

# 第一部分 MySQL 的使用

## 第1章 MySQL 与 SQL 介绍

本章介绍 MySQL 关系数据库管理系统 (RDBMS) 及其所采用的结构化查询语言 (SQL)。文中给出了应该掌握的基本术语和概念, 并介绍了本书中使用的样例数据库, 提供了怎样利用 MySQL 创建数据库并对其进行存取访问的指导。

在此, 如果您对数据库不熟悉, 可能还不能肯定是否需要一个数据库或是否能够使用一个数据库。或者, 如果您对 MySQL 或 SQL 一无所知, 需要一种入门性的指导, 那么应该仔细阅读本章。对 MySQL 或对数据库系统具有一定经验的读者可能希望跳过这一章。但是, 不管是否是初学者都应该阅读 1.2 节“一个样例数据库”, 因为这一节中给出的数据库是熟悉数据库的用途和内容的一个最好的样例, 本书将要反复地用到它。

### 1.1 MySQL 的用途

本节介绍 MySQL 的应用场合。提供 MySQL 能够做什么, 以何种方式做的一个大致概念。如果您不需要了解数据库的用途, 或许您已经在头脑中有了要解决什么问题的想法, 只是希望知道怎样用 MySQL 来帮助解决它, 那么可以跳到 1.2 节“一个样例数据库”。

数据库系统本质上是一种用来管理信息列表的手段。这些信息可来自不同的地方。例如, 它可以代表研究数据、业务记录、顾客请求、运动数据统计、销售报告、个人爱好信息、人事记录、问题报告或学生成绩等。虽然数据库系统能够处理广泛的信息, 但您不会仅仅是为用它而用它。如果一项工作很容易, 那么就没有理由非得仅为了使用数据库而将数据库引入这项工作。杂货单就是一个很好的例子: 开列一个购物清单, 购买后在上面画叉, 然后将它扔了, 极不可能为此事使用一个数据库。即使您有一台便携式电脑, 也只会为杂货单使用记事本, 而不会启用数据库。

数据库系统的力量只在组织和管理的信息很庞大或很复杂, 用手工处理极为繁重时才能显示出来。当然, 每天处理数百万个业务的大公司可以使用数据库。但是, 即使只涉及个人爱好的单一人员维护信息的小公司也可能需要数据库。不难想像由于在信息变得难于管理之前, 使用了数据库而带来的好处。考虑下列情形:

您的木工店有几个员工。需要保存员工和工资记录, 以便知道给谁付过工资, 什么时候付的, 并且必须对这些记录进行汇总以便能向税务部门报收益表。还需要明了您的公司雇人所做的工作以及对每项工作所做的安排。

您有一个汽车零部件的库房网, 需要知道哪些库房中有给定的零件, 以便能填写顾客订单。

作为玩具销售商, 要特别关注所进货物是否流行。需要知道某项物品的当前销售曲线, 以便能够估计是否需要增加库存量 (对越来越流行的物品), 或减少其库存量 (从而用

不着存放一大堆销售不好的东西)。

多年课题研究收集的大量研究数据需要进行分析以便发表。希望对大量的原始数据进行加工,得出结论性的信息,并为更详细的统计分析筛选出观察样本子集。

您是位受欢迎的演讲者,到全国各地的各种集会上进行演讲,如在毕业典礼、商务会议、城市集会和行政大会上讲演。作了这么多讲演,自己很难记住在什么地方讲了些什么,因此一定很愿意保存过去讲演的记录,以帮助准备以后的演说。如果您回到了一个以前曾作过演说的地方,肯定不愿意作一个与上一次类似的演讲,到过的地方都有一个记录能帮助您避免重复。您必定也愿意注意讲演受欢迎的程度。(您在“大都会狗窝”俱乐部所做的演讲“我为什么喜欢猫”不太成功,那么下次去那儿时一定不希望再犯同样的错误。)

您是个教师,需要知道学分和出勤情况。每当您进行测验或考试时,都要记录学生的学分。将考试成绩写在学分簿上很容易,但以后利用这个学分簿却很费事。因此,在学期末确定最终成绩时,您宁可不进行学分排序,而且宁可不汇总每个学生的学分。要统计出每个学生的缺旷课情况也不是一件简单的事。

您是某机构的秘书,这个机构有一个庞大的会员姓名地址簿。(所谓机构可以是任何组织,如一个专业团体、俱乐部、交响乐团或球迷俱乐部等。)您每年都要根据会员信息变化,用字处理器进行编辑,然后为每个会员们打印一个地址名录。

您厌倦了以这种方式维护这个地址簿,因为它限止了您利用它可做的事。用它难于以不同的方式对各条目排序,不能方便地选择每个条目的特定部分(如给出仅由姓名和电话号码组成的清单)。也不能查出某组会员,如那些不久就需要更新其会员资格的人员,如果可能的话,应该取消为了找到哪些需要发送补充说明的会员而每个月都要查找所有条目的工作。

而且,您一定不愿意自己做地址簿的编辑工作,但是团体没有那么多的预算,请人会产生问题。您听说过“无纸化办公”,这是一种导致电子化保存记录的方法,但您没有看到任何好处。现在会员记录是电子化的,但具有讽刺意义的是,除了地址簿的打印外,没省多少事。

上述情形中有的涉及信息量较大,有的涉及信息量较小。它们的共同特征都是所涉及的任务可由手工完成,但是用数据库系统来做会有效得多。

使用如像 MySQL 这样的数据库系统希望看到什么样的效果呢?这有赖于您的特定需求,正如上面的例中所看到的那样,其效果的差异是相当大的。我们来考虑一种常见的情形,从而也是一种相当有代表性的数据库应用。

通常利用数据库管理系统来处理诸如人们用文件柜来完成的那样一类的任务。确实在某种意义上说,数据库就像一个大文件柜,只不过是一个内建的文件编排系统而已。电子化处理记录相对手工处理记录有很多优点。例如,如果您在某种保存有客户记录的办公设施内工作,那么 MySQL 可在某些方面向您提供帮助:

减少记录编档时间。不必为寻找增加新记录的位置而查看橱柜的所有抽屉。只要将记录放入文件编排系统,并令文件编排系统为您将该记录放入正确的位置即可。

减少记录检索时间。在查找记录时,不需要自己去寻看每个记录以找到含有所需信息的那个记录。假如您在一个牙科诊所中工作。如果想给所有近来未到诊断做过检查的病人发催询单,只要求文件编排系统查找合适的记录即可。当然,这样做会有别于吩咐别人去做。吩咐别人去做,您只需说,“请确定哪些病人最近 6 个月内没来过。”

而使用数据库，则需要发出一串奇怪的“咒语”：

```
SELECT last_name, first_name, last_visit FROM patient  
WHERE last_visit < DATE_SUB(CURRENT_DATE, INTERVAL 6 MONTH)
```

如果您从来没有看到过类似的东西，可能会感到相当吓人，但是在一两秒内就能得到结果远胜于用一个小时来查找，这应该是很具有吸引力的。（不管怎样，不用太发愁。这些“咒语”用不了多久就会不奇怪了。事实上，只要您读完本章就能完全理解其含义。）

灵活的查找序列。不需要按记录存放的固定序列去查看它们（例如，按姓查找）。可以要求文件编排系统以任意的序列查出记录；如按姓、保险公司名、最后光临日期等提出记录。

灵活的输出格式。在查找到感兴趣的记录后，不需要手工拷贝其信息。可以让文件编排系统为您生成一份清单。有时，您可能只需要打印这些信息。有时，您又可能希望在其他程序中使用这些信息。（如，在生成误了看牙预约的病人清单后，可将这些信息送入一个字处理器，打出送给这些病人的通知单。）或者您只对汇总信息感兴趣，如对所选出记录数感兴趣。不必自己数它们；文件编排系统可自动生成汇总。

多个用户同时访问记录。对纸上的记录，如果两个人想同时查找一个记录，那么其中一个人必须等另一个人找完才能查找。MySQL 提供多个用户同时查找的能力，从而两个人可同时访问记录。

记录的远程访问与电子传输。纸面记录需要有该记录在手边才能使用，或者需要有人做拷贝再发送给您。而电子记录可以远程访问或进行电子化传输。如果您的牙医专家在多个诊所工作，那么他们可从自己的所在地访问您的记录，不需要给他们发快信。如果需要记录的某个人没有与您的数据库软件相同的软件，但有电子邮件，那么您可以选择所需的记录，用电子文档发送。

如果您以前使用过数据库管理系统，已经了解数据库的上述诸般好处，可能会想，怎样才能超越“取代文件柜”的用途。现在，数据库系统已经可以用来提供过去不能，直到最近才能够提供的服务。例如，许多机构以一种与 Web 结合的方式使用数据库，这种方式过去是做不到的。

假如您的公司有一个库存数据库，在顾客询问库房中是否有某项物品，它的价格是多少时，服务台人员使用这个数据库，这是数据库的一种较为传统的应用。但是，如果您的公司向顾客提供一个可供访问的 Web 站点，那么可以提供另一项服务，即：提供一个允许顾客确定物品价格和可得性的搜索页。这给顾客提供了他们所需的信息，提供的方法是让顾客自动地搜索存放在库存中的物品信息。顾客可以立即得到信息，不用听预先录好的音，或受服务台是否正在工作的限制。对于每个使用您的 Web 站点的顾客，所花的费用比服务台工作人员转接电话的费用还少。（或许，该 Web 站点已为这个付了费。）

还有比上述更好的利用数据库的方法。基于 Web 的库存查询请求可以不仅仅为顾客提供信息，而且还可以为您自己提供信息。该查询请求告诉您顾客在找什么，而查询的结果又可以让您知道能否满足他们的请求。您可能在不能满足顾客需求的方面丧失商机。因此，记录有关库存搜索的信息是很有意义的，如记录：顾客在找什么、库存有没有。然后，可以利用这些信息调整您的库存，更好地为顾客提供服务。

数据库的另一新用途是在 Web 页上做标题广告。我也和您一样不喜欢它们，但事实这是一种很流行的 MySQL 应用，可用 MySQL 来存储广告，然后检索它们为 Web 服务器的显

示而用。此外，MySQL 还可以用来进行跟踪，这种跟踪涉及哪些广告起了作用、它们被显示了多少次、哪个站点访问了它们等等信息。

因此，知道如何利用 MySQL 最好的办法是自己试试，为此目的您应该有一个试验性的数据库。

## 1.2 一个样例数据库

本节介绍一个样例数据库，这个数据库在本书各个部分都可能用到。在学习将 MySQL 投入工作时，这个数据库为您提供了参考的例子。我们主要从前面描述过的两种情形来给出例子：

机构的秘书方案。我们需要一些比“机构”更为明确的信息，所以现在就来构造一个，它具有这样一些特性：它由为了研究美国历史这个共同目的而聚集在一起的一群人组成（一时找不到更好的名称，就暂且称为美国历史同盟）。在交会费的基础上定期更新各会员的资格。会费构成了此同盟的活动经费，如出版报纸“美国编年历”。此联盟也有一个小 Web 站点，但开发出的功能不多。迄今这个站点只限于提供一些基本的信息，如本团体的性质，负责人是谁，什么样的人可以参加等。

学分保持方案。在学分时段中，需要管理被测试者、记录得分并赋予得分等级。然后确定最后的得分等级，将其与出勤率一道交给学校办公室。

现在让我们根据如下两个要求来进一步考虑这些情况：

必须确定希望从数据库中得到什么信息，即，希望达到什么目的。

必须计划好要向数据库输入什么，即将要保存什么数据。

或许，在考虑向数据库输入什么数据以前，逆向考虑一下需要从数据库输出什么数据。在能够对数据进行检索前，必须将数据送入数据库。但是，使用数据库的方法是受您的目标驱动的，这些方法与希望从数据库取出何种信息的关系较之与向数据库输入何种信息的关系更为紧密。除非打算以后使用这些信息，否则肯定不会浪费时间和精力将它们输入数据库。

### 1.2.1 美国历史同盟

这个方案的初期状况是您作为同盟的秘书，利用字处理文档维护会员清单。这样就生成一个打印的姓名地址录来说还是可以应付的，但是在利用这些信息做别的事时就会受到限制。假定您打算做下列工作：

希望能够利用该姓名地址录产生不同格式的输出，并且只给出相应用途所需的信息。目标之一是生成每年的打印姓名地址录，这是该同盟过去就需要的，您打算继续打印。除此之外，可以设想将姓名地址录中的信息派一些别的用途，如在全盟的年度宴会上所提供的节目单中给出一个当前的会员清单。这个应用涉及不同的信息集合。打印的姓名地址录中使用了每个会员条目的所有内容。而对于宴会节目单，只需要取出会员名字即可（如果采用字处理器要做到这一点有时是不太容易的）。

希望搜索姓名地址录查找其条目满足某些条件的会员。例如，希望知道哪些会员不久就需要更新其会员资格。另外涉及搜索的应用是由于需要维护每个会员的关键字列表而产生的。这些关键字描述了每个会员特别感兴趣的美国历史的某个方面（如内战、经济萧条、公民权利或托马斯·杰佛逊的生活等）。会员们有时会向您要一份与他们自



己有类似爱好的会员的清单，您一定乐于满足他们的这种要求。

希望让姓名地址名录在联盟的 Web 站点上联机使用。这对会员和您都是很有好处的。如果您能够将姓名地址录用某种合适的自动过程转换为 Web 页，则这个姓名地址录的联机版就可以一种比打印版更及时的方式保持最新信息。而且如果能使这个联机姓名地址录可供搜索，那么会员就能够自己方便地查找信息了。例如，某个会员希望知道其他对内战感兴趣的会员，他就可以自己将这些会员找出而不用请您帮他查找，而您也不用花时间去做了。

我们清楚地知道，数据库并不是世界上最令人激动的东西，因此，我们也不打算狂热地声称，使用数据库可以促进创造性的思维。但是，当您停止将信息视为某种必须与之搏斗的东西（在用字处理文档时确实是这样的），并开始将其想像为某种可以相对容易地操纵的事物（正如希望用 MySQL 所做到的那样）时，您提出某种使用或表示信息的新方法的能力将会得到某种程度的解放，例如下面这些例子就是一些新方法：

如果数据库中的信息能够以联机姓名地址录的形式移到 Web 站点中，那么您可能会让信息以其他的方式流动。例如，如果会员能够联机编辑自己的条目，对数据库进行更新，那么您就不必自己做所有的编辑工作了，这样有助于使姓名地址录中的信息更为准确。

如果您在数据库中存储 Email 地址，那么可以利用它们来发送 Email 给那些相当长的一段时间没有更新自己的条目的会员。发出的消息可以向这些会员显示他们的条目内容，请他们查看，然后指示怎样利用 Web 站点提供的实用工具做所需的修改。

数据库不仅以关联到会员表的方式帮助使 Web 站点更为有用。比方说，联盟出版了一份报纸“美国编年史”，每一期中都有一个给小孩子的版面，内含历史试题。最近有几期主要集中在美国总统的传记上。联盟的 Web 站点也可以包含给孩子的版面，这样使试题联机。通过放置从数据库中取出的试题并让 Web 服务器对随机给出的问题进行查询，或许甚至可以使这个版面成为交互式的。

至此，您可能已经想起了许多数据库的用途，这使您有点不能自控了。在回到现实之前，您开始问一些特殊的问题：

这是不是有点野心勃勃了？在准备时是不是要做大量的工作？当然，如果只是想而不去做，则任何事情都很简单，我并不伪称上述所有事情实现起来都是微不足道的。然而，在本书结束时，我们所描述的这些事都实现了。只需记住一件事，没必要一次做完所有的事。我们将对工作进行分解，每次只做一部分。

MySQL 能够完成所有这些事吗？不，它不能够。例如，MySQL 没有直接的 Web 能力。虽然由 MySQL 自身不能完成我们所讨论的每样事情，但是可以得到与 MySQL 一起工作的工具，从而完善和扩展了 MySQL 的能力。

我们将用 Perl 脚本语言和 DBI（数据库接口）Perl 模块来编写访问 MySQL 数据库的脚本。Perl 具有极为出色的文本处理能力，它允许以一种高度灵活的方式处理查询结果以产生各种格式的输出。例如，我们可以用 Perl 来生成多信息文本格式（RTF）的姓名地址录，这是一种可被所有字处理器读取的格式。

我们也可以使用另一种脚本语言 PHP。PHP 特别适合于编写 Web 应用，而且它与数据库一起工作。这使得能从 Web 页运行 MySQL 查询并生成包含数据库查询结果的新页。PHP 与

Apache（世界上最流行的 Web 服务器）一起工作得很好，这使得完成诸如给出一个搜索窗口并显示搜索结果之类的事情很容易。

MySQL 与这些工具集成得很好，并向您提供了以自己的方式组合它们的灵活性，可以进行选择以实现您的设想。不用受限于那些大肆推销的所谓“集成”功能而实际工作起来也只是彼此之间的固定组合。

最后，有一个大问题，那就是所有这些东西要花多少钱？首先，同盟的预算是有限的。回答是，大概什么钱也不用花，这可能会令您吃惊。如果您熟悉一般的数据库系统，就会知道，它们一般相当昂贵。但是，MySQL 一般是免费的。在某些环境下，确实不需要许可证，而且如果用户数量不限也只需花 \$200。（关于许可证的一般介绍请参阅前言，特定的细节可参阅 MySQL 参考指南。）我们将使用的其他工具（Perl、DBI、PHP、Apache）也是免费的，因此，所有东西都考虑到了，可以相当便宜地组成一个有用的系统。

开发这个数据库的操作系统的选择取决于您。我们介绍的所有软件都可运行在 UNIX 下，其中大多数可以运行在 Windows 下。作者推荐在 UNIX 下运行 MySQL 和其他工具。它们全都是发源于 UNIX 下，然后才转到 Windows 的。这表示它们的 Windows 版本成熟期较短，尚未经过彻底的测试和使用。

现在，让我们来考虑一下使用样例数据库的其他情形。

## 1.2.2 学分保存方案

初步的想法是，作为一个老师，有保存学分的职责。老师希望将学分处理从学分簿上的手工操作转到 MySQL 上用电子表示。在此情形下，想从数据库得到的是含在学分簿中的东西：

