CENGAGE Learning®

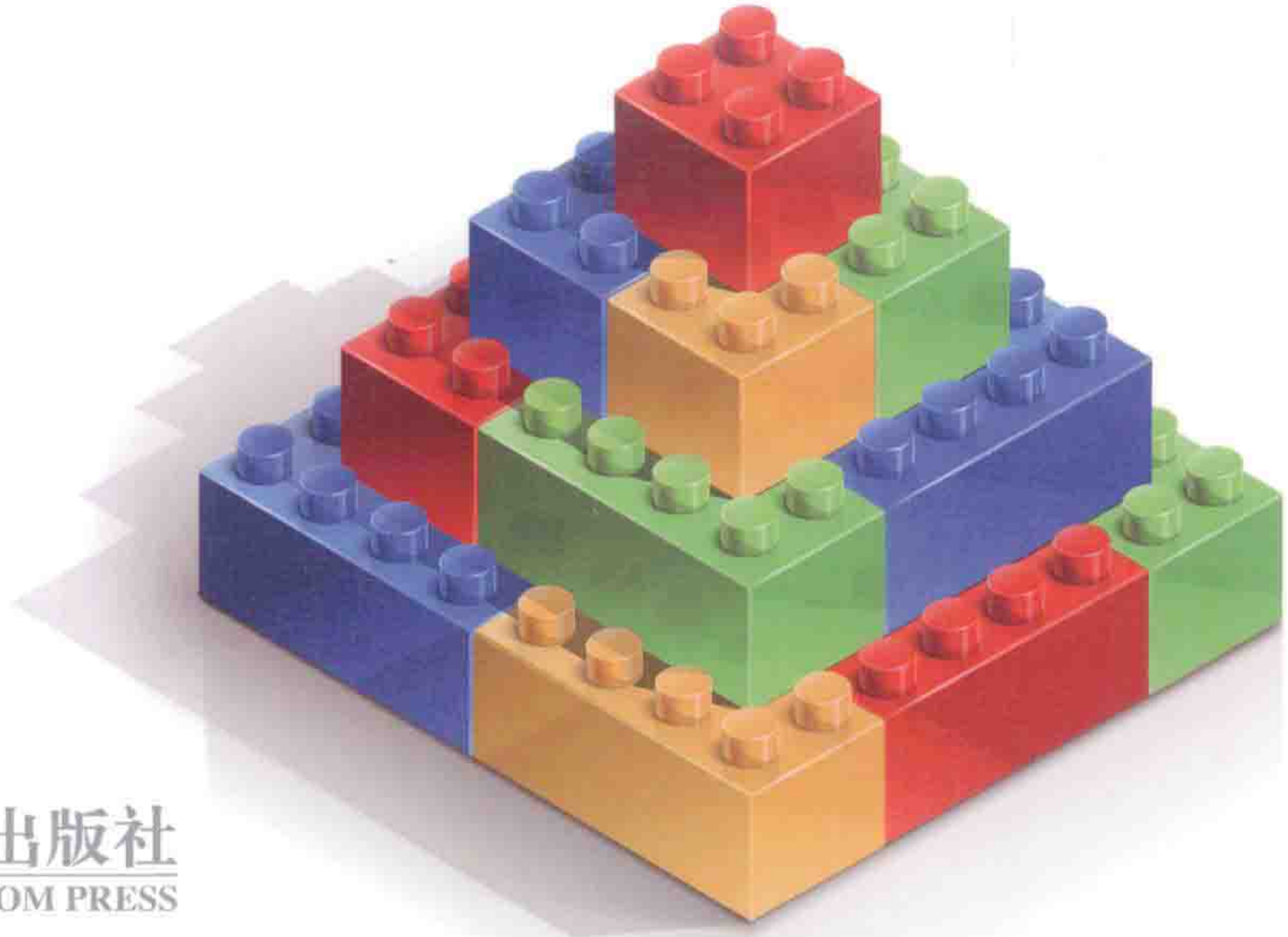
亚马逊畅销SQL入门图书

SQL

初学者指南

The Language of SQL

[美] Larry Rockoff 著　　李强 译



人民邮电出版社
POSTS & TELECOM PRESS

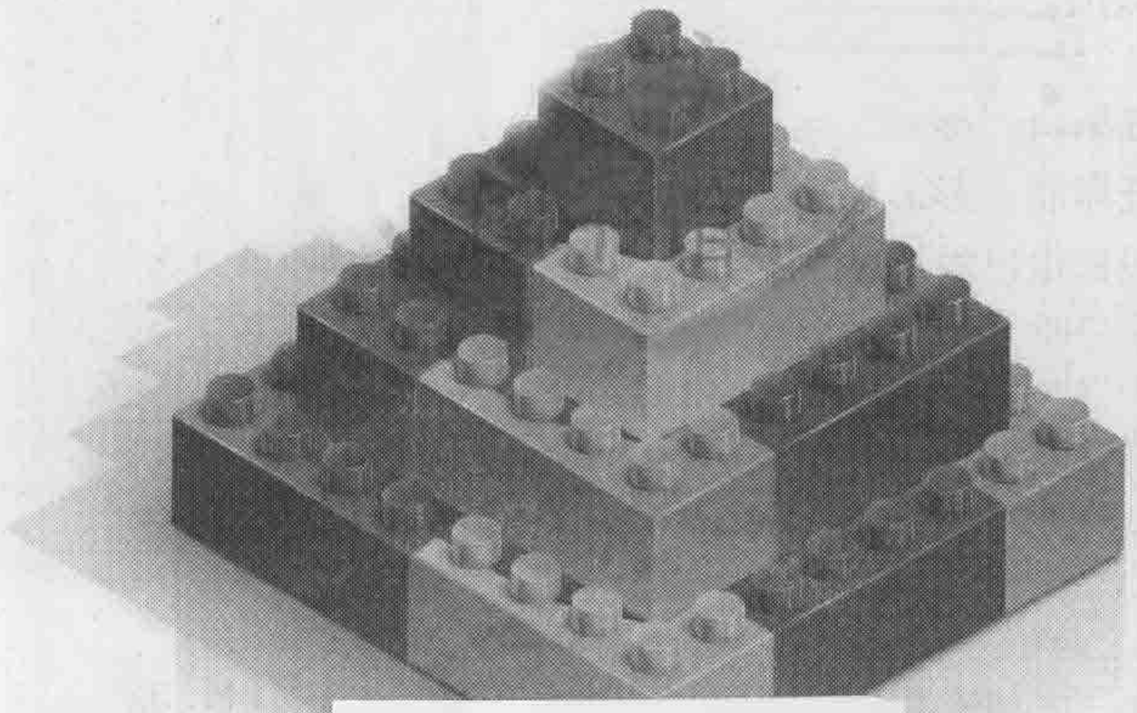
CENGAGE Learning

SQL

初学者指南

The Language of SQL

[美] Larry Rockoff 著　　李强 译



人民邮电出版社

北京

图书在版编目（CIP）数据

SQL初学者指南 / (美) 洛克夫 (Rockoff,L.) 著 ; 李强译. -- 北京 : 人民邮电出版社, 2014.11
ISBN 978-7-115-37122-5

Ⅰ. ①S… Ⅱ. ①洛… ②李… Ⅲ. ①关系数据库系统 Ⅳ. ①TP311.138

中国版本图书馆CIP数据核字(2014)第226637号

内 容 提 要

这是一本针对 SQL 初学者的图书。本书覆盖了所有核心的 SQL 概念，并且配以丰富的示例进行讲解。本书以直观和逻辑的顺序来组织主题，一次只介绍一个 SQL 关键字，新的关键字或概念是建立在之前理解的基础之上。本书介绍了 3 种广泛使用的数据库语法，它们是：Microsoft SQL Server、MySQL 和 Oracle。专门的“数据库的差异”板块则展示了 3 种数据库语法的不同之处。此外，书中还介绍了如何下载和安装这些数据库的免费版本。

本书适合 SQL 的初学者和初级的数据库管理员学习和参考，也可以作为高等院校相关专业的教学参考书。

◆ 著　[美] Larry Rockoff
译　李　强
责任编辑　陈冀康
责任印制　彭志环　焦志炜
◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷
◆ 开本：800×1000　1/16
印张：13.25
字数：247 千字　2014 年 11 月第 1 版
印数：1 – 3 000 册　2014 年 11 月北京第 1 次印刷
著作权合同登记号　图字：01-2013-9312 号

定价：39.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

版权声明

978-7-115-37122-5

Cengage Learning Asia Pte. Ltd.

151 Lorong Chuan, #02-08 New Tech Park, Singapore 556741

前　言

研究表明，为了节省时间大部分的读者都试图跳过前言，直接开始阅读正文。

了解了这一点，所以我们在前言中只介绍一些相对来说并不重要的内容，诸如说明通过阅读本书，你能够学到什么，以及无法学到什么。

转念一想，或许前言真的很重要，所以你不妨坚持一下。我们尽量让它简洁。

即便你对 SQL 还不是很熟悉，我们也可以说它是一种拥有很多组件和特性的、复杂的语言。本书将要关注的一个主要话题是：

- 如何使用 SQL 从数据库中检索数据。

此外，我们还会介绍：

- 如何在数据库中修改数据；
- 如何创建和维护数据库；
- 如何设计关系型数据库；
- 检索到数据之后，显示数据的策略。

以下几点特色使得本书在众多介绍 SQL 的书籍中独树一帜：

- 当你阅读本书时，无需下载软件或者使用计算机。

我们的目标是，让你直接阅读本书就能够理解所提供的 SQL 示例。书中包含了较小的数据示例，使你能够清晰地看到 SQL 语句是如何工作的。

- 使用基于语言的方法，让你像学习英语一样学习 SQL。

以直观和逻辑的顺序来组织主题。一次只介绍一个 SQL 关键字，当你遇到新的单词或概念时，它是建立在之前的理解的基础之上。

- 本书介绍了 3 种广泛使用的数据库的语法，它们是：Microsoft SQL Server、MySQL 和 Oracle。

如果这 3 种数据库之间有任何的差异，我会在本书的正文中介绍 Microsoft SQL Server 的

语法。专门的“数据库的差异”板块则描述和解释了 MySQL 或 Oracle 语法的不同之处。

- 重点介绍使用 SQL 检索数据的相关方面。

这种方法对于那些只需要结合报表工具使用 SQL 的人很有帮助。

此外，第 20 章介绍了检索到数据之后，显示数据的策略，包括如何使用交叉报表和透视表。

最后，我们在前言中介绍一个额外的问题：

SQL 是如何发音的？

实际上，有两种方法。一种方法是按照单个字母直接念出来，就像“S-Q-L”。另一种是像“sequel”一样发音。有人说这两种发音中只有一种是正确的，但是对于这个问题没有达成真正的共识。它基本上是个人喜好的问题。

对于字母 S-Q-L 的含义，大部分人同意它们表示“结构化查询语言（structured query language）”。然而，少数人却认为 SQL 根本没有含义，因为这门语言是派生自 IBM 的一门叫作 sequel 的老语言，sequel 并不表示结构化查询语言。

不管怎样，本书的前言就是这样了。现在，给出一些有用的信息。

配套网站下载指南

附录 D 介绍了本书的配套网站上可供使用的文件。这些文件列出了本书中所有的 SQL 语句并且提供了书中给出的所有数据。

你可以从 www.courseptr.com/downloads 下载配套网站上的文件。

作者简介

Larry Rockoff 多年从事商业智能和 SQL 的开发。他重点研究的领域是使用报表工具在复杂数据库中探索、提取和分析数据。他目前为 Dell 公司的子公司 ASAP 软件公司开发了一套 BI 工具。他从芝加哥大学获取了 MBA 的学位，他所学的专业是管理科学。

要了解作者的当前工作或与他联系，请访问 Larry Rockoff.com。

目　录

第1章
关系型数据库和SQL

在本章中，我们将介绍一些背景知识，以便于你能够很快地上手，能在后续的章节中编写SQL语句。本章有两个主题。首先是对本书所涉及到的数据库做一个概述，并且介绍和这些数据库是如何与SQL语言相关的。我还将介绍本书的特点，这能让你快速地决定，针对你正在使用的数据库，应该采用什么样的SQL语法。

其次，我们将介绍关系型数据库的一些关键的设计特点，并且会介绍表、行、列、键以及数据类型。在掌握了这些基本信息后，你马上就可以工作了。事不宜迟，让我们开始吧。

1.1 语言和逻辑

我必须承认，本书的书名并不是十分恰当。尽管本书的名称是“*The Language of SQL*”，但是用“*The Logic of SQL*”作为书名可能更恰当。这是因为，就像所有的计算机语言一样，SQL语言具有比英语词汇更严格、更固定的逻辑。

尽管如此，SQL拥有与众多其他计算机语言不同的、独特的基于语言的语法。和许多编程工具不同，SQL使用普通的英语单词，诸如WHERE、FROM和HAVING等，作为其语法中的关键字。因此，SQL可能会比你以往见过的其他语言少了很多神秘感。

在熟悉SQL语言后，你可能会发现，SQL命令的思维方式会和英语语句很类似，同样能表达某种含义。

例如，对比下面这句话：

```
I would like a hamburger and fries from your value menu,
and make it to go.
```

和这条SQL语句：

```
Select city, state
from customers
order by state
```

这条 SQL 语句表示我们想要从数据库的 customer 表中获取 city 和 state 字段，并且希望结果按照 state 来排序，具体的细节稍后介绍。

在这两个示例中，我们指定了想要的项（hamburger/fries 或 city/state），从哪里获取（value 菜单或 customer 表），以及一些额外的指令（整体处理或将结果按照 state 来排序）。

所以，本书的一个重要目标，就是用一种既简单又直观的方式来学习 SQL，就像你学习英语一样。我的方法是，每次介绍一个单词，同时构建起语言的逻辑用途和含义。

本书还有第二层意思，这可能并没有明确地在书名中表示出来。人们经常会把 SQL 语言和 SQL 数据库搞混。有许多销售数据库管理系统（Database Management Systems，DBMS）软件的公司。通常，这些类型的软件包中的数据库指的是 SQL 数据库，而 SQL 语言是管理和访问这些数据库中的数据的主要方法。一些厂商甚至把 SQL 作为其数据库名称的一部分。例如，Microsoft 把它最新的 DBMS 叫做 SQL Server 2008。

但实际上，更准确地讲，SQL 是一门语言，而不是一个数据库。本书的重点是介绍 SQL 的语言，而不是任何一种特定的数据库。

1.2 SQL 的定义

那么到底什么是 SQL 呢？简而言之，SQL 就是维护和使用关系型数据库中的数据的一种标准的计算机语言。简单来说，SQL 就是能让用户和关系型数据库进行交互的一种语言。SQL 语言有很长的发展历史，很多组织都对它的发展做出了贡献，它最早的历史可以追溯到 20 世纪 70 年代。1986 年，美国国家标准局（American National Standards Institute，ANSI）发布了该语言的第一套标准，从那时起，它经历过多次的修订。

一般来讲，SQL 语言有 3 个主要的组成部分。第 1 个部分叫做数据操纵语言（Data Manipulation Language，DML）。SQL 语言的这个模块让我们可以检索、修改、增加或删除数据库中的数据。第 2 个部分叫做数据定义语言（Data Definition Language，DDL）。DDL 使得我们能够创建和修改数据库本身。例如，DDL 提供了 ALTER 语句，它让我们可以修改数据库中的表的设计。第 3 个部分是数据控制语言（Data Control Language，DCL），用于维护数据库的安全。

许多主要的软件厂商，像 Microsoft 和 Oracle，为了各自的目的，都会修改这个标准，并且对该语言增加了大量的扩展和修改。尽管每个厂商对于 SQL 都有自己独特的解释，但是仍然会有底层的基础语言，它对于所有厂商几乎都是一致的。这正是本书所要介绍的内容。

作为一种计算机语言，与其他你可能熟悉的语言（如 Visual Basic 或 C++）相比，SQL 并不相同。其他语言本质上往往趋向于过程化。这就意味着，它们允许你指定特定的过程来完成想要实现的任务。SQL 更趋向于是一种声明式语言（Declarative Language）。在 SQL 中，经常用一条单独的语句来声明预期的目标。SQL 的结构之所以如此简单，是因为它只关注关系型数据库，而不是整个计算机系统。

1.3　Microsoft SQL Server、Oracle 和 MySQL

尽管我的目标是介绍 SQL 的核心语言，因为它适用于所有的实现，但是我也会提供 SQL 语法的一些具体示例。因为各个厂商的语法各异，所以我决定重点关注如下这 3 种数据库所使用的 SQL 语法：

- Microsoft SQL Server；
- Oracle；
- MySQL。

我会在本书的正文中介绍 Microsoft SQL Server 的语法。然后，如果这 3 种数据库之间有任何的差异，我会像下面这样，专门指出 MySQL 或 Oracle 的语法的不同之处。

数据库的差异

> 当我要介绍 Oracle 数据库或 MySQL 数据库中的不同的语法时，就会以这样的版块给出。Microsoft SQL Server 的语法将出现在正文中。
>
> 这个版块的标题将指明，这里的提示是针对 MySQL、Oracle 还是两者皆适用。

Microsoft SQL Server 有好几个可用的版本。最新的版本叫做 Microsoft SQL Server 2008。既有基础的 Express 版，又有功能齐全的企业（Enterprise）版。尽管 Express 版是免费的，但是它仍然有大量的功能，可以让你进行完整的数据库开发。企业版包括许多高级的数据库管理功能，以及高级的商务智能组件。

Oracle 也有多个可用的版本。最新的版本叫做 Oracle Database 11g。和 Microsoft 一样，

Oracle 也提供了一个免费的 Express 版的数据库。

MySQL 是一款开源的数据库，这意味着没有一家独立的机构拥有和控制它的开发。尽管 Sun Microsystems 于 2008 年收购了 MySQL，但它仍然是开源软件的首选之一。后来 Oracle 收购了 Sun Microsystems。作为一个开源数据库，除了 Windows 外，MySQL 还可以在许多平台上运行，诸如 Mac OS X 和 Linux。MySQL 提供了社区版本（Community Edition）供免费下载。

刚开始学习时，按你的选择下载数据库，有时候是很有用的，你可以去体验一下。但是，本书不要求你这么做。本书的编写方法是，允许你通过只阅读正文来学习 SQL。在正文中，我会提供足够的数据，你无需下载软件或亲自输入语句，也能理解各种 SQL 语句的结果。

尽管如此，如果你想要下载这些数据库的免费版本，本书的附录 A 到附录 C，针对如何下载给出了一些介绍和建议。附录 A 针对如何开始使用 Microsoft SQL Server 给出了详尽的说明，包括如何安装软件以及执行 SQL 命令的详细介绍。附录 B 介绍的是 MySQL，而附录 C 介绍的是 Oracle。

此外，附录 D 是辅助材料，它列出了本书中的 3 种数据库中的所有 SQL 语句。如前文所述，本书的正文中所有的 SQL 语句都是采用 Microsoft SQL Server 语法。大多数情况下，这些语句在 MySQL 和 Oracle 中也能工作，但是偶尔可能会略有不同。

大部分的读者会发现根本不需要下载软件，或者不需要阅读附录 D 中的辅助材料。本书中所有的示例，都是一看便知的，不需要为了理解这些内容而做任何事情。但是，如果你愿意这样做的话，那么就利用好这些额外的特性吧。

1.4 其他数据库

除了 Microsoft SQL Server、Oracle 和 MySQL 以外，还有很多其他的 SQL 数据库软件。一些比较流行的软件如下所示：

- Microsoft 的 Microsoft Access；
- IMB 的 DB2；
- IBM 的 Informix；
- Sybase 的 SQL Anywhere；

- 开源数据库 PostgreSQL。

在列出的这些数据库中，Microsoft Access 有一些特别，它对于那些想要学习 SQL 语言的新手特别有用。其实，Access 是关系型数据库的一个图形化界面。换句话讲，Access 允许我们完全通过图形化的方法，来为关系型数据库创建一个查询。对于初学者来讲，Access 最有用的一点，就是可以用可视化的方法创建一个查询，然后切换到 SQL 视图去查看刚创建的 SQL 语句。所以，我们可以尝试去做不同的事情，并且能够快速地看到相应的 SQL 语法是什么样子的。

与所列出的其他数据库相比，Access 的另一个不同之处在于，它是一个桌面数据库。因此，它有很强的灵活性。不仅可以使用 Access 来创建一个完全以单个的文件形式保存在计算机上的数据库；而且，还可以连接到用其他工具（诸如 Microsoft SQL Server）创建的更复杂的数据库。

1.5　关系型数据库

我们来了解一下关系型数据库的基础知识以及它们是如何工作的。

基本上，关系型数据库就是一个数据集合，它保存了许多个表。术语“关系（relational）”用来表示各表彼此相互关联。例如，我们来看数据库的一个简单示例，它只有两个表：Customers 表和 Orders 表。Customers 表为每位下订单的客户保存一条记录。Orders 表针对每个订单保存一条记录。每个表可以包含任意多个字段，字段用来存储与每条记录相关的不同属性。例如，Customers 表可以保存诸如 First Name 和 Last Name 这样的字段。

这时，可视化一些表和表中所包含的数据是很有用的。通常习惯是，把表显示为由行和列组成的一个表格。每一行表示表中的一条记录，每一列表示表中的一个字段。行头通常是字段名。剩余的其他行显示实际的数据。

在 SQL 术语中，记录（record）和字段（field）实际上就称为行（row）和列（column），这和视觉上的表现是对应的。因此，今后我们使用术语“行”和“列”来说明关系型数据库中表的设计，而不再使用记录和字段。

我们来看关系型数据库中一个可能是最简单的示例。在这个数据库中，只有两个表，分别是 Customers 表和 Orders 表。这两个表看上去如下所示。

Customers 表：

CustomerID	FirstName	LastName
1	William	Smith
2	Natalie	Lopez
3	Brenda	Harper

Orders 表：

OrderID	CustomerID	OrderAmount
1	1	50.00
2	1	60.00
3	2	33.50
4	3	20.00

在这个示例中，Customers 表包含了 3 个列：CustomerID、FirstName 和 LastName。目前，表中有 3 行，分别表示 William Smith、Natalie Lopez 和 Brenda Harper。每一行表示一个不同的客户，每一列表示该客户的一段不同的信息。与之类似，Orders 表有 4 行和 3 列。这表示数据库中有 4 笔订单，每笔订单有 3 种属性。

当然，这个示例非常简单，并且只是提示了哪些数据类型可以存储到一个真实的数据库中。例如，Customers 表通常会包含描述客户的其他属性的许多附加的列，诸如 city、state、ZIP 和 phone。同理，Orders 表一般也会有一些描述订单的其他属性的列，诸如 order date、sales tax 以及该订单的 salesperson。

1.6 主键和外键

请留意每个表的第一列：即 Customers 表中的 CustomerID 和 Orders 表中的 OrderID。这些列通常称为主键（primary key）。主键之所以有用和有必要，有两个原因。首先，它们使你能唯一地标识表中一个单独的行。例如，如果想要查找 William Smith 这一行，我们可以只使用 CustomerID 列来获取数据。主键还确保了唯一性。当指定 CustomerID 列作为主键时，就保证了表中的该列针对每一行都拥有一个唯一的值。即使在数据库中，有两个不同的人都叫 William Smith，这两行的 CustomersID 列的值也会不同。

在这个示例中，主键列的值没有任何特殊含义。在 Customers 表中，CustomerID 列在表的 3 行中的值分别为 1、2 和 3。通常情况下，我们会以这样的一种方式来设计数据库的表：当表中增加新的行时，主键列会自动生成顺序的编号。通常，我们把这种设计特性叫做自增型（auto-increment）。

使用主键的第 2 个原因是，可以很容易地把一个表和另一个表进行关联。在这个例子中，Orders 表中的 CustomerID 列，指向了 Customers 表中对应的一行。查看一下 Orders 表的第 4 行，会发现其 CustomerID 列的值是 3。这就意味着，CustomerID 为 3 的客户下了这个订单，这位客户名为 Brenda Harper。在表之间使用共同的列，这是关系型数据库中一项基本的设计要素。

除了可以指向 Customers 表，还可以把 Orders 表中的 CustomerID 列指定为外键（foreign key）。我会在第 18 章中详细地介绍外键，这里只需要知道，定义外键是要确保这一列有一个有效的值。例如，我们希望 Orders 表中的 CustomerID 列的值，必须是 Customer 表中真正存在的一个 CustomerID 值。指定一列作为外键，就可以实现这种限制。

1.7　数据类型

主键和外键为数据库表添加了结构。它们确保了数据库中所有的表都是可访问的，表之间有正确的关联。表中的每一列的另一个重要属性是数据类型。

数据类型是定义一个列所能包含数据的类型的一种方法。要为每个表中的每一列都指定一个数据类型。遗憾的是，各种关系型数据库所允许的数据类型和它们所代表的含义，有很大的不同。例如，Microsoft SQL Server、MySQL 和 Oracle，各自都有超过 30 种不同的可用数据类型。

即使只有 3 种数据库，我们都不可能去介绍每种可用的数据类型的细节及细微差别。但是，我所要做的，是通过讨论大部分数据库中常用的数据类型的主要类别，来概括这种情况。一旦了解了这些类别中的重要的数据类型，当遇到其他可能的数据类型时，也都可以迎刃而解。

一般来讲，有 3 种重要的数据类型：数字（Numeric）、字符（Character）以及日期/时间（Date/Time）。

数字数据类型有很多种，包括位（bit）、整数（integer）、浮点数（decimal）和实数（real

number）。bit 是数字类型，它只允许有两个值，0 和 1。bit 也经常用来定义只有 true 和 false 值的一个属性。integer 是没有小数点的数字。decimal 可以包含小数点。与 bit、integer 和 decimal 不同，实数的精确值只能是内部近似地定义。所有数字类型的一个共同的显著特征，就是它们都能用于算术运算中。如下是 Microsoft SQL Server、MySQL 和 Oracle 中的数字类型的一些典型示例。

通用说明	Microsoft SQL Server 数据类型	MySQL 数据类型	Oracle 数据类型	示例
bit	bit	bit	（none）	1
integer	int	int	number	43
decimal	decimal	decimal	number	58.63
real	float	float	number	80.62345

有时把字符类型称作 string 或 character string 类型。和数字类型不同，字符类型不再限定为数字。它们可以包括任意的字母、数字，甚至可以包括星号这样的特殊字符。当在 SQL 语句中为字符类型提供一个值时，总是需要使用单引号把这个值括起来。相比之下，数字类型就从不使用引号。如下是字符类型的一些典型示例。

通用说明	Microsoft SQL Server 数据类型	MySQL 数据类型	Oracle 数据类型	示例
可变长度	varchar	varchar	varchar2	'Thomas Edison'
固定长度	char	Char	char	'60601'

在第 2 个例子中，60601 看上去应该是一个数字类型，因为它是由数字组成的。这种情形并不常见。即便 ZIP 编码只包含数字，但通常还是把它定义成字符数据类型，因为不需要对它进行算术运算。

日期/时间类型是用来表示日期和时间的。就像字符类型一样，日期/时间类型也需要用单引号括起来。

这些数据类型允许对所涉及的日期进行特殊的运算。例如，我们可以使用一种特殊的方法，来计算任意两个日期之间的天数。如下是日期/时间类型的一些典型示例。

通用说明	Microsoft SQL Server 数据类型	MySQL 数据类型	Oracle 数据类型	示例
日期	Date	date	（none）	'2009-07-15'
日期和时间	Datetime	datetime	date	'2009-07-15 08:48:30'

1.8　空值

表中每个单独列的另一个重要属性是，该列是否允许包含空值。空值表示某个特定的数据元素没有数据。按照字面意思解释就是没包含数据。空值不等同于空格或空白。从逻辑上讲，空值和空格要区分对待。在第 8 章中，我们会详细介绍如何检索包含空值的数据。

许多 SQL 数据库在显示带有空值的数据时，使用大写的单词 NULL 来表示。这么做是要让用户能够识别它包含的是一个空值，而不是一个空格。我也会遵循这个惯例，在书中用 NULL 来强调它表示一个特殊类型的值。

数据库的主键不能包含 NULL 值。这是因为，按照定义，主键必须包含唯一的值。

1.9　SQL 的重要性

在我们离开关系型数据库的主题之前，为了让你对关系型数据库的优点和 SQL 的重要性有更深入的了解，我们来回顾一下历史。

回到计算机的石器时代（20 世纪 60 年代），人们通常把数据保存在磁带上，或者保存在磁盘存储器上的文件中。使用诸如 FORTRAN 和 COBOL 这样的语言编写的计算机程序，通常通过输入文件进行读取，并且一次只处理一条记录，最终将数据移动到输出文件。过程必然是很复杂的，因为需要把过程分解成多个单独的步骤，涉及临时表、排序以及多次数据传递，直到能够生成正确的输出。

到了 20 世纪 70 年代，随着分层和网络数据库的发明和使用，数据库取得了长足的发展。这些新的数据库，通过复杂的内部指针系统，使得读取数据更容易。例如，程序可以读取客户的记录，自动指向该客户的所有订单，然后指向每笔订单的所有详细信息。但是，基本上仍然是一次只能处理一条记录的数据。

在关系数据库之前，数据存储的主要问题不是如何存储数据，而是如何访问数据。当开发出 SQL 语言时，关系型数据库才真正取得了突破，因为它采用了一种全新的方法来访问数据。

和早期的数据检索方法不同，SQL 允许用户每次访问一大批的数据。通过一条语句，SQL 命令就能够检索或者修改多个表中的数千条记录。这就避免了很多的复杂性。当想要处理每一条记录时，计算机程序不再需要按照特定的顺序一次读取一条记录。过去需要数百行程序代码才能完成的任务，现在只需要几行代码就可以完成。

1.10 小结

本章介绍了关系型数据库的背景知识，以便我们能继续学习主要的话题，这些话题涉及从数据库中检索数据。我们已经讨论过关系型数据库的一些重要的特性，诸如主键、外键和数据类型。我们还介绍了数据中可能存在的 NULL 值。我们会在第 8 章中进一步讨论空值，在第 18 章中，再回到数据库维护的一般性主题，并在第 19 章中介绍数据库设计。

为什么和数据库设计相关的所有重要的主题，都放在了本书后边去介绍？在现实世界中，先要设计和创建数据库，然后才能检索数据。为什么在本书中，我没有遵循相同的顺序呢？简而言之，我发现不考虑数据库设计的细节，而直接投入到 SQL 的使用，会更有成效。事实上，数据库设计是一门艺术，也是一门科学。因此，在对检索数据的细节和细微差别有了一些认识之后，再来学习数据库的设计原理，会更有意义。因此，我们将暂时忽略如何设计数据库这个问题，从下一章开始，直接进入数据检索主题。

第2章 基本数据检索

关键字：SELECT、FROM

在本章中，我们将介绍 SQL 中最重要的主题：如何从数据库中检索数据。无论是在大企业还是小企业，SQL 开发人员最常遇到的需求就是报表需求。当然，把数据放入到数据库中也不是轻松的活儿。不过，一旦数据存在于数据库中，业务分析师的精力就转向他们所拥有的数据财富，以及他们希望从所有数据中获取有用的信息；而 SQL 语言的乐趣和意义就在于此。

本书所介绍的数据检索的重点，和现实世界中 SQL 开发人员所面临的需求密切相关。数据分析师并不关心数据是如何进入到数据库中的，但是他们很关心如何从数据库中取出某些内容。要帮助企业破解数据存入数据库的秘密，你还需要学习更多的 SQL 知识。

2.1 一条简单的 SELECT 语句

在 SQL 中，数据的检索可以通过 SELECT 语句来完成。无需太多解释，我们来看一条最简单的 SELECT 语句的示例：

```
SELECT * FROM Customers
```

和所有的计算机语言一样，在 SQL 中，有些单词是关键字。这些单词有特殊的意义，而且必须以特定的方法来使用。在这条语句中，单词 SELECT 和 FROM 是关键字。关键字 SELECT 表示你开始编写一条 SELECT 语句。

关键字 FROM，用来表示从哪个表中检索数据，表名紧紧跟在 FROM 之后。在这个示例中，表名是 Customers。

按照惯例，我们会把关键字都用大写字母印刷出来，这样就能确保它们足够醒目。

这个示例中的星号（*），是一种特殊的符号，它表示“所有的列”。

总结一下，这条语句的意思是：从 Customers 表中查找所有的列。

如果 Customers 表如下所示：

CustomerID	FirstName	LastName
1	William	Smith
2	Natalie	Lopez
3	Brenda	Harper

那么，这条 SELECT 语句将返回如下的数据：

CustomerID	FirstName	LastName
1	William	Smith
2	Natalie	Lopez
3	Brenda	Harper

换句话讲，它返回了表中的所有内容。

在第 1 章中，我们曾经介绍过，为所有表指定一个主键是一种通用做法。在前面的示例中，CustomerID 列就是该表的主键。我们还介绍过，有时当表中增加新的行时，主键能够自动按照数字序列产生一个顺序编号。前面的示例就是这种情况。在本书中，我们所展示的大部分示例数据，都会有一个类似的列，它既是主键，又定义为自增型的。按照惯例，该列通常是表的第一列。

2.2 语法注释

当编写 SQL 语句时，一定要记住两点。首先，SQL 语句中的关键字不区分大小写。单词 SELECT 等同于“select”或“Select”。

其次，可以把 SQL 语句写成任意多行。例如，SQL 语句：

```
SELECT * FROM Customers
```

等同于：

```
SELECT *
```

```
FROM Customers
```

把重要的关键字作为单独一行的开始，这通常是一个好主意。当遇到较为复杂的 SQL 语句时，这会让我们更容易快速掌握语句的含义。

最后，当我们在本书中介绍不同的 SQL 语句时，经常会既使用具体示例，又给出更为一般的格式。例如，前面这条语句的一般格式如下所示：

```
SELECT *
FROM table
```

斜体用来表示通用表达式。斜体的单词 *table*，表示在这里，你可以用自己的任意表名来替代该单词。所以，当你看到本书中的任何 SQL 语句中的斜体字时，那就是我想告诉你，请在这里放上任何有效的单词或短语。

数据库的差异：MySQL 和 Oracle

许多 SQL 实现要求在每条语句的末尾放一个分号（;）。这适用于 MySQL 和 Oracle，但是不适用于 Microsoft SQL Server。为了简单起见，本书中的 SQL 语句都没有分号。如果你使用 MySQL 或 Oracle，那么需要在语句的末尾加上一个分号。因此，上述语句应该是这样：

```
SELECT *
FROM Customers;
```

2.3　指定列

到目前为止，我们只是显示了一个表中的所有数据，其他什么也没做。但是，如果想只选择其中某些列，该怎么办呢？例如，还是使用同样的表，但是我们只想显示客户的姓氏。SELECT 语句如下所示：

```
SELECT LastName
FROM Customers
```

并且，会得到如下的结果数据：

LastName
Smith
Lopez
Harper

如果想要选择多个列，但不是所有列，SELECT 语句如下所示：

```
SELECT
FirstName,
LastName
FROM Customers
```

输出如下所示：

FirstName	LastName
William	Smith
Natalie	Lopez
Brenda	Harper

这条语句的一般格式是：

```
SELECT columnlist
FROM table
```

需要记住重要的一点，如果要在 columnlist 中指定多个列，那么这些列必须要用逗号隔开。还要注意，我们把 columnlist 中的每一列都放在单独的一行，这样有助于提高可读性。

2.4 带有空格的列名

如果列的名称中有空格，该怎么办？例如，假设 LastName 列的列名用的是 Last Name（两个单词间插入了一个空格）。显然，下面这条语句是无效的：

```
SELECT
Last Name
FROM Customers
```

因为 Last 和 Name 不是列名，所以这条语句被认为无效。况且，即使 Last 和 Name 是正确的列名，它们之间也需要用逗号隔开。解决方案就是，用特殊的字符把所有包含空格的列名括起来。根据所使用的数据库，所采用的字符也不同。对于 Microsoft SQL Server，使用的字符是方括号，正确的语法如下所示：

```
SELECT
[Last Name]
FROM Customers
```

关于语法，还有另外一点要注意：就像关键字不区分大小写一样，表名和列名也不区分大小写。例如，前面的示例等同于：

```
select
[last name]
from customers
```

为了清晰起见，在本书中，我们会把所有的关键字全部用大写形式印刷，把表名和列名用首字母大写的形式印刷，但是，并不是一定要这么做。

数据库的差异：MySQL 和 Oracle

在 MySQL 中，用重音符（`）把包含空格的列名括起来。上述示例的语法如下：

```
SELECT
`Last Name`
FROM Customers;
```

在 Oracle 中，用双引号把带有空格的列名括起来。示例语法如下：

```
SELECT
"Last Name"
FROM Customers;
```

此外，与 Microsoft SQL Server 和 MySQL 不同的是，在 Oracle 中，用双引号括起来的列名是区分大小写的。这就意味着，上述的语句不能等同于：

```
SELECT
"LAST NAME"
FROM Customers;
```

2.5　小结

在本章中，我们首先介绍了如何使用 SELECT 语句来检索数据。我们介绍了基本的语法，并且介绍了如何选择特定的列。但是，在实际工作中，这些还远远不够。特别是，我们还没有介绍如何针对数据检索应用各种类型的查询条件。例如，尽管我们知道如何选择所有的客户，但还不知道如何只选择纽约州的客户。

我们在第 7 章中才会介绍查询条件。那么接下来我们会做什么呢？在接下来几章中，我们会来看看使用 SELECT 语句的 columnlist 部分能够做些什么。在下一章中，我们会继续介

绍选取列的更多方法，从而能够在一个单独的列中实现复杂的运算。我们还会介绍重新命名列的方法，从而使列名更具有描述性。

在第 4 章到第 6 章中，我们会介绍创建更加复杂和功能更为强大的 columnlists 的方法，以便在开始介绍查询条件的主题时，你能够有足够的技术可供使用。

第3章
计算和别名

关键字：AS

本章所介绍的主题，可以让你用更方便和更有趣的格式，向查看数据的人展现信息。这里主要介绍的技术是计算字段（calculated fields）。这种技术允许我们对那些从数据库中检索到的单个数据项进行计算。

通过这种方法，我们可以把客户名称按照需要进行格式化，还可以针对业务或企业进行特定的数学运算和展示。作为一名 SQL 开发人员，我们经常需要能够定制单列的内容，以便能够成功地把数据转换成更加智能的内容。计算字段是非常有用的工具，它能够帮助我们实现这一目标。

