
  <td>1999</td>
  <td>Adrian Snell</td>
  <td>City of Peace</td>
</tr>
<tr>
  <td>1999</td>
  <td>Charlie Peacock</td>
  <td>Kingdom Come</td>
</tr>
<tr>
  ... other rows here ...
</tr>
<td>1982</td>
<td>Undercover</td>
<td>Undercover</td>
</tr>
</table>
```

将查询结果集转换为HTML表格，需要将每个行中的值包含在表格单元标签内，每行在一个行标签里，整个结果集放在表格标签中。一个JSP页面往往从如下cd表格查询中生成HTML表格：

```
<table border="1">
<tr>
  <th>Year</th>
  <th>Artist</th>
  <th>Title</th>
</tr>
<sql:query dataSource="${conn}" var="rs">
  SELECT year, artist, title FROM cd ORDER BY artist, year
</sql:query>
<c:forEach items="${rs.rows}" var="row">
  <tr>
    <td><c:out value="${row.year}" /></td>
    <td><c:out value="${row.artist}" /></td>
    <td><c:out value="${row.title}" /></td>
  </tr>
</c:forEach>
</table>
```

在Perl脚本中，CGI.pm函数table()、tr()、td()以及th()用来生成表格、行、数据单元以及头单元。然而，产生表格行的函数tr()应该被作为Tr()来调用，这是为了不和Perl中内建的一个名为tr()的转换字符的函数发生冲突（注1）。于是，将cd表显示为HTML表格的过程在Perl中为：

```
my $sth = $dbh->prepare ("SELECT year, artist, title
                           FROM cd ORDER BY artist, year");
$sth->execute ();
my @rows = ();
push (@rows, Tr (th ("Year"), th ("Artist"), th ("Title")));
while (my ($year, $artist, $title) = $sth->fetchrow_array ())
{
    push (@rows, Tr (
        td (escapeHTML ($year)),
        td (escapeHTML ($artist)),
        td (escapeHTML ($title))
    )));
}
print table ({-border => "1"}, @rows);
```

有时候，如果你将相邻行以交替的两种颜色显示，那么表格中的内容将更易于阅读。为了达到这个效果，需要在每个<th>和<td>中添加bgcolor属性，然后设置每行的颜色。常用的一个简单办法是使用一个变量。在以下示例中，\$bgcolor变量总是在silver和white两个值之间交替变换：

```
my $sth = $dbh->prepare ("SELECT year, artist, title
                           FROM cd ORDER BY artist, year");
$sth->execute ();
my $bgcolor = "silver"; # 行颜色变量
my @rows = ();
push (@rows, Tr (
    th ({-bgcolor => $bgcolor}, "Year"),
    th ({-bgcolor => $bgcolor}, "Artist"),
    th ({-bgcolor => $bgcolor}, "Title")
));
while (my ($year, $artist, $title) = $sth->fetchrow_array ())
{
    # 调整行颜色变量
    $bgcolor = ($bgcolor eq "silver" ? "white" : "silver");
    push (@rows, Tr (
        td ({-bgcolor => $bgcolor}, escapeHTML ($year)),
        td ({-bgcolor => $bgcolor}, escapeHTML ($artist)),
        td ({-bgcolor => $bgcolor}, escapeHTML ($title))
    ));
}
print table ({-border => "1"}, @rows);
```

注1：如果你使用面向对象的接口CGI.pm，那么就没有任何含混不清的地方了。在这种情形下，你可以通过一个CGI对象调用tr()方法而不必像调用TR()一样：
\$cgi->tr();

在上述表格产生的例子中，列标题直接写入代码中。当然，我们还可以写出一个更通用的函数，这个函数只需要数据库的句柄和任一语句，执行这个语句后，返回的结果就是HTML表格。这个函数能够从语句元数据中自动获得列标记。为生成与表格列名不同的标记，在语句中指定列别名：

```
my $tbl_str = make_table_from_query (
    $dbh,
    "SELECT
        year AS Year, artist AS Artist, title AS Title
    FROM cd
    ORDER BY artist, year"
);
print $tbl_str;
```

任何一个返回结果集的语句，都可以传到这个函数上。例如，你可以使用它用以在从CHECK TABLE语句返回的结果集中构建一个HTML表格，这个语句返回指示check操作的输出的结果集：

```
print p("Result of CHECK TABLE operation:");
my $tbl_str = make_table_from_query ($dbh, "CHECK TABLE profile");
print $tbl_str;
```

那么，make_table_from_query()函数看起来像什么呢？以下是它的一个Perl实现：

```
sub make_table_from_query
{
    # db句柄，查询字符串，被绑定到占位符的参数（如果有）
    my ($dbh, $stmt, @param) = @_;

    my $sth = $dbh->prepare ($stmt);
    $sth->execute (@param);
    my @rows = ();
    # 在头部行的表格中使用列名称
    push (@rows, Tr (th ([ map { escapeHTML ($_) } @{$sth->{NAME}} ])));
    # 获取每个数据行
    while (my $row_ref = $sth->fetchrow_arrayref ())
    {
        # 编码网格值，避免对于未定义值初现警告并对于空的网格使用 
        my @val = map {
            defined ($_) && $_ !~ /\s*\$/ ? escapeHTML ($_) : " " }
            @{$row_ref};
        my $row_str;
        for (my $i = 0; $i < @val; $i++)
        {
            # 右对齐数值列
            if ($sth->{mysql_is_num}->[$i])
            {
                $row_str .= td ({-align => "right"}, $val[$i]);
            }
            else
            {
```

```

        $row_str .= td ($val[$i]);
    }
}
push (@rows, Tr ($row_str));
}
return (table ({-border => "1"}, @rows));
}

```

函数`make_table_from_query()`执行了额外的操作去右校验数字列，从而可以让数值排列得更加美观大方。该函数同时还允许你传数值以限定语句中的占位符，这样做只需在语句字符串后声明它们即可：

```

my $tbl_str = make_table_from_query (
    $dbh,
    "SELECT
        year AS Year, artist AS Artist, title AS Title
    FROM cd
    WHERE year < ?
    ORDER BY artist, year",
    1990
);
print $tbl_str;

```

这里 ` `是生成空表单元的一个技巧。

有时候，包含边框的HTML表格会出现一个显示的问题：当表单元为空或者仅包含空格，浏览器就不会显示该表单元的边框，这时表格看起来就非常的不规则。为了避免这个问题，`make_table_from_query()`函数在空的表单元中一律填充了` `字符，这样表单元的边框就能正常显示。

关于程序生成表格，有一个问题需要特别注意：浏览器只有收到所有表单元的信息后，才会绘出这个表。所以，如果你有一个非常大的结果集，那么它可能会耗费很长的时间才能显示出来。处理这类问题的基本方法，可以是将你的数据划分到多个表中，这多个表可以放在同一个Web页面，也可以是采用多Web页面的方式，每页面显示一部分数据。如果你采用的是单页多表格方法，那么你很可能要在头单元和数据单元中添加一些属性信息。否则，每个表格的宽度和高度将受限于它所包含的内容，所以一页看下来，表格可能会显得杂乱无章。

参考

18.10节讨论了如何用模板来生成表格。

如果想生成这样一个表格：用户用鼠标单击列标题，就能对所在列的表格内容排序，请阅读19.11节。

18.4 将查询结果显示为超链接

Displaying Query Results as Hyperlinks

问题

你希望根据数据库内容生成可点击的超链接。

解决方案

为内容添加合适的标签以生成anchor元素。

讨论

本章前面几节的例子，讨论的都是静态文本，但有时候数据库内容同样能用来生成超链接。如果你将网站的URL或者电子邮件地址存储在数据库表中，那么你很容易就能把这些内容转化为Web页面中的链接。你需要做的就是将信息恰当地编码并添加合适的HTML标签。

假设你手上有一个表格，这个表格包含了公司的名字和网站地址，例如下面所示的book_vendor这个包含了图书销售商和发行商的表：

```
mysql> SELECT * FROM book_vendor ORDER BY name;
+-----+-----+
| name | website |
+-----+-----+
| Amazon.com | www.amazon.com |
| Barnes & Noble | www.bn.com |
| Bookpool | www.bookpool.com |
| O'Reilly Media | www.oreilly.com |
+-----+-----+
```

这个表包含了可以直接生成超文本链接的内容。为了从结果集的每行中生成超链接，我们可以将http://这个协议头直接加到网站地址上，并把结果作为[标签中的href属性值，同时使用name的值作为链接标记。例如：Barnes和Noble行可以像这样改写。](#)

```
<a href="http://www.bn.com">Barnes & Noble</a>
```

如下JSP代码展示了如何从表格文本中生成超链接：

```
<sql:query dataSource="${conn}" var="rs">
    SELECT name, website FROM book_vendor ORDER BY name
</sql:query>

<ul>
<c:forEach items="${rs.rows}" var="row">
    <li>
```

```
<a href=">
    <c:out value="\${row.name}" /></a>
</li>
</c:forEach>
</ul>
```

当在一个Web中被显示时，每个图书销售商的名字就成了活动链接，这个链接被选中后能直接访问该销售商的网站。同样的例子可以用Python写为：

```
stmt = "SELECT name, website FROM book_vendor ORDER BY name"
cursor = conn.cursor ()
cursor.execute (stmt)
items = []
for (name, website) in cursor.fetchall ():
    items.append ("<a href=\"">%s</a>" \
                  % (urllib.quote (website), cgi.escape (name, 1)))
cursor.close ()

# 输出各项，但不对它们进行编码；它们已经被编码过了
print make_unordered_list (items, False)
```

基于CGI.pm的Perl脚本是通过调用a()函数来生成超链接的，如下：

```
a ({-href => "url-value"}, "link label")
```

类似地，生成销售商链接的函数可以这样写：

```
my $stmt = "SELECT name, website FROM book_vendor ORDER BY name";
my $sth = $dbh->prepare ($stmt);
$sth->execute ();
my @items = ();
while (my ($name, $website) = $sth->fetchrow_array ()) {
    push (@items, a ({-href => "http://$website"}, escapeHTML ($name)));
}
print ul (li (\@items));
```

在Ruby中，我们可以使用CGI模块中的方法去生成超链接文本：

```
stmt = "SELECT name, website FROM book_vendor ORDER BY name"
list = ""
dbh.execute(stmt) do |sth|
    sth.fetch do |row|
        list << cgi.li {
            cgi.a("href" => "http://#{row[1]}", CGI.escapeHTML(row[0].to_s))
        }
    end
end
list = cgi.ul { list }
```

根据E-mail地址内容生成超链接文本，是一个常见的Web编程工作。假设你有一个叫newsstaff的表，这个表列出了电视台聘用的每个新闻节目主持人和记者的部门、姓名和邮件地址（如果知道的话），如：

```
mysql> SELECT * FROM newsstaff;
```

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

department	name	email
Sports	Mike Byerson	mbyerson@wrrr-news.com
Sports	Becky Winthrop	bwinthrop@wrrr-news.com
Weather	Bill Hagburg	bhagburg@wrrr-news.com
Local News	Frieda Stevens	NULL
Local Government	Rex Conex	rconex@wrrr-news.com
Current Events	Xavier Ng	xng@wrrr-news.com
Consumer News	Trish White	twhite@wrrr-news.com

从这个表里，你想要做一个在线的目录，这个目录包含了链向所有员工的E-mail地址，这样访问者很容易就能够把E-mail发给每个员工。例如，一个叫Mike Byerson的体育报道人，他的E-mail地址是mbyerson@wrrr-news.com看起来就是这样：

```
Sports: <a href="mailto:mbyerson@wrrr-news.com">Mike Byerson</a>
```

根据表的内容，制作这样一个在线目录是件很容易的事情。首先，我们将编写一个辅助函数，这个函数能处理相应文本并生成E-mail地址的超文本。后续的多个脚本，都将复用这个函数。在Perl中，这个函数是这样的：

```
sub make_email_link
{
my ($name, $addr) = @_;

$name = escapeHTML ($name);
# 如果地址是undef或为空，作为静态文本返回名称
return $name if !defined ($addr) || $addr eq "";
# 否则返回一个超链接
return a ({-href => "mailto:$addr"}, $name);
}
```

如果正好遇到一个没有E-mail信息的人，那么这个函数仅返回他姓名的静态文本。要使用这个函数，我们还需要编写一个循环并把姓名和E-mail地址作为输入参数，然后在每个人所在单位信息前显示他的E-mail地址：

```
my $stmt = "SELECT department, name, email FROM newsstaff
           ORDER BY department, name";
my $sth = $dbh->prepare ($stmt);
$sth->execute ();
my @items = ();
while (my ($dept, $name, $email) = $sth->fetchrow_array ())
{
    push (@items,
          escapeHTML ($dept) . ":" . make_email_link ($name, $email));
}
print ul (li (\@items));
```

该辅助函数，在Ruby、PHP和Python中为：

```
- def make_email_link(name, addr = nil)
  name = CGI.escapeHTML(name.to_s)
  # 如果地址是nil或为空，作为静态文本返回名称
  return name if addr.nil? or addr == ""
  # 否则返回一个超链接
```

```

        return "<a href=\"mailto:#{$addr}\">#${name}</a>"  
end  
  
function make_email_link ($name, $addr = NULL)  
{  
    $name = htmlspecialchars ($name);  
    # 如果地址是NULL或为空, 作为静态文本返回名称  
    if ($addr === NULL || $addr == "")  
        return ($name);  
    # 否则返回一个超链接  
    return (sprintf ("<a href=\"mailto:%s\">%s</a>", $addr, $name));  
}  
  
def make_email_link (name, addr = None):  
    name = cgi.escape (name, 1)  
    #如果地址是None或为空, 作为静态文本返回名称  
    return name  
    #否则返回一个超链接  
    return "<a href=\"mailto:%s\">%s</a>" % (addr, name)

```

在一个JSP页面中, 你可以生成如下newsstaff的列表:

```

<sql:query dataSource="${conn}" var="rs">  
    SELECT department, name, email  
    FROM newsstaff  
    ORDER BY department, name  
</sql:query>  
  
<ul>  
<c:forEach items="${rs.rows}" var="row">  
    <li>  
        <c:out value="${row.department}"/>:  
        <c:set var="name" value="${row.name}"/>  
        <c:set var="email" value="${row.email}"/>  
        <c:choose>  
            <%-- null or empty value test --%>  
            <c:when test="${empty email}">  
                <c:out value="${name}"/>  
            </c:when>  
            <c:otherwise>  
                <a href="mailto:<c:out value="${email}" />"><c:out value="${name}"/></a>  
            </c:otherwise>  
        </c:choose>  
    </li>  
</c:forEach>  
</ul>

```

18.5 根据数据库内容创建导航索引

Creating a Navigation Index from Database Content

问题

在Web页面中，数据项列表往往很长。你希望用户能够很轻松地浏览该Web页面。

解决方案

创建一个导航索引，其中包含多个链接，能够到达列表的不同部分。

讨论

在Web页面中显示列表是比较容易的（参见18.2节）。但是，如果一格列表包含了太多的数据项，那么该Web页面将会变得非常长。在这种情况下，通常需要将该列表拆分成多个块，并提供一个链接到这些块的索引项，看起来像个超文本链接，用户不需要滚动浏览该页面便能快速查看列表的任一部分。例如，如果你从一个表中获取所有的行，并把这些行以块组织起来，那么你就能包含一个索引使得用户能轻易地在任一块间跳转阅读。同样的思想，也适用于多页浏览：通过在每个Web页面中提供导航索引，用户能够轻松地跳转到制定的页面。

在光盘中我们提供了两个例子以介绍这种技术，这两个例子都基于5.15节中介绍的kjv表。这两个例子分别提供一种显示方法，用到了kjv表中存储的Esther书中的诗句：

- 一个单Web页面的显示，列出了Esther所有章节中的所有诗篇。这个列表被拆分成10个块（每块对应Esther的一章），导航索引链接到每一块的开始部分。
- 一个多Web页面的显示，每个Web页面分别显示Esther的一章，有一个主页面用以让用户选择浏览任一章的内容。同时在每一个Web页面中，都有链接到其他页面的导航索引。

构建一个单Web页面的导航索引

这个例子在一个Web页面中显示了Esther中所有的诗句。根据所属的章节，这些诗句被划分到不同的块中。为了显示页面，每个块都包含一个导航标记，我们在每个章节前放置了一个的anchor元素：

```
<a name="1">Chapter 1</a>
... list of verses in chapter 1...
<a name="2">Chapter 2</a>
... list of verses in chapter 2...
<a name="3">Chapter 3</a>
... list of verses in chapter 3...
...
```

这里生成一个列表，它包含一组标记1、2、3等。要构建导航索引，就需要构建一组超链接，每个超链接指向一个name标记：

```
<a href="#1">Chapter 1</a>
<a href="#2">Chapter 2</a>
<a href="#3">Chapter 3</a>
...

```

每个href属性中的#表示了指向同一页中指定位置的链接。比如，`href="#3"`指向属性为`name="3"`的anchor元素。

为了执行这类导航索引，你需要使用以下方法：

- 获取诗句行并存入内存，并决定哪个入口将用到导航索引中，接下来输出索引和诗句。
- 预先指出所有可用的anchor，并首先构建索引。在这里，章节的编号可由以下语句决定：

```
SELECT DISTINCT cnum FROM kjv WHERE bname = 'Esther' ORDER BY cnum;
```

你可以使用查询结果集来构建导航索引，然后获取每章对应诗句，并构建Web块让索引能够指向这些Web块。

这里我们提供一个脚本esther1.pl，这个脚本采用的是第一个方法。它是18.2节中嵌套列表的一种扩展。

```
#!/usr/bin/perl
# esther1.pl - 在一页中显示Esther的书，带有导航索引

use strict;
use warnings;
use CGI qw(:standard escape escapeHTML);
use Cookbook;

my $title = "The Book of Esther";

my $page = header ()
    . start_html (-title => $title, -bgcolor => "white")
    . h3 ($title);

my $dbh = Cookbook::connect ();

# 查询Esther书中的诗句并将其每个与其所属章的诗句列表关联起来

my $sth = $dbh->prepare ("SELECT cnum, vnum, vtext FROM kjv
                           WHERE bname = 'Esther'
                           ORDER BY cnum, vnum");
$sth->execute ();
my %verses = ();
while (my ($cnum, $vnum, $vtext) = $sth->fetchrow_array ())
{

```

```

# 如果这是该章中的第一节，将该章的诗句列表初始化为空数组，然后添加诗句号/文本到数
# 组中。
$verses{$cnum} = [] unless exists ($verses{$cnum});
push (@{$verses{$cnum}}, p (escapeHTML ("$vnum. $vtext")));
}

# 确定所有章的编号并用它们来构建导航索引。这些链接是以<a href="#num>Chapter
# num</a>的形式，这里num是章的编号，而'#'表示页内部的链接。这儿不需要URL或HTML
# 编码（这儿显示的文本不要它）。确定按数值来排序章编号（使用{a<=>b}）。用空格来分隔链
# 接。
my $nav_index;
foreach my $cnum (sort { $a <=> $b } keys (%verses))
{
    $nav_index .= "&nbsp;" if $nav_index;
    $nav_index .= a ({-href => "#$cnum"}, "Chapter $cnum");
}
}

# 现在显示每章的诗句列表。在每节之前加上一个标签，这个标签显示章编号和导航索引的一个
# 复制。
foreach my $cnum (sort { $a <=> $b } keys (%verses))
{
    # 为这节的状态显示添加一个<a name>锚
    $page .= p (a ({-name => $cnum}, font ({-size => "+2"}, "Chapter $cnum"))
        . br ()
        . $nav_index);
    $page .= join ("", @{$verses{$cnum}}); # 为该章添加诗句数组
}

$dbh->disconnect ();
$page .= end_html ();
print $page;

```

构建一个多Web页面的导航索引

本例采用了一个Perl脚本esther2.pl，这个脚本根据Esther的内容能够生成任一Web页面。首页面显示的是Esther中所有章节的列表，根据这个列表，用户能选择一个章节。这个列表中的每一章节都是一个超链接，通过它再调用脚本来显示相应章节中的诗句列表。由于这个脚本是用来构建多Web页面索引的，因此它必须能够判断每次运行它所要显示的页面。为了达到这个目的，脚本检查它的URL以获取章节参数来指明要显示章节的数目。如果没有章节参数或者章节参数的值不是整数，脚本就显示首页面。

主页面对应的URL如下：

<http://localhost/cgi-bin/esther2.pl>

链接向每一章节的URL采用了如下形式（其中cnum是章节号）：

http://localhost/cgi-bin/esther2.pl?chapter=cnum

脚本esther2.pl使用了CGI.pm中的函数param()，以获取章节参数的值：

```
my $cnum = param ("chapter");
if (!defined ($cnum) || $cnum !~ /^ \d+$/)
{
    # 没有章编号或者它有所缺失
}
else
{
    # 章编号存在
}
```

如果URL中没有章节参数，那么\$cnum就是undef。否则，&cnum就是一个章节号，我们要检查确认其是一个整数值。（这可避免一个垃圾值被想要破坏脚本的人指定。）

如下是整个esther2.pl的脚本：

```
#!/usr/bin/perl
# esther2.pl - 多页显示Esther的书，每章一页，具有导航索引

use strict;
use warnings;
use CGI qw(:standard escape escapeHTML);
use Cookbook;

# 构建导航索引，作为一个链接列表，这些链接指向Esther书中的每章的页。标签是"Chapter
# n"的形式；章编号作为chapter=num参数合并入链接。

# $dbh是数据库句柄，$cnum是信息正被显示的章的编号。对应此编号的章列表中的标签以静
# 态文本显示；其他显示成指向其他章页面的超链接。传递0使所有的条目都是超链接（正确的
# 章编号不为0）。

# 不需要编码，因为章编号是数字，不需要它

sub get_chapter_list
{
my ($dbh, $cnum) = @_;
my $nav_index;
my $ref = $dbh->selectcol_arrayref (
```

```

        "SELECT DISTINCT cnum FROM kjv
        WHERE bname = 'Esther' ORDER BY cnum"
    );
foreach my $cur_cnum (@{$ref})
{
    my $link = url () . "?chapter=$cur_cnum";
    my $label = "Chapter $cur_cnum";
    $nav_index .= br () if $nav_index;      # 用<br>来分隔条目
    # 如果条目是当前章的使用静态粗体文本，否则使用一个超链接
    $nav_index .= ($cur_cnum == $cnum
        ? strong ($label)
        : a ({-href => $link}, $label));
}
return ($nav_index);
}

# 获得给定章的诗句列表。如果没有，章编号是不合法的，但要小心处理此情况。
sub get_verses
{
my ($dbh, $cnum) = @_;
my $ref = $dbh->selectall_arrayref (
    "SELECT vnum, vtext FROM kjv
     WHERE bname = 'Esther' AND cnum = ?",
    undef, $cnum);
my $verses = "";
foreach my $row_ref (@{$ref})
{
    $verses .= p (escapeHTML ("$row_ref->[0]. $row_ref->[1]"));
}
return ($verses eq ""      # 没有诗句?
        ? p ("No verses in chapter $cnum were found.")
        : p ("Chapter $cnum:") . $verses);
}

# -----
my $title = "The Book of Esther";
my $page = header () . start_html (-title => $title, -bgcolor => "white");
my ($left_panel, $right_panel);
my $dbh = Cookbook::connect ();
my $cnum = param ("chapter");
if (!defined ($cnum) || $cnum !~ /^\\d+$/)
{
    # 没有或不全的章；显示主页面，左面的面板用超链接列出所有的章，右面的面板提供指示。
    $left_panel = get_chapter_list ($dbh, 0);
    $right_panel = p (strong ($title))
}

```

```

        . p ("Select a chapter from the list at left.");
    }
else
{
    # 给定章编号; 左面的面板列出所有章
    # 章以超链接的方式存在 (除当前章是粗体文本外)
    # 右面的面板列出当前章的诗句
    $left_panel = get_chapter_list ($dbh, $cnum);
    $right_panel = p (strong ($title))
        . get_verses ($dbh, $cnum);
}
$dbh->disconnect ();

# 将页面划分为一行, 三列的表 (中间的表网格是一个空格)

$page .= table (Tr (
    td ({-valign => "top", -width => "15%"}, $left_panel),
    td ({-valign => "top", -width => "5%"}, "&nbsp;"),
    td ({-valign => "top", -width => "80%"}, $right_panel)
));

$page .= end_html ();
print $page;

```

参考

脚本esther2.pl通过param()检查它的执行环境，19.5节进一步讨论了Web脚本参数处理。

19.10节讨论了另外一种导航问题：如何将一个结果集拆分到多Web页面中，并提供向前翻页和向后翻页两个导航索引。

18.6 存储图片或其他二进制数据

Storing Images or Other Binary Data

问题

你希望在MySQL数据库中存储图片。

解决方案

这并不困难，只要你遵循正确的预防措施以编码图片数据。

讨论

网站并不仅限于显示文本。它同时还能提供多种方式的二进制数据。例如图片、音乐和PDF文件等。图片是一种常见的二进制数据，且由于图片存储是数据库的一种非常自然的应用，

那么我们常常需要回答诸如：“如何在MySQL中存储数据？”许多人就是这么回答的：“别这么做！”。在“你应不应该在你的数据库中存储图片？”的话题中，讨论了其中的一些原因。了解如何操作二进制数据是非常重要的，所以本节介绍如何在MySQL中存储图片。不过，也许这并不是最好的方式，所以本节同时还会介绍图片是如何通过文件系统存储的。

尽管此处的讨论仅限于图片，但其中的原则适用于任意类型的二进制数据，如PDF文件或者压缩的文本。事实上，这些原则还用于普通文本。尽管人们常把图片当成特殊的东西，其实它们不是。

图片存储之所以比其他像文本或数字类型的数据存储更让人感到困惑的一个原因是：我们无法像输入普通文本一样手工输入图片数据。例如，你能够轻松地使用MySQL以执行一条INSERT语句，向表格中插入诸如3.48这样的数字，或者像“Je voudrais une bicyclette rouge”这样的字符串，但图片中包含的是二进制数据，这是不能用值来引用它们的。所以你需要其他的方法，比如，你可以：

- 使用LOAD_FILE()函数。
- 写一个程序，该程序读入图片文件并为你自动构建合适的INSERT语句。

你应该在数据库中存储图片吗？

- 在何处存储图片往往是很考虑的一个折中。无论你是在数据库中，还是在文件系统中存储图片，都有它的优点和不足：
- 在数据库中存储图片会撑胀一个表。如果涉及到大量的图片，那么你很可能会达到操作系统给你的数据库表格所设置的上限。另一方面，如果你把图片存储在文件系统中，那么目录查看将变得缓慢。为避免这个问题，你可能需要运行某些层次化的存储或使用能够提供高效目录查看的文件系统（如在Linux上的Reiser）。
- 使用一个数据库集中管理存储图片，这些图片由不同主机上多个Web服务器来使用。存储在文件系统中的图片，一般都放在web服务器主机的本地文件中。但遇到多主机的情况时，往往意味着你需要把这些图片文件在所有主机上都复制一份。如果你在MySQL中存储图片，只需要一份图片拷贝即可，因为每个web服务器都可以从同一个数据库服务器中获取图片。
- 当图片存储在文件系统中时，他们本质上建立了一种外键的关系。图片操作涉及到两个步骤：一个在数据库中，另一个在文件系统中。这反过来说明，如果你要求事务行为，那么将很难实现——你不仅要有两步操作，而且分别属于不同领域。将图片信息存储在数据库中相对简单，这是因为图片的插入、更新、删除仅仅需要一个行操作。因此完全没有必要在关系数据库和文件系统之间同步。
- 从文件系统中读取图片一般比从数据库中要快，这是因为Web服务器本身也需要打

——待续

开文件，读取其中的内容，并把它写到客户端去。而在数据库中的图片必须被读取和写入两次。首先，MySQL服务器从数据库中读取图片，然后把它写入你的脚本中，然后这个脚本读取图片，并把它写到客户端。

- 存储在文件系统中的图片能直接通过Web页面访问，比如标签就提供了这个功能。而存储在MySQL中的图片必须经由一个脚本来读取并发送到客户端。然而，即使存储在文件系统中的图片可由Web服务器直接访问，你还是有可能通过脚本来操作这个图片。比如你需要计算每个图片的访问次数，或者你在用户请求时再根据上下文来选择图片。
- 如果你把图片存储在数据库中，那么你需要使用BLOB数据类型。这是一种可变长的类型，所以对应的表会拥有各种长度的行。对于MySQL存储引擎而言，操作固定长度的行通常效率会比较高，所以你可以将图片存储在文件系统中以获取较高的表检查速度，同时对image表中的列使用固定长度类型。

使用LOAD_FILE()函数存储图片

函数LOAD_FILE()采用一个参数用以标明需要读出或写入数据库中的文件。例如，一个图片存储在/tmp/myimage.png，它可以这样导入数据库中：

```
INSERT INTO mytbl (image_data) VALUES(LOAD_FILE('/tmp/myimage.png'));
```

使用LOAD_FILE()函数把图片文件导入数据库中，需要满足以下条件：

- 图片必须放在MySQL服务器运行的主机上。
- 服务器必须能够访问该文件。
- 你必须拥有FILE权限。

这三个限制意味着LOAD_FILE()函数只能由部分MySQL用户使用。

使用脚本存储图片

如果LOAD_FILE()函数不可用，或者你不打算使用它，那么你可以写一小段程序来导入你的图片。这个程序应该既能读一个图片文件的内容，又能创建一个包含该内容的行，或者创建一行，同时该行能指明在文件系统的哪个位置存放着相关的文件。如果你选择在MySQL中存储图片，那么就像其他类型的数据一样在行创建语句中包含图片数据。也就是说，你要么使用一个占位符并把文件数据绑定到其上，要么将数据编码并直接放入语句中执行。

在光盘中的脚本 `store_image.pl`，可以通过命令行运行，它存储一个图片文件。这个脚本并不是用来证明采用文件系统好，还是采用数据库好。它只是用来展示如何使用这两种方法。当然，这需要双倍的空间。为了能让这个脚本为你所用，你只需要保留适用于你想实现的任一存储方法。在本节最后讨论了一些必要的修改。

脚本 `store_image.pl` 使用一个图片表，这个表包含下列图片的ID、名字、MIME类型，以及用来存储图片的列：

```
CREATE TABLE image
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT, # 图片ID编号
    name    VARCHAR(30) NOT NULL,                  # 图片名称
    type    VARCHAR(20) NOT NULL,                  # 图片MIME类型
    data    MEDIUMBLOB NOT NULL,                  # 图片数据
    PRIMARY KEY (id),                           # id和name是唯一的
    UNIQUE (name)
);
```

`name`列指示了存储在文件系统中的图片文件的名字。`Data`列是 `MEDIUMBLOB` 类型，它非常适合小于16MB的图片。如果你需要更大的存储空间，请使用 `LONGBLOB` 类型。

使用 `name` 列来存储图片的完全路径是没问题的，但如果你已经把图片文件都放在一个目录下了，那么你只需要存储相对于该路径的名字便可，这样还可以节省很多空间。脚本 `store_image.pl` 就是这么做的。它需要知道图片所在目录的路径，这就是 `$image_dir` 变量的用途。你应该检查这个变量的值，并在运行脚本前根据需要修改它。默认值是倾向存储图片的地方，但是你要根据自己的参数选择来改变它。请确认如果在运行脚本前，这个目录不存在，先创建它，并设置好它的访问权限，这样 web 服务器可以读写那里的文件。你还需要检查 `display_image.pl` 脚本中的图片目录路径名，这个脚本在本章的后面将会介绍。



提示： 图片存储目录应该在 Web 服务器文件目录的外面。否则，一个用户将有可能知道或猜测出这个目录的位置并上传可执行文件，并通过 HTTP 请求来执行这个文件。

脚本 `store_image.pl` 如下：

```
#!/usr/bin/perl
# store_image.pl - 读取一个图片文件，存入image表和文件系统中。(通常，你应该将图
# 片仅存在这两者中的一个中；这个脚本阐述了如何做到这两个。)
```

```

use strict;
use warnings;
use Fcntl;      # for O_RDONLY, O_WRONLY, O_CREAT
use FileHandle;
use Cookbook;

# 默认的图片存储目录和路径名分隔符（可根据需要进行改变）
# 位置不应该在web服务器的文档树下
my $image_dir = "/usr/local/lib/mcb/images";
my $path_sep = "/";

# 对于Windows/DOS重设目录和路径名分隔符
if ($^O =~ /MSWin/i || $^O =~ /dos/)
{
    $image_dir = "C:\\mcb\\images";
    $path_sep = "\\";
}

-d $image_dir or die "$0: image directory ($image_dir) does not exist\n";

# 如果脚本不合理调用输出帮助信息

(@ARGV == 2 || @ARGV == 3) or die <<USAGE_MESSAGE;
Usage: $0 image_file mime_type [image_name]

image_file = name of the image file to store
mime_type = the image MIME type (e.g., image/jpeg or image/png)
image_name = alternate name to give the image

image_name is optional; if not specified, the default is the
image file basename.
USAGE_MESSAGE

my $file_name = shift (@ARGV);      # 图片文件名
my $mime_type = shift (@ARGV);      # 图片MIME类型
my $image_name = shift (@ARGV);      # 图片名称(可选)

# 如果未指定图片名称，使用文件名的基本名称（允许使用\或/作为分隔符）
($image_name = $file_name) =~ s|.*[\\/]| unless defined $image_name;

my $fh = new FileHandle;
my ($size, $data);

sysopen ($fh, $file_name, O_RDONLY)
or die "Cannot read $file_name: $!\n";
binmode ($fh);      # 对二进制数据非常有用
$size = (stat ($fh))[7];
sysread ($fh, $data, $size) == $size
or die "Failed to read entire file $file_name: $!\n";
$fh->close ();

# 将图片文件存储到文件系统中的$image_dir下。（如果存在旧版本覆盖该文件。）

```

```

my $image_path = $image_dir . $path_sep . $image_name;

sysopen ($fh, $image_path, O_WRONLY|O_CREAT)
    or die "Cannot open $image_path: $!\n";
binmode ($fh);    # 对于二进制数据有用
syswrite ($fh, $data, $size) == $size
    or die "Failed to write entire image file $image_path: $!\n";
$fh->close ();

# 将图片保存到数据库表中。(使用REPLACE来剔除同名的旧图片。)

my $dbh = Cookbook::connect ();
$dbh->do ("REPLACE INTO image (name,type,data) VALUES(?, ?, ?)",
           undef,
           $image_name, $mime_type, $data);
$dbh->disconnect ();

```

如果你不带参数调用该脚本，它将打印出帮助信息。否则，它需要两个参数，一个用来声明文件的路径，另一个用来说明文件的MIME类型。默认情况下，文件的basename（最后组件）也将用作数据库表和图片目录中存储的图片名。如果你想使用其他的名字，请提供第三个参数。

这个脚本看起来是非常直接的，它顺序执行以下操作：

1. 检查是否有合适数量的参数，并根据参数初始化变量。
2. 确定文件的目录是存在的，如果没有，则退出。
3. 打开图片文件，读入其中的内容。
4. 将图片作为文件存储在图片目录中。
5. 在image表中存储包含标识信息以及图片数据的行。

脚本*store_image.pl*使用REPLACE而不是INSERT，这样你就能使用同名的新文件将旧的文件替换。语句没有指明图片的ID信息；id是一个AUTO_INCREMENT列。注意，如果你把同名的文件替换掉了，REPLACE将使用新生成的id值。如果你想保留旧的id值，那你应该使用INSERT ... ON DUPLICATE KEY UPDATE语句（11.14节）。如果名字并不存在，这条语句会插入一行，如果存在，则会更新图片值。

将图片信息存储在MySQL中的REPLACE语句：

```

$dbh->do ("REPLACE INTO image (name,type,data) VALUES(?, ?, ?)",
           undef,
           $image_name, $mime_type, $data);

```

如果你仔细检查这个语句，并寻找操作二进制数据的特殊技巧，你会失望的。因为包含图片的\$data变量是无法被特殊对待的。这个语句指的是所有列都统一使用了?占位符，且对应的数值将由do()函数传送过来。还有另一种方法，就是显式地在列值上执行escape处理，并把它们直接写入语句中：

```
$image_name = $dbh->quote ($image_name);
$mime_type = $dbh->quote ($mime_type);
$data = $dbh->quote ($data);
$dbh->do ("REPLACE INTO image (name,type,data)
VALUES ($image_name,$mime_type,$data)");
```

许多人认为图片处理是非常麻烦的。如果你在语句中正确地使用了占位符处理图片数据或对其进行编码，那就不会有错误。否则你会经常犯错误。其实这是很简单的原则。跟你处理其他任一类型的数据一样，即使是文本。毕竟，如果你在语句中插入一段文本，而它恰恰包含引号等诸如此类的特殊字符，那么这个语句肯定执行不了。所以，采用占位符或进行编码并不是针对图片的一种特殊的手段——对所有数据都是必要的。跟我一起说：“我想一直使用占位符或者编码我的列数据值。一直，一直，一直，一直。”（已经说过，我觉得有必要指出如果你足够了解给定的值——比如，如果你绝对确信它是一个整数——有很多机会你可以不必破坏这条规则。然而，遵循这条规则永远不会错。）

如果想尝试运行这段脚本，请到光盘中的apache/images目录。这个目录下包含了store_image.pl脚本，以及在其flags子目录下的一些示例图片（它们是一些国家的国旗图片）。为了存储这些图片中的某一张，可以在Unix中运行如下脚本：

```
% ./store_image.pl flags/iceland.jpg image/jpeg
```

在Windows下，是这样的：

```
C:\> store_image.pl flags\iceland.jpg image/jpeg
```

脚本store_image.pl处理图片存储，下一节我将讨论如何读取图片并将它显示在Web上。那么，删除图片的操作呢？我把删除你不再需要的图片的操作交给你来编写。如果你正在文件系统中存储图片，请千万记得从数据库中删除响应的行以及该行指向的图片文件。

脚本store_image.pl将图片同时存储在数据库和文件系统中，这毫无疑问是低效的。我们在早先提到过，如果你在自己的程序中使用这个脚本，请记得修改它通过仅有的一种存储方法。建议您这样修改：

- 使用该脚本仅在MySQL数据库中存储图片，不必创建一个图片目录，那么你需要检查目录的存在与否，并把往该目录中写入图片的代码删掉。
- 仅在文件系统中使用该脚本存储图片，去掉表中的data列，并修改REPLACE语句，使它不用总指向该列。

上述修改建议，同样适用于18.7节中的display_image.pl脚本。

参考

18.7节介绍如何读取并在Web上显示图片。19.8节讨论的是如何上传一个图片，并把它存入MySQL。

18.7 检索图片或其他二进制数据

Retrieving Images or Other Binary Data

问题

你能使用18.6节中讨论的技术在数据库中存储图片或其他二进制文件，但如何把它们读取出来呢？

解决方案

你只需要执行SELECT语句。当然，得到信息后如何处理就不是本节关心的事情了。

讨论

像在18.6节中介绍的那样，通过在语句中手工向数据库添加图片数据是非常困难的，一般你使用LOAD_FILE()函数或者编写脚本来实现图像数据的存储。然而，在读取图片数据时并不存在这样的困难：

```
SELECT * FROM image WHERE id = 1;
```

但二进制信息一般都无法在文本显示终端上显示，所以你一般不会在命令行方式下调用上述语句。如果你不幸这么做了，那么后果就是你的显示器上布满了乱七八糟的字符。常见的情况是，图像信息显示在Web页面上，或者是将其发送到客户端供下载，对于像PDF文件的非图片二进制数据，这种情况更常见。（18.9节讨论了如何下载。）

在Web页面中显示图像，需要使用标签以告诉客户端的浏览器到哪儿去获取图片。如果你已经用文件的方式在一个web服务器访问的目录下存好这个图片了，那么你只能直接

引用该图片。例如，如果文件iceland.jpg位于/usr/local/lib/mcb/images目录中，你需要这样：

```

```

如果你采用的是上述方法，请确保每个文件名都有合适的后缀（如.gif或.png），这将帮助Web服务器确定当其向客户端发送文件时产生的HTTP应答中的Content-Type头部类型。

现在，假设图像数据存储在数据库的表中，或在一个Web服务器不能访问的位置，那么标签需要指向一个脚本，由这个脚本来把图像传送给客户端。要做到这一点，脚本向客户端发送响应，该响应应该是通过发送一个指示图片格式的Content-Type头部、一个指示图片数据字节数的Content-Length头部来响应一个空白行及最后作为响应体的图片。

接下来的脚本display_image.pl，示范了如何在Web上提供图片数据。它需要名字参数以标明要显示哪一个图片，并允许可选的位置信息来表明是从文件系统还是数据库中获取图像。默认情况下，我们从image表中获取图像数据。例如，以下的URL表明待显示的图像分别来自数据库和文件系统：

```
http://localhost/cgi-bin/display_image.pl?name=iceland.jpg  
http://localhost/cgi-bin/display_image.pl?name=iceland.jpg;location=fs
```

脚本是这样的：

```
#!/usr/bin/perl  
# display_image.pl - 在Web上显示图片  
  
use strict;  
use warnings;  
use CGI qw(:standard escapeHTML);  
use FileHandle;  
use Cookbook;  
  
sub error  
{  
    my $msg = escapeHTML ($_[0]);  
  
    print header (), start_html ("Error"), p ($msg), end_html ();  
    exit (0);  
}  
  
# -----  
  
# 默认的图片存储目录和路径名分隔符 (根据需要改变)  
my $image_dir = "/usr/local/lib/mcb/images";  
# 位置不应该在web服务器的文档树中  
my $path_sep = "/";  
  
# 对于Windows/DOS重设目录和路径名分隔符
```

```

if ($^O =~ /MSWin/i || $^O =~ /dos/)
{
    $image_dir = "C:\\mcb\\images";
    $path_sep = "\\";
}

my $name = param ("name");
my $location = param ("location");

# 确定指定了图片名
defined ($name) or error ("image name is missing");
# 如果位置没有指定或者不是"db"或"fs", 使用默认值"db"
($defined ($location) && $location eq "fs") or $location = "db";

my $dbh = Cookbook::connect ();

my ($type, $data);

# 如果位置是"db", 从image表获取图片数据和MIME类型。
# 如果位置是"fs", 从image表获取MIME类型, 从文件系统读取图片数据。

if ($location eq "db")
{
    ($type, $data) = $dbh->selectrow_array (
        "SELECT type, data FROM image WHERE name = ?",
        undef,
        $name)
    or error ("Cannot find image with name $name");
}
else
{
    $type = $dbh->selectrow_array (
        "SELECT type FROM image WHERE name = ?",
        undef,
        $name)
    or error ("Cannot find image with name $name");
    my $fh = new FileHandle;
    my $image_path = $image_dir . $path_sep . $name;
    open ($fh, $image_path)
    or error ("Cannot read $image_path: $!");
    binmode ($fh); # helpful for binary data
    my $size = (stat ($fh))[7];
    read ($fh, $data, $size) == $size
    or error ("Failed to read entire file $image_path: $!");
    $fh->close ();
}

$dbh->disconnect ();

# 将图片发送到客户端, 前跟Content-Type: 和Content-Length: 头部。

```

```
print header (-type => $type, -Content_Length => length ($data));
print $data;
```

18.8 提供标语广告

Serving Banner Ads

问题

你希望能从一组图片中，随机选出一张并显示在标语广告中。

解决方案

用一个脚本从图像表中随机选择一行，然后把图像发送到客户端。

讨论

18.7节中的脚本display_image.pl假设URL中包含一个参数，该参数给出了需发送到客户端的图像文件名称。另一个应用需要决定选择哪张图片来显示。与图像相关的一个比较流行的Web编程是支持Web页面上的标语广告。实现标语广告的一个简单方法，是编写一段脚本，该脚本在每次被调用时随机选择一张图片。如下的Python脚本banner.py正式用于这个目的，演示了如何实现该目标，其中“ads”是image表中的图片标记：

```
#!/usr/bin/python
# 从image表中随机选取banner广告（如果没有图片不发送响应）

import MySQLdb
import Cookbook

conn = Cookbook.connect ()

stmt = "SELECT type, data FROM image ORDER BY RAND() LIMIT 1"
cursor = conn.cursor ()
cursor.execute (stmt)
row = cursor.fetchone ()
cursor.close ()
if row is not None:
    (type, data) = row
    # 发送图片到客户端，前面是Content-Type:和Content-Length: 头部。
    # Expires:,Cache-Control:, 和Pragma:头部帮助避免浏览器缓存图片，并在后继请求中重用它。
    print "Content-Type: %s" % type
    print "Content-Length: %s" % len (data)
    print "Expires: Sat, 01 Jan 2000 00:00:00 GMT"
    print "Cache-Control: no-cache"
    print "Pragma: no-cache"
    print ""
```

```
print data  
conn.close ()
```

脚本banner.py发送一些头给客户端，同时还有常见的Content-Type和Content-Length头部。这些额外的头部信息有助于客户端浏览器缓存图片。标签Expire规定了一个日期，并告诉浏览器该图片在何时会过期。标签Cache-Control和Pragma告诉浏览器不用缓存该图片。上述脚本同时发送这两个标签，这是因为有部分浏览器只能识别其中之一。

为什么禁止缓存呢？因为如果你不这么做，浏览器会在第一次看到banner.py脚本时发送一个请求。在后续的对该脚本的请求中，浏览器将重用这个图片，这将导致每次浏览该页面时看到的总是相同的图片。

将banner.py脚本安装到你的cgi-bin目录下。然后，在Web页面中放置一个标语，使用标签来调用这个脚本。例如，如果这个脚本被放置在/cgi-bin/banner.py，那么下面的页面将这么使用它：

```
<!-- bannertest1.html - 具有指向旗帜广告脚本的单链接的页面 -->  
<html>  
<head>  
<title>Banner Ad Test Page 1</title>  
</head>  
<body bgcolor="white">  
  
<p>You should see an image below this paragraph.</p>  
  
  
  
</body>  
</html>
```

如果你请求这个页面，它会显示一张图片，当你不停刷新该页面时，你将看到一系列随机的图片。如果你修改banner.py脚本，让它不发送与缓存相关的HTML标签，那么即使你刷新该页面多次，你也很可能将看到相同的图片。

缓存控制头封存了后续所有页面请求过程中发生的连接到banner.py的缓存。另外，有一种复杂的情况是一个页面中有多个指向banner.py的链接：

```
<!-- bannertest2.html - 具有多个指向旗帜广告脚本的链接的页面-->  
<html>  
<head>  
<title>Banner Ad Test Page 2</title>  
</head>  
<body bgcolor="white">
```

```
<p>You should see two images below this paragraph,  
and they probably will be the same.</p>  
  
  
  
  
<p>You should see two images below this paragraph,  
and they probably will be different.</p>  
  
  
  
