第二篇　使用MySQL

第 8 章　设计Web数据库

现在，我们已经熟悉了PHP的基础知识，我们将在本章开始介绍如何将数据库集成到脚本中。在第2章中，我们介绍了使用关系数据库代替普通文件的优点。这些优点包括：

- 关系数据库比普通文件的数据访问速度更快。
- 关系数据库更容易查询并提取满足特定条件的数据。
- 关系数据库具有专门的内置机制处理并发访问，因此作为程序员，不需要为此担心。
- 关系数据库可以提供对数据的随机访问。
- 关系数据库具有内置的权限系统。

对于一些更具体的例子来说，使用关系数据库能够更快速、更便捷地查询和回答客户是从什么地方来的、哪个产品卖得最好，或哪种类型的客户的消费能力最强。这些信息有助于改进站点，从而吸引更多的新客户并挽留老客户。但是，如果通过普通文件，这些特性的实现将会是特别困难的。

在本篇中，我们使用的数据库是MySQL。在开始下一章详细介绍MySQL之前，我们将讨论：

- 关系数据库的概念和术语。
- Web数据库的设计。
- Web数据库的架构。

而本篇的其他章节将包括如下内容：

- 第9章将介绍将MySQL数据库连接到Web所需的基本配置。我们将学习如何创建用户、数据库、表格和索引，以及MySQL的不同存储引擎。
- 第10章将介绍如何在命令行下查询数据库，增加、删除或更新记录。
- 第11章将介绍如何将PHP和MySQL数据库联系到一起，这样就可以通过Web界面使用和管理数据库。我们还将学习实现此操作的两种方法：使用PHP的MySQL库和使用PEAR:DB数据库抽象层。
- 第12章，将详细介绍MySQL的管理，包括权限系统、安全和优化的细节。
- 第13章，将详细介绍存储引擎，包括事务、全文搜索和存储过程。

8.1　关系数据库的概念

至今为止，关系数据库是最常用的数据库类型。在关系代数方面，它们具有很好的理论基础。当使用关系数据库的时候，并不需要了解关系理论（这是一件好事），但是还是需要理解

一些关于数据库的基本概念。

8.1.1 表格

关系数据库由关系组成，这些关系通常称为表格。顾名思义，一个表格就是一个数据的表格。电子数据表就是一种表格。

下面，我们看一个例子。图8-1是一个示例表格。这个表格包括了Book-O-Rama书店客户的姓名与地址。

CUSTOMERS

CustomerID	Name	Address	City
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

图8-1　表中为Book-O-Rama书店的客户资料

该表具有一个名称（Customers），几个数据列，每一列对应于一种不同的数据；以及对应于一个客户的数据行。

8.1.2 列

表中的每一列都有唯一的名称，包含不同的数据。此外，每一列都有一个相关的数据类型。例如，在图8-1所示的Customers表中，可以看到CustomerID列是一个整型数据，而其他3列是字符串类型。有时候，列也叫做域或者属性。

8.1.3 行

表中的每一行代表一个客户。每一行具有相同的格式，因而也具有相同的属性。行也称为记录或。

8.1.4 值

每一行由对应于每一列的单个值组成。每个值必须与该列定义的数据类型相同。

8.1.5 键

我们必须有一个能够识别每一个特定客户的方法。通常，名称并不是一个很好的方法——如果名字很普通，我们就会明白为什么。以Customers表中的Julie Smith客户为例，当打开电话本的时候，会发现里面同样的名字不计其数。

我们可以通过几种不同的方法来区分Julie。例如，如果Julie Smith所住的地方只有一个Julie Smith，可以用“Julie Smith, of 25 Oak Street, Airport West”来识别。但是，它太冗长，听起来像法律措辞，而且当在表中显示时，也需要几列的宽度。