3.1 计算字段

当从一个表中选择数据时，对于表中出现的列，我们并没有限制。计算字段的概念考虑到了许多其他的可能。使用计算字段，我们可以做如下的事情：

- 选择特定的单词或数值；
- 对单个或多个列进行计算；
- 把列和直接量组合在一起。

接下来，我们来看一些示例，这些示例都来自于如下的 Orders 表：

OrderID	FirstName	LastName	QuantityPurchased	PricePerItem
1	William	Smith	4	2.50
2	Natalie	Lopez	10	1.25
3	Brenda	Harper	5	4.00

3.2 直接量

计算字段的第一个示例实际上并不是一个计算。我们准备选择一个特定的值作为一个列，尽管这个直接量和表中的数据没有任何关系。这种类型的表达式叫做直接量（literal value）。示例如下：

```
SELECT
'First Name: ',
FirstName
FROM Orders
```

这条语句返回的数据如下所示：

(no column name)	FirstName
First Name:	William
First Name:	Natalie
First Name:	Brenda

在这条语句中，我们选择了两个数据项。首先是直接量'First Name:'。请注意，单引号用来表示这是一个字符直接量。其次是 FirstName 列。

我们需要注意两点。首先，直接量'First Name:'在每一行中都会重复出现。其次，第 1 列没有表头信息。当在 Microsoft SQL Server 中运行时，列的表头显示为“（no column name）”。没有表头的原因很简单，因为这是一个计算字段，所以没有列名可以用做表头。

数据库的差异：MySQL 和 Oracle

MySQL 和 Oracle 都会在表头行为直接量返回一个值。在 MySQL 中，会把上述示例的第 1 列的表头显示成：

```
First Name:
```

在 Oracle 中，会把第 1 列的表头显示成：

```
'FIRSTNAME:'
```

你可能会问一个问题，为什么表头行如此重要。如果使用 SELECT 语句把返回的数据带入到计算机程序中，那么你可能不需要关心这个表头了。因为，我们只需要数据。但是，如果用 SELECT 语句检索到的数据，是要作为报表展示给用户，显示在纸张上或者屏幕上，那

么表头就至关重要了。毕竟，当用户查看一列数据时，他们通常都想知道这一列的含义。对于直接量来讲，这一列确实没有含义，所以表头也不是那么必要。但是，在其他类型的计算字段中，这个列需要一个有意义的标签。在本章后面，我们会介绍列的别名这一概念，别名是在这种情况下为列提供表头的一种方法。

除了为没有表头的列提供表头以外，列的别名还可以把列的名称改得更有意义，从而方便用户查看数据。例如，一个数据库设计者，可能会给 last name 列起一个含糊不清的名称，诸如 LstNm222。在这种情况下，可以采用列的别名，将其改为一个更具描述性的名称。

关于直接量，我们还有一点需要介绍。你可能认为所有的直接量都需要加上引号，但并不一定要这样。例如，如下所示的语句：

```
SELECT
5,
FirstName
FROM Orders
```

会返回如下数据：

(no column name)	FirstName
5	William
5	Natalie
5	Brenda

尽管直接量值 5 完全没有意义，但它仍然是一个合法的值。

由于没有引号，那么就可以把 5 当成是一个数字。

3.3　算术运算

我们来看一个更典型的计算字段的示例。算术运算使得我们可以对表中一个或多个列进行计算。例如：

```
SELECT
OrderID,
QuantityPurchased,
PricePerItem,
QuantityPurchased * PricePerItem
FROM Orders
```

将返回如下数据：

OrderID	QuantityPurchased	PricePerItem	(no column name)
1	4	2.50	10.00
2	10	1.25	12.50
3	5	4.00	20.00

上述 SELECT 语句的前 3 列同之前的示例相比没有任何变化。第 4 列是一个使用算术表达式计算的列：

```
QuantityPurchased * PricePerItem
```

在这个示例中，星号是一个数学符号，它表示乘法。它的含义同上一章中所见到的表示“所有的列”的星号有所不同。除了星号以外，还可以使用许多其他的算术运算符。下面是一些最常用的算术运算符：

Arithmetic Operator	Meaning
+	addition
–	subtraction
*	multiplication
/	division

还需要注意的是，和直接量一样，第 4 列也没有表头，因为它并非派生自一个单独的列。

3.4 连接字段

连接（Concatenation）是一个有趣的计算机术语，它表示把字符数据组合或连接到一起。就像算术运算符能够处理数字数据一样，字符数据也可以连接到一起。连接的语法，取决于所使用的数据库。如下是 Microsoft SQL Server 中的示例：

```
SELECT
OrderID,
FirstName,
LastName,
FirstName + ' ' + LastName
FROM Orders
```

检索到的数据如下所示：

OrderID	FirstName	LastName	(no column name)
1	William	Smith	William Smith
2	Natalie	Lopez	Natalie Lopez
3	Brenda	Harper	Brenda Harper

前 3 列依旧没有什么新的内容。第 4 列的表达式如下：

```
FirstName + ' ' + LastName
```

这里的加号表示连接。既然这个操作涉及的是字符而不是数字，SQL 会足够聪明，知道加号表示连接而不是表示加法。在这个示例中，连接表达式由 3 个部分组成：FirstName 列、空格直接量（''）和 LastName 列。空格直接量是必要的，以免把 William Smith 显示成 WilliamSmith。

数据库的差异：MySQL 和 Oracle

MySQL 不用符号（诸如+）来表示连接，而是需要调用一个名为 CONCAT 的函数来做连接。我们会在下一章中介绍这个函数，但是在这里，我们先来看看在 MySQL 中等价语句的形式：

```
SELECT
OrderID,
FirstName,
LastName,
CONCAT (FirstName, ' ', LastName)
FROM Orders;
```

Oracle 使用两根竖线（||）而不是加号（+）来表示连接。在 Oracle 中，等价的语句如下所示：

```
SELECT
OrderID,
FirstName,
LastName,
FirstName || ' ' || LastName
FROM Orders;
```

3.5　列的别名

在本章前面所有的示例中，我们看到的计算字段都没有描述性的表头。我们现在准备解

决这个问题，为这些类型的列指定标题。解决方法是使用列的别名。别名（alias）本意是指替代名称。下面的示例展示了如何对前面的 SELECT 语句的 Microsoft SQL Server 版本，使用列的别名：

```
SELECT
OrderID,
FirstName,
LastName,
FirstName + ' ' + LastName AS 'Name'
FROM Orders
```

请注意，列的别名'Name'要用单引号括起来。输出结果如下：

OrderID	FirstName	LastName	Name
1	William	Smith	William Smith
2	Natalie	Lopez	Natalie Lopez
3	Brenda	Harper	Brenda Harper

现在，第 4 列就有了表头。使用关键字 AS 来指定一个列的别名，别名紧跟在该关键字之后。

数据库的差异：MySQL 和 Oracle

在 MySQL 中，等价的语句如下所示：

```
SELECT
OrderID,
FirstName,
LastName,
CONCAT (FirstName, ' ', LastName) AS 'Name'
FROM Orders;
```

在 Oracle 中，不需要用单引号把列的别名括起来。但是，如果列的别名包含了嵌套的空格，那么要使用双引号把列的别名括起来。在 Oracle 中，相同的语句如下所示：

```
SELECT
OrderID,
FirstName,
LastName,
FirstName || ' ' || LastName AS Name
FROM Orders;
```

除了为计算字段提供标题，如果你想更换表中一个不好理解的列名，也经常会用到列的别名。例如，假设一个表有一个名为 Qty 的列，我们可以使用如下的语句，把该列显示为

Quantity Purchased:

```
SELECT
Qty AS 'Quantity Purchased'
FROM table
```

3.6　表的别名

除了为列提供替换的名称以外，也可以使用相同的关键字 AS，来为表指定别名。使用表的别名一般有 3 个原因。

首先，是针对那些不好理解或者复杂的表名。例如，假设有一个名为 Orders123 的表，我们可以使用如下的 SELECT 语句，给它一个 Orders 别名。

```
SELECT
LastName
FROM Orders123 AS Orders
```

与列的别名不同，表的别名不需要用引号括起来。在使用表的别名时，也可以选择使用表的别名作为任何选中列的前缀。例如，前面的示例也可以这样编写：

```
SELECT
Orders.LastName
FROM Orders123 AS Orders
```

现在，添加了 Orders 作为 LastName 的前缀，使用点来分隔前缀和列名。在这种情况下，前缀不是必需的。但是，当从多个表中选取数据时，有时是需要前缀的。在后边的章节中，我们会看到这种情形。

数据库的差异：Oracle

在 Oracle 中，指定表的别名的时候，不使用关键字 AS。在 Oracle 中，上述语句的语法如下：

```
SELECT
Orders.LastName
FROM Orders123 Orders;
```

在第 11 章和第 14 章中，我们会分别介绍使用表的别名的另外两个原因，它们是：

- 从多个表中进行选择的情形；

- 在一条 SELECT 语句中使用子查询的情形。

在第 14 章中，子查询（subquery）这个术语的含义将会变得明了。

3.7 小结

在本章中，我们学习了在 SELECT 语句中创建计算字段的 3 种主要方法。首先，用直接量来选择具体的单词或数值。其次，对一列或多列执行算术运算。最后，用连接把列和直接量组合在一起。我们还介绍了列的别名的相关主题，当使用计算字段时，经常会用到列的别名。

在下一章中，我们将介绍函数这个主题，它提供了更加复杂的方法来进行运算。如前所述，现在还没有到针对语句应用查询条件的时候。我们仍然还是在介绍使用 SELECT 语句的 columnlist 所能做的基本工作。别着急，我们很快就要接触到令人兴奋的内容了。同时，你对这种有条不紊的学习方法所付出的耐心，很快也会得到回报。

第4章
使用函数

关键字：LEFT、RIGHT、SUBSTRING、LTRIM、RTRIM、CONCAT、UPPER、LOWER、GETDATE/NOW/CURRENT_DATE、DATEPART/DATE_FORMAT、DATEDIFF、ROUND、RAND、PI、CAST 和 ISNULL/IFNULL/NVL

对于 Microsoft Excel 这样为人们所熟知的电子表格软件，其中的函数为典型的电子表格用户提供了大量的功能。如果不会使用函数，那么电子表格中大部分可用数据的价值就会大大减少。在 SQL 的世界也是这样的。熟悉那些最常用的 SQL 函数，将会大大增强你创建动态结果的能力，从而为那些使用你报表的人提供更好的服务。

本章会介绍多种最常用的函数，这些函数分为四种类型：字符函数、日期/时间函数、数值函数和转换函数。此外，我们还会介绍复合函数（Composite function），它是一种把多种函数组成一个表达式的方式。

4.1 函数的作用

和第 3 章所介绍过的计算类似，函数提供了另一种方法来操作数据。正如我们所看到的，计算通常使用算术运算符（如乘号或者连接）把多个字段包含进来。相比之下，函数则常常对单个的一列进行操作。

函数是什么？函数只是使用特定的公式把一个（或一些）值转换成另一个值的一种规则。例如，可以用 SUBSTRING 函数来确定名称 JOAN 的首字母是 J。有两种类型的函数：标量函数（Scalar function）和聚合函数（Aggregate function）。标量（Scalar）这个词来源于数学，指的是只针对单个数字进行运算。在计算机领域中，它表示针对单行中的数据执行该函数。例如，LTRIM 函数用于删除一个特定值中的起始空格。

相反，聚合函数要对一个较大的数据集合进行操作。例如，SUM 函数可以用来计算一个特定的列中所有值的总合。由于聚合函数应用于数据集合或数据组，所以我们将在第 10 章中介绍它们。

每种 SQL 数据库都会提供几十个标量函数。在不同数据库之间，函数的差异非常大，既有名称的差异也有工作方式的差异。因此，我们只会介绍最有用的函数中的一些有代表性的示例。

最常用的标量函数可以分为 3 种类型：字符函数、日期/时间函数和数值函数。显然，这些函数允许我们操作字符类型、日期/时间类型和数字类型。

此外，我们将介绍一些有用的转换函数，可以用它们把数据从一种类型转换成另一种类型。

4.2 字符函数

字符函数是那些能够操作字符数据的函数。正如人们有时把字符类型称作 string 类型，人们有时也会把字符函数称作 string 函数。我们将介绍如下的 8 个字符函数的示例：LEFT、RIGHT、SUBSTRING、LTRIM、RTRIM、CONCAT、UPPER 和 LOWER。

在本章中，我们将直接使用带有直接量的 SELECT 语句，而不会从特定的表中检索数据。我们从第一个示例，即 LEFT 函数开始。当执行如下的 SQL 命令时：

```
SELECT
LEFT ('sunlight',3) AS 'The Answer'
```

它会返回如下的数据：

The Answer
sun

因为语句中包含了一个列的别名，所以得到的数据看上去很棒。请注意，在这条 SELECT 语句中，没有 FROM 子句。我们从一个直接量（即'sunlight'）中查询数据，而不是从一个表中检索数据。严格来讲，SELECT 语句中不一定要有 FROM 子句，而且在实际工作中，也很少编写像这样的 SELECT 语句。我用这种不带 FROM 子句的方式编写 SELECT 语句，只是因为它更易于简洁明了地说明函数是如何工作的。

数据库的差异：Oracle

与 Microsoft SQL Server 和 MySQL 不同，Oracle 要求在所有的 SELECT 语句中都要有 FROM 子句。如果在 Oracle 中运行，本章中的所有示例都需要添加一条 FROM 子句。但是，在 FROM 子句中提供的表，不一定需要是一个真正的表。Oracle 提供了一个名为 DUAL 的虚拟表，DUAL 表的使用将在本章稍后介绍。

我们来仔细看一下这个函数的格式。LEFT 函数的一般格式如下：

```
LEFT (CharacterValue, NumberOfCharacters)
```

函数可以有任意多个的参数，并且要把参数放在圆括号中。例如，前面的 LEFT 函数有两个参数：CharacterValue 和 NumberOfCharacters。参数（argument）是一个常用的数学术语，它是函数的组成部分，当参数不正确时，函数是无法工作的。基本上，每个函数都是唯一的，并且为每个函数所定义的不同参数，才真正地确定了函数的意义。在 LEFT 函数示例中，当调用该函数时，参数 CharacterValue 和 NumberOfCharacters 都需要提供，才能确定该函数做些什么事情。

LEFT 函数有两个参数，而其他函数则可能有更多或更少的参数，甚至允许函数不带参数。但是，即使没有参数，函数也要有一组圆括号跟在关键字后面。圆括号告诉我们，这是一个函数而不是其他的东西。

LEFT 函数的计算公式的含义是：从指定的 CharacterValue 中，从左边开始查看 NumberOfCharacters 个字符，并且返回结果。在前面的示例中，它在 CharacterValue 即‘sunlight’中，查找最左边的 3 个字符，并返回结果“sun”。

要记住的一点是，对于任何想要使用的函数，你需要在数据库的参考指南中查找该函数，以确定它需要多少个参数以及各个参数的含义是什么。

第 2 个要介绍的字符函数是 RIGHT 函数。它和 LEFT 函数类似，只不过它从右边开始指定字符。例如：

```
SELECT
RIGHT ('sunlight',5) AS 'The Answer'
```

返回结果如下：

The Answer
light

在这个示例中，我们需要指定 5 作为数字参数。如果使用的数字是 3 而不是 5，返回的结果就会是“ght”。

数据库的差异：Oracle

Oracle 并没有提供 LEFT 函数或 RIGHT 函数。在 Oracle 中，SUBSTR 函数提供了相同的功能，本章稍后会介绍该函数。

你需要了解的是字符数据经常在其右边包含空格。让我们来看一个示例，在只有一行的一个表中，包含了一个名为 President 的列，把它定义为 20 个字符的长度。

President
George Washington

如果针对这个表，执行如下的 SELECT 语句：

```
SELECT
RIGHT (President,10) AS 'Last Name'
FROM table1
```

那么得到的返回数据如下：

Last Name
hington

我们想得到“Washington”，但是却只得到了“hington”。为什么会这样？因为整个列的长度是 20 个字符。因此，“George Washington”的右边会有 3 个空格。所以，当我们要求得到最右边的 10 个字符时，会得到 3 个空格加上“George Washington”中的另外 7 个其他字符。我们很快会看到，要得到想要查找的数据，需要使用 RTRIM 函数。

你可能想知道如何从一个值的中间选择数据。可以使用 SUBSTRING 函数来实现它。这个函数的一般格式如下：

```
SUBSTRING (CharacterValue, StartingPosition, NumberOfCharacters)
```

例如：

```
SELECT
SUBSTRING ('thewhitegoat', 4, 5) AS 'The Answer'
```

返回如下数据：

The Answer
white

这个函数从第 4 个字符开始，选取了 5 个字符。第 4 个字符是 w，所以最后得到的结果是单词“white”。

数据库的差异：MySQL 和 Oracle

MySQL 有时要求函数名称和左括号之间不能有空格。这取决于所使用的具体函数。例如，在 MySQL 中，上述语句等同于：

```
SELECT
SUBSTRING('thewhitegoat', 4, 5) AS 'The Answer';
```

在 Oracle 中，和 SUBSTRING 函数等价的是 SUBSTR。Oracle 版的 SUBSTR 的区别之一是，第 2 个参数（StartingPosition）可以是负数。这个参数是负数的时候，表示需要从列的右边开始计数。

如前所述，Oracle 不允许编写没有 FROM 子句的 SELECT 语句。但是，Oracle 为这种情形提供了一个名为 DUAL 的虚拟表，一条等价的带有 SUBSTRING 函数的 SELECT 语句如下：

```
SELECT
SUBSTR ('thewhitegoat', 4, 5) AS "The Answer"
FROM DUAL;
```

接下来，我们要介绍两个函数，它们可以删除一个值中所有的空格，既可以从左边开始删除，也可以从右边开始删除。LTRIM 函数就是从字符的左侧来“修剪”字符。例如：

```
SELECT
LTRIM ('the apple') AS 'The Answer'
```

返回数据如下：

The Answer
the apple

LTRIM 函数可以删除“the apple”左边的空格。请注意，LTRIM 足够聪明，不会删除句

子中间的空格，只会删除字符左边的空格。

同理，RTRIM 函数可以删除字符右边的空格。在下一节介绍复合函数时，我们会给出 RTRIM 的示例。

接下来要介绍的字符函数是CONCAT。这里介绍的CONCAT函数只能在MySQL和Oracle中使用。在第 3 章中，我们介绍过，在 Microsoft SQL Server 中，使用加号（+）运算符来处理连接。

让我们看一下第 3 章中的连接示例。输入数据都来自于如下的 Orders 表：

OrderID	FirstName	LastName	QuantityPurchased	PricePerItem
1	William	Smith	4	2.50
2	Natalie	Lopez	10	1.25
3	Brenda	Harper	5	4.00

在 MySQL 中，用 CONCAT 函数连接 FirstName 列和 LastName 列的语法如下：

```
SELECT
OrderID,
FirstName,
LastName,
CONCAT (FirstName, ' ', LastName) AS 'Name'
FROM Orders
```

在这个示例中，CONCAT 函数连接了 3 个数值：FirstName 列、一个空格直接量和 LastName 列。上述语句的结果如下所示：

OrderID	FirstName	LastName	Name
1	William	Smith	William Smith
2	Natalie	Lopez	Natalie Lopez
3	Brenda	Harper	Brenda Harper

数据库的差异：Oracle

Oracle 版的 CONCAT 函数只允许带有两个参数。换句话讲，一次只能连接两个值。为了实现上述示例中的连接，必须使用如下的一条语句：

```
SELECT
OrderID,
```

```
FirstName,
LastName,
CONCAT (CONCAT (FirstName, ' '), LastName) AS "Name"
FROM Orders;
```

这条语句用到了复合函数，我们会在下一节介绍相关概念。在这个示例中，里边的CONCAT 函数连接了 FirstName 列和一个空格直接量，外边的 CONCAT 函数将其结果与 LastName 列连接了起来。

最后要介绍的两个字符函数是 UPPER 和 LOWER。这两个函数把任意的单词或短语转换成全部大写或全部小写。在显示数据时，这两个函数有时会很有用。它们的语法简单而又直接。

如下是这两个函数的示例：

```
SELECT
UPPER ('Abraham Lincoln') AS 'Convert to Uppercase',
LOWER ('ABRAHAM LINCOLN') AS 'Convert to Lowercase'
```

输出如下：

Convert to Uppercase	Convert to Lowercase
ABRAHAM LINCOLN	abraham lincoln

4.3 复合函数

函数的一个重要特性是，无论它们是字符函数、数值函数还是日期/时间函数，都可以把两个或两个以上的函数组合成一个复合函数。由两个函数组成的复合函数，可以称为函数的函数。还是以 George Washington 来举例说明，我们仍然使用如下数据：

President
George Washington

请记住，President 列的长度是 20 个字符。换句话讲，在 George Washington 这个值的右边有 3 个空格。除了说明复合函数，接下来的示例也会使用到前面介绍过的 RTRIM 函数。

SELECT 语句如下：

```
SELECT
RIGHT (RTRIM (President),10) AS 'Last Name'
FROM table1
```

返回数据如下：

Last Name
Washington

为什么这次得到了正确的结果呢？让我们来看一下它是如何工作的。这条 SELECT 语句涉及到两个函数：RIGHT 和 RTRIM。当执行复合函数时，总是从里边开始，然后向外执行。在这个示例中，最里边的函数是：

```
RTRIM (President)
```

这个函数接受 President 列的值作为参数，并且把右侧的空格全部删除。做完这些后，对该结果应用 RIGHT 函数，以返回预期的值。因为：

```
RTRIM (President)
```

等同于“George Washington”，也可以把如下的 SELECT 语句：

```
SELECT
RIGHT (RTRIM (President),10)
```

看成如下：

```
SELECT
RIGHT ('George Washington', 10)
```

换句话讲，你可以先对输入数据应用 RTRIM 函数，然后把这个表达式放入到 RIGHT 函数中。

数据库的差异：Oracle

如上所述，在 Oracle 中，需要使用 SUBSTR 函数，而不是像在 Microsoft SQL Server 和 MySQL 中一样使用 RIGHT 函数。在 Oracle 中，上述语句的等效形式是：

```
SELECT
SUBSTR (RTRIM (President), -10, 10) AS "Last Name"
FROM table1;
```

4.4　日期/时间函数

日期/时间函数允许我们操作日期值和时间值。这些函数的名称会因为所使用的数据库的不同而各不相同。在 Microsoft SQL Server 中，我们将介绍的函数有：GETDATE、DATEPART 和 DATEDIFF。

最简单的日期/时间函数是返回当前的日期和时间。在 Microsoft SQL Server 中，该函数名为 GETDATE。这个函数没有参数，它只是返回当前的日期和时间。例如：

```
SELECT GETDATE ( )
```

它会返回当前日期和时间的一个表达式。由于 GETDATE 函数没有参数，所以括号中是空的。请记住，日期/时间字段是一种特殊的数据类型，它在一个单个的字段中既包含了日期也包含了时间。示例如下：

```
2009-07-15 08:48:30
```

这个值指的是 2009 年 7 月 15 日上午 8 点 48 分 30 秒。

数据库的差异：MySQL 和 Oracle

在 MySQL 中，和 GETDATE 等价的函数是 NOW。在 Oracle 中，等价的函数是 CURRENT_DATE。

接下来要介绍的日期/时间函数，能够分析任意具体的日期，并且返回诸如该日期是该月中的第几天或者是该年份的第几个周等这样的信息。该函数的名称也取决于所使用的数据库，在 Microsoft SQL Server 中，这个函数叫做 DATEPART，其一般格式如下所示：

```
DATEPART (DatePart, DateValue)
```

参数 DateValue 是任意的日期。参数 DatePart 可以是许多不同的值。如下都是有效的值：year、quarter、month、dayofyear、day、week、weekday、hour、minute 和 second。

下面显示了对于数据‘7/2/2009’，执行带有不同的 DatePart 参数的 DATEPART 函数后所得到的结果：

DATEPART Function Expression	Resulting Value
DATEPART (month, '7/2/2009')	7
DATEPART (day, '7/2/2009')	2
DATEPART (week, '7/2/2009')	27
DATEPART (weekday, '7/2/2009')	5

查看上面的值，我们会发现‘7/2/2009’的 month 是 7，day 是 2；week 是 27，因为 2009 年 7 月 2 日是一年中的第 27 周；weekday 是 5，因为 2009 年 7 月 2 日是星期四，而这是一周中的第 5 天。

数据库的差异：MySQL 和 Oracle

在 MySQL 中，同 DATEPART 等价的函数名为 DATE_FORMAT，并且它使用不同的值作为 DateValue 参数。例如，在 MySQL 中执行如下的 SELECT 语句，可以返回‘7/2/2009’这个日期是该年份中第几天：

```
SELECT DATE_FORMAT('2009-07-02', '%d');
```

Oracle 中没有和 DATEPART 等价的函数。

最后要介绍的日期/时间函数可以让我们得到任意两个日期之间相差的天数（或周数、月数等）。这个函数的名称也根据数据库的不同而不同。在 Microsoft SQL Server 中，这个函数名为 DATEDIFF，其一般的格式如下：

```
DATEDIFF (DatePart, StartDate, EndDate)
```

对于这个函数的 DatePart 参数来说，有效值包括：year、quarter、month、dayofyear、day、month、hour、minute 和 second。

下图展示了用 DATEDIFF 函数计算‘8/14/2009’和‘8/14/2009’之间相差的结果，注意，我们分别使用了不同的值作为 DatePart 参数：

DATEDIFF Function Expression	Resulting Value
DATEDIFF (day, '7/8/2009', '8/14/2009')	37
DATEDIFF (week, '7/8/2009', '8/14/2009')	5
DATEDIFF (month, '7/8/2009', '8/14/2009')	1
DATEDIFF (year, '7/8/2009', '8/14/2009')	0

上图表明了这两个日期之间相差 37 天、5 周、1 个月和 0 年。

数据库的差异：MySQL 和 Oracle

在 MySQL 中，DATEDIFF 函数只允许我们计算两个日期之间的天数，如果想要返回一个正数，结束的日期通常要作为第一个参数。其一般格式如下：

```
DATEDIFF (EndDate, StartDate)
```

Oracle 中没有和 DATEDIFF 等价的函数。

4.5　数值函数

数值函数允许操作数字，有时也被称为数学函数。我们将介绍的数值函数有：ROUND、RAND 和 PI。

ROUND 函数允许我们对任意数值进行四舍五入。其一般格式如下所示：

```
ROUND (NumericValue, DecimalPlaces)
```

参数 NumericValue 既可以是正数也可以是负数，既可以带小数也可以不带小数，诸如 712.863 和−42。

参数 DecimalPlaces 有点复杂，它可以是正整数、负整数或零。如果 DecimalPlaces 是正整数，它表示将数字四舍五入到指定的那么多个小数位；如果 DecimalPlaces 是零，它表示没有小数部分；如果 DecimalPlaces 是一个负整数，它表示对小数点左侧前几位进行四舍五入。下面展示了如何对数字 712.863 进行四舍五入，不同的 DecimalPlaces 参数，所得到的结果不同：

ROUND Function Expression	Resulting Value
ROUND (712.863, 3)	712.863
ROUND (712.863, 2)	712.86
ROUND (712.863, 1)	712.9
ROUND (712.863, 0)	713
ROUND (712.863, −1)	710
ROUND (712.863, −2)	700

RAND 函数用来产生随机数。什么是随机数？为什么需要产生随机数？当选择一个随机事件时，需要这种类型的操作，例如要从参加比赛的客户中选取获胜者。该函数的一般格式如下：

```
RAND ([seed])
```

参数 seed 外边的方括号表示这是一个可选的参数。根据是否提供 seed 参数，RAND 函数的操作会有所不同。在大多数情况下，不需要 seed 参数。如果没有使用该参数，那么 RAND 函数会返回 0 到 1 之间的一个随机数。如果在一行中执行 RAND 函数 10 次，它会返回 10 个不同的数字。SELECT 语句如下所示：

```
SELECT
RAND ( ) AS 'Random Value'
```

如果指定了 seed 参数，该参数需要是一个整数。当执行带有一个 seed 参数的 RAND 函数时，每次将返回相同的值。SELECT 语句如下所示：

```
SELECT
RAND (100) AS 'Random Value'
```

当改变参数 seed 的值时，它将返回一个不同的数值。

PI 函数只会返回数学运算中的 pi 的值（回忆一下你在几何课上所学的内容）。实际上，这个函数很少会用到，但是它很好地说明了数值函数可以不需要有任何参数。例如，如下语句：

```
SELECT PI ( )
```

会返回数值 3.14159265358979。

数据库的差异：Oracle

Oracle 中没有和 RAND 或 PI 等价的函数。

如果你想要把 pi 四舍五入到两位小数，该怎么做呢？很简单，你只需要用 PI 函数和 ROUND 函数创建一个复合函数即可。首先使用 PI 函数来得到一个初始值，然后使用 ROUND 函数对它进行两位小数的四舍五入。如下语句返回的数值是 3.14：

```
SELECT ROUND (PI ( ), 2)
```

4.6 转换函数

所有上述函数都是使用特定的方法操作字符类型、日期/时间类型或数字类型。但是，你可能需要把数据从一种类型转换成另一种类型，或者把 NULL 值转换成有意义的内容。本章剩余部分将介绍用于这种情况的两个特殊函数。

CAST 函数允许我们把数据从一种类型转换成另一种类型。该函数的一般格式如下：

```
CAST (Expression AS DataType)
```

实际上，在许多情况下都不需要用到 CAST 函数。假设我们要执行如下的语句，其中的 Quantity 列被定义为字符列：

```
SELECT
2 * Quantity
FROM table
```

你可能认为这条语句会失败，因为 Quantity 没有定义成一个数字列。但是，大多数 SQL 数据库足够聪明，能够自动把 Quantity 列转换成数字列，所以可以把它乘以 2。

如下是需要使用CAST函数的一个示例。假设我们把一列日期保存到了一个字符列中，我们想要把这些日期转换成真正的日期/时间列，如下语句说明了 CAST 函数如何进行这一转换：

```
SELECT
'2009-04-110 AS 'Original Date',
CAST ('2009-04-110 AS DATETIME) AS 'Converted Date'
```

输出结果是：

Original Date	Converted Date
2009-04-11	2009-04-11 00:00:00

Original Date 列看上去像是一个日期，但实际上它是字符数据。相反，Converted Date 列是一个真正的日期/时间列，现在所显示出来的时间值可以证明这点。

数据库的差异：Oracle

在 Oracle 中，与上述 CAST 函数等价的语句是：

```
SELECT
'2009-04-110 AS "Original Date",
CAST ('11-APR-20090 AS DATE) AS "Converted Date"
FROM DUAL;
```

另一个有用的转换函数也可以把 NULL 值转换成一个有意义的值。在 Microsoft SQL Server 中，把这个函数叫做 ISNULL。

正如第 1 章所介绍的，NULL 值是那些没有数据的值，NULL 值和空格或零不同。假设我们有如下的一个关于产品的表：

Product	Description	Color
1	Chair A	Red
2	Chair B	NULL
3	Lamp C	Green

请注意，Chair B 在 Color 列中有一个 NULL 值，它表示还没有为这种椅子提供颜色。假设你想创建所有产品的一个列表，如果执行如下的 SELECT 语句：

```
SELECT
Description,
Color
FROM Products
```

它会显示为：

Description	Color
Chair A	Red
Chair B	NULL
Lamp C	Green

但是，对于没有颜色的情况，用户可能更想要看到像“Unknown”这样的内容，而不是 NULL。解决方案如下所示：

```
SELECT
Description,
ISNULL (Color, 'Unknown') AS 'Color'
FROM Products
```

检索到的数据如下所示：

Description	Color
Chair A	Red
Chair B	Unknown
Lamp C	Green

数据库的差异：MySQL 和 Oracle

ISNULL 函数在 MySQL 中名为 IFNULL。在 MySQL 中，和上述语句等价的语句是：

```
SELECT
Description,
IFNULL (Color, 'Unknown') AS 'Color'
FROM Products;
```

ISNULL 函数在 Oracle 中名为 NVL。等价的语句是：

```
SELECT
Description,
NVL (Color, 'Unknown') AS Color
FROM Products;
```

此外，与 Microsoft SQL Server 和 MySQL 不同，当遇到 NULL 值时，Oracle 显示的是破折号而不是单词 NULL。

4.7　小结

本章介绍了很多不同的函数，它们基本上是把一组值转换成另一个值的预定义规则。就像电子表格提供了操作数据的内建函数一样，SQL 也提供了类似的功能。除了介绍基本的字符函数、日期/时间函数、数值函数和转换函数以外，我们还介绍了如何使用其中的两个或多个函数来创建复合函数。

介绍函数的很多资料都是枯燥乏味的，因为真的有如此之多的函数，它们具备各种不同的功能。要讨论每个可用函数的每一处细微差别，几乎是不可能的。

要记住的一点是，当需要使用函数时，我们可以很容易地在数据库的参考指南中查找它们。在线参考材料可以作为搞清楚每个函数如何工作的一种方便的资源。所以，当你需要使用任何特定的函数时，可以直接查看在线资料，以验证函数的语法。

在下一章中，我们将暂时停止讨论 columnlist 的话题，而介绍一些更有趣的内容——如何对数据进行排序。排序可以满足许多用途，并且能够满足用户按照特定顺序方式来查看数据的基本需求。有了排序，我们将开始对所展示的信息进行整体性的考虑，而不是只考虑单个的数据项。

第5章 排序数据

关键字： ORDERBY、ASC、DESC

要完成手边的任务，对数据进行排序的这种能力经常是必不可少的。例如，如果你以随机顺序来显示一个非常大的客户列表，那么就会发现很难找到任何一个特定的客户。然而，如果相同的列表是以字母顺序排列的，那么你就能很快地找到所需的客户。

这种按照字母顺序对数据进行排序的想法可以应用到很多的情况中，即使是严格来说数据不具有字母特性的时候，也可能使用这种排序。例如，你可能想要按照订货日期和时间对一个订单列表进行排序，以便能够快速地找到在特定日期和时间所产生的订单。或者你可能想要按照订货量对一个订单列表进行排序，以便能够从小到大地浏览订单。无论采用哪种排序形式，当向最终用户展示数据时，排序都是一种组织数据的有效方式。

5.1 添加排序

到目前为止，我们还没有按照任何特定顺序返回过数据。当执行 SELECT 时，你不会知道哪一行会先出现。如果在程序中执行查询，没有人能在那里及时地看到数据，所以排序确实无关紧要。但是，如果要立刻把数据显示给用户，那么行的顺序往往就会变得很重要。使用 ORDER BY 子句，可以很容易地为 SELECT 语句添加排序。

带有 ORDER BY 子句的 SELECT 语句的一般格式如下所示：

```
SELECT columnlist
FROM tablelist
ORDER BY columnlist
```

ORDER BY 子句总是在 FROM 子句之后，而 FROM 子句总是在关键字 SELECT 之后。关键字 SELECT 和 ORDER BY 后边的斜体字 columnlist，表示可以列出任意多的列。Columnlist

中的列，可以是单个的列，也可以是较为复杂的表达式。关键字 SELECT 和 ORDER BY 后边指定的列，可能是完全不同的列。斜体的 tablelist 表示可以列出任意多的表，尽管我们还没有介绍列出多个表的语法。

来看一个示例，我们将使用如下的 Customers 表中的数据：

CustomerID	FirstName	LastName
1	William	Smith
2	Janet	Smith
3	Natalie	Lopez
4	Brenda	Harper

5.2 升序排序

如果你想要按照从 A 到 Z 的字母顺序来排序，那么只需要在 SELECT 语句中加上 ORDER BY 子句。例如：

```
SELECT
FirstName,
LastName
FROM Customers
ORDER BY LastName
```

返回的数据如下所示：

FirstName	LastName
Brenda	Harper
Natalie	Lopez
William	Smith
Janet	Smith

由于这里有两个 LastName 是 Smith，他们的 FirstName 分别是 William 和 Janet，那么就没有办法预测哪一个会排在前面。这是因为我们只按照 LastName 排序，而又有多个行的 LastName 是相同的。

同理，如果执行如下的 SELECT 语句：

```
SELECT
```

```
FirstName,
LastName
FROM Customers
ORDER BY FirstName
```

那么，检索到的数据如下所示：

FirstName	LastName
Brenda	Harper
Janet	Smith
Natalie	Lopez
William	Smith

这次的顺序完全不同，因为我们是按照 FirstName 进行排序的。

SQL 提供了一个叫做 ASC 的特殊关键字，它表示升序（Ascending）。这个关键字是可选的，而且大多数时候是没有必要的，因为所有排序默认都是升序排序。如下所示的 SELECT 语句，使用了 ASC 关键字，返回与之前相同的数据：

```
SELECT
FirstName,
LastName
FROM Customers
ORDER BY FirstName ASC
```

关键字 ASC 用来强调以升序进行排序这个事实，它与降序排序相反。

5.3　降序排序

关键字 DESC 的排序顺序和 ASC 相反。这样的排序不是升序，而是降序（Descending）。

例如：

```
SELECT
FirstName,
LastName
FROM Customers
ORDER BY FirstName DESC
```

检索到的数据如下所示：

FirstName	LastName
William	Smith
Natalie	Lopez
Janet	Smith
Brenda	Harper

现在所有的 FirstName 就是按从 Z 到 A 的字母顺序排序的。

5.4 根据多列来排序

我们现在再来看看如何处理两个 Smith 的问题。如果你想要用 LastName 排序，但是又有两个人有相同的 LastName，那么就需要添加 FirstName 作为第 2 个排序列，如下所示：

```
SELECT
FirstName,
LastName
FROM Customers
ORDER BY LastName, FirstName
```

这会返回：

FirstName	LastName
Brenda	Harper
Natalie	Lopez
Janet	Smith
William	Smith

既然指定了第 2 个排序列，现在你可以确定 Janet Smith 会出现在 William Smith 之前了。请注意，ORDER BY 子句需要把 LastName 放在 FirstName 之前。列的顺序很重要，你想要的是先按照 LastName 排序，然后按照 FirstName 排序，所以需要把 LastName 列放在前边。

