第10章　使用MySQL数据库

在本章中，我们将介绍SQL（结构化查询语言）以及它在数据库查询中的应用。通过学习如何插入、删除、更新数据，以及如何与数据库交互，我们将继续Book-O-Rama示例数据库的开发。

在本章中，我们将主要介绍以下内容：

- SQL是什么
- 在数据库中插入数据
- 从数据库中取回数据
- 表的连接
- 使用子查询
- 更新数据库中的记录
- 创建后修改表
- 删除数据库中的记录
- 删除表

我们将首先介绍什么是SQL，以及掌握它有什么意义。

如果你还没有创建Book-O-Rama数据库，就必须在执行本章中的SQL查询之前创建它。请参阅第9章了解创建这个数据库的具体说明。

10.1　SQL是什么

SQL的全称是Structured Query Language。它是访问*关系数据库管理系统*（RDBMS）的标准语言。SQL可以用来将数据保存到数据库中，以及从数据库中取回数据。它应用于常见的数据库系统，例如MySQL、Oracle、PostgreSQL、Sybase和Microsoft SQL Server等。

SQL也有一个ANSI标准，通常，常见的数据库系统（例如，MySQL）都实现了这个标准。

当然，MySQL的SQL与标准的SQL之间还是存在一些细微的差别。这些细微差别的一部分将在MySQL的以后版本中成为MySQL的标准，而另一部分则可能是专门设计的差异。

当我们介绍到这些差异时将专门指出。MySQL的SQL与任何版本的ANSI SQL之间差异的完整列表可以在MySQL的在线手册中找到。在如下URL或其他位置，都可以找到该页面：http://www.mysql.com/doc/en/Compatibility.html

我们可能已经听说过*用于定义数据库的数据定义语言*（Data Definition Language，DDL）和*用于查询数据库的数据操作语言*（Data Manipulation Language，DML）。SQL包含这两个基础部分。在第9章中，我们已经介绍了SQL中的数据描述（DDL）语言，因此我们已经可以使用一些 DDL了。当最初建立数据库的时候，使用的是DDL。

因为DML是用来保存和获得数据库中真正数据的部分，因此我们将更频繁地使用它。

10.2　在数据库中插入数据

在可以使用数据库完成许多操作之前，必须在其中保存一些数据。完成此操作的方法通常是使用SQL的INSERT语句。

回忆一下，RDBMS都包含表，这些表都包含按列组成行的多行数据。通常，表中的每一行都描述了现实世界中的一些对象或关系，而一行中的列值则存储关于现实世界对象的信息。我们可以用INSERT语句插入一行数据到数据库中。

INSERT语句通常格式如下所示：

```
INSERT [INTO] table [(column1, column2, column3, ...)] VALUES
(value1, value2, value3, ...);
```

例如，要在Book-O-Rama的customers表中插入一个记录，可以输入如下所示命令：

```
insert into customers values
  (NULL, 'Julie Smith', '25 Oak Street', 'Airport West');
```

可以看到，我们用插入数据的实际表名称代替了“*table*”，而用特定值代替了“*values*”。在这个例子中，所有值都包含在双引号中。MySQL中的字符串应该包含在一对单引号或双引号中。(在本书中，我们将使用这两种情况)。数字和日期并不需要引号。

使用INSERT语句需要注意一些有趣的事情。我们所指定的值将按出现顺序添加到表中的列。如果只针对一些列添加内容，或者如果按不同的顺序来指定它们，那么可以在语句中列部分给出指定的列。例如：

```
insert into customers (name, city) values
( 'Melissa Jones', 'Nar Nar Goon North' );
```

如果只有特定记录的部分数据或记录中的某些字段有可选项时，这种方法非常有用。也可以通过如下所示的语法实现此功能：

```
insert into customers
set name = 'Michael Archer',
    address = '12 Addersley Avenue',
    city = 'Leeton';
```

此外，还需要注意的是，在添加Julie Smith用户的时候，我们将customerid列指定为NULL，而添加其他顾客的时候则忽略了该列。我们可能还记得，在创建该数据库的时候，我们将customerid创建为customers表的主键，因此这看起来可能有些奇怪。但是，我们已经将该字段指定为AUTO_INCREMENT。这就意味着，如果在该列中给出了NULL或者没有为该列指定任何值，MySQL将自动生成自动增加数字序列中的下一个数字值数字，并将其赋值给该列。这是非常有用的。

也可以一次将多行插入到一个表中。而每一行应该出现在自己的括号里，每组括号之间要用逗号分开。

INSERT只能带有少数几个关键字。在INSERT后面，可以添加LOW_PRIORITY或DELAYED关键字。LOW_PRIORITY关键字意味着当数据不是从表格读出时，系统必须等待并