在这个例子中我们已经做的，以及可能要在应用程序中做的就是为每个客户分配一个唯一的CustomerID。其原则与我们拥有唯一的银行账号或俱乐部会员号一样，它使得将详细信息存到数据库的操作更为方便。手动分配的身份标识号能够保证唯一性。对于一些真实信息的组合，同样也具有这个特性。

表中的标志列称为键或主键。一个键可能由几列组成。例如，如果选择将“Julie Smith,of 25 Oak Street, Airport West”来标识Julie，那么该键包含名字、地址、城市3列，而且这样还不能保证其唯一性。

通常，数据库由多个表组成，可以使用键作为表格之间的引用。在图8-2中，我们在原数据库中增加了一个表格。这个表格存储了客户的订单。Orders表中每一行表示一个订单，该订单由一个客户所预订。我们知道客户是谁，因为存储了他们的CustomerID。例如，我们可以在Orders表中OrderID值为2的行中看到该订单，进而看到订购该订单的客户的CustomerID值为1。如果再查看Customers表，可以看到CustomerID值为1的行表示Julie Smith。

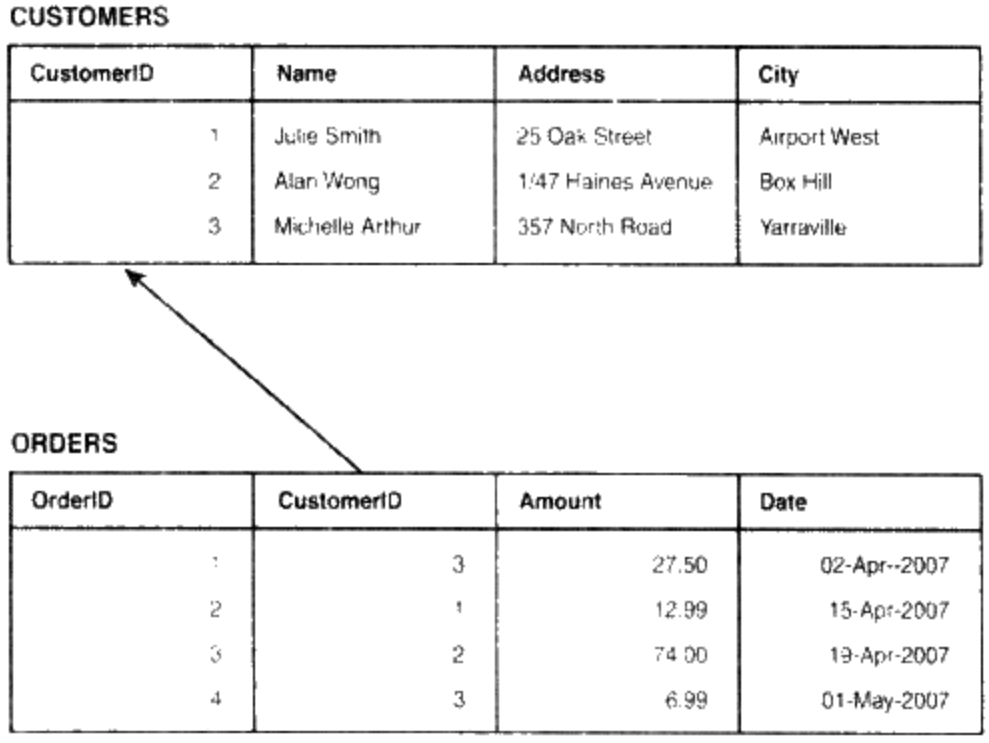

图8-2 Orders表格中每个订单都对应于Customers表中的一个客户

这种关系用关系数据库术语来描述就是外键。CustomerID是Customers表的主键，但当它出现在其他表，例如Orders表中的时候，我们就称它为外键。

读者可能会奇怪为什么会有两个不同的表，为什么不将Julie的地址和订单放到一个表中呢？下面，我们将详细探讨这个问题。

8.1.6 模式

数据库整套表格的完整设计称为数据库的模式。它是数据库的设计蓝图。一个模式应该显示表格及表格的列、每个表的主键和外键。一个模式并不会包含任何数据，但是我们可能希望在模式里使用示例数据来解析这些数据的含义。模式可以在非正式的图表中表示、用实体关系图表表示（本书中不包含此内容），或者使用文本格式表示，例如：

```
Customers(CustomerID, Name, Address, City)

Orders(OrderID, CustomerID, Amount, Date)
```