对于每次测验或测试，要记录学分。对测试，将学分排序，以便能确定每个字符（A、B、C、D 和 F）所代表等级的得分范围。

在学分时段结束时，计算每个学生的总得分，然后排序总的得分并根据它们确定得分等级。总的得分可能涉及权重计算，因为大概会希望使测试的得分比测验和得分权重更大。

在每个学分时段结束时，提供出勤信息给学校办公室。

目的是避免手工排序和汇总学分及出勤率记录。换句话说，希望 MySQL 在学分时段结束时对学分排序并完成每个学生的总分和缺课数的计算。为了达到这个目的，需要班级中的学生名册、每次测验和测试的分数以及学生缺课的日子。

## 1.2.3 样例数据库怎样才能满足需求

如果您对历史同盟或学分保存不太感兴趣，可能会奇怪为什么必须做这些例子呢？答案是这些样例方案本身并不是目的，只是用它们说明利用 MySQL 及其相关的工具能做什么事。

加上一点想像，您将会看到样例数据库的查询怎样应用到所希望解决的问题上。假设您在前面提到的牙科诊所上班，将会在本书中看到许多牙科方面的查询。例如，确定历史同盟的哪些会员需要立即更新他们的会员资格，这是一件类似于确定哪些病人近来没有来看牙医的事情。两者都是基于日期的查询，因此，一旦学会了编写会员更新的查询，便可以将该技

术用来编写更为感兴趣的延误的预约病人查询。

### 1.3 基本数据库术语

您可能会注意到，已经读了本书这么多页，但是还没有看到几句行话和术语。虽然我们大致提了一下怎样利用样例数据库，但事实上，关于什么是“数据库”，我们一点东西都还没有介绍。不过，我们现在打算设计该数据库，然后开始实现它，这样就不能再避而不谈数据库术语了。介绍数据库术语就是本节的目的。本节介绍的一些术语全书都要用到，因此必须对其熟悉。所幸的是，关系数据库中的许多概念是相当简单的。事实上，关系数据库的吸引力主要来源于其基本概念的简单性。

#### 1.3.1 基本术语

在数据库世界中，MySQL 归类为关系数据库管理系统（RDBMS）。所谓关系数据库管理系统的含义如下：

数据库（RDBMS 中的“DB”）是存储信息的仓库，以一种简单的、规则的方式进行组织：

数据库中的数据集组织为表。

每个表由行和列组成。

表中每行为一个记录。

记录可包含几段信息；表中每一列对应这些信息中的一段。

管理系统（“MS”）是允许通过插入、检索、修改或删除记录来使用数据的软件。

“关系”（“R”）一词表示一种特殊种类的 DBMS，它通过寻找相互之间的共同元素使存放在一个表中的信息关联到存放在另一个表中的信息。关系数据库的能力在于它能够从这些表中方便地取出数据，并将关联各表中的信息相结合得出问题的答案，这些答案只依据单个表的信息是不可能得到的。

这里有一个例子，示出了关系数据库怎样将数据组织成表并将一个表中的信息与另一个表中的信息相关联。假定您管理一个含有标题广告服务的 Web 站点。您与公司有协议，这些公司希望有人在拜访您的站点上的网页时显示他们的广告。每当一个拜访者点击您的页面一次，您就向该拜访者的浏览器提供了嵌在页面中的广告的一次服务，并且给公司估算一点费用。为了表示这些信息，要保存三个表（请参阅图 1-1）。一个是 company 表，它含有公司名、编号、地址和电话号码等列。另一个是 ad 表，它列出广告编号、拥有该广告的公司的编号以及每次点击时的计费数。第三个 hit 表按广告编号记录广告点击次数以及广告提供服务的日期。

利用单个表的信息可以回答某些问题。为了确定签协议的公司数目，只需对 company 表中的行数计数即可。类似地，为了确定某个给定时间段中的点击次数，只需查看 hit 表即可。其他问题要更为复杂一些，而且必须考虑多个表以确定答案。例如，为了确定 Pickles 公司的每个广告在7月14日点击了多少次，应该按如下步骤使用这些表：

1) 查询 company 表中的公司名（Pickles, Inc）以找到公司编号（14）。

2) 利用公司编号查找 ad 表中匹配的记录以便能够确定相关的广告编号。有两个这样的广告，48 和 101。

3) 对 ad 表中匹配的记录，利用该记录中的广告编号查找 hit 表中在所需日期范围内的匹配记录，然后对匹配的记录进行计数。广告编号为 48 的匹配记录有三个，广告编号为 101 的匹配记录有两个。

听起来很复杂！而这正是关系数据库系统所擅长的。这种复杂性在某种程度可以说是一种幻觉，因为上述每一步只不过是一个简单的匹配操作，它通过将一个表的行中的值与另一个表的行中的值相匹配，把一个表与另一个表相关联。这个简单的操作可以各种方式使用来回回答各种各样的问题。每个公司有多少个不同的广告？哪个公司的广告最受欢迎？每个广告带来的收入是多少？当前记账期中每个公司的总费用是多少？

现在我们已经介绍了关系数据库的理论，足以理解本书其余部分了，我们不必探究第三范式、实体关系图以及所有这一类的东西。如果您确实需要了解这些东西，那就太令人恐怖了，而且这也不是地方。建议您从阅读 C.J.Date 和 E.F.Codd 的某些书籍入手。

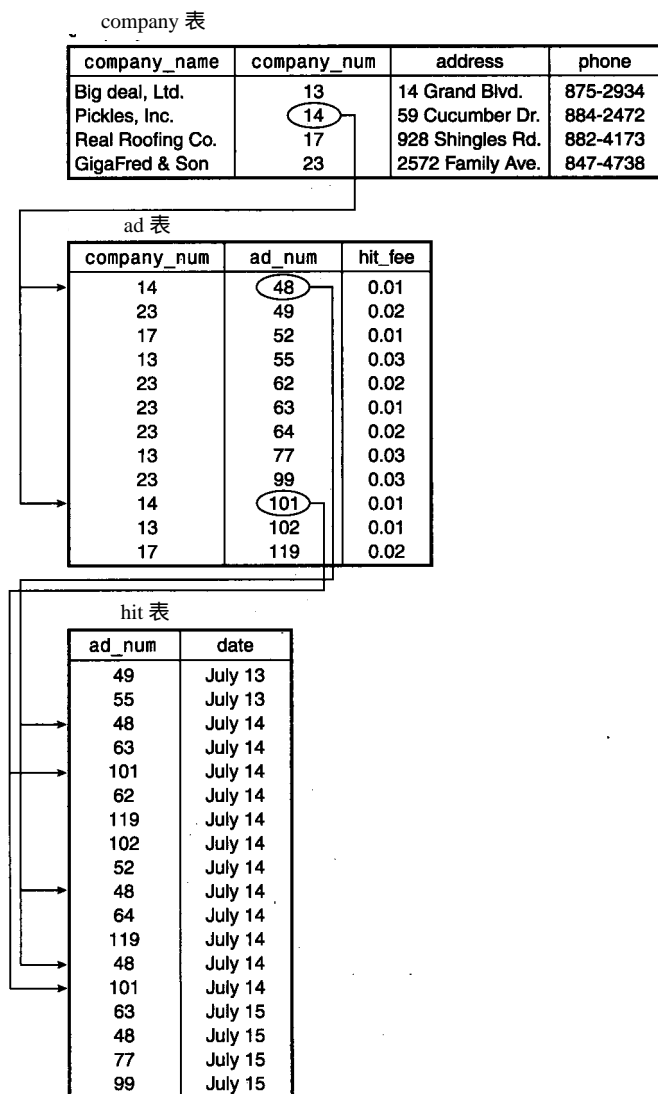


图1-1 标题广告的各表



### 1.3.2 查询语言术语

MySQL使用一种称为 SQL ( Structured Query Language ) 的语言。SQL 是当今的标准数据库语言，所有主要的数据库系统都使用它。SQL 具有多种不同的语句，所有语句都是以一种不枯燥并有用的方式设计来与数据库进行交互的。

正如其他语言一样，SQL 在初次接触时可能会令人感到有些古怪。例如，为了创建一个表，需要告诉 MySQL 表结构应该是什么样的。我们可能会根据图表来想像一个表，但MySQL 不会，因此，在创建表时需要告诉 MySQL 一些东西，如下所示：

```
CREATE TABLE company
(
    company_name CHAR(30),
    company_num INT,
    address CHAR(30),
    phone CHAR(12)
)
```

如果您不熟悉 SQL 语句，可能会对这样的语句留下深刻的印象，但您不必以程序员的身份来学习怎样有效地使用 SQL。如果逐步熟悉了 SQL 语言之后，就会以一种不同的眼光来看待 CREATE TABLE 语句，会认为它是一个有助于描述自己信息的伙伴，而不是一种奇怪的胡言乱语。

### 1.3.3 MySQL 的体系结构术语

在您使用 MySQL 时，实际正使用以下两个程序，因为 MySQL 采用的是客户机/服务器体系结构：

数据库服务器是一个位于存放您的数据的机器上的程序。它监听从网络上传过来的客户机的请求并根据这些请求访问数据库的内容，以便向客户机提供它们所要求的信息。

客户机是连接到数据库服务器的程序，这些程序告诉服务器需要什么信息的查询。

MySQL 分发包括服务器和几个客户机程序。可根据要达到的目的来使用客户机。最常用的客户机程序为 mysql，这是一个交互式的客户机程序，它能发布查询并看到结果。其他的客户机程序有：mysqldump 和 mysqlimport，分别转储表的内容到某个文件或将文件的内容导入某个表；mysqladmin 用来查看服务器的状态并完成管理任务，如告诉服务器关闭等。如果具有标准的客户机不适合的应用，那么 MySQL 还提供了一个客户机编程库，可以编写自己的程序。客户机编程库可直接从 C 程序中调用，如果希望使用 C 语言以外的其他语言，还有几种其他的接口可用。

MySQL 的客户机/服务器体系结构具有如下好处：

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

不必在数据库所在的机器上注册。MySQL 知道怎样在因特网上工作，因此您可以在任何位置运行一个客户机程序，此客户机程序可以连接到网络上的服务器。距离不是问题，可从世界上的任何地方访问服务器。如果服务器位于澳大利亚的某台机器上，那么当您带着自己的便携式电脑到冰岛去旅行时，仍然可以访问自己的数据库。

这是否意味着任何人只要连接到因特网就可以访问您的数据？答案是否定的。MySQL 含有一个灵活的安全系统，只允许那些有权限访问数据的人访问。可以保证那些人只能做允许他们做的事。或许记账办公室的 Sally 能够读取和更新（修改）记录，而服务台的 Phil 只能查看记录。可以设置使用人员的权限。如果希望运行一个内含系统（独立系统），只要设置访问权限使客户机只能从服务器运行的主机上进行连接即可。

## 1.4 MySQL 教程

现在我们已经具备了所需的所有基础知识；可以将 MySQL 投入工作了！

本节提供一个教程，帮助熟悉 MySQL。在完成这个教程时，将创建一个样例数据库和这个数据库中的表，然后增加、检索、删除和修改信息与数据库进行交互。此外，在操作这个样例数据库的过程中，将能学到下列东西：

如何利用 mysql 客户机程序与 MySQL 通信。

SQL 语言的基本语句。（如果您曾经使用过其他 RDBMS，从而熟悉 SQL，那么浏览一下这个教程，看看 SQL 的 MySQL 版与您熟悉的版本有何差别也是很好的。）

正如上一节所述，MySQL 采用客户机/服务器体系结构，其中服务器运行在存放数据库的机器上，而客户机通过网络连接到服务器。这个教程主要基于 mysql 客户机的应用。mysql 读取您的 SQL 查询，将它们发送给服务器，并显示结果。mysql 运行在 MySQL 所支持的所有平台上，并提供与服务器交互的最直接的手段，因此，它首先是一个逻辑上的客户机。

在本书中，我们将用 samp\_db 作为样例数据库的名称。但是有可能在您完成本例子的过程中需要使用另一个数据库名。因为可能在您的系统上已经有某个人使用了 samp\_db 这个名称，或者管理员给您指定了另一个数据库名称。在后面的例子中，无论是哪种情况，都用数据库的实际名称代替 samp\_db。

表名可以像例子所显示的那样精确地使用，即使系统中的多个人都具有他们自己的样例数据库也是如此。顺便说一下，在 MySQL 中，如果有人使用了相同的表名也没什么关系。一旦各个用户都具有自己的数据库，MySQL 将一直保留这些数据库名，防止各用户互相干扰。

### 1.4.1 基本要求

为了试验这个教程中的例子，必须安装 MySQL。特别是必须具有对 MySQL 客户机和某个 MySQL 服务器的访问权。相应的客户机程序必须位于您的机器上。至少需要有 mysql 程序，最好还有 mysqlimport 程序。服务器也可以位于您的机器上，尽管这不是必须的。实际上，只要允许连接到服务器，那么服务器位于何位置都没有关系。

若服务器正巧运行在您的机器上，适当的客户机程序又已经安装，那么就可以开始试验了。如果您尚需设法搞到 MySQL，可参阅附录 A“获得和安装软件”的说明。如果您正自己安装 MySQL，可参阅这一章，或把它给管理员看。如果网络访问是通过一个因特网服务商（ISP）进行的，那么可查看该服务商是否拥有 MySQL。如果该 ISP 不提供 MySQL 服务，可查看附录 J“因特网服务商”以得到某些选择更适合的服务商的建议。

除 MySQL 软件外，还需要得到创建样例数据库及其表的权限。如果您没有这种权限，可以向 MySQL 管理员咨询。管理员可通过运行 mysql 并发布如下的命令提供这种权限：

```
GRANT ALL ON samp_db.* TO paul@localhost IDENTIFIED BY "secret"  
GRANT ALL ON samp_db.* TO paul@% IDENTIFIED BY "secret"
```

### MySQL 与 mysql 的区别

为了避免混淆，应该说明，“MySQL”指的是整个 MySQL RDBMS，而“mysql”代表的是一个特定的客户机程序名。它们的发音都是相同的，但可通过不同的大小写字符和字体来区分。

关于发音，MySQL 的发音为“my-ess-queue-ell”。我们知道这是因为 MySQL 参考指南中是这样发音的。而 SQL 的发音为“sequel”或“ess-queue-ell”。我不认为哪个发音更好一些。愿意读哪个音都可以，不过在您对别人读的时候，他可能会用他认为是“正确”的发音对您进行纠正。

前一个命令在 paul 从 localhost（服务器运行在正运行的同一主机）连接时，允许它完全访问 samp\_db 数据库及它的所有表。它还给出了一个口令 secret。第二个命令与第一个类似，但允许 paul 从任何主机上连接（“%”为通配符）。也可以用特定的主机名取代“%”，使 paul 只能从该主机上进行连接。（如果您的服务器允许从 localhost 匿名访问，由于服务器搜索授权表查找输入连接匹配的方式的原因，这样一个 GRANT 语句可能是必须的。）关于 GRANT 语句以及设置 MySQL 用户账号的更详细信息，可在第 11 章“常规的 MySQL 管理”找到。

#### 1.4.2 取得样例数据库的分发包

这个教程在某些地方要涉及来自“样例数据库分发包”中的文件。有的文件含有帮助来设置样例数据库的查询或数据。为了得到这个分发包，可参阅附录 A。在打开这个分发包时，将创建一个名为 samp\_db 的目录，此目录中含有所需的文件。无论您在哪个地方试验与样例数据库有关的例子，建议都移入该目录。

#### 1.4.3 建立和中止与服务器的连接

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

```
% mysql options
```

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

mysql 命令行的 options 部分可能是空的，但更可能的是发布一条类似如下的命令：

```
% mysql -h host_name -u user_name -p
```

在激活 mysql 时，有可能不必提供所有这些选项；确切使用的命令请咨询 MySQL 管理员。此外，可能还需要至少指定一个名称和一个口令。

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

下面介绍选项的含义：

-h host\_name（可选择形式：--host=host\_name）

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

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

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

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

```
set USER=paul
```

`-p` (可选择的形式：`--password`)

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

```
% mysql -h host_name -u user_name -p
```

```
Enter password:
```

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

如果完全省略了 `-p` 选项，mysql 就认为您不需要口令，不作提示。

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

例如，假定我的 MySQL 用户名和口令分别为 paul 和 secret，希望连接到在我注册的同一机器上运行的服务器上。下面的 mysql 命令能完成这项工作：

```
% mysql -u paul -p
```

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

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

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

```
% mysql -u paul -p
```

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

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

```
Your MySQL connection id is 1805 to server version: 3.22.25-log
```

```
Type 'help' for help.
```

```
mysql>
```

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

```
% mysql -h pit-viper.snake.net -u paul -p
```

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

有很多设置账号的方法，从而不必在每次运行 mysql 时都在连接参数中进行键入。这个问题在 1.5 节 “与 mysql 交互的技巧” 中介绍。您可能会希望现在就跳到该节，以便找到一些更易于连接到服务器的办法。

在建立了服务器的一个连接后，可在任何时候键入下列命令来结束会话：

```
mysql> QUIT
Bye
```

还可以键入 Control-D 来退出，至少在 UNIX 上可以这样。

#### 1.4.4 发布查询

在连接到服务器后，就可以发布查询了。本节介绍有关与 mysql 交互应该了解的一些知识。

为了在 mysql 中输入一个查询，只需键入它即可。在查询的结尾处，键入一个分号（“；”）并按 Enter 键。分号告诉 mysql 该查询是完整的。（如果您喜欢键入两个字符的话，也可以使用 “\g” 终止查询。）

在键入一个查询之后，mysql 将其发送到服务器上。该服务器处理此查询并将结果送回 mysql，mysql 将此结果显示出来。

下面是一个简单的查询例子和结果：

```
mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 1999-07-24 11:02:36 |
+-----+
1 row in set (0.00 sec)
```

它给出当前的日期和时间。（NOW() 函数本身并无多大用处，但可将其用于表达式中。如比较当前日期和其他日期的差异。）

mysql 还在结果中显示行数计数。本书在例子中一般不给出这个计数。

因为 mysql 需要见到分号才发送查询到服务器，所以在单一的行上不需要键入分号。如果有必要，可将一个查询分为几行，如下所示：

```
mysql> SELECT NOW(),
-> USER(),
-> VERSION()
-> ;
+-----+-----+-----+
| NOW() | USER() | VERSION() |
+-----+-----+-----+
| 1999-07-24 11:06:16 | paul@localhost | 3.23.1-alpha-log |
+-----+-----+-----+
```

请注意，在键入查询的第一行后，提示符从 ‘mysql’ 变成了 ‘->’；这表示 mysql 允许继续键入这个查询。这是一个重要的提示，因为如果在查询的末尾忘记了分号，此提示将有助于提醒您查询尚不完整。否则您会一直等下去，心里纳闷为什么 mysql 执行查询为什么这么长的时间还没完；而 mysql 也搞不清为什么结束查询的键入要花您那么多的时间！