且稍后再插入。DELAYED关键字意味着插入的数据将被缓存。如果该服务器繁忙，我们可以继续运行其他查询，而不是等待这个INSERT操作的完成。

这两个关键字以后，可以指定IGNORE（可选的）。这意味着如果尝试插入任何可能导致重复唯一键的记录行，这些记录行将被自动忽略。另一种办法是在INSERT语句的末尾指定ON DUPLICATE KEY UPDATE expression。这可以使用一个常规的UPDATE语句（稍后详细介绍）修改重复值。

我们已经将一些简单的样本数据保存到数据库中，因此可以开始使用该数据库。这只是一系列使用该多行插入方法的简单的INSERT语句。可以在本书附带的光盘中找到这个脚本文件，该文件位于\chapter10\book_insert.sql。程序清单10-1也给出了该脚本。

程序清单10-1　book_insert.sql——操作Book-O-Rama数据库表的SQL脚本

```
use books;

insert into customers values
  (3, 'Julie Smith ', '25 Oak Street', 'Airport West'),
  (4, 'Alan Wong', '1/47 Haines Avenue', 'Box Hill'),
  (5, 'Michelle Arthur', '357 North Road', 'Yarraville');

insert into orders values
  (NULL, 3, 69.98, '2007-04-02'),
  (NULL, 1, 49.99, '2007-04-15'),
  (NULL, 2, 74.98, '2007-04-19'),
  (NULL, 3, 24.99, '2007-05-01');

insert into books values
  ('0-672-31697-8', 'Michael Morgan',
   'Java 2 for Professional Developers', 34.99),
  ('0-672-31745-1', 'Thomas Down', 'Installing Debian GNU/Linux', 24.99),
  ('0-672-31509-2', 'Pruitt, et al.', 'Teach Yourself GIMP in 24 Hours', 24.99),
  ('0-672-31769-9', 'Thomas Schenk ',
   'Caldera OpenLinux System Administration Unleashed', 49.99);

insert into order_items values
  (1, '0-672-31697-8', 2),
  (2, '0-672-31769-9', 1),
  (3, '0-672-31769-9', 1),
  (3, '0-672-31509-2', 1),
  (4, '0-672-31745-1', 3);

insert into book_reviews values
  ('0-672-31697-8', 'The Morgan book is clearly written and goes well beyond
                     most of the basic Java books out there.');
```

可以通过MySQL输入如下命令运行该脚本：

```
> mysql -h host -u bookorama -p books < /path/to/book_insert.sql
```

10.3　从数据库中获取数据

在SQL中，经常使用的是SELECT语句。它通过选择匹配表中指定规则的行从数据库中获取数据。在许多情况下，可以通过不同的方法和选项来使用SELECT语句。

一个SELECT语句的基本格式如下所示：

```
SELECT [options] items
[INTO file_details]
FROM tables
[ WHERE conditions ]
[ GROUP BY group_type]
[ HAVING where_definition]
[ ORDER BY order_type]
[LIMIT limit_criteria]
|PROCEDURE proc_name(arguments)]
|lock_options]
;
```

在接下来的内容中，我们将介绍该语句的每个子句。首先，查看没有任何可选子句的查询，该子句从特定的表中选择一些元素。通常，这些元素是表中的列。（它们也可以是任何MySQL表达式的结果。在本节后续内容中，我们将详细介绍一些有用的MySQL表达式。）该查询列出了customers表中的name列和city列的内容：

```
select name, city
from customers;
```

假设我们已经输入了程序清单10-1给出的样本数据，以及本章前面所给出的两个示例INSERT语句，该查询的执行结果如下所示：

```
+----------------+--------------------+
| name           | city               |
+----------------+--------------------+
| Julie Smith    | Airport West       |
| Alan Wong      | Box Hill           |
| Michelle Arthur | Yarraville        |
| Melissa Jones  | Nar Nar Goon North |
| Michael Archer | Leeton             |
+----------------+--------------------+
```

可以看到，我们已经从指定的表customers中得到了包含选中的元素——name和city。该数据显示的是customers表中各行数据。

通过在select关键字后给出列名称，可以指定任何数量的列。也可以指定一些其他元素。其中一个有用的是通配符“*”，它可以匹配指定的一个或多个表中所有列。例如，要获得order_items表中所有的列和行，可以使用如下所示的命令：

```
select *
from order_items;
```

以上命令的输出结果如下所示：

```
+---------+---------------+----------+
| orderid | isbn          | quantity |
+---------+---------------+----------+
|       1 | 0-672-31697-8 |        2 |
|       2 | 0-672-31769-9 |        1 |
|       3 | 0-672-31769-9 |        1 |
|       3 | 0-672-31509-2 |        1 |
|       4 | 0-672-31745-1 |        3 |
+---------+---------------+----------+
```

10.3.1　获取满足特定条件的数据

要访问一个表中行的子集，需要指定一些选择条件。可以使用子句WHERE来实现。例如：

```
select *
from orders
where customerid = 3;
```