在一个模式中，带有下画线的元素表示该元素是所在关系的主键。斜体元素表示该元素是其所在关系的外键。

8.1.7 关系

外键表示两个表格数据的关系。例如，Orders表到Customers表的关系表示Orders中一行与Customers表一行的关系。

关系数据库中有3种基本的关系类型。根据关系双方所含对象的多少，可以将这些关系分为一对一、一对多、多对多3种关系。

一对一关系表示关系双方只有一个对象相互对应。例如，如果将Addresses放入与Customers表分离出的一个独立表中，则该表和Customers表就是一对一关系。从Addresses表到Customers表或者从Customers表到Addresses表也可以有外键（两者都不是必要的）。

在一对多关系里，一个表中的一行与另一表中的多行具有相互关联的关系。在这个例子中，一个用户可能有许多订单。在这些关系中，包含多行的表对应于包含一行的表应该有一个外键。在这里，我们将CustomerID放到Order表以显示其关系。

在多对多的关系中，表中的多行与另一个表中的多行具有相互关联的关系。例如，如果有两个表Books和Authors，我们会发现一本书可能由两个作者完成，这两个作者又独自著有或者与其他人合著有其他著作。通常，这种关系类型各自都要有一个表，因此，可能需要Books、Authors和Books_Authors三个表。第三个表只包含其他两个表中的键，将其作为外键对，用来显示哪些作者写了哪些书。

8.2 设计Web数据库

知道什么时候需要一个新表，以及需要哪些键，需要掌握很高的技巧。关于实体关系图和数据库规范化也有很多资料介绍，但是它已经超出了本书的范围，所以本书将不再详细介绍这些内容。但是在大多数情况下，我们可以遵循一些基本的设计原则。下面以Book-O-Rama的内容为例。

8.2.1 考虑要建模的实际对象

当创建一个数据库时，我们经常为现实世界的实体和关系建立模型，并且存储这些实体对象与关系的信息。

通常，要建模的每一种现实世界对象都需要有自己的表。考虑这样一个问题：我们要保存所有客户的同类信息。如果有一组属于同一类型的数据，就可以很容易根据这些数据创建一个表。

在Book-O-Rama的例子中，我们希望保存客户、所有出售的图书和订单的详细情况的信息。所有客户都有姓名和地址。每一个订单都有日期、总金额和所订购的图书。而每一本图书都有

国际标准图书号（ISBN）、作者、标题和价格。

这些信息集将告诉我们，在这个数据库中，至少需要建立3个表：Customers、Orders和Books。这个初始模式如图8-3所示。

CUSTOMERS

CustomerID	Name	Address	City
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

ORDERS

OrderID	CustomerID	Amount	Date
1	3	27.50	02-Apr-2007
2	1	12.99	15-Apr-2007
3	2	74.00	19-Apr-2007
4	3	6.99	01-May-2007

BOOKS

ISBN	Author	Title	Price
0-672-31697-8	Michael Morgan	Java 2 for Professional Developers	34.99
0-672-31745-1	Thomas Down	Installing GNU/Linux	24.99
0-672-31509-2	Pruitt et al.	Teach Yourself GIMP in 24 Hours	24.99

图8-3　Customers表、Orders表和Books表组成了初始模式

现在，通过模型，我们还无法知道哪本图书在哪个订单中被订购了。稍后我们将处理这个问题。

8.2.2　避免保存冗余数据

此前，我们曾经问过这样一个问题：“为什么不能将Julie Smith的地址保存在Orders表中？”