5.5 根据计算字段来排序

我们使用第 3 章所介绍的计算字段和别名，来说明更多的可能性。SELECT 语句如下所示：

```
SELECT
LastName + ', ' + FirstName AS 'Name'
```

```
FROM Customers
ORDER BY Name
```

返回的数据如下所示:

Name
Harper, Brenda
Lopez, Natalie
Smith, Janet
Smith, William

正如你所看到的，我们能够在 ORDER BY 子句中引用列的别名（Name）。这也说明了别名是如此有用的另一个原因。另外，还要注意计算字段自身的设计。我们在 LastName 列和 FirstName 列之间插入一个逗号和一个空格以分隔它们，并以一种常用的格式来显示它们，这种格式也可以很方便地进行排序。这种用逗号分隔 LastName 和 FirstName 的显示姓名的方法，是一种很方便的技巧，要牢记在心，用户经常希望看到按照这种方式组织的名称。

但是，如果你想要把计算字段直接放到 ORDER BY 子句中，而且不将其用作列的别名，那该怎么办呢？与上述示例类似，也可以直接在 ORDER BY 子句中指定计算字段，如下所示：

```
SELECT
FirstName,
LastName
FROM Customers
ORDER BY LastName + FirstName
```

结果如下所示：

FirstName	LastName
Brenda	Harper
Natalie	Lopez
Janet	Smith
William	Smith

数据的排序与前边示例相同。唯一的区别就是，我们在 ORDER BY 子句中指定的是一个计算字段，而没有使用列的别名。

5.6 排序序列的更多内容

在前面的示例中，所有数据都是字符类型，都是由字母 A 到 Z 组成，没有数字或特殊字符。另外，也没有考虑大写字母和小写字母的区分。在升序排序中，“dog”既可能出现在“DOG”的前边，也可能出现在“DOG”的后边。

每个数据库都允许用户指定或定制排序规则（Collation）设置，该设置可以管理数据排序的细节。在各个数据库中，这种设置都会有所不同，但是一般都会遵循 3 个原则。

首先，当数据以升序排序时，带有 NULL 值的数据会最先出现。前面介绍过，NULL 值就是没有任何数据。在 NULL 值后边是数字，然后是字符，数字排在字符之前。当以降序排序时，字符会最先出现，然后是数字，然后是 NULL 值。

其次，对于字符数据，通常大写和小写没有区别。一般会把 e 和 E 看作是相同的。

数据库的差异：Oracle

与 Microsoft SQL Server 和 MySQL 不同，在 Oracle 中，如果以升序排序的话，NULL 值会出现在列表的最后。

在 Oracle 中，可以为 ORDER BY 子句添加一个特殊的关键字 NULLS FIRST，它强制 NULL 值在升序排序中最先出现。该语句的一般格式如下：

```
SELECT columnlist
FROM tablelist
ORDER BY columnlist NULLS FIRST
```

如果关键字 NULLS FIRST 出现在降序排序中，就像通常的情况一样，NULL 值会出现在最后。

此外，与 Microsoft SQL Server 和 MySQL 不同，Oracle 在排序列表中会区分字母的大小写。在 Oracle 中，升序排序时，大写字母总是出现在小写字母之前。例如，在 Oracle 中，单词“DOG”会出现在单词“dog”之前。然而，在 Microsoft SQL Server 和 MySQL 中，Dog 和 dog 会相同对待。

最后，对于字符数据来讲，单个字符组成的值从左到右来计算。如果我们正在讨论字母，那么 AB 将排在 AC 的前边。我们来看一个示例，数据取自下表：

TableID	CharacterData	NumericData
1	23	23
2	5	5
3	Dog	NULL
4	NULL	–6

在这个表中，把 CharacterData 列定义为字符列，假设是 VARCHAR 型（一个可变长度的数据类型）。类似的，把 NumericData 列定义为数值列，假设是 INT 型（一个整数类型）。没有数据的值显示为 NULL。

当针对该表执行如下的 SELECT 语句时：

```
SELECT NumericData
FROM tablename
ORDER BY NumericData
```

结果为：

NumericData
NULL
–6
5
23

请注意，NULL 排在最前面，然后按照数字顺序排列。如果我们想要把 NULL 值假设成默认值 0，那么我们就可以使用第 4 章所介绍的 ISNULL 函数，来执行这条 SELECT 语句：

```
SELECT
ISNULL (NumericData, 0)
FROM tablename
ORDER BY ISNULL (NumericData, 0)
```

结果如下：

NumericData
–6
0
5
23

ISNULL 函数把 NULL 值转换成零，这就会导致以不同的顺序来排序。

把 NULL 值显示为 NULL 还是零，这取决于你正在使用的具体的应用程序。基本上，如

果用户考虑把 NULL 值的含义当成零，那么 NULL 值就应该显示为 0。然而，如果用户把 NULL 值当成没有数据，那么显示单词 NULL 会更准确。

对于相同的表，使用不同的 ORDER BY 子句，会得到什么样的结果？如果我们执行如下的 SELECT 语句：

```
SELECT
CharacterData
FROM tablename
ORDER BY CharacterData
```

它会显示为：

CharacterData
NULL
23
5
Dog

和预期一样，NULL 最先出现，然后是数字，然后是按照字母顺序排列的字符。请注意，23 排在了 5 前边。这是因为，23 和 5 是按照字符来计算的，而不是按照数字计算的。因为字符数据是从左到右进行计算，而 2 要比 5 小，所以要先显示 23。

5.7 小结

在本章中，我们介绍了按照特定的顺序对数据进行排序的基本可能性。我们介绍了如何对多个列进行排序。我们还介绍了用计算字段进行排序。最后，我们介绍了一些特殊的排序，特别是数据中有 NULL 值以及字符列中有数字的情况。

在本章的开头，我们介绍了排序的一些常规用法。其中最重要的就是，能够简单地把数据按照容易理解的顺序排列，这样用户就能快速找到他们想要查找的信息。人们通常喜欢按照顺序来查看数据，排序可以实现这个目标。另一个有意思的排序用法会在第 7 章中介绍。在第 7 章中，我们会介绍关键字 TOP 以及使用该关键字排序的另一种方法。这种技术通常称为 Top N 排序，例如，它能够让你显示某个时间段内的前 5 笔订单的客户。

在下一章中，我们将结束关于 columnlists 的分析。我们将使用 CASE 语句和基于列的逻辑，在 columnlist 表达式中引入一些真正的逻辑。

第6章 基于列的逻辑

关键字： CASE、WHEN、THEN、ELSE 和 END

本章的重点是介绍 CASE 表达式。从本章的标题可以看出，CASE 表达式是基于列的逻辑的一种形式。这个术语表明，这些表达式是针对列而不是针对行来应用逻辑的。有时候也把 CASE 表达式称作有条件的逻辑（conditional logic）。基本上，当 CASE 表达式应用于特定的列或数据元素时，它允许我们根据逻辑条件来改变想要展现给用户的输出。

作为一名初级 SQL 程序员，你应该知道 CASE 表达式是相当高级的主题。不使用 CASE 表达式，我们仍然可以获取数据，并且还能编写一些有用的查询。但是，这个主题的知识，却能够让你与众不同。实际上，当你通读全书后，该主题可能是你想要再次回顾的主题之一，你会想起，使用该技术可以完成一些有趣的事情。

6.1 IF-THEN-ELSE 逻辑

我们接下来看一些实实在在的逻辑。到目前为止，我们已经介绍了如何从一张表中选择列、应用一些计算和函数以及添加排序。但是，我们还没有做任何与逻辑相关的事。

SQL 中的 CASE 表达式，让我们能够在 SELECT 语句中，针对单个的表达式应用传统的 IF-THEN-ELSE 类型的逻辑。IF-THEN-ELSE 是面向过程编程语言所使用的一种通用的逻辑结构。一般情况下，这种类型的逻辑如下所示：

```
IF some condition is true
THEN do this
ELSE do that
```

CASE 表达式是能够出现在 SELECT 语句中很多地方的一种结构。在本章中，我们所重点关注的 CASE 表达式，出现在紧跟在关键字 SELECT 之后的 columnlist 中。包含列和 CASE

表达式的 SELECT 语句，大概如下所示：

```
SELECT
column1,
column2,
CaseExpression
FROM table
```

6.2 简单格式

CASE 表达式有两种一般格式，通常称为简单格式和查询格式。简单格式如下所示：

```
SELECT
CASE ColumnOrExpression
WHEN value1 THEN result1
WHEN value2 THEN result2
(repeat WHEN-THEN any number of times)
[ELSE DefaultResult]
END
```

可以看出，CASE 表达式还使用了除 CASE 以外的一些关键字：WHEN、THEN、ELSE 和 END，需要使用这些额外的关键字来完整定义 CASE 表达式的逻辑。关键字 WHEN 和 THEN 定义了要计算的条件。如果 WHEN 后边的值是 true，那么使用 THEN 后边的结果。关键字 WHEN 和 THEN 可以重复任意多次。当有一个 WHEN 时，必须要有一个对应的 THEN。如果没有 WHEN-THEN 条件是 true，那么用关键字 ELSE 来定义一个默认值以供使用。中括号表示，关键字 ELSE 不是必需的。但是，在每个 CASE 表达式中包含关键字 ELSE 以明确声明一个默认值，这通常是一个好主意。

我们来看一个具体的示例，它用到了如下的 Products 表：

ProductID	CategoryCode	ProductDescription
1	F	Apple
2	F	Orange
3	S	Mustard
4	V	Carrot

使用这个表中的数据并带有 CASE 表达式的一条 SELECT 语句看上去如下所示：

```
SELECT
CASE CategoryCode
WHEN 'F' THEN 'Fruit'
```

```
WHEN 'V' THEN 'Vegetable'
ELSE 'Other'
END AS 'Category',
ProductDescription AS 'Description'
FROM Products
```

产生的结果如下：

Category	Description
Fruit	Apple
Fruit	Orange
Other	Mustard
Vegetable	Carrot

我们逐行来看一下前面这条 SELECT 语句。第 1 行包含了关键字 SELECT；第 2 行用到了关键字 CASE，告诉我们要分析 CategoryCode 列；第 3 行，引入了第一个 WHEN-THEN 条件，它表示如果 CategoryCode 列等于 F，那么值要显示为“Fruit”。下一行表示如果 CategoryCode 列等于 V，那么值要显示为“Vegetable”。如果 CategoryCode 既不是 F 也不是 V，ELSE 行会提供一个默认值“Other”供使用。END 行结束了 CASE 语句，并且还包含了一个关键字 AS，为这个 CASE 表达式提供了一个列的别名。

下一行的 ProductDescription 是另一个列，和 CASE 表达式没有关系。

CASE 表达式对于把不好理解的值转换成有意义的描述是很有用的。在这个示例中，Products 表中的 CategoryCode 列只包含了一个字符编码，用来表示产品的类型。它用 F 表示 Fruit，V 表示 Vegetables，S 表示 Spices 等。CASE 子句允许我们指定这种转换。

6.3　查询格式

查询格式的 CASE 表达式，其一般格式如下所示：

```
CASE
WHEN condition1 THEN result1
WHEN condition2 THEN result2
repeat WHEN-THEN any number of times)
[ELSE DefaultResult]
END
```

使用这种格式，前面的 SELECT 语句的等价形式是：

```
SELECT
CASE
WHEN CategoryCode 52 'F' THEN 'Fruit'
WHEN CategoryCode 52 'V' THEN 'Vegetable'
ELSE 'Other'
END AS 'Category',
ProductDescription AS 'Description'
FROM Products
```

检索到的数据和第一种格式是相同的，注意细微的差别。在简单格式中，待计算的列名放在了关键字 CASE 之后，WHEN 后边的表达式是一个简单的直接量。在查询格式中，待计算的列名不是放在关键字 CASE 后边。相反，这种格式允许在关键字 WHEN 后面放置较为复杂的条件表达式。

在上述示例中，可以选用任意一个 CASE 子句的格式，并且都会得到相同的结果。我们再来看另外一个示例，该示例只有用查询格式才能获得想要的结果。

接下来的示例将使用如下的数据：

ProductID	Fruit	Vegetable	Spice	ProductDescription
1	X			Apple
2	X			Orange
3			X	Mustard
4		X		Carrot

在这种情况下，数据库包含了多个列，用来表示产品是水果、蔬菜还是香料。为这些数据创建相同输出的 CASE 表达式如下：

```
SELECT
CASE
WHEN Fruit 52 'X' THEN 'Fruit'
WHEN Vegetable 52 'X' THEN 'Vegetable'
ELSE 'Other'
END AS 'Category',
ProductDescription AS 'Description'
FROM Products
```

同样，结果如下所示：

Category	Description
Fruit	Apple
Fruit	Orange
Other	Mustard
Vegetable	Carrot

既然数据现在用 3 个分开的列来表示产品是水果、蔬菜还是香料，那么我们需要使用查询格式的 CASE 子句，以实现所需要的逻辑。而简单格式只是对单个的列进行分析。

因为 IF-THEN-ELSE 逻辑的固有复杂性，CASE 表达式是本书中最富有挑战的主题之一。在本章中，我们关注的重点是在 SELECT columnlist 中使用 CASE 表达式。但是，CASE 表达式也可以用于其他的 SQL 子句中，诸如 ORDER BY 子句，以及其他尚未讨论的子句中，如 WHERE 和 HAVING 子句。

我们只给出使用 CASE 表达式的一个额外示例。尽管还没有介绍 WHERE 子句，但是假设我们知道一些关于它的内容，正如我们会在第 7 章中所介绍的，WHERE 子句允许我们针对要展现给用户的行应用查询条件。一个典型的表达式如下所示：

```
WHERE ProductDescription = 'White Glove'
```

这是一个非常具体的指令，我们只想看到产品是白色手套的那些行。CASE 表达式的价值在于，它允许我们对查找的值（可能是基于其他列的值）应用条件逻辑。例如，你可能有另外一个名为 ProductType 的列，它给出了关于产品的更多信息。使用 CASE 表达式，如果 ProductType 等于 X，那么可以选择的产品是白色手套；如果 ProductType 等于 Y，那么可以选择的产品就是袜子。其实，你可以在 WHERE 子句中用 CASE 表达式代替‘White Glove’值，从而描述一些更为复杂的逻辑。

6.4　小结

使用 CASE 表达式，可以为 SELECT columnlist 中的列或表达式提供逻辑计算。这种表达式有两种基本格式：简单格式和查询格式。典型的用法是，为那些值不太好理解的数据项提供转换。最后，尽管本章的标题是“基于列的逻辑”，但是 CASE 表达式也可以用于 SELECT columnlist 以外的其他地方，可以把它们用于任何你想要为特定的列或数据元素指定带条件的逻辑的地方。

在下一章中，我们将不再介绍对列值应用逻辑，而是介绍如何对选定的整个行应用逻辑。毫无疑问，这是个需要耐心等待的话题。在 SELECT 语句中指定查询条件的能力，对于大多数常规查询来说，都是至关重要的。在现实世界中，很少会执行没有查询条件的一条 SELECT 语句。下一章所要介绍的主题，使得我们能够实现这一目标。

第7章
基于行的逻辑

关键字：WHERE 和 TOP/LIMIT/ROWNUM

我们终于要介绍如何对表应用查询条件了。到目前为止，我们的 SELECT 语句总是返回表中的每一行数据。在现实世界中，很少会出现这样的情形。通常，我们只是想要获取满足特定条件的数据。本章的主题将会解决这个问题。

如果你正在查询客户，那么通常只会看到所有客户的一个子集。如果你正在检索客户的订单，那么可能只想看到符合特定条件的订单。如果你正在查看产品，那么可能只想查看特定类型的产品。很少有人想去直接查看每一项内容。你或者其他人感兴趣的数据，通常都会导向一个小的数据子集，以便能够分析或者查看某个特定的方面。

7.1 应用查询条件

SQL 中的查询条件是从 WHERE 子句开始的，关键字 WHERE 完成了选择行的一个子集的任务。关键字 WHERE 所使用的逻辑，就是建立在第 6 章所介绍的基于列的逻辑之上的。不同之处在于，CASE 表达式只允许对一个特定的列应用逻辑，而现在则可以对表中所有的行应用逻辑。

如下是 SELECT 语句的一般格式，包括 WHERE 子句和之前介绍过的其他子句：

```
SELECT columnlist
FROM tablelist
WHERE condition
ORDER BY columnlist
```

可以看到，WHERE 子句必须总是在 FROM 和 ORDER BY 子句之间。实际上，任何的子句都必须按照这个顺序来使用。

来看一个示例，数据取自如下的 Orders 表：

OrderID	FirstName	LastName	QuantityPurchased	PricePerItem
1	William	Smith	4	2.50
2	Natalie	Lopez	10	1.25
3	Brenda	Harper	5	4.00

我们从带有简单的 WHERE 子句的一条语句开始：

```
SELECT
FirstName,
LastName,
QuantityPurchased
FROM Orders
WHERE LastName = 'Harper'
```

其输出如下：

FirstName	LastName	QuantityPurchased
Brenda	Harper	5

由于 WHERE 子句规定只查询 LastName 等于‘Harper’的行，那么表中的 3 行记录中只有一行返回。

请注意，因为 LastName 是一个文本列，所以需要的 LastName 列中的值要用引号括起来。对于数字字段，就不需要引号了。例如，下面的 SELECT 语句同样有效，并且会返回相同的数据：

```
SELECT
FirstName,
LastName,
QuantityPurchased
FROM Orders
WHERE QuantityPurchased = 5
```

7.2 WHERE 子句操作符

在前边语句中，WHERE 子句中的等号（=）用做操作符，表示对相等性的判断。上面显示的一般格式表明了条件要放在关键字 WHERE 的后边，这个条件由一个操作符及其两边的两个表达式组成。

下面是基本操作符的一个列表，它们都可以用于 WHERE 子句中：

WHERE Operator	Meaning
=	equals
<>	does not equal
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to

更高级的操作符会在下一章中介绍。

等于(=)和不等于(<>)的意思应该是很明显的。下面是一个带“大于”操作符的 WHERE 子句的示例，同样也使用 Orders 表中的数据：

```
SELECT
FirstName,
LastName,
QuantityPurchased
FROM Orders
WHERE QuantityPurchased > 6
```

结果如下：

FirstName	LastName	QuantityPurchased
Natalie	Lopez	10

在这个示例中，只有一行记录满足 QuantityPurchased 列的值大于 6 的条件。尽管“大于”操作符不常用，但是还是有可能对文本列使用它的。示例如下：

```
SELECT
FirstName,
LastName
FROM Orders
WHERE LastName > 'K'
```

结果如下：

FirstName	LastName
William	Smith
Natalie	Lopez

由于判断条件是 LastName 中的文本要大于 K，所以只返回了 Smith 和 Lopez，而没有返回 Harper。当用于文本字段时，大于操作符和小于操作符表示按照值的字母顺序来选择。在这个示例中，返回 Smith 和 Lopez，因为 S 和 L 在字母表中排在 K 的后边。

最后，需要注意的是，在查询格式的 CASE 表达式中，所有这些操作符也可以和 WHEN 关键字一起使用。例如，一个有效的 CASE 表达式可能是这样的：

```
CASE
WHEN column1 > value1 THEN result1
END
```

7.3 限制行

如果你想要查询表中行的一个小的子集，但是并不关心返回哪些行，那该怎么办？假设有一个带有 50000 行数据的表，而你只想知道其中几行的数据看上去是什么样子。用 WHERE 子句来完成这个任务就不是很合理，因为你并不关心返回哪些行。

解决办法是使用一个特殊的关键字来指定你想要限制有多少行返回，这又是数据库之间语法各不相同的另一种情况。在 Microsoft SQL Server 中，完成这个任务的关键字是 TOP。

一般格式如下：

```
SELECT
TOP number
columnlist
FROM table
```

数据库的差异：MySQL 和 Oracle

MySQL 使用关键字 LIMIT 而不是 TOP。一般格式如下：

```
SELECT
columnlist
FROM table
LIMIT number
```

Oracle 使用关键字 ROWNUM 而不是 TOP。关键字 ROWNUM 需要在 WHERE 子句中指定，如下所示：

```
SELECT
columnlist
FROM table
WHERE ROWNUM <= number
```

在本章剩余部分中，你会看到语句都使用 Microsoft 的关键字 TOP。如果你正在使用 MySQL 或者 Oracle，那么可以直接使用关键字 LIMIT 或 ROWNUM 做等价的替换。

假设你想看到一个表中前 10 行的数据。完成这个任务的 SELECT 语句如下所示：

```
SELECT
TOP 10 *
FROM table
```

这条语句返回了表中前 10 行的所有列。和任何没有 ORDER BY 子句的 SELECT 语句一样，没有办法预测会返回哪 10 行，这取决于数据在表中是如何进行物理存储的。

同理，你可以列出要返回的指定的列：

```
SELECT
TOP 10
column1, column2
FROM table
```

实际上，关键字 TOP 实现了和 WHERE 子句类似的功能，因为它允许你返回指定的表中行的一个小的子集。可是要记住，用关键字 TOP 返回的行，并不是统计意义上的一个真正的随机样例。它们只是根据数据在数据库中的物理存储方式，限定了前几行的数据。

7.4　用 Sort 限制行数

关键字 TOP 的另一个用途是，把它和 ORDER BY 子句结合起来使用，基于特定分类，得到带有最大值的一定数量的行。通常把这种类型的数据查询称为 Top N 查询。下面是一个示例，它使用如下的 Book 表中的数据：

BookID	Title	Author	CurrentMonthSales
1	Pride and Prejudice	Austen	15
2	Animal Farm	Orwell	7
3	Merchant of Venice	Shakespeare	5
4	Romeo and Juliet	Shakespeare	8
5	Oliver Twist	Dickens	3
6	Candide	Voltaire	9
7	The Scarlet Letter	Hawthorne	12
8	Hamlet	Shakespeare	2

假设你想看到本月销量最多的 3 本书。实现该任务的 SELECT 语句如下：

```
SELECT
TOP 3
Title AS 'Book Title',
CurrentMonthSales AS 'Quantity Sold'
FROM Books
ORDER BY CurrentMonthSales DESC
```

输出如下：

Book Title	Quantity Sold
Pride and Prejudice	15
The Scarlet Letter	12
Candide	9

让我们仔细看一下这条语句。第 2 行的 TOP 3 表示只返回 3 行数据。主要问题在于，如何决定要显示哪 3 行数据。答案在 ORDER BY 子句中。如果没有 ORDER BY 子句，那么 SELECT 将会直接返回任意 3 行数据，但那不是你想要的结果。你想要看到销量最多的 3 行数据。为了完成这个任务，你需要让 CurrentMonthSales 列以降序排序。为什么是降序？因为以降序排序时，最大的数值会最先出现。如果以升序排序，你会得到销量最少的书，而不是销量最多的图书。

现在，让我们再增加一些背景条件。 假设你只想看到 Shakespeare 所著的销量最多的书。为了实现这一目标，需要添加一个 WHERE 子句，如下所示：

```
SELECT
TOP 1
Title AS 'Book Title',
CurrentMonthSales AS 'Quantity Sold'
FROM Books
WHERE Author = 'Shakespeare'
ORDER BY CurrentMonthSales DESC
```

返回数据如下所示：

Book Title	Quantity Sold
Romeo and Juliet	8

这条 WHERE 子句增加了只查看 Shakespeare 所著的书这一条件，你也可以把关键字 TOP

改为 TOP 1，这就表示只想看到一行数据。

数据库的差异：Oracle

在 Oracle 中，对行的限制和排序的过程稍有点复杂，因为在 Oracle 的语法中，ROWNUM 在 WHERE 子句中。你需要先对数据排序，然后再应用 ROWNUM 查询条件。一般格式如下：

```
SELECT *
FROM
(SELECT
columnlist
FROM table
ORDER BY columnlist DESC)
WHERE ROWNUM <= number
```

这是子查询的一个较早的示例，我们会在第 14 章详细介绍子查询。简而言之，这条语句包含了两个分开的 SELECT 语句。里边的 SELECT 用圆括号包围起来，根据指定的 columnlist 对所需数据按照降序排序。外边的 SELECT 语句从里边的 SELECT 中检索数据，并使用关键字 ROWNUM 来限制所要显示的行数。

7.5 小结

本章介绍的主题是如何对查询应用查询条件。本章介绍了许多基本操作符，诸如等于和大于。能够指定一些基本的查询条件，这在很大程度上使得 SELECT 语句变得真正有用起来。使用 WHERE 子句，我们现在可以执行一条语句来获取来自 New York 的所有客户。

本章还介绍了限制查询中返回的行数的相关主题。最后，限制行数的功能和 ORDER BY 子句组合起来，可以得到有用的 Top N 类型的数据查询。

在下一章中，我们将引入一批新的关键字，它们可以给 WHERE 子句添加复杂的逻辑，从而增强设置查询条件的能力。没错，你现在确实可以选择 New York 的客户了。但是，如果你想要选择在 New York 或者 California 并且不在 Buffalo 或 Los Angeles 的那些客户，那该怎么办呢？下一章所介绍的关键字，使你能够实现这一目标。

第8章
布尔逻辑

关键字：AND、OR、NOT、BETWEEN、IN、IS 和 NULL

在第 7 章中，我们介绍了查询条件的概念，但是只介绍了它的最简单的形式。我们现在将扩展这个概念，从而增强指定 SELECT 语句返回行的能力。接下来，纯粹的 SQL 逻辑要发挥作用了。在本章中，我们会介绍用于创建复杂逻辑表达式的一些操作符。

具备了这些新的能力之后，如果一个用户告诉你，她想要居住地区的邮政编码为 60601 到 62999 之间的所有女性客户的一个列表，但是该列表不包括年龄在 30 岁以下或者没有 Email 地址的人，那么，你就能够为她提供想要的内容。

8.1 复杂的逻辑条件

第 7 章中介绍的 WHERE 子句只用到了最简单的查询条件。你看到的 WHERE 子句如下所示：

```
WHERE QuantityPurchased = 5
```

在这个 WHERE 子句中所表达的条件非常简单：它返回了 QuantityPurchased 列的值等于 5 的所有行。

在现实世界中，选择数据经常很复杂。因此，我们现在把注意力转到在查询条件中指定一些更复杂的逻辑条件的方法。

这种设计复杂逻辑条件的能力叫做布尔逻辑（Boolean logic）。这个术语来自于数学，指的是构造出能够计算是真还是假的复杂逻辑的能力。在上述示例中，针对表中的每一行，条件 QuantityPurchased = 5 会计算为真或假。显然，你只想看到条件结果为真的那些行。

用于生成复杂的布尔逻辑的关键字有：AND、OR 和 NOT。这 3 个操作符都用来为 WHERE

子句添加额外的功能。AND、OR 和 NOT 操作符同圆括号一起，通过适当的组合，能够指定你可以想象得到的任何逻辑表达式。

8.2 AND 操作符

以下的示例都使用 Orders 表中的数据：

OrderID	CustomerName	State	QuantityPurchased	PricePerItem
1	William Smith	IL	4	2.50
2	Natalie Lopez	CA	10	1.25
3	Brenda Harper	NY	5	4.00

使用 AND 操作符的一个 WHERE 子句的示例如下：

```
SELECT
CustomerName,
QuantityPurchased
FROM Orders
WHERE QuantityPurchased > 3
AND QuantityPurchased < 7
```

AND 子句表示，对于选中的行，所有条件的计算结果都必须为真。

这条 SELECT 语句指定只检索那些 QuantityPurchased 的值大于 3 且小于 7 的行。因此，只有两行数据返回：

CustomerName	QuantityPurchased
William Smith	4
Brenda Harper	5

请注意，没有返回 Natalie Lopez 行。为什么？Natalie 购买的数量为 10，实际上，它满足了第一个条件（QuantityPurchased > 3）。但是，它没有满足第 2 个条件（QuantityPurchased < 7），因此它不为真。当使用 AND 操作符时，所有指定的条件都必须为真。

8.3 OR 操作符

我们现在来看一下 OR 操作符。AND 子句意味着对于所选中的行，所有的条件都必须为

真。OR 子句意味着，如果确定任意的条件为真，那么就选中该行。

下面是一个示例，取自同样的表：

```
SELECT
CustomerName,
QuantityPurchased,
PricePerItem
FROM Orders
WHERE QuantityPurchased > 8
OR PricePerItem > 3
```

这条 SELECT 语句返回的数据如下所示：

CustomerName	QuantityPurchased	PricePerItem
Natalie Lopez	10	1.25
Brenda Harper	5	4.00

为什么显示了 Natalie Lopez 行和 Brenda Harper 行，而没有显示 William Smith 行呢？选中 Natalie Lopez 行是因为它满足第一个条件的要求（QuantityPurchased > 8）。第 2 个条件（PricePerItem > 3）不为真，但没什么关系，因为 OR 条件只需要有一个条件为真即可。

同理，选中 Brenda Harper 行是因为对于该行来说，第 2 个条件（PricePerItem > 3）为真。没有选中 William Smith 行，是因为对它来说，这两个条件都不满足。

8.4　使用圆括号

假设你只对来自 Illinois 或 California 的客户的订单感兴趣。另外，你只想看到购买数量大于 8 的订单。为了满足这些要求，你需要把这些条件都放在如下这条 SELECT 语句中：

```
SELECT
CustomerName,
State,
QuantityPurchased
FROM Orders
WHERE State ='IL'
OR State ='CA'
AND QuantityPurchased > 8
```

当执行这条语句时，预期只会返回一行数据，也就是 Natalie Lopez 行。这是因为记录行中有两个客户（Smith 和 Lopez）在 Illinois 或 California。但是，只有其中一名客户（Lopez）

的购买数量大于 8。

但是，当执行这条语句时，会看到如下结果：

CustomerName	State	QuantityPurchased
William Smith	IL	4
Natalie Lopez	CA	10

出了什么问题？为什么返回了两行而不是一行？答案就在于 SQL 如何解释这条既包含 AND 操作符又包含 OR 操作符的 WHERE 子句。和其他计算机语言一样，SQL 有一个预定义的计算顺序，它指定了对于不同操作符的解释顺序。除非指令另有安排，SQL 总是会先处理 AND 操作符，然后才处理 OR 操作符。所以，上述语句中，先看到 AND 并执行如下的条件：

```
State ='CA'
AND QuantityPurchased > 8
```

满足该条件的是 Natalie Lopez 行。然后计算 OR 操作符，它考虑了 State 等于 IL 的那些行，这就增加了 William Smith 行。因此，这就使得 William Smith 和 Natalie Lopez 这两行都满足了条件。

显然，这是不符合本意的。当一个 WHERE 子句中组合使用 AND 和 OR 操作符时，经常会引发这类问题。解决这种二义性的方法是，使用圆括号来明确地指定想要的计算顺序，圆括号中的任何内容总是要先计算的。

在下面的例子中，为之前的 SELECT 语句添加了圆括号，以修正结果：

```
SELECT
CustomerName,
State,
QuantityPurchased
FROM Orders
WHERE (State = 'IL'
OR State ='CA')
AND QuantityPurchased > 8
```

当执行这条语句时，现在可以看到如下的数据：

CustomerName	State	QuantityPurchased
Natalie Lopez	CA	10

SELECT 语句中的圆括号，强制先计算 OR 表达式（State = ‘IL’ OR State= ‘CA’）。这

会得到预期的结果。

8.5　多组圆括号

假设你想要从 Oreders 表中选择两组不同的行：首先是在 New York 的客户，其次是购买数量在 3 到 10 之间的 Illinois 的客户。如下的 SELECT 语句可以完成这个任务：

```
SELECT
CustomerName,
State,
QuantityPurchased
FROM Orders
WHERE State ='NY'
OR (State ='IL'
AND (QuantityPurchased >= 3
AND QuantityPurchased <= 10))
```

其结果如下：

CustomerName	State	QuantityPurchased
William Smith	IL	4
Brenda Harper	NY	5

请注意，在这条语句中有两组圆括号。圆括号在这里的用法，类似于我们在第 4 章中讲到的复合函数中圆括号的用法。对于函数，如果有多于一组的圆括号，总是先计算最里边的函数。在布尔表达式中，圆括号的作用也是一样的。在这个例子中，最里边的那组圆括号包含：

```
(QuantityPurchased >= 3
AND QuantityPurchased <= 10)
```

在针对每一行执行完计算后，再执行外边的那组圆括号：

```
(State ='IL'
AND (QuantityPurchased >= 3
AND QuantityPurchased <= 10))
```

最后，在 WHERE 子句中添加最后的一行(它没有包含在任何的圆括号中)：

```
WHERE State ='NY'
OR (State ='IL'
AND (QuantityPurchased >= 3
AND QuantityPurchased <= 10))
```

8.6 NOT 操作符

除了 AND 和 OR 操作符以外，我们经常还会使用 NOT 操作符来表示复杂的逻辑条件。NOT 表示对其后边的内容的否定或取反。如下是一个简单的示例：

```
SELECT
CustomerName,
State,
QuantityPurchased
FROM Orders
WHERE NOT State ='NY'
```

其结果如下：

CustomerName	State	QuantityPurchased
William Smith	IL	4
Natalie Lopez	CA	10

这条语句指定了所选取行的 State 不等于 NY。在这个简单的示例中，NOT 操作符不是必需的。上述语句也可以通过如下的等价语句来实现：

```
SELECT
CustomerName,
State,
QuantityPurchased
FROM Orders
WHERE State <> 'NY'
```

在这种情况下，不等于操作符（<>）完成了和 NOT 操作符同样的事情。

如下是更复杂的带 NOT 操作符的一个示例：

```
SELECT
CustomerName,
State,
QuantityPurchased
FROM Orders
WHERE NOT (State ='IL'
OR State = 'NY')
```

其结果如下：

CustomerName	State	QuantityPurchased
Natalie Lopez	CA	10

当在一组圆括号前使用 NOT 操作符时，该操作符否定了圆括号中的一切内容。在这个示例中，我们查找 State 不是 Illinois 或 New York 的所有行。

请注意，这个例子中的 NOT 操作符同样不是必需的。上述查询也可以通过如下等价的语句来实现：

```
SELECT
CustomerName,
State,
QuantityPurchased
FROM Orders
WHERE State <> 'IL'
AND State <> 'NY'
```

你可能需要思考一下，为什么上述的两种语句是等价的。第 1 种语句使用了 NOT 操作符和带 OR 操作符的一个逻辑表达式。第 2 种语句把相同的逻辑转换成带 AND 操作符的表达式。

如下是在复杂语句中使用 NOT 操作符的最后一个示例：

```
SELECT
CustomerName,
State,
QuantityPurchased
FROM Orders
WHERE NOT (State ='IL'
AND QuantityPurchased > 3)
```

其结果如下：

CustomerName	State	QuantityPurchased
Natalie Lopez	CA	10
Brenda Harper	NY	5

同之前一样，也有另外一种不使用 NOT 的方法，可以表示上述语句，如下所示：

```
SELECT
CustomerName,
State,
QuantityPurchased
FROM Orders
```

```
WHERE State <> 'IL'
OR QuantityPurchased <= 3
```

通过这些示例可以看出，在带有诸如等号（=）或小于号（<）这样的算术运算符的复杂表达式中，使用 NOT 操作符在逻辑上可能不是必需的。但是，与把表达式转换成不使用 NOT 的表达式相比，在逻辑表达式前放置 NOT 会更加直观。换句话讲，NOT 操作符提供了一种有用的方法来表达逻辑思维。

8.7 BETWEEN 操作符

我们现在来看看两个特定的表达式，它们能够简化那些通常需要 OR 操作符或 AND 操作符的表达式，这就是 BETWEEN 操作符和 IN 操作符。BETWEEN 操作符使得我们可以把带大于等于（>=）操作符和小于等于（<=）操作符的一个 AND 表达式，简化为只带一个操作符的简单表达式。

下面是一个示例。假设你想选择购买数量在 5 到 20 之间的所有行。可以执行如下的 SELECT 语句：

```
SELECT
CustomerName,
QuantityPurchased
FROM Orders
WHERE QuantityPurchased >= 5
AND QuantityPurchased <= 20
```

或者，可以执行使用 BETWEEN 操作符的等价语句：

```
SELECT
CustomerName,
QuantityPurchased
FROM Orders
WHERE QuantityPurchased BETWEEN 5 AND 20
```

在这两个示例中，SELECT 都会返回如下数据：

CustomerName	QuantityPurchased
Natalie Lopez	10
Brenda Harper	5

BETWEEN 操作符总是需要在两个数字之间相应地放置一个 AND。

注意，BETWEEN 操作符相对比较简单。要注意关键字 BETWEEN 只相当于大于等于(>=)和小于等于（<=）操作符。它不能用来表示只是大于（>）或小于（<）某个数字范围的内容。这个示例中，选中了 Brenda Harper 行，因为他的购买数量等于 5，因此在 5 到 20 之间。

NOT 操作符可以和 BETWEEN 一起使用。如下面这条 SELECT 语句所示：

```
SELECT
CustomerName,
QuantityPurchased
FROM Orders
WHERE QuantityPurchased NOT BETWEEN 5 AND 20
```

检索到如下的数据：

CustomerName	QuantityPurchased
William Smith	4

8.8　IN 操作符

就像 BETWEEN 表示了 AND 操作符的一个特例一样，IN 操作符也可以表示 OR 的一个特例。假设你想看到所有 State 是 Illinois 或 New York 的行，可以执行如下的 SELECT 语句：

```
SELECT
CustomerName,
State
FROM Orders
WHERE State ='IL'
OR State ='NY'
```

或者，可以使用带 IN 操作符的等价语句：

```
SELECT
CustomerName,
State
FROM Orders
WHERE State IN ('IL', 'NY')
```

这两条语句都会检索到如下的数据：

CustomerName	State
William Smith	IL
Brenda Harper	NY

请注意，使用逗号来分隔关键字 IN 后边的圆括号中的所有值。

这个示例中的 IN 操作符的用途可能并不明显，因为这里只列出了两个 State。但是，当你想要列出几十个特定的值时，使用 IN 就会使得情况变得简单了。对于这样的语句，使用 IN 会大大减少所需的录入工作。IN 操作符的另一个方便的用法是，你想要在 SQL 语句中使用 Excel 中的数据。如果你想使用 SQL 语句从电子表格的相邻单元格中获取多个值，那么，Excel 允许复制这些的值（以带有逗号分隔符的形式），结果可以粘贴到 IN 操作符后边的圆括号中。

和 BETWEEN 操作符一样，也可以把 NOT 操作符和 IN 一起使用，如下例所示：

```
SELECT
CustomerName,
State
FROM Orders
WHERE State NOT IN ('IL', 'NY')
```