将选择orders表中的所有列，但是只有customerid为5的行将被选中。如下所示的是以上代码的输出：

```
+---------+------------+--------+------------+
| orderid | customerid | amount | date       |
+---------+------------+--------+------------+
|       1 |          5 |  69.98 | 2007-04-02 |
|       4 |          5 |  24.99 | 2007-05-01 |
+---------+------------+--------+------------+
```

WHERE子句指定了用于选择特定行的条件。在这个例子中，选择了customerid为5的行。

等于符号可以用来测试两者是否相等。请注意，这与PHP不同，当它们在一起使用的时候，容易造成混淆。

除了相等，MySQL还支持所有比较操作符和正则表达式。表10-1给出了在WHERE子句最经常用到的比较操作符。请注意，该表并不没有给出所有——如果需要使用本表中未列出的操作符，请查阅相关MySQL手册。

表10-1　WHERE子句的实用比较运算符

运 算 符	名称（如果可以应用）	例　子	描　述
=	等于	customerid=3	测试两个值是否相等
>	大于	amount>60.00	测试一个值是否大于另一个值
<	小于	amount<60.00	测试一个值是否小于另一个值
>=	大于或等于	amount>=60.00	测试一个值是否大于或等于另一个值
<=	小于或等于	amount<=60.00	测试一个值是否小于或等于另一个值
!=或<>	不等于	quantity!=3	测试两个值是否不等
IS NOT NULL	n/a	地址不为空	测试字段是否包含一个值

（续）

运 算 符	名称（如果可以应用）	例 子	描 述
IS NULL	n/a	地址为空	测试字段是否不包含一个值
BETWEEN	n/a	0到60.00之间的数量	测试一个值是否大于或等于最小值并小于或等于最大值
IN	n/a	city in ("Carlton", 'Moe')	测试一个值是否在特定的集合里
NOT IN	n/a	city not in ("Carlton", 'Moe')	测试一个值是否不在特定的集合里
LIKE	模式匹配	name like ('Fred %')	用简单的MySQL模式匹配检查一个值是否匹配于一个模式
NOT LIKE	模式匹配	name not like ('Fred %')	检查一个值是否不匹配于一个模式
REGEXP	常规表达式	name regexp	检查一个值是否匹配一个常规表达式

在上表的最后3行提到了LIKE与REGEXP。两者都是模式匹配形式。

LIKE使用简单的SQL模式匹配。模式可以由常规文本加上匹配任何数量字符的“%”（百分号），和只匹配一个字符的“_”（下画线）组成。

REGEXP关键词用于正则表达式的匹配。MySQL使用POSIX正则表达式。除了REGEXP之外，也可以使用RLIKE，它是REGEXP的别名。POSIX正则表达式也可以在PHP中使用。

请参阅第4章，了解关于正则表达式的详细介绍。

也可以使用简单的操作符和模式匹配语法以及适用AND和OR组合成更加复杂的条件，例如：

```
select *
from orders
where customerid = 3 or customerid = 4;
```

10.3.2 从多个表中获取数据

通常，要通过数据库回答一个问题，必须使用多个表中的数据。例如，如果要知道哪些顾客在本月中有订单，就需要查阅customers表和orders表。特别地，如果还希望知道他们订购了什么，还需要查阅order_items表。

这些数据分布在不同的的表中，因为它们与现实世界的对象相关。这是设计优秀数据库的原则之一，我们在第8章中已经介绍了。

要在SQL中将这些信息放到一起，必须执行一个名为关联的操作。简单地说，这意味着需要根据数据间的关系将两个或更多的表关联到一起。例如，如果要查看顾客Julie Smith的订单，我们就要查阅customers表中值为Julie的customerid，然后从orders表查找该customerid所对应的订单。

尽管关联的概念非常简单，但是它是SQL中比较微妙和复杂的一部分。MySQL中实现了许多不同种类的连接，而每一个连接都具有不同的用途。

1. 简单的双表关联

现在，让我们以刚刚讨论的一些对Julie Smith的SQL查询为开始：

```
select orders.orderid, orders.amount, orders.date
from customers, orders
where customers.name = 'Julie Smith'
and customers.customerid = orders.customerid;
```

该查询的输出结果如下所示：

```
+---------+--------+------------+
| orderid | amount | date       |
+---------+--------+------------+
|       1 |  69.98 | 2007-04-02 |
|       4 |  24.99 | 2007-05-01 |
+---------+--------+------------+
```

这里有几点需要注意。首先，因为是通过来自两个表的信息来完成这个查询，因此我们必须将两个表都列在这里。

通过列出两个表，也指定了关联的类型，尽管可能还不知道它。表名称之间的逗号等价于输入INNER JOIN或CROSS JOIN。这是一种类型的关联，有时也称为完全关联（full join）或表的笛卡儿乘积（Cartesian product）。其意思是，“将多个表列出来，形成一个大表。该表应该有一行来自所有表的每一行的所有可能组合，无论它是否有意义”。换句话说，我们得到了一个表，customers表的每一行都在该表中，并且这些行都与orders表中每一行相匹配，而不管顾客是否下了一个特定的订单。