大部分情况下，用大写字符、小写字符或大小写字符混合键入查询没什么关系。下列查询全是等价的：

```
SELECT USER()
select user()
SeLeCt UsEr()
```

本书中的例子用大写字符表示 SQL 关键字和函数名，用小写字符表示数据库、表和列名。



如果在查询中调用一个函数，在函数名和后跟的圆括号中间不允许有空格，例：

```
mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 1999-07-17 12:44:52 |
+-----+
```

```
mysql> SELECT NOW ();
```

```
ERROR 1064: You have an error in your SQL syntax near '()' at line 1
```

这两个查询看上去差别不大，但第二个失败了，因为圆括号并没有紧跟在函数名的后面。

如果已经开始键入一个多行的查询，而又不想立即执行它，可键入 ‘\c’ 来跳过（放弃）它，如：

```
mysql> SELECT NOW(),
-> VERSION(),
-> \c
mysql>
```

请注意，提示符又变回了 ‘mysql>’，这表示 mysql 为键入的新查询作好了准备。

可将查询存储在一个文件中并告诉 mysql 从文件中读取查询而不是等待键盘输入。可利用外壳程序键入重定向实用程序来完成这项工作。例如，如果在文件 my\_file.sql 中存放有查询，可如下执行这些查询：

```
% mysql < my_file.sql
```

可用这种办法调用任何所需的文件。这里用后缀为 “.sql” 来表示该文件含有 SQL 语句。

执行 mysql 的这种方法将在输入数据到 samp\_db 数据库时的“增加新记录”中使用。为了装载一个表，让 mysql 从某个文件中读取 INSERT 语句比每次用手工键入这些语句更为方便。

本教程的其余部分向您提供了许多可以自己试试的查询。这些查询以 ‘mysql>’ 提示为前导后跟结束分号，这些例子通常都给出了查询输出结果。可以按给出的形式键入这些查询，所得到的结果应该与自学材料中的相同。

给出的查询中无提示符的或无分号语句结束符的只是用来说明某个要点，不用执行它们。（如果愿意您可以试一下，但如果试的话，请记住给语句末尾加一个分号。）

本书后面的章节中，我们一般不给出 ‘mysql>’ 提示或 SQL 语句的分号。这样做的原因是为了可以在非 mysql 客户机程序的语言环境（如在 Perl 脚本中或 PHP 脚本中）中发布查询，在这些语言环境中，既无提示符也不需要分号。在专门针对 mysql 输入一个查询的场合会作出相应的说明。

#### 1.4.5 创建数据库

现在开始创建 samp\_db 样例数据库及其表，填充这些表并对包含在这些表中的数据进行一些简单的查询。

使用数据库涉及几个步骤：

- 1) 创建（初始化）数据库。
- 2) 创建数据库中的表。
- 3) 对表进行数据插入、检索、修改或删除。

检索现有数据是对数据库执行的最简单且常见的操作。另外几个最简单且常见的操作是

插入新数据、更新或删除现有数据。较少使用的操作是创建表的操作，而最不常用的操作是创建数据库。

我们将从头开始，先创建数据库，再插入数据，然后对数据进行检索。

为了创建一个新的数据库，用 `mysql` 连接到数据库然后发布 `CREATE DATABASE` 语句，此语句指定了数据库名：

```
mysql> CREATE DATABASE samp_db;
```

在创建表以及对这些表进行各种操作之前，必须先创建 `samp_db` 数据库。

创建数据库后，这个新创建的数据库并不是当前数据库。这可从执行下面的查询看出：

```
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
|             |
+-----+
```

为了使 `samp_db` 成为当前数据库，发布 `USE` 语句即可：

```
mysql> USE samp_db
```

`USE` 为少数几个不需要终结符的语句之一，当然，加上终结符也不会出错。`HELP` 是另一个不需要终结符的语句。如果了解不需要终结符的语句有哪些，可发布 `HELP` 语句。

在发布了 `USE` 语句后，`samp_db` 成为缺省数据库：

```
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| samp_db    |
+-----+
```

使数据库成为当前数据库的另一个方法是在激活 `mysql` 时在命令行上指定它，如下所示：

```
% mysql samp_db
```

事实上，这是一个命名要使用的数据库的方法。如果需要连接参数可在数据库名前指定。例如，下列两个命令使我们能连接到在本地主机和 `pit-viper.snake.net` 上的 `samp_db` 数据库上：

```
% mysql -u paul -p samp_db
% mysql -h pit-viper.snake.net -u paul -p samp_db
```

除非另有指定，否则后面的例子都假定在激活 `mysql` 时，在命令行上给出 `samp_db` 使其成为当前数据库。如果激活数据库时忘了在命令行上指定数据库，只需发布 `USE samp_db` 语句即可。

#### 1.4.6 创建表

本节中，我们将创建样例数据库 `samp_db` 所需的表。我们首先考虑美国历史同盟需要的表。然后再考虑学分保存方案所需的表。在某些数据库的书籍中，在这里要大讲分析与设计、实体—关系图、标准程序以及诸如此类的东西。这里确实也可以讲这些东西，但是我宁可只讲点实用的东西，比方说，我们的数据库应该是怎样的：数据库中将包含什么内容，每个表中有哪些数据以及由决定如何表示数据而带来的一些问题。

这里所作出的关于数据表示的选择并不是绝对的。在其他场合下，可能会选择不同的方式来表示类似的数据，这取决于应用的需要以及打算将数据派何用途。

## 1. 美国历史同盟所需的表

美国历史同盟的表设计相当简单：

总统(president)表。此表含有描述每位总统的记录。同盟站点上的联机测验要使用这个表。

会员(member)表。此表用来维护同盟每个会员的当前信息。这些信息将用来建立会员地址名录的书面和联机版本、发送会员资格更新提示等等。

### (1) president表

president 表很简单，因此我们先讨论它。这个表将包含每位美国总统的一些基本信息：

姓名。姓名在一个表中可用几种方式表示。如，可以用一个单一的列来存放完整的姓名，或者用分开的列来分别容纳名和姓。当然用单一的列更为简单，但是在使用上会带来一些限制，如：

如果先输入只有名的姓名，则不可能对姓进行排序。

如果先输入只有姓的姓名，就不可能对具有名的姓名进行显示。

难以对姓名进行搜索。例如，如果要搜索某个特定的姓，则必须使用一个特定的模式，并且查找与这个模式匹配的姓名。这样较之只查找姓效率更低和更慢。

member 表将使用单独的名和姓的列以避免这些限制。

名列还存放中名（注：西方国家的姓名一般将名放在前，姓放在后，而且除了有名和姓外，有时还有中名，这是在位置上介于名和姓之间的中间名字）或首字母。这样应该不会削弱我们可能进行的任何一种排序，因为一般不可能对中名进行排序（或者甚至不会对名进行排序）。姓名即可以“Bush, George W.”格式显示，也可以“George W.Bush”格式显示。

还有一种稍显复杂一点的情形。一个总统（Jimmy Carter）在其姓名的末尾处有一个“Jr.”，这时怎样做？根据名字打印的格式，这个总统的姓名显示为“James E.Carter,Jr.”或“Carter, James E., Jr.”，“Jr.”与名和姓都没有关系，因此我们将建另外一个字段来存放姓名的后缀。这表明在试图确定怎样表示数据时，即使一个特殊的值也可能会带来问题。它也表明，为什么在将数据放入数据库前，尽量对数据值的类型进行了解是一个很好的想法。如果对数据了解不够，那么有可能在已经开始使用一个表后，不得不更改该表的结构。这不一定是个灾难，但通常应该避免。

出生地（城市和州）。就像姓名一样，出生地也可以用单个列或多个列来表示。使用单列更为简单些，但正如姓名中的情形一样，独立的多个列使我们完成用单个列不方便完成的事情。例如，如果城市和州分别给出，查找各位总统出生在哪个州的记录就会更容易一些。

出生日期和死亡日期。这里，唯一特殊的问题是我们不能要求都填上死亡日期，因为有的总统现在还健在。MySQL 提供了一个特殊的值 NULL，表示“无值”，可将其用在死亡日期列中以表示“仍然健在”。

### (2) member 表

存储历史同盟会员清单的 member 表在每个记录都包含单个人员的基本描述信息这一点上，类似于 president 表。但是每个 member 的记录所含的列更多，member 表的各列如下：

姓名。使用如 president 表一样的三个列来表示：姓、名（如果可能的话还有中名）、后缀。

ID 号。这是开始记录会员时赋给每个会员的唯一值。以前同盟未用 ID 号，但现在的记录做得更有系统性，所以最好开始使用 ID 号。（我希望您找到有利于使用 MySQL 并考虑到其他的将它用于历史同盟记录的方法。使用数字，将 member 表中的记录与其他与会员有关的表中的记录相关联要更容易一些。）

截止日期。会员必须定期更新他们的会员资格以免作废。对于某些应用，可能会用到最近更新的日期，但是近更新日期不适合于历史同盟。

会员资格可在可变的年数内（一般为一年、二年、三年或五年）更新，而最近更新的日期将不能表示下一次更新必须在何时进行。此外，历史同盟还允许有终生会员。我们可以用未来一个很长的日期来表示终生会员，但是用 NULL 似乎更为合适，因为“无值”在逻辑上对应于“永不终止”。

电子邮件地址。对于有电子邮件地址的会员，这将使他们能很容易地进行相互之间的通信。作为历史同盟秘书，这使您能电子化地发送更新通知给会员，而用不着发邮政信函。这比到邮局发送信函更容易，而且也不贵。还可以用电子邮件给会员发送他们的地址名录条目的当前内容，并要求他们在有必要时更新信息。

邮政地址。这是与没有电子邮件（或没有返回信息）的会员联络所需要的。将分别使用街道地址、城市、州和 Zip 号。街道地址列又可以用于有诸如 P.O. Box 123 而不是 123 Elm St. 的会员的信箱号。

我们假定所有同盟会员全都住在美国。当然，对于具有国际会员的机构，此假设过于简化了。如果希望处理多个国家的地址，还需要对不同国家的地址格式作一些工作。例如，这里的 Zip 号就不是一个国际标准，有的国家有省而不是州。

电话号码。与地址字段一样，这个列对于联络会员也是很有用的。

特殊爱好的关键词。假定每个会员一般都对美国历史都有兴趣，但可能有的会员对某些领域有特殊的兴趣。此列记录了这些特殊的兴趣。会员可以利用这个信息来找到其他具有类似兴趣的会员。

### (3) 创建表

现在我们已经作好了创建历史同盟表的准备。我们用 CREATE TABLE 语句来完成这项工作，其一般格式如下：

```
CREATE TABLE tbl_name ( column_specs )
```

其中 tbl\_name 代表希望赋予表的名称。column\_specs 给出表中列的说明，以及索引的说明（如果有的话）。索引能使查找更快；我们将在第 4 章“查询优化”中对其作进一步的介绍。

president 表的 CREATE TABLE 语句如下所示：

```
CREATE TABLE president
(
    last_name VARCHAR(15) NOT NULL,
    first_name VARCHAR(15) NOT NULL,
    suffix VARCHAR(5) NULL,
    city VARCHAR(20) NOT NULL,
    state VARCHAR(2) NOT NULL,
    birth DATE NOT NULL,
    death DATE NULL
)
```

如果想自己键入这条语句，则调用 mysql，使 samp\_db 为当前数据库：

```
% mysql samp_db
```

然后，键入如上所示的 CREATE TABLE 语句。（请记住，语句结尾要增加一个分号，否则 mysql 将不知道哪儿是语句的结尾。）

为了利用来自样例数据库分发包的预先写下的描述文件来创建 president 表，可从外壳程序运行下列命令：

```
% mysql samp_db < create_president.sql
```

不管用哪种方法调用 mysql，都应该在命令行中数据库名的前面指定连接参数（主机名、用户名或口令）。

CREATE TABLE 语句中每个列的说明由列名、类型（该列将存储的值的种类）以及一些可能的列属性组成。

president 表中所用的两种列类型为 VARCHAR 和 DATE。VARCHAR(n) 代表该列包含可变长度的字符（串）值，其最大长度为 n 个字符。可根据期望字符串能有多长来选择 n 值。state 定义为 VARCHAR(2)；即所有州名都只用其两个字符的缩写来表示。其他的字符串列则需要更长一些，以便存放更长的值。

我们使用过的其他列类型为 DATE。这种列类型表示该列存储的是日期值，这一点也不令人吃惊。而令人吃惊的是，日期的表示以年份开头。其标准格式为“YYYY-MM-DD”（例如，“1999-07-18”）。这是日期表示的 ANSI SQL 标准。

我们用于 president 表的唯一列属性为 NULL（值可以缺少）和 NOT NULL（必须填充值）。多数列是 NOT NULL 的，因为我们总要有它们的值。可有 NULL 值的两个列是 suffix（多数姓名没有后缀）和 death（有的总统仍然健在，所以没有死亡日期）。

member 表的 CREATE TABLE 语句如下所示：

```
CREATE TABLE member
(
    last_name VARCHAR(20) NOT NULL,
    first_name VARCHAR(20) NOT NULL,
    suffix VARCHAR(5) NULL,
    expiration DATE NULL DEFAULT "0000-00-00",
    email VARCHAR(100) NULL,
    street VARCHAR(50) NULL,
    city VARCHAR(50) NULL,
    state VARCHAR(2) NULL,
    zip VARCHAR(10) NULL,
    phone VARCHAR(20) NULL,
    interests VARCHAR(255) NULL
)
```

将此语句键入 mysql 或执行下列外壳程序命令：

```
% mysql samp_db < create_member.sql
```

从列的类型来看，member 表并不很有趣：所有列中，除了一列之外，其他列都是可变长字符串。这个例外的列就是 expiration，为 DATE 型。终止日期值有一个缺省值为“0000-00-00”，这是一个非 NULL 的值，它表示未输入合法的日期值。这样做的原因是 expiration 可以是 NULL，它表示一个会员是终身会员。但是，因为此列可以为 NULL，除非另外指定一个不同的值，否则它将取缺省值“0000-00-00”。如果创建了一个新会员记录，但忘了指定终止日期，该会员将成为一个终身会员！通过采用缺省值“0000-00-00”的方法，避免了这个问题。它还向我们提供了一种手段，即可以定期地搜索这个值，以找出过去未正确输入终止日期的记录。



请注意，我们“忘了”放入会员 ID 号的列。这是专门为了以后练习使用 ALTER TABLE 语句而遗留下的。

现在让我们来验证一下 MySQL 是否确实如我们所期望的那样创建了表。在 mysql 中，发布下列查询：

```
mysql> DESCRIBE president;
```

Field	Type	Null	Key	Default	Extra
last_name	varchar(15)				
first_name	varchar(15)				
suffix	varchar(5)	YES		NULL	
city	varchar(20)				
state	char(2)				
birth	date			0000-00-00	
death	date	YES		NULL	

与 MySQL 3.23 一样，此输出还包括了显示访问权限信息的另一个列，这里没有给出，因为它使每行太长，不易显示。

这个输出结果看上去和我们所期望的非常一致，除了 state 列的信息显示它的类型为 CHAR(2)。这就有点古怪了，我们不是定义它为 VARCHAR(2) 了吗？是的，是这样定义的，但是 MySQL 已经悄悄地将此类型从 VARCHAR 换成了 CHAR。原因是为了使短字符串列的存储空间利用更为有效，这里不多讨论。如果希望详细了解，可参阅第 3 章中关于 ALTER TABLE 语句的介绍。但对这里的使用来说，两种类型没有什么差别。

如果发布一个 DESCRIBE member 查询，mysql 也会显示 member 表的类似信息。

DESCRIBE 在您忘了表中的列名、需要知道列的类型、了解列有多宽等的时候很有用。它对于了解 MySQL 存储表行中列的次序也很有用。列的这个存储次序在使用 INSERT 或 LOAD DATA 语句时非常重要，因为这些语句期望列值以缺省列的次序列出。

DESCRIBE 可以省写为 DESC，或者，如果您喜欢键入较多字符，则 DESCRIBE tbl\_name 另一个等价的语句为 SHOW COLUMNS FROM tbl\_name。

如果忘了表名怎么办？这时可以使用 SHOW TABLES。对于 samp\_db 数据库，我们目前为止创建了两个表，其输出结果如下：

```
mysql> SHOW TABLES;
```

Tables_in_samp_db
member
president

如果您甚至连数据库名都记不住，可在命令行上调用 mysql 而不用给出数据库名，然后发布 SHOW DATABASES 查询：

```
mysql> SHOW DATABASES;
```

Database
menagerie
mysql
samp_db
test

数据库的列表在不同的服务器上是不同的，但是至少可以看到 `samp_db` 和 `mysql`；后一个数据库存放控制 MySQL 访问权限的授权表。

`DESCRIBE` 与 `SHOW` 查询具有可从外壳程序中使用的命令行等同物，如下：

```
% mysqlshow          与 SHOW DATABASES 一样列出所有数据库
% mysqlshow db_name   与 SHOW TABLES 一样列出给定数据库的表
% mysqlshow db_name tbl_name 与 DESCRIBE tbl_name 一样，列出给定表中的列
```

## 2. 用于学分保存方案的表

为了知道学分保存方案需要什么表，我们来看看在原来学分簿上是怎样记学分的。图 1-2 示出学分簿的一页。该页的主体是一个记录学分矩阵。还有一些对学分有意义的必要信息。学生名和 ID 号列在矩阵的一端。（为了简单好看，只列出了四个学生。）在矩阵顶端，记录了进行测验和测试的日期。图中示出 9月3号、6号、16号和23号进行测验，9月9号和10月1号进行测试。

为了利用数据库来记录这些信息，需要一个学分表。这个表中应该包含什么记录呢？很明显，每一行都需要有学生名、测验或测试的日期以及学分。图 1-3 示出了用这样的表表示的一些来自学分簿的学分。（日期以 MySQL 的表示格式“YYYY-MM-DD”表示。）

students		scores						
ID	name	Q 9/3	Q 9/6	T 9/9	Q 9/16	Q 9/23	T 10/1	...
1	Billy	14	10	73	14	15	67	...
2	Missy	17	10	68	17	14	73	...
3	Johnny	15	10	78	12	17	82	...
4	Jenny	14	13	85	13	19	79	...
...	...	...	...	...	...	...	...	...