检索到的数据如下所示：

CustomerName	State
Natalie Lopez	CA

关于 IN 操作符，还有最后一点需要注意。还有一种使用 IN 操作符的方法，它和刚才介绍过的语法大不相同。在 IN 操作符的第 2 种格式中，在一个圆括号中指定一条完整的 SELECT 语句，从而在需要时允许逻辑地创建多个单个的值，这叫做子查询，我们会在第 14 章详细介绍它。

8.9 布尔逻辑和 NULL 值

在本章开头，我们介绍过，SQL 中的布尔逻辑在计算复杂的表达式时，要么是真要么是假，这种说法并不完全正确。当在 WHERE 子句中进行条件判断时，其实有 3 种可能：真、假或未知。未知这种可能性，源于 SQL 数据库的列有时是允许有 NULL 值的。第 1 章曾经介绍过，NULL 值是那些缺少数据的值。

SQL 提供了一个特殊的关键字来判断 WHERE 子句中指定的列是否为 NULL。这个关键字就是 IS NULL。我们来看一个示例，数据取自如下的 Products 表：

ProductID	ProductDescription	Weight
1	Printer A	NULL
2	Printer B	0
3	Monitor C	2
4	Laptop D	4

在这个示例中，假设在为 Products 表添加行的时候，开始并没有为 Weight 列赋任何值。最初该列的值是 NULL，系统用户稍后会为产品的重量赋值。

假设你想要使用如下的 SELECT 语句来查找没有重量值的产品：

```
SELECT
ProductDescription,
Weight
FROM Products
WHERE Weight= 0
```

这将返回如下的数据：

ProductDescription	Weight
Printer B	0

这并不是我们想要的结果，重量等于 0 和重量为 NULL 值并不相同。为了修正它，你需要执行如下的语句：

```
SELECT
ProductDescription,
Weight
FROM Products
WHERE Weight =0
OR Weight IS NULL
```

返回结果如下：

ProductDescription	Weight
Printer A	NULL
Printer B	0

也可以使用 IS NOT NULL 表示对关键字 IS NULL 的取反，它允许你检索指定的列不为空的所有行。

在第 4 章中，我们介绍过的 ISNULL 函数，它可以作为关键字 IS NULL 的一个替代。和上述 SELECT 语句等价并且使用了 ISNULL 函数的语句如下所示：

```
SELECT
ProductDescription,
Weight
FROM Products
WHERE ISNULL(Weight, 0) =0
```

这条 SELECT 语句检索到的是同样的两行数据。ISNULL 函数把所有 Weight 列为 NULL 的值都转换为 0。虽然 WHERE 子句判断值是否为 0，但实际上，它是在判断值是否为 0 或 NULL。

还可以把 ISNULL 函数和关键字 IS NULL 组合到一条 SELECT 语句中使用，例如：

```
SELECT
ProductDescription,
ISNULL(Weight, 0) AS 'Weight'
FROM Products
WHERE Weight =0
OR Weight IS NULL
```

这会得到如下的数据：

ProductDescription	Weight
Printer A	0
Printer B	0

8.10 小结

本章介绍的重要话题是如何创建复杂的查询逻辑表达式。使用到的基本布尔逻辑有 AND、OR 和 NOT。我们还介绍了 BETWEEN 和 IN 操作符，在特定条件下，它们使得 AND 操作符和 OR 操作符的语句更为简洁。圆括号是复杂表达式公式中的另一个必不可少的工具。通过使用圆括号或多组圆括号，我们可以创建几乎各种能够想象到的逻辑条件。最后，我们介绍了如何处理选取的数据是 NULL 值的情况。

在下一章中，我们将把兴趣转移到为指定的查询条件提供一些替代的方法。我们首先介绍模式匹配的话题。这将允许你对单词或短语做部分匹配，并且做类似这样的事情：查找出包含单词“white”的所有产品。该章的后半部分将探讨按照单词或短语的读音进行匹配的可能性。例如，允许你找出名字听起来像 Haley 的所有顾客，即便其名字的实际拼写是 Hailey。

第9章 模糊匹配

关键字：LIKE、SOUNDEX 和 DIFFERENCE

我们现在来看两种非精确定义的数据检索的情况。第一种情况，我们看一下根据单词或短语的模糊匹配来数据检索的情况。例如，你可能想要查找名字中包含单词“bank”的客户。

第二种情况，我们将扩展模糊匹配的思路，以根据单词或短语的读音来进行匹配。例如，你可能想要查找名字读音类似“Smith”的客户，尽管它可能不是这么拼写的。

9.1 模式匹配

我们先来看一个短语中的不精确匹配，通常我们称之为模式匹配。在 SQL 中，WHERE 子句使用 LIKE 操作符来查找针对列值的某个部分的匹配。LIKE 操作符需要使用特殊的通配符，来准确地指定匹配如何工作。让我们从一个示例开始，该示例采用如下的 Movies 表：

MovieID	MovieTitle
1	Love Actually
2	His Girl Friday
3	Love and Death
4	Sweet and Lowdown
5	Everyone Says I Love You
6	Down with Love
7	101 Dalmatians

使用 LIKE 操作符的 SELECT 语句的第一个示例如下所示：

```
SELECT
MovieTitle AS 'Movie'
FROM Movies
```

```
WHERE MovieTitle LIKE '%LOVE%'
```

在本示例中，百分号（%）用做通配符，%通配符表示任意的字符，它也可以表示 0 个字符的一个列表。把%放在 LOVE 前边，表示将接受 LOVE 前边有任意多个字符的短语。类似地，把%放在 LOVE 后边，表示将接受 LOVE 后边带有任意多个字符的短语。

数据库的差异：Oracle

与 Microsoft SQL Server 和 MySQL 不同，当判断与直接量的匹配时，Oracle 会区分大小写。在 Oracle 中，LOVE 和 love 是不相同的。在 Oracle 中，等价的语句是：

```
SELECT
MovieTitle AS Movie
FROM Movies
WHERE MovieTitle LIKE '%Love%';
```

在 Oracle 中，一个更好的解决方案是使用 UPPER 函数把数据转换成大写形式，如下所示：

```
SELECT
MovieTitle AS Movie
FROM Movies
WHERE UPPER (MovieTitle) LIKE '%LOVE%';
```

换句话讲，我们要查找电影名称中包含 LOVE 短语的所有影片。如下是上述 SELECT 语句所返回的数据：

Movie
Love Actually
Love and Death
Everyone Says I Love You
Down with Love

请注意，LOVE 可能作为第一个单词、最后一个单词或中间的单词出现在电影名称中。

现在，我们只查找名称以 LOVE 开头的电影。如果执行如下的 SELECT 语句：

```
SELECT
MovieTitle AS 'Movie'
FROM Movies
WHERE MovieTitle LIKE 'LOVE%'
```

将只会检索到如下数据：

Movie
Love Actually
Love and Death

既然我们现在在短语 LOVE 之后指定了%通配符，那么将只会得到以 LOVE 开头的电影。

类似的，如果执行了如下语句：

```
SELECT
MovieTitle AS 'Movie'
FROM Movies
WHERE MovieTitle LIKE '%LOVE'
```

只会得到如下数据：

Movie
Down with Love

这是因为现在我们指定了短语必须要以 LOVE 结尾。

如果只想观看其名称中包含 LOVE 的影片，而又不想看 LOVE 在电影名称的开头和结尾的那些影片，该怎么办呢？

解决方法如下：

```
SELECT
MovieTitle AS 'Movie'
FROM Movies
WHERE MovieTitle LIKE '% LOVE %'
```

请注意，在短语 LOVE 及其左右两边的%通配符之间插入了一个空格。这就确保了在 LOVE 两边都至少有一个空格。这条语句返回的数据如下所示：

Movie
Everyone Says I Love You

9.2　通配符

%符号是最常见的和 LIKE 操作符一起使用的通配符。然而，还有一些其他可用的通配

符。包括下划线（_）、方括号括起来的 characterlist，以及用方括号括起来的脱字符号（^）加上 characterlist。下表列出了这些通配符和它们的含义：

Wildcard	Meaning
%	any characters (can be zero characters)
_	exactly one character (can be any character)
[*characterlist*]	exactly one character in the character list
[^*characterlist*]	exactly one character not in the character set

在本章剩余的部分，我们将使用如下的 Actors 表来说明语句：

ActorID	FirstName	LastName
1	Cary	Grant
2	Mary	Steenburgen
3	Jon	Voight
4	Dustin	Hoffman
5	John	Wayne
6	Gary	Cooper
7	Julie	Andrews

使用下划线（_）通配符的示例如下：

```
SELECT
FirstName,
LastName
FROM Actors
WHERE FirstName LIKE '_ARY'
```

这条 SELECT 语句的输出如下：

FirstName	LastName
Cary	Grant
Mary	Steenburgen
Gary	Cooper

这条语句检索到了 3 位演员，因为他们的 first name 都是一个字符加上 ARY。

同理，如果执行如下语句：

```
SELECT
FirstName,
LastName
FROM Actors
WHERE FirstName LIKE 'J_N'
```

它会产生如下结果：

FirstName	LastName
Jon	Voight

没有选中演员 John Wayne，因为 John 不符合 J_N 模式。下划线只能表示一个字符。

我们最后要介绍的通配符是[characterlist]和[^characterlist]，它们使你能在一个位置指定多个通配符值。

数据库的差异：MySQL 和 Oracle

在 MySQL 和 Oracle 中，没有[characterlist]通配符和[^ characterlist]通配符。

下面的示例说明了如何使用[characterlist]通配符：

```
SELECT
FirstName,
LastName
FROM Actors
WHERE FirstName LIKE '[CM]ARY'
```

这条语句检索出 FirstName 以 C 或 M 开头并以 ARY 结尾的所有行。其结果如下：

FirstName	LastName
Cary	Grant
Mary	Steenburgen

下面的示例说明了如何使用[^characterlist]通配符：

```
SELECT
FirstName,
LastName
FROM Actors
WHERE FirstName LIKE '[^CG]ARY'
```

这条 SELECT 语句检索出 FirstName 不以 C 或 G 开头，并且以 ARY 结尾的所有行。其结果如下：

FirstName	LastName
Mary	Steenburgen

最后要注意的是，NOT 操作符可以和 LIKE 组合起来使用，如下例所示：

```
SELECT
FirstName,
LastName
FROM Actors
WHERE FirstName LIKE '%ARY%'
AND FirstName NOT LIKE '[MG]ARY'
```

其结果如下：

FirstName	LastName
Cary	Grant

9.3 按照读音匹配

让我们从字母和字符的匹配转向读音的匹配。SQL 提供了两个函数，它们提供了一些有趣的方法来比较单词或短语的读音。这两个函数是 SOUNDEX 和 DIFFERENCE。我们先来看看使用 SOUNDEX 函数的一个示例：

```
SELECT
SOUNDEX ('Smith') AS 'Sound of Smith',
SOUNDEX ('Smythe') AS 'Sound of Smythe'
```

其结果如下：

Sound of Smith	Sound of Smythe
S530	S530

SOUNDEX 函数总是返回由 4 个字符组成的一种代码，用来表示短语的读音。第一个字符总是这个短语的第一个字母。在本例中，第一个字符是 S，因为 Smith 和 Smythe 都是以 S

开头的。

另外 3 个字符是根据剩余短语的读音，经过计算后得到的。在内部，这个函数先删除所有元音和字母 Y。所以，该函数从 SMITH 中得到 MITH，然后把 MITH 转换成 MTH。

同理，它从 SMYTHE 中得到 MYTHE，然后把 MYTHE 转换成 MTH。然后，它分配一个数字表示短语的读音。在这个示例中，这个数字是 530。

因为对于 Smith 和 Smythe，SOUNDEX 返回的值都是 S530，所以可以得出的结论是它们的发音非常相似。

Microsoft SQL Server 提供了另一个名为 DIFFERENCE 的函数，它可以和 SOUNDEX 函数一起使用。

数据库的差异：MySQL 和 Oracle

在 MySQL 和 Oracle 中，没有 DIFFERENCE 函数。

如下示例使用了同样的单词：

```
SELECT
DIFFERENCE ('Smith', 'Smythe') AS 'The Difference'
```

其结果如下：

The Difference
4

DIFFERENCE 函数总是需要两个参数。在内部，这个函数先检索每个参数的 SOUNDEX 值，然后比较这两个值。上述示例中，如果它返回的值是 4，那么意味着 SOUNDEX 值中的 4 个字符全部相等，返回值为 0 表示字符中没有相等的值。因此，DIFFERENCE 的值为 4 表示是最大可能的匹配，DIFFERENCE 的值为 0 表示是最小可能的匹配。

如下示例展示了如何用 DIFFERENCE 函数来检索与指定短语发音类似的值。我们将从 Actors 表中查找 First Name 发音类似于 John 的行。这条 SELECT 语句如下所示：

```
SELECT
FirstName,
LastName
FROM Actors
WHERE DIFFERENCE (FirstName, 'John') =4
```

其结果如下所示：

FirstName	LastName
Jon	Voight
John	Wayne

DIFFERENCE 函数推断出，John 与 Jon 这两个名字和指定值 John 的 DIFFERENCE 值都是 4。

如果你想精确地分析为什么会选中这两行，可以修改 SELECT 语句，以显示表中所有行的 SOUNDEX 和 DIFFERENCE 值：

```
SELECT
FirstName,
LastName,
DIFFERENCE (FirstName, 'John') AS 'Difference Value',
SOUNDEX (FirstName) AS 'Soundex Value'
FROM Actors
```

返回结果如下所示：

FirstName	LastName	Difference Value	Soundex Value
Cary	Grant	2	C600
Mary	Steenburgen	2	M600
Jon	Voight	4	J500
Dustin	Hoffman	1	D235
John	Wayne	4	J500
Gary	Cooper	2	G600
Julie	Andrews	3	J400

请注意，Jon Voight 和 John Wayne 的 SOUNDEX 值都是 J500，他们的 First Name 的 DIFFERENCE 值都是 4，这就解释了为什么会选中他们。还要注意，Julie Andrews 的 DIFFERENCE 值是 3。如果你指定一条 WHERE 子句的 DIFFERENCE 值等于 3 或 4，那么也会选中 Julie Andrews。

9.4 小结

本章介绍了按照模式或者读音对短语进行匹配。按照模式匹配是 SQL 中一个重要并且广

泛使用的功能。当在一个查询框中输入一个单词，试图去检索所有包含该单词的条目时，就要使用模式匹配。我们很少按照读音进行匹配，尽管存在这种技术，但是把单词转换成读音，这本身有难度。英语或任何其他重要的语言，包含了很多读音方面的怪异之处和例外情况，这使得很难进行可靠的读音匹配。

在下一章中，我们的注意力将转向把分散数据汇集成组，以及使用各种统计方法对那些组中的数据进行汇总。在第 4 章中，我们介绍过标量函数。下一章将介绍另一种类型的函数，我们称之为聚合函数（Aggregate functions）。这种聚合函数允许我们用许多有用的方法来汇总数据。例如，我们将能够查看任意一组订单，确定订单的数量，所有订单的总价以及订单尺寸的平均值。通过这些技术，我们能够超越具体数据的报告，开始发布汇总信息，以便真正地为用户提升价值。

第10章 汇总数据

关键字： DISTINCT、SUM、AVG、MIN、MAX、COUNT、GROUP BY 和 HAVING

到目前为止，我们所展示的数据基本上都是已经存在于数据库中的数据。的确，我们已经可以使用一些函数来移动数据，并且创建了一些其他的计算，但是我们检索到的行与底层数据库中的行是对应的。现在，我们来看看汇总数据的各种方法。

按照计算机的术语，通常把这种类型的工作称作聚合（aggregation），它表示“合并到组中”。聚合和汇总数据的能力是从单纯地显示数据到接近真实信息的关键性跨越。当用户看到一个报表中汇总的数据时，这其中有一些神奇之处。我们已经能够从数据库的庞大数据中挖掘出一些真实含义，以便用更加清晰的画面来展现数据所表示的含义，用户会更加理解和欣赏。

10.1 消除重复

汇总数据的最基本方法是消除重复，尽管这并没有提供一个真正的聚合。SQL 有一个名为 DISTINCT 的关键字，它提供了一种简单的方法来删除输出中重复的行。

如下是关键字 DISTINCET 的一个示例，我们使用如下的 SongTitles 表中的数据：

SongID	Artist	Album	Title
1	The Beatles	Abbey Road	Come Together
2	The Beatles	Abbey Road	Sun King
3	The Beatles	Revolver	Yellow Submarine
4	The Rolling Stones	Let It Bleed	Monkey Man
5	The Rolling Stones	Flowers	Ruby Tuesday
6	Paul McCartney	Ram	Smile Away

假设你想要查看表中的 Artist 的一个列表，通过如下语句来完成这个任务：

```
SELECT
DISTINCT
Artist
FROM SongTitles
ORDER BY Artist
```

其结果如下：

Artist
Paul McCartney
The Beatles
The Rolling Stones

关键字 DISTINCT 总是紧随在关键字 SELECT 之后。DISTINCT 表示只会返回后面的 columnlist 的列值唯一的那些行。在这个示例中，只有 3 个唯一的 artist，所以只返回了 3 行。

如果你想看看 artist 和 albums 的唯一组合，执行如下的语句：

```
SELECT
DISTINCT
Artist,
Album
FROM SongTitles
ORDER BY Artist, Album
```

其结果如下：

Artist	Album
Paul McCartney	Ram
The Beatles	Abbey Road
The Beatles	Revolver
The Rolling Stones	Flowers
The Rolling Stones	Let It Bleed

请注意，尽管表中该专辑有两首歌曲，Abbey Road 只列出了一次。这是因为关键字 DISTINCT 导致列出的列中只能显示唯一的值。

10.2 聚合函数

我们在第 4 章中介绍的函数都是标量函数（scalar function），这些函数只能对单个的数字

或值进行计算。相反，聚合函数意味着可以用于分组数据。最常用的聚合函数有 COUNT、SUM、AVG、MIN 和 MAX。它们提供了对分组数据进行计数、加和、取平均值、取最小值和最大值等方法。

所有聚合函数的示例，都采用如下两个和学生相关的表中的数据：Fees 和 Grades。Fees 表包含数据如下：

FeeID	Student	FeeType	Fee
1	George	Gym	30
2	George	Lunch	10
3	George	Trip	8
4	Janet	Gym	30
5	Alan	Lunch	10

Grades 表如下所示：

GradeID	Student	GradeType	Grade
1	Susan	Quiz	92
2	Susan	Quiz	95
3	Susan	Homework	84
4	Kathy	Quiz	62
5	Kathy	Quiz	81
6	Kathy	Homework	NULL
7	Alec	Quiz	58
8	Alec	Quiz	74
9	Alec	Homework	88

先从 SUM 函数开始，假设你想要查看所有学生健身费用的合计，如下语句能够完成这个任务：

```
SELECT
SUM (Fee) AS 'Total Gym Fees'
FROM Fees
WHERE FeeType ='Gym'
```

结果数据如下：

Total Gym Fees
60

可以看到，SUM 函数将满足 WHERE 子句中查询条件的行的 Fee 列的全部值加和。由于

columnlist 中唯一的表达式是一个聚合函数，所以该查询语句只返回了一行数据，给出了汇总值。

数据库的差异：MySQL

第 4 章提到过，MySQL 有时候要求在函数名称和左括号之间不能有空格。这同样适用于大多数聚合函数。例如，在 MySQL 中，上述语句会写成如下所示：

```
SELECT
SUM(Fee) AS 'Total Gym Fees'
FROM Fees
WHERE FeeType = 'Gym'
```

函数 AVG、MIN 和 MAX 非常相似，如下是 AVG 函数的一个示例。在这个示例中，我们要查找 Grades 表中所有‘Quiz’的平均成绩：

```
SELECT
AVG (Grade) AS 'Average Quiz Score'
FROM Grades
WHERE GradeType ='Quiz'
```

其结果如下：

Average Quiz Score
77

在单个的 SELECT 语句中，可以使用多个聚合函数。在一条 SELECT 语句中使用 AVG、MIN 和 MAX 函数，如下所示：

```
SELECT
AVG (Grade) AS 'Average Quiz Score',
MIN (Grade) AS 'Minimum Quiz Score',
MAX (Grade) AS 'Maximum Quiz Score'
FROM Grades
WHERE GradeType='Quiz'
```

其结果如下：

Average Quiz Score	Minimum Quiz Score	Maximum Quiz Score
77	58	95

我们看到分别计算的数字，输出显示了 Grades 表中所有‘Quiz’的平均成绩、最高分和

最低分。

10.3　COUNT 函数

COUNT 函数稍有点复杂，可以用 3 种不同的方式来使用它。

首先，COUNT 函数可以用来返回所有选中行的数目，而不管任何特定列的值。例如，如下语句返回了 GradeType 为‘HomeWork’的所有行的数目：

```
SELECT
COUNT (*) AS 'Count of Homework Rows'
FROM Grades
WHERE GradeType ='Homework'
```

其结果如下：

Count of Homework Rows
3

圆括号中的星号表示“所有列”。SQL 检索了那些选中行的所有列，然后它返回了行的数目。

COUNT 函数的第 2 种格式指定了一个具体的列而不是星号。如下例所示：

```
SELECT
COUNT (Grade) AS 'Count of Homework Scores'
FROM Grades
WHERE GradeType = 'Homework'
```

其结果如下：

Count of Homework Scores
2

注意前面两条 SELECT 语句的细微差别。在第一条语句中，我们只计算 GradeType 列等于“Homework”的行数，结果是 3 行。在第二条语句中，我们计算 GradeType 列等于“Homework”并且 Grade 列存在值的行数。在这个示例中，满足 GradeType 列等于“Homework”条件的有 3 行，但其中有 1 行的 Grade 列是 NULL 值，所以没有把它计算在内。记住，NULL 表示数

据不存在。

COUNT 的第 3 种格式允许我们除了使用列名还使用关键字 DISTINCT。如下所示：

```
SELECT
COUNT (DISTINCT FeeType) AS 'Number of Fee Types'
FROM Fees
```

这条语句计算了 FeeType 列中的唯一值的行数。

其结果如下：

Number of Fee Types
3

这表示在 FeeType 列中找到了 3 个不同的值。

10.4 分组数据

前面的聚合函数的示例非常有趣，但是应用价值受到限制。当我们介绍完分组数据的概念之后，聚合函数的真正强大之处将展现出来。

我们使用关键字 GROUP BY 把 SELECT 语句返回的数据分成任意数目的组。例如，当查看前面的 Grades 表时，你可能对根据 GradeType 来分析测验分数感兴趣。换句话讲，你想把数据分成两个组：Quiz 组和 Homework 组。我们可以用 GradeType 列的值来决定每一行应该属于哪一组。

一旦把数据分到组中，就可以使用聚合函数对可以计算和比较的每一组进行汇总统计了。

我们通过如下的示例来介绍关键字 GROUP BY：

```
SELECT
GradeType AS 'Grade Type',
AVG (Grade) AS 'Average Grade'
FROM Grades
GROUP BY GradeType
ORDER BY GradeType
```

其结果如下：

Grade Type	Average Grade
Homework	86
Quiz	77

在这个示例中，关键字 GROUP BY 指定了根据 GradeType 列的值来创建组。SELECT 语句的 columnlist 中的两列分别是 GradeType 列和使用了 AVG 函数的一个计算字段。GradeType 列包含在了 columnlist 中，因为在创建一个组时，把分组所参照的列包含到 columnlist 中，通常是一个好办法。计算字段“Average Grade”根据每组中所有行的值来进行聚合。

注意“Homework”的平均成绩是 86 分。虽然 Homework 类型的某行其 Grade 列为 NULL 值，SQL 也足够聪明，在计算平均值时会忽略带有 NULL 值的那一行。如果你想要把 NULL 值当作 0，那么可以用 ISNULL 函数把 NULL 转换成 0，如下所示：

```
AVG (ISNULL(Grade, 0)) AS 'Average Grade'
```

需要注意的一点是，当使用关键字 GROUP BY 时，columnlist 中的所有列要么是 GROUP BY 子句中的列，要么是在聚合函数中使用的列。

例如，如下 SELECT 语句将会出错：

```
SELECT
GradeType AS 'Grade Type',
AVG(Grade) AS 'Average Grade',
Student AS 'Student'
FROM Grades
GROUP BY GradeType
ORDER BY GradeType
```

这条语句的问题是：Student 列既不在 GROUP BY 子句中，也不在任何聚合函数中。因为所有内容都在组中出现，所以 SQL 不知道该如何处理 Student 列。

数据库的差异：MySQL

与 Microsoft SQL Server 和 Oracle 不同，在 MySQL 中，上述语句不会报错，但是会产生错误的结果。

10.5　多列和排序

组的概念可以扩展，从而根据多个列来分组。我们再来看看最后的那条 SELECT 语句，

并且把 Student 列添加到 GROUP BY 子句中和 columnlist 中。它现在看上去如下所示：

```
SELECT
GradeType AS 'Grade Type',
Student AS 'Student',
AVG (Grade) AS 'Average Grade'
FROM Grades
GROUP BY GradeType, Student
ORDER BY GradeType, Student
```

最终数据如下：

Grade Type	Student	Average Grade
Homework	Alec	88
Homework	Kathy	NULL
Homework	Susan	84
Quiz	Alec	66
Quiz	Kathy	71.5
Quiz	Susan	93.5

你现在不仅看到 GradeType，还可以看到 Student。Average Grade 是根据每组数据计算的结果。注意 Kathy 的 Homework 行显示了一个 NULL 值，因为她只有一个 Homework 行，那行的 grade 是一个 NULL 值。

GROUP BY 子句中列的顺序没有意义，如果 GROUP BY 子句如下所示，结果也是相同的：

```
GROUP BY Student, GradeType
```

但是，和通常情况一样，ORDER BY 子句中列的顺序是有意义的。如果修改 ORDER BY 子句中列的顺序，如下所示：

```
ORDER BY Student, GradeType
```

那么，结果如下：

Grade Type	Student	Average
Homework	Alec	88
Quiz	Alec	66
Homework	Kathy	NULL
Quiz	Kathy	71.5
Homework	Susan	84
Quiz	Susan	93.5

这看上去仍然有点奇怪，因为乍一看很难知道数据是先按照 Student 排序，然后再按照 Grade Type 排序。作为一般规则，如果按照排序的列的顺序来列出这些列，往往是有帮助的。

一条更易于理解的 SELECT 语句如下所示：

```
SELECT
Student AS 'Student',
GradeType AS 'Grade Type',
AVG (Grade) AS 'Average Grade'
FROM Grades
GROUP BY GradeType, Student
ORDER BY Student, GradeType
```

数据现在如下所示：

Student	Grade Type	Average Grade
Alec	Homework	88
Alec	Quiz	66
Kathy	Homework	NULL
Kathy	Quiz	71.5
Susan	Homework	84
Susan	Quiz	93.5

这样就更容易理解了，因为列的顺序和排序的顺序是对应的。

由于 GROUP BY 子句和 ORDER BY 子句之间的不同，有时候会造成混淆。要记住 GROUP BY 只是创建了组，我们仍然需要使用 ORDER BY 来按照正确的顺序显示数据。

10.6　基于聚合查询条件

我们还有另一个话题需要讨论，这就是汇总数据。一旦创建了组，查询条件就会变得更加复杂。当针对带 GROUP BY 的一条 SELECT 语句应用任何查询条件时，人们必须要问查询条件是应用于单独的行还是整个组。

实际上，WHERE 子句是对单独的行执行查询条件。SQL 提供了一个名为 HAVING 的关键字，它允许对组级别使用查询条件。

再来看 Grades 表，假设你只想查看‘Quiz’成绩在 70 分及以上的 Grades。你想查看的 Grade 是单独的 Grade，所以可以按照常规方式使用 WHERE 子句。SELECT 语句如下所示：

```
SELECT
Student AS 'Student',
GradeType AS 'Grade Type',
Grade AS 'Grade'
FROM Grades
WHERE GradeType ='Quiz'
AND Grade >= 70
ORDER BY Student, Grade
```

最终的数据如下所示：

Student	GradeType	Grade
Alec	Quiz	74
Kathy	Quiz	81
Susan	Quiz	92
Susan	Quiz	95

请注意，‘Quiz’成绩小于 70 分的记录不会显示。例如，你可以看到 Alec 的‘Quiz’成绩是 74 分的记录，而看不到他的成绩是 58 分的记录。

但是，如果你想要只显示‘Quiz’的平均成绩在 70 分及以上的学生的数据，该怎么办呢？你想要选择一个平均值，而不是单独的行，这就需要用到关键字 HAVING。你需要先按照 Student 分组，然后把查询条件应用到基于全组的一个聚合统计上。如下语句可以完成这个任务：

```
SELECT
Student AS 'Student',
AVG (Grade) AS 'Average Quiz Grade'
FROM Grades
WHERE GradeType ='Quiz'
GROUP BY Student
HAVING AVG (Grade) >= 70
ORDER BY Student
```

其输出如下：

Student	Average Quiz Grade
Kathy	71.5
Susan	93.5

这条 SELECT 语句既有 WHERE 子句也有 HAVING 子句。WHERE 确保你只选择了 GradeType 是“Quiz”的行。HAVING 保证你只选择平均成绩至少是 70 分的学生。

如果你想要在结果中添加 GradeType 的值，该怎么办呢？如果你试图在 SELECT columnlist 中添加一个 GradeType，这条语句将出错。这是因为所有列都必须要么出现在 GROUP BY 中，要么包含在一个聚合函数中。如果你想要显示这个 GradeType 列，就必须要把它添加到 GROUP BY 子句中，如下所示：

```
SELECT
Student AS 'Student',
GradeType AS 'Grade Type',
AVG (Grade) AS 'Average Grade'
FROM Grades
WHERE GradeType = 'Quiz'
GROUP BY Student, GradeType
HAVING AVG (Grade) >= 70
ORDER BY Student
```

最终数据如下：

Student	Grade Type	Average Grade
Kathy	Quiz	71.5
Susan	Quiz	93.5

现在，把 HAVING 子句添加到这条混合语句中，让我们重新梳理一下 SELECT 语句的一般格式，如下所示：

```
SELECT columnlist
FROM tablelist
WHERE condition
GROUP BY columnlist
HAVING condition
ORDER BY columnlist
```

需要强调的一点是，当在 SELECT 语句中使用上面的任意一个关键字时，都需要按照这里给定的顺序输入。例如，关键字 HAVING 总是要放在 GROUP BY 之后，但是要放在 ORDER BY 之前。

10.7　小结

在本章中，我们介绍了几种聚合形式，从其中最简单的形式——剔除重复记录开始。然后，我们介绍了多种聚合函数，这是与第 4 章介绍的标量函数不同类型的函数。当聚合函数

和关键字 GROUP BY 组合使用时，聚合函数的强大才真正体现出来，它允许针对划分成组的数据进行真正的聚合。最后，我们介绍了关键字 HAVING，它允许我们对聚合函数中的值应用组级别的查询条件。

在下一章中，我们将开始介绍 SQL 中一个关键的主题，也就是访问多个表中数据的功能。到目前为止，所有的 SELECT 查询都是针对单个的表进行的。在现实世界中，这是不切实际的事情。关系数据库的真正价值在于操作多个有相关数据的表的能力人们很少需要单个表中的数据。

在第 11 章和第 12 章中，我们将直接讨论访问多个表中数据的主题。第 11 章介绍内连接（Inner Join），第 12 章介绍外连接（Outer Join）。随后，第 13 章到第 15 章将介绍同一主题的各种不同情况。在学习完接下来的 5 章后，我们将掌握从多个表中获取数据的基本技能。

第 11 章
用内连接来组合表

关键字： INNER JOIN 和 ON

在第 1 章中，我们曾经讨论过关系数据库和之前数据库相比的最大优势。关系数据库最重要的成就是，能够把数据组织到任意多个相互关联的表中，但同时这些又是彼此独立的。与较早期的数据库不同，关系数据库中的表之间的关系并不是通过一系列的指针来显式地定义的。相反，关系是通过表中共有的列来推断的。有时候，这些关系通过主键和外键来体现，但是主键和外键并不总是必需的。

关系数据库的最大优点在于，人们可以分析业务实体然后进行适当的数据库设计，这样就可以具有最大的灵活性。

我们来看一个常见的示例。大部分企业都有“客户”这样一个为人所熟知的业务实体。同样，数据库通常包含一个 Customers 表，该表定义了每个客户。这样的一个表通常包含一个主键来唯一地标识每个客户，以及任意多个带有描述客户的属性的列。常见的属性可能包含电话号码、地址、城市和州等等。

主要的思路就是把关于客户的所有信息都保存到一个单独的表中，并且只保存在这个表中，这就简化了修改数据的工作。当一个客户更改了他的电话号码时，只需要修改一个表即可。但是，这种安排的不利之处在于，任何时候有人需要关于一个客户的任何信息，这个人都需要访问 Customers 表来检索该信息。

这就为我们引入了连接（join）的概念。假设某人要分析购买过的产品，除了产品的信息，往往也需要提供购买了每件产品的客户信息。例如，分析人员可能需要获取客户的邮政编码来做地理分析，邮政编码只保存在 Customers 表中，产品信息保存在 Products 表中。为了从 Customers 和 Products 表中获取信息，必须采用能够正确匹配信息的方法把这两张表连接到一起。

实际上，关系数据库就是以任何想要的方式把表连接到一起，从而来实现关系的。

11.1 连接两个表

为了介绍连接过程，我们再次回顾在第 3 章中见过的 Orders 表：

OrderID	FirstName	LastName	QuantityPurchased	PricePerItem
1	William	Smith	4	2.50
2	Natalie	Lopez	10	1.25
3	Brenda	Harper	5	4.00

在前边的章节中，这个表的用法有些误导性。实际上，一个称职的数据库设计者不会创建这样的一个表。问题就在于它包含了关于两个独立的实体信息：customers 和 orders。在现实世界中，这些信息至少要拆分到两个独立的表中，其中 Customers 表可能如下所示：

CustomerID	FirstName	LastName
1	William	Smith
2	Natalie	Lopez
3	Brenda	Harper
4	Adam	Petrie

Orders 表可能如下所示：

OrderID	CustomerID	Quantity	PricePerItem
1	1	4	2.50
2	2	10	1.25
3	2	12	1.50
4	3	5	4.00

在本章中，我们将在示例中使用这两个表，另外还有几处添加。Customers 表目前只包含了关于客户的信息。Orders 表目前只有关于购买项的信息。我们为 Orders 表添加了一个 CustomerID 列，用来识别下订单的客户。你可能还记得，在第 1 章中，我们把 CustomerID 列称为外键。我们还为 Orders 表添加了一行，以表示下了多笔订单的一位客户（Natalie Lopez）。此外，我们还为 Customers 表添加了新的一行，表示还没有下订单的一位潜在客户（Adam Petrie）。

当然，这里还少了很多信息。例如，Orders 表通常会包含一些其他的列，诸如用于存储

订单日期的列。而且，Orders 表通常会有一个带有 ProductID 的外键列，以便可以把订单和所销售产品的信息关联起来。实际上，Orders 表本身也可以分成多个表，以便整个订单的相关信息（如订单日期）都可以和每个订购项的相关信息（假设一位客户可以在一笔订单中订购多个项）分别保存。

换句话讲，这仍然不是一个完全真实的示例。但是，我们现在把信息拆分成两个单独的表，我们知道如何创建能够同时从两个表中获取数据的 SELECT 语句。

在弄清楚这条 SELECT 语句之前，我们需要考虑另外一个问题，就是如何可视化地表示两个表以及它们之间存在的隐性关系。之前，对于每个表，我们在首行显示了列名，在后续的行显示了相应的数据。既然多个表要处理，我们将介绍另一种类型的可视化表示。如图 11.1 所示，该图中两张表的首行是表名，后续的各行是列名。这张图是实体关系图(entity-relationship diagram)的一个简化版本。实体（entity）指的是表，关系（relationship）指的是在这些表中的数据元素之间所画的线。

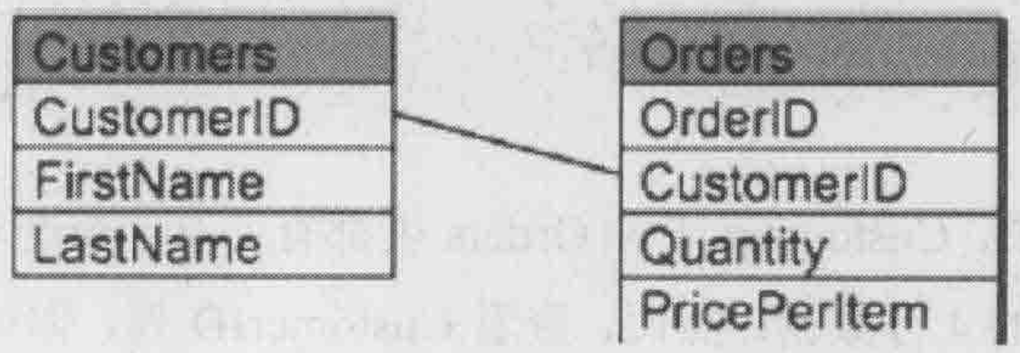


图 11.1　实体-关系图

需要注意的重要一点是，我们从 Customers 表中的 CustomerID 画了一条线到 Orders 表中的 CustomerID。这表示这两个表之间存在关系。两个表共享了 CustomerID 列中存储的值。

11.2　内连接（Inner Join）

现在，我们可以给出一条带内连接的 SELECT 语句：

```
SELECT *
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
```

我们来逐行查看该语句，第 1 行的关键字 SELECT，表示我们想要从两个表中获取所有的列（*）；第 2 行，FROM 子句表示我们想指定的第一个表是 Customers；第 3 行引入了一个

新的关键字 INNER JOIN，该关键字用来指定我们想要连接的另外一个表。在这个示例中，我们想添加的是 Orders 表。

最后，第 4 行引入了关键字 ON，关键字 ON 和 INNER JOIN 组合使用。ON 指定两个表如何准确地连接。在这个示例中，我们把 Customers 表的 CustomerID 列（Customers.CustomerID）和 Orders 表的 CustomerID 列（Orders.CustomerID）连接起来。由于 Customers 表和 Orders 表都有相同名称的 CustomerID 列，我们需要在 ON 子句中指定表名作为 CustomerID 列名的前缀。这个前缀使得我们能够把两个独立表中的这些列区分开。