在大多情况下，这样做并没有很大的意义。通常，我们要做的是查看真正匹配的行，即匹配特定顾客的该顾客所订的订单。

我们通过在WHERE子句中使用关联条件（join condition）来完成。这是一类条件语句，它解释了哪些属性显示两个表之间的关系。在这个例子中，关联条件是：

```
customers.customerid = orders.customerid
```

以上代码将告诉MySQL，如果customers表中的customerid与orders表中的customerid相匹配，那么就将行显示在结果表中。

通过在查询中添加此关联条件，我们实际上已经将关联转变成另一种类型，可以称之为等价关联（equi-join）。

注意我们使用了点号以使得来自某个表的某列这种关系看起来更清晰。也就是，customers.customerid表示来自customers表的customerid，而orders.customerid则表示orders表中的customerid。

如果一列的名称不具有唯一性，也就是，如果某列出现在多于一个表中的时候，我们需要使用点号。作为其扩充，也可以用它表示来自不同数据库的非模糊列。在这个例子中，使用了table.column表示方法。可以用database.table.column来指定数据库，例如，要测试如下所示的条件：

```
books.orders.customerid = other_db.orders.customerid
```

然而，在查询中，可以使用点号表示方法来表示所有被引用的列。这也是一个好主意，特别是在查询开始变得复杂之后。MySQL不要求它，但是它的确可以使查询变得更易读和易于维护。注意我们在前面介绍的查询其余部分采用了这一惯例，例如，使用条件：

```
customers.name = 'Julie Smith'
```

name列只出现在表customers中，我们并不需要指定它。MySQL将不会产生混淆。然而对编程人员来说，name本身就很模糊，因此当指定customer.name时，这使得代码更清晰。

2. 关联多个表

关联多于两个表的情况并不比两个表的关联更复杂。按照通常的规则，必须利用关联条件成对地关联表。可以把它想像为从一个表到一个表再到另一个表地跟踪数据间的关系。 例如，如果我们要知道哪些顾客已经订购了关于Java的图书（可能我们要向他们发送一本关于java的新书信息），需要在几个表中搜索这些关系。

我们需要找到在关于Java的order_item中至少订了一个订单的顾客。从customers表到orders表，可以如前所述使用customerid。从orders表到order_items表，可以使用orderid。要从order_items表到books表中特定的书。可以使用ISBN。完成所有这些连接后，就可以测试书刊标题中是否包含“Java”一词，并返回订购了其中任何一本书的顾客名称。

看看我们如何完成所有这些查询：

```
select customers.name
from customers, orders, order_items, books
where customers.customerid = orders.customerid
and orders.orderid = order_items.orderid
and order_items.isbn = books.isbn
and books.title like '%Java%';
```

该查询将返回如下输出：

```
+-----------------+
| name            |
+-----------------+
| Julie Smith     |
+-----------------+
```

请注意，我们在4个不同表之间跟踪数据，并利用等价关联来获得这些数据，而在这个过程中，需要使用3个不同的关联条件。要为每一对即将关联的表设置一个条件，因此关联条件的总数量应该比将要关联的表数少一个，这通常是正确的。这条重要的规则在测试查询效率不高时非常实用。然后核对关联条件并确认已经完成了从已知到需要知道的自始至终的跟踪过程。

3. 查找不匹配行

在MySQL中，我们还经常使用的另一个重要关联类型是左关联。

在前面的例子中可以注意到，只有那些在表之间有匹配的行才能被包含进来。有时，我们特别需要没有匹配的行。例如，从来没有订单的顾客，或从没被订购过一次的图书。

在MySQL中，实现此类查询最简单的方法就是使用左关联。左关联是在两个表之间指定的关联条件下匹配的数据行。如果右边的表中没有匹配行，结果中就会增加一行，该行右边的列

为NULL。

我们看下面的例子：

```
select customers.customerid, customers.name, orders.orderid
from customers left join orders
on customers.customerid = orders.customerid;
```

该SQL查询使用一个左关联将customers表和orders表关联起来。注意左关联的连接条件所使用的语法有些不同；在这个例子中，关联条件出现在SQL语句的特殊子句ON子句中。

该查询的输出结果如下所示：

```
+------------+------------------+---------+
| customerid | name             | orderid |
+------------+------------------+---------+
|          3 | Julie Smith      |       1 |
|          3 | Julie Smith      |       4 |
|          4 | Alan Wong        |    NULL |
|          5 | Michelle Arthur  |    NULL |
+------------+------------------+---------+
```

该输出只显示了orderid为非空的客户。

如果我们只需查看没有订购任何商品的顾客，可以检查右边表的主键字段是否为NULL（在这个例子中，orderid），因为在任何真正的数据行中这个不应该为NULL：

```
select customers.customerid, customers.name
from customers left join orders
using (customerid)
where orders.orderid is null;
```

其结果为：

```
+------------+-----------------+
| customerid | name            |
+------------+-----------------+
|          4 | Alan Wong       |
|          5 | Michelle Arthur |
+------------+-----------------+
```