  
</body>  
</html>
```

第一组指向banner.py的链接是完全一样的，你可能发现当你请求这个页面时，你的浏览器将发现这个情况，并只发送一个请求，然后使用在页面中两个链接都返回的图片。这一组显示的图片总是完全一样的。第二组指向banner.py的链接解决了这个问题。这两个链接在URL末尾处，包含了额外的内容，这导致它们的图片是不同的。尽管banner.py是一样的，当时不同的URL链接会骗过浏览器，让它发送两个请求，从而分别获取一个随机的图片，因此最后显示的往往是两张不同的图片。这两个链接在URL末尾处，包含了额外的内容，这导致它们的图片是不同的。

18.9 提供可下载的查询结果

Serving Query Results for Download

问题

你希望把数据库中的信息发送给客户端，让客户端下载，而不是显示。

解决方案

很遗憾，并没有好的办法强制客户端下载某个内容。浏览器会根据HTTP的Content-Type头部值来处理发给它的信息，如果浏览器有该值的一个处理器，它会据此处理信息。然而，你还是可以欺骗浏览器，发送一个“generic”的内容类型，浏览器或许没有该值的一个处理器。

讨论

本章前面几节讨论了如何将数据库查询的结果集成到Web页面中，并把结果集显示为段落、列表、表格、或者图片。但如何生成一个让用户下载到文件中的结果集呢？产生响应本身并不困难：在信息前发送一个Content-Type头部，比如针对普通文本的text/plain，是针

对JPEG图片的image/jpg，针对PDF或Excel文档的application/pdf或application/msexcel。然后发送一个空行和查询的结果。可问题是，没有办法能强制浏览器下载这些信息。如果它从类型中知道该怎么做，那么它会尝试去在这些信息上执行相关操作。打个比方，如果它知道如何显示文本和图片，那么它就会执行相关的操作。如果它认为待处理的对象是PDF或Excel文档，它会打开一个PDF或Excel视窗。绝大多数浏览器允许用户显式地决定是否下载（比如右键单击超链接，并从菜单中选择另存为），但那也只是客户端的机制。在服务器端，你没有访问它的方法。

我们唯一能做的就是通过伪造内容类型来欺骗浏览器。一种最常见的类型是application/octet-stream。绝大多数用户不会指定处理这种类型的程序，所以如果你使用这种类型发送一个应答，这或许会触发浏览器的一个下载。这种做法的不好之处在于，应答中包含一个关于它所含有的信息的错误类型。通过提供一个在浏览器保存文件时供其使用的默认文件名能缓解这个问题。如果文件名有一个指示文件类型的后缀名，比如.txt,.jpg,.pdf,或.xls，将有助于客户端（或者客户端主机上的操作系统）决定如何处理这个文件。为提供一个名字，要在响应中包含Content-Disposition头部：

```
Content-disposition: attachment; filename="suggested_name"
```

下面的PHP脚本download.php，说明了生成可下载内容的一种方法。但第一次被调用时，它生成一个包含一个链接的页面，可选择该链接发起一个下载。这个链接指向download.php但包含一个下载的参数。当你选中这个链接时，它再次调用脚本，该脚本看到了参数并通过一个查询做出响应，获取了一个数据集，并把它发回给客户端的浏览器以供下载。而在应答中的Content-Type和Content-Disposition头部是调用header()函数来设置的。（这些必须在脚本产生任何其他输出前完成，否则header()函数没有任何效果。）

```
<?php  
# download.php - 检索结果集并将其作为一次下载发送给用户，而不是在一个web页面中显示  
  
require_once "Cookbook.php";  
require_once "Cookbook_Webutils.php";  
  
$title = "Result Set Downloading Example";  
  
# 如果没有下载参数，显示指示页面  
  
if (!get_param_val ("download"))  
{  
    # 构建包含下载参数的自指的URL  
    $url = get_self_path () . "?download=1";  
?>  
  
<html>.
```

```

<head>
<title><?php print ($title); ?></title>
</head>
<body bgcolor="white">

<p>
Select the following link to commence downloading:
<a href=<?php print ($url); ?>>download</a>
</p>

</body>
</html>

<?php
exit ();
} # end of "if"

# 存在下载参数; 检索一个结果集并将其作为一个制表符分隔的, 换行结束的文档发送到客户端。
# 使用application/octet-stream作为内容类型以触发浏览器的下载响应, 并建议一个默
# 认的文件名"result.txt"。

$conn =& Cookbook::connect ();
if (PEAR::isError ($conn))
    die ("Cannot connect to server: "
        . htmlspecialchars ($conn->getMessage ()));

$stmt = "SELECT * FROM profile";
$result =& $conn->query ($stmt);
if (PEAR::isError ($result))
    die ("Cannot execute query: "
        . htmlspecialchars ($result->getMessage ()));

header ("Content-Type: application/octet-stream");
header ("Content-Disposition: attachment; filename=\"result.txt\"");

while ($row =& $result->fetchRow ())
    print (join ("\t", $row) . "\n");
$result->free ();

$conn->disconnect ();
?>

```

脚本download.php使用一组我们曾介绍过的函数：

- `get_self_path()`返回脚本自己的路径。这可以被用来构建一个URL，该URL指向download.php脚本，并包含一个下载参数。
- `get_param_val()`函数决定该参数是否存在。

这些函数都包含在Cookbook_Webutils.php文件中，并会在19.1和19.5节中讨论。

另外一种构造可下载内容的方法，是生成查询结果集，并把它写入服务器端的某个文件中，接着压缩这个文件，并把压缩后的结果发给客户端浏览器。那么，浏览器很可能会运行某个解压缩工具来恢复原始数据。

18.10 使用模板系统生成Web页面

Using a Template System to Generate Web Pages

问题

你的脚本常常把从数据库中获取信息和生成HTML页面混在一起，现在你希望能将这部分的代码分开，这样它们能够单独执行。

解决方案

使用模板系统能让你设计Web页面的基本外观，同时允许你根据具体的请求填充数据。这样一来，你就可以个别地获取数据，然后把这些数据传送给模板系统以供生成输出。

讨论

在本章中，我们介绍的方法都是在一段脚本中，既获取页面数据又生成页面HTML代码。这样做好处是编写简单，但它也存在不足之处，当然也有取代的方法。

主要的不足之处在于，所有的事情都混在一起了，在获取页面数据的处理逻辑（业务或应用逻辑）和格式化数据以供显示的处理逻辑（表现逻辑）之间，函数的耦合度高。

一种替代的Web页面产生方法是使用某种模板系统，它具有将页面生成和处理阶段分离的功能。模板系统通过将业务逻辑从表现逻辑分离来实现这一点。如果你偏爱使用模型-视图-控制（MVC）体系结构的话，那么模板正好就能帮你执行MVC的逻辑，而且它能很好地处理MVC中的视图部分。

有很多的模板包可供选择。这里我们简要地讨论其中的两个（Ruby中的PageTemplate和PHP中的Smarty），这两个模板都是使用如下方法：

1. Web服务器调用一段脚本，来响应客户端对一个Web页面的请求。
2. 这段脚本决定必须显示什么，并获取所有相关的数据。
3. 这段脚本调用模板引擎，传送给引擎一个页面模板和要插入模板的相关数据。

4. 模板引擎在页面模板中寻找特殊的标记，这些标记指示在哪里插入数据，并用合适的值替换它们，然后输出结果。输出的就是作为对客户端请求的响应而返回给客户端的Web页面。

最典型的情况是，模板引擎用在Web上下文中以生成HTML，但这取决于你的需求是什么，你也可以生成其他格式的输出，比如文本或者XML。

模板系统提供的功能解耦的好处是：

- 程序员能在无须关心显示效果的情况下，改变获取将要显示信息的方式。例如，如果你要更换数据库系统，那么你只须在应用逻辑部分做相关调整，而无须改变模板（表现逻辑）。
- 页面设计者能够改变内容的显示方式，而无须关心其中的数据是怎么来的。这些数据可以来自文件、数据库或任意其他地方。该设计者仅须假设数据是可用的，而无须知道它来自何处。特别是，该设计者不须知道如何编写程序。

上述的最后一句话，很可能会引起争论。尽管在很多情况下设计者确实不用关心程序设计语言，但是模板中内嵌了特殊标记，该标记表明在哪里显示数值。根据待显示数据的复杂性和模板包的标记语法，标记很可能看起来非常简单，也可能看起来就像一段程序。但不管怎样，弄明白标记的含义，总是比理解使用通用语言编写的代码要简单得多。

在本节中，我们会重新学习前面的一些内容：生成段落（18.1节）、列表（18.2节）、还有表格（18.3节）。但在这里，我们关注的是如何设计页面模板以达到相同的显示效果。在开始之前，我们先简要地介绍一下如何使用标记：

值替换

如何指示在何处将值替换成模板。一个给定的数值，可以用来替换标记，也可以首先进行HTML编码或URL编码。

条件判断

如何根据一个测试的结果选择或者跳过模板的部分内容。

循环

如何重复地处理模板的部分内容，比如处理数组或哈希表数据结构中的每个元素。这在生成列表和表格时是非常有用的。

附录A指出如何获得PageTemplate和Smarty。在后面的内容中，我假设你已经成功安装了它

们。本节中讨论的示例模板和脚本，可以在光盘的apache目录下的pagetemplate和smarty子目录中找到。

示例中，后缀为.tpl的是模板文件，但这并不意味着你就必须使用这样的后缀约定。(比如，Smarty应用程序经常使用.tpl。)

在Ruby中使用模板生成Web页面

一个PageTemplate的Web应用程序由两个部分组成，一个HTML模板文件用于输出页面，一段Ruby脚本收集模板需要的数据，并触发模板引擎以处理指定的模板。在PageTemplate中，标记符为[%和%]。在这一对标记符之间的内容告诉PageTemplate该执行哪些操作。

值替换。要标明在PageTemplate模板文件中，一个值该放在哪，常使用的是[%var var_name%]指令：

```
[%var myvar%]
```

PageTemplate把这条指令替换为变量var_name的值。默认情况下，这个值直接被替代。如果要执行HTML编码或者URI编码，那么还需要在指令上加上合适的预处理器的名字：

```
[%var myvar :escapeHTML%]
[%var myvar :escapeURI%]
```

在这里，:escapeHTML具有和CGI.escapeHTML()函数一样的效果，它能保证特殊字符比如<和&被转化为<和&。而:escapeURI具有和CGI.escape()一样的功能。

指令[%var%]指向的任何值都必须是一个字符串，或者在使用to_s方法后能够产生字符串。

条件测试。每个条件测试语句都以[%if var_name%]开始，并以[%endif%]结束，中间可以使用[%else%]指令：

```
[%if myvar%
  myvar is true
[%else%
  myvar is false
[%endif%]
```

如果myvar变量的值是真，PageTemplate将处理紧接在[%if%]之后的内容。否则它处理[%else%]之后的内容。通常一个很简单没有“else”的条件判断，会省掉[%else%]部分的内容。

循环。指令[%in var_name%]提供了循环机制。在[%in%]和[%endin%]之间的模板内容将被循环单独处理。假设mylist是一个包含数据项的数组，每个元素是一个具有如下结构的hash表：

```
{ "first_name" => value, "last_name" => value }
```

这个hash表的成员是first_name和last_name，所以迭代能够处理该列表，并如下引用该项成员：

```
[%in mylist%]
  Name: [%var first_name%] [%var last_name%]
[%endin%]
```

如果一个列表是简单的有索引列表，那么它的成员不会有名字。在这种情况下，你需要在[%in%]指令中提供一个名字，通过这个名字引用成员元素：

```
[%in mylist: myitem%]
  Item: [%var myitem%]
[%endin%]
```

接下来的模板文件pt_demo tmpl，示范了上述概念：

```
<!-- pt_demo tmpl -->
<html>
<head>
<title>PageTemplate Demonstration</title>
</head>
<body>

<p>Value substitution:</p>
<p>
  My name is [%var character_name%],
  and I am [%var character_role%].
</p>

<p>Value substitution with encoding:</p>
<p>
  HTML-encoding: [%var str_to_encode :escapeHTML%];
  URL-encoding: [%var str_to_encode :escapeURI%]
</p>

<p>Conditional testing:</p>
<p>
  [%if id%]
    You requested information for item number [%var id%].
  [%else%]
    You didn't choose any item!
  [%endif%]
</p>

<p>Iteration:</p>
<p>
  Colors of the rainbow:
  [%in colors: color%]
    [%var color%]
  [%endin%]
</p>
</body>
</html>
```

我们还需要一个与此模板相应的脚本。一个使用PageTemplate的Ruby脚本基本框架如下：

```

#!/usr/bin/ruby -w

# 访问必须的模块
require "PageTemplate"
require "cgi"

...obtain data to be displayed in the output page...

# 创建模板对象
pt = PageTemplate.new

# 载入模板文件
pt.load("template_file_name")

# 赋值给模板变量
pt["var_name1"] = value1
pt["var_name2"] = value2
...

# 生成输出(cgi.out添加头部)
cgi = CGI.new("html4")
cgi.out { pt.output }

```

这段脚本首先包含了需要的Ruby库文件：PageTemplate来生成页面，cgi模块的输出方法提供一个简单的方法，向客户端发送包含所有必需头部的页面。

下一步是创建一个模板对象，导入模板文件，并把数值和模板中的变量关联起来。其中，filename 应该是一个绝对变量或是一个路径名，该路径名与脚本当前的工作目录相关。把一个值付给模板中变量的语法是：

```
pt["var_name"] = value
```

最后，这段脚本输出结果。这是通过模板对象的output方法实现的。如上例所示，脚本使用cgi对象的out方法，从模板中打印输出，out方法能够较好地添加HTML头。

现在，我们调整以下的脚本代码框架，并写下pt_demo.rb脚本以应用页面模板：pt_demo.tmpl模板文件：

```

#!/usr/bin/ruby -w
# pt_demo.rb - PageTemplate说明脚本

require "PageTemplate"
require "cgi"

pt = PageTemplate.new
pt.load("pt_demo.tmpl")
pt["character_name"] = "Peregrine Pickle"
pt["character_role"] = "a young country gentleman"
pt["str_to_encode"] = "encoded text: (<>&'\' =;)"
pt["id"] = 47846

```

```
pt["colors"] = ["red", "orange", "yellow", "green", "blue", "indigo", "violet"]
cgi = CGI.new("html4")
cgi.out { pt.output }
```

要想运行这个应用程序，首先须把pt_demo.rb和pt_demo tmpl文件拷贝到你的Ruby脚本目录，然后使用你的Web浏览器来请求这个脚本pt_demo.rb。例如，如果你将文件拷贝到通常的cgi-bin目录，那么就可以使用如下URL来请求该脚本：

```
http://localhost/cgi-bin/pt_demo.rb
```

也许你更愿意把 tmpl文件安装到非cgi-bin目录下，因为 tmpl文件毕竟不是可执行文件。如果你这么做了，那么请修改pt.load调用中的 pathname。

上面的指南对于开发模板生成显示从MySQL中检索出的信息的段落、列表和表格已经足够了。

段落的生成。我们的第一个基于PageTemplate的MySQL应用仅需要简单的段落生成。这个应用采用了18.2节中讨论的模式。将开发的脚本连接到MySQL服务器上，启动一个查询并获得一个结果集，然后输出几个段落的文字：当前时间、服务器版本、MySQL的用户名以及当前连接的数据库名字。要实现同样的功能的PageTemplate应用程序，需要一个模板文件，以及一个Ruby脚本代码，这里把它们命名为 pt_paragraphs tmpl 和 pt_paragraphs.rb。

首先，我们设计这个页面模板。根据应用的要求，这个模板其实是非常简单的。它需要一个值替换，而不需条件判断和循环。使用 [%var var_name%] 指令使模板对每个页面都包含简单段落：

```
<!-- pt_paragraphs tmpl -->
<html>
<head>
<title>[%var title :escapeHTML%]</title>
</head>
<body bgcolor="white">

<p>Local time on the MySQL server is [%var now :escapeHTML%].</p>
<p>The server version is [%var version :escapeHTML%].</p>
<p>The current user is [%var user :escapeHTML%].</p>
<p>The default database is [%var db :escapeHTML%].</p>

</body>
</html>
```

模板pt_paragraphs tmpl在 [%var%] 指令中使用了:escapeHTML() 函数，这里我们假设并不需要知道数据值是否包含特殊字符。

接下来，编写一段Ruby脚本，该脚本获取数据，并调用模板引擎。在模板中嵌入的指令指示相应的脚本需要为名为title、now、version、user和db的模板变量提供值：

```

#!/usr/bin/ruby -w
# pt_paragraph.rb - 生成HTML段落

require "PageTemplate"
require "cgi"
require "Cookbook"

title = "Query Output Display - Paragraphs"

dbh = Cookbook.connect

# 查询模板需要的数据
(now, version, user, db) =
  dbh.select_one("SELECT NOW(), VERSION(), USER(), DATABASE()")
db = "NONE" if db.nil?

dbh.disconnect

pt = PageTemplate.new
pt.load("pt_paragraphs tmpl")
pt["title"] = title
pt["now"] = now
pt["version"] = version
pt["user"] = user
pt["db"] = db

cgi = CGI.new("html4")
cgi.out { pt.output }

```

上面的脚本很像前面讨论的pt_demo.rb。主要的区别是，它使用了模块Cookbook作为它的连接方法，同时通过连接到MySQL数据库并启动查询来获得页面模板所需的数据。

列表的生成。列表包含了重复的元素，它们可以通过使用PageTemplate[%in%]指令生成，该指令迭代列表。下面的模板产生一个有序列表、一个无序循环和一个定义列表：

```

<!-- pt_lists.tml -->
<html>
<head>
<title>[%var title :escapeHTML%]</title>
</head>
<body bgcolor="white">

<p>Ordered list:</p>

<ol>
[%in list: item %]
  <li>[%var item :escapeHTML%]</li>
[%endin%]
</ol>

<p>Unordered list:</p>

<ul>

```

```

[%in list: item %]
  <li>[%var item :escapeHTML%]</li>
[%endin%]
</ul>

<p>Definition list:</p>

<dl>
[%in defn_list%]
  <dt>[%var note :escapeHTML%]</dt>
  <dd>[%var mnemonic :escapeHTML%]</dd>
[%endin%]
</dl>

</body>
</html>

```

首先，前两个列表仅仅在标签（``和``）上有所区别，它们使用的是同样的列表元素数据。这些数据来自一个简单数组，所以我们在`[%in%]`指令中提供了一个参数，这个参数能把列表的一个元素跟我们的变量联系起来。

对于定义列表，每个表单元都需要一个术语和一个定义。我们假设脚本提供了一个结构化的列表，该列表有名为`note`和`mnemonic`的两个成员。

从MySQL中获取列表数据，并处理模板的脚本如下：

```

#!/usr/bin/ruby -w
# pt_lists.rb - 生成HTML列表

require "PageTemplate"
require "cgi"
require "Cookbook"

title = "Query Output Display - Lists"

dbh = Cookbook.connect

# 获取用于有序或无序列表的项(创建一个"可伸缩"值的数组；列表实际由DBI::Row对象组成,
# 但对于单列行，对每个对象应用to_s方法会得到列值)

stmt = "SELECT item FROM ingredient ORDER BY id"
list = dbh.select_all(stmt)

# 获取用于一个定义列表的条目和定义(创建一个哈希值列表，每行一个哈希表)

defn_list = []
stmt = "SELECT note, mnemonic FROM doremi ORDER BY id"
dbh.execute(stmt) do |sth|
  sth.fetch_hash do |row|

```

```
defn_list << row
end
end

dbh.disconnect

pt = PageTemplate.new
pt.load("pt_lists.tmpl")
pt["title"] = title
pt["list"] = list
pt["defn_list"] = defn_list

cgi = CGI.new("html4")
cgi.out { pt.output }
```

回想一下，对于定义列表来说，模板期望的是每个元素都是结构化的，既有note也有mnemonic成员。脚本是通过生成一个hash数组来满足这个需求的，其中每个hash表都为如下形式：

```
{ "note" => val1, "mnemonic" => val2 }
```

通过从包含数据的表格中选择note列和mnemonic列，并用fetch_hash函数操作hash表中的值，我们很容易就实现了hash数组。

表格的生成。设计一个能产生HTML表格效果的模板，与前面的产生列表的模板是非常相似的，因为它们都有重复元素使用迭代。对于表格而言，被重复的元素是行（`<tr>`元素）。在每行中，你使用`<td>`元素为每个单元格写入数据。跟自然表格对应的数据结构是一组hash，每个hash中的元素都对应到表格的列上。

下面的示例展示了如何显示cd表中的内容，这个表有三列：year、artist和title。假设我们使用一个模板变量rows来存储表格内容，那么下面的模板就会生成表格。事实上，该模板会生成表格两次，一次是所有具有相同颜色的行，另一次是另一种颜色的行：

```
<!-- pt_tables.tmpl -->
<html>
<head>
<title>[%var title :escapeHTML%]</title>
</head>
<body bgcolor="white">

<p>HTML table:</p>

<table border="1">
<tr>
  <th>Year</th>
  <th>Artist</th>
  <th>Title</th>
</tr>
[%in rows%]
<tr>
  <td>[%var year :escapeHTML%]</td>
```

```

<td>[%var artist :escapeHTML%]</td>
<td>[%var title :escapeHTML%]</td>
</tr>
[%endin%]
</table>

<p>HTML table with rows in alternating colors:</p>

<table border="1">
<tr>
<th bgcolor="silver">Year</th>
<th bgcolor="silver">Artist</th>
<th bgcolor="silver">Title</th>
</tr>
[%in rows%]
[%if __ODD__ %]
<tr bgcolor="white">
[%else%]
<tr bgcolor="silver">
[%endif%]
<td>[%var year :escapeHTML%]</td>
<td>[%var artist :escapeHTML%]</td>
<td>[%var title :escapeHTML%]</td>
</tr>
[%endin%]
</table>

</body>
</html>

```

第一个表格模板生成一个“无格式的”表。第二个模板生成一个各行交替变换背景色的表。在相邻行之间切换颜色，可以通过一个条件判断实现。该判断使用内建的__ODD__变量，该变量遇奇数为真。

从数据库表格中获取数据，并处理模板的脚本代码如下：

```

#!/usr/bin/ruby -w
# pt_tables.rb - 生成HTML表格

require "PageTemplate"
require "cgi"
require "Cookbook"

title = "Query Output Display - Tables"

dbh = Cookbook.connect

# 获取表格行（创建一个哈希值列表，每行一个哈希表）

rows = []
stmt = "SELECT year, artist, title FROM cd ORDER BY artist, year"
dbh.execute(stmt) do |sth|
  sth.fetch_hash do |row|

```

```

    rows << row
  end
end

dbh.disconnect

pt = PageTemplate.new
pt.load("pt_tables.tmpl")
pt["title"] = title
pt["rows"] = rows

cgi = CGI.new("html4")
cgi.out { pt.output }

```

在PHP中使用Smarty生成Web页面

一个Smarty的Web应用包含两部分：产生输出页面的HTML模板文件和PHP脚本，脚本用于搜集模板需要的数据并调用模板引擎处理模板。在Smarty模板中，标记符是{and}，在标记符中是告诉Smarty你想要执行的各种命令。

值替换。为了表明在Smarty模板的哪里替换掉一个值，使用{\$ var_name }表示法。这个替换使用没有预处理的值。如果你需要使用HTML编码或者URL编码这个值，那么请加上修饰成分。下面三行代码分别是不经修改的替换一个值、经过HTML编码和URL编码：

```

{$myvar}
{$myvar|escape}
{$myvar|escape:"url"}

```

条件测试。条件测试以{if expr}开始，以{/if}结尾，中间可以使用一个或多个{elseif expr}语句或一个{else}语句。每个expr可以是一个变量或一个复杂的表达式（跟PageTemplate不同的是PageTemplate只允许一个变量）。这里有一个简单的if-then-else测试的例子：

```

{if $myvar}
  myvar is true
{else}
  myvar is false
{/if}

```

迭代。Smarty有多种迭代结构。命令{foreach}给包含要迭代的值的列表命名并指出循环体要引用值的名字。下面的结构描述了作为列表名字的\$mylist并使用myitem作为列表项的名字：

```

{foreach from=$mylist item=myitem}
  {$myitem}
{/foreach}

```

命令{section}给一个包含列表值的变量命名，该名字用于在loop中的列表变量：

```
{section loop=$mylist name=myitem}
{$mylist[myitem]}
{/section}
```

需要注意的是，指向列表元素的语法在{section}和{foreach}中是不同的。\$mylist[myitem]语法在对一个标量值列表迭代时非常有用。如果你需要在像关联数组这样的结构化值中进行迭代，那么使用\$mylist[myitem].member_name来引用每个结构成员。后续还要讨论列表和表格应用程序来解释这方面的语法。

接下来的模板文件sm_demo.tpl，展示了前面的几个基本概念：

```
<!-- sm_demo.tpl -->
<html>
<head>
<title>Smarty Demonstration</title>
</head>
<body>

<p>Value substitution:</p>
<p>
    My name is {$character_name},
    and I am {$character_role}.
</p>

<p>Value substitution with encoding:</p>
<p>
    HTML-encoding: {$str_to_encode|escape};
    URL-encoding: {$str_to_encode|escape:"url"}
</p>

<p>Conditional testing:</p>
<p>
    {if $id}
        You requested information for item number {$id}.
    {else}
        You didn't choose any item!
    {/if}
</p>

<p>Iteration:</p>
<p>
    Colors of the rainbow (using foreach):
    {foreach from=$colors item=color}
        {$color}
    {/foreach}
</p>
<p>
    Colors of the rainbow (using section):
    {section loop=$colors name=color}
        {$colors[color]}
    {/section}
</p>
</body>
</html>
```

除模板之外，我们还需要一个脚本文件。一个使用Smarty的PHP脚本概况如下：

```
<?php
require_once "Smarty.class.php";

...obtain data to be displayed in the output page...

# 创建模板对象
$smarty = new Smarty();

# 给模板变量赋值
$smarty->assign ("var_name1", value1);
$smarty->assign ("var_name2", value2);
...

# 处理模板以生成输出
$smarty->display ("template_file_name");
?>
```

在这里，Smarty.class.php文件让你有权限去访问Smarty。正如前面提到过的一样，我假设你已经正确安装了Smarty。此外，为了方便你的PHP脚本访问Smarty.class.php文件而不需要指出全路径名，你应该增加一个目录，在这里正确配置php.ini文件中的include_path变量。例如，如果Smarty.class.php安装在/usr/local/lib/php/smarty中，那么增加目录到include_path值中。

下面的这段框架性代码介绍了使用Smarty的几个主要步骤：创建一个新的Smarty模板对象，使用assign()函数指定模板变量的值，将模板的filename变量传给display()对象产生输出。

这个代码框架很容易修改产生下面的脚本，sm_demo.php和sm_demo.tmpl模板对应：

```
<?php
# sm_demo.php - Smarty说明脚本

require_once "Smarty.class.php";

$smarty = new Smarty();

$smarty->assign ("character_name", "Peregrine Pickle");
$smarty->assign ("character_role", "a young country gentleman");
$smarty->assign ("str_to_encode", "encoded text: (<>&'\" =;)\"");
$smarty->assign ("id", 47846);
$colors = array ("red", "orange", "yellow", "green", "blue", "indigo", "violet");
$smarty->assign ("colors", $colors);

$smarty->display ("sm_demo.tmpl");
?>
```

我们已经拥有了一个简单的Smarty应用（模板文件和使用它的PHP脚本），但我们还需要一点额外的设置去部署这个应用。Smarty使用一组目录，所以你需要先生成这些目录。默认

情况下，Smarty假设这些目录被放置在安装PHP脚本的目录下。下面的代码都假设该目录就是在你Apache文档根目录下的mcb（同样，在本章中使用的是相同的PHP目录）。

移动到mcb目录下，创建四个子目录：

```
% mkdir cache  
% mkdir configs  
% mkdir templates  
% mkdir templates_c
```

你刚创建的这四个目录，对于Web服务器必须是可读的，其中的两个还必须是可写的。

在Unix，要合适地设置组权限，然后更改这四个目录的组为web服务器使用。可以通过下面的命令设置组权限：

```
% chmod g+rx cache configs templates templates_c  
% chmod g+w cache templates_c
```

下一步，就是决定使用这些目录的组。请查看Apache的httpd.conf文件，并找到如下这行：

```
Group www
```

该行表明Apache运行使用www组ID。其他的公共值是nobody或apache。在下面的命令中使用组名，该命令作为root执行：

```
# chgrp www cache configs templates templates_c
```

这样，Smarty的目录设置就完成了。现在你可以部署Smarty的示范应用程序了，你只需将sm_demo.php拷贝到mcb目录下，同时将sm_demo.template拷贝到mcb/templates目录下。然后，在你的浏览器中输入：

```
http://localhost/mcb/sm_demo.php
```

对下面的每个应用程序，使用相同的规则，即PHP脚本放置在mcb目录下，Smarty模板放置在mcb/templates目录下。

如果你想将你的应用程序目录放到其他地方，那么请创建这些目录。之后，你需要让你的PHP脚本知道如何在这些目录下找到相关的文件。例如，你将Smarty目录放在/usr/local/lib/mcb/smarty下，那么请这样修改你的Smarty：

```
$smarty = new Smarty ();  
$smarty->cache_dir = "/usr/local/lib/mcb/smarty/cache";  
$smarty->config_dir = "/usr/local/lib/mcb/smarty/configs";  
$smarty->template_dir = "/usr/local/lib/mcb/smarty/templates";  
$smarty->compile_dir = "/usr/local/lib/mcb/smarty/templates_c";
```

段落的生成。这跟PageTemplate中的步骤是一样的：

```

<!-- sm_paragraphs.tpl -->
<html>
<head>
<title>{$title|escape}</title>
</head>
<body bgcolor="white">

<p>Local time on the MySQL server is {$now|escape}.</p>
<p>The server version is {$version|escape}.</p>
<p>The current user is {$user|escape}.</p>
<p>The default database is {$db|escape}.</p>

</body>
</html>

```

这个模板仅使用了 `{$ var_name}` 值替换操作，同时加上`escape`修饰符以告诉Smarty需执行HTML编码操作。

模板之后，我们需要如下脚本：

```

<?php
# sm_paragraphs.php - 生成HTML段落

require_once "Smarty.class.php";
require_once "Cookbook.php";

$title = "Query Output Display - Paragraphs";

$conn =& Cookbook::connect ();
if (PEAR::isError ($conn))
    die ("Cannot connect to server: "
        . htmlspecialchars ($conn->getMessage ()));

$result =& $conn->query ("SELECT NOW(), VERSION(), USER(), DATABASE()");
if (PEAR::isError ($result))
    die (htmlspecialchars ($result->getMessage ()));
list ($now, $version, $user, $db) = $result->fetchRow ();
$result->free ();
if (!isset ($db))
    $db = "NONE";

$conn->disconnect ();

$smarty = new Smarty ();
$smarty->assign ("title", $title);
$smarty->assign ("now", $now);
$smarty->assign ('version', $version);
$smarty->assign ("user", $user);
$smarty->assign ("db", $db);
$smarty->display ("sm_paragraphs.tpl");

?>

```

列表的生成。下面的模板生成三个列表。前面的两个分别是有序和无序列表，这两个列表使用了相同的数据源。第三个是定义列表，这需要在循环中使用两个变量：

```
<!-- sm_lists.tpl -->
<html>
<head>
<title>{$title|escape}</title>
</head>
<body bgcolor="white">

<p>Ordered list:</p>

<ol>
{foreach from=$list item=cur_item}
<li>{$cur_item|escape}</li>
{/foreach}
</ol>

<p>Unordered list:</p>

<ul>
{foreach from=$list item=cur_item}
<li>{$cur_item|escape}</li>
{/foreach}
</ul>

<p>Definition list:</p>

<dl>
{section loop=$defn_list name=cur_item}
<dt>{$defn_list[cur_item].note|escape}</dt>
<dd>{$defn_list[cur_item].mnemonic|escape}</dd>
{/section}
</dl>

</body>
</html>
```

定义列表使用了`{section}`命令和`$list [item].member`概念，这在前面已经介绍过。

下面给出的是处理模板的脚本，对于定义列表而言，这段脚本生成了一组关联的数组：

```
<?php
# sm_lists.php - 生成HTML列表

require_once "Smarty.class.php";
require_once "Cookbook.php";

$title = "Query Output Display - Lists";

$conn =& Cookbook::connect();
```

```

if (PEAR::isError ($conn))
    die ("Cannot connect to server: "
        . htmlspecialchars ($conn->getMessage ()));

# 获取用于有序或无序列表的项 (创建一个"可伸缩"值的数组)

$stmt = "SELECT item FROM ingredient ORDER BY id";
$result =& $conn->query ($stmt);
if (PEAR::isError ($result))
    die (htmlspecialchars ($result->getMessage ()) );
$list = array ();
while ($item = $result->fetchRow ())
    $list[] = $item;
$result->free ();

# 获取用于一个定义列表的条目和定义 (创建一个关联数组的数组)

$stmt = "SELECT note, mnemonic FROM doremi ORDER BY id";
$defn_list =& $conn->getAll ($stmt, array (), DB_FETCHMODE_ASSOC);
if (PEAR::isError ($defn_list))
    die (htmlspecialchars ($result->getMessage ()) );

$conn->disconnect ();

$smarty = new Smarty ();
$smarty->assign ("title", $title);
$smarty->assign ("list", $list);
$smarty->assign ("defn_list", $defn_list);
$smarty->display ("sm_lists.tpl");

?>

```

这段脚本使用了`getAll`函数去获取定义列表中的所有值，这是一个链接对象方法，该方法执行一个查询，并在单个调用中获取结果集。把访问模式设为：`DB_FETCHMODE_ASSOC`，你就能得到一组关联数组，每个数组都有名为`node`和`mnemonic`的成员。

表格的生成。要生成一个HTML表格，我们需要遍历表中的每一行，以产生对应的`<tr>`元素。像在前面应用中的定义列表一样，使用`{section}`很容易实现这点。同时通过使用`$list[item].member`语法在行中为每个单元引用相应数值。下面的代码生成两个表；一个是“无格式”的表，另一个是相邻行颜色交替变化的表：

```

<!-- sm_tables.tpl -->
<html>
<head>
<title>{$title|escape}</title>
</head>
<body bgcolor="white">

```

```

<p>HTML table:</p>

<table border="1">
<tr>
    <th>Year</th>
    <th>Artist</th>
    <th>Title</th>
</tr>
{section loop=$rows name=row}
<tr>
    <td>{$rows[row].year|escape}</td>
    <td>{$rows[row].artist|escape}</td>
    <td>{$rows[row].title|escape}</td>
</tr>
{/section}
</table>

<p>HTML table with rows in alternating colors:</p>

<table border="1">
<tr>
    <th bgcolor="silver">Year</th>
    <th bgcolor="silver">Artist</th>
    <th bgcolor="silver">Title</th>
</tr>
{section loop=$rows name=row}
<tr bgcolor="{cycle values="white,silver"}">
    <td>{$rows[row].year|escape}</td>
    <td>{$rows[row].artist|escape}</td>
    <td>{$rows[row].title|escape}</td>
</tr>
{/section}
</table>

</body>
</html>

```

为了能产生一个相邻行颜色交替变换的表，模板中使用了 {cycle} 命令，这个命令交替地选择白色或者银色，这个命令为后续的行列表迭代交替地选择白色或者银色。{cycle} 为任务提供了方便，否则你就要使用条件表达式和返回当前迭代数（从1开始）的\$smarty.section.list.iteration值来进行处理。

```

{if $smarty.section.row.iteration % 2 == 1}
<tr bgcolor="white">
{else}
<tr bgcolor="silver">
{/if}

```

下面的脚本从数据库表格中获取数据，然后处理模板：

```

<?php
# smtables.php - 生成HTML表格

require_once "Smarty.class.php";

```

```
require_once "Cookbook.php";
$title = "Query Output Display - Tables";
$conn =& Cookbook::connect ();
if (PEAR::isError ($conn))
    die ("Cannot connect to server: "
        . htmlspecialchars ($conn->getMessage ()));
# Fetch items for table
$stmt = "SELECT year, artist, title FROM cd ORDER BY artist, year";
$rows =& $conn->getAll ($stmt, array (), DB_FETCHMODE_ASSOC);
if (PEAR::isError ($rows))
    die (htmlspecialchars ($result->getMessage ()));
$conn->disconnect ();
$smarty = new Smarty ();
$smarty->assign ("title", $title);
$smarty->assign ("rows", $rows);
$smarty->display ("sm_tables.tpl");
?>
```


用MySQL处理Web输入

Processing Web Input With MySQL

19.0 引言

Introduction

前面的几章介绍了各种各样的数据库信息获取及Web页面的展示方法，这是单方向地使用MySQL数据库（即数据总是从MySQL向Web服务器流动），但在Web编程中，我们还经常需要将数据从Web服务器导入MySQL数据库中。例如，你正在处理一个汇总表单，那么你可能需要将表单的信息存储以供后续使用。如果表单中包含了搜索关键字，那么你就可以使用这些关键字，在数据库中搜索用户关心的信息。

MySQL以一种简单易懂的方式支持我们的操作，它像仓库一样存储着信息，并支持各种搜索。但需要从用户处获得输入时，你得创建Web表单并把它发送给用户。MySQL对这一点也提供了支持，我们常常需要根据数据库内容来生成表单各种元素：比如radio按钮、复选框、弹出菜单或者滚动列表：

- 你可以从表格中选择一组元素，它们可以是国家、州或者省这样的信息，然后把这组元素转化为一个下拉式菜单，这个菜单可以用来收集用户的地址信息。
- 你可以使用一组ENUM列的合法数值，该列包含所有的招呼用语（如先生、小姐、女士），然后根据这组数值生成radio按钮。
- 你还可以使用存货数据库中的颜色、尺寸、或格式等信息，生成一张衣服的订单。
- 如果你还有一个应用程序，该程序允许用户选择一个数据库或表格，那么你可以查询MySQL数据库以获取其中的数据库和表格名称，并据此生成一组元素。

根据数据库内容生成表单元素，你能够在很大程度上缓解程序员的压力，一个使用数据库内容的脚本必须根据数据库的变化而作相应的调整。比如添加一个新的国家，在表格中生

成新的一行，增加一个新的招呼用语，改变ENUM列的定义。在这几个例子中，你通过更新数据库改变了表单中的一组元素，而不是通过更新脚本来做到这一点。这是很关键的，这意味着该脚本能够自动适应数据库内容的变化。

这里我们介绍与Web输入处理相关的一些主题：

生成表单和表单元素

使用数据库内容构建表单的一种常见方法，是从数据库表格中选择一组元素，然后据此在一个HTML列表中插入选项。这种做法在ENUM列和像radio按钮或下拉菜单这样的单取表单之间是非常自然的。像radio按钮和下拉菜单这两种情况，都只允许在多个值中选择其中的一个。类似的，在SET列和多取表单如复选框组之间，也存在着自然的对应关系，其中的一个或多个选项能被同时选中。为构建基于元数据的表单元素，需要我们从INFORMATION_SCHEMA中获取表格元数据，提取合法的ENUM和SET值并把它们作为表单元素。

使用数据库内容初始化表单

除了根据数据库内容创建结构化的表单外，你还可以初始化表单。举个例子，为了允许用户修改一条记录，你需要先读出这条记录，然后把它导入表单的各项中，最后将它发送给用户。

处理Web输入

在这里，Web输入不仅包括表单项，还包括上传的文件，或者URL中出现的参数。但无论你是如何获取这些输入的，你都需要面对一组共同的问题：提取并编码信息，且执行必需的合法性检查，然后再编码信息以构建合法的语句，从而避免不规范的语句或存储不准确的信息。

接下来，我们会介绍几种常用的技巧。这其中包括使用MySQL来提供基于Web的搜索界面，创建包含向前翻页和向后翻页的多页显示，执行每页的点击计数和日志，将Apache日志导入数据库等。

本章中使用到的脚本，都放在光盘的tables目录下。其中Perl、Ruby、PHP和Python的例子，请到Apache目录下查找。例子中使用的辅助性流程，都存放在lib目录下。Java (JSP) 的例子，请到tomcat目录下寻找。我们假设你已经成功安装了mcb应用程序环境（请参考17.2节中关于如何配置Apache从而可以运行脚本并且获得他们的类库文件这部分内容）。

需要注意的是尽管在本节中使用的脚本主要是让你在Web浏览器中调用。其中大多数脚本 (JSP相关的除外)，还能够通过命令行执行（参见17.2节）。

为了提供一个供讨论的固定的上下文环境，本章中许多有关表单处理的例子，都基于下面的交互场景：在构建农场的商业行为中，你做的是按订单生产瓷牛小雕像，且你还需要设计一个在线下订单的应用，这个应用允许用户在产品的不同属性间做选择。每个订单都包含以下内容：

颜色

在任一个特定时刻列出的可选颜色经常会改变，为了方便制作，这些值可以事先存储在数据库中，如果你想改变用户能够选择的颜色，请更新数据库表格。

尺寸

尺寸通常有三种选择：小、中、大。这三个值非常自然地对应到ENUM列。

对所有瓷器都重要的附件：

包括铃铛、牛角、飘扬的尾带、鼻环。附件可以用一个SET列表示，因为客户也许需要选择多个附件。此外，根据经验你知道大多数客户会选择铃铛和牛角，所以把这两个作为列的默认值是很有必要的。

客户的名字和地址（街道、城市、州）

所有可能的州名已经存在states表中，所以我们只要直接使用便可。

根据上述讨论，一张cow_order的表可以是这样：

```
CREATE TABLE cow_order
(
    id          INT UNSIGNED NOT NULL AUTO_INCREMENT,
    # 牛的颜色，雕像尺寸和其他附加项
    color       VARCHAR(20),
    size        ENUM('small', 'medium', 'large') DEFAULT 'medium',
    accessories SET('cow bell', 'horns', 'nose ring', 'tail ribbon')
                DEFAULT 'cow bell,horns',
    # 客户姓名，街道，城市和所在州(缩写)
    cust_name   VARCHAR(40),
    cust_street VARCHAR(40),
    cust_city   VARCHAR(40),
    cust_state  CHAR(2),
    PRIMARY KEY (id)
);
```

其中 id 列为每行记录提供了唯一的标识。设计这样一个列是个好主意，而且事实上我们在19.4节中常常用到id列，第19.4节中介绍了如何使用Web表单编辑数据库中的记录。对这类活动，你必须能够告诉数据库你将要更新的是那一行，而如果没有这里的id列，这将是非常困难的。

可用颜色在cow_color表中维护：

```
CREATE TABLE cow_color (color CHAR(20));
```

作为示范，假设cow_color表包含以下几行：

color
Black
Black & White
Brown
Cream
Red
Red & White
See-Through

19.1 编写脚本生成Web表单

Writing Scripts That Generate Web Forms

问题

你希望写一段脚本来收集用户输入的信息。

解决方案

在你的脚本中构建一个Web表单，并把它发送给用户。该脚本还可以写成这样：当用户填写好Web表单并提交后，Web服务器还会调用这个脚本，并用它来处理用户的输入。

讨论

Web表单是一种非常方便的方法，它允许你的访问者提交诸如查询关键字，调查问卷这样的信息。同时，表单还给开发者带来了很多的便利，这是因为它提供了一种结构化的方法将数据和名字关联了起来。

表单以<form>和</form>标签为始终，在这两个标签间，你可以放入其他的HTML结构，包括特殊的元素比如页面中的输入域。值得注意的是，<form>标签应包含两个属性：action

和method。其中，action属性告诉浏览器在用户提交表单时该执行的操作。该操作一般是处理这个表单的脚本的URL。而method属性表明浏览器在提交表单时，该发送何种类型的HTTP请求，它的取值一般是get或者post。19.5节中讨论了这两种方法的异同，但在这里，我们将一直使用post。

本章中介绍的大多数基于表单的Web脚本，以下行为都是共同的：

- 第一次被调用时，该脚本生成一个表单并把它发送给用户。
- 表单中的action指回了这个脚本，因此当用户填完表单并提交它时，Web服务器将再次调用该脚本来处理表单的内容。
- 脚本判断它这是第一次被该用户调用，还是需要处理一个返回的表单，这是根据URL的参数来判断的。对于第一次调用，对应的URL中没有参数。

当然，你并不是非得遵守上述行为方式不可。另一种做法是将表单放在静态的HTML页面中，并让该表单指向处理它的脚本。还有一种做法是使用一个脚本生成表单，使用另一个处理它。

如果采用的是本文介绍的方法，那么一个脚本需要获知它自己在Web服务器中的目录，以及表单action属性中的值。例如，如果一个脚本被存放在/cgi-bin/myscript中，你可能会这么写你的<form>标签：

```
<form action="/cgi-bin/myscript" method="post">
```

然而，我们的每一个语言API都提供了让脚本获取工作目录的方法。这意味着没有脚本需要把这个目录信息绑定在代码中。所以，你可以把脚本安装在任何你想要的地方。

Perl

在Perl脚本中，CGI.pm模块提供三个构建<form>元素和action属性的有用方法。start_form()和end_form()函数生成开始标签和关闭标签，url()函数返回脚本的路径信息。使用这些方法生成form的一段脚本如下：

```
print start_form (-action => url (), -method => "post");
# …此处构建form元素…
print end_form ();
```

`start_form()`函数提供了一个默认的post请求，所以当你构建post的表单时，可以忽略method参数。

Ruby

在Ruby脚本中，创建一个cgi对象，并使用该对象的form方法来生成表单。该方法的参数提供了`<form>`标签属性，且方法调用后的代码块提供了表单的内容。要想获得脚本的路径，可以使用ENV环境中的SCRIPT_NAME成员：

```
cgi.out {  
  cgi.form("action" => ENV["SCRIPT_NAME"], "method" => "post") {  
    # ...此处构建form元素...  
  }  
}
```

form方法提供了一个默认的post请求方法。所以如果你打算使用post，那么你可以省略该方法的参数。

脚本的路径名还可以通过`cgi.script_name`方法获得。

PHP

在PHP中，一个脚本的路径名可以从`$HTTP_SERVER_VARS`或`$_SERVER`数组的`PHP_SELF`成员中获得。不幸的是，检查多个数据源是必须的，因为PHP在不同平台环境上的支持或配置不同。所以如果存在一个能获得工作路径的应用，该应用对PHP而言将是非常有用的。接下来的这个函数`get_self_path()`展示了如何使用`$_SERVER`（如果它可用的话，否则使用`$HTTP_SERVER_VARS`）。这个函数倾向于最近介绍的语言特点，但仍然能够在老版本的PHP环境中运行：

```
function get_self_path ()  
{  
  global $HTTP_SERVER_VARS;  
  
  $val = NULL;  
  if (isset ($_SERVER["PHP_SELF"]))  
    $val = $_SERVER["PHP_SELF"];  
  else if (isset ($HTTP_SERVER_VARS["PHP_SELF"]))  
    $val = $HTTP_SERVER_VARS["PHP_SELF"];  
  return ($val);  
}
```

`$HTTP_SERVER_VARS`是一个全局变量，但在非全局范围中必须使用`global`关键字显式声明它（比如说在一个函数体中）。`$_SERVER`是一个“超全局”数组，在任何范围内都能被访问且无须声明为`global`。

函数`get_self_path()`是Cookbook_Webutils.php库文件的一部分，该文件位于光盘的lib目录下。如果你把该文件安装在一个目录中，并让PHP在用`include`语句引入一个文件时搜索它，可以使用以下脚本：

```
include "Cookbook_Webutils.php";  
  
$self_path = get_self_path ();  
print ("<form action=\"$self_path\" method=\"post\">\n");  
# …此处构建form元素…  
print ("</form>\n");
```

Python

Python脚本能导入os模块，并访问os.environ对象中的SCRIPT_NAME成员以获得脚本的路径名：

```
import os  
  
print "<form action=\"" + os.environ["SCRIPT_NAME"] + "\" method=\"post\">"  
# …此处构建form元素…  
print "</form>"
```

Java

在JSP页面中，同时使用JSP处理器提供的显式的请求对象，能够获得请求路径的信息。使用该对象的getRequestURI()方法：

```
<form action=<%= request.getRequestURI () %> method="post">  
<%-- ... generate form elements here ... --%>  
</form>
```

参考

本节中的例子在表单开始标签和关闭标签之间，都有一个空白的部分，要想使一个表单变得有用；你需要根据你想从用户处获取信息的类型，创建body元素。当然，把这些元素放入脚本中也是可能的，19.2节和19.3节描述了MySQL如何帮助你根据数据库内容来创建各种元素。

19.2 根据数据库内容构建单取表单元素

Creating Single-Pick Form Elements from Database Content

问题

一个表单（form）需要能够提供一个域，这个域给用户提供了几个选项，并允许用户仅选择其中的一项。

解决方案

使用一个单选列表元素。这包括radio按钮组、弹出菜单、滚动列表等。

讨论

单选表单元素，允许你提供多个选择，并只允许其中的一个被选中。我们的“构建一只奶牛”示例场景中，就涉及到多组单选的选项：

- 下表是cow_color表格中的颜色列表。这些颜色可通过执行下面的语句得到：