图1-2 学分簿样例

score 表

name	date	score
Billy	1999-09-23	15
Missy	1999-09-23	14
Johnny	1999-09-23	17
Jenny	1999-09-23	19
Billy	1999-10-01	67
Missy	1999-10-01	73
Johnny	1999-10-01	82
Jenny	1999-10-01	79

图1-3 初步的学分表设计

但是，以这种方式设置表似乎有点问题。好像少了点什么。请看图 1-3 中的记录，我们分辨不出是测验的学分还是测试的学分。如果测验和测试的学分权重不同，在确定最终的学分等级时知道学分的类型是很重要的。或许可以试着从学分的取值范围来确定学分的类型（测验的学分一般比测试的学分少），但是这样做很不方便，因为这需要进行判断，而且在数据中也不明显。

可以通过记录学分的类型来进行区分，如对学分表增加一列，此列包含“T”或“Q”以表示是“测试”或是“测验”，如图 1-4 所示。这具有使学分数据类型清晰易辨的优点。不利的地方是这个信息有点冗余。显然对具有同一给定日期的记录，学分的类型列总是取相同的值。9月23日的学分总是为“Q”类型，而 10月1日的学分其类型总是具有“T”类型。这样令人很不满意。如果我们以这种方式记录一组测验或测试的学分，不仅要为每个新记录输入相同的日期，而且还要一再重复地输入相同的学分类型。谁会希望一再输入冗余的信息呢？

我们可以试试另外一种表示。不在 `score` 表中记录学分类型，而是从日期上区分它们。我们可

score 表

name	date	score	type
Billy	1999-09-23	15	Q
Missy	1999-09-23	14	Q
Johnny	1999-09-23	17	Q
Jenny	1999-09-23	19	Q
Billy	1999-10-01	67	T
Missy	1999-10-01	73	T
Johnny	1999-10-01	82	T
Jenny	1999-10-01	79	T

图1-4 `score` 表的设计，包括学分类型

以做一个日期列表，用它来记录每个日期发生的“学分事件”（测验或测试）。然后可以将学分与这个事件列表中的信息结合，确定学分是测验学分还是测试学分。这只要将 score 表记录中的日期与 event 表中的日期相匹配得出事件类型即可。图 1-5 示出这个表的设计并演示了 score 表记录与 9 月 23 日这个日期相关联的工作。通过将 score 表中的记录与 event 表中记录相对应，我们知道这个学分来自测验。

score 表			event 表	
name	date	score	date	type
Billy	1999-09-23	15	1999-09-03	Q
Missy	1999-09-23	14	1999-09-06	Q
Johnny	1999-09-23	17	1999-09-09	T
Jenny	1999-09-23	19	1999-09-16	Q
Billy	1999-10-01	67	1999-09-23	Q
Missy	1999-10-01	73	1999-10-01	T
Johnny	1999-10-01	82		
Jenny	1999-10-01	79		

图1-5 score 和 event 表，按日期关联

这比根据某些猜测来推断学分类型要好得多；我们可以根据明确记录在数据库中的数据来直接得到学分类型。这也比在 score 表中记录学分类型更好，因为我们只需对每个类型记录一次。

但是，在第一次听到这种事情时（即结合使用多个表中的信息），可能会想，“嗯，这是一个好主意，但是不是要做很多工作呢？会不会使工作更复杂了？”

在某种程度上，这种想法是对的。处理两个记录表比处理一个要复杂。但是再来考察一下学分簿（见图 1-2）。不是也记录了两套东西吗？考虑下列事实：

在学分矩阵中用两个单元记录学分，其中每个单元都是按学生名字和日期（在矩阵的旁边和顶上）进行索引的。这代表了一组记录；与 score 表的作用相同。

怎样知道每个日期代表的事件类型呢？在日期上方写了字符“T”或“Q”！因此，也在矩阵顶上记录了日期和学分类型之间的关系。它代表第二组记录；与 event 表的作用相同。

换句话说，这里建议在两个表中记录信息与用学分簿记录信息所做的工作没什么不同。唯一不同的是，这两组信息在学分簿中不是那么明显地被分开。

在图 1-5 中所示的 event 表的设计中加了一个要求，那就是日期必须是唯一的，因为要用它连接 score 与 event 表的记录。换句话说，同一天不能进行两次测验，或者同一天不能进行一次测验和一次测试。否则，将会在 score 表中有两个记录并且在 event 表中也有两个记录，全都具有相同的日期，这时就不知道应如何将 score 的记录与 event 的记录进行匹配。

如果每天不多于一个学分事件，这就是一个永远不会出现的问题，可是事实并非如此简单。有时，一天中可能会有不止一个学分事件。我常听有的人说他们的数据，“那种古怪情况从不会出现。”然而，如果这种情况确实出现时，就必须重新设计表以适应这种情况引起的问题。

最好是预先考虑以后可能出现的问题，并预先准备好怎样处理他们。因此，我们假定有时可能会需要同一天记录两组学分。我们怎样处理呢？如果出现这种情况，问题并不难解决。只要对处理数据的方式作一点小的更改，就可使同一日期上有多个事件而不会引起问题：

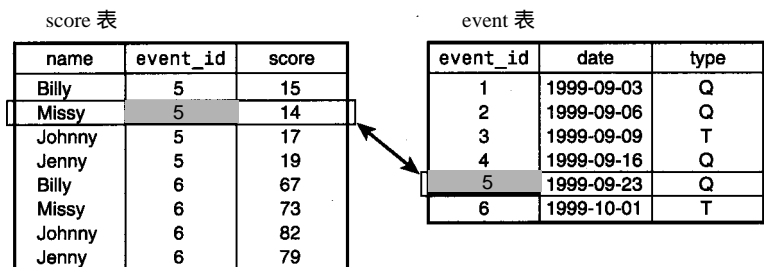
1) 增加一个列到 event 表，并用它来给表中每个记录分配一个唯一的编号。实际上这就

给了每个事件一个唯一的 ID 号，因此我们称该列为 event\_id 列。（如果觉得这好像是做傻事，可看一下图 1-2 中的学分簿，其中已经有这个特征了。事件 ID 正好与学分簿分数矩阵中列号相似。这个编号可能没有清晰地写在那儿并标上“事件 ID，”但是它确实在那儿。）

2) 当向 score 表中输入学分时，输入的是事件 ID 而不是日期。

这些改变的结果如图 1-6 所示。现在连接 score 和 event 表时，用的是事件 ID 而不是日期，而且不仅用 event 表来决定每个学分的类型，而且还用它来决定其日期。并且在 event 表中不再有日期必须唯一这个限制，而唯一的是事件 ID。这表示同一天可以有一打测试和测验，而且能够在记录里边直接保存它们。（毫无疑问，学生们听到这个一定浑身发抖。）

不幸的是，从人的观点来看，图 1-6 中的表设计较前一个更不能令人满意。score 表也更为抽象一些，因为它包含的从直观上可以理解的列更少。而图 1-4 中此表的设计直观且容易理解，因为那个 score 表具有日期和学分类型的列。当前的 score 表如图 1-6 所示，日期和学分类型的列都没有了。这极大地去除了作为人能够很容易考虑的一切。谁希望看到其中有“事件 ID”的 score 表？如果有的话，也不代表我们大多数人。



name	event_id	score
Billy	5	15
Missy	5	14
Johnny	5	17
Jenny	5	19
Billy	6	67
Missy	6	73
Johnny	6	82
Jenny	6	79

event_id	date	type
1	1999-09-03	Q
2	1999-09-06	Q
3	1999-09-09	T
4	1999-09-16	Q
5	1999-09-23	Q
6	1999-10-01	T

图1-6 score 表与 event 表，按事件 ID 关联

此时，可看到能够电子化地完成学分记录，且在赋予学分等级时不必做各种乏味的手工计算。但是，在考虑了如何实际在一个数据库中表示学分信息后，又会被怎样抽象和拆分成学分信息的表示难住了。

自然会产生一个问题：“根本不使用数据库可能会更好一些？或许 MySQL 不适合我？”正如您所猜测的那样，笔者将从否定的方面对这个问题进行回答，否则这本书就没必要再往下写了。不过，在考虑如何做一件工作时，应考虑各种情况并提问是否最好不使用数据库系统（如 MySQL）而使用一些别的东西（如电子表格等）：

学分簿有行和列，而电子表格也有。这使学分簿和电子表格在概念上和外观上都非常类似。

电子表格能够完成计算，可以利用一个计算字段来累计每个学生的学分。但是，要对测验和测试进行加权可能有点麻烦，但这也是可以办得到的。

另一方面，如果希望只查看某部分数据（如只查看学分或测试），进行诸如男孩与女孩的比较，或以一种灵活的方式显示合计信息等，情况又大有不同了。电子表格的功能显得要差一些，而关系数据库系统完成这些工作相当容易。

另外要考虑的一点是为了在关系数据库中进行表示而对数据进行抽象和分解，这个问题并不真的那么难以应付。只要考虑安排数据库使其不会以一种对您希望做的事无意义的方式来表示数据即可。但是，在确定了表示方式之后，就要靠数据库引擎来协调和表示数据了。您肯定不会希望将它视为一堆支离破碎的东西。

例如，在从 score 表中检索学时时，不希望看到事件 ID；但希望看到日期。这没有什么问题。数据库将会根据事件 ID 从 event 表中查找出日期。您还可能想要看看是测验的学分或测试的学分。这也不成问题。数据库将用相同的方法找出学分类型，也是利用事件 ID。请记住，这就是如像 MySQL 这样的关系数据库的优势所在，即，使一样东西与另一样东西相关联，以便从多个来源得出信息并以您实际想看到的形式提供出来。在学分保存数据的情况下，MySQL 确实利用事件 ID 将信息组合到了一起，而无需人工来完成这件事。

现在我们先来看看，如何使 MySQL 完成这种将一个东西与另一个东西相联系的工作。假定希望看到 1999年9月23号的学分，针对某个特定日期中给出的事件ID的学分查询如下所示：

```
SELECT score.name, event.date, score.score, event.type
FROM score, event
WHERE event.date = "1999-09-23"
AND score.event_id = event.event_id
```

相当吓人，是吗？这个查询通过将 score 表的记录与 event 表的记录连接（关联）来检索学生名、日期、学分和学分的类型。其结果如下所示：

name	date	score	type
Billy	1999-09-23	15	Q
Missy	1999-09-23	14	Q
Johnny	1999-09-23	17	Q
Jenny	1999-09-23	19	Q

您肯定注意到了，它与图 1-4 中给出的表设计相同，而且不需要知道事件 ID 就可得出这个结果，只需指出感兴趣的日期并让 MySQL 查找出哪个学分记录具有该日期即可。如果您一直担心抽象和分解会使我们损失一些东西的话，看到这个世界，就不会有这种担心了。

当然，在考虑过查询后，您可能对其他别的东西产生担心。即，这个查询看上去有点长并且也有点复杂；是不是做了很多工作写出这样的东西只是为了查找某个给定日期的学分？是的，确实是这样。但是，在每次想要发布一个查询时，有几种方法可以避免键入多行的 SQL。一般情况下，一旦您决定如何执行这样一个查询并将它保存起来后，就可以按需要多次执行它。我们将在 1.5 节“与 mysql 交互的技巧”中介绍怎样完成这项工作。

在上述查询的介绍中，我们有点超前了。不过，这个查询比起我们要实际用来得出学分的查询是有点简单了。原因是，我们还要对表的设计作更多的修改。我们将采用一个唯一的学生 ID，而不在 score 表中记录学生名。（即，我们将使用来自学分簿的“ID”列的值而不是来自“Name”列的值。）然后，创建另一个称为 student 的表来存放 name 和 student\_id 列（见图 1-7）。

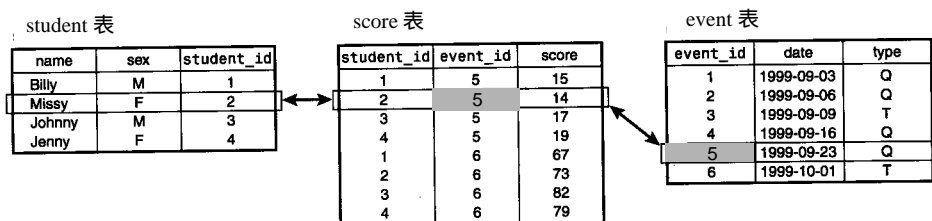


图1-7 student、score 和 event 表，按学生 ID 和事件 ID 关联



为什么要作出这种修改呢？只有一个原因，可能有两个学生有相同的名字。采用唯一的学生 ID 号可帮助区分他们的学分。（这与利用唯一的事件 ID 而不是日期来分辨出相同日期的测试或测验完全类似。）

在对表的设计作了这样的修改后，实际用来获得给定日期的学分查询变得更为复杂了一些，这个查询如下：

```
SELECT student.name, event.date, score.score, event.type
FROM event, score, student
WHERE event.date = "1999-09-23"
AND event.event_id = score.event_id
AND score.student_id = student.student_id
```

如果您不能立即清楚地读懂这个查询的意思的话，也不必担心。在进一步深入这个教程之后，就能看懂这个查询了。

将会从图 1-7 中注意到，在 student 表中增加了点学分簿中没有的东西。它包含了一个性别列。这便可以做一些简单的事情，如对班级中男孩和女孩的人数计数；也可以做一些更为复杂的事情，如比较男孩和女孩的学分。

我们已经设计完了学分保存的几乎所有的表。现在只需要另外一个表来记录出勤情况即可。这个表的内容相对较为直观，即，一个学生 ID 号和一个日期（见图 1-8）。表中的每行表示特定的学生在给定的日期缺勤。在学分时段末，我们将调用 MySQL 的计数功能来汇总此表的内容，以便得出每个学生的缺勤数。

adsence 表

student_id	date
2	1999-09-02
4	1999-09-15
2	1999-09-20

图1-8 absence 表

既然现在已经知道学分保存的各个表的结构，现在可以创建它们了。

student 表的 CREATE TABLE 语句如下：

```
CREATE TABLE student
(
    name VARCHAR(20) NOT NULL,
    sex ENUM('F','M') NOT NULL,
    student_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
)
```

将上述语句键入 mysql 或执行下列外壳程序命令：

```
% mysql samp_db < create_student.sql
```

CREATE TABLE 语句创建了一个名为 student 的表，它含有三列，分别为：name、sex 和 student\_id。

name 是一个可变长的字符串列，最多可存放 20 个字符。这个名字的表示比历史同盟表中所用的表示要简单，它只用了单一的列而不是分别的名和姓列。这是因为我们已经预先知道，不存在无需做额外的工作就使得在多个列上工作得更好的查询样例。

sex 表示学生是男孩还是女孩。这是一个 ENUM（枚举）列，表示只能取明确地列在说明中的值之一，这里列出的值为：“F”和“M”，分别表示女和男。在某列只具有一组有限值时，ENUM 类型非常有用。我们可以用 CHAR(1) 来代替它，但是 ENUM 更明确规定了列可以取什么值。如果对包括一个 ENUM 列的表发布一条 DESCRIBE tbl\_name 语句，MySQL 将确切地显示可取的值有哪些。

顺便说一下，ENUM 列中的值不一定只是单个字符。此列还可以定义为 ENUM（‘female’，‘male’）。

student\_id 为一个整数型列，它将包含唯一的 ID 号。通常，大概会从一个中心资料来源处（如学校办公室）取得学生的 ID 号，但在这里是我们自己定的。虽然 student\_id 列只包含一个数，但其定义包括几个部分：

INT 说明此列的值必须取整数（即无小数部分）。

UNSIGNED 不允许负数。

NOT NULL 表示此列的值必须填入。（任何学生都必须有一个 ID 号。）

AUTO\_INCREMENT 是 MySQL 中的一个特殊的属性。其作用为：如果在创建一个新的 student 表记录时遗漏了 student\_id 的值（或为 NULL），MySQL 自动地生成一个大于当前此列中最大值的唯一 ID 号。在录入学生表时将用到这个特性，录入学生表时可以只给出 name 和 sex 的值，让 MySQL 自动生成 student\_id 列值。

PRIMARY KEY 表示相应列的值为快速查找进行索引，并且列中的每个值都必须是唯一的。这样可防止同一名字的 ID 出现两次，这对于学生 ID 号来说是一个必须的特性。（不仅如此，而且 MySQL 还要求每个 AUTO\_INCREMENT 列都具有一个惟一索引。）

如果您不理解 AUTO\_INCREMENT 和 PRIMARY KEY 的含义，只要将其想像为一种为每个学生产生 ID 号的魔术方法即可。除了要求值唯一外，没有什么别的东西。

请注意：如果确实打算从学校办公室取得学生 ID 号而不是自动生成它们，则可以按相同的方法定义 student\_id 列，只不过不定义 AUTO\_INCREMENT 属性即可。

event 表如下定义：

```
CREATE TABLE event
(
    date DATE NOT NULL,
    type ENUM('T','Q') NOT NULL,
    event_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
)
```

将此语句键入 mysql 或执行下列外壳程序的命令：

```
% mysql samp_db < create_event.sql
```

所有列都定义为 NOT NULL，因为它们中任何一个值都不能省略。

date 列存储标准的 MySQL DATE 日期值，格式为“YYYY-MM-DD”（首先是年）。

type 代表学分类型。像 student 表中的 sex 一样，type 也是一个枚举列。所允许的值为“T”和“Q”，分别表示“测试”和“测验”。

event\_id 是一个 AUTO\_INCREMENT 列，类似于 student 表中的 student\_id 列。采用 AUTO\_INCREMENT 允许生成唯一的事件 ID 值。正如 student 表中的 student\_id 列一样，与值的惟一性相比，某个特定的值并不重要。

score 表如下定义：

```
CREATE TABLE score
(
    student_id INT UNSIGNED NOT NULL,
    event_id INT UNSIGNED NOT NULL,
    score INT NOT NULL,
    PRIMARY KEY (event_id, student_id)
)
```

将此语句键入 mysql 或执行下列外壳程序的命令：

```
% mysql samp_db < create_score.sql
```

score 为一个 INT ( 整型 ) 列。即, 假定学分值总是为一个整数。如果希望使学分值具有小数部分, 如 58.5, 应该采用浮点列类型, 如 FLOAT 或 DECIMAL。

student\_id 列和 event\_id 列都是整型, 分别表示每个学分所对应的学生和事件。通过利用它们来连接到 student 和 event 表, 我们能够知道学生名和事件的日期。我们将两个列组成了 PRIMARY KEY。这保证我们不会对同一测验或测试重复一个学生的学分。而且, 这样还很容易在以后更改某个学分。例如, 在发现学分数录入错时, 可以在利用 MySQL 的 REPLACE 语句放入一个新记录, 替换掉旧的记录。不需要执行 DELETE 语句与 INSERT 语句; MySQL 自动替我们做了。