注意在这个例子中，对关联条件也使用了不同的语法。左关联支持第一个例子中用到的ON语法，也支持第二个例子中用过的USING语法。请注意，USING语法并不需要指定连接属性所来自的表；正是由于这个原因，如果希望使用USING子句，两个表中的列必须有同样的名称。

可以通过使用子查询来满足这种查询。在本章的稍后内容，我们将了解子查询。

4. 使用表的别名：Aliases

通常，用表的别名来指定表是很方便的，偶尔也是很必要的。表的其他名称就是表的别名（aliase）。可以在一个查询的开始创建它们，然后在整个查询过程中使用。因为它们便于记忆，因此使用起来非常方便。考虑前面讨论过的庞大查询，我们可以使用别名对其重新编写：

```
select c.name
```

```
from customers as c, orders as o, order_items as oi, books as b
where c.customerid = o.customerid
and o.orderid = oi.orderid
and oi.isbn = b.isbn
and b.title like '%Java%';
```

因为可以声明所需要使用的表，因此我们增加了AS子句以声明该表的别名。我们也可以给列起别名，在接下来介绍集合函数的时候我们将详细介绍它。

当要关联一个表到表本身的时候就必须使用表别名。这听起来很深奥而难于理解。例如，如果要查找同一个表中值相同的行，它就很有意义。如果要查找住在同一城市的顾客（可能要建立一个阅读小组），可以给同一个表（customers）起两个不同的别名：

```
select c1.name, c2.name, c1.city
from customers as c1, customers as c2
where c1.city = c2.city
and c1.name != c2.name;
```

基本上，我们要做的是将表customers看作两个不同的表，c1和c2，并在city列执行关联。注意这也需要另外一个条件：c1.name!=c2.name； 这可以避免顾客作为其自身的匹配而出现。

5. 关联的总结

表10-2总结了我们已经介绍的不同类型关联。还有一些这里没有介绍的其他类型，但是本表给出的是可能遇到的主要类型。

表10-2 MySQL中的关联类型

名 称	描 述
笛卡儿乘积	所有表所有行的所有关联。实现方法，在列的名称之间指定一个逗号，而不是指定一个WHERE子句
完全关联	同上
交叉关联	同上，也可通过在关联的表名之间指定CROSS JOIN关键词而指定
内部关联	如果没有WHERE条件，等价于完全关联。通常，需要指定一个WHERE条件以使它成为真正的内部关联
等价关联	在关联中使用一个带“=”号的条件表达式匹配来自不同表中的行。在SQL中，这是带WHERE子句的关联
左关联	试图匹配表的行并在不匹配的行中填入NULL，在SQL中使用LEFT JOIN关键词。用于查找要避免的值。类似地，可以使用RIGHT JOIN

10.3.3 以特定的顺序获取数据

如果要通过查询以某一特定顺序显示查询行，可以利用SELECT语句中的ORDER BY子句。该特性可以方便地用于实现很好的可阅读格式的显示输出。

ORDER BY子句可以根据出现在SELECT子句中的一列或多列对数据行进行排序。例如：

```
select name, address
from customers
```

```
order by name;
```

该查询将以名称的字母顺序返回顾客的名称与地址，如下所示：

```
+-----------------+--------------------+
| name            | address            |
+-----------------+--------------------+
| Alan Wong       | 1/47 Haines Avenue |
| Julie Smith     | 25 Oak Street      |
| Michelle Arthur | 357 North Road     |
+-----------------+--------------------+
```

请注意，在这个例子中，因为名称的格式为“姓，名”，它们的字母顺序是以姓排序的。如果要按照名排序，则需要将它们分配到两个不同的字段。

默认的顺序是升序（从*a*到*z*或数字顺序）。如果使用了ASC关键词，也可以指定它：

```
select name, address
from customers
order by name asc;
```

还可以用DESC（descending，降序）关键词指定它为降序：

```
select name, address
from customers
order by name desc;
```

此外，也可以在多于一列的基础上进行排序。还可以使用列的别名或者甚至它们的位置数字（例如，3是表中第3列）代替其名称。

10.3.4　分组与合计数据

我们经常需要知道多少行分成一个特定的集合，或一些列的平均值。例如，每个订单的平均金额。MySQL有一组合计函数可实现这类查询。

这些合计函数可以作为一个整体应用于一个表，或者表中的一组数据。最常用的函数如表10-3所示。

表10-3　MySQL中的合计函数

名　称	描　述
AVG（列）	指定列的平均值
COUNT（项目）	如果指定一列，这将给出本列中非空（NULL）值的列数。如果在列前加DISTINCT单词，将得到本列中不同值的列数。如果指定COUNT(*)，将得到包含空值（NULL）的行在内的行数
MIN（列）	指定列的最小值
MAX（列）	指定列的最大值
STD（列）	指定列的标准背离值
STDDEV（列）	与STD（列）相同
SUN（列）	指定列的所有值的和