```
mysql> SELECT color FROM cow_color ORDER BY color;
+-----+
| color |
+-----+
| Black |
| Black & White |
| Brown |
| Cream |
| Red |
| Red & White |
| See-Through |
+-----+
```

需要注意的是其中一些颜色包含了&字符，这个字符在HTML中属于特殊字符。这也就意味着&字符在放入列表元素中时，需要经过HTML编码。（我们将对所有的列表元素执行HTML编码，这是一个好的编程习惯）。

- cow_order表格的size列包含了全部的合法雕像尺寸。这一列被声明为ENUM，所以它可能的取值，以及默认值都可以从INFORMATION_SCHEMA中获得：

```
mysql> SELECT COLUMN_TYPE, COLUMN_DEFAULT
-> FROM INFORMATION_SCHEMA.COLUMNS
-> WHERE TABLE_SCHEMA='cookbook' AND TABLE_NAME='cow_order'
-> AND COLUMN_NAME='size';
+-----+-----+
| COLUMN_TYPE | COLUMN_DEFAULT |
+-----+-----+
| enum('small','medium','large') | medium |
+-----+-----+
```

- 下表是美国各个州名字和缩写的列表。这可以从states表中获得：

```
mysql> SELECT abbrev, name FROM states ORDER BY name;
+-----+-----+
| abbrev | name |
+-----+-----+
| AL     | Alabama |
| AK     | Alaska  |
| AZ     | Arizona |
| AR     | Arkansas |
| CA     | California |
| CO     | Colorado |
| CT     | Connecticut |
...
...
```

选择的数目将随着每个列表而变化。如上所示，共有三个雕像尺寸，七种颜色，五十个州。那么不同数目的选择，直接导致如何将列表显示在表单中：

- 雕像尺寸最好是显示为一组radio按钮，或者一个弹出菜单，而不必使用一个滚动列表。这是因为，可选的选项较少。
- 颜色列表使用上述三种类型的显示方式都是合理的。这是因为，颜色列表比较小，所以一组radio按钮不会占用太多空间，但它又足够大以至于你也许想要使用滚动列表——特别是如果你想添加额外的颜色时。
- 州列表看起来需要太多的选项，你可能不希望将它们用一组radio按钮来表示，所以将州列表用弹出菜单或滚动列表来表示，将更合适些。

下面的讨论描述了这三类元素的HTML语法，并展示如何从脚本中生成它们：

Radio按钮

一组radio按钮包含了类型radio的元素，这类元素拥有相同name的属性。每个元素还包含一个value属性。在标签后，可以跟上一个显示条。为了表示一个默认显示的元素，得添加一个checked属性。下面的这组radio按钮，显示了可能的牛雕塑尺寸，并使用checked属性将medium标识为初始的被选中值：

```
<input type="radio" name="size" value="small" />small  
<input type="radio" name="size" value="medium" checked="checked" />medium  
<input type="radio" name="size" value="large" />large
```

弹出菜单

一个弹出菜单是一个列表，该列表以<select>为始，以</select>为终。列表内的每一个元素都被<option>和</option>标签包住。同时，每个<option>元素都有一个value属性，它的实体提供了显示的标语。为了表示一个默认的选项，需要在合适的<option>元素中添加一个selected属性。如果没有一个元素被标记，那么第一个元素就是默认元素，这跟下面的示例是一样的：

```
<select name="color">  
  <option value="Black">Black</option>  
  <option value="Black & White">Black & White</option>  
  <option value="Brown">Brown</option>  
  <option value="Cream">Cream</option>  
  <option value="Red">Red</option>  
  <option value="Red & White">Red & White</option>  
  <option value="See-Through">See-Through</option>  
</select>
```

滚动列表

一个滚动列表是显示在一个盒子中的一组元素。这个列表可能会包含超过该盒显示面积的元素，在这种情况下浏览器会显示一个滚动条以供用户浏览其他的元素。滚动列

表的HTML语法跟弹出菜单很相似，不同的是滚动列表的<select>标签中包含了一个size属性，该属性表明共有多少行应该是可见的。默认情况下，滚动列表是一个单选元素。19.3节中讨论了如何允许多选的方法。

下例的单选滚动列表包含了美国的每一个州，且只允许一次显示六个：

```
<select name="state" size="6">
<option value="AL">Alabama</option>
<option value="AK">Alaska</option>
<option value="AZ">Arizona</option>
<option value="AR">Arkansas</option>
<option value="CA">California</option>
...
<option value="WV">West Virginia</option>
<option value="WI">Wisconsin</option>
<option value="WY">Wyoming</option>
</select>
```

radio按钮、弹出菜单还有滚动列表有以下共同点：

元素都有名字

当用户提交表单时，浏览器会将这个名字和用户选中的值关联在一起。

一组值，每个值对应列表中的一个元素

这决定了哪些值是可被选中的。

一组标语，每个元素一个

这决定了用户能在表单中看到什么。

一个可选的默认值

这决定了哪一个元素是在浏览器显示列表时，初始被选中的。

要使用数据库内容为表单生成一个列表，需要执行一个语句以选择合适的值和标语，将其中的特殊字符编码，然后添加合适类型的HTML标签。此外，你还应该提示一个默认的选项，添加一个被选中或被选择的属性到恰当的列表元素中。

让我们考虑一下如何为颜色和州信息生成表单元素。这两种表单都是通过从表格中获取一组列值来生成的。然后我们构建雕塑尺寸列表，该列表的值来自于列的定义而非列的内容。

在JSP中，你能使用JSTL标签为这些颜色显示一组radio按钮。颜色的名字既使用了值又使用了标语，所以你应该将它显示两次：

```
<sql:query dataSource="${conn}" var="rs">
    SELECT color FROM cow_color ORDER BY color
```

```
</sql:query>

<c:forEach items="${rs.rows}" var="row">
    <input type="radio" name="color"
        value=">" /><br />
</c:forEach>
```

<c:out>函数执行HTML实体编码，所以在一些颜色中包含的&字符会被转化为&，这就避免了Web页面中可能会出现的一些显示问题。

另外，要显示一个弹出菜单，获取的语句可以是相同的，但单行获取的循环略有区别：

```
<sql:query dataSource="${conn}" var="rs">
    SELECT color FROM cow_color ORDER BY color
</sql:query>

<select name="color">
<c:forEach items="${rs.rows}" var="row">
    <option value=">">
        <c:out value='${row.color}'/>></option>
</c:forEach>
</select>
```

弹出式菜单可以轻松变为滚动列表。所以你需要做的就是在<select>标签中添加一个size属性。例如，要使三个颜色都同时可见，可以这样：

```
<sql:query dataSource="${conn}" var="rs">
    SELECT color FROM cow_color ORDER BY color
</sql:query>

<select name="color" size="3">
<c:forEach items="${rs.rows}" var="row">
    <option value=">">
        <c:out value='${row.color}'/>></option>
</c:forEach>
</select>
```

为一组州的信息生成一个表单，跟上述过程是相似的，区别在于labels与值不同。为了让labels在用户看来更有意义，需要显示州的全称。但是当表单被提交时返回的值应该是一个缩写，因为这才是存储在cow_order表中的内容。为了产生一种这样的滚动列表，同时选出缩写和全称，然后把它们插入每个列表元素的合适部位。例如：

```
<sql:query dataSource="${conn}" var="rs">
    SELECT abbrev, name FROM states ORDER BY name
</sql:query>

<select name="state">
<c:forEach items="${rs.rows}" var="row">
    <option value=">">
```

```
<c:out value="${row.name}" /></option>
</c:forEach>
</select>
```

上面的JSP示例使用一种方法来单独显示所有的列表元素。列表元素的生成过程在基于CGI.pm的Perl脚本中使用了另外一种流程：先抽取数据库内容，然后把内容传给一个函数，该函数返回一个能显示表单的字符串。这种生成单取元素的函数是radio_group()、popup_menu()和scrolling_list()。其中有几个参数是相同的：

name

你给列表元素取的名称。

values

列表中每个元素的值。这个值应该是指向一个数组的。

labels

标语是跟每个值关联在一起的，这个参数是可选的。如果没有labels，CGI.pm应该指向一个hash函数，该函数将每一个值和它对应的labels连接在一起。例如，为牛的颜色生成一个列表，那么值和labels就是一样的，所以就不需要labels参数。然而，如果生成一个州信息的列表，labels就应链接到一个hash表上，该hash表把每个州的缩写和它的全称链接在一起。

default

在列表中初始被选中的元素，这个参数是可选的。以radio按钮为例，如果没有这个参数，CGI.pm自动选择第一个按钮作为默认值。为了避免这种行为，你可以提供一个不在列表中的值作为默认值。

部分函数还接受额外的参数。比如radio_group()，你能提供一个linebreak的参数以规定该组按钮是垂直显示，还是水平显示。函数scrolling_list()接受一个size参数，该参数标识共多少元素可以被一起显示。(CGI.pm的文档还描述了这里根本不用的参数。例如，存在将我们的radio按钮显示为列表形式风格的参数，但是在这里我们并不打算使用这些参数。)

要使用cow_color表格中的信息构建一个表单，我们需要：

```
my $color_ref = $dbh->selectcol_arrayref (
    "SELECT color FROM cow_color ORDER BY color");
```

函数selectcol_arrayref()返回一个指向数组的索引，该数组就是CGI.pm函数要创建列表元素时使用的参数。为了创建一组radio按钮，一个弹出菜单，或者一个单取滚动列表，需要调用下面的函数：

```

print radio_group (-name => "color",
                  -values => $color_ref,
                  -linebreak => 1);      # 垂直的显示按钮

print popup_menu (-name => "color",
                  -values => $color_ref);

print scrolling_list (-name => "color",
                      -values => $color_ref,
                      -size => 3);          # 每次显示3项

```

颜色列表中，值和标语是一样的，所以不需要再使用labels参数。默认情况下，CGI.pm会使用值作为标语。需要注意的是，在这里我们没有将颜色做HTML编码，即使它们中有些值包含&符号。CGI.pm函数在生成表单元素时，自动执行HTML编码，这跟其他的非表单函数不同。

要生成一个州列表，且其值是缩写，其labels是全称的，我们就需要使用labels参数。这个参数应该是指向一个hash表，该表能将每个值映射为对应的labels：

```

my @state_values;
my %state_labels;
my $sth = $dbh->prepare ("SELECT abbrev, name
                           FROM states ORDER BY name");
$sth->execute ();
while (my ($abbrev, $name) = $sth->fetchrow_array ())
{
    push (@state_values, $abbrev);  # 保存数组中的每个值
    $state_labels{$abbrev} = $name;  # 将每个值映射到它的标签
}

```

将生成的列表和hash表传递给popup_menu()函数还是scrolling_list()函数，取决于你想要生成的列表类型：

```

print popup_menu (-name => "state",
                  -values => \@state_values,
                  -labels => \%state_labels);

print scrolling_list (-name => "state",
                      -values => \@state_values,
                      -labels => \%state_labels,
                      -size => 6);          # 每次显示6项

```

Ruby的cgi模块也提供了生成radio按钮、弹出菜单、滚动列表的方法。你可以查看form_element.rb脚本以了解如何使用这些方法。但在这里，我不打算详细讨论它们，因为我发现它们用起来有点别扭，特别是当你需要确保每个值都已经被恰当地过滤且一定的成员被默认选中时。

如果你正使用的API不提供能生成表单的函数（或者这些函数不方便使用），那么你就得在从MySQL中获得列表元素之后显示HTML，或者编写应用脚本以生成表单元素。下面讨论的考虑了这两种方法，使用的是PHP和Python环境。

在PHP中，从cow_color表中获取的值列表，可以显示在一个弹出菜单中：

```
print ("<select name=\"color\">\n");
$stmt = "SELECT color FROM cow_color ORDER BY color";
$result =& $conn->query ($stmt);
if (!PEAR::isError ($result))
{
    while (list ($color) = $result->fetchRow ())
    {
        $color = htmlspecialchars ($color);
        print ("<option value=\"$color\">$color</option>\n");
    }
    $result->free ();
}
print ("</select>\n");
```

完成相同功能的Python是：

```
stmt = "SELECT color FROM cow_color ORDER BY color"
cursor = conn.cursor ()
cursor.execute (stmt)
print "<select name=\"color\">"
for (color, ) in cursor.fetchall ():
    color = cgi.escape (color, 1)
    print "<option value=\"%s\">%s</option>" % (color, color)
cursor.close ()
print "</select>"
```

州列表需要不同的值和labels，所以代码看起来就显得稍微复杂些。在PHP中：

```
print ("<select name=\"state\">\n");
$stmt = "SELECT abbrev, name FROM states ORDER BY name";
$result =& $conn->query ($stmt);
if (!PEAR::isError ($result))
{
    while ($row = & $result->fetchRow ())
    {
        $abbrev = htmlspecialchars ($row[0]);
        $name = htmlspecialchars ($row[1]);
        print ("<option value=\"$abbrev\">$name</option>\n");
    }
    $result->free ();
}
print ("</select>\n");
```

在Python中，是这样的：

```
stmt = "SELECT abbrev, name FROM states ORDER BY name"
cursor = conn.cursor ()
cursor.execute (stmt)
print "<select name=\"state\">"
for (abbrev, name) in cursor.fetchall ():
```

```
abbrev = cgi.escape (abbrev, 1)
name = cgi.escape (name, 1)
print "<option value=\"%s\">>%s</option>" % (abbrev, name)
cursor.close ()
print "</select>"
```

radio按钮和滚动列表也能用类似的方式生成。但与其这么做，不如让我们来尝试一种不同的方法，并构建一组在给定合适信息后能生成表单元素的函数。这些函数的返回值是一个字符串。如下所示：

```
make_radio_group (name, values, labels, default, vertical)
make_popup_menu (name, values, labels, default)
make_scrolling_list (name, values, labels, default, size, multiple)
```

这些函数有几个共同的参数：

name

表单的名字。

values

表单中所有元素对应的一组值（一个数组）。

labels

另外一个数组，它为每一个值提供了对应的用于显示的标语。这两个数组必需大小一样（如果你想直接将值作为标语，那么就将相同的数组给函数传递两次）。

default

表单的默认值。这应该是一个标量值，除非使用make_scrolling_list()函数。我们会编写一个函数来处理单取或多取列表，所以它的默认值可以是一个标量或者是一个数组。如果没有默认值，那就传给表单一个它并不包含的值，最典型的做法是传一个空字符串。

有些函数还提供额外的参数，这些参数仅适用于某些表单类型：

vertical

这仅用于radio按钮组。如果是true，它表明所有元素都应该按照垂直的方向入栈，而不是水平方向。

size, multiple

这些参数提供给滚动列表。size表示在列表中有多少个元素是可见的，multiple是true时表明列表允许多选。

这些列表生成函数的某些执行，我们将在那里讨论。同时你还可以在光盘的lib目录下找到所有函数的代码。所有函数都和CGI.pm中的表单生成函数类似，特别是在自动执行HTML编码参数这一点上。（Ruby环境下的库函数包含了辅助方法以生成这些元素，即使是在它的

cgi模块已经有方法能做到的情况下。所以我猜想辅助方法会比cgi模块中的方法好用些)。

在PHP中，make_radio_group()函数创建一组radio按钮：

```
function make_radio_group ($name, $values, $labels, $default, $vertical)
{
    $str = "";
    for ($i = 0; $i < count ($values); $i++)
    {
        # 选择和默认值对应的项
        $checked = ($values[$i] == $default ? " checked=\"checked\" " : "");
        $str .= sprintf (
            "<input type=\"radio\" name=\"%s\" value=\"%s\" %s />%s",
            htmlspecialchars ($name),
            htmlspecialchars ($values[$i]),
            $checked,
            htmlspecialchars ($labels[$i]));
        if ($vertical)
            $str .= "<br />"; # 垂直显示这些项
        $str .= "\n";
    }
    return ($str);
}
```

上述函数构建表单，并返回一个字符串。要使用make_radio_group()函数来展现颜色列表，应在从cow_color中获取信息之后调用它：

```
$values = array ();
$stmt = "SELECT color FROM cow_color ORDER BY color";
$result =& $conn->query ($stmt);
if (!PEAR::isError ($result))
{
    while ($row =& $result->fetchRow ())
        $values[] = $row[0];
    $result->free ();
}
print (make_radio_group ("color", $values, $values, "", TRUE));
```

\$values数组传递给make_radio_group()函数两次，是因为它既被当作值又被当作标语。

如果你想知道如何展示一个弹出式菜单，请使用下面的函数：

```
function make_popup_menu ($name, $values, $labels, $default)
{
    $str = "";
    for ($i = 0; $i < count ($values); $i++)
    {
        # 选择和默认值对应的项
        $checked = ($values[$i] == $default ? " selected=\"selected\" " : "");
        $str .= sprintf (
            "<option value=\"%s\"%s>%s</option>\n",
            htmlspecialchars ($values[$i]),
            $checked,
```

```

        $checked,
        htmlspecialchars ($labels[$i]));
    }
$str = sprintf (
    "<select name=\"%s\">\n%s</select>\n",
    htmlspecialchars ($name),
    $str);
return ($str);
}

```

函数make_popup_menu()没有\$vertical参数，但你可以像调用make_radio_group()一样使用它：

```
print (make_popup_menu ("color", $values, $values, ""));
```

函数make_scrolling_list()跟make_popup_menu()是非常相似的，但这里我不会给出它们的执行代码。调用该函数以生成一个单取列表，传送相同的变量给make_popup_menu()，但标明有多少行可见，添加一个为FALSE的multiple参数：

```
print (make_scrolling_list ("color", $values, $values, "", 3, FALSE));
```

州列表使用的label跟它的值不同：

```

$values = array ();
$labels = array ();
$stmt = "SELECT abbrev, name FROM states ORDER BY name";
$result =& $conn->query ($stmt);
if (!PEAR::isError ($result))
{
    while ($row =& $result->fetchRow ())
    {
        $values[] = $row[0];
        $labels[] = $row[1];
    }
    $result->free ();
}

```

然后使用值和label数组去生成你想要的列表：

```
print (make_popup_menu ("state", $values, $labels, ""));
print (make_scrolling_list ("state", $values, $labels, "", 6, FALSE));
```

Ruby和Python中，辅助函数的执行跟PHP是相似的。例如，Python版本的make_popup_menu()函数看起来是这样：

```

def make_popup_menu (name, values, labels, default):
    result_str = ""
    # 确保名字和默认值是字符串
    name = str (name)
    default = str (default)
    for i in range (len (values)):
        # 确保名字和默认值是字符串
        value = str (values[i])
        label = str (labels[i])

```

```

# 选择和默认值对应的项
if value == default:
    checked = " selected=\"selected\""
else:
    checked = ""
result_str = result_str + \
    "<option value=\"%s\"%s>%s</option>\n" \
    % (cgi.escape (value, 1),
       checked,
       cgi.escape (label, 1))

result_str = "<select name=\"%s\">\n%s</select>\n" \
    % (cgi.escape (name, 1), result_str)
return result_str

```

要将牛的颜色数组显示在一个表单中，可以这样获取它们：

```

values = []
stmt = "SELECT color FROM cow_color ORDER BY color"
cursor = conn.cursor ()
cursor.execute (stmt)
for (color, ) in cursor.fetchall ():
    values.append (color)
cursor.close ()

```

然后使用下面调用之一将列表转化为一个表单：

```

print make_radio_group ("color", values, values, "", True)

print make_popup_menu ("color", values, values, "")

print make_scrolling_list ("color", values, values, "", 3, False)

```

要展示州列表，可如下获取名字和缩写：

```

values = []
labels = []
stmt = "SELECT abbrev, name FROM states ORDER BY name"
cursor = conn.cursor ()
cursor.execute (stmt)
for (abbrev, name) in cursor.fetchall ():
    values.append (abbrev)
    labels.append (name)
cursor.close ()

```

并将它们传递给合适的函数：

```

print make_popup_menu ("state", values, labels, "")

print make_scrolling_list ("state", values, labels, "", 6, False)

```

在lib目录中的Ruby和Python辅助方法所执行的某些操作，在PHP中的对应函数是不做的，只是显式地将列表中的值转化为字符串形式。这是非常必需的，因为Ruby的CGI.escape-HTML()和Python的cgi.escape()方法在你使用它们去HTML编码非空字符串时，会触发例外处理。

我们已经思考如何从cow_color和states表中获取每一行数据，并将它们转化为表单元素很久了。另外一个需要成为在线应用表单中一部分的元素是牛雕像尺寸域。这个域的合法值来自于cow_order表格中size列的定义。该列是ENUM类型的，所以为表单元素获得它的合法值就是一个获得列定义并对其进行解析的问题。也就是我们需要使用列元数据，而不是列数据。

与此同时，本任务中的许多工作都已经在9.7节中涉及，9.7节提供了辅助程序以获得ENUM和SET列的元数据。例如，在Perl中，调用get_enumorset_info()函数就能获得size列的元数据：

```
my $size_info = get_enumorset_info ($dbh, "cookbook", "cow_order", "size");
```

结果的\$size_info值是一个指向hash表的索引，它有几个成员，其中两个跟我们的目的相关：

```
$size_info->{values}  
$size_info->{default}
```

这里，values成员是一个指向合法枚举数值的索引，default的是列的默认值。这些信息数据使用了一种能直接转化为表单的格式：

```
print radio_group (-name => "size",  
                  -values => $size_info->{values},  
                  -default => $size_info->{default},  
                  -linebreak => 1); # 垂直显示按钮  
  
print popup_menu (-name => "size",  
                  -values => $size_info->{values},  
                  -default => $size_info->{default});
```

默认的值是medium，这是当浏览器显示表单时会默认选中的值。

同样功能的Ruby元数据获取方法返回的是一个hash表。如下使用这个hash来生成表单：

```
size_info = get_enumorset_info(dbh, "cookbook", "cow_order", "size")  
  
form << make_radio_group("size",  
                           size_info["values"],  
                           size_info["values"],  
                           size_info["default"],  
                           true) # 垂直显示这些项  
  
form << make_popup_menu("size",  
                           size_info["values"],  
                           size_info["values"],  
                           size_info["default"])
```

PHP的元数据函数返回一个关联数组，该数组将被如下使用：

```
$size_info = get_enumorset_info ($conn, "cookbook", "cow_order", "size");

print (make_radio_group ("size",
    $size_info["values"],
    $size_info["values"],
    $size_info["default"],
    TRUE)); # 垂直显示这些项

print (make_popup_menu ("size",
    $size_info["values"],
    $size_info["values"],
    $size_info["default"]));
```

Python版本的元数据函数返回一个字典：

```
size_info = get_enumorset_info (conn, "cookbook", "cow_order", "size")

print make_radio_group ("size",
    size_info["values"],
    size_info["values"],
    size_info["default"],
    True) # 垂直显示这些项

print make_popup_menu ("size",
    size_info["values"],
    size_info["values"],
    size_info["default"])
```

当你像这样使用ENUM值来创建列表时，数值以它们在列定义中的顺序依次显示。表中的size列定义规定了数值的顺序（small、medium、large），但对于那些你想要不同枚举顺序的列，你得把其中的值排好序。

要示范如何处理列元数据并在JSP中生成表单元素，我计划在页面中嵌入一个函数。更好的方法是在标签库中编写一个定制的动作，该动作影射到一个能返回信息的类上，但如何编写一个定制的标签已经超出了本书的范围。下面的示例采用了三个步骤：

1. 使用JSTL标签去查询INFORMATION_SCHEMA以获得ENUM列的定义，然后把定义移动到页面的上下文中。
2. 调用一个函数，从页面上下文中抽取定义，分析这个定义，并把它存入一组单独的枚举值中，然后把该组数值移回页面上下文。
3. 使用JSTL的迭代器访问该数组，并显示列表中的每一个数值。在处理每个数值时，将它与列的默认值比较，如果相同的话就把它标记为初始选中的元素。

从ENUM或SET列定义中抽取合法数值的函数，取名叫getEnumOrSetValues()。你可以像下面这样把它放入JSP页面中：

```
<%@ page import="java.util.*" %>
<%@ page import="java.util.regex.*" %>

<%
// 定义一个类方法来分解ENUM和SET类型的值。
// typeDefAttr - 包含列类型定义的页面上下文属性名
// valListAttr - 用来存放列值list的页面上下文属性名

void getEnumOrSetValues (PageContext ctx,
                        String typeDefAttr,
                        String valListAttr)
{
    String typeDef = ctx.getAttribute (typeDefAttr).toString ();
    List values = new ArrayList ();

    // 列必须是 ENUM 或者 SET 类型
    Pattern pc = Pattern.compile ("(enum|set)\\((.*?)\\)");
    Pattern.CASE_INSENSITIVE);
    Matcher m = pc.matcher (typeDef);
    // matches() 方法失败除非匹配整个字符串
    if (m.matches ())
    {
        // split value list on commas, trim quotes from end of each word
        String[] v = m.group (2).split ",";
        for (int i = 0; i < v.length; i++)
            values.add (v[i].substring (1, v[i].length() - 1));
    }
    ctx.setAttribute (valListAttr, values);
}

%>
```

上述函数有三个参数：

ctx

页面上下文对象。

typeDefAttr

包含了列定义的页面元素的名称。这是函数的“输入”。

valListAttr

页面属性的名字，该属性中能放入合法列值的结果数组，这是函数的“输出”。

从size列中生成一个列表，可以先获取列的元数据，抽取值数组并放入JSTL中的名为values的变量中，同时默认值放入名为default的变量中：

```

<sql:query dataSource="${conn}" var="rs">
    SELECT COLUMN_TYPE, COLUMN_DEFAULT
    FROM INFORMATION_SCHEMA.COLUMNS
    WHERE TABLE_SCHEMA = 'cookbook' AND TABLE_NAME = 'cow_order'
    AND COLUMN_NAME = 'size'
</sql:query>
<c:set var="typeDef" scope="page" value="${rs.rowsByIndex[0][0]}"/>
<% getEnumOrSetValues (pageContext, "typeDef", "values"); %>
<c:set var="default" scope="page" value="${rs.rowsByIndex[0][1]}"/>

```

接下来用这个值的List和默认值来构造一个form元素。例如，生成一组如下所示的单选按钮：

```

Then use the value list and default value to construct a form element.
For example, produce a set of radio buttons like this:
<c:forEach items="${values}" var="val">
    <input type="radio" name="size"
        value=<c:out value="${val}" />
        <c:if test="${val == default}">checked="checked"</c:if>
    /><c:out value="${val}" /><br />
</c:forEach>

```

或者一个如下所示的浮动菜单：

```

<select name="size">
    <c:forEach items="${values}" var="val">
        <option
            value=<c:out value="${val}" />
            <c:if test="${val == default}">selected="selected"</c:if>
        >
        <c:out value="${val}" /></option>
    </c:forEach>
</select>

```

别忘了将表单中所有列表的内容都HTML编码

在光盘中介绍的Ruby、PHP和Python的辅助应用有生成列表元素，执行HTML编码，构建HTML标签等。它们还能编码标语。我注意到许多公布的列表生成并不做这方面的工作，或者它们编码标语而不编码数值。这是错误的。如果标语或者数值中包含一个特殊字符比如&或<，浏览器就可能错误地解释它们，你的应用也可能会因此而出错。同样重要的事情还有，确保你的编码函数将双引号转化为“实体（或等价的“）”，因为标签属性是经常被包含在双引号中的。

如果你正在使用Perl的CGI.pm模块或者JSTL标签去生成HTML的表单，编码会为你省下很多烦恼。CGI.pm的表单相关函数会自动执行编码。类似地，使用JSTL的<c:out>标签来编写JSP页面中的属性值，将生成恰当编码的值。

这儿讨论的列表生成方法并不绑定到任何一个特定的数据库表，所以它们能用来为所有类型的数据生成列表元素，而不仅仅是在本示例场景中用到的这些。例如，要允许用户在数

数据库管理应用中选择一个表名，你可以生成一个滚动列表，数据库中每一个表格在该列表中都有一个对应的元素。一个基于CGI.pm的脚本看起来会像这样：

```
my $stmt = "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
           WHERE TABLE_SCHEMA = 'cookbook' ORDER BY TABLE_NAME";
my $table_ref = $dbh->selectcol_arrayref ($stmt);
print scrolling_list (-name => "table",
                      -values => $table_ref,
                      -size => 10);      # 每次显示10项
```

查询结果不必跟数据库表格有任何关联。例如，如果你想在JSP页面中提供一个列表，该列表包含了一个指向最后七天的入口，你可以使用下面语句计算日期：

```
<sql:query dataSource="${conn}" var="rs">
```

```
SELECT
    CURDATE() - INTERVAL 6 DAY,
    CURDATE() - INTERVAL 5 DAY,
    CURDATE() - INTERVAL 4 DAY,
    CURDATE() - INTERVAL 3 DAY,
    CURDATE() - INTERVAL 2 DAY,
    CURDATE() - INTERVAL 1 DAY,
    CURDATE()
```

```
</sql:query>
```

然后使用日期来生成一个列表：

```
<c:set var="dateList" value="${rs.rowsByIndex[0]}" />
<c:forEach items="${dateList}" var="date">
    <input type="radio" name="date"
          value=<c:out value="${date}" />>
    /><c:out value="${date}" /><br />
</c:forEach>
```

当然，如果你的编程语言能够合理执行日期计算，那么在客户端生成这个日期列表将显得更为高效，而不是像现在需要往MySQL发送一个语句。

19.3 根据数据库内容构建多取表单元素

Creating Multiple-Pick Form Elements from Database Content

问题

表单需要呈现一个域，该域提供了几种选项并允许用户选择任意数量的选项。

解决方案

使用一个多选列表，比如一组复选框或一个滚动列表。

讨论

一个多取表单允许你呈现多个选项，任意数量的选项都可以被选中，或一个都不选。在我们的示例场景中，客户在线订购牛雕像，多取元素是通过一组可用附件呈现出来的。在 cow_order 表格中的 accessory 列是 SET 类型，所以允许的值和默认值可以通过如下语句获得：

```
mysql> SELECT COLUMN_TYPE, COLUMN_DEFAULT FROM INFORMATION_SCHEMA.COLUMNS
-> WHERE TABLE_SCHEMA='cookbook' AND TABLE_NAME='cow_order'
-> AND COLUMN_NAME='accessories';
+-----+-----+
| COLUMN_TYPE | COLUMN_DEFAULT |
+-----+-----+
| set('cow bell','horns','nose ring','tail ribbon') | cow bell,horns |
+-----+-----+
```

在定义中列出的值能非常合理地被呈现为一组复选框或多取滚动列表。这两种方法中的任一种，cow bell 和 horns 都应该被初始选中，因为它们是列的默认值。下面讨论这几个元素的 HTML 语法，然后描述如何在脚本中生成它们。



提示：在本节中的材料非常依赖 19.2 节，那里讨论了 radio 按钮、弹出菜单、单取滚动列表。下面假设你已经阅读了该章节。

复选框

一组复选框跟一组 radio 按钮很相似，比如它们都包含了有相同 name 的 <input> 元素。然而，该元素的 type 属性是 checkbox 而非 radio。同时，在默认情况下，你还可以将多个元素设置为选中。如果没有元素被标记为选中，那么初始时就一个都不会被选中。下面的复选框组显示了 cow 的附件元素，且头两个元素被默认选中：

```
<input type="checkbox" name="accessories" value="cow bell"
       checked="checked" />cow bell
<input type="checkbox" name="accessories" value="horns"
       checked="checked" />horns
<input type="checkbox" name="accessories" value="nose ring" />nose ring
<input type="checkbox" name="accessories" value="tail ribbon" />tail ribbon
```

滚动列表

多取滚动列表跟单取滚动列表在语法方面，有很多相同点。区别在于你在开始标签 <select> 中包含了 multiple 属性，并且有不同的默认值。对于一个单取列表，你可以添加 selected 属性，但最多为一个元素，如果你不提供 selected 的话，第一个元素就

默认被选中。对于一个多取列表而言，你可以在任意数量的元素中添加selected属性，也可以一个都不添加。

如果cow附件信息被显示为一个多取滚动列表，且cow bell和horns被初始选中，那么相关的代码就是这样：

```
<select name="accessories" size="3" multiple="multiple">
<option value="cow bell" selected="selected">cow bell</option>
<option value="horns" selected="selected">horns</option>
<option value="nose ring">nose ring</option>
<option value="tail ribbon">tail ribbon</option>
</select>
```

在基于CGI.pm的Perl脚本中，你调用checkbox_group()或scrolling_list()来创建复选框或滚动列表。这两个函数需要姓名、值、标语和默认参数，这就像你在使用单取列表时一样。当多个元素可以被初始选中时，CGI.pm允许默认的参数被设置为一个标量值或一个指向数组的索引。

要获得一个SET列的合法值，我们在19.2节中对ENUM做过相同的事情，那就是调用一个辅助应用，该应用返回列的元数据：

```
my $acc_info = get_enumset_info ($dbh, "cookbook", "cow_order", "accessories");
```

然而，如果一个表单是直接用于表单生成的话，SET列的默认值是不在该表单中的。MySQL将SET默认值表示为一列零或更多的元素，用逗号分开。例如accessories列的默认值是cow bell和horns。这并不符合CGI.pm函数期望的列表数值格式，所以有必要将默认值用逗号分开以组成一个数组。下面的表达式显示了如何做到这点：

```
my @acc_def = (defined ($acc_info->{default}))
    ? split (/, /, $acc_info->{default})
    : ();
```

在将默认值分割开后，把结果数组传递给任一生成列表的函数：

```
print checkbox_group (-name => "accessories",
    -values => $acc_info->{values},
    -default => \@acc_def,
    -linebreak => 1);    # 垂直显示按钮

print scrolling_list (-name => "accessories",
    -values => $acc_info->{values},
    -default => \@acc_def,
    -size => 3,          # 每次显示3项
    -multiple => 1);    # 创建一个多选单
```

当你像这样使用SET值去创建列表时，数值以它们在列定义中给定的顺序显示。这可能跟你想要他们出现的顺序不一样，如果确实不同，那就重新排序。

在Ruby、PHP和Python中，我们能够创建辅助函数来生成多选元素，这些函数将遵循以下调用语法：

```
make_checkbox_group (name, values, labels, default, vertical)
make_scrolling_list (name, values, labels, default, size, multiple)
```

这里name、values、labels参数跟19.2节单取辅助函数中定义的用法是一样的。函数make_checkbox_group()还需要一个vertical参数来确定其中的元素该按垂直顺序入栈，还是按水平顺序入栈。函数make_scrolling_list()已经在19.2节中描述了。在这儿是使用它，multiple参数就应该是设为true。对这两个函数而言，如果初始选定多个元素，那么默认的参数应该是一个多值数组。

函数make_checkbox_group()看起来应如下使用（这儿用的是Ruby）：

```
def make_checkbox_group(name, values, labels, default, vertical)
  # 确保默认值是一个数组（将标量转化为数组）
  default = [ default ].flatten
  str = ""
  for i in 0...values.length do
    # 选择和这些默认值其中之一相等的项
    checked = (default.include?(values[i])) ? " checked=\"checked\"" : ""
    str << sprintf(
      "<input type=\"checkbox\" name=\"%s\" value=\"%s\"%s
      />%s",
      CGI.escapeHTML(name.to_s),
      CGI.escapeHTML(values[i].to_s),
      checked,
      CGI.escapeHTML(labels[i].to_s))
    str << "<br />" if vertical  # 垂直显示这些项
    str << "\n"
  end
  return str
end
```

要获取牛的附件信息并使用复选框把它们呈现出来，就这样：

```
acc_info = get_enumset_info(dbh, "cookbook", "cow_order", "accessories")
if acc_info["default"].nil?
  acc_def = []
else
  acc_def = acc_info["default"].split(",")
end

form << make_checkbox_group("accessories",
  acc_info["values"],
  acc_info["values"],
  acc_def,
  true)      # 垂直显示这些项
```

如果想使用滚动列表来显示，就调用make_scrolling_list()函数：

```
form << make_scrolling_list("accessories",
    acc_info["values"],
    acc_info["values"],
    acc_def,
    3,          # 每次显示3项
    true)       # 创建一个多选单
```

在PHP中，可以这样获取辅助信息，并用复选框或滚动列表显示出来：

```
$acc_info = get_enumorset_info ($conn, "cookbook", "cow_order", "accessories");
$acc_def = explode (",", $acc_info["default"]);

print (make_checkbox_group ("accessories[]",
    $acc_info["values"],
    $acc_info["values"],
    $acc_def,
    TRUE));   # 垂直显示这些项

print (make_scrolling_list ("accessories[]",
    $acc_info["values"],
    $acc_info["values"],
    $acc_def,
    3,          # 每次显示3项
    TRUE));   # 创建一个多选单
```

需要注意的是在PHP例子中用到的域名，被指定为accessories[]而不是accessories。在PHP中，你必须在允许多个数值的域名后添加[]。如果你忽视了[]，用户在填写表单时还是能够选择多个元素，而PHP却只会返回其中的一个脚本给你。当我们在19.5节中讨论如何处理已提交表单内容时，还将讨论这个问题。

在Python中，获取牛的附件信息并使用复选框或滚动列表呈现，可以这样：

```
acc_info = get_enumorset_info (conn, "cookbook", "cow_order", "accessories")
if acc_info["default"] == None:
    acc_def = ""
else:
    acc_def = acc_info["default"].split (",", "")

print make_checkbox_group ("accessories",
    acc_info["values"],
    acc_info["values"],
    acc_def,
    True)   # 垂直显示这些项

print make_scrolling_list ("accessories",
    acc_info["values"],
    acc_info["values"],
    acc_def,
```

```
3,          # 每次显示3项  
True)    # 创建一个多选单
```

在JSP页面中，早先使用`getEnumOrSetValues()`函数来获取size列的值组，该函数也能用在accessory列上。列的定义和默认值可以使用INFORMATION_SCHEMA来获得。查询COLUMNS表，解析类型定义并存入一个名为values的数组中，同时把默认值放入defList，如下所示：

```
<sql:query dataSource="${conn}" var="rs">  
    SELECT COLUMN_TYPE, COLUMN_DEFAULT  
    FROM INFORMATION_SCHEMA.COLUMNS  
    WHERE TABLE_SCHEMA = 'cookbook'  
    AND TABLE_NAME = 'cow_order'  
    AND COLUMN_NAME = 'accessories'  
</sql:query>  
<c:set var="typeDef" scope="page" value="${rs.rowsByIndex[0][0]}"/>  
<% getEnumOrSetValues (pageContext, "typeDef", "values"); %>  
<c:set var="defList" scope="page" value="${rs.rowsByIndex[0][1]}"/>
```

对一个SET列而言，defList值也许包含多个以逗号分隔的值。它不需要特殊的处理，JSTL的`<c:forEach>`标签能迭代遍历这个字符串，所以我们可如下初始化一组复选框的默认值：

```
<c:forEach items="${values}" var="val">  
    <input type="checkbox" name="accessories"  
        value=<c:out value="${val}"/>"  
    <c:forEach items="${defList}" var="default">  
        <c:if test="${val == default}">checked="checked"</c:if>  
    </c:forEach>  
    /><c:out value="${val}"/><br />  
</c:forEach>
```

换成多取滚动列表，可以这样做：

```
<select name="accessories" size="3" multiple="multiple">  
<c:forEach items="${values}" var="val">  
    <option  
        value=<c:out value="${val}"/>"  
        <c:forEach items="${defList}" var="default">  
            <c:if test="${val == default}">selected="selected"</c:if>  
        </c:forEach>  
    >  
    <c:out value="${val}"/></option>  
</c:forEach>  
</select>
```

19.4 将一条数据库记录导入表单

Loading a Database Record into a Form

问题

你希望显示一个表单，且使用一条数据库记录的内容来初始化这个表单。这允许你呈现一个记录——编辑的表单。

解决方案

像你平常使用的那样生成表单，但把数据库内容装进去。这意味着，不是把表单元素的默认值设置为它们的常规值，而是把它们设置为数据库中的记录。

讨论

光盘中早先的例子示范了如何在没有默认值的情况下，或者使用ENUM或SET列定义的默认值，生成表单的域。这非常适合构建一个“空白的”表单，且你期望用户填写该表单。然而，对那些基于Web界面来编辑记录的应用而言，更常见的情况可能是你使用数据库中已有记录的内容来填写表单的初始值。本节将详细讨论如何做到这点。

下面的例子示范了如何为cow_order表格的各行生成一个编辑表单。一般情况下，你会允许用户选择要编辑的行。为简单起见，假设使用id为1的记录：

```
mysql> SELECT * FROM cow_order WHERE id = 1\G
***** 1. row *****
    id: 1
  color: Black & White
    size: large
accessories: cow bell,nose ring
  cust_name: Farmer Brown
cust_street: 123 Elm St.
  cust_city: Katy
  cust_state: TX
```

要生成一个表单，且该表单的内容对应到数据库的一个记录，如下使用列的值：

- 对<input>元素比如radio按钮或复选框，添加一个checked属性到每个列表元素中。
- 对<select>元素比如弹出菜单或滚动列表，添加一个selected属性到每个列表元素。
- 对用来输入文本的<input>元素类型的文本域，设置value属性为对应的列值。例如，一个60字符的用于cust_name的域可以初始化显示为Farmer Brown：

```
<input type="text" name="cust_name" value="Farmer Brown" size="60" />
```

如果要呈现一个<textarea>元素，设置body为列的值。如果要创建一个40列宽3行高的区域，代码可以这样写：

```
<textarea name="cust_name" cols="40" rows="3">
Farmer Brown
</textarea>
```

- 在一个记录编辑的环境下，最好在表单中包含一个唯一的值，因为这样你就能在用户提交表单时，辨别当前表单内容对应到哪一条记录。使用一个隐藏的域来做这件事，它的值不会显示给用户，但浏览器会在把它跟其他域值一起返回。我们的示例记录有一个id列，值为1，所以隐藏域看起来就像这样：

```
<input type="hidden" name="id" value="1" />
```

下面的例子展示了如何生成一个表单，该表单有一个隐藏的id域、弹出菜单的颜色域、radio按钮的尺寸域，以及复选框的附件域。客户信息被显示在文本输入框中，另外cust_state是一个单取滚动列表。当然，你可以做其他选择，比如将尺寸域用弹出菜单而不是radio按钮来呈现。

本例子对应的光盘中的脚本，名为cow_edit.pl、cow_edit.jsp等。

接下来的过程描述了如何将示例cow_table的记录导入编辑表格中，这里使用的是基于CGI.pm的Perl脚本：

1. 获取该记录的列值：

```
my $id = 1;      # 选择记录号1
my ($color, $size, $accessories,
    $cust_name, $cust_street, $cust_city, $cust_state) =
$dbh->selectrow_array (
    "SELECT
        color, size, accessories,
        cust_name, cust_street, cust_city, cust_state
    FROM cow_order WHERE id = ?",
    undef, $id);
```

2. 开始这个表单：

```
print start_form (-action => url ());
```

3. 生成隐藏域，该域包含唯一确定cow_order记录的id值：

```
print hidden (-name => "id", -value => $id, -override => 1);
```

这里，override参数强制CGI.pm使用在value参数中设置的值，并把这个值作为隐藏域的值。如果override不是true，CGI.pm一般会尝试去使用脚本执行环境中的值来初始化表单的域，即使你在表单生成调用中提供了值。

4. 创建描述牛雕像规格的各个域，它们采用了19.2节和19.3节中相同的方法来生成，唯一的区别是默认值来自于记录1。这里的代码将color显示为弹出菜单，size表示为Radio按钮，accessories表示为一组复选框。需要注意的是，它将accessories的值用逗号分隔开来以产生一组值，因为它可能存在多个附件元素：

```
my $color_ref = $dbh->selectcol_arrayref (
    "SELECT color FROM cow_color ORDER BY color");

print br (), "Cow color:", br ();
print popup_menu (-name => "color",
    -values => $color_ref,
    -default => $color,
    -override => 1);

my $size_info = get_enumorset_info ($dbh, "cookbook",
    "cow_order", "size");

print br (), "Cow figurine size:", br ();
print radio_group (-name => "size",
    -values => $size_info->{values},
    -default => $size,
    -override => 1,
    -linebreak => 1);

my $acc_info = get_enumorset_info ($dbh, "cookbook",
    "cow_order", "accessories");
my @acc_val = (defined ($accessories)
    ? split (/, /, $accessories)
    : ());

print br (), "Cow accessory items:", br ();
print checkbox_group (-name => "accessories",
    -values => $acc_info->{values},
    -default => \@acc_val,
    -override => 1,
    -linebreak => 1);
```

5. 创建客户信息域。除了使用单取滚动列表的州信息外，这些域用文本输入框来表示：

```
print br (), "Customer name:", br ();
print textfield (-name => "cust_name",
    -value => $cust_name,
    -override => 1,
    -size => 60);

print br (), "Customer street address:", br ();
print textfield (-name => "cust_street",
    -value => $cust_street,
    -override => 1,
    -size => 60);

print br (), "Customer city:", br();
```

```

printtextfield (-name => "cust_city",
               -value => $cust_city,
               -override => 1,
               -size => 60);

my @state_values;
my %state_labels;
my $sth = $dbh->prepare ("SELECT abbrev, name
                           FROM states ORDER BY name");
$sth->execute ();
while (my ($abbrev, $name) = $sth->fetchrow_array ())
{
    push (@state_values, $abbrev); # 将各个值保存到数组中
    $state_labels{$abbrev} = $name; # 将各值映射到各自的标签中
}

print br (), "Customer state:", br ();
print scrolling_list (-name => "cust_state",
                      -values => \@state_values,
                      -labels => \%state_labels,
                      -default => $cust_state,
                      -override => 1,
                      -size => 6); # 一次显示6项

```

6. 创建一个表单的提交按钮，并结束该表单：

```

print br (),
      submit (-name => "choice", -value => "Submit Form"),
      end_form ();

```

相同的过程还可应用到其他API中。例如，在一个JSP页面中，你可以获取记录以编辑，或者抽取其中的内容并放到标量变量中：

```

<c:set var="id" value="1"/>
<sql:query dataSource="${conn}" var="rs">
  SELECT
    id, color, size, accessories,
    cust_name, cust_street, cust_city, cust_state
  FROM cow_order WHERE id = ?
  <sql:param value="${id}" />
</sql:query>

<c:set var="row" value="${rs.rows[0]}" />
<c:set var="id" value="${row.id}" />
<c:set var="color" value="${row.color}" />
<c:set var="size" value="${row.size}" />
<c:set var="accessories" value="${row.accessories}" />
<c:set var="cust_name" value="${row.cust_name}" />
<c:set var="cust_street" value="${row.cust_street}" />
<c:set var="cust_city" value="${row.cust_city}" />
<c:set var="cust_state" value="${row.cust_state}" />

```

然后使用值去初始化各种各样的表单元素：

- 对应到ID的隐藏域：

```
<input type="hidden" name="id" value=<c:out value="${id}" /> />
```

- 颜色相关的弹出菜单：

```
<sql:query dataSource="${conn}" var="rs">
    SELECT color FROM cow_color ORDER BY color
</sql:query>
<br />Cow color:<br />
<select name="color">
<c:forEach items="${rs.rows}" var="row">
    <option
        value=<c:out value="${row.color}" />">
        <c:if test="${row.color == color}">selected="selected"</c:if>
    <c:out value="${row.color}" /></option>
</c:forEach>
</select>
```

- 文本域cust_name：