上述 SELECT 语句产生数据如下：

Customer ID	First Name	Last Name	Order ID	Customer ID	Quantity	PricePerItem
1	William	Smith	1	1	4	2.50
2	Natalie	Lopez	2	2	10	1.25
2	Natalie	Lopez	3	2	12	1.50
3	Brenda	Harper	4	3	5	4.00

我们来分析一下结果，Customers 表和 Orders 表都有 4 行，查看 OrderID 列，你可以看到我们有 Orders 表中全部的 4 行数据。然而，查看 CustomerID 列，你可能会注意到我们只显示了 3 位客户。这是为什么？答案是 CustomerID 为 4 的客户在 Orders 表中并不存在。既然我们通过 CustomerID 字段把两个表连接起来，而且在 Orders 表中没有行能够和 Customers 表中 CustomerID 为 4 的记录匹配，那么结果中就没有 CustomerID 为 4 的行。

这就给我们带来重要的结论：内连接只会返回关联的两个表之间相匹配的数据。在第 12 章中，我们将介绍连接表的另外一种方法，它将允许显示 CustomerID 为 4 的客户信息，即便在 Order 表中没有这位顾客。

如下是第 2 条重要的结论：请注意，在上面的结果中客户 Natalie Lopez 的信息出现了两次，而她在 Customers 表中只有一条记录。那么为什么她的客户数据会显示两次呢？答案就是显示了所有可能的匹配。因为在 Orders 表中有两行 Natalie 的记录，这两行都和 Customers 表中的 Natalie 的行匹配，因此返回了两条关于她的客户信息。

最后，你可能想知道为什么把这种连接称为内连接。实际上，主要有两种不同的连接：内连接和外连接，我们将在第 12 章介绍外连接。

11.3　内连接中表的顺序

内连接返回两个指定的表之间匹配的数据。在上面的 SELECT 语句中，我们在 FROM 子句中指定 Customers 表，在 INNER JOIN 子句中指定 Orders 表。但是，先指定哪张表重要吗？实际上，逆转了表列出的顺序，所得到的结果是相同的。如下两条 SELECT 语句在逻辑上是相同的，并且会返回相同的数据：

```
SELECT *
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID =Orders.CustomerID

SELECT *
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID =Customers.CustomerID
```

唯一的不同是第一条语句将首先显示 Customers 表中的列，其次才会显示 Orders 表中的列。第二条语句将首先显示 Orders 表中的列，其次才会显示 Customers 表中的列。

请记住，SQL 不是过程式语言。SQL 不会指定执行任务的先后次序。SQL 只是指定了需要的逻辑，并且让数据库的内部机制来决定如何执行任务。同样，SQL 也不会决定数据库该如何物理地检索数据，数据库软件会决定获取数据的最优化方式。

11.4　内连接的另一种规范

在上述示例中，我们使用了关键字 INNER JOIN 和 ON 来指定内连接，也可以只使用 FROM 和 WHERE 子句来指定内连接。

我们已经看过连接 Customers 表和 Orders 表的语句：

```
SELECT *
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID =Orders.CustomerID
```

指定相同的内连接，而不使用 INNER JOIN 和 ON 关键字的方式是：

```
SELECT *
```

```
FROM Customers, Orders
WHERE Customers.CustomerID =Orders.CustomerID
```

在这条替代的规范中，没有使用关键字 INNER JOIN 来指定要连接的表，我们只是在 FROM 子句中列出了所有要连接的表。我们使用 WHERE 子句指定表之间的关系，而没有使用 ON 子句来指定表是如何关联的。

即使这种替代格式可以很好地使用，并且会得到相同的结果，但是我们不建议使用这种格式。关键字 INNER JOIN 和 ON 的优点在于它们显式地表示了连接的逻辑，那是它们唯一的用途。尽管在 WHERE 子句中指定关系也是可能的，但是当把 WHERE 子句用作查询条件并且要表示多个表之间的关系时，这条 SQL 语句的含义却不够明显。

11.5 再谈表的别名

我们看看从上面的 SELECT 语句返回的列。既然我们指定了所有的列（*），可以看到两个表中所有的列。我们会看到 CustomerID 列两次，因为这两个表中都存在该列，其实我们不想要重复的数据。下面是 SELECT 的一种替代版本，它现在只指定了你只想看到一列。此外，现在我们显式地指定了表的别名和列的别名。我们通过在关键字 FROM 和 INNER JOIN 的后边插入关键字 AS，从而指定表的别名（C 表示 Customers，O 表示 Orders）。语法如下所示：

```
SELECT
C.CustomerID AS 'Cust ID',
C.FirstName AS 'First Name',
C.LastName AS 'Last Name',
O.OrderID AS 'Order ID',
O.Quantity AS 'Qty',
O.PricePerItem AS 'Price'
FROM Customers AS C
INNER JOIN Orders AS O
ON C.CustomerID =O.CustomerID
```

其结果如下：

Cust ID	First Name	Last Name	Order ID	Qty	Price
1	William	Smith	1	4	2.50
2	Natalie	Lopez	2	10	1.25
2	Natalie	Lopez	3	12	1.50
3	Brenda	Harper	4	5	4.00

请注意，我们使用关键字 AS 来指定列的别名和表的别名。还要注意的是，关键字 AS 完全是可选的。所有的关键字 AS 都可以从 SELECT 语句中删除，这条语句仍然是有效的并且会返回相同的结果。但是，为了清楚起见，我建议还是要使用关键字 AS。

数据库的差异：Oracle

正如第 3 章所述，在 Oracle 中，不使用关键字 AS 来指定表的别名。在 Oracle 中，该语句的语法如下：

```
SELECT
C.CustomerID AS 'Cust ID',
C.FirstName AS 'First Name',
C.LastName AS 'Last Name',
O.OrderID AS 'Order ID',
O.Quantity AS 'Qty',
O.PricePerItem AS 'Price'
FROM Customers C
INNER JOIN Orders O
ON C.CustomerID =O.CustomerID;
```

11.6　小结

在查询中把表连接到一起的能力是 SQL 的基本特性。不使用连接的话，关系数据库的作用就会小很多。本章的重点是介绍内连接的格式，它返回关联的两个表之间匹配的数据。我们还介绍了内连接的另一种替代方法以及表的别名的用途。

在第 12 章中，我们将看到另外一种重要的连接类型，也就是外连接（outer join）。我们介绍过，内连接只允许查看连接的表之间匹配的数据。所以，如果你有客户没下过订单，当在 Customers 表和 Orders 表之间做内连接时，你看不到该客户的任何信息。即使该客户没有订单，外连接也允许我们查看客户的信息。换句话讲，外连接让我们能看到使用内连接所无法看到的数据。

第12章
用外连接来组合表

关键字： LEFT JOIN、RIGHT JOIN 和 FULL JOIN

现在我们要从内连接转向学习外连接。内连接的主要限制是，要显示任何的结果，都要在所有连接的表中有相应的匹配。如果把 Customers 表连接到 Orders 表，如果客户没有订单的话，就不会显示客户的数据。这看上去像是一个微不足道的问题，但是对于不同类型的数据，它往往会变得非常重要。

例如，假设我们有 Orders 表和 Refunds 表。Refunds 表通过 OrderID 和 Orders 表相关联。换句话讲，所有退货都和特定的订单绑定在了一起，没有订单就没有退货。当你想在一条查询中看到订单和退货信息，问题就出现了。如果使用内连接把这两个表连接起来，要是该订单没有退货的话，你将看不到任何订单。而大部分的订单都没有退货。相反，即使 Orders 没有和 Refunds 匹配，外连接也允许你查看 Orders 表，因此这是需要了解和使用的一种基本技术。

12.1 外连接

在第 11 章中，我们看到的所有连接都是内连接。因为内连接是最常用的连接类型，SQL 把内连接指定为默认连接，所以可以只使用关键字 JOIN 来指定一个内连接，而不一定非要用关键字 INNER JOIN。

与内连接不同，外连接有 3 种类型：左连接（LEFT OUTER JOIN）、右连接（RIGHT OUTER JOIN）和全连接（FULL OUTER JOIN）。可以直接表示为：LEFT JOIN、RIGHT JOIN 和 FULL JOIN。在最后一种情况下，关键字 OUTER 不是必需的。总结一下，我们认为有 4 种类型的连接：

- 内连接
- 左连接
- 右连接
- 全连接

这就保持了语法的一致性并且也便于记忆。

我们准备在示例中使用 3 个表来介绍外连接。首先，Customers 表保存了每个客户的信息。其次，Orders 表保存了每笔订单的数据。最后，我们添加了一个 Refunds 表，用来保存客户产生的所有退货信息。

图 12.1 展示了这 3 个表是如何连接的。

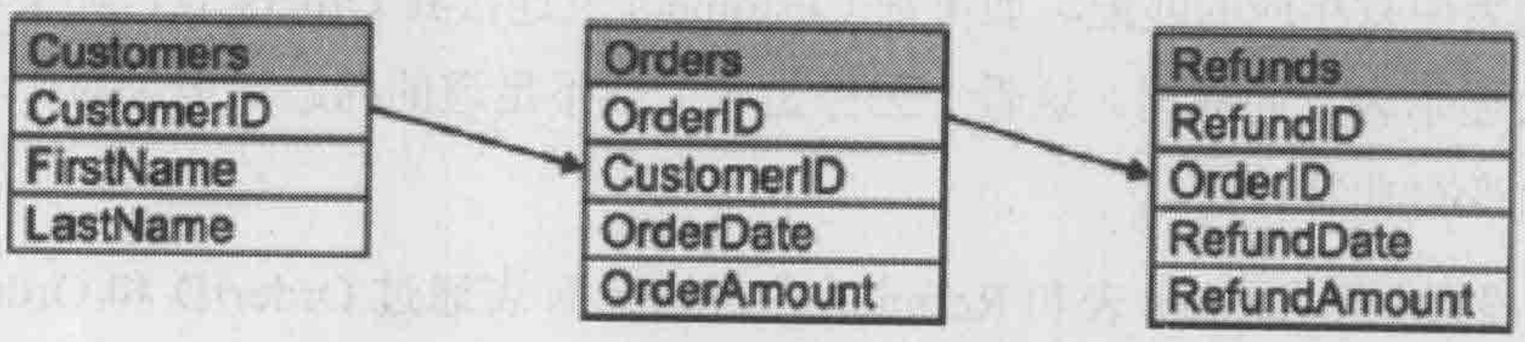


图 12.1　实体-关系图

与在第 11 章中看到的图不同，现在连接表的线是箭头。可以看到，一个箭头从 Customers 表的 CustomerID 字段向下指向 Orders 表的 CustomerID 字段。这个箭头表示，Customers 和 Orders 表之间的连接是单向的，从这个意义上说，并不是所有的客户都有订单。此外，一个客户可能有多笔订单。类似的，Orders 表和 Refunds 表之间所画的箭头表示，并不是所有的订单都有退货，并且一笔订单可能有多次退货。

Customers 表和 Orders 表之间的线在 CustomerID 列上，因为该列是这两个表之间共同的连接。类似的，Orders 表和 Refunds 表之间的线在 OrderID 列上，因为该列是这两个表之间共同的连接。

换句话讲，Orders 表是通过客户和 Customers 表相关联的，一条订单必须存在相对应的一位客户。Refunds 表是通过订单和 Orders 表相关联的，在退货发生前，必须要有一笔订单。请注意，Refunds 表并没有和 Customers 表直接相关联。然而，通过把所有这 3 个表关联起来，我们就可以知道哪位客户退货了。

我们现在查看一下每个表的内容。Customers 表有如下值：

CustomerID	FirstName	LastName
1	William	Smith
2	Natalie	Lopez
3	Brenda	Harper
4	Adam	Petrie

Orders 表有如下数据：

OrderID	CustomerID	OrderDate	OrderAmount
1	1	2009-09-01	10.00
2	2	2009-09-02	12.50
3	2	2009-10-03	18.00
4	3	2009-09-15	20.00

Refunds 表包含数据如下：

RefundID	OrderID	RefundDate	RefundAmount
1	1	2009-09-02	5.00
2	3	2009-10-12	18.00

请注意，4 位客户中有 3 位下了订单。类似的，已下的 4 笔订单中只有 2 笔有退货。

12.2　左连接

现在，我们使用左连接创建一条 SELECT 语句，把所有 3 个表连接到一起的，如下所示：

```
SELECT
Customers.FirstName AS 'First Name',
Customers.LastName AS 'Last Name',
Orders.OrderDate AS 'Order Date',
Orders.OrderAmount AS 'Order Amt',
Refunds.RefundDate AS 'Refund Date',
Refunds.RefundAmount AS 'Refund Amt'
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID =Orders.CustomerID
LEFT JOIN Refunds
ON Orders.OrderID =Refunds.OrderID
```

```
ORDER BY Customers.CustomerID, Orders.OrderID, RefundID
```

所得到的数据如下：

First Name	Last Name	Order Date	Order Amt	Refund Date	Refund Amt
William	Smith	2009-09-01	10.00	2009-09-02	5.00
Natalie	Lopez	2009-09-02	12.50	NULL	NULL
Natalie	Lopez	2009-10-03	18.00	2009-10-12	18.00
Brenda	Harper	2009-09-15	20.00	NULL	NULL
Adam	Petrie	NULL	NULL	NULL	NULL

数据库的差异：Oracle

与 SQL Server 和 MySQL 不同，Oracle 显示日期的典型格式是 DD-MMM-YY。例如，前面表中的 2009-09-02，在 Oracle 中显示为 02-SEP-09。然而，无论使用什么样的数据库，显示日期的具体格式都会有所不同，这取决于如何安装数据库。

在分析上述 SELECT 语句之前，我们看到有两个地方的数据很有趣。首先，Adam Petrie 除了名字以外没有其他的数据。缺少数据的原因是，在 Orders 表中没有和该客户相关的行。外连接的强大之处在于，即使 Adam Petrie 没有订单，你仍然可以看到他的一些数据。如果我们指定的是 INNER JOIN 而不是 LEFT JOIN，你根本就不会看到 Adam 行。

类似的，你可以看到 Natalie Lopez 在 2009-09-02 的订单和 Brenda Harper 的订单没有退货数据，因为 Refunds 表中没有和那些订单相关联的行。如果我们指定的是 INNER JOIN 而不是 LEFT JOIN，那么就看不到那两条订单的行。

我们来看看这条 SELECT 语句本身，指定列的前几行和之前看到的没有什么不同。请注意，我们没有使用表的别名，而是列出了所有列的全名，其中用表名作为前缀。

列出的第 1 个表是 Customers 表。这个表出现在关键字 FROM 的后边。给出的第 2 个表是 Orders 表，它出现在第一个 LEFT JOIN 之后。后续的 ON 子句指定了如何把 Orders 表连接到 Customers 表。给出的第 3 个表是 Refunds 表，它出现在第二个 LEFT JOIN 之后。后边的 ON 子句指定了如何把 Refunds 表连接到 Orders 表。

请注意，对于关键字 LEFT JOIN 来说，表排列的顺序是至关重要的。当指定一个 LEFT JOIN 时，LEFT JOIN 左边的表总是主表（primary table），LEFT JOIN 右边的表是从表（secondary table）。

当连接主表和从表时，我们需要主表的所有行，即使在从表中没有任何行与之匹配。在

第 1 个指定的 LEFT JOIN 中，Customers 表在 LEFT JOIN 左边，Orders 表在 LEFT JOIN 右边，这就表示 Customers 表是主表，Orders 表是从表。换句话讲，我们想看到 Customers 表中所有选中的数据，即使在从表中没有与之相匹配的行。

类似的，在第 2 个 LEFT JOIN 中，Orders 表在左边，Refunds 表在右边。这意味着我们指定 Orders 表为主表，Refunds 是这个连接的从表。我们想要所有的订单，即使在 Refunds 中没有与该订单匹配的行。

最后，我们加入了一条 ORDER BY 子句。请注意，在最初的 columnlist 中，没有选中 ORDER BY 子句中指定的字段。

12.3　判断 NULL 值

在前面SELECT语句中，我们有1位没有订单的客户，以及2笔没有退货的订单。与INNER JOIN 不同，LEFT JOIN 允许这些没有值的行出现。

为了测试你对 LEFT JION 的理解，现在我们要求只列出那些没有退货的订单。解决方案需要添加一条判断空值的 WHERE 子句，如下所示：

```
SELECT
Customers.FirstName AS 'First Name',
Customers.LastName AS 'Last Name',
Orders.OrderDate AS 'Order Date',
Orders.OrderAmount AS 'Order Amt'
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
LEFT JOIN Refunds
ON Orders.OrderID =Refunds.OrderID
WHERE Orders.OrderID IS NOT NULL
AND Refunds.RefundID IS NULL
ORDER BY Customers.CustomerID, Orders.OrderID
```

最终数据如下：

First Name	Last Name	Order Date	Order Amt
Natalie	Lopez	2009-09-02	12.50
Brenda	Harper	2009-09-15	20.00

WHERE 子句先判断 Orders.OrderID 列，以确保该列不为空。这么做就保证了你不会看到没有下订单的客户。WHERE 子句的第 2 部分判断了 Refunds.RefundID 列，以确保该列为空。这就保证了你只会看到没有退货的订单。

12.4 右连接

前面 SELECT 语句使用了关键字 LEFT JOIN。关于右连接的好消息是，它在概念上和左连接相同。左连接和右连接之间唯一的不同就是，在连接中列出两个表的顺序不同。

在左连接中，主表在关键字 LEFT JOIN 的左边列出。而从表可能包含匹配行也可能不包含匹配行，会在关键字 LEFT JOIN 的右边列出。

在右连接中，主表在关键字 RIGHT JOIN 的右边列出，从表在关键字 RIGHT JOIN 的左边列出。这就是两种连接的唯一不同。

前面的 SELECT 语句的 FROM 子句如下所示：

```
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID =Orders.CustomerID
LEFT JOIN Refunds
ON Orders.OrderID =Refunds.OrderID
```

如果我们想要用右连接来重新表述，可以把该语句改写为：

```
FROM Refunds
RIGHT JOIN Orders
ON Orders.OrderID = Refunds.OrderID
RIGHT JOIN Customers
ON Customers.CustomerID = Orders.CustomerID
```

需要注意的是，在 RIGHT JOIN 前后所列出的表的顺序很重要。关键字 ON 后边列出的列的顺序并不重要。

这就意味着，基本上没有必要一定使用关键字 RIGHT JOIN。可以用 RIGHT JOIN 指定的任何内容，也都可以用 LEFT JOIN 指定。我们的建议是使用 LEFT JOIN，因为人的直觉上往往认为先列出的表更为重要。

12.5　外连接中表的顺序

我们之前提到过，在内连接中指定表的顺序并不重要。这和外连接不同，因为在左连接或右连接中，列出表的顺序是至关重要的。然而，当有 3 个表或更多表时，列出表的顺序上就会有一些灵活性。如果需要的话，可以交换左（右）连接的关键字顺序。

我们再看看前面 SELECT 语句最初的 FROM 子句：

```
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
LEFT JOIN Refunds
ON Orders.OrderID =Refunds.OrderID
```

我们已经看到，可以把 Refunds 表列在前边，把 Customers 列在后边，只要把所有内容转换为右连接就可以了，如下所示：

```
FROM Refunds
RIGHT JOIN Orders
ON Orders.OrderID = Refunds.OrderID
RIGHT JOIN Customers
ON Customers.CustomerID = Orders.CustomerID
```

把 Customers 表放在前边，然后是 Refunds 表，然后是 Orders 表，可以这么做吗？是可以的，只要你愿意把左连接和右连接混在一起，并且放在一些圆括号中。和前面的 SELECT 语句等价的语句如下所示：

```
FROM Customers
LEFT JOIN (Refunds
RIGHT JOIN Orders
ON Orders.OrderID =Refunds.OrderID)
ON Customers.CustomerID = Orders.CustomerID
```

最初很简单的语句，现在变得毫无必要的复杂。当设计有多个表的复杂 FROM 子句时，我们建议使用关键字 LEFT JOIN 并且避免使用圆括号。

12.6　全连接

还有一种外连接类型是全连接。我们已经介绍了左连接和右连接，其中一个表是主表，

而另一个表是从表。我们也可以说一个表是必需的，而另一个表是可选的，这表示当匹配两个表时，从表（或可选表）中的行不一定必须存在。

在内连接中，两个表都是主表（或必需的表）。当匹配两个表时，对于选中的行数据，在两个表之间必须有一个匹配。

在全连接中，两个表都是从表（或可选的表）。在这种情况下，如果表 A 和表 B 的行匹配，那么会显示：

（1）表 A 中的所有行，即使它在表 B 中没有匹配的行；

（2）表 B 中的所有行，即使它在表 A 中没有匹配的行。

数据库的差异：MySQL

与 Microsoft SQL Server 和 Oracle 不同，MySQL 不提供全连接。

我们看一个示例，它包含了如下两个表的匹配行。首先，Movies 表如下所示：

MovieID	MovieTitle	Rating
1	Sleepless in Seattle	PG
2	Lost in America	R
3	Bambi	G
4	North by Northwest	Not Rated
5	Forrest Gump	PG-13
6	The Truman Show	PG

其次，Ratings 表如下所示：

RatingID	Rating	RatingDescription
1	G	General Audiences
2	PG	Parental Guidance Suggested
3	PG-13	Parents Strongly Cautioned
4	R	Restricted
5	NC-17	No One 17 and Under Admitted

Movies 表包含一个电影列表，并且包含美国电影协会（MPAA）对每部电影的评级。Ratings

表有一个评级列表以及对它们的说明。假设我们想找到两个表之间的所有匹配，我们准备使用 FULL JOIN 来显示 Movies 表中的所有行以及 Ratings 表中的所有行。全连接会显示所有行，即使没有在其他表中找到一个匹配。SELECT 语句如下所示：

```
SELECT
MovieTitle AS 'Movie',
RatingDescription AS 'Rating Description'
FROM Movies
FULL JOIN Ratings
ON Movies.Rating =Ratings.Rating
ORDER BY RatingDescription, MovieTitle
```

这条语句的结果如下所示：

Movie	Rating Description
North by Northwest	NULL
Bambi	General Audience
NULL	No One 17 and Under Admitted
Sleepless in Seattle	Parental Guidance Suggested
The Truman Show	Parental Guidance Suggested
Forrest Gump	Parents Strongly Cautioned
Lost in America	Restricted

请注意，在数据中有两个空的单元格，这是使用 FULL JOIN 所造成的直接结果。在第 1 种情况中，没有显示 North by Northwest 的评级，因为在 Ratings 表中没有那部电影的匹配行。在第 2 种情况中，“No One 17 and Under Admitted”评级说明没有对应的电影，因为在 Movies 表中没有匹配该评级的行。

在实际工作中，很少会用到全连接，因为表之间这种类型的关系是非常少见的。实际上，全连接显示了两个表之间双向都没有匹配的数据。

12.7　小结

本章把话题扩展到左连接、右连接和全连接。左连接让我们可以把一个主表和一个从表连接在一起。左连接显示了主表的所有行，即使在从表中没有匹配数据。右连接是左连接的直接反转，交换了主表和从表的顺序。最后，全连接使得两个表都是从表，它显示了任何一

个表中所有的行，即使在另一个表中没有匹配。

在第 13 章中，我们准备花点时间讨论两个相关的话题。首先，我们将讨论自连接（Self Join）,它是允许连接一个表及其自身的一种特殊技术。在某种程度上，这会创建表的一个虚拟视图，在某种意义上，我们现在可以从两个不同的角度来查看这个表。在第 13 章中，另一个重要话题是，把自连接的概念扩展为创建多个表的虚拟视图的一种更为通用的方法。

第13章 自连接和视图

关键字：CREATE VIEW、ALTER VIEW 和 DROP VIEW

在第 11 章和第 12 章中，内连接和外连接使用了不同的方法来组合多个表中的数据。现在，我们将介绍使用和定义表的另一种方法。之前，我们总是假设所查找的数据物理地存在于数据库的表中，现在我们介绍两种以虚拟方式查看数据的技术。

首先要介绍的技术是自连接（Self Join），它允许我们两次引用同一个表，该表就像是两个独立的表一样。实质上，自连接创建了表的一个虚拟视图，允许多次使用这个虚拟视图。

其次，我们将介绍数据库的视图，它是使我们能够随意创建新的虚拟表的一种有用的概念。

13.1 自连接

自连接允许我们把一个表和它自身连接起来。自连接最常见的用途是，处理那些本质上是自引用的表。这类表中的一列指向了同一表中的另一列。这种类型关系的一个常见示例是，包含员工信息的表。

在这个示例中，Personnel 表中的每一行都有一个列指向同一表中的另一行，这一列表示该员工的经理是谁。在某种程度上，这和外键的概念很相似。主要的区别在于，外键指向的是其他表的列，而现在的列指向了同一表中的行。

我们看一下 Personnel 表中的数据：

EmployeeID	EmployeeName	ManagerID
1	Susan Ford	NULL
2	Harold Jenkins	1
3	Jacqueline Baker	1
4	Richard Fielding	1
5	Carol Bland	2
6	Janet Midling	2
7	Andrew Brown	3
8	Anne Nichol	4
9	Bradley Cash	4
10	David Sweet	5

ManagerID 列表明该员工要汇报给哪位经理，这列中的 ID 号和 EmployeeID 列的编号相对应。例如，Harold Jenkins 的 ManagerID 是 1。这表示 Harold 的经理是 Susan Ford，因为 Susan 的 EmployeeID 是 1。

类似的，可以看到向 Susan Ford 汇报的有 3 个人：Harold Jenkins、Jacqueline Baker 和 Richard Fielding。请注意，Susan Ford 在 ManagerID 列中没有值。这表示她是公司的负责人，她没有经理。

现在，假设我们想列出所有的员工，并显示每位员工所要汇报的经理姓名。为了完成该任务，我们将创建 Employees 表到自己的一个自连接。自连接中必须要用到表的别名，以便有办法区分表中的每个实例。表的第 1 个实例使用 Employees 作为别名，表的第 2 个实例使用 Managers 作为别名。语句如下所示：

```
SELECT
Employees.EmployeeName AS 'Employee Name',
Managers.EmployeeName AS 'Manager Name'
FROM Personnel AS Employees
INNER JOIN Personnel AS Managers
ON Employees.ManagerID =Managers.EmployeeID
ORDER BY Employees.EmployeeID
```

所得到的数据如下：

Employee Name	Manager Name
Harold Jenkins	Susan Ford
Jacqueline Baker	Susan Ford
Richard Fielding	Susan Ford
Carol Bland	Harold Jenkins
Janet Midling	Harold Jenkins
Andrew Brown	Jacqueline Baker
Anne Nichol	Richard Fielding
Bradley Cash	Richard Fielding
David Sweet	Carol Bland

这条 SELECT 语句最复杂的地方是连接中的 ON 子句。要让自连接正常工作，我们需要使用 ON 在 Personnel 表的 Employees 视图的 ManagerID 列和该表的 Managers 视图的 EmployeeID 列之间建立一种关系。换句话讲，所指定的经理也是一名员工。

请注意，在前面的数据中，没有显示 Susan Ford，因为在这条语句中，我们使用的是内连接。既然 Susan Ford 没有经理，那么 Personnel 表的 Managers 视图就没有相应的匹配。如果我们想要把 Susan 包含进来，只需要把如下一行：

```
INNER JOIN Personnel AS Managers
```

改为：

```
LEFT JOIN Personnel AS Managers
```

检索到的数据如下：

Employee Name	Manager Name
Susan Ford	NULL
Harold Jenkins	Susan Ford
Jacqueline Baker	Susan Ford
Richard Fielding	Susan Ford
Carol Bland	Harold Jenkins
Janet Midling	Harold Jenkins
Andrew Brown	Jacqueline Baker
Anne Nichol	Richard Fielding
Bradley Cash	Richard Fielding
David Sweet	Carol Bland

13.2 创建视图

自连接允许我们为同一个表创建多个视图。我们现在将把这个概念扩展到为任意表或任意表的组合创建新的视图。

视图(View)只是保存在数据库中的 SELECT 语句。一旦保存了，就可以像引用数据库中的任意表一样来引用视图。数据库的表保存了物理的数据，视图不包含数据，但是允许我们像处理真实表的数据一样来处理视图。

为什么需要视图？在本章后边，我们将详细介绍视图的优点，但是简而言之，视图增加了访问数据的灵活性。无论数据库运行了 1 天还是多年，数据都是以非常具体的形式来存储于数据库的表之中。随着时间流逝，访问数据的需求会有所变化，但是很难去重新组织数据库中的数据以满足新的需求。视图的最大优势是，它们允许为数据库中已经存在的数据创建新的虚拟视图。视图使得我们无需物理地重新组织数据，就可以创建等价的新表。同样，视图为我们增加了始终能保持数据库设计不断更新的能力。

视图在数据库中是如何保存的呢？所有关系型数据库都是由许多不同的对象类型组成的，最重要的类型就是表。然而，大部分数据库管理软件都允许用户保存很多其他的对象类型。其中最常用的是视图和存储过程。在数据库中，经常有许多其他的对象类型。例如，Microsoft SQL Server 允许用户创建许多其他的对象类型，诸如函数(function)和触发器(trigger)。

SQL 提供了关键字 CREATE VIEW 来创建新的视图。其语法如下所示：

```
CREATE VIEW ViewName AS
SelectStatement
```

视图创建之后，用 ViewName 表示从 SelectStatement 返回到视图中的数据。

如下是一个示例。在第 12 章中，我们曾看到过这条 SELECT 语句：

```
SELECT
Customers.FirstName AS 'First Name',
Customers.LastName AS 'Last Name',
Orders.OrderDate AS 'Order Date',
Orders.OrderAmount AS 'Order Amt',
Refunds.RefundDate AS 'Refund Date',
Refunds.RefundAmount AS 'Refund Amt'
FROM Customers
```

```
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
LEFT JOIN Refunds
ON Orders.OrderID = Refunds.OrderID
ORDER BY Customers.CustomerID, Orders.OrderID, RefundID
```

这条语句返回了如下的数据：

First Name	Last Name	Order Date	Order Amt	Refund Date	Refund Amt
William	Smith	2009-09-01	10.00	2009-09-02	5.00
Natalie	Lopez	2009-09-02	12.50	NULL	NULL
Natalie	Lopez	2009-10-03	18.00	2009-10-12	18.00
Brenda	Harper	2009-09-15	20.00	NULL	NULL
Adam	Petrie	NULL	NULL	NULL	NULL

如何把这条 SELECT 语句创建为一个视图？我们直接把整个 SELECT 语句放到一条 CREATE VIEW 语句中，如下所示：

```
CREATE VIEW CustomersOrdersRefunds AS
SELECT
Customers.FirstName AS 'First Name',
Customers.LastName AS 'Last Name',
Orders.OrderDate AS 'Order Date',
Orders.OrderAmount AS 'Order Amt',
Refunds.RefundDate AS 'Refund Date',
Refunds.RefundAmount AS 'Refund Amt'
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
LEFT JOIN Refunds
ON Orders.OrderID = Refunds.OrderID
```

在上述 CREATE VIEW 中，只是缺少了最初 SELECT 语句中的 ORDER BY 子句。因为视图没有保存物理的数据，所以在视图中不能包含 ORDER BY 子句。

13.3　引用视图

当我们执行前面的 CREATE VIEW 语句时，它创建了名为 CustomersOrdersRefunds 的一个视图。创建视图不会返回任何数据。它只是定义了可供随后使用的视图。

要使用这个视图来返回与前面一样的数据，我们执行如下的 SELECT 语句：

```
SELECT *
FROM CustomersOrdersRefunds
```

检索结果如下：

First Name	Last Name	Order Date	Order Amt	Refund Date	Refund Amt
William	Smith	2009-09-01	10.00	2009-09-02	5.00
Natalie	Lopez	2009-09-02	12.50	NULL	NULL
Natalie	Lopez	2009-10-03	18.00	2009-10-12	18.00
Brenda	Harper	2009-09-15	20.00	NULL	NULL
Adam	Petrie	NULL	NULL	NULL	NULL

如果只想查看视图中某个特定客户的一些列，该怎么办呢？可以执行如下的一条 SELECT 语句：

```
SELECT
[First Name],
[Last Name],
[Order Date]
FROM CustomersOrdersRefunds
WHERE [Last Name] = 'Lopez'
```

其输出如下：

First Name	Last Name	Order Date
Natalie	Lopez	2009-09-02
Natalie	Lopez	2009-10-03

需要注意的最重要的一点是，当引用这个视图中的列的时候，需要指定列的别名，而该别名是在创建视图时指定的。你不可以再引用最初的列名。例如，视图给 Customers.FirstName 列分配了一个列的别名‘First Name’。在 Microsoft SQL Server 中，这些列的别名因为名字中包含了空格，所以都括在了方括号中。

数据库的差异：MySQL 和 Oracle

在第 2 章中介绍过，MySQL 和 Oracle 使用不同的字符把包含空格的列名括起来。MySQL 使用的是重音符（`），Oracle 使用的是双引号（"）。

13.4　视图的优点

前面的示例描述了使用视图的最重要的优点。一旦创建了视图，就可以像引用表一样来引用视图。即使创建的视图来自于多个关联起来的表，它现在在逻辑上就像是一个表一样。

我们总结一下使用视图的优点：

- **视图可以减少复杂性**。首先，视图可以简化那些特别复杂的 SELECT 语句。例如，如果你有一条关联了 6 个表的 SELECT 语句，在两个表或 3 个表之间创建视图可能会很有用。你可以在 SELECT 语句中引用那些视图，和最初的语句相比，复杂性会大大降低。
- **视图可以增加可复用性**。如果遇到 3 个表总是连接在一起的情况，那么你可以为这 3 个表创建一个视图。然后，当需要这 3 个表中的数据时，可以直接引用预定义的视图，而不用每次都要连接这 3 个表。
- **视图可以正确地格式化数据**。如果在数据库中有 1 列没有正确地格式化，可以使用 CASE 或其他函数把该列正确地格式化为你想要的结果。例如，在数据库中，有一个日期列按照 YYYYMMDD 的格式存储为整数。用户可能更想以日期/时间列来查看这个数据，以便可以像真实的日期一样显示和使用它。为了完成这个任务，可以为这个表创建一个视图，把该列转换成正确的格式。在此之后，对该表的所有引用，都可以引用新的视图而不是该表。
- **视图可以创建计算的列**。假设表中有两个列：Quantity 和 PricePerItem。用户通常关心的是 TotalPrice 的数据，它是把 Quantity 列和 PricePerItem 列相乘而得到的。你可以为最初的表创建带有一个视图，其中带有新的计算的列，该列包含了这个计算结果。然后用户可以引用这个新的视图，并且总是有可用的计算结果。
- **视图可以用来重新命名列的名称**。如果数据库包含了不好理解的列名，可以创建一个带有列的别名的视图，把那些名称转换成更有意义的内容。
- **视图可以创建数据子集**。假设你有一个包含所有客户的表。大部分用户只需要看到去年下过订单的客户。你可以很容易地创建包含这个有用的数据子集的一个视图。
- **视图可以用来加强安全性限制**。可能有这样一种情形，你想要让某些特定用户只能访问给定的表中的某些列。为了完成这个任务，你可以为那些用户创建表的视图。然后，

你可以利用数据库的安全性特点，授予那些用户访问这个新视图的权限，以限制他们访问底层的表。

13.5 修改和删除视图

在创建一个视图之后，可以使用 ALTER VIEW 语句来轻松地修改该视图。语法如下：

```
ALTER VIEW ViewName AS
SelectStatement
```

当修改一个视图时，你需要完整地指定包含在视图中的整个 SELECT 语句。视图中最初的 SELECT 语句，将由你所指定的新的 SELECT 语句替代。

假设你最初用如下语句创建了一个视图：

```
CREATE VIEW CustomersView AS
SELECT
FirstName AS 'First Name',
LastName AS 'Last Name'
FROM Customers
```

现在，如果你想修改这个视图，添加一个新的'Middle Name'列，可以执行如下的语句：

```
ALTER VIEW CustomersView AS
SELECT
FirstName AS 'First Name',
MiddleName AS 'Middle Name',
LastName AS 'Last Name'
FROM Customers
```

同样，创建和修改视图并不会返回任何数据。它只会创建或修改视图的定义。

数据库的差异：Oracle

与 Microsoft SQL Server 和 MySQL 不同，在 Oracle 中，ALTER VIEW 有更多的限制。要在 ORACLE 中完成前面的 ALTER VIEW 所做的事情，需要执行一条 DROP VIEW，然后执行带有新的视图定义的 CREATE VIEW。

用 DROP VIEW 语句来删除之前创建的视图，语法如下所示：

```
DROP VIEW ViewName
```

如果你想删除之前创建的 CustomersView 视图，可以执行如下语句：

```
DROP VIEW CustomersView
```

13.6　小结

自连接和视图是以虚拟方式查看数据的两种不同的方法。自连接允许我们把一个表连接到其自身，视图则更加灵活。

基本上，任何的 SELECT 语句都可以保存成一个视图，然后可以像引用任何常规的表一样来引用视图。和表不同的是，视图并不包含数据。它们只是在已存在的表中定义了一个新的虚拟的数据视图。视图具有多种功能，从减少复杂度到重新格式化数据。创建了视图之后，可以通过 ALTER VIEW 语句和 DELETE VIEW 语句来修改或删除视图。

在第 14 章中，我们将要回到一个话题，它与前边讨论过的如何把表连接起来更直接相关。子查询提供了一种把表关联起来的方法，而无需显式地使用内连接或外连接。由于有各种类型的子查询，并且可以用不同的方法来使用自查询，所以该话题可能是本书中最难但又最具有潜在价值的话题。如何使用子查询以及何时使用子查询，实际上都有很大的灵活性。因此，子查询在查询设计中为你提供了一定创造性。

第14章 子查询

关键字：EXISTS

在第4章中，我们介绍过复合函数,也就是包含了其他函数的函数。同样，SQL查询也能以类似的方法来包含其他查询。包含在其他查询中的查询叫做子查询（Subquery）。

子查询的话题有点复杂，主要是因为有许多种不同的方式可以使用。可以在SELECT语句的许多不同的地方使用子查询，每个子查询都有细微的差别和不同的需求。作为包含在另一个查询中的查询，子查询可以和主查询相互关联并依赖于主查询，也可以完全地独立于主查询。同样，这种区别又导致了不同的使用需求。