我们来看看一些例子，以前面提到的一个例子为开始。可以如下计算一个订单总金额的平均值：

```
select avg(amount)
from orders;
```

输出如下所示：

```
+--------------+
| avg(amount)  |
+--------------+
|    54.985002 |
+--------------+
```

要获取更详细的信息，可以使用GROUP BY子句。这使我们可以按分组浏览订单总量的平均值。例如，按照顾客数分组浏览。我们将知道哪些顾客的订单总金额最大：

```
select customerid, avg(amount)
from orders
group by customerid;
```

当通过合计函数使用GROUP BY子句的时候，它实际上改变了该函数的行为。该查询并不是给出表中的平均订单总量，而是给出每个顾客（或者，更具体地说，是每个customerid）的平均订单总金额：

```
+------------+--------------+
| customerid | avg(amount)  |
+------------+--------------+
|          1 |    49.990002 |
|          2 |    74.980003 |
|          3 |    47.485002 |
+------------+--------------+
```

在使用分组和合计函数的时候，需要注意的是：在ANSI SQL中，如果使用了一个合计函数或GROUP BY子句，出现在SELECT子句中的必须是合计函数名称和GROUP BY子句的列名称。同样，如果希望在一个GROUP BY子句中使用一列，该列名称必须在SELECT子句中列出。

MySQL实际上留了一点回旋余地。它支持一种*扩展语法*（extended syntax），该语法可以在SELECT子句中略去一些实际上并不需要的项目。

除了分组与合计数据，我们实际上还可以使用HAVING子句测试一个合计的结果。它可以直接放在GROUP BY子句后，有些类似于只用于分组与合计的WHERE子句。

对前面的例子进行扩展，如果希望知道哪些顾客的平均订单总金额超过$50，可以使用如下所示的查询：

```
select customerid, avg(amount)
from orders
group by customerid
having avg(amount) > 50;
```

请注意，HAVING子句应用于这些组。该查询将返回如下所示的输出：

```
+------------+-------------+
| customerid | avg(amount) |
+------------+-------------+
|          2 |   74.980003 |
+------------+-------------+
```

10.3.5 选择要返回的行

SELECT语句中的一个可能在Web应用中特别实用的子句是LIMIT子句。它可以用来指定输出中哪些行应该返回。它带两个参数：起始行号与返回行数。

下例中的查询说明了LIMIT的使用：

```
select name
from customers
limit 2, 3;
```

该查询意思是："从customers表中选择name列，返回3行，从返回结果的第2行开始。"请注意，行号是以0开始索引的；也就是说，结果的第1行其行号为0。

对于Web应用程序，这是很有意义的，例如，顾客浏览一个目录中的产品时，每页显示10个项目。但是，请注意，LIMIT并不是ANSI SQL的一部分。它是MySQL的扩展，因此使用这个关键字将使得SQL与大多数其他RDBMS不兼容。

10.3.6 使用子查询

子查询是一个嵌套在另一个查询内部的查询。虽然大多数子查询功能可以通过连接和临时表的使用而获得，但是子查询通常更容易阅读和编写。

1. 基本的子查询

子查询的最常见用法是用一个查询的结果作为另一个查询的比较条件。例如，如果希望找到一个金额最大的订单，可以使用如下所示的查询：

```
select customerid, amount
from orders
where amount = (select max(amount) from orders);
```

该查询将给出如下所示的结果：

```
+------------+--------+
| customerid | amount |
+------------+--------+
|          2 |  74.98 |
+------------+--------+
```

在这个例子中，子查询返回了单一值（最大金额），然后再用作输出查询的比较条件。这是使用子查询的好例子，因为这个特定查询无法使用ANSI SQL的连接来完成。

但是，这个关联查询将产生相同的输出：

```
select customerid, amount
```

```
from orders
order by amount desc
limit 1;
```

由于它依赖LIMIT，这个查询与大多数RDBMS并不兼容，但是在MySQL中，它的执行比子查询版本的查询效率更高。

MySQL很长时间没有采纳子查询的主要原因之一在于，多数查询可以在没有子查询的情况下完成。从技术角度看，可以创建具有相同作用的单一合法的ANSI SQL查询，但是这将依赖低效率的MAX-CONCAT。

就像本例一样，可以在所有常见比较操作符中使用子查询值。还有一些可供使用的特殊子查询比较操作符，这将在下一节详细介绍。

2. 子查询和操作符

特殊的子查询操作符共有5个。其中有4个可以在常规子查询中使用，而另一个（EXISTS）通常只在相关联的子查询中使用，相关联的子查询将在下一节介绍。表10-4列出了常见的4个子查询操作符。