```
<br />Customer name:<br />
<input type="text" name="cust_name"
    value=<c:out value="${cust_name}" />">
    size="60" />
```

对Ruby、PHP或Python而言，使用19.2节和19.3节提到的辅助函数来创建表单。你可以浏览cow_edit.rb、cowe_edit.php、cow_edit.py脚本以了解细节。

19.5 收集Web输入

Collecting Web Input

问题

你希望从已被提交的表单或URL的末尾处抽取输入参数。

解决方案

使用你的API，它提供了在Web脚本执行环境中访问输入参数中名字和值的方法。

讨论

本章中早先，讨论了如何从MySQL中获取信息并生成各种各样的输出，比如静态文本、列表、超链接还有表单。在本节中，我们讨论反方向的一个问题：如何从Web中收集输入。这样的应用是很多的。例如，你可以使用这里的技术抽取用户提交表单的内容。你也许将该内容解释为一组搜索关键字，然后在一个产品目录上执行查询，并将匹配的结果显示给用户。在这种情况下，你使用Web来收集信息，并据此判断用户的兴趣，然后你构建一个恰

当的查询语句，并最终显示查询结果。如果一个表单是一份调查问卷、一个邮件列表的注册单或一个选举，你或许需要把值存储下来，并使用这些数据来创建一条新的数据库记录（或更新一条已有的记录）。

从Web中获得输入并据此跟MySQL交互的脚本一般按以下步骤处理信息：

1. 从执行环境中获取输入。当一个包含输入参数的请求到来时，Web服务器把输入放入处理该输入的脚本的环境中，脚本可以查询它的环境以获取参数。如果你API提供的获取机制不支持字符解码，那么把输入中特殊字符做HTML编码以获取最初的字符是非常必要的（比如将%20转化为空格）。
2. 验证输入以确保它是合法的。你不能相信用户总是输入合法的数据，所以应该检查输入参数以确保它们的合法性。例如，如果你期望用户输入一个数字，那么你就应该检查相应的值以确定它真的是个数字。如果表单中包含一个弹出菜单，该菜单使用ENUM列允许的值来构建，你或许期望返回的值正好就是允许的值之一。但是，只有当检查后你才能确定这些。记住：你甚至不知道在网络另一头的是否是一个真正的用户。它说不定是段恶意的脚本，该脚本不停地在你的表单处理代码中查找你的系统漏洞。

如果你不检查输入，就等于是冒着往数据库中存入垃圾数据，或破坏已有数据的风险。你确实能够使用严格的SQL模式，阻止那些不符合列定义的非法数据输入你的数据库。然而，当你的应用需要判别什么是合法时，上述作法会带来额外的文法限制，那么在你尝试把数据存入时还是有必要先在脚本中检查数据有效性的。而且，在你的脚本中执行检查，你可以给用户提供更有意义的错误信息，而不是MySQL返回的那些用户难懂的信息。基于这两个原因，最好把严格的SQL模式当成有价值的额外保护，但这个保护本身是不够的。这意味着，你需要把服务器端的保护和客户端的检验结合起来。请查阅10.20节以了解如何设置严格的SQL模式。

3. 基于输入构建一个语句。典型的情况是，输入参数被用来在数据库中添加一条记录，或者从数据库中获取信息并显示给客户。在这两种情况下，你使用输入来构建一个语句并把它发送给MySQL服务器。语句的构建过程是基于用户输入的，所以应该非常小心，记得使用恰当的技巧以避免生成错误的或者有危险的SQL语句。使用占位符是个好办法。

接下来我们先介绍输入流程的第一步。19.6节和19.7节将介绍第二、三步的内容。第一个步骤（将输入从执行环境中抽取出来）跟MySQL是一点关系都没有的，我们在这里介绍它是因为后续的步骤都需要使用它获取的信息。

有几种方式可以从Web中获得输入，其中两种比较常见的方式为：

- 作为get请求的一部分，输入参数被添加到URL的末尾。例如，下面的URL调用了PHP脚本price_quote.php，并规定item和quantity参数为D-0214和60：

`http://localhost/mcb/price_quote.php?item=D-0214&quantity=60`

当用户选择一个超链接或者提交一个表单，该表单在`<form>`标签中设置了`method="get"`时，我们常接到这种请求：在URL中的一个参数列表以`?>`开始，并包含`name=value`这样的以`=`或`&`分隔的对。（将需要提交到服务器的信息放置在URL的中部也是也可以的，但是本书不讨论那种情况）

- 作为post请求的一部分，比如一个表单提交，该表单的`<form>`标签中设置了`method="post"`。post请求中该表单的内容被作为输入参数放入请求的主体中，而不是URL的末端。

你或许有机会去处理其他类型的输入，比如上传的文件。这些使用的是post请求，但是作为一种特殊表单的一部分。19.8节将讨论文件上传。

当你从Web脚本中获得输入时，你应思考这些输入是如何传送的。（有些API把get和post请求传来的输入区分开来，但有些不区分）然而，在你已经把发送的输入提取出来后，使用何种方法请求就无关紧要了。检验和语句构建阶段并不需要知道参数是通过get还是post请求传过来的。

光盘的apache/params目录（在JSP中是tomcat/mcb）下有一些脚本处理输入参数。每个脚本都允许你提交get或post请求，并展示如何提取和显示提交的参数。查看这些脚本可以了解不同API中的参数提取方法是如何被使用的。脚本中调用的辅助函数可以在光盘的lib目录下的库模块中找到。

Web输入提取惯例

要获得传送给脚本的输入参数，你应该让自己尽可能地熟悉你的API惯例，这样你就明白它为你做了什么，以及你须自己做什么。例如，你应该知道下面问题的答案：

- 如何判断哪个参数是可用的？

- 如何将一个参数的值从环境中提取出来？
- 这样获得的值就是用户提交的值吗？或者你需要解码这些值吗？
- 多值参数是怎么处理的？（例如，当一组中的几个复选框都被选中时。）
- 对于一个URL中提交的参数，你的API期望参数间的分隔符是什么？对于某些API而言是`&`，有些则倾向于`,`，因为它不是HTML中的特殊字符。但有许多浏览器或客户端代理总是使用`&`来分隔参数。如果你在脚本中构建一个URL，该URL在末尾包含参数，请确认使用了一个接受端脚本能够理解的参数分隔符。

Perl. Perl中的CGI.pm模块通过`param()`函数让参数对脚本是可用的。函数`param()`通过`get`或者`post`来访问输入，这样就简化了你作为脚本编写者的压力。你不必知道一个表单在提交时使用的是那种方法，你也不必执行任何解码。这些`param()`都为你做好了。

要获得可用参数的全部名字，调用`param()`但不输入参数：

```
@param_names = param();
```

要获得一个特定参数的值，把该参数的名字传给`param()`。在标量的上下文中，`param()`当参数为单值时返回参数值，为多值时返回第一个参数值，而当参数不可用时返回值`undef`。在数组的上下文中，`param()`返回一个包含了该参数所有值的列表，如果该参数不可用则返回一个空列表。

```
$id = param ("id");
$options = param ("options");
```

一个有给定名字的参数也许是不可用的，而这在对应的表单域为空时是确实存在的，当表单中根本没有对应的域时更是如此。需要注意的是，一个参数值可以是有定义且为空的。为了更好地权衡，你也许想要检查这两种情况。例如，检查一个`age`参数，并在参数丢失或为空时，赋予它一个`unknown`：

```
$age = param ("age");
$age = "unknown" if !defined ($age) || $age eq "";
```

CGI.pm知道`:`和`&`都是URL中的参数分隔符。

Ruby. Ruby脚本使用`cgi`模块，Web脚本参数通过你为生成HTML创建的`cgi`对象使用。它的`param`方法返回一个参数名和参数值的hash表，所以你能够访问这个hash或者如下获得参数名字：

```
params = cgi.params
param_names = cgi.params.keys
```

一个特定参数的对应值可通过参数名和参数值的hash表获得，也可直接访问cgi对象获得：

```
id = cgi.params["id"]
id = cgi["id"]
```

这两个访问方法稍微有点区别。params方法像数组一样返回每个参数值。如果参数有多个值，那么数组就包含多个入口，如果参数不存在，那么数组就是空的。cgi对象返回的是一个字符串。如果参数有多个值，那就只返回第一个，如果参数不存在，那就返回一个空字符串。这两种方法，都使用has_key()方法来测试一个参数是否存在。

下面的列表展示了如何获得参数名字，并循环以打印每个参数的名字和值，多值参数被打印为一个用逗号分隔的列表：

```
params = cgi.params
param_names = cgi.params.keys
param_names.sort!
page << cgi.p { "Parameter names: " + param_names.join(", ") }

list = ""
param_names.each do |name|
  val = params[name]
  list << cgi.li {
    "type=#{val.class}, name=#{name}, value=" +
    CGI.escapeHTML(val.join(", "))
  }
end
page << cgi.ul { list }
```

cgi模块知道;和&是URL的参数分隔符。

在PHP中，输入参数通过以下几种方法能获得，当然这取决于你的配置：

- 如果track_vars变量是开的，参数信息就在\$_HTTP_GET_VARS和\$_HTTP_POST_VARS数组中。例如，如果有表单包含一个id域，那么该域对应的值就是\$_HTTP_GET_VARS["id"]或者\$_HTTP_POST_VARS["id"]，具体使用哪个取决于该表单是用get还是用post方法提交的。如果你在一个非全局的作用域内访问\$_HTTP_GET_VARS和\$_HTTP_POST_VARS，比如一个函数内，你必须使用global关键字将它们声明为可访问的。
- 在PHP4.1中，如果track_vars变量是开的，参数也可以使用\$_GET和\$_POST数组来获得。这跟上面的\$_HTTP_GET_VARS和\$_HTTP_POST_VARS是非常相似的，区别在于\$_GET和\$_POST是超全局数组，因此在任何作用域中都是自动可见的。（例如，没有必要在函数中使用global关键字来声明\$_GET和\$_POST）\$_GET和\$_POST数组是访问输入参数的好办法。

- 如果register_globals变量是开的，参数就被赋值给相同名字的全局变量。在这种情况下，一个名为id的域，它的值就可以使用\$id直接读取，且不论对应的请求是通过get还是post方法。这是危险的，很快我们将介绍其中的原因。

track_vars和register_globals设置都可以编译到PHP环境中，或者通过php.ini文件来设置。在PHP 4.0.3中，track_vars一直都是开的，所以我假设在你的PHP安装中也是如此。

register_globals让通过全局变量访问输入参数变得容易，但是PHP开发者出于安全的考虑建议我们禁用这个选项。假设你正在写一个脚本，它需要用户输入密码，该密码在脚本中使用\$password变量表示。那么你就可以这样检查密码：

```
if (check_password ($password))
    $password_is_ok = 1;
```

这里假设密码匹配了，脚本把\$password_is_ok置为1。否则\$password_is_ok就不设置（对应布尔表达式中的false）。假如有人这样调用你的脚本：

```
http://your.host.com/chkpass.php?password_is_ok=1
```

如果register_globals是开的，PHP看到的就是password_is_ok参数被设置为1，并且把对应的\$password_is_ok变量置为1。结果就是当你的脚本执行时，变量\$password_is_ok总是1，无论密码是什么，或即使没有输入密码。register_globals的问题是，它允许外面的用户为全局变量提供默认值。最好的解决方案是关闭register_globals，此时你需要检查全局数组来获得输入参数的值。如果你不能关闭register_globals，不要去假设PHP变量没有初始值。除非你期望由一个输入参数来设置全局变量的初始值，否则最好把它显式设置为一个已知的值。密码检查代码可以这样写以确保仅仅\$password变量能被设置为输入参数。这样，\$password_is_ok被脚本设置为：

```
$password_is_ok = 0;
if (check_password ($password))
    $password_is_ok = 1;
```

本书中的PHP脚本不依靠register_globals设置。它们从全局参数数组中获取输入。

在PHP中，获取输入参数的另一个复杂因素是它们需要解码，这取决于配置参数magic_quotes_gpc的值。如果允许魔术转义（magic quotes），那么在输入参数值中的任何引号、反斜线、NUL字符都将被去掉反斜线。我假设这将节省你一步操作，因为它允许你直接提出数值并构建SQL语句字符串。然而，只有当你打算在语句中使用Web输入，且不执行预处

理及合法性检查时，这才是有用的。你应该先检查你的输入，在这种情况下去掉斜线是必要的。这意味着允许魔术转义（magic quotes）其实并不是真的很有用。

假设输入参数在多个给定的数据源中都可用，且事实是它们可能包含额外的反斜线，那么在PHP脚本中提取输入信息将是一个非常有趣的问题。如果你能控制你的服务器并且能设置各种配置，你当然可以在这些设置上编写脚本。但如果你不能控制你的服务器，或者打算编写在多台机器上都运行的脚本，而且你或许不知道这些机器的配置。幸运的是，再付出一点努力，编写一段通用的且正确工作的参数抽取代码是完全可能的，这段代码只对你的运行环境做非常少的假设。下面的辅助函数get_param_val()，只接受参数名作为它唯一的参数，返回对应的参数值。如果参数是不可用的，那么该函数返回一个未经设置的值。get_param_val()可以使用一个帮助函数strip_slash_helper()，该函数后续将介绍：

```
function get_param_val ($name)
{
    global $HTTP_GET_VARS, $HTTP_POST_VARS;

    $val = NULL;
    if (isset ($_GET[$name]))
        $val = $_GET[$name];
    else if (isset ($_POST[$name]))
        $val = $_POST[$name];
    else if (isset ($HTTP_GET_VARS[$name]))
        $val = $HTTP_GET_VARS[$name];
    else if (isset ($HTTP_POST_VARS[$name]))
        $val = $HTTP_POST_VARS[$name];
    if (isset ($val) && get_magic_quotes_gpc ())
        $val = strip_slash_helper ($val);
    return ($val);
}
```

要使用这个函数，获取单取参数id的值，你可以这样调用它：

```
$id = get_param_val ("id");
```

你可以测试\$id以判断参数id是否出现在输入中：

```
if (isset ($id))
    ... id parameter is present ...
else
    ... id parameter is not present ...
```

对于一个可能有多值的表单域（比如复选框组或者多选滚动列表），你应该将它用以[]结尾的名字表示。例如，一个由cow_order表格的SET列accessories中构建的列表，对每个允许的set值都有一个元素。想确定PHP是否将元素值作为数组对待，把该域命名为

`accessories[]`, 而不是`accessories` (19.3节中有个例子)。当表单被提交时, PHP把一组数值放在没有`[]`的名字的参数中。想访问这个参数, 你可以这样:

```
$accessories = get_param_val ("accessories");
```

变量`$accessories`的值会是一个数组, 不论该参数是多值、单值, 还是根本没值。确定性的因素并不是该参数究竟有无值, 而是你是否在表单中用`[]`方法给该参数命名了。

函数`get_param_val()`检查`$_GET`、`$_POST`、`$_HTTP_GET_VARS`和`$_HTTP_POST_VARS`数组。因此, 它可以正确地工作而不论该请求是通过GET还是POST, 也不论`register_globals`是否打开。该函数唯一假设的事情是`track_vars`是打开的。

函数`get_param_val()`在魔术转义 (magic quotes) 是否打开的情况下, 都能正常工作。它使用一个帮助函数`strip_slash_helper()`来从参数值中去掉反斜线:

```
function strip_slash_helper ($val)
{
    if (!is_array ($val))
        $val = stripslashes ($val);
    else
    {
        foreach ($val as $k => $v)
            $val[$k] = strip_slash_helper ($v);
    }
    return ($val);
}
```

函数`strip_slash_helper()`检查一个值是标量还是数组, 并据此处理它。为数组值使用递归算法的原因是, 在PHP中根据输入参数构建嵌套的数组是可能的。

为了让获取一组参数名变得简单些, 使用了另一个辅助函数:

```
function get_param_names ()
{
    global $HTTP_GET_VARS, $HTTP_POST_VARS;

    # 创建一个数组, 这个数组的每个元素都将一个参数名当作它的key和value (用name当作
    # key, 用来消除重复)
    $keys = array ();
    if (isset ($_GET))
    {
        foreach ($_GET as $k => $v)
            $keys[$k] = $k;
    }
    else if (isset ($HTTP_GET_VARS))
    {
        foreach ($HTTP_GET_VARS as $k => $v)
            $keys[$k] = $k;
    }
}
```

```
if (isset ($_POST))
{
    foreach ($_POST as $k => $v)
        $keys[$k] = $k;
}
else if (isset ($HTTP_POST_VARS))
{
    foreach ($HTTP_POST_VARS as $k => $v)
        $keys[$k] = $k;
}
return ($keys);
}
```

函数get_param_names()返回在HTTP变量数组中出现的参数名，如果数组间有重叠的话还会去掉冗余重复的名字。返回的值是一个数组，该数组的关键字和值都设为参数的名字。这样，你就能使用关键字或者值来作为多维列表的名字了。下面的例子使用值来打印名字：

```
$param_names = get_param_names ();
foreach ($param_names as $k => $v)
    print (htmlspecialchars ($v) . "<br />\n");
```

要构建末尾有参数的URL，并让其指向PHP脚本，你应该将参数用&字符分隔开。如果要使用一种不同的字符（比如;），请在PHP初始化文件中使用arg_separator配置变量来改变分隔符。

Python在Python中，cgi模块提供了在脚本环境中访问输入参数的方法。导入这个模块，并创建一个FieldStorage对象：

```
import cgi

params = cgi.FieldStorage ()
```

FieldStorage对象包含了通过GET或者POST方法提交的参数的信息，所以你无须知道是使用哪种方法发送请求的。环境中存在的每个变量，在该对象中都有对应的元素。它的key()方法返回一组可用的变量名字：

```
param_names = params.keys ()
```

如果一个变量、姓名是单值的，那么与它关联的值就是标量，你可以这么访问它：

```
val = params[name].value
```

如果该参数是多值的，params[name]就是一组MiniFieldStorage对象，该对象有name和value两个属性。所有这些对象的名字都是相同的。要为该参数创建一组包含所有值的对象时，请这么做：

```
val = []
for item in params[name]:
    val.append(item.value)
```

你可以检查类型，以区分该参数是单值还是多值的。下面的代码显示如何获得参数名字，并在每个参数中循环以打印它的名字和值，多值参数中每个值用逗号分开。这段代码要求你导入cgi和types模块：

```
params = cgi.FieldStorage()
param_names = params.keys()
param_names.sort()
print "<p>Parameter names:", param_names, "</p>"
items = []
for name in param_names:
    if type(params[name]) is not types.ListType: # 它是一个标量类型
        ptype = "scalar"
        val = params[name].value
    else: # 它是一个list类型
        ptype = "list"
        val = []
        for item in params[name]: # 遍历整个MiniFieldStorage
            val.append(item.value) # 用来得到item值
        val = ",".join(val) # 转化成字符串用以打印输出
    items.append("type=" + ptype + ", name=" + name + ", value=" + val)
print make_unordered_list(items)
```

当你尝试访问一个FieldStorage对象中不存在的参数时，Python会弹出一个例外。为避免这点，使用has_key()以确定该参数是否存在：

```
if params.has_key(name):
    print "parameter " + name + " exists"
else:
    print "parameter " + name + " does not exist"
```

单值参数有属性而不是值。例如，一个代表了上传的文件的参数，拥有额外的可用于获取文件内容的属性。19.8节进一步讨论了这个问题。

cgi模块期望的是被&字符分隔的URL参数。如果你基于cgi模块生成超链接URL，且该URL中还包含参数，那么请不要用;作分隔符。

Java在JSP页面中，隐式的请求对象提供了访问请求参数的途径：

```
getParameterNames()
```

返回一个String对象的枚举值，每个请求中出现的参数，都有一个枚举值。

```
getParameterValues(String name)
```

返回一个String对象数组，每个跟参数相关联的值都有一个对象，如果参数不存在的话，数组为null。

```
getParameterValue(String name)
```

返回跟参数关联的第一个值，如果该参数不存在则为null。

下面的例子介绍如何使用这些方法来显示参数：

```
<%@ page import="java.util.*" %>

<ul>
<%
    Enumeration e = request.getParameterNames ();
    while (e.hasMoreElements ())
    {
        String name = (String) e.nextElement ();
        // 如果参数是多值的，则要使用数组
        String[] val = request.getParameterValues (name);
        out.println ("<li> name: " + name + "; values:");
        for (int i = 0; i < val.length; i++)
            out.println (val[i]);
        out.println ("</li>");
    }
%>
</ul>
```

请求参数在JSTL标签中也是可用的，但需要使用特殊的变量param和paramValues。Param [name]返回给定参数的第一个值，因此非常适合于单值参数：

```
color value:
<c:out value="${param['color']}
```

paramValues[name]返回参数对应的一组值，所以非常适合于有多值的参数：

```
accessory values:
<c:forEach items="${paramValues['accessories']}" var="val">
    <c:out value="${val}"/>
</c:forEach>
```

如果一个参数名字作为对象属性名是合法的，你可以使用下面的概念访问该参数：

```
color value:
<c:out value="${param.color}"/>
accessory values:
<c:forEach items="${paramValues.accessories}" var="val">
    <c:out value="${val}"/>
</c:forEach>
```

生成一组参数对象，该对象有key和value属性，迭代paramValues变量：

```
<ul>
    <c:forEach items="${paramValues}" var="p">
        <li>
```

```
name:  
<c:out value="\${p.key}"/>;  
values:  
<c:forEach items="\${p.value}" var="val">  
    <c:out value="\${val}"/>  
</c:forEach>  
</li>  
</c:forEach>  
</ul>
```

构建指向JSP页面的URL，并把参数放在末尾，用&作为参数分隔符。

19.6 验证Web输入

Validating Web Input

问题

在从提交给脚本的参数中抽取信息后，你想要检查以确保它们是合法的。

解决方案

Web输入处理是一种数据导入，迄今为止你已经抽取了输入参数，那么你可以使用第10章中讨论的技术来验证这些参数。

讨论

Web表单处理的一个阶段是从用户提交的表单中抽取输入。当然从URL的末尾获得参数信息也是可能的。对于这两种方法，如果你打算将输入存入数据库的话，最好检查数据的合法性。

当客户端通过Web向你发送输入时，你不是真的明白它们到底传过来什么。当你呈现一个表单让用户填写时，大部分时间里它们可能是友善的，并输入你期望类型的数据。当一个恶意的用户会把表单存入文件中，修改这个文件以你不期望的表单选项，把文件重载到浏览器窗口中，然后提交修改过的表单。你的表单处理脚本并不知道这其中的不同。如果你只是编写脚本处理你期望的善意用户会输入的数据，那么这个脚本就会表现异常，甚至崩溃，或者影响你的数据库。(19.7节讨论了到底会产生哪些坏的影响)出于这个考虑，在使用Web输入构建SQL语句之前，执行一些有效性检验是非常必要的。

即使没有恶意的用户，初步的检验也是不错的。如果你需要用户填写一个域而他忘了，你就需要提醒他提供一个值。这可能会调用一个简单的“这个参数存在吗？”检查，或者涉及更多的东西。典型的检验步骤包含：

- 检查内容格式，例如确保一个值看起来像个整数或日期。这会涉及重新格式化数据以确保MySQL能接受（比如，把一个日期从MM/DD/YY的格式转化为ISO格式）。
- 确定一个值是否在一组合法的值内。也许该值必须被列在一个ENUM或者SET列的定义中，或者必须出现在一个查找表格中。
- 过滤掉无关字符，比如电话号码和信用卡号中的空格和破折号等。

这些操作中，有一部分跟MySQL并无关系，除非你期望处理的值必须和列的定义保持一致。例如，你正准备将一个值存入INT列中，你要先确保这个值是整数（这里以Perl脚本为例）：

```
$val =~ /^\\d+$/  
or die "Hey! '" . escapeHTML ($val) . "' is not an integer!\n";
```

对于其他类型的检验，就与MySQL紧密相关了。如果一个域的值将被存入某ENUM列，你得检查在INFORMATION_SCHEMA中的列定义，从而确保该值是一个合法的枚举值。

在介绍了上述这些你或许想执行的Web输入的验证之后，其他就不多讨论了。这里的以及其他地方的验证测试都已在第10章中介绍。第10章的内容大部分是面向批量输入验证的，但那里讨论的技术同样适用于Web编程的，因为处理表单输入或URL参数在本质上都是一种数据导入操作。

19.7 将Web输入存入数据库

Cleaning Web Input in a Database

问题

从Web中获取的输入在执行恰当的检查前，是不能被信任也不应被放入数据库的。

解决方案

使用占位符或者一个引用函数来清洁数据，从而让你构建的SQL语句是合法的，且不会引发“SQL注入攻击”。同样地，允许严格SQL模式将导致MySQL服务器拒绝对列数据类型而言非法的值。

讨论

你已经在Web脚本中抽取出输入参数值，并执行检查以确保它们是合法的之后，接下来你就会在构建SQL语句时用到它们。这事实上是输入流程的一个简单部分，尽管你并不需要

执行恰当的检查以避免那些可能会让你后悔的错误。首先，考虑一下什么是可能出错的，然后看看如何防治它们。

假设你有一个查询表单，该表单包含一个keyword域并表现得像一个简单搜索引擎的前端。当用户提交一个关键字时，你想使用它构建一个语句并在某个表中查找匹配的行：

```
SELECT * FROM mytbl WHERE keyword = 'keyword_val'
```

这里keyword_val表示用户输入的值。如果这个值是eggplant之类的东西，那么语句就是：

```
SELECT * FROM mytbl WHERE keyword = 'eggplant'
```

假设这将生成一个小的结果集，这个语句返回所有匹配eggplant的列。但如果用户很狡猾，他输入以下的值以搅乱你的脚本：

```
'eggplant' OR 'x'='x'
```

在这种情况下，语句变为：

```
SELECT * FROM mytbl WHERE keyword = 'eggplant' OR 'x'='x'
```

这个语句匹配了表格中所有的行！如果这个表非常大，那么该输入实质上就成了一个拒绝服务攻击，因为它导致你的系统把应给常规请求的资源都用来执行无用工作了。这种类型的攻击也就是SQL注入，因为用户往你的语句中注射了可执行的SQL代码，而你期望接受的却是一个非可执行的数值。多数情况下，SQL注入攻击的结果是：

- MySQL服务器额外的负载。
- 当脚本尝试消化从MySQL中返回的结果集时，会引发内存溢出问题。
- 当脚本把结果发送给客户端时，消耗额外的网络带宽。

如果你的脚本生成DELETE或UPDATE语句，那么这类搅局行为的后果将更严重。你的脚本也许会执行一个语句，该语句完全清空了一个表，或者当你的本意是修改一行时它却更新了该表的所有行。

前面的讨论意味着为你的数据库提供一个Web接口，会把你暴露在很多安全漏洞面前。然而，你可以通过一个简单的预警检查来阻止这些问题的发生：别把数据值放入语句字符串中，而是使用占位符或者一个解码函数。例如，在Perl中你可以使用占位符来处理一个输入参数：

```
$sth = $dbh->prepare ("SELECT * FROM mytbl WHERE keyword = ?");  
$sth->execute (param ("keyword"));  
# ...得到结果集...
```

或者使用quote()：

```
$keyword = $dbh->quote (param ("keyword"));
$stmt = $dbh->prepare ("SELECT * FROM mytbl WHERE keyword = $keyword");
$stmt->execute ();
# ...得到结果集...
```

上述任一方法中，如果用户输入搅局的数据，那么语句是无害的：

```
SELECT * FROM mytbl WHERE keyword = 'eggplant\' OR \'x\'='x'
```

结果，该语句在表中匹配不到任何一行记录——这无疑是对那些想破坏你脚本的人的一个合适的回应。

占位符和引用技术在Ruby、PHP、Python和Java中是相似的，这在2.5节中讨论了。对于使用JSTL标签库编写的JSP页面而言，你通过占位符和<sql:param>标签能引用输入参数值（见17.3节）。例如，在SELECT语句中使用名为keyword的表单参数的值，如下：

```
<sql:query dataSource="${conn}" var="rs">
    SELECT * FROM mytbl WHERE keyword = ?
    <sql:param value="${param['keyword']}"/>
</sql:query>
```

占位符技术没有考虑的一个问题是如何恰当的对空字符串或Null进行解释：如果一个表单域是可选的，那么用户把它置为空时，如何在数据库中存入相应信息呢？也许该值是一个空字符串，或者它应该被解释为NULL。一种解决方案是咨询列的元数据。如果列能包含NULL值，那么就将空字符串当作NULL，否则使用一个空域来表示空字符串。

占位符和编码函数仅适用于SQL数据值。一个没有考虑的问题是如何处理用于其他类型的语句的Web输入，比如数据库、表格和列的名字。如果你本意是在语句中包含这些值，那么你必须逐字地输入它们，这意味着你得先检查它们。例如，如果你构建一个如下的语句，你应该验证\$tbl_name是否包含一个合法的值：

```
SELECT * FROM $tbl_name;
```

但“合法”是什么意思？如果你没有一个表名是包含特殊字符的，那么只要确保\$tbl_name只包含数字字符串或下划线就足够了。另一种方法是执行一条语句来判断这个表到底是不是存在的。（你可以检查INFORMATION_SCHEMA或使用SHOW TABLES）这更简单使用，但代价是执行一条额外的语句。

还有一个更好的选择，就是使用标识符-引用程序，前提是你要有这种程序。这种方法不要求执行额外的语句，因为它在一个语句中确保任意字符串的安全。如果标识符不存在，该语句应该失败，2.6节已经讨论了这种方法。

为了Web脚本的额外保护，可以把客户端的检查和服务器端的检查结合起来。你可以把SQL服务器在接受输入值时的模式设为严格的，这样它会拒绝任何不匹配你列定义的数据。关于这方面的讨论，请查阅第10.20节。

你应该尝试去破坏你的脚本

这部分的讨论可以表达为防止其他用户攻击你的脚本。但把你自己的位置放在攻击者的位置，并考虑“我如何破坏这个应用呢？”其实不是个坏点子。也就是说，考虑是否能提交一些应用无法处理的输入，以及一些能导致畸形语句的输入。如果你能让一个应用表现失常，那么其他人，无论是有意还是无意的，肯定也能这么做。小心糟糕的输入，并据此写好你的应用。总之，更好的准备总比空期望要强。

参考

本章的其他章节示范了如何将Web输入结合到语句中来。19.8节展示如何上传文件并把它们存入MySQL。19.9节使用输入作为搜索关键字，示范了一个简单的搜索应用。19.10和19.11节处理通过URL提交的参数。

19.8 处理文件上传

Processing File Uploads

问题

你想允许文件上传到你的Web服务器上，并存储在你的数据库中。

解决方案

给用户呈现一个Web表单，该表单包含一个文件域。当用户提交表单时，提取文件并存储它。

讨论

一种特殊类型的Web输入是文件上传。文件被作为post请求的一部分发送，但跟其他的post参数相比，它被服务器区别对待。这是因为文件包含几部分不同的信息，比如它的内容、它的MIME类型、它的原始文件名，以及它在Web服务器主机上的临时存储文件名。

要处理文件上传，你必须给用户发送一种特殊类型的表单。无论你使用什么样的API，你都必须这么做。当用户提交该表单时，检查和处理上传文件的操作才跟具体使用的API相关。

要生成一个允许文件上传的表单，开始标签`<form>`应该使用`post`方法且必须包含一个`enctype`属性及一个`multipart/form-data`的值：

```
<form method="post" enctype="multipart/form-data" action="script_name">
```

如果表单使用`application/x-www-form-urlencoded`编码类型来提交，文件传输就无法正常工作。

要在表单的文件上传域中包含一个文件，使用一个`<input>`元素。例如，要表示一个60字符的文件域`upload_file`，需要做如下操作：

```
<input type="file" name="upload_file" size="60" />
```

浏览器将这个域显示为一个文本输入框，用户可以手工地在这个框里输入名字。它还可以是一个浏览按钮，该按钮允许用户通过文件浏览系统对话框选择文件。当用户选择一个文件并提交对话框时，浏览器编码文件的内容，并把它放入`post`请求中。在下一刻，Web服务器接受到请求并调用你的脚本来处理，处理的细节因API的不同而不同。但文件传输基本上是这样一个过程：

- 当你的上传处理脚本开始工作时，文件已经上传并存储在一个临时文件夹中。你脚本所需要做的事情就是读取文件。这个临时的文件是以打开文件描述子文件或临时文件名的方式供你脚本使用的。文件的尺寸可以通过文件描述子获知。API也许还可以得到文件的其他信息，比如它的MIME类型（要注意的是有些浏览器不会发送MIME值）。
- 当你的脚本执行结束时，Web服务器自动删除上传的文件。如果你在脚本执行完后，仍然要保持该文件的内容，那么脚本必须将文件存到一个更持久的位置，比如说数据库或者文件系统中的某处。如果你把文件存储在文件系统中，那么你存储的目录对于Web浏览器必须是可访问的。（但别把文档放到根目录或者任意用户目录下，否则一个远程攻击者将能够在你的Web服务器上安装脚本或者HTML页面。）
- API或许能让你控制临时文件目录的位置，或者上传文件的最大长度。将这个目录改变为一个仅仅由你的Web服务器就能够访问的位置，相比于其他用户服务器主机能访问的位置，安全性将有所增强。

这里我们讨论如何生成包含文件上传域的表单。它也说明了如何使用Perl脚本`post_image.pl`

来处理上传。这个脚本跟使用命令行来上传图片的脚本store_image.pl是相似的。脚本post_image.pl的特点在于，它允许你通过Web上传图片，并仅将图片存储在MySQL中，而store_image.pl脚本不仅将图片存在MySQL中，还将它存储在文件系统中。

这里讨论的是如何使用PHP和Python获取文件上传信息，这并不是重复Perl中图片上传的整个场景，光盘中包含了post_image.pl在其他语言下的实现。

在Perl中的上传

你可以基于CGI.pm用好几种方法来设置一个表单的multipart编码。下面的语句是等价的：

```
print start_form (-action => url (), -enctype => "multipart/form-data");
print start_form (-action => url (), -enctype => MULTIPART ());
print start_multipart_form (-action => url ());
```

第一个语句设置了编码的类型。第二个使用CGI.pm的MULTIPART()函数，比起记住手工编码的值，该函数更容易一点。第三个语句是最容易的，因为start_multipart_form()自动提供了enctype参数。(像start_form()、start_multipart_form()使用一个默认的post请求，所以你不用包含方法参数。)

下面是一个简单的表单，它包含一个文本域供用户输入图片的名字，一个文件域供用户选择图片文件，以及一个提交按钮：

```
print start_multipart_form (-action => url (),
    "Image name:", br (),
    textfield (-name =>"image_name", -size => 60),
    br (), "Image file:", br (),
    filefield (-name =>"upload_file", -size => 60),
    br (), br (),
    submit (-name => "choice", -value => "Submit"),
    end_form ();
```

当用户提交一个上传文件时，通过提取文件域的参数值来处理它：

```
$file = param ("upload_file");
```

文件上传参数的值在CGI.pm中是特殊的，因为你能以两种方法使用它。你可以把它作为一个打开的文件句柄，从而读取文件的内容或把它传递给uploadInfo()，并获得一个指向hash函数的句柄，该hash函数提供了诸如文件MIME类型这样的信息。接下来的列表表示了post_image.pl如何显示和处理一个提交的表单。当第一次被调用时，post_image.pl生成一个有文件域的表单。对于初始调用，没有文件被上传，所以该脚本别的什么都不做。如果用户提交一个图片文件，该脚本获得图片的名字，读入文件的内容，判断它的MIME类

型，并在image表中存入新的一行。作为示例，post_image.pl还显示了所有uploadInfo()函数能够获得的上传文件的信息：

```
#!/usr/bin/perl
# post_image.pl - 允许用户通过post请求来上传图片文件

use strict;
use warnings;
use CGI qw(:standard escapeHTML);
use Cookbook;

print header (), start_html (-title => "Post Image", -bgcolor => "white");

# 使用multipart编码，因为表单包含一个文件上传域

print start_multipart_form (-action => url (),
    "Image name:", br (),
    textfield (-name =>"image_name", -size => 60),
    br (), "Image file:", br (),
    filefield (-name =>"upload_file", -size => 60),
    br (), br (),
    submit (-name => "choice", -value => "Submit"),
    end_form ());

# 获得图片文件的句柄及图片的名称

my $image_file = param ("upload_file");
my $image_name = param ("image_name");

# 必须要么不带参数（此时脚本是第一次被调用），要么带有两个参数（此时表单已经填满）。
# 如果仅填充了一个，用户并未完全填满表单。

my $param_count = 0;
++$param_count if defined ($image_file) && $image_file ne "";
++$param_count if defined ($image_name) && $image_name ne "";

if ($param_count == 0)      # 初始调用
{
    print p ("No file was uploaded.");
}
elsif ($param_count == 1)    # 不完全的表单
{
    print p ("Please fill in BOTH fields and resubmit the form.");
}
else                      # 文件被上传
{
    my ($size, $data);

    # 如果一个图片文件被上传，输出一些关于它的信息，然后将其保存进入数据库

    # 获得指向包含文件相关信息的哈希表的引用，然后以“key=x,value=y”的格式显示信息
```

```

my $info_ref = uploadInfo ($image_file);
print p ("Information about uploaded file:");
foreach my $key (sort (keys (%{$info_ref})))
{
    printf p ("key="
              . escapeHTML ($key)
              . ", value="
              . escapeHTML ($info_ref->($key)));
}
$size = (stat ($image_file))[7]; # 从文件句柄获得文件大小
print p ("File size: " . $size);

binmode ($image_file); # 对于二进制数据有用
if (sysread ($image_file, $data, $size) != $size)
{
    print p ("File contents could not be read.");
}
else
{
    print p ("File contents were read without error.");

# 获得MIME类型，如果不存在使用通用的默认值

my $mime_type = $info_ref->{'Content-Type'};
$mime_type = "application/octet-stream" unless defined ($mime_type);

# 将图片保存到数据库表中。(使用REPLACE将有相同名称的旧的图片删除。)

my $dbh = Cookbook::connect ();
$dbh->do ("REPLACE INTO image (name,type,data) VALUES(?, ?, ?)",
           undef,
           $image_name, $mime_type, $data);
$dbh->disconnect ();
}

print end_html ();

```

在PHP中的上传

要在PHP中编写一个上传表单，得包括一个文件域。如果你喜欢，你还可以在文件域之前加入一个名为MAX_FILE_SIZE的隐藏域，以及一个你期望接受的最大的文件长度：

```

<form method="post" enctype="multipart/form-data"
      action=<?php print (get_self_path ()); ?>>
<input type="hidden" name="MAX_FILE_SIZE" value="4000000" />
Image name:<br />
<input type="text" name="image_name" size="60" />
<br />
Image file:<br />
<input type="file" name="upload_file" size="60" />
<br /><br />

```

```
<input type="submit" name="choice" value="Submit" />  
</form>
```

需要注意的是，`MAX_FILE_SIZE`仅仅是建议值，因为它非常容易被篡改。要规定一个不能被超越的值，请在PHP的配置文件`php.ini`中使用`upload_max_filesize`变量。还有一个变量叫`file_uploads`，它决定了我们是否允许文件上传。

当用户提交表单时，文件上传信息也可以用如下方法获得：

- 在PHP 4.1中，`post`请求的文件上传信息被放置在一个数组`$_FILES`中，每个上传的文件在该数组中都有入口。每个入口本身又是一个数组，这个数组有四个元素。比如，一个表单有文件域`upload_file`且用户提交了一个文件，那么该文件的信息就是：

```
$_FILES["upload_file"]["name"]  
$_FILES["upload_file"]["tmp_name"]  
$_FILES["upload_file"]["size"]  
$_FILES["upload_file"]["type"]
```

这些变量表示的是在客户端的原始文件名、在服务器主机的临时文件名、以字节为单位的文件大小以及文件的MIME类型。需要小心的是，因为即使用户没有提交文件，也可能存在一个对应文件上传域的数组入口。在这种情况下，`tmp_name`是空字符串或者`none`。

- 在PHP 4之前的版本中，文件上传信息是一个数组`$HTTP_POST_FILES`，该数组入口结构很像`$_FILES`。一个文件域为`upload_file`，关于该文件的信息就是：

```
$HTTP_POST_FILES["upload_file"]["name"]  
$HTTP_POST_FILES["upload_file"]["tmp_name"]  
$HTTP_POST_FILES["upload_file"]["size"]  
$HTTP_POST_FILES["upload_file"]["type"]
```

`$_FILES`是一个超全局数组。`$HTTP_POST_FILES`在非全局的作用域下比如说在一个函数中使用时，必须使用`global`关键字声明。

为弄清到底哪个数组包含文件上传信息，有必要写一个辅助函数。下面的函数`get_upload_info()`使用了一个参数，该参数对应文件上传域的名字。这个函数检查`$_FILES`和`$HTTP_POST_FILES`数组，并返回与文件相关的信息，如果信息不可用则返回`NULL`。对于一个成功的调用，数组元素关键字是`"tmp_name"`、`"name"`、`"size"`和`"type"`：

```
function get_upload_info ($name)  
{  
    global $HTTP_POST_FILES, $HTTP_POST_VARS;  
  
    # 首先寻找PHP 4.1 $_FILES数组中的信息。检查tmp_name成员以确定存在一个文件。  
    # (即使文件没有被上传，$_FILES中的条目也可能存在。)  
    if (isset ($_FILES))
```

```
{  
    if (isset ($_FILES[$name])  
        && $_FILES[$name] ["tmp_name"] != ""  
        && $_FILES[$name] ["tmp_name"] != "none")  
        return ($_FILES[$name]);  
}  
# 接下来寻找PHP 4 $HTTP_POST_FILES数组中的信息  
else if (isset ($HTTP_POST_FILES))  
{  
    if (isset ($HTTP_POST_FILES[$name])  
        && $HTTP_POST_FILES[$name] ["tmp_name"] != ""  
        && $HTTP_POST_FILES[$name] ["tmp_name"] != "none")  
        return ($HTTP_POST_FILES[$name]);  
}  
return (NULL);  
}
```

查看post_image.php脚本以了解如何使用这个函数来获取图片信息，并把这些信息存入MySQL中。

PHP配置变量upload_tmp_dir决定了上传文件存储在哪里。很多系统中都是在/tmp目录下，但你也可以重载这个变量，并重新配置PHP来使用一个不同的属于Web服务器用户ID的目录。

在Python中的上传

在Python中一个简单的上传表单是这样：

```
print """  
<form method="post" enctype="multipart/form-data" action="%s">  
Image name:<br />  
<input type="text" name="image_name", size="60" />  
<br />  
Image file:<br />  
<input type="file" name="upload_file", size="60" />  
<br /><br />  
<input type="submit" name="choice" value="Submit" />  
</form>  
""% (os.environ["SCRIPT_NAME"])
```

当用户提交表单时，它的内容可以通过cgi模块中的FieldStorage()方法来获得参见19.5节。每一个输入参数在结果对象中都有一个对应元素。对于一个文件上传域而言，你可以这样获取信息：

```
form = cgi.FieldStorage ()  
if form.has_key ("upload_file") and form["upload_file"].filename != "":  
    image_file = form["upload_file"]  
else:  
    image_file = None
```

根据我所阅读的绝大多数文档，当文件已经上传后，与文件域所对应的对象的文件属性应

该是真的。可不幸的是，即使用户提交表单是文件域为空，这个文件属性看起来也是真的。甚至有可能在没有文件被真正上传时，类型属性也被设置了（例如，该属性被设置为application/octet-stream）。根据我的经验，判断一个文件是否上传的更可靠的方法是测试filename属性：

```
form = cgi.FieldStorage ()  
if form.has_key ("upload_file") and form["upload_file"].filename:  
    print "<p>A file was uploaded</p>"  
else:  
    print "<p>A file was not uploaded</p>"
```

假设一个文件已经上传，访问参数的value以读取文件及它的内容：

```
data = form["upload_file"].value
```

查看post_image.py脚本以了解如何使用这个函数来获取图片信息并把信息存入MySQL。

19.9 执行搜索并显示结果

Performing Searches and Presenting the Results

问题

你想实现一个基于Web的搜索接口。

解决方案

提供一个表单，该表单包含允许用户提供搜索参数比如关键字的域。使用关键字来构建一个数据库查询，然后显示查询结果。

讨论

执行基于Web的脚本的搜索接口为访问你网站的人们提供了方便，因为他们不需要知道任何SQL就能在你的数据库中找到信息。用户提供关键字来描述他们感兴趣的事情，你的脚本构建出合适的语句并执行。一个共同的很普遍的范例涉及到包含一个或多个用于输入搜索参数的表单。用户填写这个表单，提交表单，并收到一个新的Web页面，该页面呈现了匹配参数的搜索结果。

作为这样一个脚本的编写者，你需要执行一下操作：

1. 生成表单并把它发送给用户。

- 解释提交的表单，并基于它的内容构建一个SQL语句。这包括恰当地使用占位符或引用，来避免错误输入导致的应用崩溃或脚本破坏。
- 执行语句并显示它的结果。如果结果集比较小的话，显示会很简单。但结果集越大，显示越复杂。在下面的例子中，你也许需要使用一个分页显示方法来罗列匹配的记录——这意味着，一个包含多页的显示，每页中显示整个语句结果的一部分。多页显示的好处是不会因为太多的信息而一下子过载客户端。19.10节讨论了如何实现多页显示。

这里给出一个脚本，该脚本实现了一个最低程度的搜索接口：只有一个关键字域的表单，据此构建一个最多返回一行结果的语句。这个脚本在states表上执行了一个两步的搜索。这就是说，如果用户输入州名，它就会查找对应的缩写。相反的，如果用户输入缩写，它就会查找对应的全称。这个脚本是search_state.pl：

```
#!/usr/bin/perl
# search_state.pl - 简单的“州名搜索”应用

# 展现具有一个输入域和一个提交按钮的表单。
# 用户在输入域中输入州名缩写或者州名并提交表单。脚本找到缩写并显示完全名称，或者找到
# 州名并显示缩写。

use strict;
use warnings;
use CGI qw(:standard escapeHTML);
use Cookbook;

my $title = "State Name or Abbreviation Lookup";

print header (), start_html (-title => $title, -bgcolor => "white");

# 提取keyword参数。如果它存在且非空，视图执行一次查找。

my $keyword = param ("keyword");

if (defined ($keyword) && $keyword !~ /^$s*/)
{
    my $dbh = Cookbook::connect ();
    my $found = 0;
    my $s;

    # 首先试图作为州名缩写来查找keyword；如果失败，尝试作为名称来寻找它。
    $s = $dbh->selectrow_array ("SELECT name FROM states WHERE abbrev = ?",
                                undef, $keyword);
    if ($s)
    {
        ++$found;
    }
}
```

```

print p ("You entered the abbreviation: " . escapeHTML ($keyword));
print p ("The corresponding state name is : " . escapeHTML ($s));
}
$s = $dbh->selectrow_array ("SELECT abbrev FROM states WHERE name = ?",
                           undef, $keyword);
if ($s)
{
    ++$found;
    print p ("You entered the state name: " . escapeHTML ($keyword));
    print p ("The corresponding abbreviation is : " . escapeHTML ($s));
}
if (!$found)
{
    print p ("You entered the keyword: " . escapeHTML ($keyword));
    print p ("No match was found.");
}

$dbh->disconnect ();
}

print p (qq{
Enter a state name into the form and select Search, and I will show you
the corresponding abbreviation.
Or enter an abbreviation and I will show you the full name.
});

print start_form (-action => url (),
                  "State: ",
                  textfield (-name => "keyword", -size => 20),
                  br (),
                  submit (-name => "choice", -value => "Search"),
                  end_form ());

print end_html ();

```

这个脚本首先检查环境，并了解是否有一个keyword。如果有，它执行语句使用该参数在states表格中查找一个匹配，并显示结果。然后它提供表单以让用户输入新的搜索。

当你浏览这个脚本时，你会发现keyword域的值从一个调用延续到下一个。这主要是因为CGI.pm使用脚本环境中的值来初始化表单域。如果你不喜欢这种行为，可以每次显式地为各个域提供一个空值，并在textfield()调用中重载一个参数：

```

print textfield (-name => "keyword",
                 -value => "",
                 -override => 1,
                 -size => 20);

```

另外一种方法是，在生成域之前，清除环境中参数的值：