请注意, 它是惟一的 event\_id 和 student\_id 的组合。在 score 表中, 两者自身都可能不惟一。一个 event\_id 值可有多个学分记录 ( 每个学生对应一个记录 ), 而每个 student\_id 值都对应多个记录 ( 每个测验和测试有一个记录 )。

用于出勤情况的 absence 表如下定义:

```
CREATE TABLE absence
(
    student_id INT UNSIGNED NOT NULL,
    date DATE NOT NULL,
    PRIMARY KEY (student_id, date)
)
```

将此语句键入 mysql 或执行下列外壳程序的命令:

```
% mysql samp_db < create_absence.sql
```

student\_id 和 date 列两者都定义为 NOT NULL, 不允许省略值。应定义这两列的组合为主键, 以免不当心建立了重复的记录。重要的是不要对同一天某个学生的缺旷进行重复计数。

#### 1.4.7 增加新记录

至此, 我们的数据库及其表都已经创建了, 在下一节 “检索信息” 中, 我们将看到怎样从数据库中取出数据。现在我们先将一些数据放入表中。

在数据库中加入数据有几种方法。可通过发布 INSERT 语句手工将记录插入某个表中。还可以通过从某个文件读取它们来增加记录, 在这个文件中, 记录既可以是利用 LOAD DATA 语句或 mysqlimport 实用程序装入的原始数据值, 也可以是预先写成可馈入 mysql 的 INSERT 语句的形式。

本节介绍将记录插入表的每种方法。您所应做的是演习各种方法以明了它们是如何起作用的。然后到本节结束处运行那儿给出的命令来清除表并重装它们。这样做, 能够保证表中含有作者撰写下一节时所处理的相同记录, 您也能得到相同的结果。

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

可指定所有列的值:

```
INSERT INTO tbl_name VALUES(value1,value2,...)
```

例如:

```
mysql> INSERT INTO student VALUES('Kyle','M',NULL);
mysql> INSERT INTO event VALUES("1999-9-3","Q",NULL);
```

“ INTO ” 一词自 MySQL 3.22.5 以来是可选的。( 这一点对其他形式的 INSERT 语句也成

立。) VALUES 表必须包含表中每列的值, 并且按表中列的存放次序给出。(一般, 这就是创建表时列的定义次序。如果不能肯定的话, 可使用 DESCRIBE tbl\_name 来查看这个次序。)

在 MySQL 中, 可用单引号或双引号将串和日期值括起来。上面例子中的 NULL 值是用于 student 和 event 表中的 AUTO\_INCREMENT 列的。(插入“错误”的值将导致下一个 student\_id 或 event\_id 号的自动生成。)

自 3.22.5 以来的 MySQL 版本允许通过指定多个值的列表, 利用单个的 INSERT 语句将几行插入一个表中, 如下所示:

```
INSERT INTO tbl_name VALUES(...),(...),...
```

例如:

```
mysql> INSERT INTO student VALUES('Abby','F',NULL),('Kyle','M',NULL);
```

这比多个 INSERT 语句的键入工作要少, 而且服务器执行的效率也更高。

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

```
INSERT INTO tbl_name (col_name1,col_name2,...) VALUES(value1,value2,...)
```

例如:

```
mysql> INSERT INTO member (last_name,first_name) VALUES('Stein','Waldo');
```

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

```
mysql> INSERT INTO student (name,sex) VALUES('Abby','F'),('Kyle','M');
```

在列的列表中未给出名称的列都将赋予缺省值。

自 MySQL 3.22.10 以来, 可以 col\_name = value 的形式给出列和值。

```
INSERT INTO tbl_name SET col_name1=value1, col_name2=value2, ...
```

例如:

```
mysql> INSERT INTO member SET last_name='Stein',first_name='Waldo';
```

在 SET 子句中未命名的行都赋予一个缺省值。

使用这种形式的 INSERT 语句不能插入多行。

将记录装到表中的另一种方法是直接从文件读取数据值。可以用 LOAD DATA 语句或用 mysqlimport 实用程序来装入记录。

LOAD DATA 语句起批量装载程序的作用, 它从一个文件中读取数据。可在 mysql 内使用它, 如下所示:

```
mysql> LOAD DATA LOCAL INFILE "member.txt" INTO TABLE member;
```

该语句读取位于客户机上当前目录中数据文件 member.txt 的内容, 并将其发送到服务器装入 member 表。

如果您的 MySQL 版本低于 3.22.15, 则 LOAD DATA LOCAL 不起作用, 因为那时从客户机读取数据的能力是在 LOAD DATA 上的。(没有 LOCAL 关键字, 被读取的文件必须位于服务器主机上, 并且需要大多数 MySQL 用户都不具备的服务器访问权限。)

缺省时, LOAD DATA 语句假定列值由 tab 键分隔, 而行则以换行符结束。还假定各个值是按列在表中的存放次序给出的。也有可能需要读取其他格式的文件, 或者指定不同的列次序。更详细的内容请参阅附录 D 的 LOAD DATA 的条款。

mysqlimport 实用程序起 LOAD DATA 的命令行接口的作用。从外壳程序调用

mysqlimport，它生成一个 LOAD DATA 语句：

```
% mysqlimport --local samp_db member.txt
```

mysqlimport 生成一个 LOAD DATA 语句，此语句使 member.txt 文件被装入 member 表。如果您的 MySQL 版本低于 3.22.15，这个实用程序不起作用，因为 --local 选项需要 LOAD DATA LOCAL。正如使用 mysql 一样，如果您需要指定连接参数，可在命令行上数据库名前指定它们。

mysqlimport 从数据文件名中导出表名（它将文件名第一个圆点前的所有字符作为表名）。例如，member.txt 将被装入 member 表，而 president.txt 将被装入 president 表。如果您有多个需要装入单个表的文件，应仔细地选择文件名，否则 mysqlimport 将不能使用正确的表名。对于如像 member1.txt 与 member2.txt 这样的文件名，mysqlimport 将会认为相应的表名为 member1 和 member2。不过，可以使用如 member.1.txt 和 member.2.txt 或 member.txt1 和 member.txt2 这样的文件名。

在试用过这些记录追加的方法后，应该清除各个表并重新装载它们，以便它们的内容与下一节假定的内容相同。

从外壳程序执行下列命令：

```
% mysql samp_db < insert_president.sql
% mysql samp_db < insert_member.sql
% mysql samp_db < insert_student.sql
% mysql samp_db < insert_score.sql
% mysql samp_db < insert_event.sql
% mysql samp_db < insert_absence.sql
```

每个文件都含有一个删除可能曾经插入到表中的记录的 DELETE 语句，后跟一组 INSERT 语句以初始化表的内容。如果不希望分别键入这些命令，可试一下下列语句：

```
% cat insert_*.sql | mysql samp_db
```

#### 1.4.8 检索信息

现在各个表已经创建并装有数据了，因此让我们来看看可以对这些数据做些什么。

SELECT 语句允许以一般的或特殊的方式检索和显示表中的信息。它可以显示表的整个内容：

```
SELECT * FROM president
```

或者只显示单个行中单个列的内容：

```
SELECT birth_date FROM president WHERE last_name = "Eisenhower"
```

SELECT 语句有几个子句（部件），可以根据需要用来检索感兴趣的信息。每个子句都可简单、可复杂，从而 SELECT 作为一个总的语句也繁简皆宜。但是，可以放心，本书中不会有花一个钟头来编写的长达数页的查询。（我在书中看到有很长的查询时，一般会立即跳过它们，因此我猜您也会这样。）

SELECT 语句的一般形式为：

SELECT 要选择的東西

FROM 一个或多个表

WHERE 数据必须满足的条件

记住，SQL 为一个自由格式的语言，因此在您编写 SELECT 查询时，语句的断行不必严格依照本书。



为了编写 SELECT 语句，只需指定需要检索什么，然后再选择某些子句即可。刚才给出的子句“FROM”、“WHERE”是最常用的，还有一些其他的子句，如 GROUP BY、ORDER BY 和LIMIT 等。

FROM 子句一般都要给出，但是如果不从表中选择数据，也可不给出。例如，下列查询只显示某些可以直接计算而不必引用任何表的表达式的值，因此不需要用 FROM 子句：

```
mysql> SELECT 2+2, "Hello, world", VERSION();
+-----+-----+-----+
| 2+2 | Hello, world | VERSION() |
+-----+-----+-----+
| 4 | Hello, world | 3.23.0-alpha-log |
+-----+-----+-----+
```

在确实使用一个 FROM 子句指定了要从其中检索数据的表时，SELECT 语句的最“普通”的格式是检索所有内容。用“\*”来表示“所有列”。下面的查询将从 student 表中检索所有行并显示：

```
mysql> SELECT * FROM student;
+-----+-----+-----+
| name | sex | student_id |
+-----+-----+-----+
| Joseph | M | 1 |
| Kyle | M | 2 |
| Abby | F | 3 |
| Nathan | M | 4 |
...
```

各列按它们 MySQL 在表中存放的次序出现。该次序与发布 DESCRIBE student 语句时显示的列次序相同。（例子末尾的“...”表示此查询返回的输出行比这里显示的还要多。）

可明确地命名希望得到的一列或多列。如果只选择学生名，发布下列语句：

```
mysql> SELECT name FROM student;
+-----+
| name |
+-----+
| Joseph |
| Kyle |
| Abby |
| Nathan |
...
```

如果名字不止一列，可用逗号分隔它们。下列的语句与 SELECT \* FROM student 等价，只是明确地指出了每一列：

```
mysql> SELECT name, sex, student_id FROM student;
+-----+-----+-----+
| name | sex | student_id |
+-----+-----+-----+
| Joseph | M | 1 |
| Kyle | M | 2 |
| Abby | F | 3 |
| Nathan | M | 4 |
...
```

可按任意次序给出列：

```
SELECT name, student_id FROM student
SELECT student_id, name FROM student
```

如果有必要，同一列甚至也可以给出多次，虽然这样做一般是没有意义的。

列名在 MySQL 中不区分大小写的。下面的查询是等同的：

```
SELECT name, student_id FROM student
SELECT NAME, STUDENT_ID FROM student
SELECT nAmE, sTuDeNt_Id FROM student
```

数据库和表名有可能区分大小写的；这有取决于服务器主机上使用的文件系统。在 UNIX 上运行的服务器对数据库名和表名是区分大小写的，因为 UNIX 的文件名是区分大小写的。Windows 的文件名不区分大小写，因此运行在 Windows 上的服务器对数据库名和表名不区分大小写。

MySQL 允许您一次从多个表中选择列。我们将这个内容留到“从多个表中检索信息”小节去介绍。

### 1. 指定检索条件

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

可查找数字值：

```
mysql> SELECT * FROM score WHERE score > 95;
```

student_id	event_id	score
4	3	96
5	3	97
7	3	97
13	6	96
14	6	99
22	6	97
24	6	97
26	6	96

也可以查找串值。（注意，一般串的比较是不区分大小写的。）

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name="ROOSEVELT";
```

last_name	first_name
Roosevelt	Theodore
Roosevelt	Franklin D.

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name="roosevelt";
```

last_name	first_name
Roosevelt	Theodore
Roosevelt	Franklin D.

可以查找日期值：

```
mysql> SELECT last_name, first_name, birth FROM president
-> WHERE birth < "1750-1-1";
```

last_name	first_name	birth
-----------	------------	-------

```

| Washington | George   | 1732-02-22 |
| Adams      | John     | 1735-10-30 |
| Jefferson  | Thomas   | 1743-04-13 |
+-----+

```

可搜索组合值：

```

mysql> SELECT last_name, first_name, birth, state FROM president
-> WHERE birth < "1750-1-1" AND (state="VA" OR state="MA");

```

```

+-----+
| last_name | first_name | birth      | state |
+-----+
| Washington | George   | 1732-02-22 | VA    |
| Adams      | John     | 1735-10-30 | MA    |
| Jefferson  | Thomas   | 1743-04-13 | VA    |
+-----+

```

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

表 1-1 算术运算符

运 算 符	说 明	运 算 符	说 明
+	加	*	乘
-	减	/	除

表 1-2 比较运算符

运 算 符	说 明	运 算 符	说 明
<	小于	!= 或 <>	不等于
<=	小于或等于	>=	大于或等于
=	等于	>	大于

在用表达式表示一个需要逻辑运算的查询时，要注意别混淆逻辑与运算符与我们平常使用的“与”的含义。假如希望查找“出生在 Virginia 的总统与出生在 Maryland 的总统”。应该注意怎样表示“与”的关系，能写成如下的查询吗？

```

SELECT last_name, first_name, state FROM president
WHERE state="VA" AND state="MA"

```

错了，因为这个查询的意思是“选择既出生在 Virginia 又出生在 Maryland 的总统”，不可能有同时出生在两个地点的总统，因此这个查询无意义。在英语中，可以用“and”表示这种选择，但在 SQL 中，应该用 OR 来连接两个条件，如下所示：

```

mysql> SELECT last_name, first_name, state FROM president
-> WHERE state="VA" OR state="MA";

```

```

+-----+
| last_name | first_name | state |
+-----+
| Washington | George   | VA    |
| Adams      | John     | MA    |
| Jefferson  | Thomas   | VA    |
+-----+

```

表 1-3 逻辑运算符

运 算 符	说 明
AND	逻辑与
OR	逻辑或
NOT	逻辑非

Madison	James	VA
Monroe	James	VA
Adams	John Quincy	MA
Harrison	William H.	VA
Tyler	John	VA
Taylor	Zachary	VA
Wilson	Woodrow	VA
Kennedy	John F.	MA
Bush	George W.	MA

这有时是可以觉察到的，不仅仅是在编写自己的查询时可以觉察到，而且在为他人编写查询时也可以知道。最好是在他人描述想要检索什么时仔细听，但不一定使用相同的逻辑运算符将他人的描述转录成 SQL 语句。对刚才所举的例子，正确的英语等价描述为“选择出生在 Virginia 或者出生在 Maryland 的总统。”

## 2. NULL 值

NULL 值是特殊的；因为它代表“无值”。不可能以评估两个已知值的相同方式将它与已知值进行评估。如果试图与通常的算术比较运算符一道使用 NULL，其结果是未定义的：

```
mysql> SELECT NULL < 0, NULL = 0, NULL != 0, NULL > 0;
```

```
+-----+-----+-----+-----+
| NULL < 0 | NULL = 0 | NULL != 0 | NULL > 0 |
+-----+-----+-----+-----+
|      NULL      |      NULL      |      NULL      |      NULL      |
+-----+-----+-----+-----+
```

事实上，甚至不能将 NULL 与它自身比较，因为不可能知道比较两个未知值的结果：

```
mysql> SELECT NULL = NULL, NULL != NULL;
```

```
+-----+-----+
| NULL = NULL | NULL != NULL |
+-----+-----+
|      NULL      |      NULL      |
+-----+-----+
```

为了进行 NULL 值的搜索，必须采用特殊的语法。不能用 = 或 != 来测试等于 NULL 或不等于 NULL，取而代之的是使用 IS NULL 或 IS NOT NULL 来测试。

例如，因为我们将健在总统的死亡日期表示为 NULL，那么可按如下语句查找健在的总统：

```
mysql> SELECT last_name, first_name FROM president WHERE death IS NULL;
```

```
+-----+-----+
| last_name | first_name |
+-----+-----+
| Ford      | Gerald R.  |
| Carter    | James E.   |
| Reagan    | Ronald W.  |
| Bush      | George W.  |
| Clinton   | William J. |
+-----+-----+
```

为了找到具有后缀部分的姓名，可利用 IS NOT NULL：

```
mysql> SELECT last_name, first_name, suffix
-> FROM president WHERE suffix IS NOT NULL;
```

```
+-----+-----+-----+
| last_name | first_name | suffix |
+-----+-----+-----+
| Carter    | James E.   | Jr.    |
+-----+-----+-----+
```

MySQL3.23 及以后的版本具有一个特殊的 MySQL 专有的比较运算符“<=>”，即使是 NULL 与 NULL 的比较，它也是可行的。用这个比较运算符，可将前面的两个查询重写为：

```
mysql> SELECT last_name, first_name FROM president WHERE death <=> NULL;
```

last_name	first_name
Ford	Gerald R
Carter	James E.
Reagan	Ronald W.
Bush	George W.
Clinton	William J.

```
mysql> SELECT last_name, first_name, suffix  
-> FROM president WHERE NOT (suffix <=> NULL);
```

last_name	first_name	suffix
Carter	James E.	Jr.

### 3. 对查询结果进行排序

有时我们注意到，在一个表装入初始数据后，对其发布一条 `SELECT * FROM tbl_name` 查询，检索出的行与这些行被插入的顺序是相同的。但不要认为这种情况是有规律的。如果在初始装入表后进行了行的删除和插入，就会发现服务器返回表的行次序被改变了。（删除记录在表中留下了未使用的“空位”，MySQL 在以后插入新记录时将会试图对其填补。）

缺省时，如果选择了行，服务器对返回行的次序不作任何保证。为了对行进行排序，可使用 `ORDER BY` 子句：

```
mysql> SELECT last_name, first_name FROM president  
-> ORDER BY last_name;
```

last_name	first_name
Adams	John
Adams	John Quincy
Arthur	Chester A.
Buchanan	James

...

在 `ORDER BY` 子句中，可在列名之后利用 `ASC` 或 `DESC` 关键字指定排序是按该列值的升序或降序进行的。例如，为了按倒序（降序）名排列总统名，可如下使用 `DESC`：

```
mysql> SELECT last_name, first_name FROM president  
-> ORDER BY last_name DESC;
```

last_name	first_name
Wilson	Woodrow
Washington	George
Van Buren	Martin
Tyler	John

...

