TURING 图灵程序设计丛书 数据库系列

SAMS

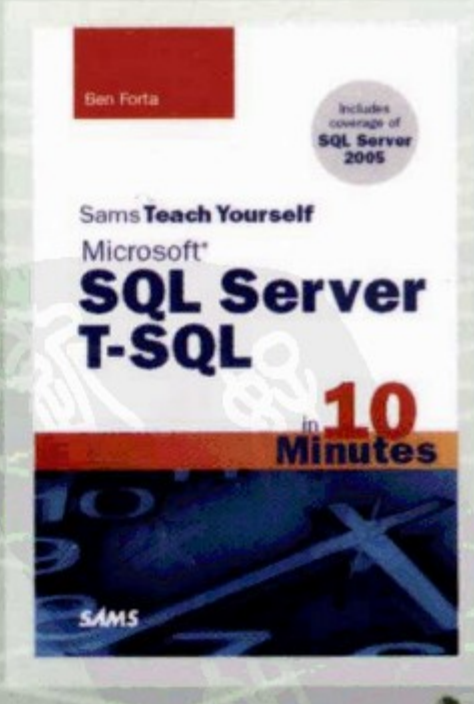
Sams Teach Yourself Microsoft SQL Server T-SQL in 10 Minutes

SQL Server 编程必知必会

[英] Ben Forta 著

刘晓霞 钟鸣 等译

- 《SQL必知必会》作者新作
- Amazon全五星评价
- T-SQL学习与使用必备图书



人民邮电出版社
POSTS & TELECOM PRESS

“盼望这本书已经很久了……虽然我是一个经验丰富的数据库管理员和数据库开发人员，但这本书仍使我受益匪浅。”

——Pinal Dave，微软SQL-MVP，SQLAuthority.com创办人

Sams Teach Yourself Microsoft SQL Server T-SQL in 10 Minutes

SQL Server编程必知必会

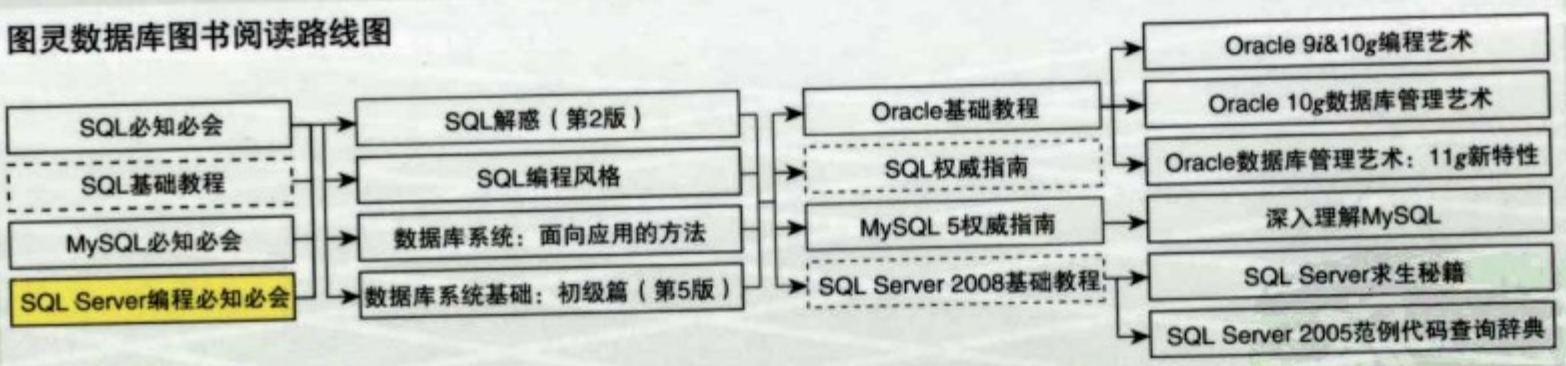
Microsoft SQL Server是世界上应用最广的数据库管理系统之一。

本书是作者继经典畅销书《SQL必知必会》之后，应众多读者的请求编写的，专门针对SQL Server T-SQL用户。书中继承了《SQL必知必会》的优点，在精练然而透彻地阐述了数据库基础理论之后，紧贴实战需要，很快转向数据检索，逐步深入各种复杂的内容，包括联结的使用、子查询、基于全文本的搜索、函数和存储过程、游标、触发器、表约束、XML，等等。对于每个知识点，都给出了实用的代码及其解析，并有丰富的技巧提示和常犯错误警示。通过本书，读者能够掌握扎实的基本功，迅速成为SQL Server高手。