```

param (-name => "keyword", -value => "");
print textfield (-name => "keyword", -size => 20);

```

19.10 生成上一页和下一页链接

Generating Previous-Page and Next-Page Links

问题

一个语句匹配了如此多的行，以至于在单一Web页中呈现这些结果看起来很笨拙。

解决方案

将语句的输出在几个Web页面间拆分开来，并包含链接以方便用户在页面间导航。

讨论

如果一个语句匹配了大量的行，把它们在单个页面内显示出来将导致无法浏览的结果。对于这样的情况，一种更便捷的方法是你把这些结果分开并放到多个页面中，从而避免一次传递给用户太多的信息，但它相对于单页面显示更复杂：

```
SELECT name, abbrev, statehood, pop FROM states ORDER BY name;
```

MySQL使得从一个结果集中选择部分内容变得容易了：添加一个LIMIT子句，这表明你想要多少行。双参数形式的LIMIT还表明你希望跳过结果集中前面的多少行，以及选择多少行。因此，选择states表中一部分内容变成：

```
SELECT name, abbrev, statehood, pop FROM states ORDER BY name  
LIMIT skip,select;
```

因此，一个随之而来的问题就是确定跳过和选择合适的值。另一个问题就是生成指向其他页面或查询结果的链接，这儿你可以决定为这些链接选择哪种分页风格？

- 一种分页风格显示“上一页”和“下一页”链接。要实现这点，你需要知道是否在这之前或之后的行都显示在当前页中了。
- 另外一种分页风格是为每一个可用页面显示一个链接。这允许用户直接跳转到任一页，而不仅仅是前页和后页。要呈现这种类型的导航，你必须知道结果集中所有行的数目，以及每页的行数，这样你才能决定一共有多少页。

上一页和下一页的分页显示

下面的脚本state_pager1.pl以包括指向前页和后页的分页方式显示了states表中的行。对于给定的一页，你可以如下判断需要哪些链接：

- 如果当前页面的行，在结果集中不是处于最前面，那么需要一个“前一页”链接。反之如果当前页以第一行开始，那么就不需要这个链接。
- 如果当前页面的行，在结果集中不是处于最后面，那么需要一个“后一页”链接。你可以通过执行SELECT COUNT(*)语句来判断共有多少行。另外一种方法是多选择一行。举个例子，你计划一次显示10行，那么尝试选择11行，如果你得到11行，那么就需要下一页；如果你只得到10行或者更少，那么就没有下一页。脚本state_pager1.pl使用了后面这种方法。

要判断结果集的当前位置及显示多少行，state_pager1.pl查找start和per_page输入参数。当你第一次调用这个脚本时，这些参数不会出现，所以它们被分别初始化为1到10。之后，脚本生成了“前一页”和“后一页”链接，该链接的URL中包含了合适的参数值以供脚本在下次被调用时判断。

```
#!/usr/bin/perl
# state_pager1.pl - 州名的分页显示，带有上页/下页链接

use strict;
use warnings;
use CGI qw(:standard escape escapeHTML);
use Cookbook;

my $title = "Paged U.S. State List";

my $page = header ()
    . start_html (-title => $title, -bgcolor => "white")
    . h3 ($title);

my $dbh = Cookbook::connect ();

# 收集确定显示中我们所在位置的参数并验证它们是整数。
# 如果参数不存在或有所缺失，默認為结果集的起始位置，并且每页10条记录。

my $start = param ("start");
$start = 1
    if !defined ($start) || $start !~ /^ \d+$/ || $start < 1;
```

```

my $per_page = param ("per_page");
$per_page = 10
if !defined ($per_page) || $per_page !~ /\d+$/ || $per_page < 1;

# 如果start>1, 然后我们需要一个可用的"上一页"链接。为确定是否有下一页, 尝试选择比
# 我们所需要的记录多一个。如果我们获得了那么多, 仅显示第一个$per_page记录, 但要增
# 加一个可用的"下一页"链接。

# 选择记录集当前页中的记录, 然后试图获得一个额外的记录。(如果我们获得了额外的一个,
# 我们不会显示它, 但它的出现告诉我们存在下一页。)

my $stmt = sprintf ("SELECT name, abbrev, statehood, pop
                     FROM states
                     ORDER BY name LIMIT %d,%d",
                     $start - 1,           # 跳过的记录数
                     $per_page + 1);      # 选择的记录数

my $tbl_ref = $dbh->selectall_arrayref ($stmt);

$dbh->disconnect ();

# 以HTML table的形式显示结果
my @rows;
push (@rows, Tr (th ([ "Name", "Abbreviation", "Statehood", "Population"])));
for (my $i = 0; $i < $per_page && $i < @{$tbl_ref}; $i++)
{
    # 获得的$i行中的数据值
    my @cells = @{$tbl_ref->[$i]};
    # 将值匹配到HTML编码的值, 或者如果为null或空时, 匹配到nbsp;
    @cells = map {
        defined ($_) && $_ ne "" ? escapeHTML ($_) : " " }
    @cells;
    # 添加表格到table中
    push (@rows, Tr (td (\@cells)));
}

$page .= table ({-border => 1}, @rows) . br ();

# 如果我们不在查询结果的起始位置, 展现一个前页的活跃链接。否则, 展现静态文本。

if ($start > 1)          # 可用的链接
{
    my $url = sprintf ("%s?start=%d;per_page=%d",
                        url (),
                        $start - $per_page,
                        $per_page);
    $page .= "[ " . a ({-href => $url}, "previous page") . "] ";
}
else                      # 静态文本
{
    $page .= "[previous page]";
}

```

```

}

# 如果我们获得了多余的记录，展现到下一页的可用链接。否则展现静态文本。

if (@{$tbl_ref} > $per_page) # 可用链接
{
    my $url = sprintf ("%s?start=%d;per_page=%d",
                       url (),
                       $start + $per_page,
                       $per_page);
    $page .= "[" . a ({-href => $url}, "next page") . "]";
}
else                                # 静态文本
{
    $page .= "[next page]";
}

$page .= end_html ();

print $page;

```

指向每一页链接的分页显示

接下来的脚本state_pager2.pl，和state_pager1.pl非常类似，但它呈现的是一种包含指向结果集中每个页面的导航链接。要做到这点，需要知道一共有多少行。脚本state_pager2.pl通过执行SELECT COUNT(*)语句来确定出有多少行。在知道全部行的数量之后，脚本在显示一部分结果时不需要再选择额外的行。

省略跟state_pager1.pl相同的部分，脚本state_pager2.pl的中间部分如下获取及生成链接：

```

# 确定记录的完全数目

my $total_recs = $dbh->selectrow_array ("SELECT COUNT(*) FROM states");

# 选择结果集当前页中的记录

my $stmt = sprintf ("SELECT name, abbrev, statehood, pop
                     FROM states
                     ORDER BY name LIMIT %d,%d",
                     $start - 1,      # 跳过的基本数目
                     $per_page);      # 选择的基本数目

my $tbl_ref = $dbh->selectall_arrayref ($stmt);

$dbh->disconnect ();

# 以HTML table的形式显示结果
my @rows;
push (@rows, Tr ([["Name", "Abbreviation", "Statehood", "Population"]]));
for (my $i = 0; $i < @{$tbl_ref}; $i++)
{
    # 获取$i行中的数据值

```

```

my @cells = @{$tbl_ref->[$i]}; # 将值匹配到HTML编码的值，或者如果为null或
# 空时，匹配到$nbsp;
@cells = map {
    defined ($_) && $_ ne "" ? escapeHTML ($_) : "&nbsp;" # 添加表格到table中
} @cells;
push (@rows, Tr (td (\@cells)));
}

$page .= table ({-border => 1}, @rows) . br ();

# 生成到结果集所有页的链接。所有的链接都是可用的，除了指向当前页的一个，它被显示为静
# 态文本。链接标签格式为 "[m 到 n]"，m 和 n 分别是页面上显示的记录的第一条和最后一条
记
# 录。

for (my $first = 1; $first <= $total_recs; $first += $per_page)
{
    my $last = $first + $per_page - 1;
    $last = $total_recs if $last > $total_recs;
    my $label = "$first to $last";
    my $link;

    if ($first != $start) # live link
    {
        my $url = sprintf ("%s?start=%d;per_page=%d",
                           url (),
                           $first,
                           $per_page);
        $link = a ({-href => $url}, $label);
    }
    else # static text
    {
        $link = $label;
    }
    $page .= "[{$link}] ";
}

```

19.11 生成点击排序的表格头单元

Generating "Click to Sort" Table Headings

问题

你希望把查询结果作为表格显示在Web页面中，并允许用户选择一列以根据该列排序。

解决方案

每列以一个超链接打头，该链接能重新显示表格，并根据对应的列来排序表格中各行。

讨论

当一个Web脚本执行时，它通过检查环境，找到来自用户的参数及对应的值，从而确定要执行什么操作。在很多情况下，这些参数来自一个用户，但脚本也可以自己往URL上添加参数。还有种方法是一个脚本在被调用后，能往它的下一次调用发送信息。结果就是脚本通过它生成的URL跟自己通信。这个技术的一个典型例子是显示一个查询的结果，这样用户可以选择某列并根据该列通信排序结果集。一般做法是，每列以超链接打头，该链接能够重新显示这个表格，并根据该列把表格内容排序。

下面的例子使用了mail表，该表有以下内容：

```
mysql> SELECT * FROM mail;
+-----+-----+-----+-----+-----+-----+
| t   | srcuser | srchost | dstuser | dsthost | size |
+-----+-----+-----+-----+-----+-----+
| 2006-05-11 10:15:08 | barb    | Saturn   | tricia  | mars    | 58274 |
| 2006-05-12 12:48:13 | tricia  | mars    | gene    | venus   | 194925 |
| 2006-05-12 15:02:49 | phil    | mars    | phil    | saturn  | 1048  |
| 2006-05-13 13:59:18 | barb    | Saturn   | tricia  | venus   | 271   |
| 2006-05-14 09:31:37 | gene    | venus   | barb    | mars    | 2291  |
| 2006-05-14 11:52:17 | phil    | mars    | tricia  | saturn  | 5781  |
| 2006-05-14 14:42:21 | barb    | venus   | barb    | venus   | 98151 |
| 2006-05-14 17:03:01 | tricia  | saturn  | phil    | venus   | 2394482 |
| 2006-05-15 07:17:48 | gene    | mars    | gene    | saturn  | 3824  |
| 2006-05-15 08:50:57 | phil    | venus   | phil    | venus   | 978   |
| 2006-05-15 10:25:52 | gene    | mars    | tricia  | saturn  | 998532 |
| 2006-05-15 17:35:31 | gene    | Saturn   | gene    | mars    | 3856  |
| 2006-05-16 09:00:28 | gene    | venus   | barb    | mars    | 613   |
| 2006-05-16 23:04:19 | phil    | venus   | barb    | venus   | 10294 |
| 2006-05-17 12:49:23 | phil    | mars    | tricia  | saturn  | 873   |
| 2006-05-19 22:21:51 | gene    | saturn  | gene    | venus   | 23992 |
+-----+-----+-----+-----+-----+-----+
```

你能够使用18.3节中讨论的技术，获取表的内容并把它显示为一个HTML表格。这儿我们使用相同的概念，但作些小调整以生成“点击排序”的列头。

一个“空白的”HTML表格包括一组列头，该列头仅包含列的名字：

```
<tr>
  <th>t</th>
  <th>srcuser</th>
  <th>srchost</th>
  <th>dstuser</th>
  <th>dsthost</th>
  <th>size</th>
</tr>
```

要生成超链接的列头，且该链接能够重新调用脚本以生成一个按给定列名排序的显示，我们需要生成一个如下的标题行：

```
<tr>
<th><a href="script_name?sort=t">t</a></th>
<th><a href="script_name?sort=srcuser">srcuser</a></th>
<th><a href="script_name?sort=srchost">srchost</a></th>
<th><a href="script_name?sort=dstuser">dstuser</a></th>
<th><a href="script_name?sort=dsthost">dsthost</a></th>
<th><a href="script_name?sort=size">size</a></th>
</tr>
```

要生成这样的标题，脚本需要知道表格中每个列的名字，以及它自己的URL。9.6节和19.1节说明了如何运用语句的元数据和脚本环境来获取这样的信息。例如，在PHP中，只需使用以下的一个语句，脚本便可给所有的列生成标题行。该语句中`tableInfo()`返回的是一个包含元数据的数组。`$info[i]`包含了*i*列的信息，`$info[i]["name"]`包含了列的名字：

```
$self_path = get_self_path ();
print ("<tr>\n");
$info =& $conn->tableInfo ($result, NULL);
if (PEAR::isError ($info))
    die (htmlspecialchars ($result->getMessage ()));
for ($i = 0; $i < $result->numCols (); $i++)
{
    $col_name = $info[$i]["name"];
    printf ("<th><a href=\"%s?sort=%s\"%s</a></th>\n",
        $self_path,
        urlencode ($col_name),
        htmlspecialchars ($col_name));
}
print ("</tr>\n");
```

接下来的脚本clicksort.php，实现了这种类型的表格显示。它检查环境以获取一个`sort`参数，该参数表明用哪一行来排序。脚本使用该参数来构建一个如下的语句：

```
SELECT * FROM $tbl_name ORDER BY $sort_col LIMIT 50
```

这类脚本有一个小的初始化问题。你第一次调用该脚本时，环境中没有用于排序的列名，所以该脚本并不明白该使用哪一列来做初始化的排序。该怎么办？有多个解决办法：

- 获取非排序的结果。
- 把某个列名捆绑到代码中。
- 在INFORMATION_SCHEMA中查找列名，然后使用其中的一个作为默认值：

```
SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = "cookbook" AND TABLE_NAME = "mail"
AND ORDINAL_POSITION = 1;
```

下面的脚本从INFORMATION_SCHEMA中查找名字。在获取结果时，它还使用了一个LIMIT子句以防止脚本向表格输出太大的数据量：

```
<?php
# clicksort.php - 用“点击以排序”列头部以HTML table的形式显示查询结果

# 数据库表中的行以一个HTML table显示。
# 列头部作为超链接出现，这个链接可以反复触发脚本根据相应列排序来重新显示表格。
# 在表很大的情况下显示被限制为50行。

require_once "Cookbook.php";
require_once "Cookbook_Webutils.php";

$title = "Table Display with Click-To-Sort Column Headings";

?>

<html>
<head>
<title><?php print ($title); ?></title>
</head>
<body bgcolor="white">

<?php
# -----
# 数据库，表，以及默认排序列的名称；根据需要修改。
$db_name = "cookbook";
$tbl_name = "mail";

$conn =& Cookbook::connect ();
if (PEAR::isError ($conn))
    die ("Cannot connect to server:
        . htmlspecialchars ($conn->getMessage ()));

print ("<p>" . htmlspecialchars ("Table: $db_name.$tbl_name") . "</p>\n");
print ("<p>Click on a column name to sort by that column.</p>\n");

# 获得要排序的列名：如果没有，使用第一列。

$sort_col = get_param_val ("sort");
if (!isset ($sort_col))
{
    $stmt = "SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
        WHERE TABLE_SCHEMA = ? AND TABLE_NAME = ?
        AND ORDINAL_POSITION = 1";
    $result = & $conn->query ($stmt, array ($db_name, $tbl_name));
    if (PEAR::isError ($result))
        die (htmlspecialchars ($result->getMessage ()));
    if (!(list ($sort_col) = $result->fetchRow ()))
        die (htmlspecialchars ($result->getMessage ()));
    $result->free ();
}
```

```

}

# 构建查询从表中选取记录，根据命名列来进行排序。(列名来源于环境，所以引用它以避免SQL
# 注入攻击。)
# 将输出限制为50行以避免堆放大数据表的完整内容。

$stmt = "SELECT * FROM $db_name.$tbl_name";
$stmt .= " ORDER BY " . $conn->quoteIdentifier ($sort_col);
$stmt .= " LIMIT 50";

$result =& $conn->query ($stmt);
if (PEAR::isError ($result))
    die (htmlspecialchars ($result->getMessage ()));

# 以HTML table的形式显示查询结果。使用查询元数据来获得列名，并在表的第一行将名称以
# 链接来进行显示，该链接会引发表根据相应表列排序以重新显示。

print ("<table border=\"1\">\n");
$self_path = get_self_path ();
print ("<tr>\n");
$info =& $conn->tableInfo ($result, NULL);
if (PEAR::isError ($info))
    die (htmlspecialchars ($result->getMessage ()) );
for ($i = 0; $i < $result->numCols (); $i++)
{
    $col_name = $info[$i]["name"];
    printf ("<th><a href=\"%s?sort=%s\"%s</a></th>\n",
           $self_path,
           urlencode ($col_name),
           htmlspecialchars ($col_name));
}
print ("</tr>\n");
while ($row =& $result->fetchRow ())
{
    print ("<tr>\n");
    for ($i = 0; $i < $result->numCols (); $i++)
    {
        # 编码值，对于空的表格使用 ;
        $val = $row[$i];
        if (isset ($val) && $val != "")
            $val = htmlspecialchars ($val);
        else
            $val = " ";
        printf ("<td>%s</td>\n", $val);
    }
    print ("</tr>\n");
}
$result->free ();
print ("</table>\n");

$conn->disconnect ();

?>

```

```
</body>
</html>
```

\$sort_col的值来自于环境的sort参数，所以它应被认为是危险的：一个攻击者也许会提交一个带sort参数的URL，该参数是经过精心设计的，能触发SQL注入攻击。要避免这种情况，当你构建SELECT语句来获取行时，\$sort_col应该被引用。你不能使用一个占位符去引用该值，因为这种技术只适用于数据值。有些MySQL API，比如PEAR DB提供了一种标识-引用函数。clicksort.php脚本使用了quoteIdentifier()方法来确保\$sort_col被安全地放入SQL语句中。

另外一种验证列名的方法是检查INFORMATION_SCHEMA的COLUMNS表。如果排序列没有在这里列出来，那么它就是非法的。clicksort.php脚本并不实现那一点。然而，光盘中包含一个Perl的对应脚本，clicksort.pl执行了这种检查。如果你想了解更多信息的话可以参考。

在标题头行之后的行包含许多小格，这些格中存放的是来自数据库表的数据值，在这里被显示为静态文本。空格子中的是 ，这是因为空格子和非空格子有相同的边框。

19.12 Web页面访问计数

Web Page Access Counting

问题

你希望计算一个页面被访问的次数。

解决方案

实现一个点击计数器，以你想要计数的页面为关键字。这可以用来在页面上显示一个计数器。相同的技术还可以用来记录其他类型的信息，例如一组标语广告中每个标语被使用的次数。

讨论

本节讨论访问技术，并在下面的例子中使用点击计数器。计数器显示的是一个Web页面被访问的次数。计数器现在并不是那么显眼了，这是因为现在很多页面制作者发现大多数访问者并不真的关心一个页面的流行程度。在不少上下文中这种普遍的观念仍有应用。例如，如果你在页面中显示一个标语广告，你也许会根据你提供广告的访问次数向广告主收费。要做到这一点，你需要计算每个标语的访问次数。你可以将本节的讨论用于这个目的。

要显示一个页面被访问的次数，有好几种方法。最基本的方法是维护一个文件计数。当一

个页面被请求时，打开文件，读入计数，加一后把新的计数写回文件中，并把它显示在页面上。这种方法的好处是容易实现，缺点是对于每个页面，它需要一个计数文件，而且在两个客户同时访问一个页面时，它无法正常工作，除非你在文件访问过程中显式地运行某种加锁协议。在一个文件中保持多个计数器可以减少计数文件的数目，但这将让访问文件中特定的值变得更加困难，而且它还是无法解决同时访问的问题。事实上，这种方法只会更糟。因为相比于单计数器文件，多计数器的文件有更高的可能性被多个客户端同时访问，所以为了处理文件内容，你将运行存储及获取方法，以及加锁协议来保证多个处理彼此之间不干扰。嗯…这听起来真类似于诸如MySQL这样的数据库管理系统已经花心思处理的问题。在数据库中保持计数，并把这些计数都放入一个表中，SQL提供了存储和读取的接口，同时加锁问题也不存在了，因为MySQL能够线性化所有多表的访问，因此客户端之间也就不可能相互干扰了。进一步地，根据你管理计数器的方式，你或许需要更新计数器并使用一个语句来获取新的序列值。

我假设你想记录超过一个页面的点击次数。要做到这点，需要创建一个表格，每个页面在表格中都有对应的一行。这意味着，每页都需要一个唯一的标识符，因此针对不同页的计数就不会混淆。你可以设置标识，但使用页面在你Web服务器目录树下的路径作为标识会简单得多。Web编程语言中一般都极容易获得路径信息。事实上，我们在19.1节中已经讨论了获取路径的方法。在这个基础上，你再创建一个点击计数表格：

```
CREATE TABLE hitcount
(
    path VARCHAR(255)
        CHARACTER SET latin1 COLLATE latin1_general_cs NOT NULL,
    hits BIGINT UNSIGNED NOT NULL,
    PRIMARY KEY (path)
);
```

该表格的定义涉及到一些假设：

- Path列存储的是页面的路径，它有一个latin1和区分大小写的latin1_general_cs校验的字符组。在路径信息是区分大小写的Web开发环境中使用latin1_general_cs校验的字符组是非常合适的，这对大多数版本的Unix更是如此。对于Windows或Mac OS X下的HFS+文件系统而言，文件名不区分大小写，所以你需要选择一种不区分大小写的校验，比如latin1_swedish_ci。如果你的文件系统是使用一种不同的字符集来启动的，你应该改变字符集和校验集。

- 路径列最长255字符，这限制了你页面路径的长度。
- 路径列使用PRIMARY KEY作索引，它需要保证唯一性。PRIMARY KEY或者UNIQUE索引是必需的，因为我们是使用带ON DUPLICATE KEY UPDATE子句的INSERT语句来执行点击-计数算法。当路径在表格中不存在时，我们插入新行，否则更新已有行。（11.14节提供更多的背景信息以帮助你了解ON DUPLICATE KEY UPDATE。）
- 该表格用来给一个文档树中的页面计数，就好像你的Web服务器用来为一个域下的文档服务一样。如果你在某台支持多域名的主机上运行一个点击计数机制，你会需要添加一个域名列。这个值可以通过SERVER_NAME来获得，Apache已经把它放入了你脚本的执行环境中。在这种情况下，计数表索引应该包含主机名和页面路径两部分。

点击计数维护的一般逻辑是为每一页的对应行增加计数，然后获取更新后的计数值：

```
UPDATE hitcount SET hits = hits + 1 WHERE path = 'page path';
SELECT hits FROM hitcount WHERE path = 'page path';
```

不幸的是，如果你使用这种方法，你可能经常得不到正确的值。如果几个客户端同时请求相同的页面，那么几个UPDATE语句会在很接近的时间内触发。下面的SELECT语句就不会获得相应的点击值，但这是能够避免的。使用事务或锁定点击计数表格都行，不过这两种做法都会减缓计数的过程。MySQL提供了一种解决方案，它允许每个客户端获取它自己的计数，而不用管同时发生的其他的更新：

```
UPDATE hitcount SET hits = LAST_INSERT_ID(hits+1) WHERE path = 'page path';
SELECT LAST_INSERT_ID();
```

更新计数的基础是LAST_INSERT_ID(expr)，这已在11.14节中详细讨论。UPDATE语句查找相关的行，并增加它的计数器值。使用LAST_INSERT_ID(hits+1)而不是hits+1来告诉MySQL把该值当作AUTO_INCREMENT来对待。这允许它在第二个使用LAST_INSERT_ID的语句中被使用。函数LAST_INSERT_ID返回一个跟链接相关的值，所以你永远都得到相同链接发出的UPDATE的值。此外，SELECT语句不必查询一个表，所以它的执行速度很快。

然而，还有一个问题。如果该页在hitcount表中没有列呢？在这种情况下，UPDATE语句找不到一行来修改，而你会获得一个为0的计数。你可以要求任何一个被计数的页面，在发布之前都必须在hitcount表格中注册，从而解决这个问题。一个更容易的办法是使用MySQL的INSERT ... ON DUPLICATE KEY UPDATE语法，如果指定行不存在的话，它会在hitcount表中插入相应记录且把counter置为1：

```
INSERT INTO hitcount (path,hits) VALUES('some path',LAST_INSERT_ID(1))
ON DUPLICATE KEY UPDATE hits = LAST_INSERT_ID(hits+1);
```

计数器的值可以通过LAST_INSERT_ID()函数来获得：

```
SELECT LAST_INSERT_ID();
```

你首次请求某页面的计数时，语句在表中插入一行，因为该页在表中还不存在记录。语句创建了一个新的计数，然后返回1。对每一个后续的请求，语句用新的计数更新已有行。这样，Web页面设计者能够在页面中包含计数器，而无需事先为每个页面在hitcount表中插入一行。

通过整个去掉SELECT语句，我们还可以进一步提高效率。如果你的API提供了一种直接获取最近序列号的方法，那么去掉SELECT语句是可行的。例如，在Perl中，你可以仅使用一个SQL语句来更新计数并获得新的值：

```
$dbh->do ("INSERT INTO hitcount (path,hits) VALUES(?,LAST_INSERT_ID(1))
    ON DUPLICATE KEY UPDATE hits = LAST_INSERT_ID(hits+1)",
    undef, $page_path);
$count = $dbh->{mysql_insertid};
```

要让计数机制更易于使用，你可以把这些代码放入一个辅助函数中，该函数以页面路径为参数，并返回计数值。在Perl中，一个点击计数函数可以是这样的：

```
sub get_hit_count
{
my ($dbh, $page_path) = @_;
my $count;

$dbh->do ("INSERT INTO hitcount (path,hits) VALUES(?,LAST_INSERT_ID(1))
    ON DUPLICATE KEY UPDATE hits = LAST_INSERT_ID(hits+1)",
    undef, $page_path);
$count = $dbh->{mysql_insertid};
return $count
}
```

CGI.pm脚本script_name()函数返回URL的本地部分，所以你可以这样使用get_hit_count()：

```
my $count = get_hit_count ($dbh, script_name ());
print p ("This page has been accessed $count times.");
```

计数更新机制涉及到一个简单的SQL语句，所以不必使用事务或显示的表格锁定来防止竞争情况，这种情况在多个客户同时请求一个页面时会出现。

Ruby版本的点击计数代码如下所示：

```
def get_hit_count(dbh, page_path)
  dbh.do("INSERT INTO hitcount (path,hits) VALUES(?,LAST_INSERT_ID(1))
    ON DUPLICATE KEY UPDATE hits = LAST_INSERT_ID(hits+1)",
```

```
    page_path)
  return dbh.func(:insert_id)
end
```

如下使用计数方法：

```
self_path = ENV["SCRIPT_NAME"]
count = get_hit_count(dbh, self_path)
page << cgi.p { "This page has been accessed " + count.to_s + " times." }
```

在Python中，计数函数如下：

```
def get_hit_count (conn, page_path):
  cursor = conn.cursor ()
  cursor.execute ("""
    INSERT INTO hitcount (path,hits) VALUES (%s, LAST_INSERT_ID(1))
    ON DUPLICATE KEY UPDATE hits = LAST_INSERT_ID(hits+1)
    """, (page_path,))
  cursor.close ()
  return (conn.insert_id ())
```

可如下使用该函数：

```
self_path = os.environ["SCRIPT_NAME"]
count = get_hit_count (conn, self_path)
print "<p>This page has been accessed %d times.</p>" % count
```

光盘中包括了Perl、Ruby、PHP和Python的点击计数脚本，它们放在apache/hits目录下。一个JSP版本放在了tomcat/mcb目录下。把这些安装在你的Web目录树下，通过你的浏览器调用几次，观察计数的增加。首先，你需要创建一个hitcount表格，像在19.13节中介绍的hitlog表格那样。（点击计数脚本显示一个计数，以及一个最近访问计数的日志。19.13节讨论了日志机制）这两个表都能使用hits.sql脚本创建，该脚本放在tables目录下。

19.13 Web页面访问日志

Web Page Access Logging

问题

你想知道一个页面除了被访问次数之外更多的信息，比如访问时间和发出请求的主机。

解决方案

维持一个点击日志而不是一个简单的计数。

讨论

在19.12节中使用的hitcount表仅记录了在表中注册的每个页面的访问次数。如果你想记录

关于页面访问的其他信息，得使用另一种方法。假设你想跟踪每个请求的客户端主机和访问时间，你需要为每次页面访问记录一行信息而不仅仅是一个计数。但你可以使用绑定了路径和AUTO_INCREMENT序列的多列索引来保持计数：

```
CREATE TABLE hitlog
(
    path  VARCHAR(255)
        CHARACTER SET latin1 COLLATE latin1_general_cs NOT NULL,
    hits  BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    t     TIMESTAMP,
    host  VARCHAR(255),
    INDEX (path,hits)
) ENGINE = MyISAM;
```

请查阅19.12节为path列选择字符集和校验集部分。要在hitlog表中插入新的列，使用如下语句：

```
INSERT INTO hitlog (path, host) VALUES(path_val,host_val);
```

例如，在一个JSP页面中，点击可以记录成如下这样：

```
<c:set var="host"><%= request.getRemoteHost () %></c:set>
<c:if test="${empty host}">
    <c:set var="host"><%= request.getRemoteAddr () %></c:set>
</c:if>
<c:if test="${empty host}">
    <c:set var="host">UNKNOWN</c:set>
</c:if>

<sql:update dataSource="${conn}">
    INSERT INTO hitlog (path, host) VALUES(?,?)
    <sql:param><%= request.getRequestURI () %></sql:param>
    <sql:param value="${host}" />
</sql:update>
```

hitlog表有如下有用的属性：

- 当你插入新行时，TIMESTAMP类型的t列自动记录了访问时间。
- 把path列链接到AUTO_INCREMENT类型的列hits上，给定一个页面的计数值，它将在你插入新行时自动增长。每个不同的路径值都分别对应一个维护的计数值。这种计数机制要求你使用MyISAM（或者BDB）存储引擎，这就是为什么表格定义中包含一个显示的ENGINE=MyISAM子句。（关于多列序列如何工作的更多信息，请看11.11节。）
- 没有必要去检查某个页面的计数器是否存在，因为你每记录一次页面的点击时，都会插入一个新的行，而不仅仅是第一次点击时才这样。
- 要确定每个页面的当前计数值，根据不同的path值来选取拥有最大hits值的行：

```
SELECT path, MAX(hits) FROM hitlog GROUP BY path;
```

要获知一个给定页面的计数，可执行以下语句：

```
SELECT MAX(hits) FROM hitlog WHERE path = 'path_name';
```

19.14 使用MySQL存储Apache日志

Using MySQL for Apache Logging

问题

你期望仅使用MySQL来为一些页面记录访问日志如同19.13节中所示。你期望记录下所有页面的访问日志，而且不想把日志代码显式地放进每一个页面中。

解决方案

告诉Apache把页面访问日志存入MySQL表格中。

讨论

在Web环境下MySQL的使用并不仅限于页面的生成和处理。你还可以用MySQL来帮助你运行Web服务器。举个例子，大多数Apache服务器都设置为对每个页面的页面请求记一条日志。但把日志记录发送给一个程序也是可能的，这样你就可以在任何你喜欢的地方写下记录——比如写入数据库中。把日志记录放入数据库而不是一个扁平的文件中，日志会变得非常结构化，同时你还可以应用SQL分析的各种技术。日志文件分析工具，也许会能提供一定的可扩展性，但这经常是一个关于哪些总结需显示，而哪些需压缩的问题。更困难的是告诉一个工具，让它显示它自己不提供的信息。把日志条目放入表格中，你可以获得更多的可扩张性。想看到一种特定的报告吗？请编写生成它的SQL语句。要以一种特定的格式显示该报告，你可以把SQL语句放入API中，然后利用你脚本语言的输出特性。

通过使用分开的过程来处理日志条目的生成和存储，你就获得了额外的可扩张性。还有些情况可能是，把日志从多个Web服务器发送给一个MySQL服务器，或者把一个Web服务器生成的不同日志发送给不同的MySQL服务器。

这里介绍如何把Web请求日志从Apache传送到MySQL，并示范一些你可能会觉得有用的整体查询。

设立数据库日志

Apache日志是由httpd.conf配置文件中的指示性语句控制的。例如，一个典型日志调整使用了LogFormat和CustomLog指示性语句：

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog /usr/local/apache/logs/access_log common
```

LogFormat这一行给日志记录定义了一种格式，并给了它一个昵称common。CustomLog表明每行都需要按照这种格式写入Apache的日志目录下的access_log文件中。而要把日志放入MySQL，需要使用下面的过程（注1）：

1. 决定你要记录什么，构建一个表以包含合适的列。
2. 编写一个程序，从Apache中读入日志各行，并把它们写入数据库。
3. 写一行LogFormat，定义以何种形式写日志，以及一行CustomLog告诉Apache把日志写入程序而不是某个文件中。

假设你想记录每个请求的日期、时间、发出请求的主机、请求中使用的方法、URL路径名、状态代码、传输的字节数、相关的页面及用户代理（一般是浏览器或蜘蛛的名字）。一个包含上述列的表为：

```
CREATE TABLE httpdlog
(
    dt      DATETIME NOT NULL,          # 请求日期
    host    VARCHAR(255) NOT NULL,       # 客户端主机
    method  VARCHAR(4) NOT NULL,         # 请求方法 (GET, PUT, 等等)
    url     VARCHAR(255)                # URL路径
            CHARACTER SET latin1 COLLATE latin1_general_cs NOT NULL,
    status   INT NOT NULL,              # 请求状态
    size     INT,                      # 传输的字节数
    referrer VARCHAR(255),             # 参考页面
    agent    VARCHAR(255)              # 用户代理
);
```

大部分的字符列使用了VARCHAR且不区分大小写。唯一的例外是url列，它被声明为区分大小写校验的，这对于运行在区分大小写的系统上的Web应用是恰当的选择。请查看19.12节以了解如何为path列选择字符集和校验集。

httpdlog表格的定义中不包含任何索引。如果你想运行汇总查询，那你应该往表格中添加合适的索引。否则，在表格不断变大时，汇总过程将非常的慢。选择使用哪一列来构建索引将取决于你编写的用于分析表格内容的SQL语句。举个例子，在host列上创建索引，将让分析客户端主机值分布的语句受益。

下一步，你需要一个程序来处理Apache生成的各行日志，并把它们插入httpdlog表中。下面的脚本httpdlog.pl打开一个与MySQL服务器之间的链接，并循环读取输入文件的各行。

注1：如果你正使用诸如TransferLog等日志指令而非LogFormat和CustomLog，你需要调整本节的指令。

它将每一行解析为多个列，并把这些列值插入数据库中。当Apache退出时，它关闭了通往日志程序的管道。这导致httpdlog.pl在它的输入中看到了文件的结束，从而中止循环，断开到MySQL的链接，然后退出：

```
#!/usr/bin/perl
# httpdlog.pl - 将Apache请求记录到httpdlog表

# 包含Cookbook.pm的目录路径（根据需要改变）
use lib qw(/usr/local/lib/mcb);
use strict;
use warnings;
use Cookbook;

my $dbh = Cookbook::connect ();
my $sth = $dbh->prepare (qq{
    INSERT DELAYED INTO httpdlog
        (dt,host,method,url,status,size,referer,agent)
    VALUES (?,?,?,?,?,?);
});

while (<>) # 循环读取输入
{
    chomp;
    my ($dt, $host, $method, $url, $status, $size, $refer, $agent)
        = split (/\\t/, $_);
    # 对于某些列将"--"匹配为NULL
    $size = undef if $size eq "--";
    $agent = undef if $agent eq "--";
    $sth->execute ($dt, $host, $method, $url,
                   $status, $size, $refer, $agent);
}
$dbh->disconnect ();
```

把httpdlog.pl脚本安装在你希望Apache查看的地方。在我的系统上，Apache根目录是/usr/local/apache，所以/usr/local/apache/bin是一个合适的目录。这个目录的路径很快就会被用来构建CustomLog指示语句，该语句告诉Apache如何把日志发给脚本。

包含use lib行的目的，是让Perl找到Cookbook.pm模块。如果被Apache调用的脚本的运行环境不允许Perl查找该模块，那么这一行就是必需的。必要的话，你可以根据自己的系统来改变路径。

这个脚本使用了INSERT DELAYED而不是INSERT。使用DELAYED的优点是MySQL服务器把行缓存在内存中，然后一次性地插入一堆行，这样做更有效。这也允许客户端立刻继续操作，而不是在表格繁忙时等候着。不足之处是如果MySQL服务器崩溃了，那么在内存中缓存的所有行都丢失了。我得指出：这样的可能性并不大，而且一些日志记录的丢失并非一个严重的问题。当然如果你不同意，那就去掉DELAYED。

脚本httpdlog.pl假设输入的行包含httpdlog列值，且该值以TAB为界（这让拆分输入的各行变得容易），因此Apache必须以一种匹配的格式编写日志条目。生成在下列表中显示的合适的值的LogFormat域标识如下：

标识	含义
%{%Y-%m-%d %H:%M:%S}t	请求的日期和时间，使用MySQL的DATETIME格式
%h	发出请求的主机
%m	请求的方法（get, post等等）
%U	URL路径
%>s	状态代码
%b	传输的字节数
%{Referer}i	被引用的页面
%{User-Agent}i	用户代理

要定义一个名为mysql的日志格式，该格式使得两个值之间有TAB，那么可以把下面一段LogFormat指示语句加入你的httpd.conf文件中：

```
LogFormat \
" %{%Y-%m-%d %H:%M:%S}t\t%h\t%m\t%U\t%>s\t%b\t%{Referer}i\t%{User-Agent}i" \
mysql
```

现在大部分条件都已具备。我们有一个log表，一个往表中写东西的程序，一个生成日志条目的mysql格式。剩下来的就是告诉Apache往httpdlog.pl中写入条目了。然而，直到你明白输出格式确实正确，以及程序能正确生成条目之后，告诉Apache往程序中直接写入日志才是可靠的做法。为了让测试和调试更简单些，让Apache把mysql格式的条目先写入一个文件。这样，你就能检查该文件以确认输出的格式，然后再将它作为httpdlog.pl的输入以验证程序是否正确工作。让Apache以mysql格式向log目录中名为test_log的文件中输入日志，你可以这样：

```
CustomLog /usr/local/apache/logs/test_log mysql
```

然后，重启Apache以运行新的日志指示语。在你的Web服务器接受到请求时，看看test_log文件，确保其中的内容正像你所期望的那样，然后把文件发给httpdlog.pl：

```
% /usr/local/apache/bin/httpdlog.pl test_log
```

在httpdlog.pl结束后，看看httpdlog表以确保它是正确的。一旦你满意了，告诉Apache把日志条目直接发送给httpdlog.pl：

```
CustomLog "|/usr/local/apache/bin/httpdlog.pl" mysql
```

在 pathname 最前方的 | 字符告诉 Apache httpdlog.pl 是一个程序，而不是一个文件。重启 Apache 之后新的条目就应该出现在 httpdlog 表中了。

迄今为止，你并没有改变原先做日志的方式。举个例子，如果你原来把日志放入 access_log 文件中，你现在还是这么做的。也就是说，Apache 把日志条目既发送给了原先的日志文件，又发送给了 MySQL。如果这就是你想要的，Apache 并不关心你往多个目的地存入日志。如果你需要更多的磁盘空间，请关闭文件日志，把你原先的 CustomLog 指示标语注释掉，在前面加上 # 即可实现。

分析日志文件

现在你已经让 Apache 把日志存入数据库中，那么你将如何处理这些信息呢？这取决于你想知道什么。这里有些例子，能回答某些使用 MySQL 时常见的问题：

- 请求日志中有多少行？

```
SELECT COUNT(*) FROM httpdlog;
```

- 多少个不同的客户端发送了请求？

```
SELECT COUNT(DISTINCT host) FROM httpdlog;
```

- 客户端请求了多少个不同的页面？

```
SELECT COUNT(DISTINCT url) FROM httpdlog;
```

- 最受欢迎的 10 个页面是什么？

```
SELECT url, COUNT(*) AS count FROM httpdlog  
GROUP BY url ORDER BY count DESC LIMIT 10;
```

- 有些浏览器喜欢检查 favicon.ico 文件，那么这个文件接收了多少次请求？

```
SELECT COUNT(*) FROM httpdlog WHERE url LIKE '%/favicon.ico%';
```

- 日志中的日期跨度是多长？

```
SELECT MIN(dt), MAX(dt) FROM httpdlog;
```

- 每天收到的请求数量是多少？

```
SELECT DATE(dt) AS day, COUNT(*) FROM httpdlog GROUP BY day;
```

回答这些问题，要求我们从 dt 值中剥离 time-of-day 部分的内容，这样在给定日期里收到的请求就能组织起来了。上述语句使用 DATE() 函数，把 DATETIME 值转化为 DATE 值，从而达到这个目的。然而，如果你打算运行许多语句，这些语句都只使用 dt 值的日期部分，构建 httpdlog 表格时使用分开的 DATE 和 DATETIME 列会比较合适，这样还需要改变 LogFormat 指示语句以生成分开的日期和时间，当然还需要调整一下 httpdlog.pl 脚本。

然后你就能直接在日期上执行操作，而无需再剥离什么部分，同时你还可以在日期列上构建索引以提高性能。

- 每小时请求的柱状图是什么？

```
SELECT HOUR(dt) AS hour, COUNT(*) FROM httpdlog GROUP BY hour;
```

- 平均每天接收的请求是多少？

```
SELECT COUNT(*) / (DATEDIFF(MAX(dt), MIN(dt)) + 1)  
FROM httpdlog;
```

这里分子是表中所有的请求数，分母是记录跨越的天数。

- 表中最长的URL是什么？

```
SELECT MAX(LENGTH(url)) FROM httpdlog;
```

如果url列被定义为VARCHAR(255)并且上述语句的执行结果是255，那么很可能某些URL实在太长了以至于需要被截断一部分才能放进表中。为避免这点，可以更改列的定义，允许更多的字符。例如，允许url列以容纳最多5 000字符：

```
ALTER TABLE httpdlog  
MODIFY url VARCHAR(5000)  
CHARACTER SET latin1 COLLATE latin1_general_cs NOT NULL;
```

- 发送的全部字节数是多少，平均每个请求发送的字节数是多少？

```
SELECT  
    COUNT(size) AS requests,  
    SUM(size) AS bytes,  
    AVG(size) AS 'bytes/request'  
FROM httpdlog;
```

上述语句使用COUNT(size)而不是COUNT(*)来记录那些有非NULL-size值的请求数量，如果一个客户端请求一个页面两次，服务器可能会发送一个头以响应第二个请求，该头表明这个页面自你上次请求后还没有更新。在这种情况下，请求对应记录条目中size列为NULL。

- 每种类型的文件都有多少流量（基于文件后缀名.html,.jpg或.php）？

```
SELECT  
    SUBSTRING_INDEX(SUBSTRING_INDEX(url, '?', 1), '.', -1) AS extension,  
    COUNT(size) AS requests,  
    SUM(size) AS bytes,  
    AVG(size) AS 'bytes/request'  
FROM httpdlog  
WHERE url LIKE '%.%'  
GROUP BY extension;
```

WHERE子句仅选择有句点的url，这样就去除了指向没有后缀名的文件的路径。在输出列表中去除后缀，使用内调用SUBSTRING_INDEX()可以把URL右边跟参数有关的字符串都去掉。(这会把/cgi-bin/script.pl?id=43变为/cgi-bin/script.pl，如果没有参数部分，SUBSTRING_INDEX()返回的就是整个字符串)外调用SUBSTRING_INDEX()将把除扩展名之外的所有东西都去除。

其他日志问题

前面讨论了一种简单的能把Apache链接到MySQL上的方法，该方法涉及到编写一小段脚本，该脚本与MySQL通信并告诉Apache向它写入日志，而不是向文件。如果你把所有请求都写入一个文件，那么这种方法会工作得非常好，但对于每一个可能的配置而言它当然不可能总是合适的。例如，你在httpd.conf文件中定义了虚拟服务器，你或许为每个服务器都单独定义了CustomLog指示。要把内容全导入MySQL中，你能改变每一条指示语句，让它写向httpdlog.pl。但这样的话，在每个虚拟服务器上，你都分别有一个日志进程。这带来两个问题：

- 你如何将日志记录关联到恰当的虚拟服务器上？一种解决方案是，为每个服务器创建一个单独的日志表，并修改httpdlog.pl使其接收一个参数，该参数指明该使用哪个表。另一种解决方案是使用一个有vhost列的表，一个包含%v虚拟主机格式标识的Apache日志格式及一个日志脚本，该脚本在生成INSERT语句时使用vhost值。光盘中的目录apache/httpdlog下，包含了这么做的信息。
- 你真的需要一堆运行的httpdlog.pl处理进程吗？如果你有很多虚拟服务器，也许要考虑使用一个日志模块，该模块安装在Apache下。能仅使用一个通往数据库服务器的链接，把多虚拟主机的日志变得多元化，从而减少日志行为中的资源消耗。

往数据库中写日志，而不是往文件中写，允许你使用MySQL的全部力量以执行日志分析。但这不意味着你就不用考虑空间管理的问题。无论你把日志写入文件还是数据库，Web服务器能产生一大堆行为，并生成日志记录。那么，一种节省空间的办法是偶尔让日志过期。例如，删除一年前的日志记录：

```
DELETE FROM httpdlog WHERE dt < NOW() - INTERVAL 1 YEAR;
```

如果你有MySQL 5.1或更高版本，你可以设立一个事件，该事件按计划运行这个DELETE语句(参见16.7节)。

考虑到被日志活动消耗的磁盘空间，需注意的是如果你在MySQL中开启了查询日志，那么不仅每个请求会被写入httpdlog表，每个查询语句也会被写入日志文件。所以，你也许会发现磁盘空间以一种比你预计要快得多的速度在消失。因此，使用某些日志文件旋转或过期技术，会是个好主意。

使用基于MySQL的Web会话管理

Using MySQL-Based Web Session Management

20.0 引言

Introduction

许多万维网应用程序都是通过一系列请求与用户进行交互的，因此需要同时记住从一个请求到下一个请求的信息。一套相关的请求被称为一个会话。会话类似执行登录操作，通过后来的请求与已登录的用户发生联系，管理一个多步的在线订单过程，从某个时期的用户收集输入（组织之前用户反映的问题），以及从各访问中记录用户的偏好等行为非常有效。不幸的是，HTTP是一个无状态协议，这意味着万维网服务器将每一个请求都视为独立的——除非你采用其他步骤来确保这一点。

本章展示了如何让信息持续地通过多重请求，这将有助于你为一个保存之前信息记忆的信息开发应用程序。在这里展示的技术总的来说已经足够，你应该能够让它们适应多种需要保持状态的万维网应用程序。

会话管理问题

一些会话管理方法依赖于存储在客户端的信息。一个实现客户端存储的方法是使用cookies，这将实现特殊请求和响应报头中信息的前向后向传递。当开始一个会话时，应用程序将生成和发送包括被存储的初始信息的cookie。客户端在后续请求中将cookie返回给服务器以确认它本身，同时让应用程序能够与属于同一客户端会话的请求联系。在会话中的每一步，应用程序使用cookie中的数据来确认客户端的状态（状况）。为了改变会话的状态，应用程序将包括更新信息的cookie发送给客户端以替代旧的cookie。这个机制允许数据在请求之间

持续传递，为应用程序提供机会以更新必需的信息。cookies使用起来很容易，但是它仍然有些缺陷。例如，对客户端而言，它可以修改cookie的内容而导致应用程序产生错误的行为。其他的客户端会话存储的技术也面临同样的缺陷。

客户端存储的一个改进方法是在服务器端存储一个多请求会话的状态。在这个思想的指导下，客户端正在进行的操作的相关信息将被存储在服务器上的某个位置，如一个文件、共享内存或者是一个数据库中。在客户端仅有的信息就是在会话开始时由服务器生成并发送到客户端的唯一标识符。客户端在每次后续请求时都将该标识符发送给服务器，这样服务器可以将客户端与正确的会话联系起来。普通的跟踪会话ID的技术是将其存储在cookie中或者将其编码在请求的URLs中。（后者通常用于客户端禁止使用cookie时。）服务器可以以cookie数值的形式或者通过从URL中提取得到该ID。