不管怎么使用子查询，它们都可以为SQL查询的编写增加许多的灵活性。在许多情况下，子查询提供的功能也可以通过其他方式来完成。在这些实例中，当决定是否使用子查询解决方案时，个人偏好很关键。然而，正如你所看到的，在某些情况下，要完成手边的任务，子查询是必不可少的。

我们从子查询的基本类型的概述开始介绍。

14.1 子查询的类型

子查询不仅可以用于SELECT语句中，而且可以用于INSERT、UPDATE和DELETE语句，这些内容会在第17章中介绍。然而，在本章中，我们将只介绍SELECT语句的子查询。

如下是我们曾看过的一般的SELECT语句：

```
SELECT columnlist
FROM tablelist
WHERE condition
```

```
GROUP BY columnlist
HAVING condition
ORDER BY columnlist
```

可以把子查询插入到 SELECT 语句中的任何子句中。然而，声明和使用子查询的方式略有不同，这取决于子查询是用于 columnlist、tablelist 还是 condition 中。

但是子查询到底是什么？子查询只是插入到 SELECT 语句中的另一条 SELECT 语句。此外，一条 SELECT 语句可以有多个子查询。

可以用 3 种主要的方式来指定子查询，总结如下：

- 当子查询是 tablelist 的一部分时，它指定了一个数据源。
- 当子查询是 condition 的一部分时，它成为查询条件的一部分。
- 当子查询是 columnlist 的一部分时，它创建了一个单个的计算的列。

本章剩余部分将详细介绍这 3 种情形。

14.2 使用子查询作为数据源

当把一个子查询指定为 FROM 子句的一部分时，它立即创建了一个新的数据源。这和创建一个视图并在 SELECT 语句中引用该视图的概念类似。唯一的区别是，视图是永久地保存在数据库中的，而作为数据源的子查询是不保存的，它只是作为 SELECT 语句的一部分临时存在。

我们先来看一个示例，它展示了如何把子查询作为数据源使用。假设我们有一个 Customers 表，如下所示：

CustomerID	CustomerName
1	William Smith
2	Natalie Lopez
3	Brenda Harper
4	Adam Petrie

还有一个 Orders 表，如下所示：

OrderID	CustomerID	OrderAmount	OrderType
1	1	22.25	Cash
2	2	11.75	Credit
3	2	5.00	Credit
4	2	8.00	Cash
5	3	9.33	Credit
6	3	10.11	Credit

我们想要看到客户的列表，以及他们所下的现金订单的总金额。完成这个任务的 SELECT 语句如下所示：

```
SELECT
CustomerName AS 'Customer Name',
ISNULL (CashOrders.SumOfOrders, 0) AS 'Total Cash Orders'
FROM Customers
LEFT JOIN

 (SELECT
CustomerID,
SUM (OrderAmount) as 'SumOfOrders'
FROM Orders
WHERE OrderType = 'Cash'
GROUP BY CustomerID) AS CashOrders

ON Customers.CustomerID = CashOrders.CustomerID
ORDER BY Customers.CustomerID
```

这里插入了两个空行，以便把子查询和语句的其他部分清楚地分开。子查询是这条语句的中间部分。

其结果如下：

Customer Name	Total Cash Orders
William Smith	22.25
Natalie Lopez	8.00
Brenda Harper	0
Adam Petrie	0

Adam Petrie 没有现金订单，因为他没有下任何订单。尽管 Brenda Harper 下了两笔订单，但都是通过信用卡支付的，所以她也显示为没有现金订单。请注意，ISNULL 函数把 Adam 和 Brenda 的 NULL 值转换成了 0。

现在，我们来分析一下子查询是如何工作的。上述语句中的子查询如下所示：

```
SELECT
CustomerID,
SUM (OrderAmount) as 'Total Cash Orders'
FROM Orders
WHERE OrderType = 'Cash'
GROUP BY CustomerID
```

在一般格式中，上面的例子的主 SELECT 语句是：

```
SELECT
CustomerName AS 'Customer Name',
ISNULL (OrderCounts.SumOfOrders, 0) AS 'Total Cash Orders'
FROM
Customers
INNER JOIN
(subquery) AS CashOrders
ON Customers.CustomerID = CashOrders.CustomerID
ORDER BY Customers.CustomerID
```

如果只执行子查询本身，结果如下所示：

CustomerID	SumOfOrders
1	2.25
2	8

我们只看到 CustomerID 为 1 和 2 的数据。子查询中的 WHERE 子句限制我们只能看到现金订单。

在这个示例中，把整个子查询当作一个独立的表或视图来引用。请注意，给子查询赋了一个表别名 CashOrders，允许在主 SELECT 语句中引用子查询中的列。在主 SELECT 语句中，如下的行引用了子查询中的数据：

```
ISNULL (CashOrders.SumOfOrders, 0) AS 'Total Cash Orders'
```

CashOrders.SumOfOrders 是取自于子查询中的一个列。

真的有必要使用子查询来获取需要的数据吗？在这个示例中，答案是肯定的。我们可能试图通过左连接来直接关联 Customers 表和 Orders 表，如下所示：

```
SELECT
CustomerName AS 'Customer Name',
```

```
Sum (OrderAmount) AS 'Total Cash Orders'
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
WHERE OrderType = 'Cash'
GROUP BY Customers.CustomerID, CustomerName
ORDER BY Customers.CustomerID
```

然而，这条语句产生了如下的数据：

Customer Name	Total Cash Orders
William Smith	22.25
Natalie Lopez	8.00

我们看不到任何 Brenda Harper 行或 Adam Petrie 行，因为排除现金订单的 WHERE 子句在主查询中，而不是在子查询。最终，我们不会看到没有下现金订单的客户的任何行。

14.3　在查询条件中使用子查询

在第 8 章中，我们介绍了 IN 操作符的第 1 种格式。我们曾经使用的示例如下：

```
WHERE State IN ('IL', 'NY')
```

在这种格式中，IN 操作符只在括号中列出了多个值。现在我们介绍的是 IN 的第 2 种格式，即把整个 SELECT 语句插入到括号中。例如，可以像下面这样来指定一个州的列表：

```
WHERE State IN
(SELECT
States
FROM StateTable
WHERE Region 'Midwest')
```

和列出单独的州不同，第 2 种格式允许我们通过更复杂的逻辑来创建州的一个列表。

我们通过使用 Customers 表和 Orders 表的一个示例来举例说明。假设我们想获取曾经用现金支付订单的客户的列表。完成这个任务的 SELECT 语句如下所示：

```
SELECT CustomerName AS 'Customer Name'
FROM Customers
WHERE CustomerID IN
(SELECT CustomerID
FROM Orders
WHERE OrderType ='Cash')
```

最终数据如下：

Customer Name
William Smith
Natalie Lopez

列表中没有包含 Brenda Harper，因为她没有用现金下订单。请注意，子查询 SELECT 语句完全放在了关键字 IN 后边的括号中。还需要注意的是，用 CustomerID 列连接了两个查询。尽管我们显示的是 CustomerName，但是我们使用 CustomerID 来定义 Customers 表和 Orders 表之间的关系。

同样，我们可以询问这条子查询是否也可以表示成一条普通的查询，答案是肯定的。如下是返回相同数据的一条等价查询：

```
SELECT CustomerName AS 'Customer Name'
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID =Orders.CustomerID
WHERE OrderType = 'Cash'
GROUP BY Customers.CustomerID, Customers.CustomerName
```

没有使用子查询，我们可以直接连接 Customers 和 Orders 表。然而，现在需要一条 GROUP BY 子句来确保每位客户只返回一行记录。

14.4 关联子查询

到目前为止，我们看到的子查询都是非关联子查询。通常来说，所有的子查询可以分为非关联的（uncorrelated）和关联的（correlated）两种。这两个术语描述了子查询是否和包含它的查询相关联。非关联子查询是不关联的。当一个子查询是非关联的时，这就意味着它完全独立于外围的查询语句。非关联查询作为整个 SELECT 语句的一部分，只会计算和执行一次。

相反，关联子查询是专门和外围的查询语句进行关联的。由于这种显式的关系，关联子查询需要针对返回的每一行逐行计算，并且可能在每次执行子查询的时候得到不同的结果。

用示例来解释是最好的方式。再来看 Customers 表和 Orders 表，假设我们需要创建订单

总金额少于 20 美金的一个客户列表。如下是完成这个任务的语句：

```
SELECT
CustomerName as 'Customer Name'
FROM Customers
WHERE
(SELECT
SUM (OrderAmount)
FROM Orders
WHERE Customers.CustomerID = Orders.CustomerID)
< 20
```

其结果如下：

Customer Name
Brenda Harper

是什么使得这个子查询是关联的而不是非关联的？通过查看这个子查询自身，可以得到答案：

```
SELECT
SUM (OrderAmount)
FROM Orders
WHERE Customers.CustomerID = Orders.CustomerID
```

这个子查询是关联的，因为它无法单独执行。如果单独运行它，这个子查询将报错，因为子查询的上下文中没有 Customers.CustomerID 列。

要理解发生了什么，以一般的方式来查看整个 SELECT 语句会很有用：

```
SELECT
CustomerName as 'Customer Name'
FROM Customers
WHERE
SubqueryResult < 20
```

作为一个关联子查询，子查询需要针对每位客户进行计算。还要注意，只有返回单独的值，这种类型的子查询才会工作。

再问一次，可以把这个子查询转换成一条常规的 SELECT 语句吗？在这种情况中，是可以的。如下是产生相同结果的一条等价的语句：

```
SELECT
CustomerName as 'Customer Name'
```

```
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID =Orders.CustomerID
GROUP BY Customers.CustomerID, CustomerName
HAVING SUM (OrderAmount) < 20
```

请注意，不使用子查询，等价语句现在需要 GROUP BY 子句和 HAVING 子句。GROUP BY 子句创建了客户的组，HAVING 子句限制了每组订单金额必须要小于 20 美金。

14.5 EXISTS 操作符

关联子查询的另一种技术是使用一个名为 EXISTS 的操作符。这个操作符让我们确定关联子查询中是否存在数据。假设你想要找到是哪个客户下了订单。完成这个任务的语句如下所示：

```
SELECT
CustomerName AS 'Customer Name'
FROM Customers
WHERE EXISTS
(SELECT *
FROM Orders
WHERE Customers.CustomerID =Orders.CustomerID)
```

这条语句返回如下内容：

Customer Name
William Smith
Natalie Lopez
Brenda Harper

Adam Petrie 没有出现在结果中，因为他没有下任何的订单。如果关联子查询的 SELECT 语句返回了任意数据，上述语句中的关键字 EXISTS 的计算结果为真。请注意，子查询选取了所有的列（SELECT *）。因为它并不真的关心在子查询中选中哪些特定的数据，所以我们使用星号来返回所有的数据。我们只关心确定子查询中是否存在任何数据。

和前面一样，这条语句中的逻辑可以用另一种方式来表达。如下语句得到了相同的结果，该语句使用了带 IN 操作符的一个子查询：

```
SELECT
CustomerName AS 'Customer Name'
FROM Customers
WHERE CustomerID IN
(SELECT CustomerID
FROM Orders)
```

不用子查询而检索到相同数据的另一条语句如下：

```
SELECT
CustomerName AS 'Customer Name'
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID =Orders.CustomerID
GROUP BY CustomerName
```

14.6　使用子查询作为一个计算的列

子查询的最后一种一般性用途是作为计算的列。假设我们需要看到一列客户以及他们所下的订单的数量。

使用如下语句，无需子查询就可以完成这个任务：

```
SELECT
CustomerName AS 'Customer Name',
COUNT (OrderID) AS 'Number of Orders'
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.CustomerID, CustomerName
ORDER BY Customers.CustomerID
```

其输出如下：

Customer Name	Number of Orders
William Smith	1
Natalie Lopez	3
Brenda Harper	2
Adam Petrie	0

但是，另外一种得到相同结果的方法是，使用子查询作为一个计算的列：

```
SELECT
```

```
CustomerName AS 'Customer Name',
(SELECT
COUNT (OrderID)
FROM Orders
WHERE Customers.CustomerID = Orders.CustomerID)
AS 'Number of Orders'
FROM Customers
ORDER BY Customers.CustomerID
```

请注意，在这个示例中，查询和子查询是相关联的。在 SELECT columnlist 中，把子查询用作一个计算的列。换句话讲，执行完子查询后，它会返回一个单个的值，然后把这个值包含到 columnlist 中。如下是上面的语句的一般格式：

```
SELECT
CustomerName AS 'Customer Name',
SubqueryResult AS 'Number of Orders'
FROM Customers
ORDER BY Customers.CustomerID
```

14.7 小结

在本章中，我们看到了 3 种使用子查询的方式：作为数据源、在查询条件中和用作一个计算的列。我们还看到关联子查询以及非关联子查询的示例。我们实际上只接触到一些使用（以及滥用）子查询的方法。使问题变得复杂的是，许多子查询本可以用其他方式来表达。是否选择使用子查询，取决于个人喜好以及语句的性能。一般来讲，带子查询的 SELECT 语句运行速度要比不带子查询的等价语句慢一些。关于更多使用子查询的优缺点，我们留待在更高级的 SQL 书籍中讨论。

通过使用连接和子查询，我们已经掌握了很多方法来从多个表中选择数据。在第 15 章中，我们将看到把整个查询组合到一条单个 SQL 语句的方法。这是一种特殊类型的逻辑，使我们能够把多个数据集合合并到一个结果中。我们将看到，为了显示彼此之间只有部分关联的数据集，有时需要设置逻辑过程。和子查询一样，设置逻辑的技术为 SQL 语句提供了额外的灵活性和逻辑可能性。

第15章 集合逻辑

关键字：UNION、UNION ALL、INTERSECT 和 EXCEPT/MINUS

在前几章中，各种连接和子查询以不同的方式来处理多个表的数据组合。然而，最终结果仍然是单独的一条 SELECT 语句。现在，我们将把组合表的概念扩展到从几个完整查询组合数据的能力。换句话讲，我们将找到一种方法，只是编写一条 SQL 语句来组合多条 SELECT 语句，以查询数据。

合并查询的概念通常称为集合逻辑（set logic），这是一个来自于数学的术语。我们可以把每个 SELECT 查询作为一个集合来引用。本章所介绍的集合逻辑，将关注 4 种情况。假设我们在 SET A 和 SET B 中有数据，如下是各种可能的情况：

- 数据在 SET A 或 SET B 中；
- 数据在 SET A 和 SET B 中；
- 数据在 SET A 中，但是不在 SET B 中；
- 数据在 SET B 中，但是不在 SET A 中。

我们将从第 1 种情况开始，数据在 SET A 或 SET B 中。我们将看到，这是最普遍和最重要的集合逻辑的可能情况。

15.1 使用 UNION 操作符

在 SQL 中，UNION 操作符用来处理这样的逻辑：选择在 SET A 中或者在 SET B 中的数据。我们先来看一个实例。假设在数据库中有两个表。第 1 个表是包含客户所下的订单数据的 Orders 表，如下所示：

OrderID	CustomerID	OrderDate	OrderAmount
1	1	2009-10-13	10
2	2	2009-10-13	8
3	2	2009-12-05	7
4	2	2009-12-15	21
5	3	2009-12-28	11

第 2 个表是 Returns 表，包含了客户退货的商品数据，如下所示：

ReturnID	CustomerID	ReturnDate	ReturnAmount
1	1	2009-10-23	2
2	2	2009-12-07	7
3	3	2009-12-28	3

有一点很重要，和第 12 章所看到的 Refunds 表不同，这里的 Returns 表并没有直接和 Orders 表相关联。换句话讲，退货没有和特定的订单绑定。在这种情况下，一个客户可能对多笔订单进行退货。

我们想要创建某位客户的所有订单和退货信息的一个报表。如果返回的数据是订单，我们希望按照订单日期来排序；如果返回的数据是退货，我们希望按照退货日期来排序。完成这个任务的语句如下所示。在这条语句中，我们插入了一些额外的空白行，以强调它包含了两条完全独立的 SELECT 语句，只不过是用 UNION 操作符把它们合并到一起而已：

```
SELECT
OrderDate AS 'Date',
'Order' AS 'Type',
OrderAmount AS 'Amount'
FROM Orders
WHERE CustomerID = 2

UNION

SELECT
ReturnDate AS 'Date',
'Return' AS 'Type',
ReturnAmount AS 'Amount'
FROM Returns
```

```
WHERE CustomerID = 2

ORDER BY Date
```

所得到的日期如下：

Date	Type	Amount
2009-10-13	Order	8
2009-12-05	Order	7
2009-12-07	Return	7
2009-12-15	Order	21

如上所示，UNION 操作符把两条完全独立的 SELECT 语句区分开。在最后，还有一条 ORDER BY 子句，它应用于两条 SELECT 语句的结果。前面一条语句的一般格式如下所示：

```
SelectStatementOne
UNION
SelectStatementTwo
ORDER BY columnlist
```

要使用 UNION，需要遵循 3 个规则：

- 使用 UNION 合并到一起的所有 SELECT 语句，在其 SELECT columnlist 中必须有相同数目的列；
- 每个 SELECT columnlist 中的所有列必须以相同的顺序排列；
- 每个 SELECT columnlist 中所有相对应的列，都必须有相同的或可兼容的数据类型。

参照这些规则，注意到前面查询中的两条 SELECT 语句都有 3 个列，3 个列中的每一列数据，都具有相同的顺序和相同的数据类型。

当使用 UNION 时，我们应该使用列的别名为所有相对应的列赋予相同的列名。在我们的示例中，第一条 SELECT 语句的第 1 列有一个初始名 OrderDate，第二条 SELECT 语句的第 1 列有一个初始名 ReturnDate。为了确保最终结果中的第 1 列具有想要的名称，Date 作为列的别名都赋给 OrderDate 和 ReturnDate，这也允许我们在 ORDER BY columnlist 中引用 Date 列。

还需要注意每条 SELECT 语句的第 2 列使用了直接量。我们创建了一个名为 Type 的计算的列，它的值既可能是 Order，也可能是 Return。这让我们可以识别出每一行来自于哪个表。

最后，请注意，ORDER BY 子句应用于组合到一起的两条查询语句的最终结果，正应该如此，因为对单个的查询应用排序是没有意义的。

此刻，有必要回过头来讨论一下，为什么需要使用操作符 UNION，而不是在一条单个的 SELECT 语句中直接把 Orders 表和 Returns 表关联起来。既然这两个表都有 CustomerID 列，为什么不直接用这个列把这两个表关联起来？这种可能性的问题是，两个表彼此之间并没有真正地直接关联。客户可以下订单，客户也可以退货，但是在 Orders 表和 Returns 表之间没有直接的关联。

此外，即使在两个表之间有直接的关联，连接也无法完成需要的任务。使用一个适当的连接，可以把相关的信息放在同一行。然而，在这个示例中，我们想要的是在不同的行中显示订单和退货，必须使用 UNION 操作符才能以这种方式显示数据。

实际上，UNION 允许我们在一条 SELECT 语句中检索不相关的或部分相关的数据。

15.2 UNION 和 UNION ALL

实际上有两种 UNION 操作符：UNION 和 UNION ALL，它们之间只有一点细微的差别。UNION 操作符排除了所有重复的行，UNION ALL 操作符要求包含所有的行，即使它们是重复的。

UNION 操作符排除重复数据的方式，与我们此前见过的关键字 DISTINCT 类似。DISTINCT 应用于一条单独的 SELECT 语句，而 UNION 排除了通过 UNION 组合到一起的所有 SELECT 语句中的重复数据。

在前面的 Orders 表和 Returns 表的示例中，没有数据重复的可能性，所以使用哪种 UNION 并不重要。如下是描述不同情形的一个示例。假设你只对订货日期或退货日期感兴趣。你不想看到同样的日期出现在多个行中。完成这个任务的语句如下所示：

```
SELECT
OrderDate AS 'Date'
FROM Orders
UNION
SELECT
ReturnDate AS 'Date'
FROM Returns
Order by Date
```

所得到的数据如下所示：

Date
2009-10-13
2009-10-23
2009-12-05
2009-12-07
2009-12-15
2009-12-28

请注意，2009-12-28 只出现了一次。即使 Orders 表中有 1 行 2009-12-28，并且 Returns 表中也有 1 行 2009-12-28，但是 UNION 操作符确保 2009-12-28 日期只出现了一次。

我们修改了这条语句，为每条单独的 SELECT 语句添加一个 DISTINCT，但是使用的是 UNION ALL 而不是 UNION，如下所示：

```
SELECT
DISTINCT
OrderDate AS 'Date'
FROM Orders
UNION ALL
SELECT
DISTINCT
ReturnDate AS 'Date'
FROM Returns
ORDER BY Date
```

其输出结果如下：

Date
2009-10-13
2009-10-23
2009-12-05
2009-12-07
2009-12-15
2009-12-28
2009-12-28

DISTINCT 确保每个订单日期或退货日期只列出一次。即使有两个订单的日期都是 2009-10-13，那个日期也只显示一次。然而，UNION ALL 允许在 Orders SELECT 语句和 Returns

SELECT 语句之间有重复的数据。所以你可以看到 2009-12-28 出现了两次，一条来自于 Orders 表，一条来自于 Returns 表。

15.3 交叉查询

操作符 UNION 和 UNION ALL 所返回的数据，处于组合在一起的两条 SELECT 语句中的任意一条所指定的集合中。这就像使用 OR 操作符把两个逻辑集合中的数据组合到一起一样。

SQL 提供了一个名为 INTERSECT 的操作符，它只把在两个集合中都能找到的数据提取出来。INTERSECT 类似于 AND 操作符，它处理本章开头提到过的第二种情况：

- 即数据在 SET A 和 SET B 中。

数据库差异：MySQL

MySQL 不支持 INTERSECT 操作符。

使用同样的 Orders 表和 Returns 表，假设你想要查看的日期既有订单又有退货。完成这个任务的语句如下：

```
SELECT
OrderDate AS 'Date'
FROM Orders
INTERSECT
SELECT
ReturnDate AS 'Date'
FROM Returns
ORDER BY Date
```

其结果如下所示：

Date
2009-12-28

结果中只显示了一行数据，因为只有一个日期既包含在 Orders 表中，又包含在 Returns 表中。

EXCEPT 操作符提供了 INTERSECT 操作的另一种变体。INTERSECT 返回了两个集合中

都存在的数据，而 EXCEPT 返回了在一个集合中存在而在另一个集合中不存在的数据，如本章前边所提到的第三种和第四种情况：

- 数据在 SET A 中，但是不在 SET B 中；
- 数据在 SET B 中，但是不在 SET A 中。

EXCEPT 的一般格式是：

```
SelectStatementOne
EXCEPT
SelectStatementTwo
ORDER BY columnlist
```

这条语句将会显示在 SelectStatementOne 中但是不在 Select-StatementTwo 中的数据。示例如下：

```
SELECT
OrderDate AS 'Date'
FROM Orders
EXCEPT
SELECT
ReturnDate AS 'Date'
FROM Returns
ORDER BY Date
```

其结果如下：

Date
2009-10-13
2009-12-05
2009-12-15

这个数据结果显示了有订单而没有退货的日期。请注意，没有出现 2009-12-28，因为那天产生了退货。

数据库的差异：MySQL 和 Oracle

MySQL 不支持 EXCEPT 操作符。

在 Oracle 中，EXCEPT 操作符的等价操作符是 MINUS。

15.4 小结

在本章中，我们看到把多条 SELECT 语句组合到一条单独语句中的不同方式。最常用的操作符是 UNION，它允许我们把两个不同的集合中的数据组合起来。UNION 类似于 OR 操作符。UNION ALL 是 UNION 的一个变体，它允许显示重复的数据。类似的，INTERCEPT 操作符允许显示在组合的两个集合中都存在的数据，它类似于 AND 操作符。最后，EXCEPT 操作符允许选取在一个集合中存在，但是在另一个集合中不存在的数据。

在第 16 章中，我们将介绍如何把多条 SQL 语句保存到一个存储过程中，以及在这些存储过程中使用参数为 SQL 命令增加一定程度的通用性。我们还会讨论创建定制函数的可能性，并且解释这些函数和存储过程的区别。就像第 13 章所介绍的视图一样，存储过程和定制函数都是可以创建和保存到数据库中的有用对象，它们为系统提供了一些额外的优点和功能。

第16章
存储过程和参数

关键字： CREATE PROCEDURE、BEGIN、EXEC/CALL、ALTER PROCEDURE 和 DROP PROCEDURE

到目前为止，我们检索的所有数据都是由一条单独的语句完成的，即使第 15 章中的集合逻辑也是通过把多条 SELECT 语句合并成一条单独的语句来实现的。现在，我们将讨论一个新的情况，即把多条语句保存到一个叫作存储过程（stored procedure）的单独对象中。

大体上，使用存储过程有两个主要的原因：

- 把多条 SQL 语句保存到一个单独的过程中；
- 把参数和 SQL 语句结合使用。

实际上，存储过程可以由不包含参数的单条 SQL 语句构成。但是，当存储过程包含了多条语句或参数时，其真正价值才体现出来。

存储过程的主题非常复杂。在这个简短的介绍中，我们将重点放在所提到的第 2 个原因：即在存储过程中使用参数。这多少和如何从数据库中以最佳的方式获取数据这个问题直接相关。你将看到，为 SELECT 语句添加参数的能力，成为日常工作中非常有用的一项功能。

使用包含多条语句的存储过程，这超出了本书的范围。基本上，能够在一个存储过程中保存多条语句，意味着我们能够创建复杂的逻辑并将其作为一个独立的事务一次性地执行全部内容。例如，你可能有这样的一个业务需求：接收客户即将下的订单并且在接受该订单之前进行快速的评估。这个评估过程可能包含检查以确保该产品有货，验证该客户有一个良好的信用评分，并且在发货时获得一个初始的估算。如果所有这些评估表明该笔订单有些不对劲，那么就需要添加多条逻辑的 SQL 语句来决定返回什么样的消息。所有这些逻辑可以放在一个单独的存储过程中，它会增强系统的模块化。将所有内容放入到存储过程中，任意调用程序都可以执行其逻辑，并且得到相同的返回结果。

16.1 创建存储过程

在详细了解如何使用存储过程之前，我们先来介绍一下创建和维护存储过程的机制。在各个数据库中，其语法有很大的不同。

在 Microsoft SQL Server 中，创建存储过程的一般格式是：

```
CREATE PROCEDURE ProcedureName
AS
OptionalParameterDeclarations
BEGIN
SQLStatements
END
```

关键字 CREATE PROCEDURE 允许执行一条单独的命令来创建存储过程。过程本身可以包含任意数目的 SQL 语句，并且还能够包含参数的声明。稍后，我们将介绍参数声明的语法，在关键字 BEGIN 和 END 之间列出 SQL 语句。

数据库的差异：MySQL 和 Oracle

在 MySQL 中，创建存储过程的一般格式会稍有点复杂。格式如下：

```
DELIMITER $$
CREATE PROCEDURE ProcedureName()
BEGIN
SQLStatements
END$$
DELIMITER ;
```

当执行多条语句时，MySQL 要求有分隔符。常规的分隔符是一个分号，上述代码的第一行把分隔符从逗号临时改为两个$符号。所需的任何参数，在 CREATE PROCEDURE 行的圆括号中指定。在关键字 BEGIN 和 END 之间列出的每条 SQL 语句，在语句的末尾都必须有一个分号。关键字 END 后边的$符号，表示 CREATE PROCEDURE 命令结束了。最后，把另一条 DELIMITER 语句放在末尾，以便把分隔符改回为分号。

在 Oracle 中创建存储过程的过程有点复杂，这超出了本书的范围。在 Oracle 中，要为一条 SELECT 语句创建存储过程，需要先创建一个叫作包（package）的对象。这个包会包含两个基本部分：包声明（specification）和包体（body）。包声明部分指定如何与包体部分通信。包体部分包含 SQL 语句，它是存储过程的核心。

下面是创建存储过程的一个示例，它可以用来执行如下这条 SELECT 语句：

```
SELECT *
FROM Customers
```

把这个过程命名为 ProcedureOne。在 Microsoft SQL Server 中，创建该存储过程的语句如下：

```
CREATE PROCEDURE ProcedureOne
AS
BEGIN
SELECT *
FROM Customers
END
```

数据库的差异：MySQL

在 MySQL 中，前面的示例如下所示：

```
DELIMITER $$
CREATE PROCEDURE ProcedureOne ()
BEGIN
SELECT *
FROM Customers;
END$$
DELIMITER ;
```

要记住，创建存储过程不会执行任何内容，它只是直接创建了一个过程，以便可以在后边执行它。与表和视图一样，过程在数据库管理工具中是可见的，所以我们可以查看它的内容。

16.2　存储过程中的参数

到目前为止，我们所见到的所有 SELECT 语句都是静态的，因为把它们都编写为以某种特定的方式来检索数据。为 SELECT 语句添加参数的能力，为我们带来了更大的灵活性。

在 SQL 语句中，术语参数（parameter）和其他计算机语言中所使用的术语变量（variable）类似。参数实际上是一个值，通过调用程序可以把它传递给 SQL 语句。在调用时，它可以拥有用户所指定的任意的值。

让我们先来看一个简单的示例。我们有一条 SELECT 语句，它从 Customers 表中检索数据。我们想要让这条 SELECT 语句只选取特定 CustomerID 编号的客户数据，而不要选取所有

的客户。然而，我们不想要在 SELECT 语句中直接写出编号我们希望这条 SELECT 语句足够通用，以便它可以接受所提供的任意的 CustomerID 编号，然后再用这个值来执行查询。不带任何参数的 SELECT 语句非常简单：

```
SELECT *
FROM Customers
```

我们的目标是增加一条 WHERE 子句，它允许我们选取一个特定客户的数据。我们希望这条 SELECT 语句的一般格式如下所示：

```
SELECT *
FROM Customers
WHERE CustomerID = ParameterValue
```

在 Microsoft SQL Server 中，创建这样一个存储过程的语句如下所示：

```
CREATE PROCEDURE CustomerProcedure
(@CustID INT)
AS
BEGIN
SELECT *
FROM Customers
WHERE CustomerID = @CustID
END
```

请注意第 2 行中添加的内容，它在存储过程中指定了参数 CustID。在 Microsoft SQL Server 中，@符号用来表示一个参数。关键字 INT 放在参数的后边，用来表示这个参数将是一个整数值，在 WHERE 子句中使用相同的参数名称。

数据库的差异：MySQL

在 MySQL 中，创建等价的存储过程的命令是：

```
DELIMITER $$
CREATE PROCEDURE CustomerProcedure
(CustID INT)
BEGIN
SELECT *
FROM Customers
WHERE CustomerID = CustID;
END$$
DELIMITER ;
```

请注意，MySQL 不要求用@符号来表示参数。

当执行一个存储过程时，调用程序为参数传递一个值，如果这个值是 SQL 语句的一部分，那么执行该语句。

还需要注意的是，前边所讨论的参数是输入参数。同样，它们包含了传递到存储过程中的值。存储过程也可以包含输出参数，其中包含了传递给调用程序的值。关于如何在存储过程中指定输入参数和输出参数的各种细微差别的讨论已超出了本书的范围。

16.3　执行存储过程

在创建了存储过程之后，如何执行它们？数据库之间的语法各异。Microsoft SQL Server 提供了关键字 EXEC 来运行存储过程。

在 Microsoft SQL Server 中，如下语句将执行 ProcedureOne 存储过程：

```
EXEC ProcedureOne
```

当执行这个存储过程时，它会返回存储过程中所包含的 SELECT 语句的结果。

ProcedureOne 过程没有包含任何参数，所以语法很简单。如何执行带有输入参数的存储过程呢？以 CustID 值为 2 来执行 CustomerProcedure 过程，语句如下所示：

```
EXEC CustomerProcedure
@CustID = 2
```

数据库的差异：MySQL

MySQL 使用关键字 CALL 来执行存储过程，而不使用 EXEC，并且其带参数的存储过程的语法稍有不同。在 MySQL 中，和前面两个 EXEC 语句等价的语句是：

```
CALL ProcedureOne;
CALL CustomerProcedure (2);
```

16.4　修改和删除存储过程

一旦创建了存储过程，就可以修改它。就像使用关键字 ALTER VIEW 来修改视图一样，我们使用关键字 ALTER PROCEDURE 来修改存储过程。其语法与 CREATE PROCEDURE 的语法相同，只不过是用 ALTER 替换了 CREATE。就像不同的数据库的 CREATE PROCEDURE

的语法略有不同一样，不同的数据库的 ALTER PROCEDURE 也不尽相同。

我们已经看到了在 Microsoft SQL Server 中创建这个存储过程的示例：

```
CREATE PROCEDURE CustomerProcedure
(@CustID INT)
AS
BEGIN
SELECT *
FROM Customers
WHERE CustomerID =@CustID
END
```

在创建了这个过程之后，如果你想把这个过程修改为只选取 Customers 表中的前 5 行记录，如下命令可以完成这个任务：

```
ALTER PROCEDURE CustomerProcedure
(@CustID INT)
AS
BEGIN
SELECT
TOP 5 *
FROM Customers
WHERE CustomerID =@CustID
END
```

数据库的差异：MySQL

MySQL 提供了 ALTER PROCEDURE 命令，但是它的功能有限。要在 MySQL 中修改存储过程的内容，需要执行一条 DROP PROCEDURE 命令，然后执行带有新的内容的 CREATE PROCEDURE。

删除一个存储过程甚至更简单一些，就像 DROP VIEW 语句删除视图一样，DROP PROCEDURE 语句删除存储过程。

删除命名为 CustomerProcedure 的存储过程，如下所示：

```
DROP PROCEDURE CustomerProcedure
```

16.5 再谈函数

在第 4 章中，我们讨论过在 SQL 中有内建的标量函数可供使用。例如，我们使用像 LEFT

一样的字符函数和像 ROUND 一样的数学函数。在第 10 章中，我们讨论了像 MAX 一样的聚合函数。

除了 SQL 中的内建函数以外，程序员可以创建自己的函数并把它们保存到数据库中。创建函数的过程和创建存储过程的过程非常相似。SQL 提供了关键字 CREATE FUNCTION、ALTER FUNCTION 和 DROP FUNCTION，它们的用法很像 CREATE PROCEDURE、ALTER PROCEDURE 和 DROP PROCEDURE。

由于这一话题很高级，我们将不提供该功能的特定示例。然而，我们会花点时间来解释使用存储过程和函数之间的不同。存储过程和函数都保存在数据库中，这些实体都作为单独的对象保存到数据库中，就像表或视图一样。保存和修改存储过程，与保存和修改函数非常类似。针对存储过程的 CREATE、ALTER 和 DROP 命令，同样也可以用于函数。

存储过程和函数两者之间的区别在于：如何使用它们以及它们自身的能力。存储过程和函数之间两点主要的区别是：

- **存储过程可以有任意数目的输出参数**。它们甚至可以没有任何输出参数。相反，函数总是必须只有一个输出参数。换句话讲，当你调用一个函数时，总是会得到一个单个的返回值。
- **通过调用程序来执行存储过程**。不能在 SELECT 语句中直接引用存储过程。相反，可以在语句中引用函数，正如我们在第 4 章和第 10 章中所见到的那样。在定义了自己的函数之后，当创建了这个函数之后，你将通过所指定的名称来引用该函数。

16.6　小结

在本章中，我们看到参数的使用可以大大增加处理检索数据的过程的灵活性。例如，参数允许我们把 SQL 语句通用化，以便在执行该语句时，可以指定值以作为查询条件。我们还介绍了如何创建和修改存储过程的基础知识。最后，我们解释了存储过程和用户定义的函数之间的一些区别。

尽管本章的示例关注于数据检索上，但是存储过程和函数在应用数据更新方面也非常有

用。在第 17 章中，我们将完全脱离数据检索的领域，直接开始讨论关于更新数据的问题。维护数据的业务不会像数据检索一样展现出各种分析的可能性，但是它是任何企业必不可少的任务。好在我们介绍过的 SELECT 语句应用的大部分技术，等同于我们在第 17 章中所讨论的修改过程。

第17章 修改数据

关键字：INSERT INTO、VALUES、DELETE、TRUNCATE TABLE 和 UPDATE

之前我们都是讨论从数据库中检索数据，现在我们将转向如何修改数据的话题。修改数据可能有 3 种基本的情况：

- 插入新行到表中；
- 删除表中的行；
- 更新表中指定的行与列中已有的数据。

你可能猜得到，插入和删除行相对比较直接。然而，更新已有的数据会稍微复杂一些。我们将从插入函数和删除函数开始，最后介绍更新。

17.1 修改策略

修改数据的机制非常直接。然而，这个过程的本质表明这是一个充满危险的领域。作为人类，肯定会犯错误。你可能轻易地用一条命令错误地删除几千行记录。或者，你可能应用了大量很难恢复的更新。

作为一个实际问题，有各种可以用来帮助防止灾难性错误的策略。例如，当从一个表中删除行时，你可以使用一种软删除（soft delete）技术，这意味着并不是真的删除行，可以用表中一个特定的列来标记每行是有效的还是无效的。我们只是把一个行标记为无效，而不是删除它。使用这种方法，如果删除是误操作，你可以通过修改有效/无效列很容易地恢复它。

当执行插入时，也可以使用类似的技术。当增加一行记录时，你可以在一个特定的列中标记准确的插入日期和时间。如果稍后认为是错误地添加了行，你可以找到特定时间范围内所有添加的所有的行，并且删除它们。

当更新数据时，问题会更加复杂。通常，用一个单独的表来保存事务所要更新的数据，是明智的选择。如果犯了任何错误，你可以回到事务表中找到数据修改之前和之后的值，并且用来撤销之前的任何错误。

上述策略只是众多可以采用的方法中的一部分。这个话题超出了本书的范围。只是要确保，更新数据时要非常小心。和许多用户友好的桌面程序不同，SQL 中没有撤销命令。

17.2 插入数据

SQL 提供了关键字 INSERT，用来把数据插入到表中。有两种基本的 INSERT 方式：

- 插入在 INSERT 语句中指定的数据。
- 插入用一条 SELECT 语句获取的数据。

我们先通过一个示例来展示如何插入数据，其中，在 INSERT 语句中指定数据值。假设已经有了一个带有如下数据的 Customers 表：

CustomerID	FirstName	LastName	State
1	William	Smith	IL
2	Natalie	Lopez	WI
3	Brenda	Harper	NV