表10-4 子查询操作符

名 称	示例语法	描 述
ANY	SELECT c1 FROM t1 WHERE c1 > ANY (SELECT c1 FROM t2);	如果子查询中的任何行比较条件为true，返回true
IN	SELECT c1 FROM t1 WHERE c1 IN (SELECT c1 from t2);	等价于 = ANY
SOME	SELECT c1 FROM t1 WHERE c1 > SOME (SELECT c1 FROM t2);!	ANY的别名，有时候更容易阅读
ALL	SELECT c1 FROM t1 WHERE c1 > ALL (SELECT c1 from t2);	如果子查询中的所有行比较条件为true，返回true

这些操作符都只可以出现在比较操作符之后，除了IN，它相当于隐藏了比较操作符（=）。

3. 关联子查询

在关联子查询中，情况变得更加复杂。在关联子查询中，可以在内部查询中使用外部查询的结果。例如：

```
select isbn, title
 from books
 where not exists
  (select * from order_items where order_items.isbn=books.isbn);
```

这个查询说明了关联子查询和最后一个特殊子查询操作符（EXISTS）的使用。它将检索任何还没有被订购的图书（这与使用左关联所检索到的信息相同）。请注意，内部查询只能包括FROM列表中的order_items表，但是还是引用了books.isbn。换句话说，内部查询将引用外部查

询的数据。这是关联子查询的定义：查询匹配（或者，在这个例子中，是不匹配）外部行的内部行。

如果子查询中存在任何匹配行，EXISTS操作符将返回true。相反，如果子查询中没有任何匹配行，NOT EXISTS将返回true。

4. 行子查询

目前介绍的所有子查询都将返回单一的值，虽然在大多数情况下，该值为true或false（就像前面使用EXISTS的例子）。行子查询将返回整行，它可以与外部查询的整行进行比较。通常，这种方法用来在一个表中查找存在于另一个表的行。在图书数据库中，并没有一个很好的例子。但是，该语法的常规例子可以如下所示：

```
select c1, c2, c3
from t1
where (c1, c2, c3) in (select c1, c2, c3 from t2);
```

5. 使用子查询作为临时表

可以在一个外部查询的FROM子句中使用子查询。这种方法允许有效地查询子查询的输出，并将其当作一个临时表。

作为最简单的例子，临时表的使用如下所示：

```
select * from
(select customerid, name from customers where city='Box Hill')
as box_hill_customers;
```

请注意，我们将子查询放在了FROM子句中。在子查询后面就是结束的括号，必须为子查询的结果定义一个别名。我们可以将其当作外部查询的任何表。

10.4　更新数据库记录

通常，除了从数据库中获得数据，我们还希望修改这些数据。例如，我们可能要提高数据库中图书的价格。可以使用UPDATE语句来完成这个任务。

UPDATE语句的常用格式是：

```
UPDATE [LOW_PRIORITY] [IGNORE] tablename
SET column1=expression1,column2=expression2,...
[WHERE condition]
[ORDER BY order_criteria]
[LIMIT number]
```

其基本思想是更新名为tablename的表，设置每列的名称为适当的表达式。可以通过WHERE子句限制UPDATE到特定的行，也可以使用LIMIT子句限制受影响的总行数。ORDER BY通常只在LIMIT子句的连接中使用；例如，如果只更新前10行，可以将它们放置在前面的位置。如果指定了LOW_PRIORITY和IGNORE关键字，就会像在INSERT语句中一样工作。

接下来，我们看一些例子。如果要将图书的价格提高10%，可以使用一个没有WHERE子句的UPDATE语句，如下所示：

```
update books
set price = price*1.1;
```

另一方面，如果希望修改一行（例如，要更新一个顾客的地址）可以使用如下所示语句：

```
update customers
set address = '250 Olsons Road'
where customerid = 4;
```

10.5 创建后修改表

除了可以更新行，可能还需要改变数据库中表的结构。要实现这个目的，可以利用灵活的ALTER TABLE语句。ALTER TABLE语句基本格式如下：

```
ALTER TABLE [IGNORE] tablename alteration [, alteration ...]
```

请注意，在ANSI SQL中，每个ALTER TABLE语句只可实现一次修改，但是在MySQL中允许实现多次修改。每个修改子句可用于修改表的不同部分。

如果指定了IGNORE子句并且尝试的修改可能会产生重复的主键，第一个重复的主键将进入修改后的表，而其他重复的主键将被删除。如果没有指定（默认情况），该修改将失败并且被回滚。