服务器端会话存储比将信息存储在客户端更安全，因为应用程序维持了整个会话的内容的控制。唯一在客户端表现的数值就是会话ID，这样客户端不能修改会话数据除非应用程序允许它这么做。对客户端而言仍然有修改ID并发将其发送给服务器的可能性，但是如果ID都是唯一的并且是从一个非常巨大的候选数值池中挑选出来的，那么恶意的客户端几乎不可能能够猜测出另一个有效会话的ID。如果你担心其他客户端通过网络窥视窃取有效的会话ID，你应该建立一个安全连接（例如，使用SSL）。但是那超出了本书的范围。

管理会话的服务器端方法通常将会话内容存储在稳定的存储器中，例如文件或者数据库。数据库支持的存储与文件支持的存储具有不同的特性，例如你消除了众多会话文件导致的文件系统混乱，同时你可以使用同一个MySQL服务器来为多个网络服务器处理会话通信。如果这一点吸引了你，本章展示的技术将帮助你将基于MySQL的会话管理集成到你的应用程序中。本章展示了如何为我们的几个不同的API语言实现服务器端数据库支持的会话管理：

- 对Perl，Apache::Session模块包括你需要管理的会话能力的绝大部分。它能够将会话信息存储在文件或者任意的几个数据库系统之中，包括MySQL、PostgreSQL以及Oracle。
- 在Ruby中，CGI::Session类提供了默认使用临时文件的会话处理的能力。然而，该实现允许使用其他存储管理器。一个可能的方法就是mysql-session包，它允许会话通过MySQL存储。
- PHP包括直接的会话支持。它通过默认使用临时文件来实现，但是它却非常灵活，允

许应用程序为会话存储支持自己的处理程序。这让插入能够写入信息到MySQL中的存储模块成为可能。

- 对于运行在Tomcat网络服务器之中的基于Java的网络应用程序而言，Tomcat提供了服务器级的会话支持。你所需要做的全部事情就是修改服务器配置为会话存储使用MySQL。应用程序无需为获取该能力带来的优势而做额外的工作，所以在应用程序层没有任何改变。

会话使用不同的方法来支持不同的API。对于Perl，语言本身不提供会话支持，所以如果它想实现一个会话，其脚本必须包括明确的包含类似Apache::Session这样的模块。Ruby的实现方法与之类似。在PHP中，会话管理器是内建的。脚本无需特殊的准备就可以使用它，不过仅仅就是在它们想使用默认存储方法的时候，它是将会话信息存储在文件中。为了使用不同的方法（例如在MySQL中存储会话），应用程序必须提供其本身的程序给会话管理器使用。仍然有其他方法供运行在Tomcat之下的Java应用程序使用，因为Tomcat本身管理许多与会话管理相关的细节，包括存储会话数据的位置。而单个的应用程序不需要知道或关心相关信息被存储在什么地方。

本章中不讨论Python。我还没有找到我感觉适合在这里讨论的单独的Python会话管理模块，同时我也不想从零开始编写一个相应的模块。如果你正在编写一个需要会话支持的Python应用程序，你可能需要查看诸如Zope、WebWare或者Albatross等工具包。

抛开不同的实现方式不谈，典型的会话管理包含一套公共的任务：

- 确定客户端是否提供会话ID。如果答案是否定的，那么必须生成一个唯一的会话ID并把它发送给客户端。一些会话管理器会给出如何在服务器和客户端之间自动地传递会话ID。PHP就是这样做的，与JAVA程序的Tomcat类似。Perl的Apache::Session模块将其留给应用开发人员来管理ID传输。
- 将后续请求使用的数据存储到会话中同时获取之前请求放置到会话中的数据。这包括执行与会话数据相关的任何必需的行为：增加计数，确认登录请求，更新购物手推车等。
- 当会话不再被需要时终止它。一些会话管理器提供一项功能：当会话在一定时间不活动时就自动终止。如果请求明确指出终止会话，那么会话也会明确地被终止（例如客户端请求注销时）。作为响应，会话管理器会销毁会话记录。告知客户端释放信息可能也是必须的。如果客户端采用cookie的方式发送会话标识符，那么应用程序应该告知客户端丢弃cookie。否则，在cookie的用途结束之后客户端仍将持续提交它。另一个会

话“终止”的方法是从会话记录中删除所有信息。从效果上来说，这将导致会话在下一次客户端请求时重新开始，因为之前的会话信息都已经无效了。

会话管理器对应用程序在会话记录中存储的内容几乎没有约束。会话在通常情况下可以容纳任意相关的数据，例如标量、数组或者对象。为了让存储和获取会话数据更容易，会话管理器典型的做法是在存储数据之前通过将会话信息转换为编码的标量串来将其串行化，并且在获取它之后将其还原。在你提供存储引擎时，串行化字符串的相关操作不是你必须提供的功能。唯一必须确认的是存储管理器有足够的空间来容纳串行化字符串。为了使用MySQL实现备份存储，这意味着你需要使用BLOB数据类型中的一种。我们的会话管理器使用MEDIUMBLOB，它拥有最大16MB的存储空间来容纳会话记录。（当估计存储空间的需求时，记录需要存储的数据是被串行化的，这比原始数据需要更多的空间。）

本章下面的部分展示了每个API的基于会话的脚本。每个脚本执行两个任务。它包含一个指明当前会话收到请求的计数值。同时对每个请求记录一个时间戳。通过这种方法，脚本实现了如何存储和获取一个标量（计数值）和非标量（时间戳数组）。它们只需要很少的用户交互。你只是需要重载页面来指明下一个请求，这只需要非常简单的代码就能实现。

基于会话的应用程序常常包括一些实现用户明确注销和终止会话的方法。示例脚本实现了“注销”的一个形式，但它是基于一个隐式的技术：会话被给定了至多10个请求的限制。当你重调用这个脚本时，它会检查计数器以确认该限制是否已经达到，如果已经达到那么就销毁会话数据。产生的效果是会话数据将不会出现在下一个请求里，所以脚本会开始一个新的会话。

Perl、Ruby、PHP的示例会话脚本可以在章节部分的Apache目录下找到；PHP会话模块在lib库目录下；JSP示例在tomcat目录下。创建会话存储表的SQL脚本在tables目录下。如同这里表现的，会话表在cookbook数据库中被创建并通过同样的在本书其他地方使用的MySQL账户访问。如果你不想将会话管理行为与属于其他cookbook表的行为混淆，可以考虑为会话数据建立一个独立的数据库和MySQL账户。这一点对Tomcat特别真实，因为它的会话管理在应用程序级上出现。你可能不希望Tomcat服务器将信息存储到“你的”数据库中；如果是，那么给服务器指定一个属于它的数据库。

20.1 在Perl应用程序中使用基于MySQL的会话

Using MySQL-Based Sessions in Perl Applications

问题

你想要在Perl脚本中使用会话存储。

解决方案

Apache::Session模块提供方便的方法来使用几个不同的存储类型，包括基于MySQL的一个方法。

讨论

Apache::Session是一个很容易使用的Perl模块，它包括许多交叉的页面请求的状态信息。抛开名称，这个模块并不依赖于Apache，它同时可以用于非网页内容——例如，维持命令行脚本的多个交叉请求的持续状态。另一方面，Apache::Session不能处理任何的追踪会话ID相关的内容（将其发送给客户端以响应初始化请求，同时从后续请求中提取它）。这里展示的示例应用程序在客户端能够在使用cookie的假设下使用cookie来传递会话ID。

安装Apache::Session

如果你没有Apache::Session，你可以通过CPAN（访问<http://cpan.perl.org>）获取它。安装过程简单明了，当然Apache::Session需要的几个其他模块你可能需要在一开始就准备好。（当你安装Apache::Session时，它会告知你缺少哪些需要的模块。如果你使用cpan的Apache::Session命令来安装它，那么在安装好它之后需要注意设置好相关性。）当你把一切都安装好时，建立一张存储会话记录的表。表格的格式可以从MySQL的存储处理器文档得到，你可以使用如下命令来阅读该文档：

```
% perldoc Apache::Session::Store::MySQL
```

该表可以被放置到你喜欢的任意数据库中（这里将使用cookbook数据库）。默认情况下，表的名称是sessions，但是较新的Apache::Session的版本中可以由用户指定其名称。我们将使用perl_session作为下面结构的名称：

```
CREATE TABLE perl_session
(
    id      CHAR(32) NOT NULL,    # 会话标识符
    a_session MEDIUMBLOB,        # 会话数据
    PRIMARY KEY (id)
);
```

id数据列保存了会话标识符，这是一个由Apache::Session生成的32位的MD5 编码数值。a_session数据列以指定的字符串格式保存了会话数据。Apache::Session使用Storable模块来

串行化和解串行化会话数据。(Apache::Session::Store::MySQL文档指明了a_session是一个TEXT数据列，但是任何的BLOB或者TEXT数据类型都应该有足够的空间来完成存储预期的会话记录的工作。)

Apache::Session的接口

为了在脚本中使用perl_session表，可以访问MySQL相关的会话模块：

```
use Apache::Session::MySQL;
```

Apache::Session使用混合表来表示会话信息。可以使用Perl的捆绑技术将混合表的操作映射到被底层存储管理器使用的存储和获取方法上。因此，为了开启一段会话，你应该声明一个混合表变量并将其传递给捆绑接口。捆绑接口的另一个参数是会话模块的名称、会话ID及使用的数据库信息。有两种方法来指明数据库连接。一个方法是传递一个包含连接参数的混合表引用（如果你没有使用默认的会话表名称，那么还需要将其传递出去）：

```
my %session;
tie %session,
  "Apache::Session::MySQL",
  $sess_id,
  {
    DataSource => "DBI:mysql:host=localhost;database=cookbook",
    UserName => "cbuser",
    Password => "cbpass",
    LockDataSource => "DBI:mysql:host=localhost;database=cookbook",
    LockUserName => "cbuser",
    LockPassword => "cbpass",
    TableName => "perl_session"
  };
```

在这种情形下，Apache::Session使用这些参数来开启它自己与MySQL之间的连接，当你关闭或者销毁会话之后它会自动关闭。

另一个方法是传递一个已经打开的数据库连接的句柄（这里用\$dbh表示）：

```
my %session;
tie %session,
  "Apache::Session::MySQL",
  $sess_id,
  {
    Handle => $dbh,
    LockHandle => $dbh,
    TableName => "perl_session"
  };
```

如果你如刚才展示的那样将一个句柄传递给一个打开的连接，Apache::Session会在你关闭或者销毁会话之后保持其连接，因为它假设你可能在脚本的其他位置处于别的目的正在使用该句柄。当你完成工作之后，你应该亲自关闭它。

捆绑接口的参数\$sess_id表示会话标识符。它的数值要么是未定义的以开始一个新的会话，要么是一个表示已经存在的会话记录的ID。在后一种情形下，该数值应该与目前一些存在的perl_session表中id数据列的数值相匹配。

当会话被开启后，你可以访问其内容。例如，在开启一个新的会话后，你会想确定它的标识符以将其发送给客户端。该数值可以这样获取：

```
$sess_id = $session{$_SESSION_ID};
```

以下划线开始命名的会话混合表元素（例如_session_id）被Apache::Session保留作为内部使用。除此之外，你可以为你的存储会话数值自由选择名称。

为了在会话中存储标量，可以存储其值来实现。为了访问一个标量，可以直接查阅其数值。你可能包含如下的标量计数器数值，该计数器在生成新的会话时被初始化，之后显示其增减：

```
$session{count} = 0 if !exists ($session{count}); # 初始化计数器  
++$session{count}; # 增加计数器  
print "counter value: $session{count}\n"; # 打印其数值
```

为了将类似数组或者混合表的非标量数值保存到会话记录中，可以保存其引用：

```
$session{my_array} = \@my_array;  
$session{my_hash} = \%my_hash;
```

在这种情形下，在你关闭会话前对@my_array或%my_hash所做的改变将在会话计数器中得到反映。为了保存一份会话中数组或者混合表的拷贝这样在你修改原始信息时其数值也不会被修改，可以采用如下方式创建该拷贝的引用：

```
$session{my_array} = [ @my_array ];  
$session{my_hash} = { %my_hash };
```

为了获取非标量数值，可以重复引用存储在会话中的引用：

```
@my_array = @{$session{my_array}};  
%my_hash = %{$session{my_hash}};
```

为了在你完成工作时关闭会话，可将其传递给解捆绑接口：

```
untie (%session);
```

当你关闭一个会话时，如果你对其做了改变，Apache::Session将其保存到perl_session表中。这也将导致会话数值不可访问，所以在你完成所有的访问操作之前不要关闭会话。



提示：备注Apache::Session通知改变了“顶层”会话记录数值，但是可能检测到通过引用存储的数值的数字的改变(例如数组元素)。如果这是个问题，你可以通过在关闭会话时指定任意顶层会话元素的数值来强制Apache::Session保存会话。会话的ID一直保存在会话混合表中，因此下面的习惯用法提供了一个强制会话保存的方便的方法：

```
$session{$_SESSION_ID} = $session{$_SESSION_ID};
```

一个开启的会话与其说被关闭不如说被终止。这样做可以将相应的数据行从perl_session表中移除，因此它就不能再使用了：

```
tied (%session)->delete ();
```

一个范例应用程序

下面的脚本，sess_track.pl，是一个短小但是完全实现了使用会话的应用程序。它使用Apache::Session来保持追踪会话中请求数目、每个请求的时间，以及更新和显示每次它被调用时的信息。sess_track.pl使用名为PERLSESSID的cookie来传递会话ID。这通过CGI.pm cookie管理接口的方法来完成（注1）。

```
#!/usr/bin/perl
# sess_track.pl - 会话请求计数/时间戳演示

use strict;
use warnings;
use CGI qw(:standard);
use Cookbook;
use Apache::Session::MySQL;

my $title = "Perl Session Tracker";

my $dbh = Cookbook::connect();           # 连接MySQL
my $sess_id = cookie ("PERLSESSID");    # 会话 ID (如果是新会话则为未定义)
my %session;                            # 会话混合表
my $cookie;                             # 发送到客户端的cookie

# 开启会话

tie %session, "Apache::Session::MySQL", $sess_id,
{
    Handle => $dbh,
    LockHandle => $dbh,
    TableName => "perl_session"
};
if (!defined ($sess_id))                # 新的会话
```

注1：关于CGI.pm的cookie支持信息，可以通过使用下面的命令以及阅读描述cookie()函数的章节：
%perldoc CGI。

```
{  
    # 获得新的会话ID，初始化会话data，建立cookie  
    $sess_id = $session{_session_id};  
    $session{count} = 0;          # 初始化计数器  
    $session{timestamp} = [ ];     # 初始化时间戳数组  
    $cookie = cookie (-name => "PERLSESSID", -value => $sess_id);  
}  
  
# 增加计数器并将当前时间戳加入到时间戳列表  
  
++$session{count};  
push (@{$session{timestamp}}, scalar (localtime (time ())));  
  
# 构建页面主体内容  
  
my $page_body =  
    p ("This session has been active for $session{count} requests.")  
    . p ("The requests occurred at these times:")  
    . ul (li ($session{timestamp}));  
  
if ($session{count} < 10) # 关闭（并保存）会话  
{  
    untie (%session);  
}  
else                      # 10次调用后销毁会话  
{  
    tied (%session)->delete ();  
    # 重置cookie来告知浏览器丢弃会话 cookie  
    $cookie = cookie (-name => "PERLSESSID",  
                      -value => $sess_id,  
                      -expires => "-1d");    # "过期时间为昨天"  
}  
  
$dbh->disconnect ();  
  
# 生成输出页面；如果定义了cookie则在输出页面头部将其包括  
  
print  
    header (-cookie => $cookie)  
        . start_html (-title => $title, -bgcolor => "white")  
        . $page_body  
        . end_html ();
```

通过将其安装到你的cgi-bin目录中并从你的浏览器发送请求来尝试运行该脚本。为了重新调用它，可以使用你的浏览器的重载功能。

sess_track.pl打开会话并在生成任何页面输出之前增加计数器。这样做是必须的，因为如果会话是新的那么客户端必须接收到一个包含会话名称和标识符的cookie。任何被发送的cookie都必须是响应报头的一部分，因此页面主体必须在报头被发送之后才能被生成出来。

脚本同样也生成部分在会话中使用的页面主体，但是却将其存储在一个变量中而非立即将其编写出来。原因是，当会话需要被终结时，脚本将cookie重置为告知浏览器丢弃它所拥有的会话。这必须在任何的报头或者页面内容被发送前终止。

会话到期

Apache::Session模块只需要per_session表中的id和a_session数据列，同时无需为暂停或者到期的会话做任何准备。另一方面，该模块并不限制你增加别的数据列，因此你可以在表中增加TIMESTAMP数据列来记录每个会话上次更新的时间。例如，你可以通过使用ALTER TABLE在perl_session表中增加TIMESTAMP数据列t：

```
ALTER TABLE perl_session ADD t TIMESTAMP NOT NULL, ADD INDEX (t);
```

然后，你可以通过周期地运行语句检查来清扫表并移除旧的记录来处理到期的会话。下面的语句每4个小时处理一次到期的会话：

```
DELETE FROM perl_session WHERE t < NOW() - INTERVAL 4 HOUR;
```

ALTER TABLE语句指明的t让DELETE操作更快。

20.2 在Ruby应用程序中使用基于MySQL的存储

Using MySQL-based Session in Ruby Applications

问题

你想要在Ruby脚本中使用会话存储。

解决方案

使用CGI::Session类接口。默认情况下，它使用临时文件来备份存储，不过你也可以配置它使用MySQL。

讨论

CGI::Session类管理会话存储。它通过cookies的方法来指明会话，这样可以显式地增加发送给客户端的响应。CGI::Session允许你提供一个存储管理类来代替使用临时文件的默认管理器。我们将使用mysql_session包，这是基于Ruby DBI接口同时使用MySQL来存储会话记录的。mysql-session是从Ruby应用程序文档中得来的。附录A中有获取并安装它的详细信息。

为了在脚本中使用mysql_session，你需要访问这些模块：

```
require "cgi"
require "cgi/session"
require "mysqlstore"
```

为了创建会话，首先需要创建一个CGI对象。然后调用`CGI::Session.new`，该方法需要几个参数：第一个是与脚本关联的CGI对象（该对象必须在你开启会话之前就存在）；其他参数提供关于会话本身的信息。下面的内容则与你所使用的存储管理器无关：

`session_key`

会话关键字是会话管理器用来作为发送给客户端的cookie的名称的数值。其默认的数值是`_session_key`，我们将使用`RUBYSESSID`。

`new_session`

当该参数为真时将强制产生一个新的会话，为假时则使用已经存在的会话且该会话被假设为已经在前一次请求中产生。当然如果不存在会话也可以创建一个新的会话或者使用一个当前存在的会话。为了使该操作能够正常工作，可以忽略`new_session`参数（本章的示例脚本就是这样做的）。

`database_manager`

为会话记录提供存储管理的类的名称。如果该参数被忽略，会话管理器将使用临时文件。

为了将`mysql-session`包作为存储管理器，`database_manager`参数则应该是`CGI::Session::MySQLStore`。这种情形下，`mysql-session`可以使`CGI::Session.new`方法中的其他几个参数正常工作。你可以传递参数以指示会话管理器确认它自己同MySQL的连接，或者你可以开启你自己的连接然后将数据库句柄传递给会话管理器。

下面的讨论将显示两种方法，但是对于任意一种方法，我们都需要存储会话记录的表。对于`mysql-session`，创建一个具有如下结构的名为`ruby_session`的表：

```
CREATE TABLE ruby_session
(
    session_id      VARCHAR(255) NOT NULL,
    session_value   MEDIUMBLOB NOT NULL,
    update_time     DATETIME NOT NULL,
    PRIMARY KEY (session_id)
);
```

现在我们回到会话创建。为了让会话管理器开启它自己与MySQL的连接，可以这样创建会话：

```
cgi = CGI.new("html4")
sess_id = cgi.cookies["RUBYSESSID"]
session = CGI::Session.new(
    cgi,
    "session_key" => "RUBYSESSID",
    "database_manager" => CGI::Session::MySQLStore,
    "db.host" => "127.0.0.1",
    "db.user" => "cbuser",
    "db.pass" => "cbpass",
    "db.name" => "cookbook",
```

```
    "db.table" => "ruby_session",
    "db.hold_conn" => 1
)
```

其中，代码里使用的db.xxx参数是告知mysql-session如何连接到服务器，如同为会话记录使用的数据库和表：

db.host

运行MySQL服务器的主机。

db.user, db.pass

要使用的MySQL的用户名和密码。

db.name, db.table

会话表的数据库和表的名称。

db.hold_conn

默认情况下，mysql-session在每次它需要发送语句给MySQL服务器时打开和关闭连接。如果你将db.hold_conn参数设为1，那么mysql-session将仅打开连接一次然后保持它开启直到会话结束。

另一个创建会话的方法是首先开启你自己的数据库连接然后将其句柄作为参数传递：

```
cgi = CGI.new("html4")
sess_id = cgi.cookies["RUBYSESSID"]
session = CGI::Session.new(
    cgi,
    "session_key" => "RUBYSESSID",
    "database_manager" => CGI::Session::MySQLStore,
    "db.dbh" => dbh,
    "db.name" => "cookbook",
    "db.table" => "ruby_session"
)
```

在这种情形下，db.host、db.user、db.pass及db.hold_conn这些参数都不被使用。另外，你需要在会话不再被需要时自行负责关闭连接。

不管你采用哪种方法创建会话，它的ID在其作为session.session_id属性的打开时间内都是可用的。

为关闭会话，调用会话对象的关闭方法。如果你想要销毁会话，那么调用它的删除方法。

会话管理器使用key/value对来存储数据，其中value的数值为字符串类型。实际上会话管理器并不清楚你所存储的数值的类型。我找到了如下的策略来处理关于类型转换的问题：

1. 在打开会话后，从会话中提取数值并且将其从“通用”的字符串转换为类型正确的数值。
2. 采用类型正确的数值工作直到会话需要关闭。

3. 将类型化的数值转换为字符串格式，然后将它们存储在会话中再关闭会话。

下面的脚本使用CGI::Session会话管理器来追踪一个会话中请求的数目以及每个请求的时间。在10个请求之后，脚本将删除当前会话以为下一个请求创建一个新的会话：

```
#!/usr/bin/ruby -w
# sess_track.rb - 会话请求计数/时间戳演示程序

require "Cookbook"
require "cgi"
require "cgi/session"
require "mysqlstore"

title = "Ruby Session Tracker";

dbh = Cookbook::connect

cgi = CGI.new("html4")
session = CGI::Session.new(
    cgi,
    "session_key" => "RUBYSESSID",
    "database_manager" => CGI::Session::MySQLStore,
    "db.dbh" => dbh,
    "db.name" => "cookbook",
    "db.table" => "ruby_session"
)

# 从会话中提取串值

count = session["count"]
timestamp = session["timestamp"]

# 将变量转换为正确类型

count = (count.nil? ? 0 : count.to_i)
timestamp = "" if timestamp.nil?
timestamp = timestamp.split(",")

# 增加计数器并将当前时间戳加入到时间戳列表

count = count + 1
timestamp << Time.now().strftime("%Y-%m-%d %H:%M:%S")

# 构建当前页面主体

page = ""

page << cgi.p {"This session has been active for #{count} requests."}
page << cgi.p {"The requests occurred at these times:"}
list = ""
timestamp.each do |t|
  list << cgi.li { t.to_s }
end
page << cgi.ul { list }
```

```
if count < 10
    # 将会话变量转换为串值并存储回会话同时关闭(并保存)会话
    session["count"] = count.to_s
    session["timestamp"] = timestamp.join(", ")
    session.close()
else
    # 10次调用后销毁会话
    session.delete()
end

dbh.disconnect

# 生成输出页面

cgi.out {
    cgi.html {
        cgi.head { cgi.title { title } } +
        cgi.body("bgcolor" => "white") { page }
    }
}
```

CGI::Session对于会话到期没有提供任何准备，但是你可以使用本章上一节讨论的技术来丢弃旧的会话。如果你这样做，你应该指明update_time数据列以使DELETE语句执行得更快：

```
ALTER TABLE ruby_session ADD INDEX (update_time);
```

20.3 在PHP会话管理器中使用基于MySQL的存储

Using MySQL-Based Storage with the PHP Session Manager

问题

你想要在PHP脚本中使用会话存储。

解决方案

PHP含有会话管理器。默认条件下，它使用临时文件来备份存储，但是你可以配置它使用MySQL。

讨论

本节展示了如何使用PHP原本的会话管理器以及如何通过实现在MySQL中保存会话数据的存储模块来扩展它。如果你的PHP配置中track_vars变量可用，那么会话变量可以作为\$HTTP_SESSION_VARS的全局数组或者\$_SESSION的更高层数组的元素。track_vars在PHP4.0.3中一直可用，所以我将假设你安装的PHP版本正确。即register_globals配置变

量可用，你的脚本中也存在有同样名称的全局会话变量。然而，这并不安全，所以该变量在这里假设为不可用。（19章第5节讨论了PHP的全局和更高层数组以及register_globals的蕴含安全。）

PHP会话管理接口

PHP的会话管理能力是基于一个较小的函数集的，它们的文档信息可以在PHP用户手册中查找。下面的列表描述了那些对日常会话编程最有用的功能：

`session_start`

打开一个会话并且提取之前存储在其中的变量，使它们在脚本的全局名字空间中可见。例如，一个名为x的会话变量作为`$_SESSION["x"]`或者`$HTTP_SESSION_VARS["x"]`使用。该函数必须在使用相关会话变量数组之前首先调用。

`session_register(var_name)`

通过在你的脚本中设置会话记录和变量之间的关联在会话中注册变量。例如，可以这样注册`$count`：

```
session_register ("count");
```

如果在会话保持开启期间你对变量作了任何修改，那么当会话被关闭时新的数值将被保存到会话中。

然而，我提及这个函数的唯一理由就是指出我们将不会使用它。`session_register()`仅仅在`register_globals`可用时才有效，这是有安全风险的。为了避免依靠`register_globals`，我们将从`$_SESSION`数组或者`$HTTP_SESSION_VARS`数组获取会话变量。

`session_unregister(var_name)`

注销一个会话变量可以让其不再被保存在会话记录中。与`session_register()`有所不同，这个函数并不依赖于`register_globals`的设置。

`session_write_close`

记录会话数据并关闭会话。PHP文档指明正常情况下你不需要调用该函数，因为PHP将在你的脚本运行结束时自动保存开启的会话。然而，在PHP5中，当你自行提供你的会话句柄时PHP可能并非一直自动为你保存会话。为了安全起见，最好调用该函数来保存你所做的改动。

`session_destroy`

移除会话及相关数据。

`session_name($name)`

PHP的会话管理器确定哪一个会话通过会话标识符的方法被使用。它通过检查下面的资源来查找标识符：名为`$PHPSESSID`的全局变量；cookie、获取或寄送名为`PHPSESSID`

的变量；或者是形为PHPSESSID=*value*的URL参数。（如果这些资源都没有被查找到，会话管理器将生成一个新的标识符并开启一个新的会话。）默认的标识符名称是PHPSESSID，不过你可以改变它。如果想要做一个全局（整个网站级的）修改，可以通过编辑php.ini中的session.name的配置变量来实现。为了针对单个的脚本作改动，可以在会话开始之前调用session_name(\$name)，其中\$name表示要使用的会话名称。为了确定当前的会话标识符名称，可以以无参数的形式调用session_name()函数。

指定用户定义的存储模块

前面描述的PHP会话管理接口没有提及任何类型的备份存储。也就是说，该描述并没有指明任何关于会话信息是如何实际保存的。默认情况下，PHP使用临时文件来保存会话数据，但是会话接口是可扩展的，因此可以定义其他存储模块。为了重载默认的存储方法并且将会话数据存储到MySQL中，你必须做如下几件事情：

1. 创建一个会话来维持会话记录同时编写一个实现存储模块的过程。这个工作必须在任何一个使用该模块的脚本之前完成。
2. 告知PHP你会提供一个用户定义的存储管理器。你可以在全局情况下的php.ini中完成该工作（在php.ini中你可以对它修改一次），或者在单个的脚本中完成该修改（这种情况下必须声明每个脚本的内容）。
3. 注册每一个需要使用该存储模块的脚本。

创建会话表。任何一个基于MySQL的存储模块需要存储会话信息的数据库表。创建一个包含如下数据列的名为php_session的表：

```
CREATE TABLE php_session
(
    id      CHAR(32) NOT NULL,
    data    MEDIUMBLOB,
    t       TIMESTAMP NOT NULL,
    PRIMARY KEY (id),
    INDEX (t)
);
```

你会发现该表的结构与本章第一节为Apache::Session Perl模块使用的perl_session表相当类似。*id*数据列记录会话标识符，其格式为unique的32字符的字符串（它们看起来疑似Apache::Session标识符，这并不令人惊讶，因为当PHP使用MD5 数值时，看来就与Perl模块很像）。*data*列保持会话信息。PHP在存储会话数据之前先将其串行化为字符串，因此php_session仅仅需要一个大的通用字符串数据列来保存串行化后的数值。*t*数据列是

MySQL在会话记录更新时自动更新的时间戳信息。该数据列不是被强制要求的，但是对于实现基于每个会话的最后更新时间的“垃圾”回收策略非常有效。

一个小的语句集合足够按照我们定义的方式管理php_session的内容：

- 为了获取会话数据，实现一个基于会话标识符的简单SELECT：

```
SELECT data FROM php_session WHERE id = 'sess_id';
```

- 为了记录会话数据，采用REPLACE来更新已存在的数据行（如果没有这样的数据行存在那么就创建一个新的）：

```
SELECT data FROM php_session WHERE id = 'sess_id';
```

当创建或者更新数据行时REPLACE同样也更新数据行的时间戳，这对于“垃圾”回收非常重要。一些存储管理实现了当由于具有给定会话ID的数据行已经存在的情况下进行INSERT操作失败时（或者对于具有回滚的UPDATE进行INSERT时，如果UPDATE失败那么给定ID的数据行并不存在），使用INSERT和回滚的组合来UPDATE。在MySQL中，双重语句的方法不是必须的；REPLACE可以使用一条语句实现必要的任务

- 为了销毁会话，删除对应的数据行：

```
DELETE FROM php_session WHERE id = 'sess_id';
```

- “垃圾”回收通过移除旧的数据行实现。下面的语句删除时间戳数值大于*sess_life*的数据行：

```
DELETE FROM php_session WHERE t < NOW() - INTERVAL sess_life SECOND;
```

PHP会话管理器在调用“垃圾”回收过程时支持*sess_life*数值。（*php_session*表的定义指明的*t*可以让DELETE语句执行更快。）

这些语句构成了我们的基于MySQL存储模块的基础。模块的基本功能是打开和关闭MySQL连接以及在正确的时间提供正确的语句。

编写存储管理过程。用户定义的会话存储模块具有指定的接口，通过在你注册PHP会话管理器时调用session_set_save_handler()的句柄集合实现。函数的形式如下所示，其中每个参数都是一个由指定字符串命名的过程：

```
session_set_save_handler (
    "mysql_sess_open",      # 打开会话的函数
    "mysql_sess_close",     # 关闭会话的函数
    "mysql_sess_read",      # 读取会话数据的函数
    "mysql_sess_write",     # 写会话数据的函数
    "mysql_sess_destroy",   # 销毁会话的函数
    "mysql_sess_gc"         # 过去会话的垃圾回收函数
);
```

句柄过程的顺序必须如上所示，但是你可以按照你的想法对他们命名。他们并非必须被命名为mysql_sess_open()、mysql_sess_close()，等等。这些处理过程应该根据下面的要求编写：

```
mysql_sess_open ($save_path, $sess_name)
```

在执行任何操作之前都必须先开始一个会话。`$save_path`是存储会话的位置的名称；这只对文件存储有效。`$sess_name`指明了会话标识符的名称（例如，`PHPSESSID`）。对于基于MySQL存储管理器，两个参数都可以忽略。该函数返回TRUE或FALSE指明会话是否成功打开。

```
mysql_sess_close()
```

关闭会话，返回TRUE或FALSE以指明操作成功或者失败。

```
mysql_sess_read ($sess_id)
```

获取与会话标识符相关的数据并将其作为字符串返回。如果没有会话，该函数返回一个空的字符串。如果有错误发生，它返回FALSE。

```
mysql_sess_write ($sess_id, $sess_data)
```

保存与会话标识符相关的数据，如果成功返回TRUE，否则返回FALSE。PHP本身负责处理串行化和解串行化会话内容，因此阅读和编写函数仅仅需要处理串行化的字符串。

```
mysql_sess_destroy ($sess_id)
```

销毁会话和所有相关数据，成功返回TRUE，否则返回FALSE。对于基于MySQL的存储，通过删除`php_session`表中与会话ID相关的数据行来实现销毁会话。

```
mysql_sess_gc ($gc_maxlife)
```

执行“垃圾”回收以移除旧的会话。该函数是随机调用的。当PHP收到一个使用会话的页面请求时，它将告知“垃圾”回收器会话预先定义的概率，如果概率为1（表示1%），PHP大概每100次请求调用该函数1次。如果为100，那么每次请求都将调用该函数——这很可能导致超过你预期的处理时间。

`gc()`的参数是会话的最大生存时间，单位为秒。存在时间长于该时间的会话将被移除。该函数成功返回TRUE，失败返回FALSE。

为了注册句柄过程，可以调用`session_set_save_handler()`函数，然后通知将使用用户定义存储模块的PHP协助完成注册过程。默认的存储管理方法通过`session.save_handler`配置变量定义。你可以通过修改该`php.ini`初始化文件来全局地修改该方法，或者在每个脚本文件中修改：

- 为了在全局情况下修改该存储方法，可以编辑php.ini。默认的配置指明使用基于文件的会话存储管理：

```
session.save_handler = files;
```

修改这一点指明会话将通过用户层的机制来处理：

```
session.save_handler = user;
```

如果你将PHP作为一个Apache模块使用，你需要在修改php.ini之后重新启动Apache以让PHP注意到该变化。做一个全局变化的问题使每一个使用会话的PHP脚本都将被要求提供它自己的存储管理过程。如果其他的脚本编写人员没有注意到这个变化，这将给他们带来意外的麻烦。例如，其他使用网页服务器的开发人员想继续使用基于文件的会话。

- 采用全局变化的方法需要在每一个脚本的基础上通过调用ini_set()来指明不同的存储模块：

```
ini_set ("session.save_handler", "user");
```

ini_set()函数比全局配置改变带来的影响更小。我们这里将对开发的存储管理器使用ini_set()，因此数据库备份的存储将仅仅能被那些请求使用它的脚本所激活。

为了容易地访问会话存储模块，创建库文件Cookbook_Session.php是非常有效的方法。脚本文件使用库文件唯一需要的就是在开始会话之前引用它。文件的要点如下所示：

```
<?php  
# Cookbook_Session.php - 基于MySQL的会话管理模块  
  
require_once "Cookbook.php";  
  
# 定义处理程序  
  
function mysql_sess_open ($save_path, $sess_name) ...  
function mysql_sess_close () ...  
function mysql_sess_read ($sess_id) ...  
function mysql_sess_write ($sess_id, $sess_data) ...  
function mysql_sess_destroy ($sess_id) ...  
function mysql_sess_gc ($gc_maxlife) ...  
  
# 初始化连接标识符，选择用户自定义的会话处理并该处理程序  
  
$mysql_sess_conn = FALSE;  
ini_set ("session.save_handler", "user");  
session_set_save_handler (  
    "mysql_sess_open",  
    "mysql_sess_close",  
    "mysql_sess_read",  
    "mysql_sess_write",
```

```
"mysql_sess_destroy",
"mysql_sess_gc"
);
?>
```

库文件包括Cookbook.php，因此它可以访问连接过程以开启到cookbook数据库的连接。然后它定义了句柄过程（我们很快将得到这些函数的细节）。最后，它初始化了连接标识符，告知PHP准备使用用户定义的会话存储管理器，同时注册句柄函数。因此，一个在MySQL中想存储会话的PHP脚本将通过引用Cookbook_Session.php文件来简单执行所有必需的设置：

```
require_once "Cookbook_Session.php";
```



提示：由Cookbook_Session.php库文件提供的接口给出了一个全局的数据库连接标识符变量（\$mysql_sess_conn），这与名为mysql_sess_open()、mysql_sess_close()等句柄过程集相同。使用库文件的脚本应该避免因为其他目的使用这些全局名称。

现在让我看看如何实现每一个句柄过程：

打开会话

PHP将两个参数传递给该函数：保存的路径和会话的名称。保存路径用于基于文件的存储，同时我们不需要知道会话名称，因此对于我们的目的而言两个参数都是无关的，可以被忽略。因此该函数除了开启一个MySQL的连接之外无需做别的任何事情：

```
function mysql_sess_open ($save_path, $sess_name)
{
    global $mysql_sess_conn;

    # 如果没有开启与MySQL的链接就打开它
    if (! $mysql_sess_conn)
    {
        # 此处不能使用=&操作符
        $mysql_sess_conn = Cookbook::connect ();
        if (PEAR::isError ($mysql_sess_conn))
        {
            $mysql_sess_conn = FALSE;
            return (FALSE);
        }
    }
    return (TRUE);
}
```

相对于=&操作符mysql_session_open()通常使用=&操作符来指派连接结果的调用，以保证当函数返回时连接处理程序的值不会消失。

关闭会话

关闭处理程序检查是否有MySQL的连接处于打开状态，如果是那么关闭它：

```
function mysql_sess_close ()
{
    global $mysql_sess_conn;

    if ($mysql_sess_conn)      # 如果连接开启那么关闭它
    {
        $mysql_sess_conn->disconnect ();
        $mysql_sess_conn = FALSE;
    }
    return (TRUE);
}
```

读取会话数据

mysql_sess_read() 函数使用会话ID为相应的会话记录查找并返回数据。如果没有相应的记录存在，那么它将返回空串。如果有错误发生，那么它返回FALSE：

```
function mysql_sess_read ($sess_id)
{
    global $mysql_sess_conn;

    $stmt = "SELECT data FROM php_session WHERE id = ?";
    $result = &$mysql_sess_conn->query ($stmt, array ($sess_id));
    if (!PEAR::isError ($result))
    {
        list ($data) = $result->fetchRow ();
        $result->free ();
        if (isset ($data))
            return ($data);
        return ("");
    }
    return (FALSE);
}
```

写入会话数据

如果目前没有会话那么mysql_sess_write() 创建新的纪录，如果有一个会话纪录那么就替换已有的纪录：

```
function mysql_sess_write ($sess_id, $sess_data)
{
    global $mysql_sess_conn;

    $stmt = "REPLACE php_session (id, data) VALUES (?,?)";
    $result = &$mysql_sess_conn->query ($stmt, array ($sess_id, $sess_data));
    return (!PEAR::isError ($result));
}
```

销毁会话

当一个会话不再需要时，mysql_sess_destroy() 会移除相应记录：

```

function mysql_sess_destroy ($sess_id)
{
global $mysql_sess_conn;

$stmt = "DELETE FROM php_session WHERE id = ?";
$result =& $mysql_sess_conn->query ($stmt, array ($sess_id));
return (!PEAR::isError ($result));
}

```

执行“垃圾”回收

每个会话纪录中的TIMESTAMP数据列t指明了会话最后一次更新的时间。mysql_sess_gc()使用该数值来实现“垃圾”回收。参数\$sess_maxlife指明了会话最长能存在的时间（单位为秒）。生存时间长于该数值的会话被认为是过期的且等待被移除，这可以通过删除从当前的时间以及记录时间戳计算得到的存在时间长于允许时间的会话来实现：

```

function mysql_sess_gc ($sess_maxlife)
{
global $mysql_sess_conn;

$stmt = "DELETE FROM php_session WHERE t < NOW() - INTERVAL ? SECOND";
$result =& $mysql_sess_conn->query ($stmt, array ($sess_maxlife));
return (TRUE); # 忽略错误
}

```

使用存储模块。在你的脚本可访问的公用库文件目录中安装Cookbook_Session.php文件。（在我的系统中，我将PHP库文件放置在/usr/local/lib/mcb目录下并且修改了php.ini文件因此可以使用include_path变量对该目录命名。详见第2.3节。）为了使用存储模块，需要将如下的示例脚本，sess_track.php，安装到你的网页树目录下并且调用它几次以查看显示信息的变化：

```

<?php
# sess_track.php -会话请求计数/计时演示程序

require_once "Cookbook_Session.php";
# 使用make_unordered_list(),get_session_val(),
# set_session_val()所需要的引用
require_once "Cookbook_Webutils.php";

$title = "PHP Session Tracker";

# 打开会话并提取会话值

session_start ();
$count = get_session_val ("count");
$timestamp = get_session_val ("timestamp");

# 如果会话是新的，初始化变量

if (!isset ($count))
$count = 0;
if (!isset ($timestamp))

```

```

$timestamp = array ();

# 增加计数器数值，将当前时间戳加入到时间戳列表

++$count;
$timestamp[] = date ("Y-m-d H:i:s T");

if ($count < 10) # 将修改过的数值保存到会话变量列表
{
    set_session_val ("count", $count);
    set_session_val ("timestamp", $timestamp);
}
else           # 10次调用后销毁会话变量
{
    session_unregister ("count");
    session_unregister ("timestamp");
}
session_write_close (); # 保存会话的改动部分

# 生成输出页面

?>
<html>
<head>
<title><?php print ($title); ?></title>
</head>
<body bgcolor="white">

<?php

print ("<p>This session has been active for $count requests.</p>\n");
print ("<p>The requests occurred at these times:</p>\n");
print make_unordered_list ($timestamp);

?>

</body>
</html>

```

引用Cookbook_Session.php库文件的脚本使基于MySQL的存储模块能够正常工作，同时以典型的方式使用PHP会话管理器接口。首先，它打开对话同时尝试提取会话变量。对第一个请求，会话变量将不会被设置而是被初始化。这将通过`isset()`测试确认。标量类型变量`$count`的初始数值为0，同时非标量变量`$timestamp`以一个空的数值作为开始。对于后续请求，会话变量将之前的请求为其分配的数值。

然后，脚本程序将增加计数器，将当前的时间戳加入到时间戳数组的末尾，同时产生显示计数和访问时间的输出页面。如果到达会话流程限制的上限10，那么脚本将注销该会话变量，这样`$count`和`$timestamp`将不会被保存到会话纪录中。由此产生的效果即是在下一次请求到来时重新启动会话。

最后，sess_track.php调用session_write_close()将变化写出到会话数据中。

输出页面仅仅在更新会话纪录之后才生成，因为PHP可能需要确定一个包含会话ID的cookie并将其发送到客户端。这个决定需要在生成页面主体之前作出，因为cookie是在信息头中被发送的。

如同早些时候提及的，我们假设register_globals不能被使用。因此，我们不能使用PHP session_register()函数来注册会话变量，同时必须通过其他方法来访问会话变量。有两个可能的方法：使用\$_HTTP_SESSION_VARS全局数组或使用\$_SESSION超全局数组。例如，在调用session_start()之后，名为count的会话变量将作为\$_HTTP_SESSION_VARS["count"]或者\$_SESSION["count"]使用。

也可能使用PHP会话变量数组的方法，但是仍然让你使用简单的变量名称来操作会话变量：

1. 不要使用session_register()。取而代之，可直接从全局会话变量数组复制会话变量到\$count和\$timestamp变量。
2. 在你使用完你的会话变量之后，在记录该会话之前将所有会话变量复制回会话变量数组中。

然而，确定用哪一个全局数组来作为会话变量存储是很麻烦的，这取决于你的PHP的版本。如果你不想在每次访问会话变量时都面临作一次决定，那么编写一套工具函数来完成该工作会容易得多。那也是脚本中使用的如下两个函数的目的：get_session_val()，set_session_value()。它们访问会话变量中的计数器和时间戳数组同时将修改过的数值保存回会话中：

```
function get_session_val ($name)
{
    global $HTTP_SESSION_VARS;

    $val = NULL;
    if (isset ($_SESSION[$name]))
        $val = $_SESSION[$name];
    else if (isset ($HTTP_SESSION_VARS[$name]))
        $val = $HTTP_SESSION_VARS[$name];
    return ($val);
}

function set_session_val ($name, $val)
{
    global $HTTP_SESSION_VARS;

    if (isset ($_SESSION))
        $_SESSION[$name] = $val;
    $HTTP_SESSION_VARS[$name] = $val;
}
```

这些程序可以在Cookbook_Webutils.php库文件中找到，与这些程序在一起的还有其他种类的网页脚本的参数数值（参看第19.5节）。关于它们，Cookbook_Webutils.php比Cookbook_Session.php实现得更好，因此你也可以在不选择使用Cookbook_Session.php的基于MySQL的会话存储时也调用它们。

20.4 在Tomcat中为会话支持存储使用MySQL

Using MySQL for Session-Backing Store with Tomcat

问题

你想要在基于Java的脚本中使用会话存储。

解决方案

Tomcat替你处理会话管理。默认情况下，它使用临时文件来进行备份存储，但是你可以通过在Tomcat的server.xml配置文件中提供正确的JDBC参数来配置它使用MySQL。

讨论

在本章早些时候描述的Perl、Ruby以及PHP会话机制要求应用程序显式指明它们需要使用基于MySQL的会话存储。对于Perl和Ruby，任何一个脚本都必须指明它希望使用的正确的会话模块。对于PHP，会话管理器是内建在语言中的，但是每一个希望使用MySQL存储模块的应用程序必须注册它。

运行在Tomcat中的JAVA程序具有不用的框架。Tomcat本身管理会话，因此如果你希望在MySQL中存储会话信息，需要通过配置Tomcat而非你的应用程序来实现。换句话说，基于Web的Java程序比其他必需在应用层由别的语言处理的一些会话信息要宽松很多。例如，由Tomcat服务器处理会话IDs比在应用层处理好。如果允许使用cookie，Tomcat将使用它。否则，Tomcat使用URL回写在URL中编码会话ID。应用程序开发人员并不关心哪种方法将被使用，因为不管ID如何被传送对他们而言都是可用的。

为了举例说明应用程序与Tomcat使用的会话管理方法的无关性，本节展示了一个简单的使用一个会话的JSP应用程序。它展示了如何配置Tomcat来使用MySQL而非默认的会话存储来存储会话信息——无需求应用程序做任何改动。当然，首先必须描述会话接口。

Servlet和JSP会话接口

Tomcat使用Java Servlet规范中描述的标准会话接口。该接口可以同时被Servlet和JSP页面使

用。在Servlet中，你可以通过导入javax.servlet.http.HttpSession类并调用你的HttpRequest对象的getSession()来获取访问会话的能力：

```
import javax.servlet.http.*;
HttpSession session = request.getSession();
```

在JSP页面中，会话支持是默认可用的，因此那些语句在页面开始执行时就已经被运行。也就是说，通过已经为你设置好的会话变量，会话是隐式可用的。

完整的会话接口在Java Servlet (附录D) 规范中的 HttpSession 中定义。一些有代表性的会话对象方法如下所示：

```
is New()
```

返回true或false来指明会话是否是以当前请求开始的。

```
getAttribute(String attrName)
```

会话内容包括的属性，即由名称确定的对象。为了访问会话的属性，需要指明其名称。getAttribute()方法返回由给定名称限定的对象，如果没有具有该名称的对象则返回空。

```
setAttribute(String attrName, Object obj)
```