再假设第 1 列 CustomerID 是该表的主键。在第 1 章和第 2 章中，我们介绍过，主键强化了表中每一行应该是唯一的这一要求。我们还介绍过，经常会把主键列指定为自增型的列。这就意味着，当表中增加一行记录时，会自动为该列赋一个数字。

假设把 CustomerID 列定义为自增型的列，这就意味着当我们为 Customers 表增加一行记录时，不用为 CustomerID 列指定值。当表中增加行时，会自动处理该列。我们只需要为其他的 3 个列指定值。

我们试图为这个表增加两位新客户：来自于 Ohio 的 Virginia Jones 和来自于 California 的 Clark Woodland。

如下的语句执行插入操作：

```
INSERT INTO Customers
(FirstName, LastName, State)
```

```
VALUES
('Virginia', 'Jones', 'OH'),
('Clark', 'Woodland', 'CA')
```

插入后，这个表包含的数据如下所示：

CustomerID	FirstName	LastName	State
1	William	Smith	IL
2	Natalie	Lopez	WI
3	Brenda	Harper	NV
4	Virginia	Jones	OH
5	Clark	Woodland	CA

我们依次来解释这些语句。首先，请注意，关键字 VALUES 用作插入到表中的值的列表的前缀。这条语句以隔开的一对圆括号列出每行数据。Ohio 的 Virginia Jones 在一组圆括号中，Clark Woodland 在另一组圆括号中。两组圆括号用逗号隔开。如果只需要添加一行，那么只需要一组圆括号即可。

数据库的差异：Oracle

Oracle 不允许在关键字 VALUES 后边指定多个行。前面的示例需要分为两条语句，诸如：

```
INSERT INTO Customers
(FirstName, LastName, State)
VALUES ('Virginia', 'Jones', 'OH');

INSERT INTO Customers
(FirstName, LastName, State)
VALUES ('Clark', 'Woodland', 'CA');
```

还要注意：关键字 VALUES 后边的值的顺序，要与 INSERT TO 后边的 columnlist 中所列出的列的顺序相对应。列出列的顺序不一定要和它们在数据库中的顺序一致。换句话讲，上面的插入语句的任务也可以用如下语句来完成：

```
INSERT INTO Customers
(State, LastName, FirstName)
VALUES
('OH', 'Jones', 'Virginia'),
('CA', 'Woodland', 'Clark')
```

在上述的 INSERT 语句中，我们把 State 列放在最前面而不是最后面。再说一次，列的放置顺序并不重要。

INSERT INTO 语句的一般格式如下：

```
INSERT INTO table
(columnlist)
VALUES
(RowValues1),
(RowValues2)
(repeat any number of times)
```

columnlist 中的列需要和 RowValues 中的列相对应。

如果 RowValues 中所有的列都和它们在数据库中物理存在的顺序一致，并且如果表中没有自增型列，那么 INSERT INTO 语句不需要指定 columnlist 就可以执行。然而，我们强烈反对这种做法，因为这更容易出错。

如果在 INSERT 语句中没有指定所有的列，那会怎么样呢？很简单，那些没有指定的列会赋以 NULL 值。例如，假设我们想要在 Customers 表中为名为 Tom Monroe 的客户添加一行。然而，我们不知道 Tom 的 State。INSERT 语句如下所示：

```
INSERT INTO Customers
(FirstName, LastName)
VALUES
('Tom', 'Monroe')
```

执行插入以后，他在表中的行如下所示：

CustomerID	FirstName	LastName	State
6	Tom	Monroe	NULL

因为我们没有为这个新行指定 State 列的值，所以给它赋了一个 NULL 值。

INSERT INTO 语句有两种格式。第 2 种格式适用的情形是，插入的数据是由一条 SELECT 语句获取的，这就意味着在关键字 VALUES 的后面，不是列出值，而是使用获得类似值的 SELECT 语句。

假设我们有另一个名为 CustomerTransactions 的表，它包含了我们想要插入到 Customers 表的数据。CustomerTransactions 表如下所示：

CustomerID	State	Name1	Name2
1	RI	Susan	Harris
2	DC	Michael	Blake
3	RI	Alan	Canter

如果我们想要把 CustomerTransactions 表中 Rhode Island 州的所有客户添加到 Customers 表中，如下语句将会完成这一任务：

```
INSERT INTO Customers
(FirstName, LastName, State)
SELECT
Name1,
Name2,
State
FROM CustomerTransactions
WHERE State = 'RI'
```

执行完这条 INSERT 语句以后，Customers 表包含的数据如下所示：

CustomerID	FirstName	LastName	State
1	William	Smith	IL
2	Natalie	Lopez	WI
3	Brenda	Harper	NV
4	Virginia	Jones	OH
5	Clark	Woodland	CA
6	Tom	Monroe	NULL
7	Susan	Harris	RI
8	Alan	Canter	RI

上述 INSERT 语句直接用 SELECT 语句替代了我们前面所见到的 VALUES 子句。正如我们所预期的，并没有把 Michael Blake 添加到 Customers 表，因为他不在 Rhode Island。还要注意，Customers 表中的列名和 CustomerTransactions 表中的列名并不相同。列名并不重要，只要列是以正确的顺序列出的就可以了。

17.3　删除数据

删除数据要比添加数据简单很多。DELETE 语句用来处理删除。当执行 DELETE 语句时，它会删除整个行，而不是行中的单个的列。一般格式如下：

```
DELETE
FROM table
WHERE condition
```

如下是一个简单的示例。假设我们想要删除前面提到的 Customers 表中的 Rhode Island 地区的客户的行。完成这个任务的语句如下所示：

```
DELETE
FROM Customers
WHERE State = 'RI'
```

这就是完成它的全部语句。如果在执行上述的 DLETE 语句之前，你想要测试它的结果，可以直接用一条 SELECT 语句替换 DELETE 语句，如下所示：

```
SELECT
COUNT (*)
FROM Customers
WHERE State = 'RI'
```

这将提供要删除的行的数目，这为删除操作提供了某种级别的验证。

删除数据还有一个值得一提的可选项。如果你想删除一个表中所有的数据，可以使用 TRUNCATE TABLE 语句来删除所有内容。相对于 DELETE 语句，TRUNCATE TABLE 语句的优势在于它更快。和 DELETE 不同，TRUNCATE TABLE 不记录事务的结果。我们还没有介绍数据库日志过程，但是这是大部分数据库都会提供的一种功能，它允许数据库管理员在系统崩溃的时候或出现其他类似问题时恢复数据。

如果你想要删除 Customers 表中所有的行，可以执行如下的语句：

```
TRUNCATE TABLE Customers
```

这会得到和如下语句相同的结果：

```
DELETE
FROM Customers
```

DELTE 和 TRUNCATE TABLE 之间的另一个细微差别是，TRUNCATE TABLE 重新设置了用于自增型的列的当前值，DELETE 语句则不会影响那些值。

17.4 更新数据

更新数据的过程包括指定更新哪个列，以及选中行的逻辑。UPDATE 语句的一般格式如

下所示：

```
UPDATE table
SET Column1 = Expression1,
Column2 = Expression2
(repeat any number of times)
WHERE= condition
```

这条语句和基本的 SELECT 语句很类似，只是关键字 SET 用来把新的值赋给指定的列。WHERE 条件指定要更新哪些行，但是 UPDATE 语句可以同时更新多个列。如果要更新多个列，关键字 SET 只会出现一次，但是所有更新表达式必须要使用逗号分隔开。

我们先来看一个简单的示例，假设想要把客户 William Smith 的名字改为 Bill Smythe。在 Customers 表中，他的行当前如下所示：

CustomerID	FirstName	LastName	State
1	William	Smith	IL

完成这个任务的 UPDATE 语句如下所示：

```
UPDATE Customers
SET FirstName = 'Bill',
LastName = 'Smythe'
WHERE CustomerID = 1
```

当执行完这条语句后，Customers 表中的这一行如下所示：

CustomerID	FirstName	LastName	State
1	Bill	Smythe	IL

请注意，State 列的值没有变化，因为那列没有包括在 UPDATE 语句中。还要注意 WHERE 子句是必须的。没有 WHERE 子句，就会把表中的每一行都修改为 Bill Smythe。

17.5　相关的子查询的更新

前边的 UPDATE 示例很简单但是不完全现实。更常见的 UPDATE 的示例，包括那种根据一个表中的数据来更新另一个表中的数据的情况。假设我们有如下的 Customers 表：

CustomerID	State	Zip
1	IL	60089
2	CA	92802
3	WI	53718
4	DC	20024
5	FL	32801

如下的 CustomerTransactions 表保存了已有客户的最近变化：

TransactionID	CustomerID	State	Zip
1	1	IL	60090
2	2	NV	89109
3	5	FL	32810

我们认为 Customers 表保存了客户的主要数据源。要根据 CustomerTransactions 表来完成 Customers 表的数据的更新，我们需要使用第 14 章介绍的和子查询相关的技术。因为 UPDATE 语句只能指定修改单个的一个表，所以我们需要相关的子查询。我们不能仅仅把多个表关联起来并让该语句工作。我们需要在关键字 SET 之后使用一个相关的子查询来表示数据来源。

可以用如下语句来更新 Customers 表中的 State 列和 Zip 列，数据来自于 CustomerTransactions 表中的事务。因为这条语句相当复杂，我们插入一些空行以便于随后讨论这条语句的 4 个部分。

```
UPDATE Customers
SET Customers.State =
(SELECT CustomerTransactions.State
FROM CustomerTransactions
WHERE CustomerTransactions.CustomerID = Customers.CustomerID),

Customers.Zip =
(SELECT CustomerTransactions.Zip
FROM CustomerTransactions
WHERE CustomerTransactions.CustomerID = Customers.CustomerID)

WHERE EXISTS
(SELECT *
FROM CustomerTransactions
WHERE CustomerTransactions.CustomerID = Customers.CustomerID)
```

执行完这条 UPDATE 语句后，Customers 表包含了如下的数据：

CustomerID	State	Zip
1	IL	60090
2	NV	89109
3	WI .	53562
4	MD	20814
5	FL	32810

我们详细分析一下这条 UPDATE 语句。该语句的第 1 部分由第 1 行组成，表示要对 Customers 表进行更新。

该语句的第 2 部分指定了如何更新 State 列。更新基于如下的相关的子查询：

```
SELECT CustomerTransactions.State
FROM CustomerTransactions
WHERE CustomerTransactions.CustomerID =
Customers.CustomerID
```

我们可以知道这是一条相关的子查询，因为如果试图单独执行这条 SELECT 语句，会出错。子查询从 CustomerTransactions 表中获取数据，并且通过 CustomerID 来匹配两个表。

这条语句的第 3 部分等同于第 2 部分，只是这些行与更新 Zip 列有关。还要注意关键字 SET 只需要指定一次，在第 2 部分中指定了，在第 3 部分不需要再次指定。

最后这部分 WHERE 子句中的逻辑，和整个 UPDATE 语句的查询逻辑相关。EXISTS 操作符和另一个相关的子查询一起使用，以判断针对 Customers 表中的每个 CustomerID，CustomerTransactions 表中是否存在于对应的行。

没有 WHERE 子句的话，更新语句会把 CustomerID 为 3 和 4 的客户的 State 列和 Zip 列错误地修改为 NULL 值，因为这些客户在 CustomerTransactions 表中没有行。实际上，这条 WHERE 子句中相关的子查询使得我们只对在 CustomerTransactions 表中有数据的客户进行更新。

由此可以推断出，使用相关的子查询进行更新的主题相当复杂。因此，这个话题通常会超出本书的范围。然而，我们介绍这个示例，只是为了展示数据更新涉及较大的复杂性这一思路。此外，请注意，相关的子查询在删除语句中的用法也是类似的。

17.6 小结

本章给出了更新数据的各种方法的一个概览。执行简单的插入、删除和更新的机制相对比较直接。然而，相关的子查询技术，对于现实世界中的更新和删除来说经常是必要的，而并非权宜之计。此外，对数据进行更新的整个概念都是必需的实践。当执行任何类型的更新时，要十分谨慎地对待 SQL 那种使用一个单独的命令便能够更新数千行数据的能力。在应用任何数据更新之前，要小心规划好撤销所有更新的过程。

现在我们已经讨论过表中数据的更新，接下来将进一步讨论这些表自身。在第 18 章中，我们将介绍创建表的机制，以及在那些表中正确存储数据所需的所有属性。为此，我们将更加细致地回顾在第 1 章中接触过的一些主题，诸如主键和外键。到目前为止，我们都是假设表直接可供使用。在介绍完这个主题之后，你将会对如何创建表来保存数据有更好的想法。

第18章 维护表

关键字： CREATE TABLE、DROP TABLE、CREATE INDEX 和 DROP INDEX

在本章中，我们把注意力从数据检索和数据修改转向数据设计。到目前为止，我们都是假设表直接存在并且可供使用。然而，在正常情况下，我们需要先创建表，然后才可以访问数据。现在，我们来介绍如何创建表和维护表。

我们会介绍之前接触过的一些话题，诸如主键和外键，但是现在将深入这些领域的细节，并且会引入表索引的相关话题。

18.1 数据定义语言

在第1章中，我们提到SQL有3种主要组成部分：数据操纵语言（Data Manipulation Language，DML）、数据定义语言（Data Definition Language，DDL）和数据控制语言（Data Control Language，DCL）。到目前为止，我们所讨论的大部分都是 DML。DML 语句允许我们通过检索、插入、删除和更新来操纵关系数据库中的数据。通过 SELECT 语句、INSERT 语句、DELETE 语句和 UPDATE 语句来执行这些操作。

尽管重点在 DML，但是我们已经看到过一些 DDL 的示例。CREATE VIEW 和 CREATE PROCEDURE 是 DDL，与这些语句相关的 ALTER 和 DROP 版本也是 DDL。

CREATE VIEW 语句和 CREATE PROCEDURE 语句是 DDL，因为它们只允许操作数据库本身。对于数据库中的数据，它们什么都不做。

在本章中，我们将对一些额外的 DDL 语句做一个简短的概览，这些 DDL 语句用来创建和修改表以及索引。

每个数据库都有组织自己对象的不同方式，因此可用的 DDL 语句也有所不同。例

如，针对 Databases、Events、Functions、Indexes、Logfile Groups、Procedures、Servers、Tables、TableSpaces、Triggers 和 Views 这些对象类型，MySQL 有 11 种不同的 CREATE 语句。

Oracle 有超过 30 种创建不同的对象类型的 CREATE 命令。Microsoft SQL Server 有超过 40 种不同的 CREATE 命令用于创建其对象类型。

事实上，对数据库对象（诸如视图和表）的大部分修改，都可以通过图形用户界面（graphical user interface，GUI）来完成，各个软件厂商用 GUI 来管理自己的软件。我们往往没有必要学习任何的 DDL，因为通常可以使用软件的 GUI 来操作 DDL。

然而，了解操作数据对象的一些关键语句还是很有用的。我们已经看过一些可以修改视图和存储过程的语句。在本章中，我们介绍通过 DDL 修改表和索引的一些可能的方法。

18.2 表属性

在第 1 章中，我们简单介绍了数据库表的一些属性，诸如主键、外键和数据类型。在第 2 章中，我们介绍了自增型的列。

如前所述，SQL DDL 针对数据库对象的许多类型提供了 CREATE 语句。在第 13 章中，我们介绍了操作存储过程和视图的 CREATE PROCEDURE 和 CREATE VIEW 语句。

我们现在将注意力转回到表。表或许是数据库中最重要的和最必不可少的对象类型。没有了表，也就没有了真正重要的东西。数据库中的所有数据物理地存储在表中，大部分其他的对象类型以各种方式和表相关联。视图提供了一个虚拟的表，存储过程通常在表中的数据之上执行，函数能以特定的规则来操控表中的数据。

现在，我们将关注最初是如何创建表的。有大量属性可以与表的定义相关联。我们将对一些重要的属性给出一个概览并且讨论它们的含义。

表属性的主题也和数据库设计这个更大的话题相关，我们会在第 19 章介绍数据库设计的话题。现在，我想要重点关注表本身能做什么这一机制。

对于 Microsoft SQL Server、MySQL 和 Oracle 来讲，如何具体地设计表和修改表有很大的不同。我们主要介绍表在所有 3 种数据库中的通用属性。

18.3　表的列

表可以设计为包含任意数目的列，每个列有一些针对该列的具体属性。列的第 1 个并且是最明显的属性就是列的名称，表中的每个列必须要有一个唯一的名称。

列的第 2 个属性是数据类型，这是本书第 1 章中介绍过的一个主题。我们已经描述了以下 3 种值得注意的主要数据类型：数字、字符与日期/时间。数据类型是决定每个列可以包含什么样的数据的关键。

列的第 3 个属性是，是否可以把它定义为自增型的列。在第 2 章中，我们简单地介绍了这种属性，并且在第 17 章中做了进一步的讨论。基本上，自增型的列意味着当表中每增加一行，会自动按照升序序列将一个数值赋给该列。自增型的列通常和主键一起使用，但是，也可以将其分配给一个普通的列。

请注意，术语自增型（auto-increment）是 MySQL 中特定的用法。Microsoft 使用术语 identity 表示相同类型的属性。

数据库的差异：Oracle

Oracle 没有自增型的属性。相反，Oracle 要求你把一个列定义为序列（sequence），然后创建一个触发器（trigger），以连续的值来填充该列。这个过程超出了本书的讨论范围。

列的第 4 个属性是，这个列是否允许包含 NULL 值，默认允许 NULL 值。如果你不允许一个列包含NULL值，一般可以通过对该列的描述（column description）使用关键字NOT NULL来指定。

我们将讨论的最后一个列属性是，是否给列赋一个默认值。当添加行时，如果没有为这个列提供一个值，就为该列自动赋一个默认值。例如，你的大部分客户在美国，你可能想要给包含国家代码的列指定一个默认值——U.S.。

18.4　主键和索引

我们再次回到主键的话题，并且介绍和表索引相关的属性。

索引是一种物理结构，可以为数据库表中任意的列添加索引。索引的目的是，当 SQL 语句中包含该列的时候，可以加速数据的检索。索引中的真实数据是隐藏的，但是基本上索引包含了一种结构，可以维护该列的排序信息，因此当需要查询指定的值时，就可以进行更加快速的检索。

对列进行索引的一个缺点是，在数据库中，索引需要更多的存储硬盘。另一个负面因素是，索引通常会降低相关的列的数据更新速度。这是因为，任何时候插入或者修改一行记录时，索引都必须重新计算该列中的值的正确的排列顺序。

可以对任意的列进行索引，但是只能指定一个列作为主键。指定一个列作为主键意味着两件事情：首先是这个列成为了索引，其次保证这个列包含唯一的值。

第 1 章曾经介绍过，主键为数据库用户提供了两个主要的优点：他们能够唯一标识表中的每一行；他们允许你很容易地和另一个表进行关联。现在，我们可以添加上第 3 个优点：通过成为索引，主键使得相关的列的数据行检索速度更快。

拥有主键的主要原因是，要保证表中所有的行在该列都拥有唯一的值。总是要有一种方法能够标识要更新和删除的唯一的行，而主键确保了能够实现这一点。

主键实际上可以跨越多个列，可以由两个或三个列组成。如果主键包含了多个列，只是意味着所有这些列共同包含了唯一的值。这种类型的主键通常叫作复合主键（composite primary key）。我们来看使用复合主键的一个示例，假设你有一个 Movies 表。你想要有一个键来唯一地定义表中的每部电影。你想要使用电影名称作为这个键，而不是使用整数值 MovieID 作为这个键。然而，问题是电影有时会使用相同的名称。为了解决这个问题，你可能使用两个列，也就是电影名称和出品年份，形成一个复合主键来唯一地定义每部电影。

因为主键必须包含唯一的值，所以从不允许它们包含 NULL 值，必须要指定该列的值。

最后，人们经常把主键指定为自增型的列。通过使得主键自增加，数据库程序员不需要再担心要为列赋一个唯一的值。自增型的特色满足了这个需求。

18.5 外键

SQL 数据还可以把具体的列定义为外键（foreign key）。外键是从一个表中的一个列到另一个不同的表中的一个列的直接引用。当设置一个外键时，会要求指定两个列。配置了外键列的表，通常称为位于子表（child table）中。而另一个表中被引用的列则被称为是在

父表（parent table）中。

例如，假设我们有一个 Customers 表，把它的 CustomerID 列设置为主键。我们还有一个主键为 OrderID 的 Orders 表，该表还有一个 CustomerID 列。我们想把 Orders 表中的 CustomerID 列设置为一个外键，它引用 Customer 表中的 CustomerID 列。在这个示例中，Orders 表是子表，而 Customers 表是父表。

当设置外键时，你就能够为父表中涉及到更新和删除的行设置一些具体的行为。3 种最常见的行为是：

- No Action
- Cascade
- Set Null

我们继续以 Customers 表和 Orders 表为例。最常见的行为是 No Action，如果没有具体指定的话，它是默认行为。如果把 Orders 表中的 CustomerID 列的更新设置为 No Action，就意味着，任何时候想要更新父表中的 CustomerID 列的话，都要做检查。如果试图更新 CustomerID，可能会导致子表中任意行指向一个不存在的值，No Action 防止了这种情况出现。如果为删除指定了 No Action，也同样有效。这就确保了当在两个表中都使用 CustomerID 时，Orders 表中的所有行正确地指向 Customers 表中的已有的行。

外键的第 2 个可选的指定行为是 Cascade。它意味着，当更新父表中的值时，如果影响到子表中的行，那么会自动地更新子表中的所有的行，以反映出父表中的新值。类似的，如果删除了父表中的行，并且如果影响到了子表中的行，那么就自动删除子表中受影响的行。

外键的第 3 个可选的指定行为是 Set Null，它意味着，当更新或删除父表中的一个值时，如果会影响到子表中的行，它会自动地把子表中所有受影响的行的外键更新为包含一个 NULL 值。

18.6　创建表

可以使用 CREATE TABLE 语句在数据库中创建一个新表。各个数据库之间的语法和可用的功能各不相同。我们将介绍一个简单的示例，它创建带有如下属性的一个表：

- 表的名称是 MyTable。

- 表中的第 1 列命名为 ColumnOne，并且将其定义为主键。把该列定义为 INT (integer) 数据类型，并且是自增型的列。
- 把表中的第 2 列命名为 ColumnTwo，并且定义为 INT 类型。该列不允许是 NULL 值。还把该列定义成一个外键，把删除和更新都指定为 No Action，另一个名为 RelatedTable 表中的相关联的列名是 FirstColumn。
- 把表中第 3 列命名为 ColumnThree，并且定义为 VARCHAR 类型，长度为 25。该列允许有 NULL 值。
- 把表中第 4 列命名为 ColumnFour，并且定义为 FLOAT 类型，允许有 NULL 值。给该列一个默认值为 10。

如下是在 Microsoft SQL Server 中完成该任务的 CREATE TABLE 语句：

```
CREATE TABLE MyTable
(ColumnOne INT IDENTITY (1,1) PRIMARY KEY NOT NULL,
ColumnTwo INT NOT NULL
REFERENCES RelatedTable (FirstColumn),
ColumnThree VARCHAR (25) NULL,
ColumnFour FLOAT NULL DEFAULT (10) )
```

数据库的差异：MySQL 和 Oracle

在 MySQL 中，相同的 CREATE TABLE 语句如下所示：

```
CREATE TABLE MyTable
ColumnOne INT AUTO_INCREMENT PRIMARY KEY NOT NULL
ColumnTwo INT NOT NULL,
ColumnThree VARCHAR (25) NULL,
ColumnFour FLOAT NULL DEFAULT 10,
CONSTRAINT FOREIGN KEY (ColumnTwo)
REFERENCES 'RelatedTable' (FirstColumn) );
```

在 Oracle 中，相同的 CREATE TABLE 语句如下所示：

```
CREATE TABLE MyTable
(ColumnOne INT PRIMARY KEY NOT NULL
ColumnTwo INT NOT NULL,
ColumnThree VARCHAR2 (25) NULL,
ColumnFour FLOAT DEFAULT 10 NULL,
CONSTRAINT "ForeignKey" FOREIGN KEY (ColumnTwo)
REFERENCES RelatedTable (FirstColumn) );
```

如前所述，Oracle 不允许有自增型的列。

在创建了一个表之后，可以用 ALTER TABLE 语句修改表的具体属性。因为该语句的复杂性以及在数据库之间差异巨大，所以本书不会介绍 ALTER TABLE 的语法。

作为一个示例，如果你想修改 MyTable 表，把 ColumnThree 列从表中删除，则需要执行如下的 ALTER TABLE 语句：

```
ALTER TABLE MyTable
DROP COLUMN ColumnThree
```

删除表的语法很简单。要删除 MyTable，执行如下语句：

```
DROP TABLE MyTable
```

18.7　创建索引

SQL 提供了 CREATE INDEX 语句，用来在创建了表之后，创建索引。可以使用 ALTER TABLE 语句来添加和修改索引。

如果你想要为 MyTable 表的 ColumnFour 列添加一个新的索引，Microsoft SQL Server 中的语法如下：

```
CREATE INDEX Index2
ON MyTable (ColumnFour)
```

这会创建一个名为 index2 的索引。

要删除一个索引，直接执行如下所示的 DROP INDEX 语句：

```
DROP INDEX Index2
ON MyTable
```

数据库的差异：Oracle

在 Oracle 中，DROP INDEX 的等价语句如下：

```
DROP INDEX Index2;
```

18.8　小结

添加或修改表和索引的 SQL 语句很复杂，但是我们无需了解细节。数据库软件通常提供

图形化工具来修改表的结构，而不是一定要使用 SQL 语句。本章介绍的最重要的概念是各种表的属性，包括对彼此间相互关联的索引、主键和外键的理解。

在第 19 章中，我们将从创建表这种平凡的任务转移到数据库设计这一更为广泛的话题。就像通常在访问数据之前要创建表一样，在数据库中创建表之前，通常要先设计数据库。所以在某种意义上，我们把通常应该在数据检索之前介绍的话题，按照相反的顺序来介绍。当然，数据库的具体设计是通过 SQL 提供高质量结果的能力的必不可少的一部分。如果数据库设计得很差，你将在随后试图检索数据的时候受阻。第 19 章对数据库设计原理基本知识的讨论，对确保设计一个高质量的数据库大有帮助。

第 19 章
数据库设计原理

在第 1 章中，我们介绍过这样的概念：关系数据库是一个数据集合，保存了任意数目的表。假设表彼此之间以某些方式相互关联。在第 18 章中，我们明确了数据库设计人员可以指定外键，以确保正确地维护表之间的某种关系。

然而，即使我们掌握了主键和外键的知识，仍然没有解决一个基本问题：即如何开始一个数据库的设计。要解决的主要问题如下所示：

- 如何把数据组织到一组相关的表中？
- 在每个表中应该放什么样的数据元素？

一旦定义了表和数据元素，那么数据库管理员就可以继续创建外键、索引以及合适的数据类型。

对上述两个问题，从没有一个正确的答案。除了每个企业和每个业务都是唯一的这一因素以外，任何业务都没有绝对的解决方案也是一个原因。很大程度上取决于这个业务想要它们的数据设计有多么灵活。另一个明显的因素是当前已存在的数据，很少有企业可以抛开已有数据，从零开始设计它们的数据库。

尽管有这些限制，某些数据库设计原理还是随着时间的演变而发展，以指导我们探索最优的设计架构。从头来说起，关系数据库设计最有影响的架构师是 E.F. Codd，他于 1970 年出版了开创性的著作 *A Relational Model of Data for Large Shared Data Banks*。该书为我们现在称作“关系模型（relational model）”和“规范化（normalization）”的概念奠定了基础。

19.1 规范化的目的

术语规范化（normalization）指的是允许设计人员把非结构化的数据放入到一组设计合理

的表和数据元素中这一具体的过程。

理解规范化的最好方式是说明它不是什么。为了做到这一点，我们首先展示一个有许多明显问题、设计很糟糕的表。如下是一个名为 Grades 的表，它试图展示学生在测验中所获得的所有成绩的信息。每行展示一位学生的成绩。

Test	Student	Date	Total Points	Grade	TestFormat	Teacher	Assistant
Pronoun Quiz	Amy	2009-03-02	10	8	Multiple Choice	Smith	Collins
Pronoun Quiz	Jon	2009-03-02	10	6	Multiple Choice	Jones	Brown
Solids Quiz	Beth	2009-03-03	20	17	Multiple Choice	Kaplan	NULL
China Test	Karen	2009-02-04	50	45	Essay	Harris	Taylor
China Test	Alex	2009-03-04	50	38	Essay	Harris	Taylor
Grammar Test	Karen	2009-03-05	100	88	Multiple Choice, Essay	Smith	Collins

我们先列出了表中每个列打算提供的信息。这些列是：

- Test：对考试或小测验的描述。
- Student：参加考试的学生。
- Date：考试日期。
- TotalPoints：这门考试的满分。
- Grade：学生得到的分数。
- TestFormat：考试的形式是作文、多选或两者皆有。
- Teacher：出题老师。
- Assistant：在这门课程中老师的助教。

我们假设这个表的主键是由 Test 列和 Student 列组成的复合主键。表中的每一行表示具体的考试和具体的学生的一次成绩。

现在，我们讨论这个表的两个明显的问题。首先，某些数据不必要地重复。例如，你可以看到在 2009 年 3 月 2 日，Pronoun Quiz 的满分是 10 分。然而，问题是，对于这个小测验，该信息会在每行中重复一次。如果我们可以针对特定的小测验直接地记录一次满分，效果会更好。

第 2 个问题是某个单个的单元格中的数据重复。我们有一行的 TestFormat 列是 Multiple

Choice 和 Essay。这么做是因为这次考试有两种类型的试题，但是这会使得数据很难使用。如果我们想要检索所有试题为 Essay 的考试，该怎么办呢？

这个表的主要问题是，它试图把所有信息都放入一个表中。把这个表中的信息拆分成多个实体，诸如 students、grades 和 teachers，用单独的表来表示每个实体，效果会更好。然后，可以用 SQL 的功能把表连接起来，以检索任意需要的信息。

根据这些讨论，现在让我们来概括一下规范化的过程所希望完成的任务。有两个主要的目标：

- **消除冗余的数据**。前面的示例明确地阐述了冗余数据的问题。但是，这个问题为什么这么重要？在多个行中列出相同的数据有什么问题？除了明显的重复工作以外，冗余还减少了灵活性。当数据重复时，意味着对特定数据的任何修改都会影响到多个行，而不是一行。
- **消除插入、删除和更新异常**。冗余数据的问题也和消除插入、删除和更新异常这个目标相关。例如，假设某位老师结了婚并改了名字。现在，你需要更新包含她的名字的所有行。因为冗余地存储了数据，所以你需要更新大量的数据，而不是只改一行。

  还有插入和删除异常。例如，假设你只是雇佣了一个新的老师来教音乐。你想要将此记录到数据库中。然而，这位老师还没有给出任何考试，也就没有地方能够放这一信息，因为没有为老师提供一个专门的实体表。

  类似的，如果你想删除一个行，会引起删除异常，这么做将会消除一些相关的信息。我们使用另一个示例，如果你有一个图书的数据库并且想要删除 Nathaniel Hawthorne 所著的一本书的那一行，如果这是 Hawthorne 先生唯一的一本书，那么删除这一行不仅会删除这本书，而且连 Nathanial Hawthorne 是你将来可能获取的其他书籍的作者这一信息也会删除。

19.2　如何规范化数据

我们已经抛出规范化（normalization）这个概念有一阵子了，现在是时候介绍它更为具体的含义了。

这个术语本身是由 E.F. Codd 创造出来的，它指的是在数据库设计中消除冗余和更新异常所采取的一系列的推荐步骤。这些涉及规范化过程的步骤通常称作第一范式、第二范式和第三范式等。尽管我们把某些特殊的步骤描述为第六范式，但是一般只有第一范式、第二范式

和第三范式。当数据是第三范式的形式时，通常来说，它已经足够规范化了。

我们不会介绍把数据转换成第一范式、第二范式和第三范式的全套规则和流程。还会有文章告诉你更具体的过程，为你展示如何把数据先转换成第一范式，然后转换成第二范式，并最终转换成第三范式。

我们要总结把数据转换成第三范式的规则。实际上，有经验的数据库管理员可以从非结构化的数据直接跳到第三范式，而不必遵循每一个中间过程。这里，我们也要做同样的事情。

规范化数据的 3 个主要规则，如下所示：

- **消除重复数据**。这条规则意味着不允许多值属性。在前面的示例中，我们不允许一个数据单元格中存在像“Multiple Choice, Essay”这样的值。如果一个数据单元格中存在多个值，当通过给定的值来检索数据时，会带来明显的问题。

  这条规则的核心就是不允许有重复的列。在我们的示例中，按照这条规则，数据库应该这样设计：使用名称分别为 TestFormat1 和 TestFormat2 的两个不同的列，而不是名为 TestFormat 的一个单独的列。通过这种方法，我们可以把 Multiple Choice 值放在 TestFormat1 列，把 Essay 值放在 TestFormat2 列。但是，这样也是不允许的。我们不想有重复的数据，无论是以单个列中的多个值的形式，还是用多个列来处理相似的数据。

- **消除部分依赖**。这条规则所指的情况是，表中的主键是复合主键，也就是主键由多个列组成。这条规则阐明表中没有列能够只和主键的一部分相关。

  让我们用一个示例来说明。前面提到过，Grades 表中的主键是由 Student 列和 Test 列组成的复合主键，问题出在 TotalPoints 列。TotalPoints 事实上是与考试相关的一个属性，而和学生无关。这条规则规定，表中所有的非键列指向的是整个键，而不是键的一部分。实际上，部分依赖性表示表中的数据和多个实体相关联。

- **消除过度依赖**。这条规则所指的情况是，表中的列指向的不是主键，而是指向同一表中的另一个非键的列。在这个示例中，Assistant 列实际上是 Teacher 列的一个属性。Assistant 列与老师相关而和主键（考试或学生）无关，这表明该信息不属于这个表。

  所以我们看到了问题，并且讨论了修正数据的规则。正确的数据库设计修改是如何确定的？这要依靠经验来判断。对于这个问题，通常不是只有一个解决方案。

  这就是说，以下是这个设计问题的一种解决方案。在这个新的设计中，从最初的一个

表中创建出了数个表，现在所有数据都是规范化形式的了。图 19.1 展示了新设计中的表，但是没有显示数据。

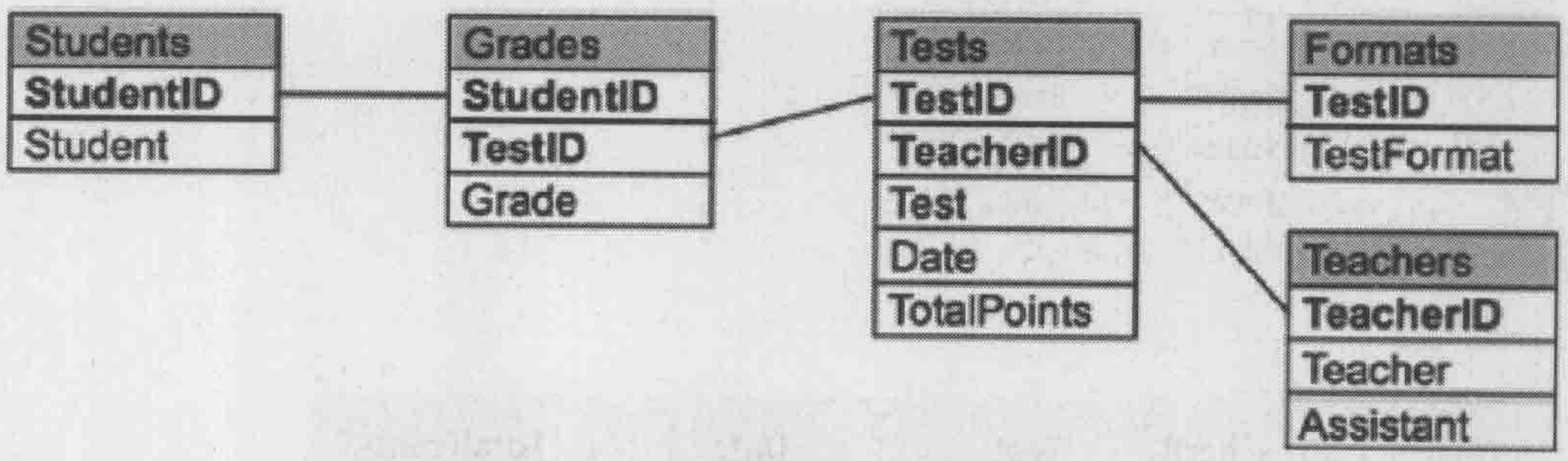


图 19.1　规范化设计

每个表的主键用粗体字表示，表中添加了自增型的 ID 列，允许定义表之间的关系，所有其他的列都和之前显示的一样。

需要重点注意的是，把示例中所讨论到的每个实体都分解成了单独的表。Students 表保存了每位学生的相关信息，表中唯一的属性是学生名称。Grades 表保存了每次成绩的相关信息，该表有复合主键 StudentID 和 TestID，因为每次成绩都与学生和特定的考试相关。

Test 表保存了每次考试的相关信息，诸如考试日期、TeacherID、考试说明以及考试满分。

Formats 表保存了考试形式的相关信息。每次考试都会添加多行数据，以显示这次考试是多选、作文还是两者皆有。

Teachers 表保存了每位老师的相关信息，包括老师的助教的信息（如果有的话）。

和最初的 Grades 表中的数据相对应，这些新表中包含的数据如下所示：

Students 表：

StudentID	Student
1	Amy
2	Jon
3	Beth
4	Karen
5	Alex

Teachers 表：

TeacherID	Teacher	Assistant
1	Smith	Collins
2	Jones	Brown
3	Kaplan	NULL
4	Harris	Taylor

Tests 表：

TestID	TeacherID	Test	Date	TotalPoints
1	1	Pronoun Quiz	2009-03-02	10
2	2	Pronoun Quiz	2009-03-02	10
3	3	Solids Quiz	2009-03-03	20
4	4	China Test	2009-03-04	50
5	1	Grammar Test	2009-03-05	100