如果Julie在Book-O-Rama书店多次订购了图书（这是我们所希望的），我们会将她的资料存储多次。可能会得到如图8-4所示的Orders表。

这种设计产生两个基本问题：

- 首先是空间的浪费。既然只要将Julie的详细信息存储一次就足够了，为什么还要保存3次呢？
- 第二个问题是它会导致数据更新的不一致，也就是说，在修改数据库之后容易产生数据不一致。数据的完整性将被破坏，以至于我们不知道哪些数据正确，哪些数据不正确，通常这会导致信息的丢失。

OrderID	Amount	Date	CustomerID	Name	Address	City
12	199.50	25-Apr-2007	1	Julie Smith	25 Oak Street	Airport West
13	43.00	29-Apr-2007	1	Julie Smith	25 Oak Street	Airport West
14	15.99	30-Apr-2007	1	Julie Smith	25 Oak Street	Airport West
15	23.75	01-May-2007	1	Julie Smith	25 Oak Street	Airport West

图8-4　保存冗余的数据库设计将占用额外的空间，并且可能引起数据异常

这里，需要避免3种情况的更新不规则：修改、插入和删除不规则。

如果Julie在下了订单后搬家了，需要在3个地方而不只是一个地方更新她的地址，进行3次相同的操作。这很容易使我们只在一个地方修改数据，从而导致数据库中的数据不一致（非常糟糕的事情）。因为这些问题发生在对数据库进行修改的时候，因此称为*修改不规则*。

使用这种设计，每次在处理订单的时候都需要插入Julie的详细信息，因此每次必须检查并确认她的数据是否与表中当前行一致。如果不检查，则可能有两行关于Julie并且相互冲突的信息。例如，一行可能告诉我们Julie住在Airport West，另一行则可能表明她住在Airport。这叫做*插入不规则*，因为它出现在插入数据的时候。

第三类不规则称为*删除不规则*，因为它在从数据库中删除一行的时候发生。例如，假设一个订单已经交货，需要将它从数据库中删除。当Julie的当前订单都已交货，那么这些订单都将从数据库中删除。这意味着我们再也没有Julie的地址记录。这样就不能再为她提供服务，若下次她希望再到这里订货，我们又需要获取其信息。

通常，数据库的设计不应该出现上述不规则中的任何一种。

8.2.3　使用原子列值

使用原子列值的意思是对每一行的每个属性只存储一个数据。例如，我们需要知道每个订单都包含哪些图书，有几种方法可以实现。

一种方法是在`Orders`表中添加一列（`Orders`表中列出了所有已订图书），如图8-5所示。

ORDERS

OrderID	CustomerID	Amount	Date	Books Ordered
1	3	27.50	02-Apr-2007	0-672-31697-8
2	1	12.99	15-Apr-2007	0-672-31745-1, 0-672-31509-2
3	2	74.00	19-Apr-2007	0-672-31697-8
4	3	6.99	01-May-2007	0-672-31745-1, 0-672-31509-2, 0-672-31697-8

图8-5　通过这个设计，每行中已订图书的属性有多个值

从各方面来分析，这并不是一个好的设计。我们真正要做的是在一列里嵌入整个表——订单与图书相关联的表。当使用这种办法来实现列时，很难回答类似这样的问题，“《Java 2 for Professional Developers》一书有多少个订单？”，系统再也不能只计算匹配字段了，而必须分析每个属性值，看系统中是否包含一个匹配。

因为我们正在创建一个表中表，所以应该创建以下这样一个新表。这个新表叫`Order_Items`，如图8-6所示。

该表在表Orders和表Books之间建立一个关联。当两个对象存在多对多的关系时，这种类型的表是很常见的。在这个例子中，一个订单由许多图书组成，而且每一本图书都可以被多人订购。