将对象加入到会话中并将其绑定到给定的名称。

```
removeAttribute(String attrName)
```

将具有给定名称的对象从会话中移除。

```
Invalidate()
```

会话及其相关的数据被设为无效。客户端提交的下一个请求将打开一个新的会话。

一个JSP会话应用程序

接下来的例子展示了JSP页面，sess_track.jsp，包括一个会话请求计数器和请求时间的日志。为了更加显式地说明会话相关操作，该页面基本由直接使用 HttpSession 会话接口的嵌入式Java代码组成：

```
<%--
 sess_track.jsp - session request counting/timestamping demonstration
--%>

<%@ page import="java.util.*" %>
<%
    // 获取会话变量，如果没设置则初始化它们

    int count;
    Object obj = session.getAttribute ("count");
    if (obj == null)
        count = 0;
    else
```

```

count = Integer.parseInt (obj.toString ());

ArrayList timestamp = (ArrayList) session.getAttribute ("timestamp");
if (timestamp == null)
    timestamp = new ArrayList ();

// 增加计数器，将当前时间戳加入到时间戳数组中

count = count + 1;
timestamp.add (new Date ());

if (count < 10) // 将更新的数值保存到会话对象中
{
    session.setAttribute ("count", String.valueOf (count));
    session.setAttribute ("timestamp", timestamp);
}
else // 10个请求之后重启会话
{
    session.removeAttribute ("count");
    session.removeAttribute ("timestamp");
}
%>

<html>
<head>
<title>JSP Session Tracker</title>
</head>
<body bgcolor="white">

<p>This session has been active for <%= count %> requests.</p>

<p>The requests occurred at these times:</p>
<ul>
<%
    for (int i = 0; i < timestamp.size (); i++)
        out.println ("<li>" + timestamp.get (i) + "</li>");
%>
</ul>

</body>
</html>

```

调用sess_track.jsp几次，通过你的浏览器来查看显示是如何变化的。

在sess_track.jsp中使用的`sess.setAttribute()`方法将信息放置到会话中，因此该信息可以在后面的脚本调用中使用。如果想查看这一特点，备份sess_track.jsp，然后从你的浏览器调用该备份。你将看到它与sess_track.jsp一样访问同样的会话信息（同一个应用程序的脚本访问相同的信息）。

在sess_track.jsp中展示的一些会话相关操作能够在JSTL中使用标签来完成，JSTL提供一个`sessionScope`变量来获得潜在的JSP会话对象：

```

<%--
  sess_track.jsp - 会话请求计数/计时演示程序
--%>

<%@ page import="java.util.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:if test="${empty sessionScope.count}">
  <c:set var="count" scope="session" value="0"/>
</c:if>
<c:set var="count" scope="session" value="${sessionScope.count+1}" />

<%
  ArrayList timestamp = (ArrayList) session.getAttribute ("timestamp");
  if (timestamp == null)
    timestamp = new ArrayList ();
  // 将当前时间戳加入到时间戳列表中，同时将结果保存到会话
  timestamp.add (new Date ());
  session.setAttribute ("timestamp", timestamp);
%>

<html>
<head>
<title>JSP Session Tracker</title>
</head>
<body bgcolor="white">

<p>This session has been active for
<c:out value="${sessionScope.count}" />
requests.</p>

<p>The requests occurred at these times:</p>
<ul>
<c:forEach items="${sessionScope.timestamp}" var="t">
  <li><c:out value="${t}" /></li>
</c:forEach>
</ul>

<%-- has session limit of 10 requests been reached? --%>

<c:if test="${sessionScope.count ge 10}">
  <c:remove var="count" scope="session"/>
  <c:remove var="timestamp" scope="session"/>
</c:if>

</body>
</html>

```

告知Tomcat将会话记录保存到MySQL中

关于会话管理的Tomcat文档可以在下面的地址中找到：

<http://tomcat.apache.org/tomcat-5.0-doc/config/manager.html>

Tomcat具有它自己默认的会话存储机制（临时文件）。为了覆盖该默认机制以通过JDBC在MySQL中保存会话，可以使用下面的流程：

1. 创建一张表来保存会话记录。
2. 确认Tomcat能够访问正确的JDBC驱动。
3. 修改Tomcat的server.xml配置文件以指明对相关的应用程序的会话管理器的持久使用。

上面的步骤都没有包括如何修改示例会话应用程序，该示例程序反映了Tomcat是如何在应用程序层上实现会话支持的。

创建Tomcat会话表。Tomcat将一个类型的信息存储到会话表中：

- 会话ID。默认情况下，ID是32字符的MD5编码数值。
- 应用程序名称。
- 会话数据。是串行化字符串。
- 会话是否可用，用一个字节表示。
- 允许的最长不活动时间，以秒为单位的32位整型数据。
- 最后一次访问时间，是一个64位的整数。

这些规范满足下面的表，你应该在继续下面的步骤之前创建该表：

```
CREATE TABLE tomcat_session
(
    id          VARCHAR(32) NOT NULL,
    app         VARCHAR(255),
    data        MEDIUMBLOB,
    valid_session CHAR(1) NOT NULL,
    max_inactive INT NOT NULL,
    last_access  BIGINT NOT NULL,
    PRIMARY KEY (id),
    INDEX (app)
);
```

将JDBC驱动放置到Tomcat能够查找到的位置。因为Tomcat本身管理会话，它必须能够访问用来将会话存储到数据库中的JDBC驱动。它通常将驱动存储到Tomcat目录的库目录下，这样所有的应用程序都可以访问到它们。但是对于一个同样能够被Tomcat访问的驱动，他应该被放置到common/lib目录中。（因此，如果你在库目录中安装了MySQL Connector/J驱动，就应将其移动到common/lib。）在重新启动之后，Tomcat将能够使用它。关于在Tomcat中使用MySQL Connector/J的更多信息，请查看第17.3节。

修改Tomcat配置文件。为了告知Tomcat使用tomcat_session表，必须修改Tomcat的conf目录中的server.xml文件。完成这个任务，需要将<Manager>元素放置到基于MySQL会话存储的每个应用程序的<Context>元素的主体中。（如果某一个应用程序没有该元素，则创建一个。）对于mcb应用程序，可以这样创建<Context>元素：

```
<Context path="/mcb" docBase="mcb" debug="0" reloadable="true">
    <Manager>
```

```
className="org.apache.catalina.session.PersistentManager"
debug="0"
saveOnRestart="true"
minIdleSwap="900"
maxIdleSwap="1200"
maxIdleBackup="600">
<Store
    className="org.apache.catalina.session.JDBCStore"
    driverName="com.mysql.jdbc.Driver"
    connectionURL=
        "jdbc:mysql://localhost/cookbook?user=cbuser&password=cbpass"
    sessionTable="tomcat_session"
    sessionIdCol="id"
    sessionAppCol="app"
    sessionDataCol="data"
    sessionValidCol="valid_session"
    sessionMaxInactiveCol="max_inactive"
    sessionLastAccessedCol="last_access"
/>
</Manager>
</Context>
```

<Manager>元素的属性指明了通常会话相关的选项。在<Manager>元素的主体中，<Store>元素属性提供与JDBC驱动相关的特性。下面的讨论焦点在于示例程序中展示的特征，但是还有些别的特征你可以使用。查阅Tomcat会话管理文档来获取更多的信息。

示例中的<Manager>属性具有如下含义：

className

指明实现相关会话存储的Java类。它必须是org.apache.catalina.session.PersistentManger。

debug

指明日志文件的细节级别。0表示不输出调试信息；更高的数字表示产生更多的输出。

saveOnRestart

让应用程序会话在服务器重启之后能够继续存在。如果你希望Tomcat在其关闭之时能够保存当前会话（同时当其启动后能够重新载入），其值应该为真。

maxIdleBackup

指明不活动的会话在多少秒之后就不再被保存到MySQL中。数值-1表示“从不(never)”，

minIdleSwap

指明会话在准备被交换出服务器（保存到MySQL并移出服务器内存）之前能够有多少秒空闲时间。-1表示“never”。

maxIdleSwap

指明会话在准备被交换出服务器之前能够有多少秒空闲时间。-1表示“从不”。如果该特性为可用，那么该数值必须比minIdleSwap和maxIdleBackup大。

在<Manager>元素的主体中，<Store>元素指明如何连接数据库服务器，其中的数据库和表用来存储会话记录，同时表中的数据列名称如下：

className

实现org.apache.catalina.Store的类的名称。对于基于JDBC存储管理器，属性的数值应该是org.apache.catalina.session.JDBCStore。

driverName

JDBC驱动的类名称。对于MySQL Connector/J驱动，该特征的数值应该是com.mysql.jdbc.Driver。

connectionURL

指明如何连接到数据库服务器。下面的URL连接到本地主机的MySQL服务器，分别使用数据库、用户名以及cookbook的密码、cbuser以及cbpass：

jdbc:mysql://localhost/cookbook?user=cbuser&password=cbpass

然而，server.xml调用入口是使用XML编写的，因此分隔用户和密码连接参数的&符号必须被写为&，如下所示：

jdbc:mysql://localhost/cookbook?user=cbuser&password=cbpass

当Tomcat读取server.xml文件，XML解析器将&转换回&，然后传递给JDBC驱动。

sessionTable

存储会话记录的表名称。对于我们的示例，这是早些时候描述的tomcat_session。(包含该表的数据库在连接URL数值中给定。)

示例程序中保留的<Store>属性指明会话表中的数据列名称。这些属性是sessionIdCol、sessionAppCol、sessionDataCol、sessionValidCol、sessionMaxInactiveCol以及sessionLastAccessedCol，这些都明显对应tomcat_session表中包含的数据列。

在你修改server.xml文件之后，重新启动Tomcat。然后调用sess_track.jsp或sess_track2.jsp脚本若干次来初始化会话。每次都必须按照之前你重新配置Tomcat的方法。在不活动时间等于<Manager>元素maxIdleBackup属性数值之后，你应该能够看见会话记录出现在tomcat_session表中。如果查看MySQL查询日志，你应该也能看见当你关闭Tomcat时会话被保存到MySQL中。

改变server.xml是一个全局改动，与改动PHP的php.ini文件session.save_handler有些类似。然而，与PHP不同的是，与改动全局初始化文件会影响到同一主机上的其他开发者以致他们不得不改动他们基于会话的脚本，修改Tomcat的配置文件以使用基于JDBC来进行会话管理的备份存储对servlets和JSP页面完全不可见。因此，你能够做这些改动而无须担心使用同一个Tomcat服务器的其他开发者会指责你的行为对他们造成影响。

Tomcat中的会话过期

会话默认存在60分钟。为了提供显式的会话管理器期限，增加maxInactiveInterval到正确的服务器conf/server.xml文件的<Manager>元素中。为了给指明的特定应用程序提供期限，在应用程序的WEB-INF/web.xml文件中增加<session-config>元素。例如，为了让变量的数值能够使用30分钟，可以这样指明：

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

如果修改了server.xml或web.xml，需要重新启动Tomcat。

在Tomcat中追踪会话

尽管你的JSP页面不需要做任何事情以设置会话或者使用JDBC来进行会话存储，它们可能需要采取一小步来确认会话能够正确地从一个请求移动到另一个请求。如果你生成包含链接到参与同一个会话的其他页面的页面，这是必须的。

Tomcat自动生成会话标识符同时追踪使用cookie的会话，当然该会话要能够从包含会话ID的客户端获取到cookie。如果客户端cookie不可用，Tomcat通过重写包括会话ID的URL来追踪会话。你不需要确定Tomcat使用哪种方法，但是你应该小心确认会话ID的正确传递方法以保证它通过URL重写被正确传递。这意味着如果你创建包括链接到会话一部分的其他页面的页面，你不应该像下面那样列出页面的路径：

```
To go to the next page,
<a href="nextpage.jsp">click here</a>.
```

该链接不包括会话ID。如果Tomcat使用URL重写来追踪会话，当用户选择链接时你将失去ID。取而代之，将链接传递给encodeURL()来让Tomcat能够将会话ID加入到必需的URL中：

```
To go to the next page,
<a href="<% response.encodeURL ("nextpage.jsp") %>">click here</a>.
```

如果Tomcat采用cookie追踪会话，encodeURL()返回未改动的URL。然而，如果Tomcat通过

URL重写的方法来追踪会话，`encodeURL()`将会话ID自动加入到页面路径，因此看起来如下所示：

```
mypage.jsp;jsessionid=xxxxxxxxxxxxxxxxxx
```

你应该使用`encodeURL()`生成URL，如同任意将用户带到当前会话的页面标签的链接。如果出于某些原因，那些标签调用在会话指定的基础上生成图片的脚本，包括`<a>`、`<form>`以及`<frame>`标签，甚至可能包括``标签。

当为基于会话的应用程序编写URL时，最好就是为过程事件开发使用`encodeURL()`的习惯。甚至如果你想每一个使用应用程序的人都使用cookie，这种假设在某一天可能被证明是错误的。

`Java.net.URLEncoder.encode()`方法与`encodeURL()`具有类似的名称，但是它是不同的。它将特殊符号转换为型如`‰xx`的字符来保证它们安全地在URL中使用。

计算机精品学习资料大放送

[软考官方指定教材及同步辅导书下载](#) | [软考历年真题解析与答案](#)

[软考视频](#) | [考试机构](#) | [考试时间安排](#)

[Java 一览无余](#): [Java 视频教程](#) | [Java SE](#) | [Java EE](#)

[.Net 技术精品资料下载汇总](#): [ASP.NET 篇](#)

[.Net 技术精品资料下载汇总](#): [C#语言篇](#)

[.Net 技术精品资料下载汇总](#): [VB.NET 篇](#)

[撼世出击](#): [C/C++编程语言学习资料尽收眼底](#) [电子书+视频教程](#)

[Visual C++\(VC/MFC\)学习电子书及开发工具下载](#)

[Perl/CGI 脚本语言编程学习资源下载地址大全](#)

[Python 语言编程学习资料\(电子书+视频教程\)下载汇总](#)

[最新最全 Ruby、Ruby on Rails 精品电子书等学习资料下载](#)

[数据库管理系统\(DBMS\)精品学习资源汇总](#): [MySQL 篇](#) | [SQL Server 篇](#) | [Oracle 篇](#)

[平面设计优秀资源学习下载](#) | [Flash 优秀资源学习下载](#) | [3D 动画优秀资源学习下载](#)

[最强 HTML/xHTML、CSS 精品学习资料下载汇总](#)

[最新 JavaScript、Ajax 典藏级学习资料下载分类汇总](#)

[网络最强 PHP 开发工具+电子书+视频教程等资料下载汇总](#)

[UML 学习电子资下载汇总](#) [软件设计与开发人员必备](#)

[经典 LinuxCBT 视频教程系列](#) [Linux 快速学习视频教程一帖通](#)

[天罗地网](#): [精品 Linux 学习资料大收集\(电子书+视频教程\)](#) [Linux 参考资源大系](#)

[Linux 系统管理员必备参考资料下载汇总](#)

[Linux shell、内核及系统编程精品资料下载汇总](#)

[UNIX 操作系统精品学习资料<电子书+视频>分类总汇](#)

[FreeBSD/OpenBSD/NetBSD 精品学习资源索引](#) [含书籍+视频](#)

[Solaris/OpenSolaris 电子书、视频等精华资料下载索引](#)

>> [更多精品资料请访问大家论坛计算机区...](#)

获取MySQL软件

Obtaining MySQL Software

本书中介绍的表格定义和程序都是可在线获取的，这样就避免了你手工录入。当然，要运行这些示例，你需要访问MySQL，以及那些你将使用的MySQL独有的编程语言接口。本附录介绍你需要的软件，以及你怎么获得这些软件。

获取示例代码和数据

Obtaining Sample Source Code and Data

本书中的示例，都基于 `recipes` 和 `mcb-kjv` 两个光盘中的代码和数据，这两个光盘的内容在 MySQL Cookbook 伙伴网站（看首页）上可以找到。请访问以下网址：

<http://www.kitebird.com/mysql-cookbook/>

其中，`recipes` 光盘包含了大部分的示例。它是一个压缩的 tar 文件 (`recipes.tar.gz`) 或者一个 ZIP 文件 (`recipes.zip`)。这两个文件在解压缩后，都会生成一个名为 `recipes` 的目录。

`recipes` 光盘中包含了本书中的程序，但在很多情况下，也会包含在其他语言中执行。例如，一个使用 Python 的脚本在 `recipes` 中可能会是 Perl、Ruby、PHP 或 Java。如果你希望把本书中的某个示例程序用其他语言实现的话，这将节省你很多时间。

Kitebird 网站提供了 `mcb-kjv` 光盘，该光盘包含了 King James 版本的圣经的文本内容。这些文本经过格式化处理，非常适合导入 MySQL 中。这些内容在第5章中使用过，作为示范 FULLTEXT 搜索的一个比较合理的大文本；后来在本书的其他地方也使用到。本光盘与 `recipes` 是分开发布的，这主要是由于它的尺寸并不小。它是一个压缩的 tar 文件 (`mcb-kjv.tar.gz`) 或者一个 ZIP 文件 (`mcb-kjv.zip`)。这两个文件在解压缩后，都会生成一个 `mcb-kjv` 文件夹。

`mcb-kjv` 光盘是从 KJV 文本中发展出来的，而 KJV 又可以从无限圣经网站 (<http://www.unboundbible.org>) 中获得。在本书中，我已经重构了这些文本以让它们更适合我们的例子。`mcb-kjv` 光盘包括了描述我所作修改的笔记。

获取MySQL和相关软件

Obtaining MySQL and Related Software

如果你正准备访问一个 MySQL 服务器，你仅需要在你机器上安装 MySQL 客户端软件。如果是要运行你自己的服务器，你需要一个完全的 MySQL 发布光盘。

要想写你自己的基于 MySQL 的程序，那么你需要使用特定语言的 API 和服务器通信。Perl、Ruby、PHP 和 Python 接口都依靠 MySQL 的 C 语言 API 客户端来处理底层的客户端-服务器协议。对于 Perl、Ruby 和 Python，你必须先安装 C 语言客户端库及头文件。PHP 包含了 MySQL 客户端支持文件，但必须先在 MySQL 支持下编译才能使用。Java 中面向 MySQL 的 JDBC 本身就自行提供了客户端-服务器协议，所以它不需要 MySQL 的 C 语言库。

你也许不需要自己去安装客户端软件，也许其他人已经编译并安装了它们。这跟你拥有一个网络服务提供商（ISP）的用于计算服务的账号一样，一个 Web 服务器也有可能已经提供了 MySQL 的访问。在这种情况下，MySQL 库和头文件肯定已经由 ISP 安装过了。

MySQL

访问以下网址以获得 MySQL 的发布版本：

<http://dev.mysql.com/>

MySQL 包含安装指令，MySQL 参考手册还提供了扩展的安装信息。该手册在 MySQL 的网站上可获得，它的打印版本可从 MySQL 出版社购买。

如果你需要安装 MySQL 的 C 语言客户端库和头文件，那么你要从一个源发布处安装 MySQL，或者要使用一个编译好的库来安装 MySQL，而不是使用 RPM 二进制发布版本。在 Linux 下，你可以选择使用 RPM 来安装 MySQL，但需留意这样并不会安装客户端库和头文件，除非你安装开发版本的 RPM。（对于 MySQL 服务器而言，有好几个分开的 RPM 文件、标准的客户端程序、开发库以及头文件。）如果你不安装开发版本的 RPM，那么你就是加入了用户组的行列，这个组经常问的问题是：“我安装了 MySQL，可我找不到库或头文件，它们在哪里？”

Perl支持

一般的 Perl 信息可在以下站点获得：

<http://www.perl.org/>

Perl 软件能够从 CPAN 网站中获得：

<http://cpan.perl.org/>

要写一个基于MySQL的Perl程序，你需要DBI模块和MySQL特有的DBD模块，DBD::mysql。

要在 Unix 下安装这些模块，让 Perl 本身来帮助你是最容易的办法了。例如，要安装 DBI 和 DBD::mysql，运行一下命令（可能需要你以 root 的身份）：

```
# perl -MCPAN -e shell  
cpan> install DBI  
cpan> install DBD::mysql
```

如果最后一行命令抱怨说测试失败，就使用强制安装 DBD::mysql。在 Windows 下的 ActiveState Perl 中，你可以使用 ppm 功能：

```
C:\> ppm  
ppm> install DBI  
ppm> install DBD-mysql
```

你能使用 CPAN 脚本或 ppm 来安装本书中提到的其他的 Perl 模块。

Ruby支持

要获得 Ruby，请访问：

<http://www.ruby-lang.org/>

Ruby 的 DBI 模块，可在 RubyForge 中获得：

<http://rubyforge.org/projects/ruby-dbi/>

你需要最低 0.1.1 版本的 Ruby DBI 以使用本书中描述的所有特性，比如 option 文件支持和 SQLSTATE 支持。

Ruby 针对 MySQL 的 DBI 驱动需要 mysql-ruby 模块，你可在如下网址下载：

<http://raa.ruby-lang.org/project/mysql-ruby/>

在第18章中用到的 PageTemplate 包，可从 RubyForge 获得：

<http://rubyforge.org/projects/pagetemplate/>

如果你计划像第20章中描述的那样使用 session 支持，那么你需要 mysql-session 包，这个包可在 Ruby Application Archive 处获得：

<http://raa.ruby-lang.org/project/mysql-session/>

对于 mysql-session，获得这个包，解包，并安装它的 mysqlstore.rb 和 sqlthrow.rb 文件在你 Ruby 解释器能够查找的目录下。

PHP支持

PHP软件发布和安装指令可在如下网址获得：

<http://www.php.net/>

PHP 源发布包括 MySQL 客户端库，所以你不需要另外再获取这部分内容了。然而，当你配置系统时，你需要显式地允许 MySQL 支持。如果你使用一个二进制的版本，请确保它包含了 MySQL 的支持。

PHP 包括一个 pear 命令行辅助功能，你可以运行它以安装各种各样的 PEAR 模块。运行它并不需要输入参数。要安装 PEAR DB 模块以支持数据库访问，使用一下命令：

```
# pear install DB
```

在第18章中使用的 Smarty 模板包，请到以下网址获取：

<http://smarty.php.net/>

Python支持

Python 软件发布和安装指令可在如下网址获得：

<http://www.python.org/>

MySQLdb，以及提供 MySQL 支持的 DB-API 驱动模块，可在 SourceForge 处获得：

<http://sourceforge.net/projects/mysql-python/>

Java支持

你需要一个 Java 编译器来编译和运行 Java 程序。Javac 和 jikes 编译器是两个可能的选择。在很多系统中，你会发现它们其实已经被安装好了。否则，你可以获得一个编译器，它是 Java Software Development Kit (SDK) 的一部分。如果在你的机器上没有安装 SDK，那么请到以下网址获取 Solaris, Linux 和 Windows 版本的 SDK：

<http://java.sun.com/j2se/>

几个 Java 驱动提供了 MySQL 链接的 JDBC 接口。本书假设使用 MySQL Connector/J，你可以在以下网址下载：

<http://dev.mysql.com/downloads/>

Web服务器

在Web编程的章节中，我们使用Apache来为Perl、Ruby、PHP 和 Python 脚本提供服务，且使用 Tomcat 来为 JavaServer Pages 脚本服务。Apache 和 Tomcat可在 Apache Software Group 处获得：

<http://httpd.apache.org/>

<http://tomcat.apache.org/>

Apache Jakarta 项目网站提供了 Jakarta 的执行版本，该版本能执行 JSP 标准标签库，该标签库在本书编写 JSP 页面时常用到：

<http://jakarta.apache.org/taglibs/>

要获得更多关于配置 Apache 和 Tomcat 以运行基于MySQL的脚本，请看第17.2节和第17.3节。

从命令行执行程序

Executing Programs from the Command Line

当你阅读本书时，你会使用 MySQL 客户端程序运行很多示例代码，而且在 *recipes* 光盘中，还有许多的程序供你尝试。当然，本书的一个目的就是让你写出自己的 MySQL 程序。结果是，你经常需要通过命令行来执行程序，这就是说，在你的 shell 中或命令解释器中。为了能够最好地使用本书，你应能轻松运行 mysql（通过输入它的名字），同时你应能执行 *recipes* 中的程序或编写你自己的程序。为了达到这些目的，一个很重要的前提是正确地设置了 PATH 环境变量，并且你明白如何让程序变得可执行。本附录介绍了这些事情，当然如果你已经知道如何做了，那么请跳过。

设置环境变量

Setting Environment Variables

一个环境变量有一个值，你运行的程序都可以访问这个值。通过设置环境变量，你修改了你的运行环境。一个这样的变量是 PATH，但你的命令解释器在查找诸如 mysql 这样的程序时，它使用 PATH 来决定到哪些目录下去搜索。如果你的 PATH 变量设置正确了，那么无论你当前的目录是什么，你都可以轻松地调用程序。但如果 PATH 没有设置对，那么你的命令解释器将找不到某些程序。例如，如果 PATH 值不包括 mysql 被安装的目录，那么当你直接输入“mysql”尝试去运行 mysql 时，命令解释器就会报出一个“command not found”的错误。你必须输入程序的完全路径，或者改变当前目录到 mysql 的安装目录，这样才能运行 mysql。这两种方法都有些不方便，特别是当你需要重复输入各种命令时。因此，更好的办法就是设置你的 PATH，让它包含你想运行的程序的目录。

在其他的上下文中，其他的环境变量是非常重要的。例如，如果你运行的 Perl 或 Ruby 脚本使用到了你已经安装的模块，那么你或许需要设置 PERL5LIB 或 RUBYLIB 变量来告诉 Perl

或 Ruby 到哪里去寻找这些模块。对于 Java 而言，JAVA_HOME 变量应该被设置为指向 Java 安装的位置，同时 CLASSPATH 被设置为你的 Java 程序需要的库和类文件位置。

下面的讨论描述了如何设置环境变量。这些例子展示了如何修改你的 PATH 设置，从而让运行 MySQL 程序变得更加容易。然而，这些讨论同样适用于其他环境变量，这是因为变量设置的语法是相同的。

下面是一些通用的原则：

- 从命令行中设置一个环境变量，但该变量直到你退出命令行解释器后才会真的被设置。要想让变量设置马上生效，就要在 Unix 的恰当的 shell 启动文件中设置它，或者在 Windows 中使用环境变量设置接口。
- 一些变量（包括 PATH）有一个值，能够列出一到多个目录的路径名。在 Unix 下，可以很方便地使用冒号（：）来分隔这些路径名。在 Windows 下，路径名能包含冒号，所以分隔符是分号（；）。
- 在 Unix 下，你可以执行一个 env 或 printenv 命令，来检查你当前的环境变量设置。在 Windows 下，在控制台中执行 set 命令也有同样效果。

在 Unix 下设置 PATH 变量

你选择的命令解释器，决定了设置环境变量的语法。此外，对于你放入一个启动文件中的设置，使用哪个文件也是跟解释器密切相关的。下面的表格展示了 Unix 中常见的解释器的典型配置。如果你从没查看过你的命令行解释器的启动文件，那么从现在起尝试去熟悉它们的内容是个不错的主意：

Command interpreter	Possible startup files
sh, bash, ksh	.profile, .bash_profile, .bash_login, .bashrc
csh, tcsh	.login, .cshrc, .tcshrc

这个例子假设 MySQL 是安装在 /usr/local/mysql/bin 目录下的，而且你希望把这个目录添加到 PATH 值中。也就是说，例子中假设已经存在一个 PATH 设置。如果你当前还没有 PATH 设置，那么就在某个文件中添加一到多行。

- 对于属于 Bourne shell 家族的某个 shell（如 sh、bash、ksh），请查看你的启动文件以找到设置并导出 PATH 变量一行：

```
export PATH=/bin:/usr/bin:/usr/local/bin
```

改变这个设置，添加合适的目录：

```
export PATH=/usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin
```

赋值和导出也许是在不同的行中：

```
PATH=/bin:/usr/bin:/usr/local/bin  
export PATH
```

按如下方式更改设置：

```
PATH=/usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin  
export PATH
```

- 对于一个属于C shell族的shell (如csh、tcsh)，查看你的启动文件以找到一个setenv PATH 的命令：

```
setenv PATH /bin:/usr/bin:/usr/local/bin
```

改变设置，添加合适目录：

```
setenv PATH /usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin
```

同样可能的是，你的路径能通过一个 set path 命令来设置：

```
set path = (/usr/local/mysql/bin /bin /usr/bin /usr/local/bin)
```

path 并不是一个环境变量，但在 C shell 家族中，设置 path 跟设置 PATH 是等效的。

为在你系统中真正使用的路径名，调整指令。

在Windows下设置PATH变量

在 Windows 下设置环境变量，点击开始菜单，右键单击 我的电脑→属性→高级→环境变量。你就能看见一个窗口，该窗口允许你定义环境变量，或者改变已有变量的值。例如，你的 PATH 变量看起来或许是这样：

```
C:\WINDOWS;C:\WINDOWS\COMMAND
```

为了让运行 MySQL 程序比如 mysql 变得更容易，改变它的值以指向程序安装的目录。添加一个目录 C:\Program Files\MySQL\MySQL Server 5.0\bin，结果是：

```
C:\Program Files\MySQL\MySQL Server 5.0\bin;C:\WINDOWS;C:\WINDOWS\COMMAND
```

执行程序

Executing Programs

本节介绍如何从命令行中执行程序。你可以使用这些信息来运行你从recipes光盘中获取的

程序，或者是你自己编写的程序。第一个指令集可用于 Perl、Ruby、PHP 或 Python 编写的脚本。第二个指令集使用于 Java 程序。

下面的示例程序可在 `recipes` 光盘的 `progdemo` 目录下找到。

执行Perl、Ruby、PHP或Python脚本

下面讨论如何执行脚本，以Perl为例。当然这些原则同样适用于Ruby、PHP和Python脚本。

下面以一个名为 `perldemo.pl` 的脚本开始，它包含了这样一行：

```
print "I am a Perl program.\n";
```

一个可在任何平台上执行的脚本执行办法，是调用 `perl` 程序并告诉它该执行的脚本的名字：

```
% perl perldemo.pl  
I am a Perl program.
```

对于一个以其他语言编写的脚本而言，调用 `ruby`、`php` 或 `python` 程序。

当然，我们也可以设置一个脚本，让它直接执行。在 Unix 和 Windows 下，这个过程是不同的。这里分别讨论。

在 Unix 下，要让一个脚本可直接运行，在文件的顶部包含一个以 `#!` 开始的行，这行设置了程序的 pathname。这里有个脚本叫 `perldemo2.pl`，它以 `#!` 开始一行：

```
#!/usr/bin/perl  
print "I am a Perl program.\n";
```

下一步，使用 `chmod +x` 命令来设置可执行权限：

```
% chmod +x perldemo2.pl
```

从这点来看，脚本能够像它早先的对应体那样运行，但是现在你也可以直接输入它的名字以运行它。然而，假设该脚本被放置在你当前目录下，你的 shell 可能找不到它。shell 查找你的 `PATH` 环境变量中设置的目录，但为了安全，shell 的查找目录一般不包括当前目录 `(.)`。在这种情况下，你需要添加一个 `./` 路径符号来显式表明脚本的位置：

```
% ./perldemo2.pl  
I am a Perl program.
```

如果你将脚本拷贝到 `PATH` 设置中包含的一个目录下，那么调用这个脚本时，就不必再输入路径符号。

在 Windows 下，让脚本可执行的过程稍微有点不同。这里，不使用 chmod。取而代之的是，程序是否可执行完全依赖于它们的文件扩展名（比如 .exe 或 .bat）。Windows 允许你设置文件名关联，这样一个程序就被关联到有特殊后缀的文件上。这意味着，你能够设置关联以让 Perl、Ruby、PHP 或者 Python 来执行 .pl、.rb、.php 或 .py 程序。然后，你可以从命令行中调用一个给定后缀的脚本：

```
C:\> perl demo.pl  
I am a Perl program.
```

这里，不需要路径符号以表明我们调用的是当前目录下的脚本。因为 Windows 的命令解释器的搜索路径默认包括当前目录。

当你运行 recipes 光盘中的脚本时，请记住上述原则。例如，如果你当前在 metadata 目录下，且你希望执行 get_server_version.rb 的 Ruby 脚本，那么你可以这样：

```
% ruby get_server_version.rb  
% ./get_server_version.rb
```

在 Windows 下，你可以使用以下任一方式：

```
C:\> ruby get_server_version.rb  
C:\> get_server_version.rb
```

编译并执行 Java 程序

Java 程序需要一个软件开发包（SDK）。如果你的系统上还没有安装 Java 的 SDK，那么请获取并安装一个。对于 Solaris、Linux 或 Windows，Java 的 SDK 可在 java.sun.com 上下载。对于 Mac OS X，当前版本已经包含了 SDK，对于早先的版本没有包含 JDK 的版本，则可以到 connect.apple.com 上下载 javac 和其他的支撑来构建 Java 应用。

在确认安装了 Java 的 SDK 之后，设置 JAVA_HOME 环境变量。它的值应该是 SDK 被安装的目录。如果在 Unix 下 SDK 安装目录是 /usr/local/java/jdk，或者在 Windows 下是 C:\jdk，那么 JAVA_HOME 就是：

- 对于 Bourne shell 家族（sh、bash、ksh）：

```
export JAVA_HOME=/usr/local/java/jdk
```

如果你正使用最初的 Bourne shell、sh，你或许需要使用两个命令：

```
JAVA_HOME=/usr/local/java/jdk  
export JAVA_HOME
```

- 对于一个C shell家族 (csh、tcsh):

```
setenv JAVA_HOME=/usr/local/java/jdk
```

- 对于Windows, 到上面描述的环境变量窗口, 设置JAVA_HOME为:

```
C:\jdk
```

请根据你系统上的实际安装路径来调整上述指令。

当一个 Java SDK 已经装好, 且 JAVA_HOME 也已经设置好, 你就可以编译和执行 Java 程序了。下面是一个示例:

```
public class JavaDemo  
{  
    public static void main (String[] args)  
    {  
        System.out.println ("I am a Java program.");  
    }  
}
```

类的声明语句暗示了它的名字, 在这里是 JavaDemo。包含该程序的对应文件也应该匹配这个名字, 并包含一个 .java 扩展名。这样程序对应的文件名是 JavaDemo.java, 使用 javac 来编译程序:

```
% javac JavaDemo.java
```

如果你倾向于使用其他的 Java 编译器, 那就替换一下姓名好了。例如, 假设你想使用 Jikes, 请如下编译:

```
% jikes JavaDemo.java
```

Java 编译器生成编译后的二进制代码, 这是一个名为 JavaDemo.class 的文件。使用java 程序来运行类文件:

```
% java JavaDemo  
I am a Java program.
```

要编译和运行基于 MySQL 的用 Java 编写的程序, 你需要使用 MySQL Connecter/JJDBC 驱动 (第2.1节)。如果 Java 不能找到驱动, 你需要设置 CLASSPATH 环境变量。它的值应至少包括你当前目录(.) 和MySQL Connector/J驱动的目录。如果驱动的路径是 /usr/local/lib/java/lib/mysql-connector-java-bin.jar (Unix 下) 或者是 C:\Java\lib\mysql-connector-java-bin.jar (Windows 下), 那么把 CLASSPATH 设置为:

- 对于一个 Bourne shell 家族中的 shell (sh、bash、ksh):

```
export CLASSPATH=.: /usr/local/lib/java/lib/mysql-connector-java-bin.jar
```

- 对于 C shell 家族中的 shell (csh、tcsh):

```
setenv CLASSPATH .:/usr/local/lib/java/lib/mysql-connector-java-bin.jar
```

- 对于 Windows, 到前面描述的环境变量设置窗口, 并设置 CLASSPATH:

```
.;C:\Java\lib\mysql-connector-java-bin.jar
```

根据你系统上的路径信息, 调整上述指令。你也许需要添加其他的 class 目录或者库到你的 CLASSPATH 设置中, 具体的信息取决于你的系统。

JSP和Tomcat知识的初步内容

JSP and Tomcat Primer



本附录描述了一些JavaServer Pages (JSP) 编程的重要概念，这些概念早先在本书17章开始处就已经使用过。必需的背景是相当多的，这正是为什么这些内容要用一个单独的附录来陈述的原因。这儿讨论的主题有：

- 纵览 Servlet 和 JSP 技术；
- 启动 Tomcat 服务器；
- Tomcat 的目录结构；
- Web 应用的布局；
- JSP 页面元素。

想了解与 JSP、Servlet 和 Tomcat 的更多信息，请看附录D。

Servlet和JavaServer Pages综述

Servlet and JavaServer Pages Overview

Java Servlet 技术是一种在 Web 环境下高效执行 Java 程序的手段。Java Servlet规范定义了环境中的种种约定，总结如下：

- Servlet 在 Servlet 容器中运行，该容器运行在 Web 服务器中，或者跟另外一个容器通信。Servlet 容器通常还被叫做 Servlet 引擎。
- Servlet 容器接收到来自 Web 服务器的请求，并执行相应的 Servlet 来处理该请求。然后，该容器接受来自 Servlet 的反馈，并把它传送给 Web 服务器，服务器把内容传送给客户端。Servlet 容器因此在 Servlet 和 Web 服务器之间建立起一个连接。容器就像是 Servlet 的运行时环境，且决定了客户端请求和 Servlet之间的映射关系。此外，还有导入、执行和上传等。

- Servlet 根据已经建立的约定，和它们的容器通信。每个 Servlet 都要求执行众所周知的方法来应对各种请求。例如，GET 和 POST 请求将被传到doGet()方法和doPost()方法。
- 一个容器所能运行的 Servlet 被逻辑上组织为“上下文”。（例如，上下文可能会对应到被容器所组织的文档树上。）上下文也能包含资源，而不仅仅是 Servlet，比如 HTML 页面，或者配置文件。
- 一个上下文提供“Web应用”的基础，在 Java Servlet 规范中是这样定义基础的：

一个 Web 应用是由一组 Servlets、HTML 页面、类和其他资源所组成的。也就是说，一个应用是一组相关的 Servlets，它们在一起运行，而不会干扰其他不相关的 Servlet。

在一个给定应用上下文中的 Servlet 能够彼此共享信息，当在不同上下文中的 Servlet 则不行。例如，一个网关或登陆 Servlet 也许会建立一个用户的信任感，这个将被放入上下文中被其他 Servlet 共享，且大家都承认该用户已经正确登陆了。当这些 Servlet 发现必要的信任感在环境中并不存在时，它们将自动地重新定向到网关 Servlet 上，并要求用户登录。在其他上下文中的 Servlet 没有办法访问这些信任感信息。因此，上下文提供了一种安全机制，能制止一个应用随意调用另一个。同时，它还能够把一个应用从其他应用的崩溃中隔离开来。此时，容器继续运行未崩溃的应用，并重新启动崩溃的应用。

- 在若干级别的作用域中共享信息。这允许 Servlets 在一个作用域或跨多个作用域中处理请求。

接下来我们看看一个简单的 Servlet 是什么样的。它是一个执行 SimpleServlet 类的 Java 程序。该类有一个doGet()方法，该方法在容器接收到一个get请求时被调用。它还有一个doPost() 方法来处理post请求。它仅仅是一个调用doGet()的包装接口。SimpleServlet 生成一个简短的 HTML 页面，每次 Servlet 运行后，该页面都包含一些相同的静态文本。还有两个动态元素（当前日期和客户端 IP 地址），这两个信息在每个客户端请求时会有变化：

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```

public class SimpleServlet extends HttpServlet
{
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        PrintWriter out = response.getWriter ();

        response.setContentType ("text/html");
        out.println ("<html>");
        out.println ("<head>");
        out.println ("<title>Simple Servlet</title>");
        out.println ("</head>");
        out.println ("<body bgcolor=\"white\">");
        out.println ("<p>Hello.</p>");
        out.println ("<p>The current date is "
                    + new Date ()
                    + "</p>");
        out.println ("<p>Your IP address is "
                    + request.getRemoteAddr ()
                    + "</p>");
        out.println ("</body>");
        out.println ("</html>");
    }

    public void doPost (HttpServletRequest request,
                       HttpServletResponse response)
        throws IOException, ServletException
    {
        doGet (request, response);
    }
}

```

正如你毫无疑问会观察到的一样，这个“简单的”Servlet 事实上并不简单。它要求大量的机制来导入需要的类，并建立 `doGet()` 和 `doPost()` 方法以提供标准的接口给容器。比较这个 Servlet 和下面的 PHP 脚本，两者其实做了同样的事：

```

<html>
<head>
<title>Simple PHP Page</title>
</head>
<body bgcolor="white">

<p>Hello.</p>
<p>The current date is <?php print (date ("D M d H:i:s T Y")); ?>.</p>
<p>Your IP address is <?php print ($_SERVER["REMOTE_ADDR"]); ?>.</p>

</body>
</html>

```

Java Servlet 和 PHP 脚本之间的差别，表明了编写 Servlet 时的一个问题——重复性的过载：

- 最少数目的类必须导入每个 Servlet 中。
- 启动 Servlet 类的框架和 `doGet()` 或 `doPost()` 方法是非常模式化的，几个不同 Servlet 之间的差别也就是类的名字而已。
- 每段 HTML 代码都需要一个 `out.print` 语句。

前面的两点可以通过使用一个原型文件来解决，你可以在开始一个新的 Servlet 时直接拷贝该原型的内容。第三点（把每行 HTML 包裹在一个 `print` 语句中），并不是那么容易解决的，而且它很可能是 Servlet 编写中唯一最繁琐的地方。它还将导致其他问题：一个 Servlet 的代码像 Java 代码那样容易阅读，但有时非常难以了解代码生成的 HTML 结构。问题是事实上尝试去同时编写两种语言的代码（编写 Java，而 Java 在编写 HTML），这对这两种语言来说都不是件好事情。

JSP页面：另一种Servlet

创造 JavaServer Pages 的一个很大的原因，就是为了减轻在创建网页过程中一大堆 `print` 语句造成的负担。JSP 使用一种概念化的方法，这跟 PHP 很相似：手工编写 HTML 代码，并把代码嵌入到特殊的标记中。下面的代码显示了一个 JSP 页面，这跟上面的 Servlet 代码是等效的，但看起来很像 PHP 的脚本：

```
<html>
<head>
<title>Simple JSP Page</title>
</head>
<body bgcolor="white">

<p>Hello.</p>
<p>The current date is <%= new java.util.Date () %>.</p>
<p>Your IP address is <%= request.getRemoteAddr () %>.</p>

</body>
</html>
```

这个JSP页面在以下几个方面比Servlet更加简洁：

- 运行 Servlet 时并不需要再导入需要的标准类集合了，因为系统会自动为你做这件事。
- HTML 的编写更自然，不需使用 `print` 语句。
- 不需要类的定义，也不需要任何 `doGet()` 或 `doPost()` 方法。
- 不需要声明 `response` 和 `out` 对象，因为它们已经作为隐式对象为你构建好了。事实上，JSP 页面根本不需要显式地引用 `out`，因为它的输出自动就是往 `out` 写入的。
- 默认的内容类型是 `text/html`，因此也不必显式再作规定了。

- 脚本在特殊的标记中包括了 Java 代码。页面被显式为 `<%=` 和 `%>`，这意味着评估这个表达式，并输出它的结果。当然，还有其他的标记，每个都用于一个特定的目的。（请查阅本附录中的 JSP 页面元素以获取一个简短的总结。）

当一个 servlet 容器接收到一个 JSP 页面请求时，它把页面看成是一个包含了手工文本和可执行代码的模板。容器从模板中生成一个输出页面，并发送给客户端。模板中的手工文本不做改动，可执行代码被替换为执行后的结果，复合的结果作为对请求的反馈返回给客户端。这就是 JSP 处理过程的一个概念视图。当一个容器处理 JSP 请求时，事实上发生的是：

1. 容器将 JSP 页面翻译为一个脚本，这就是说，翻译成一个等价的 Java 程序。模板文字的实例被转为 `print` 语句，这些语句输出文本。代码的实例被放置到程序中，这样它们就能被执行以生成想要的效果。这就是放入一个包装中，该包装提供一个唯一的类名字，并包括 `import` 语句来导入 Servlet 需要的标准类集合。
2. 容器编译 Servlet 以生成一个可执行的类文件。
3. 容器执行类文件以生成一个输出页面，该页面作为请求的反馈被发送到客户端。
4. 容器缓存了可执行的类，这样当下次对应 JSP 页面的请求到来时，容器能够直接执行该类，以跳过翻译和编译阶段。如果容器注意到一个 JSP 页面被修改了（当一个针对该页面的请求到来时），它抛弃已经缓存的类，并重新编译该页面以生成一个可执行的类。

概念上，JSP 提供了一种比 Servlet 更自然的方法来编写 Web 页面。操作上，JSP 引擎在一个页面被安装在目录树下或被改动后，提供了自动编译。当你编写一个 Servlet 时，任何改变都需要重新编译该 Servlet，卸掉旧的，导入新的。这将导致我们过多关注 Servlet 本身，而不是 Servlet 要达到的目的。JSP 把这个关注倒了过来，这样你更多考虑的是该页面是做什么的，而不是编译和导入该页面的机制。

Servlet 和 JSP 之间的不同，并不意味着仅需要选择一个而忽略另一个。在一个容器中的应用上下文，可以两种都支持。这是因为 JSP 页面总能被转化为 Servlet，因此彼此间可以互通信。

JSP 跟其他一些技术很相似，以至于总能提供一种迁移的方法。例如，JSP 很像微软公司提出的 Active Server Pages (ASP)。但是，JSP 是中立于平台和提供商的，而 ASP 是微软公司所有的。因此，JSP 为那些寻找 ASP 之外技术的人们，提供了很具吸引力的选择。

定制行为和标签库

一个 Servlet 看起来很像一个 Java 程序，因为它就是一个 Java 程序。JSP 方法鼓励一种 HTML 和代码之间更清楚的分离，因为你不需要从 Java 的 print 语句中生成 HTML 了。在其他方面，JSP 不会要求 HTML 和代码的分离，所以如果你不小心的话，还是非常有可能编写出一个嵌入了大量 Java 代码的页面。

一种在 JSP 页面中避免过多 Java 代码的方法是使用另外一种 JSP 特性——定制行为。它是采用了特殊形式的标签，它们看起来很像 HTML 标签（因为它们被写为 XML 元素）。定制行为允许标签被定义为在它们出现的页面中执行任务。例如，一个 `<sql:query>` 标签或许会和一个数据库通信以执行一个查询。定制行为最典型的情况是成组出现的，这组被认为 是标签库：

- 这些标签执行的行为，是通过一组类来实现的。这些是常规的 Java 类，唯一的例外是它们是根据一组接口规范来编写的，这些规范使得 Servlet 容器能够以标准的方式跟它们通信。典型的，这组类被打包放入一个 JAR 文件中。
- 库包含了一个标签库描述子（TLD）文件，该文件规定了每一个标签如何对应到相应的类中。这允许 JSP 处理器决定应该为 JSP 页面中出现的定制的标签调用哪个类。TLD 文件还规定了每个标签的行为，比如它是否有任何需要的属性。这些信息被用在页面翻译时间，以决定一个 JSP 页面是否正确地使用了库中的标签。例如，如果一个标签需要特定的属性，且该标签被在某个没有该属性的页面中，那么处理器就能觉察出问题，并发出一个恰当的报错信息。

使用标签概念来编写整个页面，比在标签和 Java 代码间不停切换要容易多了。这个概念很像 JSP，而不像 Java。但在 JSP 页面中放置一个定制标签的效果，跟调用一个方法是相似的。这是因为在 JSP 中的一个标签索引，对应了翻译后生成的 Servlet 中的一个方法调用。

为示范嵌入 Java 代码和标签库两种方法的不同，比较以下两段 JSP 脚本，它们建立了一个通往 MySQL 服务器的链接，并显示了在 cookbook 数据库中的一组表。第一个并不使用嵌入 Java 代码的方法：

```
<%@ page import="java.sql.*" %>
```

```

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>

<%
Connection conn = null;
String url = "jdbc:mysql://localhost/cookbook";
String user = "cbuser";
String password = "cbpass";

Class.forName ("com.mysql.jdbc.Driver").newInstance ();
conn = DriverManager.getConnection (url, user, password);

Statement s = conn.createStatement ();
s.executeQuery ("SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES"
    + " WHERE TABLE_SCHEMA = 'cookbook'"
    + " ORDER BY TABLE_NAME");
ResultSet rs = s.getResultSet ();
while (rs.next ())
    out.println (rs.getString (1) + "<br />");
rs.close ();
s.close ();
conn.close ();
%>