如果在 `ORDER BY` 子句中，对某个列名既不指定 `ASC` 又不指定 `DESC`，则缺省的次序为升序。

在对可能包含 `NULL` 值的列进行排序时，如果是升序排序，`NULL` 值出现在最前面，如果是按降序排序，`NULL` 值出现在最后。



查询结果可在多个列上进行排序，而每个列的升序或降序可以互相独立。下面的查询从 president 表中检索行，并按出生的州降序、在每个州中再按姓氏的升序对检索结果进行排序：

```
mysql> SELECT last_name, first_name, state FROM president
-> ORDER BY state DESC, last_name ASC;
```

last_name	first_name	state
Arthur	Chester A.	VT
Coolidge	Calvin	VT
Harrison	William H.	VA
Jefferson	Thomas	VA
Madison	James	VA
Monroe	James	VA
Taylor	Zachary	VA
Tyler	John	VA
Washington	George	VA
Wilson	Woodrow	VA
Eisenhower	Dwight D.	TX
Johnson	Lyndon B.	TX

#### 4. 限制查询结果

如果一个查询返回许多行，但您只想看其中的几行，则可以利用 LIMIT 子句，特别是与 ORDER BY 子句结合时更是如此。MySQL 允许限制一个查询的输出为前 n 行。下面的查询选择了 5 位出生日期最早的总统：

```
mysql> SELECT last_name, first_name, birth FROM president
-> ORDER BY birth LIMIT 5;
```

last_name	first_name	birth
Washington	George	1732-02-22
Adams	John	1735-10-30
Jefferson	Thomas	1743-04-13
Madison	James	1751-03-16
Monroe	James	1758-04-28

如果利用 ORDER BY birth DESC 按降序排序，将得到 5 位最晚出生的总统。

LIMIT 也可以从查询结果中取出中间部分。为了做到这一点，必须指定两个值。第一个值为结果中希望看到的第一个记录（第一个结果记录的编号为 0 而不是 1）。第二个值为希望看到的记录个数。下面的查询类似于前面那个查询，但只显示从第 11 行开始的 5 个记录：

```
mysql> SELECT last_name, first_name, birth FROM president
-> ORDER BY birth LIMIT 10, 5;
```

last_name	first_name	birth
Tyler	John	1790-03-29
Buchanan	James	1791-04-23
Polk	James K.	1795-11-02
Fillmore	Millard	1800-01-07
Pierce	Franklin	1804-11-23

自 MySQL 3.23.2 以来，可按照一个公式来排序查询结果。例如，利用 ORDER BY RAND() 与 LIMIT 结合，从 president 表中随机抽取一个记录：

```
mysql> SELECT last_name, first_name FROM president
-> ORDER BY RAND() LIMIT 1;
```

```

+-----+-----+
| last_name | first_name |
+-----+-----+
| McKinley  | William    |
+-----+-----+

```

### 5. 计算并命名输出的列值

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

```

mysql> SELECT 17, FORMAT(SQRT(3*3+4*4),0);
+-----+-----+
| 17 | FORMAT(SQRT(3*3+4*4),0) |
+-----+-----+
| 17 | 5                        |
+-----+-----+

```

表达式也可以引用表列：

```

mysql> SELECT CONCAT(first_name, " ",last_name),CONCAT(city," ",state)
-> FROM president;
+-----+-----+
| CONCAT(first_name, " ",last_name) | CONCAT(city," ",state) |
+-----+-----+
| George Washington                | Wakefield, VA         |
| John Adams                       | Braintree, MA         |
| Thomas Jefferson                 | Albemarle County, VA  |
| James Madison                    | Port Conway, VA       |
+-----+-----+
...

```

此查询把名和姓连接起来，中间间隔一个空格，将总统名形成一个单一字符串，而且将出生城市和州连接在一起，中间隔一个逗号，形成出生地。

在利用表达式来计算列值时，此表达式被用作列标题。如果表达式很长（如前面的一些查询样例中那样），那么可能会出现一个很宽的列。为了处理这种情况，此列可利用 AS name 结构来重新命名标题。这样的名称为列别名。用这种方法可使上面的输出更有意义，如下所示：

```

mysql> SELECT CONCAT(first_name," ",last_name) AS Name,
-> CONCAT(city," ",state) As Birthplace
-> FROM president;
+-----+-----+
| Name                                | Birthplace              |
+-----+-----+
| George Washington                 | Wakefield, VA          |
| John Adams                        | Braintree, MA          |
| Thomas Jefferson                  | Albemarle County, VA   |
| James Madison                     | Port Conway, VA        |
+-----+-----+
...

```

如果列的别名包含空格，需要用双引号括起来。

```

mysql> SELECT CONCAT(first_name, " ",last_name) AS "President Name",
-> CONCAT(city," ",state) As "Place of Birth"
-> FROM president;
+-----+-----+
| President Name                    | Place of Birth          |
+-----+-----+
| George Washington                 | Wakefield, VA          |
| John Adams                        | Braintree, MA          |
| Thomas Jefferson                  | Albemarle County, VA   |
+-----+-----+

```

```
| James Madison      | Port Conway, VA |
...
```

## 6. 使用日期

在 MySQL 中使用日期时要记住的是，在表示日期时首先给出年份。1999 年 7 月 27 日表示为“1999-07-27”，而不是像通常那样表示为“07-27-1999”或“27-07-1999”。

MySQL 提供了几种对日期进行处理的方法。可以对日期进行的一些运算如下：

按日期排序。（这点我们已经看到几次了。）

查找特定的日期或日期范围。

提取日期值的组成部分，如年、月或日。

计算日期的差。

日期增加或减去一个间隔得出另一日期。

下面给出一些日期运算的例子。

为了查找特定的日期，可使用精确的日期值或与其他日期值进行比较，将一个 DATE 列与有关的日期值进行比较：

```
mysql> SELECT * FROM event WHERE date = "1999-10-01";
+-----+-----+-----+
| date   | type | event_id |
+-----+-----+-----+
| 1999-10-01 | T   | 6        |
+-----+-----+-----+

mysql> SELECT last_name, first_name, death
-> FROM president
-> WHERE death >= "1970-01-01" AND death < "1980-01-01";
+-----+-----+-----+
| last_name | first_name | death      |
+-----+-----+-----+
| Truman   | Harry S.   | 1972-12-26 |
| Johnson  | Lyndon B.  | 1973-01-22 |
+-----+-----+-----+
```

为了测试或检索日期的成分，可使用诸如 YEAR()、MONTH() 或 DAYOFMONTH() 这样的函数。例如，可通过查找月份值为 3 的日期，找出与笔者出生在相同月份（三月）的总统。

```
mysql> SELECT last_name, first_name, birth
-> FROM president WHERE MONTH(birth) = 3;
+-----+-----+-----+
| last_name | first_name | birth      |
+-----+-----+-----+
| Madison   | James     | 1751-03-16 |
| Jackson   | Andrew    | 1767-03-15 |
| Tyler     | John      | 1790-03-29 |
| Cleveland | Grover    | 1837-03-18 |
+-----+-----+-----+
```

此查询也可以按月的名称写出：

```
mysql> SELECT last_name, first_name, birth
-> FROM president WHERE MONTHNAME(birth) = "March";
+-----+-----+-----+
| last_name | first_name | birth      |
+-----+-----+-----+
| Madison   | James     | 1751-03-16 |
| Jackson   | Andrew    | 1767-03-15 |
| Tyler     | John      | 1790-03-29 |
+-----+-----+-----+
```

```
| Cleveland | Grover      | 1837-03-18 |
+-----+-----+-----+
```

为了更详细，详细到天，可组合测试 MONTH() 和 DAYOFMONTH() 以找出在笔者的生日出生的总统：

```
mysql> SELECT last_name, first_name, birth
-> FROM president WHERE MONTH(birth) = 3 AND DAYOFMONTH(birth) = 29;
+-----+-----+-----+
| last_name | first_name | birth      |
+-----+-----+-----+
| Tyler     | John       | 1790-03-29 |
+-----+-----+-----+
```

这是一种可用来生成类似报纸上娱乐部分所刊登的那种“这些人今天过生日”清单的查询。但是，不必按前面的查询那样插入一个特殊的日期。为了查找每年的今天出生的总统，只要将他们的生日与 CURRENT\_DATE 进行比较即可：

```
SELECT last_name, first_name, birth
FROM president WHERE MONTH(birth) = MONTH(CURRENT_DATE)
AND DAYOFMONTH(birth) = DAYOFMONTH(CURRENT_DATE)
```

可从一个日期减去另一个日期。这样可以知道日期期间的间隔，这对于确定年龄是非常有用的。例如，为了确定哪位总统活得最长，可将其逝世日期减去出生日期。为此，可利用函数 TO\_DAYS() 将出生日期和逝世日期转换为天数，求出差，然后除以 365 得出大概的年龄：

```
mysql> SELECT last_name, first_name, birth, death,
-> FLOOR((TO_DAYS(death) - TO_DAYS(birth))/365) AS age
-> FROM president WHERE death IS NOT NULL
-> ORDER BY age DESC LIMIT 5;
+-----+-----+-----+-----+-----+
| last_name | first_name | birth      | death      | age |
+-----+-----+-----+-----+-----+
| Adams     | John       | 1735-10-30 | 1826-07-04 | 90  |
| Hoover    | Herbert C. | 1874-08-10 | 1964-10-20 | 90  |
| Truman    | Harry S.   | 1884-05-08 | 1972-12-26 | 88  |
| Madison   | James      | 1751-03-16 | 1836-06-28 | 85  |
| Jefferson | Thomas     | 1743-04-13 | 1826-07-04 | 83  |
+-----+-----+-----+-----+-----+
```

此查询中所用的 FLOOR() 函数截掉了年龄的小数部分，得到一个整数。

得出日期之差，还可以确定相对于某个特定日期有多长时间。这样可以告诉历史同盟的会员，他们还有多久就应该更新自己的会员资格了。计算他们的截止日期和当前日期之差，如果小于某个阈值，则不久就需要更新了。下面的查询是查找需要在 60 天内更新的会员：

```
SELECT last_name, first_name, expiration FROM member
WHERE (TO_DAYS(expiration) - TO_DAYS(CURRENT_DATE)) < 60
```

自 MySQL 3.22 以来，可使用 DATE\_ADD() 或 DATE\_SUB() 从一个日期计算另一个日期。这些函数取一个日期及时间间隔并产生一个新日期。例如：

```
mysql> SELECT DATE_ADD("1970-1-1", INTERVAL 10 YEAR);
+-----+
| DATE_ADD("1970-1-1", INTERVAL 10 YEAR) |
+-----+
| 1980-01-01                               |
+-----+
mysql> SELECT DATE_SUB("1970-1-1", INTERVAL 10 YEAR);
+-----+
| DATE_SUB("1970-1-1", INTERVAL 10 YEAR) |
+-----+
| 1960-01-01                               |
+-----+
```

本节中前面给出的一个查询选择 70 年代逝世的总统，它对选择范围的端点使用直接的日期值。该查询可以利用一个字符串日期和一个由开始日期和时间间隔计算出的结束日期来重写：

```
mysql> SELECT last_name, first_name, death
-> FROM president
-> WHERE death >= '1970-1-1'
-> AND death < DATE_ADD('1970-1-1', INTERVAL 10 YEAR);
```

last_name	first_name	death
Truman	Harry S.	1972-12-26
Johnson	Lyndon B.	1973-01-22

会员更新查询可根据 DATE\_ADD() 写出如下：

```
SELECT last_name, first_name, expiration FROM member
WHERE expiration < DATE_ADD(CURRENT_DATE, INTERVAL 60 DAY)
```

本章前面给出了一个查询如下，确定不久要来检查但还没来诊所的牙科病人：

```
SELECT last_name, first_name, last_visit FROM patient
WHERE last_visit < DATE_SUB(CURRENT_DATE, INTERVAL 6 MONTH)
```

现在回过头来看，读者会更清楚这个查询的含义了。

## 7. 模式匹配

MySQL 允许查找与某个模式相配的值。这样，可以选择记录而不用提供精确的值。为了进行模式匹配运算，可使用特殊的运算符（LIKE 和 NOT LIKE），并且指定一个包含通配符的串。字符“\_”匹配任意单个字符，而“%”匹配任意字符序列（包括空序列）。使用 LIKE 或 NOT LIKE 的模式匹配都是不区分大小写的。

下列模式匹配以“W”或“w”开始的姓：

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name LIKE "W%";
```

last_name	first_name
Washington	George
Wilson	Woodrow

下列模式匹配是错误的：

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name = "W%";
Empty set (0.00 sec)
```

此查询给出了一个常见的错误，它对一个算术比较运算符使用了模式。这种比较成功的惟一可能是相应的列确实包含串“W%”或“w%”。

下列模式匹配任意位置包含“W”或“w”的姓：

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name LIKE "%W%";
```

last_name	first_name
Washington	George
Wilson	Woodrow



```
| Eisenhower | Dwight D. |
+-----+-----+
```

下列模式匹配只含有四个字符的姓：

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name LIKE "____";
```

MySQL 还提供基于扩展正规表达式的模式匹配。正规表达式在附录 C 的 REGEXP 运算符的介绍中描述。

## 8. 生成汇总

MySQL 所能做的最有用的事情是浓缩大量的原始数据行并对其进行汇总。当学会了利用 MySQL 来生成汇总时，它就变成了用户强有力的好帮手了，因为手工进行汇总是一项冗长的、费时的、易出错的工作。

汇总的一种简单的形式是确定在一组值中哪些值是唯一值。利用 DISTINCT 关键字来删除结果中的重复行。例如，总统出生的各个州可按如下找出：

```
mysql> SELECT DISTINCT state FROM president ORDER BY state;
```

```
+-----+
| state |
+-----+
| AK    |
| CA    |
| GA    |
| IA    |
| IL    |
| KY    |
| MA    |
| MO    |
| NC    |
| NE    |
| NH    |
| NJ    |
| NY    |
| OH    |
| PA    |
| SC    |
| TX    |
| VA    |
| VT    |
+-----+
```

其他的汇总形式涉及计数，可利用 COUNT() 函数。如果使用 COUNT(\*)，它将给出查询所选择的行数。如果一个查询无 WHERE 子句，COUNT(\*) 将给出表中的行数。

下列查询给出共有多少人当过美国总统：

```
mysql> SELECT COUNT(*) FROM president;
```

```
+-----+
| COUNT(*) |
+-----+
|         41 |
+-----+
```

如果查询有 WHERE 子句，COUNT(\*) 将给出此子句选择多少行。下面的查询给出目前为止对班级进行了多少次测试：

```
mysql> SELECT COUNT(*) FROM event WHERE type = 'Q';
```

```
+-----+
| COUNT(*) |
```

```
+-----+
|      4 |
+-----+
```

COUNT(\*) 对选中的行进行计数。而 COUNT(col\_name) 只对非 NULL 值进行计数。下面的查询说明了这些差异：

```
mysql> SELECT COUNT(*),COUNT(suffix),COUNT(death) FROM president;
+-----+-----+-----+
| COUNT(*) | COUNT(suffix) | COUNT(death) |
+-----+-----+-----+
|      41 |             1 |           36 |
+-----+-----+-----+
```

这表示，总共有 41 位总统，他们中只有一个具有名字后缀，并且大多数总统都已去世。

自 MySQL 3.23.2 以来，可以将 COUNT() 与 DISTINCT 组合对选择结果集中不同的值进行计数。例如，为了对总统出生的不同州进行计数，可执行下列查询：

```
mysql> SELECT COUNT(DISTINCT state) FROM president;
+-----+
| COUNT(DISTINCT state) |
+-----+
|             19 |
+-----+
```

可以根据汇总列中单独的值对计数值进行分解。例如，您可能根据下列的查询结果知道班级中所有学生的人数：

```
mysql> SELECT COUNT(*) FROM student;
+-----+
| COUNT(*) |
+-----+
|      31 |
+-----+
```

但是，有多少是男孩？有多少是女孩？分别得出男孩、女孩的一种方法是分别对每种性别进行计数：

```
mysql> SELECT COUNT(*) FROM student WHERE sex='f';
+-----+
| COUNT(*) |
+-----+
|      15 |
+-----+

mysql> SELECT COUNT(*) FROM student WHERE sex='m';
+-----+
| COUNT(*) |
+-----+
|      16 |
+-----+
```

虽然这个方法可行，但是它很繁琐而且并不真正适合于可能有许多不同的值的列。考虑一下怎样以这种方式确定每个州出生的总统人数。您不得不找出有哪些州，从而不能省略 (SELECT DISTINCT state FROM president)，然后对每个州执行一个 SELECT COUNT(\*) 查询。很显然，有些事是可以简化的。

所幸MySQL 可以利用单个查询对一个列中不同的值进行计数。因此，针对学生表可以按如下得出男孩和女孩的人数：

```
mysql> SELECT sex, COUNT(*) FROM student GROUP BY sex;
+-----+-----+
| sex | COUNT(*) |
+-----+-----+
```

M	16
F	15

用同样形式的查询得出每个州出生的总统有多少：

```
mysql> SELECT state, COUNT(*) FROM president GROUP BY state;
```

state	COUNT(*)
AK	1
CA	1
GA	1
IA	1
IL	1
KY	1
MA	4
MO	1
NC	2
NE	1
NH	1
NJ	1
NY	4
OH	7
PA	1
SC	1
TX	2
VA	8
VT	2

如果以这种方法对值计数，GROUP BY 子句是必须的；它告诉 MySQL 在对值计数之前怎样进行聚集。如果将其省去，则要出错。

COUNT(\*) 与 GROUP BY 一起来对值进行计数比分别对每个不同的列值进行计数有更多的优点，这些优点是：

不必事先知道要汇总的列中有什么值。

不用编写多个查询，只需编写单个查询即可。

用单一查询就可以得出所有结果，因此可以对结果进行排序。

前两个优点对于更方便地表示查询很重要。第三个优点也较为重要，因为它提供了显示结果的灵活性。在使用 GROUP BY 子句时，其结果是在要分组的列上进行排序的，但是可以使用 ORDER BY 来按不同的次序进行排序。例如，如果想得到各州产生的总统人数，并按产生人数最多的州优先排出，可以如下使用 ORDER BY 子句：

```
mysql> SELECT state, COUNT(*) AS count FROM president
-> GROUP BY state ORDER BY count DESC;
```

state	count
VA	8
OH	7
MA	4
NY	4
NC	2
VT	2
TX	2
SC	1
NH	1

PA	1
KY	1
NJ	1
IA	1
MO	1
CA	1
NE	1
GA	1
IL	1
AK	1

如果希望进行排序的列是从计算得出的，则可以给该列一个别名，并在 ORDER BY 子句中引用这个别名。前面的查询说明了这一点；COUNT(\*) 列的别名为 count。引用这样的列的另一种方法是引用它在输出结果中的位置。前面的查询可编写如下：

```
SELECT state, COUNT(*) FROM president
GROUP BY state ORDER BY 2 DESC
```