ORDER_ITEMS

OrderID	ISBN	Quantity
1	0-672-31697-8	1
2	0-672-31745-1	2
2	0-672-31509-2	1
3	0-672-31697-8	1
4	0-672-31745-1	1
4	0-672-31509-2	2
4	0-672-31697-8	1

图8-6　这样的设计使得搜寻已经订购的特定书籍变得容易

8.2.4　选择有意义的键

应该确认所选择的键是唯一的。在这个例子中，我们为客户（CustomerID）和订单（OrderID）创建了一个特殊的键，因为这些现实世界中的对象可能根本就没有一个能够保证其唯一性的标识符。我们不必为图书创建一个唯一标识符，这已经实现了，因为存在ISBN。对于Order_Item，如果需要，可以添加额外的键，但是OrderID和ISBN这两个属性的组合可以是唯一的，只要一个订单中的相同图书的一个副本被当作一行。正是由于这个原因，Order_Items表还有一个数量列。

8.2.5　考虑需要询问数据库的问题

继续上一节的内容，想一想我们希望数据库回答什么问题。（回想一下在本章开始部分提到的那些问题，例如Book-O-Rama书店哪些图书卖得最好？）要回答此类问题，应该确认数据库中已经包含所有需要的数据，并且在表之间要有适当的关联。

8.2.6　避免多个空属性的设计

如果希望在数据库中添加一些图书评论，至少有两种方法可以实现。这两个方法如图8-7所示。

BOOKS

ISBN	Author	Title	Price	Review
0-672-31697-8	Michael Morgan	Java 2 for Professional Developers	34.99	
0-672-31745-1	Thomas Down	Installing GNU/Linux	24.99	
0-672-31509-2	Pruitt et al.	Teach Yourself GIMP in 24 Hours	24.99	

BOOKS_REVIEWS

ISBN	Review

图8-7　为了添加评论，可以在表Books中加一个Review列，或者专门为评论添加一个表

第一种方法意味着在Books表中加一个Review列。这样，每本书就有了一个字段来添加评论。如果数据库中的图书太多，评论员无法评论所有的书。那么在此属性项上，许多数据行就没有值。这就叫空值。

数据库里有许多空值是一件糟糕的事情。它极大地浪费空间，并且在统计列总量或对其他数值列应用计算函数时可能导致错误。当用户看到表中一部分为空的时候，他们也不知道是否因为该属性是无关的，还是数据库中有错误，或者是数据尚未输入。

通常，使用一个替代设计可以避免这种空值较多的问题。在这个例子中，可以采用图8-7给出的第二种设计。这里，Book_Reviews表中只包含带有评论的图书，当然也包含这些评论。

请注意，本设计是基于只有一个书店内部评论员的。也就是说，在Books和Reviews之间只存在一个一对一的关系。如果希望为同一本图书包含多个评论，这就是一个一对多的关系，而且必须选择第二个设计方案。此外，如果使用一本图书只有一个评论的设计，可以使用ISBN作为Book_Reviews表的主键。如果使用一本图书有多个评论的设计，必须为每一个评论引入一个唯一标识符。

8.2.7 表格类型的总结

通常，数据库由两种类型的表组成：

- 描述现实世界对象的简单表。这些表也可能包含其他简单对象的键，它们之间有一对一或一对多的关系。例如，一个客户可能有许多订单，但是一个订单只对应一个客户。这样，可以在订单里设计一行，使该行指向客户。
- 描述两个现实世界对象的多对多关系的关联表，多对多关系例如Orders与Books的关系。通常，这些表是与现实世界某种事务处理相联系的。

8.3 Web数据库架构

我们已经讨论了数据库的内部架构（或称为体系结构），下面，我们将介绍Web数据库系统的外部架构，以及Web数据库系统的开发方法。

Web服务器的基本操作如图8-8所示。这个系统由两个对象组成：一个Web浏览器和一个Web服务器。它们之间需要通信连接。Web浏览器向服务器发出请求、服务器返回一个响应。这种架构非常适合服务器发布静态页面。而分发一个基于数据库的网站架构则要复杂一些。