</body>
</html>

```

下面，用标签库来做同样的事情，比如使用JSP标准标签库（JSTL）。JSTL由几个按照功能分组的标签集组成。通过使用它的核心和数据库标签，之前的JSP页面可以按照如下方式改写以完全避免字面化Java的使用：

```

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>

<sql:setDataSource
    var="conn"
    driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/cookbook"
    user="cbuser"
    password="cbpass"
/>

```

```

<sql:query dataSource="${conn}" var="rs">
    SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_SCHEMA = 'cookbook'
    ORDER BY TABLE_NAME
</sql:query>
<c:forEach items="${rs.rowsByIndex}" var="row">
    <c:out value="${row[0]}"/><br />
</c:forEach>

</body>
</html>

```

taglib 指示语指定了页面使用的TLD文件，同时还通过前缀c和sql来暗示对应标签组能执行的操作。（事实上，一个前缀为一组标签建立了一个名字空间。）<sql:datasource>标签构建了连接一个MySQL服务器的参数，<sql:query>发放出一个查询，<c:forEach> 在结果中循环，<c:out>把每个表名添加到输出页面中。（更多细节，请查看第17.3节。）

你很可能会以跟你应用上下文中大多数 JSP 页面相同的方式，连到数据库服务器，你还可以移动 <sql:datasource> 标签来包含一个文件，这是一种更加简化的方法。如果你把文件命名为 jstl-mcb-setup.inc，并把它放置在应用的 WEB-INF 目录下，那么该应用上下文中的任何页面，都可以用一个 include 指示语访问该文件，从而建立通往 MySQL 服务器的链接：

```

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@ include file="/WEB-INF/jstl-mcb-setup.inc" %>

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>

<sql:query dataSource="${conn}" var="rs">
    SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_SCHEMA = 'cookbook'
    ORDER BY TABLE_NAME
</sql:query>
<c:forEach items="${rs.rowsByIndex}" var="row">
    <c:out value="${row[0]}"/><br />
</c:forEach>

</body>
</html>

```

当你使用标签库时，你还可以使用 Java，因为标签行为映射到 Java 类的调用上了。但是这个概念须服从 XML 的规范，这样它就不太像是编写程序代码，而更像是编写 HTML 页面元素了。如果你的组织使用 "separation of powers" 工作流来生成 Web 内容，定制行为允许页面中的每个元素都动态生成，并以一种设计者或非程序员容易处理的方式打包。它们并不开发类，或不直接使用类来工作，那些是程序员的工作。

启动Tomcat服务器

Setting Up a Tomcat Server

前面的讨论提供了一个关于 Servlet 和 JSP 的简短介绍，但还没有提及如何使用一个服务器来运行它们。本节描述如何安装 Tomcat，一个支持 JSP 的 Web 服务器。就像 Apache 一样，Tomcat 是一个 Apache 软件基金组织的开发成果。

正如前面描述的那样，Servlet 在一个容器内执行，而容器则是一个与 Web 服务器通信或插在 Web 服务器上的引擎，它处理页面的请求，并执行 Servlet 以生成页面。一些 Servlet 容器能独立运行，在这种情况下它们既是容器，又是 Web 服务器，这正是 Tomcat 的工作方式。安装 Tomcat 后，你获得一个完全工作的，能处理 Servlet 的服务器。事实上，Tomcat 是一个面向 Servlet 和 JSP 规范的参考执行，所以它也可以作为 JSP 引擎来运行，提供从 JSP 到 Servlet 的翻译服务。它的 Servlet 容器部分命名为 Catalina，而 JSP 处理器部分命名为 Jasper。

把 Tomcat 的容器部分与其他 Web 服务器放在一起运行是可能的。例如，你可以在 Apache 和 Tomcat 之间启动一个合作安排，这里 Apache 作为一个前端把 Servlet 和 JSP 请求传递给 Tomcat，同时自己处理其他的请求。你可以在 Tomcat 的网站上找到关于如何设置 Apache 和 Tomcat，并让两者协同工作的信息。

要运行一个 Tomcat 服务器，你要做三件事情：

准备一个Java软件开发包（SDK）

这是必须的，因为 Tomcat 需要编译 Java Servlet。如果你在阅读前面章节的过程中，编译过 Java 程序，那么你很可能已经安装了一个 SDK。如果你没有 SDK，请翻阅附录 B 以了解如何获取和安装一个 SDK。

得到Tomcat

Tomcat 可以从 <http://tomcat.apache.org> 上获得。大量的 Tomcat 文档也在这个网站上。特别如果你是一个 Tomcat 新手，我推荐你阅读 Introduction and the Application Developer's Guide。

学习一些XML知识

Tomcat 配置文件是用 XML 写的,而且 JSP 页面中的很多脚本元素都遵守XML语法原则。如果你是一个 XML 新手,那么《XML and XHTML in a Nutshell》的第17.0节描述了一些 XML 特性,并把 XML 与 HTML 做了些比较。

安装一个Tomcat

当前,可用的 Tomcat 有若干版本 (4.1、5.0 和 5.5)。本节描述的是 Tomcat 5.0,具体的内容也适用于 Tomcat 5.5,尽管你可能需要升级你的 Java 版本 (Tomcat 需要JS2E 5.0)。

要安装 Tomcat,请从 *tomcat.apache.org* 获取一个二进制发布包。(我假设你不需要从源代码编译它,那是比较困难的。) Tomcat 的发布包有几种文件格式:

.tar.gz

一个压缩的 tar 文件,通常用于 Unix 安装

.zip

一个 ZIP 存档文件,可用于 Unix 和 Windows

.exe

一个可执行的安装文件,仅使用于 Windows

如果你使用的是 Linux,那么 Linux 的服务提供商也许会在分发的媒介上,或者通过一个在线的 APT 或 RPM 知识库来提供 Tomcat。这里就不深入介绍这种方法了。请浏览你的 Linux 分发文档,或者联系你的服务提供商以了解更多细节。

对于 tar 或者 ZIP 文件,你应该把它放置在你想安装 Tomcat 的目录下,然后在这个目录下运行安装命令,解包。Windows 下的 .exe 安装器会要求你选定在哪儿安装 Tomcat,因此它可在任意目录下运行。下面的命令是非常典型的:

从一个压缩的 tar 文件中安装 Tomcat,先解包:

```
% tar zxf jakarta-tomcat-5.0.28.tar.gz
```

如果你在 Mac OS X 下解包时遇到困难,请使用 gnutar 而不是 tar。在老版本的 Mac OS X 上, tar 在处理长文件名时有些问题,而 gnutar 没有。在 Solaris下,也许要使用与 GNU 兼容的 tar 版本。

如果你的 tar 版本不能理解 z 选项,那么请:

```
% gunzip -c jakarta-tomcat-5.0.28.tar.gz | tar xf -
```

如果你使用一个 ZIP 存档文件，你可以使用 jar 功能来解包，或者使用任何能读懂 ZIP 格式的程序。比如，使用 jar：

```
% jar xf jakarta-tomcat-5.0.28.zip
```

Windows下的 .exe 分发是可以直接执行的。运行它，然后告诉它你希望把 Tomcat 放到哪里。如果你使用这个安装器，请确认你已经先安装了一个 Java SDK。该安装器把一些文件放到 SDK 目录结构中，如果 SDK 不存在的话，那么有些操作会失效。对于有服务管理的 Windows 系统（比如 Windows NT、2000 或 XP），.exe 安装器提供了额外的选项，询问你使用把 Tomcat 安装为一个服务，这样它在系统启动时就能自动地启动了。

大部分安装方法创建一个目录，并把 Tomcat 解包到这个目录中。解包的结果是，顶层的目录就是 Tomcat 的根目录。这里，我假设 Tomcat 在 Unix 下的根目录是 /usr/local/jakarta-tomcat。在 Windows 下的根目录是 C:\jakarta-tomcat。Tomcat的根目录包含了各种文本文件，这些文件介绍了很多有用的信息，帮助你处理一般的或平台相关的问题。它还包含了一组目录，如果你想现在浏览这些目录，请查阅 Tomcat的目录结构那一节。否则，请继续前进。

启动和停止Tomcat

Tomcat 能被手工控制，且被设置为当系统启动时自动运行。熟悉 Tomcat 在你平台上的启动和关闭命令，是非常有用的，因为你很可能发现需要经常手工地启动和停止 Tomcat，至少是当你刚开始设置它时。例如，你修改了 Tomcat 的配置文件，或安装了一个新的应用，你就必须重启 Tomcat 以让它注意到你做的改变。

在运行 Tomcat 之前，你需要设置一些环境变量。请确认 JAVA_HOME 已经设置为你的SDK 目录，这样 Tomcat 才能找到它。你也许还需要设置 CATALINA_HOME 为 Tomcat 的根目录。

要在 Unix 下手工启动和停止 Tomcat，请移动到 Tomcat 根目录下的 bin 目录。我发现把这个目录下的脚本设置为可执行是非常有用的：

```
% chmod +x *.sh
```

你可以通过以下两个 shell 脚本来控制 Tomcat：

```
% ./startup.sh  
% ./shutdown.sh
```

要想在系统启动时，自动运行Tomcat，请查找一个启动脚本如/etc/rc.local或者/etc/rc.d/rc.local（具体的路径名称依赖于你所使用的操作系统），添加以下几行：

```
export JAVA_HOME=/usr/local/java/jdk  
export CATALINA_HOME=/usr/local/jakarta-tomcat  
$CATALINA_HOME/bin/startup.sh &
```

这些命令会以 root 身份运行 Tomcat。要想使用另一个用户账号运行 Tomcat，请把最后一行命令改成：

```
su -c $CATALINA_HOME/bin/startup.sh user_name &
```

如果你使用 su 来设置一个用户名，请确认 Tomcat 的目录树对于这个用户是可访问的，否则你将遇到文件权限问题。一种方法，是在 Tomcat 的根目录下，以 root 身份运行以下命令：

```
# chown -R user_name .
```

那些从服务提供商处安装了 Tomcat 的 Linux 用户，也许会发现安装过程创建了一个名为 tomcat 的脚本，该脚本在 /etc/rc.d/init.d 目录下，可用于手工或自动启动。要手工使用这个脚本，请移动到这个目录下并执行：

```
% ./tomcat start  
% ./tomcat stop
```

对于自动启动，你必须以 root 的身份运行以下命令来激活脚本：

```
# chkconfig --add tomcat
```

Linux 的包安装也许还会创建一个用户账号，登陆名可能是 tomcat 或者 tomcat5，这个账号是用于运行 Tomcat 的。

对于 Windows 用户，发布包中有一对 batch 文件，在 bin 目录下用于手工控制 Tomcat：

```
C:\> startup.bat  
C:\> shutdown.bat
```

如果你选择把 Tomcat 作为一个服务运行在有服务管理的 Windows 下，比如 Windows NT、2000或者 XP，你应该使用服务控制台来管理 Tomcat。你可以使用这个来启动或中止 Tomcat,或者设置 Tomcat 以让它在 Windows 启动时自动运行。

要尝试一下 Tomcat，使用适合于你平台的指令，然后使用你的浏览器来请求默认的页面，页面的 URL 看起来像这样：

```
http://localhost:8080/
```

你可以适当调整主机名和端口号。例如，Tomcat 一般运行在 8080 端口，但如果你在 Linux 下以包文件安装 Tomcat，它可能会使用 8180 端口。如果你的浏览器正确接收到页面，那么你会看到 Tomcat 的标识，以及指向示例和文档的链接。在这里，点击示例链接并尝试一些 JSP 页面将是非常有用的，可以了解它们是否正确地编译和执行了。

如果你发现，尽管 JAVA_HOME 已经设置了，Tomcat 还是不能找到你的 Java 编译器，请尝试设置你的 PATH 环境变量以显式地包含该编译器所在的目录。一般地，这是你 SDK 的 bin 目录。你可能已经把 PATH 设置成别的什么了。如果是那样，你需要添加这个 bin 目录的路径信息。

Tomcat 的目录结构

Tomcat's Directory Structure

要编写 JSP 页面，并不是特别需要去熟悉 Tomcat 的目录布局。但这么做当然不会有什么损伤，所以移动到 Tomcat 的根目录下看看。你会发现，一大堆标准的目录是按照功能来组织的，这些目录在后续的讨论中将会描述。需要注意的是，你的安装布局也许跟这里描述的不完全一样。一些发布省略了部分目录，而且 logs 和 work 目录也许直到你第一次启动 Tomcat 后才会创建。

应用目录

从一个应用开发者的角度来看，webapps 目录是 Tomcat 目录结构中最重要的部分。每个应用上下文都有它自己的目录，这往往放在 webapps 目录下。

Tomcat 通过把客户端请求映射到 webapps 目录下的某个位置，来处理这些请求。对于一个以 webapps 下某目录名字开始的请求，Tomcat 在该目录中查找合适的页面。例如，Tomcat 处理下面两个请求时，使用 mcb 目录中的 index.html 和 test.jsp：

```
http://localhost:8080/mcb/index.html  
http://localhost:8080/mcb/test.jsp
```

对于那些不以 webapps 目录下某个子目录名字开始的请求，Tomcat 从一个特殊的目录 ROOT 下处理它们，这个目录提供了默认的应用上下文（注1）。对于下面的请求，Tomcat 请求 ROOT 目录下的 index.html 页面：

注1： webapps/ROOT 目录与 Tomcat 的根目录不同，它是 Tomcat 目录树的顶级目录。

<http://localhost:8080/index.html>

典型的情况是，应用程序被打包成 Web 存档文件（WAR）。默认情况下，Tomcat 在启动时，会查找它需要解包的 WAR 文件。这样，要安装一个应用，你一般需要把它的WAR 拷贝到 webapps 目录，重启 Tomcat，然后让 Tomcat 把它解包。单独一个应用的布局，将在 Web 应用结构中描述。

一个 Web 应用是一组相关的 Servlet，它们在一起工作，且不会感染其他的不相关的 Servlet。这对 Tomcat 的最大意义是，一个应用就是在 webapps 目录下某个子目录内的所有内容。因为在 Tomcat 这样的 Servlet 容器中，上下文是分离的，这种结构一个现实的提示就是，一个应用目录下的脚本不会跟其他应用目录下的任何东西混淆。

配置和控制目录

有两个目录包含了配置和控制文件。bin 目录包含了与 Tomcat 启动和关闭相关的控制脚本，conf 目录包含了 Tomcat 的配置文件，这些文件是 XML 文档。Tomcat 只有在启动时，才会读入这些配置文件。如果你修改了任一个配置文件，那么你必须重新启动 Tomcat 来让你的改变生效。

最重要的配置文件是 server.xml，它控制了 Tomcat 的总体行为。另外一个文件 web.xml，提供了应用配置的默认值。这个文件是和 web.xml 一起使用的。而文件 tomcat-users.xml 为那些能够访问受保护的用户，定义了具体细节，这样 Manager 应用就能允许你从浏览器中控制应用。这个文件可以被其他的用户信息存储机制限制住。例如，你可以在 MySQL 中存储 Tomcat 的用户记录。对于具体的做法，请浏览 recipes 中的 tomcat 目录。

类目录

几个 Tomcat 目录被用于类文件和库。它们在功能上是有区别的，这跟你想要让类对应用、Tomcat 或两者可见有关系：

shared

这个目录有两个子目录，classes 和 lib，用于存放应用可见但 Tomcat 不可见的类文件和库。

common

这个目录有两个子目录，classes 和 lib，用于存放对应用和 Tomcat 都可见的类文件和库。

server

这个目录有 classes 和 lib 两个子目录, 用于存放 Tomcat 可见但应用不可见的类文件和库。

操作目录

如果它们不是在 Tomcat 安装过程中被建立的, 那么出于操作上的目的, Tomcat 会在第一次运行时, 创建两个额外的目录。Tomcat 把日志文件写入 logs 目录, 并使用一个 work 目录来存储临时文件。

在 logs 目录中的文件, 可用于调试问题。举个例子, 如果 Tomcat 有一个启动时的问题, 该原因很可能会写入某个日志文件中。

当 Tomcat 把一个 JSP 页面翻译成 Servlet 并编译成一个可执行的类文件时, 它把结果的 .java 和 .class 文件存入 work 目录。(当你首次使用JSP页面工作时, 你可能会发现在工作目录下对JSP页面与相应的Tomcat生成的serlets进行对比是显而易见的。)

在无须重启 Tomcat 的情况下, 重启应用

Applications Without Restarting Tomcat

如果你修改了一个 JSP 页面, Tomcat 在该页面下一次被请求时, 会自动重新编译它。但如果该页面使用在应用的 WEB-INF 目录下的一个 JAR 或一个类文件, 且你更新了其中的一个, 那么 Tomcat 通常直到你重启它后才会注意到这种改变。

一种避免为一个应用而重启 Tomcat 的方法, 是在 Tomcat 的 server.xml 文件中, 为应用提供一个 `<Context>` 元素, 该元素设置 `reloadable` 属性为 `true`。这会促使 Tomcat 不仅去查找 JSP 页面的改变, 而且去留意在 WEB-INF 目录下的类和库文件的改变。例如, 一个名为 mcb 的应用, 你可以在 server.xml 中这样添加:

```
<Context path="/mcb" docBase="mcb" debug="0" reloadable="true"/>
```

`<Context>` 元素的属性告诉 Tomcat 4 件事情:

path

指示从应用映射到上下文的 URL。它的值是在主机名和端口号后的 URL 值。

docBase

只是应用上下文目录放置的位置, 该位置相对与 Tomcat 中的 webapps 目录。

debug

设置上下文调试级别。值为 0 则关闭调试输出，越高的值生成越多输出。

reloadable

但一个客户端请求某个 JSP 页面时，规定了 Tomcat 再编译的行为。默认情况下，Tomcat 仅在注意到一个页面被修改时，才重新编译该页面。设置 reloadable 为 true，就是告诉 Tomcat 还得去检查应用的 WEB-INF 目录下类和库文件的改动。

在修改了 server.xml 以添加 <Context> 元素后，重新启动 Tomcat 才能生效。

让 Tomcat 来检查类和库文件的改变，在应用开发阶段是非常有用的，因为它能避免重复的启动。然而，正像你可能认为的那样，自动的类检查带来了额外的处理负担，且导致严重的性能损伤。因为在开发系统上，而不是产品系统上使用这个选项，会更好。

另外一种让 Tomcat 理解应用改变，且不需重启整个服务器的方法，是使用 Manager 应用。这允许你从一个浏览器中重载某个应用，而不会带来 reloadable 属性设置那样的负载问题。Manager 应用可以通过在 URL 中使用 /manager 来调用。例如，下面的请求表明了哪一个内容正在运行：

`http://localhost:8080/manager/list`

要关闭和重载 mcb 应用，而不需重启 Tomcat，可以这样：

`http://localhost:8080/manager/reload?path=/mcb`

更多关于 Manager 应用，以及它允许的命令的信息，请查看：

`http://tomcat.apache.org/tomcat-5.0-doc/manager-howto.html`

这个文档也许在你的 Tomcat 服务器的主页上也可以找到。须特别注意的是描述使用 manager 角色创建一个 Tomcat 用户的那段，因为你可能需要提供一个名字和密码以获得 Manager 应用的访问权限。默认情况下，用户记录被定义在 Tomcat 的 tomcat-users.xml 配置文件中。recipes 光盘中的 tomcat 目录包含了在 MySQL 中存储 Tomcat 用户记录的相关信息。

Web应用结构

Web Application Structure

每一个 Web 应用都对应一个单独的 Servlet 上下文，且存在一组资源。其中某些资源对客户端是可见的，而其他的则不可见。例如，一个应用的 JSP 页面是客户端可用的，但配置、资产或者类文件这些 JSP 页面使用的东西，是隐藏的。应用结构中的组建，决定了客户端是否能看见它们。这允许你把资源设置为 public 或者 private。

Java Servlet 规范 2.4 定义了 Web 应用布局的标准。这提供了规范，能够帮助应用开发者决定哪里放什么，而且还明白哪些对客户端是可见的。

每个 Web 应用都对应了一个单独的 Servlet 上下文。在 Tomcat 中，这些都被表示为 webapps 目录下的子目录。在一个应用的目录下，你会发现 WEB-INF 子目录，而且往往还有其他的文件诸如 HTML 页面、JSP 页面，或者图片文件。这些放置在应用的根目录下的文件，对客户端是可见的。WEB-INF 目录有特殊的重要性，它仅仅提示 Tomcat 它的父目录是一个应用目录。WEB-INF 因此是一个 Web 应用唯一必须具备的组件。它必须存在，即使它是空的。如果 WEB-INF 非空，那么典型情况下它包含与应用相关的配置文件、类文件或者还有其他信息。三个主要的组件为：

```
WEB-INF/web.xml  
WEB-INF/classes  
WEB-INF/lib
```

web.xml 是 Web 应用发布的描述文件。它给容器提供了一种标准的方法，让容器发现如何处理应用中的资源。发布描述文件通常情况下被用来设置受保护信息的访问权限，规定了发生错误时使用的错误页面，且定义了到哪里去寻找标签库。

在 WEB-INF 下的 classes 和 lib 目录，包含了类文件和库文件，有时还包含其他信息。单独的类文件放在 classes 下，且使用一个对应到类结构的目录结构。（例如，一个实现名为 com.kitebird.jsp.MyClass 的类的文件 MyClass.class 可能被存储在 classes/com/kitebird/jsp 目录下。）类库被打包为 JAR 文件，放在 lib 目录下。Tomcat 在处理页面请求时，自动查找 classes 和 lib 目录。这使得你的页面可以最小程度地使用应用相关的信息。

目录 WEB-INF 是私有的。它的内容仅能被应用的 servlet 和 JSP 页面访问，但不能被浏览器直接请求。所以你可以把一些不应显示给客户端的信息放在这里。例如，你可以把一

个包含数据库链接参数的属性文件存储在 WEB-INF 中。如果你已经有一个应用，它允许图片文件被一个文件上载，然后被另一个文件下载，那么把图片放在 WEB-INF 下的某个目录内，让它们成为私有的。因为 Tomcat 不会直接发布 WEB-INF 中的内容，你的 JSP 页面可以执行一个访问控制策略，决定谁能执行图片操作，一个简单的策略是在允许上传图片之前要求客户指定账号和密码。WEB-INF 目录还有其他的用处，比如它给你一个存放私有文件的位置，这个位置与应用的根目录相关，而不论你在何处部署应用。

客户端尝试在请求中包含诸如 Web-Inf 这样的名字，以绕过 WEB-INF 的私有本性，但很快就发现它的名字是大小写敏感的，即使在大小写并不敏感的系统如 Windows 和 HFS+ 上。在这些系统中，你应该小心不要用 Web-Inf、web-inf 这样的名字来创建 WEB-INF 目录。操作系统本身不会考虑这些名字的差别，但是 Tomcat 会。结果是你的 JSP 页面根本无法访问在这个目录中的资源。在 Windows 上，你许要从一个控制台窗口中创建 WEB-INF 目录。(Windows Explorer 浏览器在创建或者重命名目录时可能对你所使用的字符并不怀疑，这是因为它并不需要像命令终端窗口中的命令行输入的 DIR 命令那样显示目录名称。)

前面讨论了使用目录结构来描述 Web 应用的布局。因为这是最简单的解释方法。然而，一个应用并不需要这样的解释。典型情况下，一个 Web 应用被打包成一个 WAR 文件，使用的是 Servlet 规范描述的布局方式。但是相同的容器能直接从一个 WAR 文件中运行对应的应用，而无需解包。更进一步，一个容器可以解包 WAR 文件，如果它愿意的话。

Tomcat 使用了最简单的方法，即使把一个应用用一个目录结构存储在文件系统中，该目录结构跟应用最初创建时是一样的。你可以比较 WAR 文件的结构跟 Tomcat 解包后创建的目录树，来观察两者之间的对应关系。例如，一个应用 someapp 的 WAR 文件可被这样解压：

```
% jar tf someapp.war
```

该命令展示的一列路径名，对应了 Tomcat 解压 someapp 时生成的应用目录。要验证这一点，请使用以下命令递归地列出 someapp 的目录。

```
% ls -R someapp          (Unix下)
C:\> dir /s someapp      (Windows下)
```

如果你想用手工的方式为一个应用 myapp 创建上下文，那么你大概会经过这些步骤（如果你想知道导致的应用分级是什么样，请参看章节分布的tomcat/myapp目录。）：

1. 移动工作目录到 Tomcat 目录下的 webapp 子目录。
2. 创建一个名为 myapp 的子目录，在 webapps 目录下创建一个与应用（myapp）同名的目录，然后将操作目录更改至该目录。
3. 在 myapp 目录下，创建一个 WEB-INF 目录。这个目录的存在告诉 Tomcat，myapp 是一个应用上下文，所以它是必须存在的。然后重启 Tomcat，这样它就发现了新的应用：myapp。
4. 在 myapp 目录下，创建一个测试页面 page1.html，然后使用浏览器请求该页面以确认 Tomcat 能正常工作。这是一个空白的 HTML 文件，以避免使用嵌入式 Java 代码、标签库或类似东西带来的复杂性。

```
<html>
<head>
<title>Test Page</title>
</head>
<body bgcolor="white">
<p>
This is a test.
</p>
</body>
</html>
```

要请求一个页面，请在浏览器中输入如下 URL：

<http://localhost:8080/myapp/page1.html>

5. 尝试一个简单的 JSP 页面（尽管它包含不可执行代码），拷贝 page1.html 并命名为 page2.jsp。这创建了一个合理的 JSP 页面，然后你可以请求它：

<http://localhost:8080/myapp/page2.jsp>

6. 拷贝 page2.jsp 到 page3.jsp 中，并修改后者以包含一些嵌入式的 Java 代码，以打印当前的日期和客户端代码：

```
<html>
<head>
<title>Test Page</title>
</head>
<body bgcolor="white">
<p>
This is a test.
The current date is <%= new java.util.Date() %>.
Your IP number is <%= request.getRemoteAddr () %>.
</p>
```

```
</body>
</html>
```

这里，`Date()` 方法返回当前的日期，`getRemoteAddr()` 方法返回客户端的 IP 地址。在做了这些改变之后，从你的浏览器中请求 `page3.jsp`，你会发现输出的信息中包含了当前日期和客户端主机的 IP 地址。

到这里，你已经拥有了一个简单的应用上下文，它包含了三个页面和一个空白的WEB-INF 目录。对大多数应用而言，WEB-INF 会包含 `web.xml` 文件，它作为 Web 应用的发布描述文件，告诉 Tomcat 该如何配置应用。如果你查看其他应用中的 `web.xml`，你会发现它们是非常复杂的，但一个最小的描述文件看起来可以是这样：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

</web-app>
```

往 `web.xml` 中添加信息，是一个在 `<web-app>` 和 `</web-app>` 标签之间放置新元素的过程。作为一个简单的示例，你可以添加一个 `<welcome-file-list>` 元素以规定一系列的文件，Tomcat 在客户端的请求 URL 以 `myapp` 结尾且没有具体页面时，会查看这些文件。例如，假设规定了 Tomcat 应该考虑 `page3.jsp` 和 `index.html` 作为合法的默认页面，那么如下创建 `web.xml` 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

<welcome-file-list>
  <welcome-file>page3.jsp</welcome-file>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>

</web-app>
```

重启 Tomcat，这样它会读入新的应用配置信息，然后发出以下请求：

```
http://localhost:8080/myapp/
```

myapp 目录包含了一个名为 page3.jsp 的页面，它在 web.xml 中被设为一个默认的页面，所以 Tomcat 会执行 page3.jsp 并把结果发送给你的浏览器。



JSP页面元素

Elements of JSP Pages

本附录早先的章节描述了 JSP 页面的一般特性。本节讨论更多细节：

JSP 页面是模板，它们包含静态的部分和动态的部分：

- 在 JSP 页面中的字符文本，是不包含在特殊标记中的，这部分内容不经改变地发送到客户端。在本书中的 JSP 示例生成 HTML 页面，所以 JSP 脚本的静态部分是以 HTML 编写的。但你可以编写页面以生成其他格式的输出，比如无格式的文本、XML 或 WML
- JSP 页面的非静态（动态）部分，包含了需要计算的代码。你用特殊的标记，把代码和静态文本分开。一些标记指示了页面处理指示语或者脚本。一个指示语给 JSP 引擎提供了关于如何处理页面的信息，而脚本段是一个小型程序，它将被自己的输出结果所替代。其他的标记都编写得有如 XML 元素，它们是与类关联在一起的，这些类将作为标签句柄来执行你想要的操作

下面讨论 JSP 页面能够包含的各种动态元素的类型：

脚本元素

有几组脚本标记允许你把 Java 代码或注释加入一个 JSP 页面中：

<% ... %>

<% 和 %> 标记表明了一个脚本段，这就是说，Java 代码。下面的脚本调用 print() 来写一个值到输出页面：

```
<% out.print (1+2); %>
```

<%= ... %>

这些标记表示一个需要计算的表达式。计算结果被加入输出页面，这些标记使得显示数值变得很容易，因为它不需要显式的 print 语句。例如，下面两行代码都显示 3，但第二个更短些：

```
<% out.print (1+2); %>
<%= 1+2 %>
```

<%! ... %>

<%! 和 %> 标记允许声明类变量和方法。

```
<%-- ... --%>
```

在这两个标记中的文本，将被当作注释并被忽视。JSP 注释将不在输出中出现，如果你编写一个 JSP 页面，且你希望注释能出现在最后的 HTML 中，那么请使用 HTML 的注释语法：

```
<%-- this comment will not be part of the final output page --%>
<!-- this comment will be part of the final output page -->
```

当一个 JSP 页面被翻译成一个 Servlet 时，所有的脚本元素都有效地变成了该 Servlet 的一部分。这意味着，一个在某元素中声明的变量，在其他后续的元素中也能被使用。这同样意味着，如果你在两个元素中声明一个变量，结果将是非法的。

标记 `<% ... %>` 和 `<%! ... %>`，都可以被用于声明变量，但效果有所不同。一个在 `<% ... %>` 中声明的变量，是一个对象变量，它在每次页面被请求时都初始化。而在 `<%! ... %>` 中声明的是一个类变量，它仅在页面第一次被请求时初始化。请考虑如下 JSP 页面 `counter.jsp`，它声明了 `counter1` 和 `counter2` 两个变量：

```
<%-- counter.jsp - demonstrate object and class variable counters --%>

<% int counter1 = 0; %>      <%-- object variable --%>
<%! int counter2 = 0; %>     <%-- class variable --%>
<% counter1 = counter1 + 1; %>
<% counter2 = counter2 + 1; %>
<p>Counter 1 is <%= counter1 %></p>
<p>Counter 2 is <%= counter2 %></p>
```

如果你安装了页面，并请求若干次，那么 `counter1` 的值在每次请求中都是 1。而 `counter2` 的值在请求后会增加，知道 Tomcat 的重启。

除了你自己声明的变量外，JSP 页面还能访问一定数量的对象，这些对象都是它隐式为你声明的，这将在后续的“隐式 JSP 对象”一节中讨论。

JSP指示语

`<%@` 和 `%>` 标记表明一个 JSP 指示语，它给 JSP 处理器提供了关于页面输出的信息，类，还有标签库等。

```
<%@ page ... %>
```

`page` 指示语提供了几种类型的信息，这是以一到多个 `attribute = "value"` 这样的形式提供的。下面的指示语表明页面的脚本语言是 Java，且它的输出是 `text/html` 类型：

```
<%@ page language="java" contentType="text/html" %>
```

这种特殊的指示语根本无须声明，因为 Java 和 text/html 是默认值。如果一个 JSP 页面产生非 HTML 的输出，请确保重载默认的内容类型，例如，对于一个产生无格式文本的页面，像这样：

```
<%@ page contentType="text/plain" %>
```

一个 import 属性能让 Java 类被导入。在一个规范的 Java 程序中，你会使用到 import 语句。而在 JSP 页面中，你需要这样：

```
<%@ page import="java.util.Date" %>
<p>The date is <%= new Date () %>.</p>
```

如果你仅使用某个特定的类一次，那么更方便的方式可能是去掉指示语，而直接使用该类的完整名称：

```
<p>The date is <%= new java.util.Date () %>.</p>
```

```
<%@ include ... %>
```

include 指示语把一个文件的内容直接插入页面翻译过程中。这就是说，该指示语被替换为文件的内容，之后这些内容就被翻译。下面的指示语导致了一个名为 my-setup-stuff.inc 文件被导入：

```
<%@ include file="/WEB-INF/my-setup-stuff.inc" %>
```

一个以 / 开头的指示语，表明了相对于应用目录的一个文件名。没有 / 开头则意味着这个文件相对于页面的位置。

包含文件，能够轻松地在一组 JSP 页面内共享内容（静态的或者动态的）。例如，你能使用它们来为一组 JSP 页面提供标准的 header 或 footer，或者为共同的操作执行些代码，比如建立通往某个数据库服务器的连接。

```
<%@ taglib ... %>
```

一个 taglib 指示语表示该页面使用了一个给定的 tag 库中的定制行为。该指示语包含属性以告诉 JSP 引擎如何去定位库 TLD 文件，以及你会在页面的后续部分使用的名字，从而把标签从库中提取出来。例如，一个页面使用了来自 JSTL 的 core 和 database-access 标签：

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
```

这里，URI (Uniform Resource Identifier) 属性唯一地指定了标签库，这样 JSP 引擎就能找到它的 TLD 文件。TLD 定义了行为的接口，这样 JSP 处理器就能确保该页面正确地使用了库的标签。构建唯一性 URI 的一个共同规范是，使用一个字符串，该串包含标签库的

主机。这让 `uri` 看起来很像 URL，但它其实只是一个标记符。并不意味着 JSP 引擎真的需要到那个主机上去获取描述文件。这些与 `uri` 相关的具体原则，在后续的“使用标签库”一节会有详细介绍。

前缀属性表明库中的标签是如何被调用的。这些指示语仅仅暗示着 `core` 和 `database` 标签会以 `<c: xxx>` 和 `<sql: xxx>` 的形式出现。例如，你可以如下使用来自 `core` 库中的 `out` 标签：

```
<c:out value="Hello, world." />
```

或者你也许会使用 `database` 的 `query` 标签，发出一个查询：

```
<sql:query dataSource="${conn}" var="result">
    SELECT id, name FROM profile ORDER BY id
</sql:query>
```

行动元素

行动元素标签，能引用标准的 JSP 行为，或者在标签库中定制的行为。标签的名字包含一个前缀和一个特定的行动：

- 标签名字带一个 `jsp` 前缀表明预定义的行动元素。例如 `<jsp:forward>` 把当前的请求推送给另一个页面。这个行动在一个标准的 JSP 处理器下，可适用于任一页面。
- 定制行动由标签库执行。标签名字的前缀必须匹配一个 `taglib` 指示语的 `prefix` 属性，这样 JSP 才能决定该标签属于那个库。要使用定制标签，必须先安装库文件。请查看下一节“使用标签库”以获取更多细节。

在一个 JSP 页面中，行动是写为 XML 元素的，它们的语法也遵循 XML 原则。一个元素和它的主体都放入开始和关闭标签中：

```
<c:if test="${x == 0}">
    x is zero
</c:if>
```

如果该标签没有主题，那么可以把开始和关闭标签合并在一起：

```
<jsp:forward page="some_other_page.jsp" />
```

使用标签库

假设你有一个标签库，它含有一个 JAR 文件 `mytags.jar` 和一个标签库描述文件 `mytags.tld`。要想让一个给定应用的 JSP 页面能使用这个库，需要安装这两个文件。典型的，你把 JAR 文件放在应用的 `WEB-INF/lib` 目录下，且把 TLD 文件放在 `WEB-INF` 目录下。

一个 JSP 页面使用的标签库，在使用其提供的任何行动之前，必须包含一个合适的 `taglib` 指示语：

```
<%@ taglib uri="tld-location" prefix="taglib-identifier" %>
```

这里，prefix 属性告诉 Tomcat 你希望在本 JSP 页面的后续部分，如何引用库中的标签。如果你使用 mytags 的前缀，你就能这样引用标签：

```
<mytags:sometag attr1="attribute value 1" attr2="attribute value 2">  
tag body  
</mytags:sometag>
```

prefix 值是你自己选择的一个名字，但是你必须在整个页面中始终如一地使用这个名字，你不能为两个不同的标签库选择相同的名字。

uri 属性则告诉 JSP 处理器到哪里去寻找标签库的对应 TLD 文件。这个值可以是直接的，也可以是间接的：

- 你可以直接把 uri 设置为 TLD 文件的路径：

```
<%@ taglib uri="/WEB-INF/mytags.tld" prefix="mytags" %>
```

以 / 字符开头表示该文件名是相对于应用目录的。没有 / 字符的话，就意味着文件是相对于包含 taglib 指示语文件的目录的。

如果一个应用使用了大量的标签库，管理TLD文件使其不至于拥挤WEB-INF目录的一个共同方法是，把它们放入WEB-INF下的tld子目录中。在这种情况下，uri就是：

```
<%@ taglib uri="/WEB-INF/tld/mytags.tld" prefix="mytags" %>
```

直接设置一个 TLD 文件路径有个缺点。如果一个新版本的标签库发布了，而 TLD 文件有了另一个名字，那么你需要修改所有 JSP 页面中对应的 taglib 指示语。

- 另外一种设置 TLD 文件位置的方法，是使用 JAR 文件的路径名。

```
<%@ taglib uri="/WEB-INF/lib/mytags.jar" prefix="mytags" %>
```

JSP 处理器能这样找到 TLD 文件，前提是 JAR 文件的 META-INF/taglib.tld 位置。然而，这种方法同样也有些小问题。如果一个新版本的库发布了，且其中一个 JAR 文件的路径有所改变，那么你必须修改每个 JSP 页面中对应的 taglib 指示语。而且这种方法，对那些不能在 JAR 文件中查找 TLD 的容器，是不适用的。

- 第三种方法，就是间接地指出 TLD 文件的位置。给库一个符号化的名字，同时把一个 <taglib> 条目加入应用的 web.xml 文件中，该条目把符号名字映射为 TLD 文件的位置。然后引用你 JSP 页面中的符号名字。假设你为 mytags 库定义了符号名字为：

<http://terrific-tags.com/mytags>

那么，在 web.xml 中的 <taglib> 条目列出了符号名字，并给对应的 TLD 文件提供路径。如果该文件安装在 WEB-INF 目录下，请如下编写条目：

```
<taglib>
<taglib-uri>http://terrific-tags.com/mytags</taglib-uri>
<taglib-location>/WEB-INF/mytags.tld</taglib-location>
</taglib>
```

如果该文件被安装在 WEB-INF/tld 下，那么请这样写：

```
<taglib>
<taglib-uri>http://terrific-tags.com/mytags</taglib-uri>
<taglib-location>/WEB-INF/tld/mytags.tld</taglib-location>
</taglib>
```

上述两种写法，你是在使用符号名字来引用 JSP 页面中的标签库：

```
<%@ taglib uri="http://terrific-tags.com/mytags" prefix="mytags" %>
```

使用一个符号 TLD 名字，涉及到一个间接层次，但它有一个显著的优点。那就是它通过引用 JSP 页面中的标签库，提供了一个更加稳定的方法。你在 web.xml 中设置 TLD 文件的实际位置，而不是在每个 JSP 页面中。这样，如果一个新版本的库文件发布了，且 TLD 文件有一个不同的名字，那么你只须修改 web.xml 中的 <taglib-location> 值，然后重新启动 Tomcat 就行。没有必要更改任何使用该标签的 JSP 页面。

隐式 JSP 对象

当一个 Servlet 运行时，Servlet 容器传给它两个参数：请求和回应。但你必须声明其他的对象。例如，你可以使用回应参数来获取一个输出对象：

```
PrintWriter out = response.getWriter();
```

与 Servlet 编写不同的是，JSP 提供了便捷的隐式对象。也就是说，标准的对象是作为 JSP 执行环境的一部分提供的。你可以引用其中的任意对象而不需要显式地声明它们。这样，在 JSP 页面中，out 对象可以被当作已经建立并可以使用来对待。其他有用的隐式对象为：

pageContext

一个为页面提供了环境的对象。

request

一个包含了从客户端处接收来的请求信息的对象，信息包括在一个表单中的参数。

`response`

回应被构建，并发送回客户端。例如，你可以使用它来设置回应的头。

`out`

输出对象。通过 `print()` 或 `println()` 这样的方法来往对象中写入数据，可以往回应对象中添加文本。

`session`

Tomcat 提供了访问 `session` 的方法，`session` 可以被用来存放从请求到回应的各种信息。这允许你编写可以和用户交互的应用，这个应用看起来是一系列的事件。请查看第20章以了解更多与 `session` 相关的信息。

`application`

这个对象让你能够访问应用级别共享的信息。

JSP页面的作用域层次

JSP 页面能访问几个作用域层次，这可以被用来存储各种信息。作用域层次为：

Page scope

信息仅仅在当前JSP页面有效。

Request scope

在处理当前客户端请求时，信息对任意 JSP 页面或者 Servlet 都有效。很可能一个页面在处理请求时调用了另外一个页面。把信息放在请求作用域中，允许这些页面彼此通信。

Session scope

在处理一个请求，而且该请求属于某个给定的 `session` 时，信息可用于任意页面。对于一个给定的客户端而言，`session` 作用域是跨多个请求的。

Application scope

信息对于在应用上下文中的任意页面都可用。应用作用域可以跨多个请求，`session` 或客户端。

一个上下文并不知道其他上下文，但在同一个上下文中的页面能通过注册的属性（对象）共享信息，这种共享的作用域高于页面作用域。

把信息移入或移出一个给定的作用域，请使用该域对应的隐式对象的 `setAttribute()` 或者 `getAttribute()` 方法。例如，要把一个字符串 `tomcat.example.com` 放入请求作用域，且命名为 `myhost` 属性，请使用请求对象：

```
request.setAttribute ("myhost", "tomcat.example.com");
```

`setAttribute()`把值当作对象来存储。要在后续过程中获取这个值，请使用`getAttribute()`和它的名字：

```
Object obj;
String host;
obj = request.getAttribute ("myhost");
host = obj.toString ();
```

当和`pageContext`对象一起使用时，`setAttribute()`和`getAttribute()`是默认为页面作用域的。此外，它们可以用一个额外的参数`PAGE_SCOPE`、`REQUEST_SCOPE`、`SESSION_SCOPE`或`APPLICATION_SCOPE`来调用以显式地设置作用域层次。下面的语句，有和前面语句同样的效果：

```
pageContext.setAttribute ("myhost", "tomcat.example.com",
                         pageContext.REQUEST_SCOPE);
obj = pageContext.getAttribute ("myhost", pageContext.REQUEST_SCOPE);
host = obj.toString ();
```

参考资料

References

本附录列举了一些对想进一步了解本书所讨论的内容的读者有帮助的信息。

MySQL资源

MySQL Resources

主要的 MySQL 网站是 <http://www.mysql.com/>.

参考书目

MySQL Administrator's Guide and Language Reference. MySQL AB. MySQL Press: 2006. ISBN: 0-672-32870-4

这是以MySQL Reference Manual方式打印的一本手册，也可在线获得：<http://dev.mysql.com/doc/>.

MySQL. Paul DuBois. Sams Developers Library: 2005. ISBN: 0-672-32673-6

这本书提供了关于 MySQL 使用和管理的一般信息。它使用了 C、Perl、PHP API 相关的章节介绍了 MySQL 编程。

Perl资源

Perl Resources

主要的MySQL网站是 <http://www.mysql.com/>.

额外的 DBI 和 CGI.pm 文档可通过命令行获得：

```
% perldoc DBI  
% perldoc DBI::FAQ  
% perldoc DBD::mysql  
% perldoc CGI.pm
```

或者从在线的：

<http://dbi.perl.org/>
<http://search.cpan.org/dist/CGI.pm/>

获取，以下是有关图书：

参考书目

Programming Perl. Larry Wall, Tom Christiansen, and Jon Orwant. O'Reilly Media: 2000. ISBN: 0-596-00027-8

Perl Cookbook. Tom Christiansen, and Nathan Torkington. O'Reilly Media: 2003. ISBN: 0-596-00313-7

Programming the Perl DBI. Alligator Descartes, and Tim Bunce. O'Reilly Media: 2000. ISBN: 1-565-92699-4

用整本书来介绍 DBI，作者之一 Bunce 是 DBI 的主要架构师。

Official Guide to Programming with CGI.pm. Lincoln Stein. Wiley Computer Publishing: 1998. ISBN: 0-471-24744-8

Lincoln Stein 是 CGI.pm 的作者，他的书是模块编程的标准参数。

MySQL and Perl for the Web. Paul DuBois. New Riders: 2001. ISBN: 0-735-71054-6

这本书大量使用了 CGI.pm，同时还结合 DBI 模块来展示如何把 MySQL 集成到基于 Web 的应用中。

Ruby资源

Ruby Resources

主要的 Ruby 网站是 <http://www.ruby-lang.org/>，在这里你可以下载 Ruby 发布，并找到文档的许多链接。

参考书目

Programming Ruby: The Pragmatic Programmer's Guide. Dave Thomas, Chad Fowler, and Andy Hunt. Pragmatic Press: 2004. ISBN: 0-974-51405-5

Ruby Cookbook. Lucas Carlson, and Leonard Richardson. O'Reilly Media: 2006. ISBN: 0-596-52369-6

PHP资源

PHP Resources

主要的 PHP 网站是 <http://www.php.net/>，它提供了 PHP 发布和文档。提供 PEAR、PHP 扩展和插件库的网站是 <http://pear.php.net/>，PEAR 包含了 DB 数据库抽象模块。

参考书目

Learning PHP 5. David Sklar. O'Reilly Media: 2004. ISBN: 0-596-00560-1

PHP Cookbook. David Sklar , and Adam Trachtenberg. O'Reilly Media: 2002. ISBN: 0-596-10101-5

Python资源

Python Resources

主要的 Python 网站是 <http://www.python.org/>, 它提供了 Python 发布版本和文档.

DB-API 数据库访问接口的一般信息, 可在 <http://www.python.org/topics/database/>上找到.
MySQLdb,一个MySQL 相关的DB-API驱动, 在<http://sourceforge.net/projects/mysql-python/>.

The Vaults of Parnassus 为 Python 提供了一个知识库: <http://www.vex.net/~x/parnassus/>.

参考书目

Python Essential Reference. David M. Beazley. Sams: 2006. ISBN: 0-672-32862-3

Python Cookbook. Alex Martelli, Anna Ravenscroft, and David Ascher. O'Reilly Media: 2005.
ISBN: 0-596-00797-3

Java资源

Java Resources

主要的 Java 网站是 <http://java.sun.com/>. Sun 的 Java 网站为 JDBC、Servlet、JSP 和 JSP 标准标签库提供了文档:

- JDBC 一般信息: <http://java.sun.com/javase/technologies/database.jsp>
- JDBC 文档: <http://java.sun.com/j2se/1.5.0/docs/guide/jdbc/>
- Java Servlet: <http://java.sun.com/products/servlet/>
- JavaServer Pages: <http://java.sun.com/products/jsp/>
- JSP 标准标签库: <http://java.sun.com/products/jsp/jstl/>

参考书目

Database Programming with JDBC and Java. George Reese. O'Reilly Media: 2000. ISBN: 1-565-92616-1

JavaServer Pages. Hans Bergsten. O'Reilly Media: 2003. ISBN: 0-596-00563-6

JSTL in Action. Shawn Bayern. Manning Publications: 2002. ISBN: 1-930-11052-9

其他资源

Other Resources

这些资源提供了本书中其他主题的信息：

参考书目

HTML & XHTML: The Definitive Guide. Chuck Musciano, and Bill Kennedy. O'Reilly Media: 2006. ISBN: 0-596-52732-2

Mastering Regular Expressions. Jeffrey E. F. Friedl. O'Reilly Media: 2006. ISBN: 0-596-52812-4