我不认为按位置引用列易读。如果增加、删除或重新排序输出列，必须注意检查 ORDER BY 子句，并且如果列号改变后还得记住它。别名就不存在这种问题。

如果想与计算出来的列一道使用 GROUP BY，正如 ORDER BY 一样，应该利用别名或列位置来引用它。下面的查询确定在一年的每个月中出生的总统人数：

```
mysql> SELECT MONTH(birth) as Month, MONTHNAME(birth) as Name,
-> COUNT(*) AS count
-> FROM president GROUP BY Name ORDER BY Month;
```

Month	Name	count
1	January	4
2	February	4
3	March	4
4	April	4
5	May	2
6	June	1
7	July	3
8	August	4
9	September	1
10	October	6
11	November	5
12	December	3

利用列位置，此查询可编写如下：

```
SELECT MONTH(birth), MONTHNAME(birth), COUNT(*)
FROM president GROUP BY 2 ORDER BY 1;
```

例如，COUNT() 可与 ORDER BY 和 LIMIT 组合来查找 president 表中 4 个产生总统最多的州：

```
mysql> SELECT state, COUNT(*) AS count FROM president
-> GROUP BY state ORDER BY count DESC LIMIT 4;
```

state	count
VA	8
OH	7
MA	4
NY	4

如果不想用 LIMIT 子句来限制查询输出，而是利用查找特定的 COUNT() 值来达到这个目的，可使用 HAVING 子句。下面的查询给出了产生两个以上总统的州：

```
mysql> SELECT state, COUNT(*) AS count FROM president
-> GROUP BY state HAVING count > 1 ORDER BY count DESC;
```

state	count
VA	8
OH	7
MA	4
NY	4
NC	2
VT	2
TX	2

从更为普遍意义上说，这是一种在要查找的列中重复值时执行的查询类型。

HAVING 类似于 WHERE，但它是在查询结果已经选出后才应用的，用来缩减服务器实际送到客户机的结果。

除了 COUNT() 外还有许多汇总函数。MIN()、MAX()、SUM() 和 AVG() 函数在确定列的最大、最小、总数和平均值时都非常有用，甚至可以同时使用它们。下面的查询得出给定的测试和测验的各种数字特性。它还给出有多少学分参与了每个值的计算（有的学生可能缺旷或未计入）。

```
mysql> SELECT
-> event_id,
-> MIN(score) AS minimum,
-> MAX(score) AS maximum,
-> MAX(score)-MIN(score)+1 AS range,
-> SUM(score) AS total,
-> AVG(score) AS average,
-> COUNT(score) AS count
-> FROM score
-> GROUP BY event_id;
```

event_id	minimum	maximum	range	total	average	count
1	9	20	12	439	15.1379	29
2	8	19	12	425	14.1667	30
3	60	97	38	2425	78.2258	31
4	7	20	14	379	14.0370	27
5	8	20	13	383	14.1852	27
6	62	100	39	2325	80.1724	29

当然，如果您知道这些信息是来自测验的还是测试的，则它们就会更有意义。但是，为了产生那样的信息，还需要参考 event 表；我们将在下一节“从多个表中检索信息”讨论这个查询。

汇总信息是很有意思的，因为它们是那么有用，但不太好控制，容易走样。请看下列查询：

```
mysql> SELECT
-> state AS State,
-> AVG((TO_DAYS(death)-TO_DAYS(birth))/365) AS Age
-> FROM president WHERE death IS NOT NULL
-> GROUP BY state ORDER BY Age;
```



State	Age
KY	56.208219
VT	58.852055
NC	60.141096
OH	62.866145
NH	64.917808
NY	69.342466
NJ	71.315068
TX	71.476712
MA	72.642009
VA	72.822945
PA	77.158904
SC	78.284932
CA	81.336986
MO	88.693151
IA	90.254795

此查询选择已经去世的总统，按出生地对他们进行分组，并计算出他们逝世时的年龄，计算出平均年龄（每个州的），然后按平均年龄进行排序。换句话说，此查询按所出生地确定已故总统的平均寿命。

但这说明了什么呢？它仅仅说明您可写该查询，当然并不说明此查询是否值得写。并不是用一个数据库可以做的所有事情都同样有意义；但是，人们有时在发现可以利用自己的数据库进行查询时感到很开心。这可能说明关于转播运动会的不断增加的深奥的（空洞的）统计数据在过去几年里正在不断增多的原因。运动统计者可以使用他们的数据库来计算出某个队的历史纪录，而这些数字你可能感兴趣，也可能毫无兴致。

#### 9. 从多个表中检索信息

到目前为止，我们所编写的查询都是从单个表中得到数据的。现在，我们将进行一件更为有趣的工作。以前笔者曾经提到过，关系 DBMS 的强大功能在于它能够将一样东西与另一样东西相关联，因为这样使得能够结合多个表中的信息来解答单个表不能解答的问题。本节介绍怎样编写这种查询。

在从多个表中选择信息时，需要执行一种称为连接（join）的操作。这是因为需要将一个表中的信息与其他表中的信息相连接来得出查询结果。即通过协调各表中的值来完成这项工作。

我们来研究一个例子。在前面的“学分保存方案”小节中，给出了一个检索特定日期的测验或测试学分的查询，但没有解释。现在可以进行解释了。这个查询实际涉及到三种连接方法，因此我们分两步进行研究。

第一步，我们构造一个对特定日期的学分进行选择查询，如下所示：

```
mysql> SELECT student_id, date, score, type
-> FROM event, score
-> WHERE date = "1999-09-23"
-> AND event.event_id = score.event_id;
```

student_id	date	score	type
1	1999-09-23	15	Q
2	1999-09-23	12	Q
3	1999-09-23	11	Q

```

5 | 1999-09-23 | 13 | Q |
6 | 1999-09-23 | 18 | Q |
...

```

此查询找出具有给定日期的记录，然后利用该记录中的事件 ID 查找具有相同事件 ID 的学分。对于每个匹配的事件记录和学分记录组合，显示学生 ID、学分、日期和事件类型。

此查询在两个重要方面不同于我们曾经编写过的其他查询。它们是：

FROM 子句给出了不止一个表名，因为我们要检索的数据来自不止一个表：

```
FROM event, score
```

WHERE 子句说明 event 和 score 表是由每个表中的 event\_id 值的匹配连接起来的：

```
WHERE ... event.event_id = score.event_id
```

请注意，我们是怎样利用 tbl\_name.col\_name 语法引用列，以便 MySQL 知道引用的是哪些表的列。（event\_id 出现在两个表中，如果不用表名来限定它的话将会出现混淆。）

此查询中的其他列（date、score、type）可单独使用而不用表名限定符，因为它们在表中只出现一次，从而不会出现含混。但是，一般在连接中我们对每个列都进行限定以便清晰地表示出每个列是属于哪个表。在完全限定的形式下，查询如下：

```

SELECT score.student_id, event.date, score.score, event.type
FROM event, score
WHERE event.date = "1999-09-23"
AND event.event_id = score.event_id

```

从现在起，我们将使用完全限定的形式。

第二步，我们利用 student 表完成查询以便显示学生名。（第一步中查询的输出给出了 student\_id 字段，但是名字更有意义。）名字显示是利用 score 表和 student 表两者都具有 student\_id 列，使它们中的记录可被连接这个事实来完成的。最终的查询如下：

```

mysql> SELECT student.name, event.date, score.score, event.type
-> FROM event, score, student
-> WHERE event.date = "1999-09-23"
-> AND event.event_id = score.event_id
-> AND score.student_id = student.student_id;

```

name	date	score	type
Megan	1999-09-23	15	Q
Joseph	1999-09-23	12	Q
Kyle	1999-09-23	11	Q
Abby	1999-09-23	13	Q
Nathan	1999-09-23	18	Q

此查询与前一个查询的差别在于：

student 表被增加到了 FROM 子句中，因为除了 event 表和 score 表外还用到了它。

student\_id 列现在不明确了（因为现在有两个引用到的表都含有此列），因此必须限定为 score.student\_id 或 student.student\_id 以表明使用的是哪个表。

WHERE 子句有一个附加项，它说明根据学生 ID 将 score 表记录与 student 表记录进行匹配。

此查询是显示学生名而不是学生 ID。（当然，如果愿意的话，可以两者都显示。）

利用此查询，可以加入任意日期，得到该日期的学分，用学生名和学分类型完善查询结果。不一定要了解关于学生 ID 或事件 ID 的情况。MySQL 小心地得出相关的 ID 值并利用它

们自动地使各表的行相配。

学分保存方案涉及的另一项工作是汇总学生的缺勤情况。缺勤情况是按学生 ID 和日期在 absence 表中记录的。为得到学生名（而不仅仅是 ID），我们需要根据 student\_id 的值将 absence 表连接到 student 表。下面的查询给出了学生的 ID 号和名字以及缺勤计数：

```
mysql> SELECT student.student_id, student.name,
-> COUNT(absence.date) as absences
-> FROM student, absence
-> WHERE student.student_id = absence.student_id
-> GROUP BY student.student_id;
```

student_id	name	absences
3	Kyle	1
5	Abby	1
10	Peter	2
17	Will	1
20	Avery	1

注意：虽然我们在 GROUP BY 子句中应用了一个限定符，但对于这个查询来说不是必须的。因为 GROUP BY 子句只引用选择表中（此查询的前两行）的列。在该处只有一个名为 student\_id 的列，因此 MySQL 知道应该用哪个列。这个规则对 ORDER BY 子句也成立。

如果我们希望只了解哪些学生缺过勤，则此查询所产生的输出也是有用的。但是，如果我们将此清单交给学校办公室，他们可能会说，“其他的学生呢？我们需要每个学生的情况。”这是一个稍微有点不同的问题。它表示需要知道学生的缺勤数，即使没有缺勤的学生也需要知道。因为问题的不同，查询也应该不同。

为了解决上述问题，使用 LEFT JOIN 而不涉及 WHERE 子句中的学生 ID。LEFT JOIN 要求 MySQL 对从连接首先给出的表中选择每行生成一个输出行（即 LEFT JOIN 关键字左边给出的表）。由于首先给出 student 表，我们得到了每个学生的输出结果，即使是那些在 absence 表中未给出的学生也都包括在输出中。此查询如下：

```
mysql> SELECT student.student_id, student.name,
-> COUNT(absence.date) as absences
-> FROM student LEFT JOIN absence
-> ON student.student_id = absence.student_id
-> GROUP BY student.student_id;
```

student_id	name	absences
1	Megan	0
2	Joseph	0
3	Kyle	1
4	Katie	0
5	Abby	1
6	Nathan	0
7	Liesl	0

前面，在“生成汇总”一节中，我们执行了一个查询，它生成 score 表中数据的数值特征。该查询的输出列出了事件 ID，但不包括学分日期或类型，因为我们不知道怎样将 score 表连接到 event 表以得到学分的日期和类型。现在可以做到了。下面的查询类似于早先的那个，但是它给出了学分的日期和类型而不只是简单的数字事件 ID：

```
mysql> SELECT
-> event.date,event.type,
-> MIN(score.score) AS minimum,
-> MAX(score.score) AS maximum,
-> MAX(score.score)-MIN(score.score)+1 AS range,
-> SUM(score.score) AS total,
-> AVG(score.score) AS average,
-> COUNT(score.score) AS count
-> FROM score, event
-> WHERE score.event_id = event.event_id
-> GROUP BY event.date;
```

date	type	minimum	maximum	range	total	average	count
1999-09-03	Q	9	20	12	439	15.1379	29
1999-09-06	Q	8	19	12	425	14.1667	30
1999-09-09	T	60	97	38	2425	78.2258	31
1999-09-16	Q	7	20	14	379	14.0370	27
1999-09-23	Q	8	20	13	383	14.1852	27
1999-10-01	T	62	100	39	2325	80.1724	29

可利用诸如 COUNT() 和 AVG() 这样的函数生成多个列上的汇总，即使这些列来自不同的表也是如此。下面的查询确定学分数，以及事件日期与学生性别的每种组合的平均学分。

```
mysql> SELECT event.date, student.sex,
-> COUNT(score) AS count, AVG(score) As average
-> FROM event, score, student
-> WHERE event.event_id = score.event_id
-> AND score.student_id = student.student_id
-> GROUP BY event.date, student.sex;
```

date	sex	count	average
1999-09-03	F	14	14.6429
1999-09-03	M	15	15.6000
1999-09-06	F	14	14.7143
1999-09-06	M	16	13.6875
1999-09-09	F	15	77.4000
1999-09-09	M	16	79.0000
1999-09-16	F	13	15.3077
1999-09-16	M	14	12.8571
1999-09-23	F	12	14.0833
1999-09-23	M	15	14.2667
1999-10-01	F	14	77.7857
1999-10-01	M	15	82.4000

我们可以使用一个类似的查询来完成学分保存方案的一个任务，即在学期末计算每个学生的总学分。相应的查询如下：

```
SELECT student.student_id, student.name,
SUM(score.score) AS total, COUNT(score.score) AS n
FROM event, score, student
WHERE event.event_id = score.event_id
AND score.student_id = student.student_id
GROUP BY score.student_id
ORDER BY total
```

不一定要连接必须用两个不同的表来完成。这似乎有点奇怪，但是确实可以将一个表连接到其自身。例如，可通过针对每个总统的出生地查看其他各个总统的出生地，确定几个

总统是否出生在相同城市。此查询如下：

```
mysql> SELECT p1.last_name, p1.first_name, p1.city, p1.state
-> FROM president AS p1, president AS p2
-> WHERE p1.city = p2.city AND p1.state = p2.state
-> AND (p1.last_name != p2.last_name OR p1.first_name != p2.first_name)
-> ORDER BY state, city, last_name;
```

last_name	first_name	city	state
Adams	John Quincy	Braintree	MA
Adams	John	Braintree	MA

此查询有两个技巧性的东西：

我们需要使用同一表的两个实例，因此建立了表的别名（p1、p2），并利用它们无歧义地引用表列。

每个总统的记录与自身相匹配，但是我们不希望在输出中看到同一总统出再现两次。

WHERE 子句的第二行保证比较的记录为不同总统的记录，使记录不与自身匹配。

可以编写一个查找出生在同一天总统的类似查询。出生日期不能直接比较，因为那样会错过出生在不同年份的总统。我们用 MONTH() 和 DAYOFMONTH() 来比较出生日期的月和日，相应的查询如下：

```
mysql> SELECT p1.last_name, p1.first_name, p1.birth
-> FROM president AS p1, president AS p2
-> WHERE MONTH(p1.birth) = MONTH(p2.birth)
-> AND DAYOFMONTH(p1.birth) = DAYOFMONTH(p2.birth)
-> AND (p1.last_name != p2.last_name OR p1.first_name != p2.first_name)
-> ORDER BY p1.last_name;
```

last_name	first_name	birth
Harding	Warren G.	1865-11-02
Polk	James K.	1795-11-02

利用 DAYOFYEAR() 而不是 MONTH() 和 DAYOFMONTH() 将得出一个更为简单的查询，但是在比较闰年日期与非闰年日期时将会得出不正确的结果。

迄今所执行的连接结合了来自那些在某种意义上具有逻辑关系的表中的信息，但是只有您知道该关系无意义。MySQL 并不知道（或不关心）所连接的表相互之间是否相关。例如，可将 event 表连接到 president 表以找出在某个总统生日那天是否进行了测验或测试，此查询如下：

```
mysql> SELECT president.last_name, president.first_name,
-> president.birth, event.type
-> FROM president, event
-> WHERE MONTH(president.birth) = MONTH(event.date)
-> AND DAYOFMONTH(president.birth) = DAYOFMONTH(event.date);
```

last_name	first_name	birth	type
Carter	James E.	1924-10-01	T

它产生了您所想要的东西。但说明了什么呢？这说明 MySQL 将愉快地制造出结果，至于这些结果是否有意义它不管。这是因为您使用的是计算机，所以它不能自动地判断查询的



结果有用或无用。无论如何，我们都必须为自己所做的事负责。

#### 1.4.9 删除或更新现有记录

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

DELETE 语句有如下格式：

```
DELETE FROM tbl_name WHERE 要删除的记录
```

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

```
DELETE FROM tbl_name
```

这个查询将清除表中的所有内容。一定要当心！

为了删除特定的记录，可用 WHERE 子句来选择所要删除的记录。这类似于 SELECT 语句中的 WHERE 子句。例如，为了删除 president 表中所有出生在 Ohio 的总统记录，可用下列查询：

```
mysql> DELETE FROM president WHERE state="OH";
Query OK, 7 rows affected (0.00 sec)
```

DELETE 语句中的 WHERE 子句的一个限制是只能够引用要删除记录的表中的列。

在发布 DELETE 语句以前，最好用 SELECT 语句测试一下相应的 WHERE 子句以确保实际删除的记录就是确实想要删除的记录（而且只删除这些记录）。假如想要删除 Teddy Roosevelt 的记录。下面的查询能完成这项工作吗？

```
DELETE FROM president WHERE last_name="Roosevelt"
```

是的，感觉上它能删除您头脑中打算删除的记录。但是，错了，实际上它也能删除 Franklin Roosevelt 的记录。如果首先用 WHERE 子句检查一下就安全了，如下所示：

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name="Roosevelt";
+-----+-----+
| last_name | first_name |
+-----+-----+
| Roosevelt | Theodore   |
| Roosevelt | Franklin D. |
+-----+-----+
```

从中可以看到条件还应该更特殊一些：

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name="Roosevelt" AND first_name="Theodore";
+-----+-----+
| last_name | first_name |
+-----+-----+
| Roosevelt | Theodore   |
+-----+-----+
```

现在我们明白了能选择出所需记录的 WHERE 子句了，因此 DELETE 查询可正确地构造如下：

```
mysql> DELETE FROM president
-> WHERE last_name="Roosevelt" AND first_name="Theodore";
```

似乎删除一个记录需要做许多工作，不是吗？但是安全第一！（如果想使键盘输入工作尽量少，可利用拷贝和粘贴技术或采用输入行编辑技术。更详细的信息，请参阅“与 mysql

交互的技巧”一节。)

为了修改现有记录,可利用 UPDATE 语句,它具有下列格式:

```
UPDATE tbl_name SET 要更改的列  
WHERE 要更新的记录
```

这里的 WHERE 子句正如 DELETE 语句一样,是可选的,因此如果不指定的话,表中的每个记录都被更新。下面的查询将每个学生的名字都更改为“George”:

```
mysql> UPDATE student SET name="George";
```

显然,对于这样的查询必须极为小心。

一般对正在更新的记录要更为小心。假定近来增加了一个新记录到历史同盟,但是只填写了此实体的少数几个列:

```
mysql> INSERT member (last_name,first_name)  
-> VALUES('York','Jerome');
```