Formats 表：

TestID	TestFormat
1	Multiple Choice
2	Multiple Choice
3	Multiple Choice
4	Essay
5	Multiple Choice
5	Essay

Grades 表：

StudentID	TestID	Grade
1	1	8
2	2	6
3	3	17
4	4	45
5	4	38
4	5	88

你的第一印象可能是，我们让情况变得更加复杂了，而不是让其得到改善。例如，Grades 表现在有大量的数字，在快速浏览时完全看不出这些数字的含义。

确实如此。然而，要记住 SQL 有能力把表轻松地关联起来，你也可以看到在新的设计中有更大的灵活性。我们不仅可以自由地把那些做任何特定分析所需要的表关联起来，而且现在可以更加轻易地为这些表添加新的列，而不会影响到任何其他内容。

我们的信息变得更加模块化。例如，如果我们想要获取每位学生的额外信息，诸如地址和电话，可以为 Student 表直接添加新的列。此外，当稍后我们想要修改学生的地址和电话时，也只会影响到该表中的一行。

19.3　数据库设计的艺术

最后，数据库设计远不止是简单地走完规范化的过程。数据库设计实际上是一种艺术而不仅是科学，它需要结合商业问题来提问和思考。

在 Grades 示例中，我们展示了一种可能的数据库设计，以描述如何规范化数据。实际上，这个数据库设计有多种可能性。这很大程度上取决于实际上如何访问和修改数据。可以通过提问大量的问题来确定设计是否像需要的一样灵活和有意义。例如：

- **有其他的表需要添加到数据库中吗？**一个明显的选择是 Subjects 表，以便你可以很轻松地按照科目选择测试，诸如英语或数学。如果这么做，需要把科目关联到考试或者出题的老师吗？
- **可以把多个科目的成绩加在一起吗？**或许英语和社会研究的老师要做一个联合课程，并且想要把两个科目的某次考试成绩相加。你如何考虑这种情况？
- **如果一个学生的成绩不及格，现在要参加第 2 年同样的考试，那该怎么办呢？**你如何来区分他这次的成绩和去年的成绩？
- **允许老师可能执行特殊的规则吗？**例如，在一个特殊的时间段内，删除最低的小测验成绩。
- **对数据有特殊的分析需求吗？**如果相同的科目有多名老师，你想要比较每位老师的学生的平均成绩，以确定老师没有给出不公正的虚高的成绩吗？

可能的问题列表是无穷尽的。但是，关键是，数据不是凭空存在的。需要把数据设计和现实世界的需求结合起来。需要把数据库设计成这样一种方式：允许必要的灵活性。然而，还有一种风险，即过度设计数据库以至于数据变得难以理解。一位热情的数据库管理员可能决定创建 20 个表以考虑到每一种可能的情况，那就太不明智了！数据库设计是一个平衡的话

题，要寻找足够灵活并且直观而易于让系统用户理解的设计。

19.4 规范化的替代方法

我们强调规范化是数据库设计中应该遵循的最重要的原则。然而，在某些情况下，切实可行的替代方法可能更加合理。

例如，在数据仓库系统和软件领域，许多专家提倡数据库采用星型模式（star schema）设计而不是规范化设计。在星型模式中，允许并且鼓励一定数量的冗余。强调的是所创建的数据结构能够更加直观地反映业务关系，并且允许通过专业的分析软件对数据进行快速的处理。

我们为星型模型设计给出一个概述，它的主要思想是创建一个核心的事实表，该表和任意数目的维度表相关联。事实表包含了本质上是可以相加的所有定量数字。在前面的示例中，Grade 列是这样一个数字，因为我们可以把成绩加起来得到一个有意义的总成绩。维度表包含所有实体信息，诸如科目、时间、老师和学生等，它们和中心事实表相关联。

此外，专门的分析软件允许数据库程序员从星型数据库中创建数据立方（cube）。这些数据立方扩展了分析能力，允许用户向下钻取预定义的分层，这些分层是以各种不同的维度来定义的。这样系统的一个用户能够从一名学生整个学期的成绩向下钻取，以得到这位学生在任意一个星期中的成绩。

Grades 示例用星型模型来设计数据库，如图 19.2 所示。

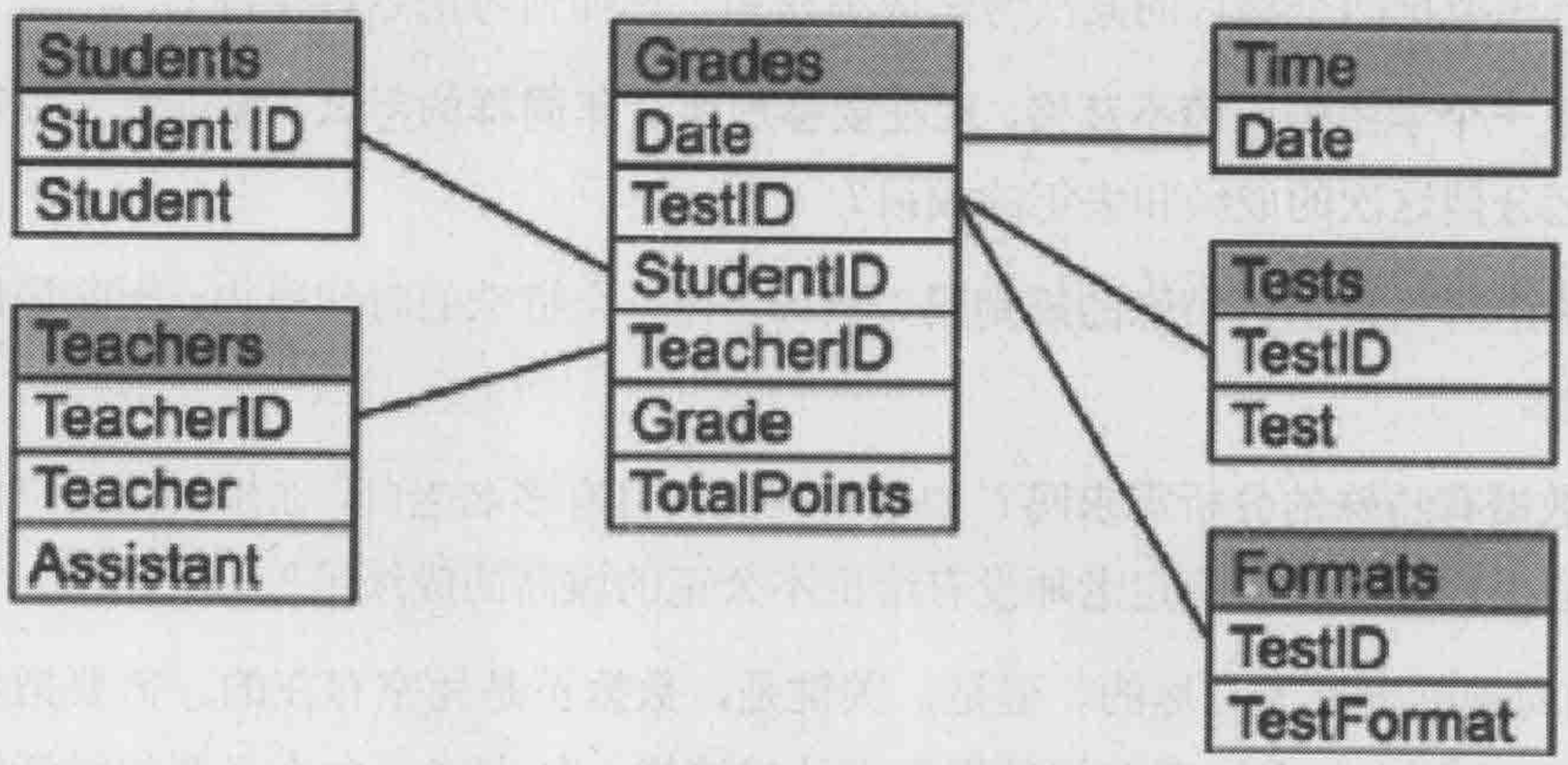


图 19.2　星型模型设计

在这种设计中，Grades 表是中心事实表，其他表是各种维度表。

Grades 表中的前 4 列（Date、TestID、StudentID 和 TeacherID）只和各个维度表相关联。另外两个列是我们讨论过的具备可加性的定量数值。请注意，现在 TotalPoints 在 Grades 表中。在我们的规范化设计中，它是 Tests 表的一个属性。把 Grade 和 TotalPoints 放在 Grades 表中，我们可以使用分析软件来轻松地为任意数据集合加和成绩并且计算平均得分率（用 Grade 除以 TotalPoints）。

当然，这只是为数据仓库进行数据库设计这一主题的一个简短介绍。它强调了有许多不同的方式来设计数据库，并且最好的方式通常跟和数据一起使用的软件的类型相关。

19.5　小结

本章介绍了数据库设计原理。我们回顾了规范化过程的基础知识，介绍了如何把只有一个表的数据库转化成一种更为灵活的结构，即带有多个表并且它们通过额外的键列相关。

我们还强调了数据库设计不仅仅是一个技术活，必须要注意组织的现实情况以及考虑数据如何使用。最后，我们简单地介绍了传统的规范化设计的另一种替代方法，主要是想要强调通常有多种设计方法。

在第 20 章中，我们将讨论使用报表软件工具补充 SQL 知识的一些相关可能性。在增强 SQL 技巧的同时，我们也不能忘记 SQL 以外的世界。当潜在目标可以通过其他手段更加有效地实现时，我们也要确保不会在 SQL 上花费过多精力。

第20章 显示数据的策略

在最后一章中，我们将回到本书的中心议题，也就是如何从关系数据库中检索数据。在之前的一些章节中，我们在更新数据、维护表和设计数据库等相关主题上花费了一点时间。

但是，现在我们想要再次把注意力集中到 SQL 在检索数据中扮演的角色。特别是，我们想要专注于通过报表软件向用户展现数据的情形。

20.1 超越 SQL

我们经常会使用专门的报表工具向用户展示数据。这种类型软件的示例包含 Microsoft Reporting Services 和 Crystal Reports。这些软件包允许程序员通过 SQL 连接到数据库，并且提供了设计良好的用户接口，允许用户很容易地通过预定义的报表访问数据。这些报表工具也能够让程序员轻松地以各种方式来展现数据。

此外，大部分报表工具允许用户把数据导出到诸如 Excel 的一个电子表格中。这就为用户提供了操作自己数据的机会，允许他们把数据转换成独特的电子表格的格式。

有鉴于此，本章的目的是，增强程序员这方面的意识：既可以通过报表软件来完成任务，也可以让用户在电子表格中操作数据来完成任务。无论哪种方式，都有机会降低其他工具或终端用户通常使用 SQL 的复杂度。数据库程序员会试图在 SQL 中完成所有的一切，相比之下，用户在最终组织的数据中扮演的角色，经常更容易也更好完成。

基本上，当通过报表工具或用户查看数据可以处理类似的功能时，我们将考虑避免 SQL 语句的过度复杂性。

20.2 报表工具和交叉报表

我们将使用 Microsoft Reporting Services 来举例说明报表工具能做什么。Microsoft Reporting Services 是众多可用的工具之一，这些工具还包括 Crystal Reports、Cognos 和 MicroStrategy 等。

特别的，我们会介绍 Microsoft 称之为 matrix 报表的一种报表类型。在 Microsoft 之外，通常把 matrix 报表称作交叉报表（crosstab reports）。

在 Microsoft Reporting Services 中，创建新报表的最简单方法是使用其报表向导（Report Wizard）。在调用向导后，会运行以下步骤：

1. 获得数据源和连接。

2. 通过 SELECT 语句或使用内建的 Query Builder，创建一个查询。

3. 指定一个报表类型，选项包括 tabular 和 matrix。

4. 通过 SELECT 语句或 Query Builder，把特定的列组织到不同的报表区域。

5. 为报表选择一种可视化风格。

步骤 3 允许我们指定报表类型。在 Microsoft 术语中，tabular 报表以常规形式展示数据。数据元素用列展示，每个事件作为新的一行显示。在报表中显示的数据就是在 SELECT 语句中或 Query Builder 中指定的数据。

matrix 报表是我们需要思考的新的功能。当使用 matrix 报表类型时，数据项不再像常规一样只放在列中。相反，可以把数据项放在报表的四个不同区域之一：行、列、明细和页。

描述 tabular 报表和 matrix 报表之间的区别的最好方式是，使用一个简单的示例。假设我们有一条 SELECT 语句如下所示：

```
SELECT
CustomerName AS 'Customer',
ProductCategory AS 'Product Category',
OrderAmount AS 'Amount'
FROM SalesTable
```

当用这条语句创建一个 tabular 报表时，报表如图 20.1 所示。

Customer	Product Category	Amount
Carter	Desks	150
Carter	Paper	5
Kraft	Chairs	120
Pollen	Chairs	40
Pollen	Chairs	200
Smith	Chairs	40
Smith	Desks	300

图 20.1　tabular 报表

这个 tabular 报表中的数据以列和行的常规组织形式来展现。

现在，如果对相同的 SELECT 语句使用 matrix 报表，我们会看到报表如下所示。在这个指定的 matrix 报表中，CustomerName 列可以放在行的区域，ProductCategory 列放在列的区域，OrderAmount 列放在在明细区域。图 20.2 显示了最终的 matrix 报表。

	Chairs	Desks	Paper
Carter	0	150	5
Kraft	120	0	0
Pollen	240	0	0
Smith	40	300	0

图 20.2　matrix 报表

matrix 报表以完全不同的方式展示数据。matrix 报表通过客户组和产品组汇总了数据，而不是展示单独的行。报表动态地判断在客户组和产品组中已有哪一个唯一的值，并且展示了必要的行和列以显示所有的值。

matrix 报表的明细区域要求带数量值的数据项,因为在 matrix 报表中会自动加和那些值。请注意，在 tabular 报表中，Pollen 有两笔椅子的订单，一笔是 40 元，另一笔是 200 元。在 matrix 报表中，这两个值自动加和，显示的是总计 240 元。

当然，这样或那样的报表工具有许多其他特性和功能。要记住的重要一点是，用报表工具完成任务可以减少程序员编写 SQL 语句的负担。

20.3　电子表格和透视表

除了报表工具以外，电子表格还提供了很多功能用于操作数据。大部分的报表工具允

许我们把数据导出到 Excel 电子表格中。然后，你就可以随心所欲地在 Excel 中格式化和分析数据。

许多基础的 Excel 功能和特性，与 SELECT 语句中所能指定的功能相重合。例如，Excel 允许很轻松地对数据排序。Excel 提供了许多内建函数，它们和 SQL 中内建的函数相似或等同。

Excel 的另一个重要特性是用 subtotals 函数分组数据的能力。这意味着，如果你既需要细节数据又需要按照组进行分类汇总，那么在 Excel 中可以轻松实现。SQL 程序员可能需要提供最底层的细节数据。在 Excel 中使用一些按键，就可以根据需要分组数据并添加分类汇总。

然而，在这点上，Excel 提供的最显著的特性是透视表（pivot table）。使用 Excel，我们能够选取电子表格上任意区域的数据，并且把这些数据转换成一个透视表。基本上，透视表相当于前面示例中看到的 matrix（交叉）报表。

然而，透视表也添加了一些关键元素，这大大提高了它的实用性：

- **首先，透视表是完全交互的**。与查看一个静态的交叉报表不同，用户通过把数据元素重新排列到行、列和数据等各个区域，就可以快速地更新透视表。此外，透视表提供了名为页（page）的第 4 个区域，它允许数据项用于过滤而不是用于显示。
- **透视表中的数据以单独的数据存储存在，甚至可以作为一个单独的文件来保存**。因为透视表中的数据独立存在，用户可以重组数据，而不会对创建透视表的数据产生影响。
- **透视表允许额外的数据选择**。例如，可以把不同的数据项的指定值排除在外，或者可以把透视表的数据区域中的聚合值从加和改为计数。
- **透视表可以添加对数据的向下钻取的功能**。例如，如果底层的数据包含像 country、state 和 city 一样的列，可以创建透视表以实现向下钻取的功能，双击 country 可以看到这个国家中的所有州，然后双击每个 state 可以看到该州下的所有城市。
- **透视表允许用户通过汇总数据钻取到底层的细节**。这样，你就可以在数据区域中双击任意单独的值，看到组成这个数字的单独的行。

对透视表细节的介绍超出了本书的范畴。但是，意识到可以用透视表完成任务，这对于 SQL 程序员来说非常有用。

图 20.3 显示了一个透视表的示例，创建它的数据来自于前面列出的相同的数据，但是添加了两个新的列：Date 和 Subcategory。Subcategory 列是 ProductCategory 的进一步分解。在

这个透视表中，Date、Product Category 和 Subcategory 都放在行的区域中，它描述了透视表如何以非常灵活的方式提供进一步的数据分解。

Sum of Amount			Customer			
Date	Product Category	Subcategory	Carter	Kraft	Pollen	Smith
12/5/2009	Chairs	Metal		120		40
		Plastic			40	
	Paper	Glossy	5			
12/6/2009	Desks	Wood	150			
12/7/2009	Chairs	Metal			200	
	Desks	Metal				300
Grand Total			155	120	240	340

图 20.3　透视表

关于这个透视表，最重要的一点是在行中有 3 种不同层级的组。首先，它按照 Date 划分所有的数据，然后按照 Product Category 划分数据，最后按照 SubCategory 划分数据。我们可以看到在 2009 年 12 月 5 日，销售了两笔不同的 Product Category：chairs 和 paper。对于 chairs，销售了两种不同的 SubCategory：metal 和 plastic。对于每个分类，透视表按照客户提供了销售的分解。

因此，透视表擅长于以多种方式来汇总数据。此外，用户几乎能以任意需要的形式来组织数据项。

20.4　小结

在本章中，我们介绍了可以用报表工具和电子表格以定制的格式来显示数据的一些方法。特别是，交叉报表添加了以某种方式汇总数据的能力，这很难严格地通过 SQL 语句来表示。Excel 中的透视表使用交叉报表的基本概念，并扩展它为用户提供更多的灵活性和功能。意识到报表和用户工具可以重新格式化数据，SQL 程序员就可以更高效地专注于检索数据，并且让报表工具或用户来处理较为复杂的显示问题。

现在，我们已经完成了本书的学习，为什么不试着自己执行一些命令？如果你还没有这样做，请阅读附录 A、附录 B 和附录 C，获取一些提示，以开始使用 Microsoft SQL Server、MySQL 或者 Oracle。附录介绍了如何安装这些数据库的免费版本，还提供了如何使用这些软件来执行 SQL 命令的一些基本信息。

最后，我真心希望本书能够成为你进入 SQL 世界的有用指南。在本书的开头，我曾说过，

SQL 包含了逻辑和语言。语言部分相当直白。在每一章，我都会重点介绍关键字以及它们背后的含义。但是，现在你已经完成了本书，我希望你能够领会到包含逻辑的 SQL 所拥有的力量。

这是单纯的逻辑，允许你把一串值组织到列和行中，并将其转换成接近信息的内容。使用 SQL 的挑战在于决定如何对现实世界的数据应用逻辑，这是理论和实践的交锋。在使用函数、聚合、连接、子查询、视图等技术时，你将处理真实的原始数据，并且学习如何使用一些曲折的逻辑来操纵它。

但是，逻辑不是问题的结束。SQL 语言扮演了一个重要的角色，我要说 SQL 的真正的美丽在于这门语言相当松散。它既不晦涩也不冗长。每一个关键字都有不同的目的，指定一些特定的逻辑，仅此而已。我不会说 SQL 已经到了像诗一般的境界，但是在计算机领域，SQL 语言绝对具有相当的美感。

附录 A

初识 Microsoft SQL Server

A.1 概览

Microsoft SQL Server 免费版本的安装过程如下。这个过程在操作系统为 Windows 7 的个人计算机上进行过测试。请注意，具体的操作可能和这里所展示的有所不同，这取决于个人电脑上已经安装了什么软件。

这些过程可能随着时间而改变，请参考网站 www.courseptr.com/Downloads 上的更新信息。

主要涉及两个步骤：

- 安装 SQL Server Express 2008；
- 安装 SQL Server Management Studio。

Microsoft SQL Server Express 2008 允许我们创建数据库。SQL Server Management Studio 是一个图形化界面，它允许执行 SQL 命令来与服务器以及你所创建的任意数据库进行交互。

上述两个软件都可以从 www.microsoft.com/sqlserver 站点下载。

A.2 安装 SQL Server Express 2008

安装 Microsoft SQL Server 数据库的步骤如下所示：

1. 浏览 www.microsoft.com/sqlserver。
2. 在最右端的 Downloads 区域，点击 SQL SERVER 2008 EXPRESS。
3. 点击 INSTALL 按钮。

4. 如果还没有安装 Microsoft Web Platform，将会弹出一个新的窗口，让你安装它。点击 GET THE MICROSOFT WEB PLATFORM 按钮。

5. 然后，可能会问你一系列的问题，诸如是否想要运行这个程序以及允许执行各种过程。对所有的问题都回答 RUN、YES 或 ALLOW。

6. 完成 Microsoft Web Platform 的安装之后，你会看到要求安装 SQL Server Express 2008 的界面，点击 INSTALL 按钮。

7. 在 License 界面，点击 ALLOW 按钮。

8. 在 Authentication Mode 界面，选择 Mixed Mode 认证并且输入 SQL Server Administrator (SA)用户的密码。

9. 安装完成之后，点击 FINISH 按钮，然后点击 EXIT 按钮。

A.3 安装 SQL Server Management Studio

安装 Microsoft SQL Server 2008 Express Management Tools 的步骤如下所示：

1. 浏览 www.microsoft.com/sqlserver。

2. 在 Downloads 区域，点击 SQL SERVER 2008 EXPRESS。

3. 在 Other Installation Options 下的表格上，点击 Management Tools 栏下边的 INSTALL 按钮。

4. 然后，可能会问你一系列的问题，问你是否允许执行各种过程，对所有的问题都回答 YES 或 ALLOW。

5. 你将看到一个询问安装 SQL Server 2008 Management Studio Express 的界面，点击 INSTALL 按钮。

6. 在 License 界面，点击 I ACCEPT 按钮，接下来会安装软件。在安装过程中，你可能会看到一个兼容性问题的警告。如果看到，点击 RUN PROGRAM 按钮。

7. 安装完成之后，点击 EXIT 按钮。

完成这些步骤之后，你将拥有如下已经安装好的软件：

- . Microsoft SQL Server Management Studio

- . Microsoft SQL Server Integration Services

A.4　使用 SQL Server Management Studio

当启动 SQL Server Management Studio 时，你将会看到一个窗口，它要求与已经安装过的 Microsoft SQL Server 建立连接。在 Server Name 框中输入“localhost\SQLEXPRESS”。在 Authentication 框中，选择 WINDOWS AUTHENTICATION。不需要输入用户名或密码，点击 CONNECT 按钮。

连接之后，你需要创建一个数据库。要做到这点，找到窗口左边的 Object Explorer。在 DATABASES 线上点击右键，然后选择 NEW DATABASE。在 NEW DATABASE 窗口，在 Database Name 框中输入一个名字（例如 FirstDatabase），点击 OK 按钮。现在，你可以看到 Databases 下的新的数据库。

要执行任何所需的 SQL 代码，可以高亮选中数据库，然后点击 NEW QUERY 按钮。将会打开一个新的查询窗口。你可以输入任意的 SQL 代码，然后点击 EXECUTE 按钮。如果你在查询窗口输入多条 SQL 语句，可以高亮选中任意数目的单独的语句，并且只执行高亮选中的部分。执行完查询之后，查询结果会显示在 Results 或 Message 面板中。如果有数据显示，结果会出现在 Results 面板中。否则，会在 Message 面板中显示一条状态信息。

Microsoft 在它们网站的 MSDN（Microsoft Development Network）部分提供了完整的 SQL Server 的在线文档。Transact-SQL 参考指南的位于：

http://msdn.microsoft.com/en-us/library/bb510741.aspx

附录 B
初识 MySQL

B.1 概览

MySQL 免费版本的安装过程如下。这个过程在操作系统为 Windows 7 的个人计算机上进行过测试。请注意，具体的操作可能和这里所展示的有所不同，这取决于个人电脑上已经安装了什么软件。

这些过程可能随着时间而改变，请参考网站 www.courseptr.com/Downloads 上的更新信息。

主要涉及两个步骤：

- 安装 MySQL Community Server；
- 安装 5.2 或更高版本的 MySQL Workbench。

MySQL Community Server 允许我们创建数据库。MySQL Workbench 是一个图形化界面，它允许执行 SQL 命令来与服务器和所创建的任意数据库进行交互。

MySQL Community Server 可以从 www.mysql.com 站点下载。在编写本书时，MySQL Workbench 下载的是一个过渡版本。为了使用这个工具来执行 SQL 语句，你需要获取 5.2 或以上版本。目前，这个版本的试用版可以从 dev.mysql.com 下载。

B.2 安装 MySQL Community Server

安装 MySQL Community Server 的步骤如下：

1. 浏览 www.mysql.com。

2. 点击 DOWNLOADS 标签页。

3. 点击 MySQL COMMUNITY SERVER。

4. 选择 MICROSOFT WINDOWS 平台。

5. 点击一个 DOWNLOAD 按钮。

6. 如果是老用户，请登录。否则，注册一个新用户。

7. 点击其中一个镜像文件旁边所显示的 HTTP。

8. 当询问运行还是保存文件时，选择 RNU 按钮。

9. 下载完成之后，当询问是否想要运行这个软件，选择 RUN 按钮。然后，将要启动安装向导。

10. 在 Welcome 界面，点击 NEXT 按钮。

11. 在 Setup Type 界面，选择 TYPICAL，然后点击 NEXT 按钮。

12. 在 Ready to Install 界面，点击 INSTALL 按钮。

13. 在 User Account Control 提示框上，点击 YES 按钮。然后安装软件。

14. 关闭 MySQL Enterprise 提示框。

15. 在 MySQL Server 5.1 Setup Wizard 框，点击 FINISH 按钮。

16. 在 User Account Control 提示框，点击 YES 按钮。

17. 在 Registration 界面上，如果需要的话，输入个人信息和注册信息。

18. 打开 MySQL Server Instance Configuration Wizard 提示框，你可能需要最小化浏览器才能看到它。

19. 在 Welcome 界面，点击 NEXT 按钮。

20. 在 Configuration Type 界面，选择 STANDARD CONFIGURATION，然后点击 NEXT 按钮。

21. 在 Windows Options 界面，勾选 INSTALL AS WINDOWS，并却确保 LAUNCH THE MYSQL SERVER AUTOMATICALLY 已经勾选，然后点击 NEXT 按钮。

22. 在 Security Option 界面，如果需要的话，输入一个密码，然后点击 NEXT 按钮。

23．在 Ready to Execute 界面，点击 EXECUTE 按钮。它将显示如下信息：“Configuration file created, Windows service MySQL installed, Service started successfully”。

24．点击 FINISH 按钮。

完成这些步骤之后，你将拥有如下已经安装好的软件：

- MySQL Command Line Client
- MySQL Server Instance Config Wizard

B.3　安装 MySQL Workbench

安装 MySQL Workbench 的步骤如下所示。请注意，为了用该工具执行 SQL，你需要 5.2 或以上版本的软件。在本书出版时，5.2 版本只有测试版可用。

1．浏览 dev.mysql.com。

2．点击 DOWNLOADS 标签页。

3．点击左边的 MySQL WORKBENCH。

4．点击 MySQL WORKBENCH，确保它是 5.2 版本或更高版本。

5．选择 MICROSOFT WINDOWS 平台。

6．点击一个 DOWNLOAD 按钮。

7．如果是老用户，请登录。否则，如果需要的话，注册一个新用户。

8．点击其中一个镜像文件旁边所显示的 HTTP。

9．当询问运行还是保存文件时，选择 RNU 按钮。然后，会启动安装向导。

10．在 Welcome 界面，点击 NEXT 按钮。

11．在 Setup Type 界面，选择 COMPLETE，然后点击 NEXT 按钮。

12．在 Ready to Install 界面，点击 INSTALL 按钮。

13．在 Completion 界面，点击 FINISH 按钮。

完成这些步骤之后，你将拥有如下已经安装好的软件：

- MySQL Workbench version 5.2

B.4 使用 MySQL Workbench

在完成初始安装后，当第一次打开 MySQL Workbench 时，你需要与已经安装过的 MySQL Server 实例建立一个连接。

要创建一个连接，选择 Database 菜单下的 MANAGE CONNECTIONS。点击 NEW 按钮来添加一个新的连接，然后输入一个连接名称（例如 MyConnection），并且点击 TEST CONNECTION 按钮。如果你之前设定过密码的话，输入一个密码，然后点击 CLOSE 按钮。

在创建了连接之后，你需要创建一个数据库。要做到这点，选择 Database 菜单下的 QUERY DATABASE。选择你刚才创建的连接，然后点击 OK 按钮。如果你之前设定过密码，输入该密码。现在，在屏幕上可以看到 SQL Editor，Object Explorer 区域出现在 SQL Editor 的左边，SQL Statements 区域出现在 SQL Editor 的右边。在 Object Explorer 中，用鼠标右键点击任意数据库，选择 CREATE SCHEMA。在 Name 文本框中输入新的数据库名称（例如 FirstDatabase）。点击 APPLY 按钮两次，然后点击 FINISH 按钮。现在，你可以看到 Object Explorer 中的新的数据库。最后，使用 Object Explorer 顶部默认的下拉框，选取新的数据库作为默认数据库。

现在，已经创建了一个连接和一个数据库，你可以在 SQL Editor 的 SQL Statements 区域中输入任意的 SQL 语句。如果你没有看到 SQL Editor，返回并按照之前所描述的选取 Database 菜单下的 Query Database。如前所述，这里会要求你选择一个连接。

在 SQL Statements 区域中输入一条 SQL 语句之后，点击 EXECUTE 按钮，它看上去像是一个闪电。如果在窗口中输入多条语句，你可以高亮选中一个单独的语句，只执行高亮选中的部分。

执行完查询之后，查询结果会显示在 Output 或 Result 面板中。如果有数据显示，结果会出现在 Result 面板中。否则，会在 Output 面板中显示一条状态信息。

MySQL 提供了完整的数据库文档。MySQL 参考手册的地址是：

http://dev.mysql.com/doc/refman/5.1/en

附录 C 初识 Oracle

C.1 概览

Oracle 免费版本的安装过程如下。这个过程在操作系统为 Windows 7 的个人计算机上进行过测试。请注意，具体的操作可能和这里所展示的有所不同，这取决于个人计算机上已经安装了什么软件。

这些过程可能随着时间而改变，请参考网站 www.courseptr.com 上的更新信息。

主要涉及一个步骤：即安装 Oracle Database Express Edition。

安装将创建一个单独的数据库，并且提供一个基于 Web 图形化的界面，它允许对数据库执行 SQL 命令。

软件可以从 www.oracle.com/database 下载。

作为安装过程的一部分，可能会要求你输入数据库用户名。你应该输入的名称是 SYSTEM。

C.2 安装 Oracle Database Express Edition

安装 Oracle Database Express Edition 数据库的步骤如下所示：

1. 浏览 www.oracle.com/database。
2. 点击页面左边的 EXPRESS EDITION。
3. 点击 FREE DOWNLOAD 按钮。

4. 点击 ORACLE DATABASE 10g EXPRESS EDITION FOR MICROSOFT WINDOWS。

5. 点击 ACCEPT LICENSE AGREEMENT。

6. 在 Oracle Database 10g Express Edition（Western European）下边，点击 ORACLEXE.EXE。

7. 然后，可能会问你一系列的问题，诸如你是否允许执行各种过程，对所有的问题都回答 RUN。

8. 如果你还没有 Oracle 账户，点击 SIGN UP NOW。否则，输入用户名和密码，然后输入个人信息。

9. 当询问运行还是保存文件时，选择 RNU 按钮。然后启动安装向导。

10. 在 Welcome 界面，点击 NEXT 按钮。

11. 在 License 界面，点击 ACCEPT 按钮，然后点击 NEXT 按钮。

12. 在 Destination 界面，点击 NEXT 按钮。

13. 在 Database Passwords 界面，输入一个密码。

14. 在 Summary 界面，点击 INSTALL。

15. 完成安装之后，点击 FINISH 按钮。

完成这些步骤之后，你将拥有如下已经安装好的软件：Oracle Database 10g Express Edition。

数据库的界面是基于 Web 的。在开始菜单下的 Oracle Database 10g Express 路径下，有可以访问的各种特性。在这个路径中，你需要使用的主要应用程序是 Go To Database Home Page。

在初始化安装之后，会显示 Database Home Page 页面上的 Database Login 界面。在开始菜单下的 Oracle Database 10g Express Edition 路径下选择 GO TO DATABASE HOME PAGE，可以进入相同的页面。

到了 Database Home Page 之后，输入用户名 SYSTEM 和你创建的密码，然后点击 LOGIN 按钮。

C.3 使用 Oracle Database Express Edition

要访问 Oracle 数据库，在开始菜单下的 Oracle Database 10g Express Edition 路径下运行

GO TO DATABASE HOME PAGE 程序。这会打开一个基于 Web 的应用程序，它允许你与数据库进行交互。

输入用户名 SYSTEM；以及在安装时指定的密码，然后点击 LOGIN 按钮登录。

然后，你将看到表示不同功能的 4 个图标：Administration、Object Browser、SQL，和 Utilities。要执行 SQL，点击 SQL 图标。然后，你将看到 3 个图标：SQL Commands、SQL Scripts 和 Query Builder。

如果你想要执行一条单独的 SQL 语句，可以使用 SQL Commands 图标。它会允许你执行一条单独的命令，并可以看到所有结果。如果输入多条 SQL 语句，你可以高亮选中一条单独的语句，并且只执行高亮选中的部分。要在 SQL Commands 窗口执行一条 SQL 语句，点击 RUN 按钮。

如果你想要执行多条 SQL 语句，但是不需要看到输出，可以使用 SQL Scripts 图标。选择这个图标之后，你既可以创建一个新的脚本，也可以编辑一个已有的脚本。要创建一个新的脚本，点击 CREATE 按钮，然后为这个脚本命名，输入你想要放在脚本中的语句。要想执行它，点击 RUN 按钮。在输入这个请求后，它会要求对执行做一个确认，再次点击 RUN 按钮，然后，通过点击 View Results 栏下的按钮，就能够看到脚本执行的结果汇总。

Oracle 为它们的数据库提供了完整的在线文档。其参考手册的地址是：

http://www.oracle.com/pls/db112

附录 D
所有 SQL 语句列表

本书中的所有 SQL 语句的列表可以在 Cengage 的网站 www.courseptr.com/downloads 找到。

这里提供了 3 个文件：

- SQL Statements and Data for SQL Server.doc
- SQL Statements and Data for MySQL.doc
- SQL Statements and Data for Oracle.doc

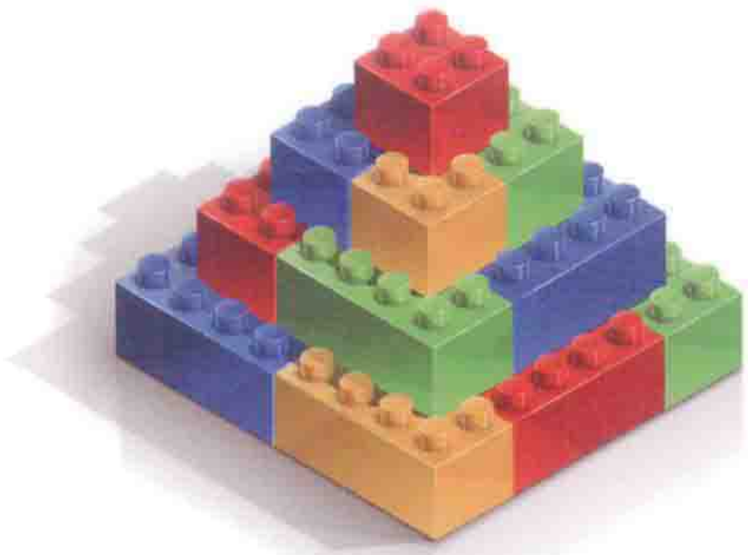
这 3 个文件列出了本书所介绍的 3 种数据库的所有 SQL 语句。此外，这些文件包含的 SQL 脚本，可以用来创建本书所用到的所有数据。当运行了安装脚本之后，你可以执行本书中的任何的语句，并且可以看到与本书中相同的输出结果。

在每个文件中，提供了执行安装脚本的具体的说明。

SQL

初学者指南

The Language of SQL



大部分的SQL书籍都试图成为SQL语法的百科全书，这是一种适得其反的方法，因为这些信息可以很容易地从主要的数据库供应商所发布的在线参考上获取。对于SQL入门图书，书籍更重要的是关注通用的概念，并提供清晰的解释以及各种语句所能够完成的任务的示例。

这是一本针对SQL初学者的图书。本书有许多特色，使它有别于其他的SQL书籍。首先，当你阅读本书时，无需下载软件或者使用计算机。本书的目标是，让你直接阅读本书就能够理解所提供的SQL示例。其次，以直观和逻辑的顺序来组织主题。一次只介绍一个SQL关键字，当你遇到新的单词或概念时，它是建立在你之前的理解的基础之上。最后，本书介绍了3种广泛使用的数据库的语法，它们是Microsoft SQL Server、MySQL和Oracle，专门的“数据库的差异”板块则展示了3种数据库语法的不同之处，还介绍了如何下载和安装这些数据库的免费版本。要获取所需的SQL和关系数据库知识，有了本书就够了。

通过本书，你将会学到：

- 如何使用SQL从数据库获取数据；
- 如何更新数据库中的数据；
- 如何构建和维护数据库；
- 获取数据后显示数据的策略。

本书配套网站www.courseptr.com/downloads提供了书中所有SQL语句和数据的下载。

作者简介

Larry Rockoff多年从事商业智能和数据仓库的开发。他重点研究的领域是使用报表工具在复杂数据库中探索、提取和分析数据。他从芝加哥大学获取了MBA的学位，他所学的专业是科学管理。

人民邮电出版社－信息技术分社
http://weibo.com/ptpitbooks

CENGAGE Learning®
www.cengageasia.com

CENGAGE Learning®
圣 智 学 习

ISBN 978-7-115-37122-5

ISBN 978-7-115-37122-5

定价:39.00 元

分类建议：计算机／程序设计／SQL
人民邮电出版社网址：www.ptpress.com.cn