使用该语句可以做不同类型的修改，这些修改如表10-5所示。

表10-5 用ALTER TABLE语句可能完成的修改

语 法	描 述
ADD[COLUMN]*column_description* [FIRST | AFTER *column*]	在指定地方添加新列（如果没有指定，就在最后一列后面）。注意*column_description*需要名称和类型与在CREATE语句中的名称和类型一致
ADD[COLUMN] (*column_description*, *column_description*, ...)	在表结尾添加一个或多个新的列
ADD INDEX[*index*](*column*, ...)	在指定的一列或几列添加一个表的索引
ADD [CONSTRAINT [*symbol*]] PRIMARY KEY(*column*, ...)	指定一列或几列为表主键。CONSTRAINT是针对使用外键的表。请参阅第13章获得详细信息
ADD UNIQUE [CONSTRAINT CONSTRAINT[*symbol*]] [*index*](*column*, ...)	在指定的一列或几列添加一个唯一的表索引。是针对使用外键的InnoDB表。请参阅第13章获得详细信息
ADD [CONSTRAINT [*symbol*]] FOREIGN KEY [*index*](*index_col*,...) [*reference_definition*]	为一个InnoDB表添加外键。请参阅第13章的详细介绍
ALTER[COLUMN]*column*{SET DEFAULT *value*| DROP DEFAULT}	添加或者删除特定列的默认值
CHANGE[COLUMN]*column* *new_column_description*	改变名为column的列，添加所列出的描述。注意，这可用于改变列的名称，因为*new_column_description*包含名称
MODIFY[COLUMN]*column_description*	类似于CHANGE。可以用来修改列类型，而不是列名称
DROP[COLUMN]*column*	删除指定的列
DROP PRIMARY KEY	删除主索引（而不是列）
DROP INDEX *index*	删除指定的索引

（续）

语　法	描　述
DROP FOREIGN KEY *key*	删除外键（但是不是列）
DISABLE KEYS	禁用索引更新
ENABLE KEYS	开启索引更新
RENAME [AS] *new_table_name*	重新命名一个表
ORDER BY *col_name*	以特定顺序的行重新创建表（请注意，在开始修改表时，行将不会保持顺序。）
CONVERT TO CHARACTER SET *cs* COLLATE *c*	将所有文本列转换成指定字符集和排序
[DEFAULT] CHARACTER SET *cs* COLLATE *c*	设置默认的字符集和排序
DISCARD TABLESPACE	删除InnoDB表的可能表空间文件（请参阅第13章关于InnoDB的详细介绍。）
IMPORT TABLESPACE	为InnoDB表重新创建可能的表空间文件（请参阅第13章关于InnoDB的详细介绍。）
table_options	允许重新设置表选项。就像CREATE TABLE一样使用相同的语法

下面，我们看看ALTER TABLE语句的一些更常见的用法。

一个经常出现的情况是：特定的列空间没有“足够大”，不能容纳它必须容纳的数据。例如，在customers表中，已经允许名称可以达到50个字符。在开始接收一些数据后，我们可能发现一些名称因为太长而被截短了。我们可以通过改变该列的数据类型，使其为长度为70个字符，以弥补这个缺点：

```
alter table customers
modify name char(70) not null;
```

另一个经常出现的问题是需要新增加一列。如果当地引进图书营业税，Book-O-Rama要将税收额加到整个订单上，但是又要将图书税与订单分开。这样，我们可以在orders表中增加一个税收列（tax），如下所示：

```
alter table orders
add tax float(6,2) after amount;
```

删除一列也是经常出现的问题。要删除一行，只要加上如下语句即可：

```
alter table orders
drop tax;
```

10.6　删除数据库中的记录

从数据库中删除行的操作非常简单。可以使用DELETE语句完成，DELETE语句常见格式如下所示：

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table
[WHERE condition]
```

```
[ORDER BY order_cols]
[LIMIT number]
```

如果将上述代码改写成：

```
delete from table;
```

所有表中的行都将被删除，因此要非常小心！通常，如果希望删除特定的行，可以使用WHERE子句指定要删除的行。例如，如果已经没有了某本书，或一个顾客已经很久没有订购订单了，而现在想整理一下数据库，那么可能要删除一些东西。

```
delete from customers
where customerid=5;
```

LIMIT子句可用于限制实际删除的最大行数。ORDER BY通常与LIMIT结合使用。

LOW_PRIORITY和IGNORE的用途与前面介绍的相同。QUICK可以使得对MyISAM表的操作执行得更快。

10.7 表的删除

有时可能要删除整个表。可以使用DROP TABLE语句来完成，该语句非常简单，如下所示：

```
DROP TABLE table;
```

这将删除表中所有行以及表本身，因此使用的时候要非常小心。

10.8 删除整个数据库

还可以更进一步，用DROP DATABASE语句删除整个数据库，该语句格式如下所示：

```
DROP DATABASE database;
```

这将删除所有行、所有表、所有索引和数据库本身，而且不会提醒我们在使用该语句时要小心。

10.9 进一步学习

在本章中，我们已经介绍了日常使用MySQL数据库时经常使用SQL命令。在接下来的两章中，我们将讨论如何将MySQL和PHP联系在一起，这样就可以通过Web访问数据库。

此外，我们还将探讨一些高级的MySQL技术。

要了解更多关于SQL的信息，请参阅ANSI SQL标准。其网址为：http://www.ansi.org/

要了解更多关于MySQL对ANSI SQL的扩充信息，请参阅MySQL网站：http://www.mysql.com

10.10 下一章

在第11章中，我们将介绍创建可以通过Web访问的Book-O-Rama数据库的过程。