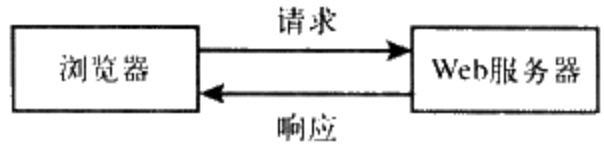

图8-8　Web浏览器和Web服务器的客户/服务器关系需要通信连接

在本书中，我们要创建的Web数据库应用程序将遵循常规的Web数据库结构，该结构如图8-9所示，我们应该已经比较熟悉大部分的这种结构了。

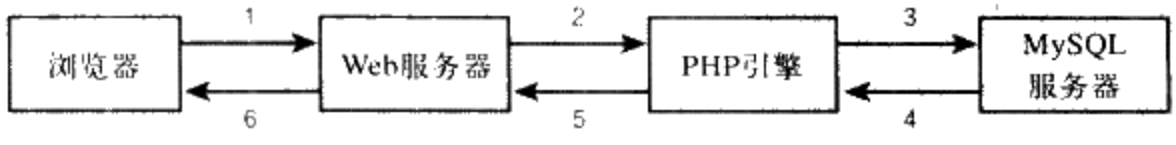

图8-9　Web数据库的基本结构包括Web浏览器、Web服务器、脚本引擎和数据库服务器

一个典型的Web数据库事务包含下列步骤，这些步骤在图8-9已经标出。以Book-O-Rama书店为例，我们逐个解释这些步骤。

1) 用户的Web浏览器发出HTTP请求，请求特定Web页面。例如，该用户可能以HTML表单

的形式，要求搜索Book-O-Rama书店里所有由Laura Thomson编写的图书。搜索结果网页称为`results.php`。

2) Web服务器收到`results.php`的请求，获取该文件，并将它传到PHP引擎，要求它处理。

3) PHP引擎开始解析脚本。脚本中有一条连接数据库的命令，还有执行一个查询（执行搜索图书）的命令。PHP打开通向MySQL数据库的连接，发送适当的查询。

4) MySQL服务器接受数据库查询并处理。将结果（一个图书的列表）返回到PHP引擎。

5) PHP引擎完成脚本运行，通常，这包括将查询结果格式化成HTML格式。然后再将输出的HTML返到Web服务器。

6) Web服务器将HTML发送到浏览器。这样用户就可看到她所搜索的图书。

这个过程基本上与脚本引擎和数据库服务器无关。通常，Web服务器软件，PHP引擎和数据库服务器都在同一台机器上运行。但是，数据库服务器在另外一台机器上运行也是非常常见的。这样做是出于保密、提高性能以及负载平衡的原因而考虑的。从开发的角度来看，要做的事情基本上是一样的，但是它能够明显提高性能。

随着应用程序在大小和复杂度上的不断增加，我们可能会将PHP应用程序分成不同的层——通常，包括与MySQL交互的数据库层、包含了应用程序核心的业务逻辑层和管理HTML输出的表示层。但是，图8-9所示的基本架构还是实用的；我们可以在PHP部分添加更多的结构。

8.4 进一步学习

在本章中，我们介绍了关系数据库设计的基本要点。如果要研究关系数据库背后深层的理论，可阅读关系数据库权威（如C.J.Date）所编写的图书。然而，这里需要提醒的是，这些资料理论性非常强，可能不能立即应用于商业Web开发。一般的Web数据库都没有那么复杂。

8.5 下一章

在下一章中，我们将开始建立MySQL数据库，首先，我们将介绍如何为一个Web站点建立一个MySQL数据库，如何查询，然后再介绍如何通过PHP对数据库进行查询。