然后意识到忘了设置其会员终止日期。那么可如下进行设置:

```
mysql> UPDATE member  
-> SET expiration='2001-7-20'  
-> WHERE last_name='York' AND first_name='Jerome';
```

可同时更新多个列。下面的语句将更新 Jerome 的电子邮件和通信地址:

```
mysql> UPDATE member  
-> SET email='jeromey@aol.com',street='123 Elm St',city='Anytown',  
-> state='NY',zip='01003'  
-> WHERE last_name='York' AND first_name='Jerome';
```

还可以通过设置某列的值为 NULL (假设此列允许 NULL 值)“不设置”此列。如果在未来的某个时候 Jerome 决定支付成为终生会员的会员资格更新费,那么可以设置其记录的终止日期为 NULL (“永久”)以标记他为终生会员。具体设置如下:

```
mysql> UPDATE member  
-> SET expiration=NULL  
-> WHERE last_name='York' AND first_name='Jerome';
```

正如 DELETE 语句一样,对于 UPDATE,用 SELECT 语句测试 WHERE 子句以确保选择正确的更新记录是一个好办法。如果选择条件范围太窄或太宽,就会使更新的记录太少或太多。

如果您试验过本节中的查询,那么必定已经删除和修改了 samp\_db 表中的记录。在继续学习下一节的内容以前,应该撤消这些更改。按 1.4.7 节“增加新记录”最后的说明重新装载表的内容来完成这项工作。

#### 1.4.10 改变表的结构

回顾我们创建历史同盟 member 表时缺了一个会员号列,因此我们可以进行一次 ALTER TABLE 语句的练习。需要用 ALTER TABLE,可以对表重新命名,增加或删除列,更改列的类型等等。这里给出的例子是关于怎样增加新列的。有关 ALTER TABLE 功能的详细内容,请参阅第3章。

增加会员号列到 member 表的主要考虑是,其值应该是唯一的,以免各会员条目混淆。AUTO\_INCREMENT 列在此是很有用的,因为我们可以增加新的号码时令 MySQL 自动地生成唯一的号码。在 CREATE TABLE 语句中,这样一个列的说明如下:

```
member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
```

对于 ALTER TABLE，相应的句法也是类似的。可执行下列查询增加该列：

```
mysql> ALTER TABLE member  
-> ADD member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY;
```

我们已经有一个存放会员号的列，现在怎样分配会员号给 member 表中的现有记录呢？很容易！MySQL 已经做了这项工作。在增加一列到某个表时，MySQL 将会用缺省值初始化该列值。对于 AUTO\_INCREMENT 列，每个行将会产生一个新的顺序号。

## 1.5 与 mysql 交互的技巧

本节介绍怎样更有效地且键入工作量较小地与 mysql 客户机程序进行交互。介绍怎样更简单地与服务器连接，以及怎样不用每次都从头开始键入查询。

### 1.5.1 简化连接过程

在激活 mysql 时，有可能需要指定诸如主机名、用户名或口令这样的连接参数。运行一个程序需要做很多输入工作，这很快就会让人厌烦。有几种方法可最小化所做的键入工作，使连接更为容易，它们分别为：

- 利用选项文件存储连接参数。

- 利用外壳程序的命令历史重复命令。

- 利用外壳程序的别名或脚本定义 mysql 命令行快捷键。

#### 1. 利用选项文件

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

为了利用选项文件方法指定连接参数，可建立一个名为 ~/.my.cnf（即主目录中的一个名为 .my.cnf 的文件）。选项文件是一个无格式的文本文件，因此可用任何文本编辑器来创建它。文件的内容如下所示：

```
[client]  
host=serverhost  
user=yourname  
password=yourpass
```

[client] 行标记客户机选项组的开始；它后跟的所有行都是为 MySQL 客户机程序获得选项值准备的，这些行一直延续到文件的结尾或另一不同的参数组的开始。在连接到服务器时，用指定的主机名、用户名和口令替换 serverhost、yourname 和 yourpass。对于笔者来说，.my.cnf 如下所示：

```
[client]  
host=pit-viper.snake.net  
user=paul  
password=secret
```

只有 [client] 行是必须的。定义参数值的行都是可选的；可以仅指定那些所需要的参数。例如，如果您的 MySQL 用户名与 UNIX 的登录名相同，则不需要包括 user 行。

在创建了 .my.cnf 文件后，设置其访问方式为某个限定值以保证别人不能读取它：

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

在 Windows 下，选项文件的内容是相同的，但其名称不同（c:\my.cnf），而且不调用 chmod 命令。

因为选项文件在版本 3.22 前未加到 MySQL，所以更早的版本不能使用它们。特别是在 Windows 下，您不能与共享 MySQL 分发包一起得到的客户机使用选项文件，因为它是基于 MySQL 3.21 的。选项文件在注册过的 MySQL 的 Windows 版本下工作得很好，否则可以从 MySQL Web 站点取得更新的支持选项文件的客户机。

关于选项文件的详细内容可参阅附录 E “MySQL 程序参考”。

## 2. 利用外壳程序的命令历史

诸如 csh、tcsh 和 bash 这样的外壳程序会在一个历史列表中记下您的命令，并允许重复该列表中的命令。如果采用的是这样的外壳程序，其历史列表可帮助免除完整命令的键入。例如，如果最近调用了 mysql，可按如下命令再次执行它：

```
% !my
```

其中“！”告诉外壳程序搜索整个命令历史找到最近以“my”开头的命令，并像您打入的一样发布它。有的外壳程序还允许利用上箭头和下箭头键（或许是 Ctrl-P 和 Ctrl-N）在历史列表中上下移动。可用这种方法选择想要的命令，然后按 Enter 执行它。tcsh 和 bash 有这种功能，而其他外壳程序也可能有。可参阅相应的外壳程序以找到更多使用历史列表的内容。

## 3. 利用外壳程序的别名或脚本

如果使用的外壳程序提供别名功能，那么可以设置允许通过键入简短名调用长命令的命令快捷键。例如，在 csh 或 tcsh 中，可利用 alias 命令设置名为 samp\_db 的别名，如下所示：

```
alias samp_db 'mysql -h pit-viper.snake.net -u paul -p samp_db'
```

而 bash 中的语法稍有不同：

```
alias samp_db='mysql -h pit-viper.snake.net -u paul -p samp_db'
```

可以定义一个别名使这两个命令等价：

```
samp_db  
mysql -h pit-viper.snake.net -u paul -p samp_db
```

显然，第一个比第二个更好键入。为了使这些别名在每次登录时都起作用，可将在外壳程序设置文件中放入一个 alias 命令（如，csh 放入 .cshrc，而 bash 放入 .bash\_profile）。

快捷键的其他形式是建立利用适当的选项执行 mysql 的外壳程序脚本。在 UNIX 中，等价于 samp\_db 别名的脚本文件如下所示：

```
#!/bin/sh  
exec mysql -h pit-viper.snake.net -u paul -p samp_db
```

如果笔者命名此脚本为 samp\_db 并使其可执行（用 chmod +x samp\_db），那么可以键入 samp\_db 运行 mysql 并连接到笔者的数据库中。

在 Windows 下，可用批命令文件来完成相同的工作。命名文件 samp\_db.bat，并在其中放入如下的行：

```
mysql -h pit-viper.snake.net -u paul -p samp_db
```

此批命令文件可通过在 DOS 控制台提示符下键入 samp\_db 来执行，也可以双击它的 Windows 图标来执行。

如果访问多个数据库或连接到多个主机，则可以定义几个别名或脚本，每一个都用不同

的选项调用 `mysql`。

### 1.5.2 以较少的键入发布查询

`mysql` 是一个与数据库进行交互的极为有用的程序，但是其界面最适合于简短的、单行的查询。当然，`mysql` 自身并不关心某个查询是否分成多行，但是长的查询很不好键入。输入一条查询也不是很有趣的事，即使是一条较短的查询也是如此，除非发现有错误才愿意重新键入它。

有几种可用来避免不必要的键入或重新键入的技巧：

利用 `mysql` 的输入行编辑功能。

利用拷贝和粘贴。

以批方式运行 `mysql`。

利用现有数据来创建新记录以避免键入 `INSERT` 语句。

#### 1. 利用 `mysql` 的输入行编辑器

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

表1-4 中列出了一些非常有用的编辑序列，除了此表中给出的以外，还有许多输入编辑命令。利用因特网搜索引擎，应该能够找到 Readline手册的联机版本。此手册也包含在 Readline 分发包中，可在 <http://www.gnu.org/> 的 GNU Web 站点得到。

表1-4 `mysql` 输入编辑命令

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

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

```
mysql> SHOW COLUMNS FROM persident;
```

如果在按 `Enter` 前，已经注意到将 “`president`” 错拼成了 “`persident`”，则可按左箭头或 `Ctrl-B` 多次移动光标到 “`s`” 的左边。然后按 `Delete` 两次删除 “`er`”，键入 “`re`” 改正错误，并按 `Enter` 发布此查询。如果没注意到错拼就按了 `Enter`，也不会有问题。在 `mysql` 显示了错误

消息后，按上箭头或 Ctrl-P 调出该行，然后对其进行编辑。

输入行编辑在 mysql 的 Windows 版中不起作用，但是可从 MySQL Web 站点取得免费的 cygwin\_32 客户机分发版。在该分发版中的 mysqlc 程序与 mysql 一样，但它支持输入行编辑命令。

## 2. 利用拷贝和粘贴发布查询

如果是在窗口环境下工作，可将认为有用的查询文本保存在一个文件中并利用拷贝和粘贴操作很容易地发布这些命令。其工作过程如下：

1) 在 Telnet 窗口或 DOS 控制窗口中激活 mysql。

2) 在一个文档窗口打开包含查询的文件。（如笔者在 Mac OS 下使用 BBEdit，在 UNIX 中使用 X Window System 下的 xterm 窗口中的 vi。）

3) 为了执行存放在文件中的某个查询，选择并拷贝它。然后切换到 Telnet 窗口或 DOS 控制台，并将该查询粘贴到 mysql。

这个过程写起来似乎有点令人讨厌，但它是一个快速录入查询的很容易的方法，实际使用时不用键入查询。

这个方法也允许在文档窗口中对查询进行编辑，而且它允许拷贝和粘贴现有查询来构造一个新的查询。例如，如果您经常从某个特定的表中选择行，但是喜欢查看以不同方式存放的输出结果，则可以在文档窗口中保存一个不同的 ORDER BY 子句的列表，然后为任意的特定查询拷贝和粘贴想使用的那个句子。

也可按其他方向拷贝和粘贴（从 Telnet 到查询文件）。在 mysql 中录入行时，它们被保存在您的主目录中的名为 .mysql\_history 的文件中。如果您手工录入了一个希望保存起来今后使用的查询，可退出 mysql，在某个编辑器中打开 .mysql\_history，然后从 .mysql\_history 拷贝和粘贴此查询到您的查询文件。

## 3. 以批方式运行 mysql

不一定必须交互式地运行 mysql。mysql 能够以非交互式（批）方式从某个文件中读取输入。这对于定期运行的查询是很有用的，因为您一定不希望每次运行此查询时都要重新键入它。只要一次性地将其放入一个文件，然后让 mysql 在需要时执行该文件的内容即可。

假定有一个查询查找 member 表的 interests 列，以找出那些对美国历史的某个方面感兴趣的历史同盟会员。如查找对大萧条期有兴趣的会员，可编写此查询如下（注意结尾处有一个分号，从而 mysql 能够知道查询语句在何处结束）：

```
SELECT last_name, first_name, email, interests FROM member
WHERE interests LIKE "%depression%"
ORDER BY last_name, first_name;
```

将此查询放入文件 interests.sql 中，然后按如下方法将其送入 mysql 执行：

```
% mysql samp_db < interests.sql
```

如果以批方式运行，缺省时，mysql 以制表符分隔格式产生输出。如果想得到与交互式地运行 mysql 时相同的表格（“方框”）式输出，可使用 -t 选项：

```
% mysql -t samp_db < interests.sql
```

如果想要保存输出结果，可将其送入文件：

```
% mysql -t samp_db < interests.sql > output_file
```



为了使用此查询来找出对 Thomas Jefferson 感兴趣的会员，可以编辑此查询文件将 depression 更改为 Jefferson 并再次运行 mysql。只要不很经常使用此查询，它工作得很好。如果经常使用，则需要更好的方法。使用此查询更为灵活的一种方法是将其放入一个外壳程序脚本中，此脚本从脚本命令行取一个参数并利用它来更改查询的文本。这样确定查询的参数，使得能够在运行脚本时指定令人感兴趣的关键词。为了了解这如何起作用，按如下编写一个较小的外壳程序脚本 interests.sh：

```
#!/bin/sh
if [ $# -ne 1 ]; then echo "Please specify one keyword"; exit; fi
mysql -t samp_db <<QUERY_INPUT
SELECT last_name, first_name, email, interests FROM member
WHERE interests LIKE "%$1%"
ORDER BY last_name, first_name;
QUERY_INPUT
```

其中第二行保证在命令行上有一个关键字；它显示一条简短的消息，或者退出。在 <<QUERY\_INPUT 和最后的 QUERY\_INPUT 之间的所有内容成为 mysql 的输入。在查询文本中，外壳程序用来自命令行的关键字替换 \$1。（在外壳程序脚本中，\$1、\$2...为命令参数。）这使相应的查询反映了执行此脚本时在命令行上指定的关键字。

在能够运行此脚本前，必须使其可执行：

```
% chmod +x interests.sh
```

现在不需要在每次运行脚本时对其进行编辑了。只要在命令行上告诉它需要查找什么就行了。如下所示：

```
% interests.sh depression
% interests.sh Jefferson
```

#### 4. 利用现有数据来创建新记录

可以用 INSERT 语句每次一行地将新记录追加到表中，但是在通过手工键入 INSERT 语句建立几个新记录后，多数人都会意识到应该有更好的追加记录的方法。一种选择是利用仅含有数据值的文件，然后利用 LOAD DATA 语句或 mysqlimport 实用程序从该文件中装入记录。

通常，可利用已经以某种格式存在的数据来建立数据文件。这些数据信息可能包含在电子表中，或许在某个其他数据库中，应该将它们转换到 MySQL。为了介绍起来简单，我们假定这些数据是在桌面微计算机的电子表中。

要将电子表数据从桌面微计算机中转换到您的 UNIX 账号下的某个文件中，可结合 Telnet 利用拷贝和粘贴。具体工作如下所示：

1) 打开UNIX 账号的一个 Telnet 连接。在 Mac OS 下，可利用诸如 Better Telnet 或 NCSA Telnet 这样的应用程序。在 Windows 下，可使用标准的 Telnet 程序。

2) 打开电子表，选择想转换的数据块，拷贝它。

3) 在 Telnet 窗中，键入下列命令开始获取数据到文件 data.txt。

```
% cat > data.txt
```

cat 命令等待输入。

4) 将从电子表拷贝来的数据粘贴到 Telnet 窗口。cat 认为您正在键入信息并忠实地将它写入到 data.txt 文件。

5) 在所有粘贴数据已经写入该文件后，如果光标停止在数据行的结尾处而不是停止在新行的开始，按 Enter。然后，按 Ctrl-D 以指示“文件结束”。cat 停止输入等待并关闭 data.txt

文件。

现在已经得到了包含有电子表中选择的数据块的 `data.txt` 文件，此文件已作好由 `LOAD DATA` 或 `mysqlimport` 加载到数据库的准备。

拷贝和粘贴是一种将数据传入 UNIX 文件的快速且简易的方法，但它最适合较小的数据集。量较大的数据可能会超出系统拷贝缓冲区。在这样的情况下，最好是以无格式文本（制表符分隔）的形式保存电子表。然后可利用 FTP 将相应文件从微机上传送到 UNIX 账号。转换文本模式（非二进制或影像模式）的文件以便行结束符转换为 UNIX 的行结束符。（UNIX 利用换行符、Mac OS 利用回车换行符、Windows 利用回车换行符/换行符对作为行结束符。）可告诉 `LOAD DATA` 或 `mysqlimport` 寻找什么换行符，但是在 UNIX 下，对含换行符的文件处理要更容易一些。

在转换了文件之后，应该检查一下在结尾处是否具空白行。如果有，应该将它们删除，否则在将该文件装载到数据库时，这些空白行将会转换为空白或畸形的记录。

来自电子表格以无格式文本保存的文件，或具有能括住包含空格的值的括号。为了在将该文件装入数据库时去掉这些括号，可利用 `LOAD DATA` 的 `FIELDS ENCLOSED BY` 子句，或利用 `mysqlimport` 的 `--fields - enclosed - by` 选项。更详细的信息请参看附录 D 中 `LOAD DATA` 的相应项。

## 1.6 向何处去

现在我们已经介绍了许多使用 MySQL 的知识。您已经知道了怎样设置数据库并创建表。能够将记录放入这些表中，并以各种方式对其进行检索，更改或删除。但是要掌握 MySQL 仍然有许多知识要学，本章中的教程仅仅给出了一些浅显的东西。通过考察我们的样例数据库就会明白这一点。我们创建了样例数据库及其表，并用一些初始的数据对其进行了填充。在这个工作过程中，我们明白了怎样编写查询，回答关于数据库中信息的某些问题，但是还有许多工作要做。

例如，我们没有方便的交互方式来输入学分保存方案的新学分记录，或输入历史同盟地址名录的会员条目。还没有方便的方法来编辑现有记录，而且我们仍然不能生成印刷或联机形式的同盟地址名录。这些任务以及一些其他的任务将在以后的各章中陆续地进行介绍，特别是在第 7 章“Perl DBI API”和第 8 章“PHP API”中将要进行详细地介绍。

下一步将阅读本书中哪部分取决于您对什么内容感兴趣。如果希望了解怎样完成已经以历史同盟和学分保存方案开始的工作，可看第一部分有关 MySQL 程序设计的内容。如果打算成为某个站点的 MySQL 管理员，本书的第三部分将对管理工作做较多的介绍。但是，笔者建议通过阅读第一部分中的其余各章，首先获得使用 MySQL 的一般背景知识。这些章节讨论了 MySQL 怎样处理数据，进一步提供有关语法和查询语句的用途，并且说明了怎样使查询运行得更快。不管您在什么环境中使用 MySQL，不管是运行 `mysql` 还是编写自己的程序，还是作为数据库管理员，用这些内容打下一个良好的基础将有助于您站在一个较高的起点上。