作者为本书专门开设了网站http://www.forta.com/books/0672328674/，提供下载、勘误和答疑论坛。

Ben Forta 世界知名的技术作家，也是Adobe技术界最为知名的专家之一，目前担任Adobe公司的高级技术推广专家。他具有计算机行业20多年工作经验，多年来撰写了十几本技术图书，包括《正则表达式必知必会》、《SQL必知必会》（人民邮电出版社出版）等世界性的畅销书，已被翻译为十几种文字。读者可以通过他的个人网站http://www.forta.com了解更多信息。

图灵数据库图书阅读路线图



SAMS

本书相关信息请访问：图灵网站 http://www.turingbook.com

读者/作者热线：(010)88593802

反馈/投稿/推荐信箱：contact@turingbook.com

分类建议 计算机/数据库/SQL Server

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-19256-1

ISBN 978-7-115-19256-1/TP

定价：29.00 元

TURING 图灵程序设计丛书 数据库系列

Sams Teach Yourself Microsoft SQL Server T-SQL in 10 Minutes

SQL Server 编程必知必会

人民邮电出版社
北　京

图书在版编目（CIP）数据

SQL Server 编程必知必会 /（英）福塔（Forta, B.）著；刘晓霞等译 .—北京：人民邮电出版社，2009.1
（图灵程序设计丛书）
书名原文：Sams Teach Yourself Microsoft SQL Server T-SQL in 10 Minutes
ISBN 978-7-115-19256-1

Ⅰ. S… Ⅱ. ①福… ②刘… Ⅲ. 关系数据库－数据库管理系统 . SQL Server Ⅳ. TP311.138

中国版本图书馆CIP数据核字（2008）第184144号

内 容 提 要

SQL Server 是世界上最受欢迎的数据库管理系统之一。本书从介绍简单的数据检索开始，全面讲述了 SQL Server 的使用，包括连接的使用、子查询、基于全文本的搜索、函数和存储过程、游标、触发器、表约束、XML 等。通过重点突出的章节，条理清晰、系统而扼要地讲述了读者应该掌握的知识，使他们不经意间“功力大增”。

本书注重实用性，操作性很强，适合于广大软件开发和数据库管理人员学习参考。

图灵程序设计丛书

SQL Server 编程必知必会

◆ 著 [英] Ben Forta
译 刘晓霞 钟 鸣 等
责任编辑 傅志红
执行编辑 刘 静

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 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：850×1168 1/32
印张：8.375
字数：258 千字 2009 年 1 月第 1 版
印数：1－4 000 册 2009 年 1 月北京第 1 次印刷

著作权合同登记号 图字：01-2008-4292 号
ISBN 978-7-115-19256-1/TP

定价：29.00 元

读者服务热线：(010)88593802 印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

Microsoft SQL Server已经成为世界上最受欢迎的数据库管理系统之一。无论是用在小型开发项目上，还是用来构建那些声名显赫的网站，SQL Server都证明了自己是个稳定、可靠、快速、可信的系统，足以胜任任何数据存储业务的需要。

本书基于我的一本畅销书*Sams Teach Yourself SQL in 10 Minutes*（中文版《SQL必知必会》，人民邮电出版社出版），那本书堪称全世界用得最多的一本SQL教程，重点讲解读者必须知道的东西，条理清晰，系统而扼要。但是，即使是那样一本广为使用的成功的书，也还存在着以下这些局限性。

- 由于要面向所有主要的数据库管理系统（DBMS），我不得不把针对具体DBMS的内容一再压缩。
- 为了简化SQL的讲解，我必须（尽可能）只写各种主要的DBMS通用的SQL语句。这要求我不得不舍弃一些更好的、针对具体DBMS的解决方案。
- 虽然基本的SQL在不同的DBMS间具有较好的可移植性，但是高级的SQL显然不是这样的。因此，那本书里无法详细讲解比较高级的内容，如触发器、游标、存储过程、访问控制、事务等。

于是就有了这本书。本书沿用了前一本书业已成功的教程模式和组织结构，除了Transact-SQL（简写为T-SQL）以外，不在其他内容上过度纠缠。书从简单的数据检索开始，逐步进入一些复杂的内容，包括连接

的使用、子查询、基于全文本的搜索、函数和存储过程、游标、触发器、表约束、XML，等等。通过重点突出的章节，条理清晰、系统而扼要地让读者学到应该学到的知识，使他们不经意间立刻功力大增。

本书针对SQL Server 2005版本写成 本书写作以SQL Server 2005为目标，包含了该版本新增的特性和技术。除两章之外，其他内容都适用于SQL Server的几个老版本，包括SQL Server 2000。

请回到第1章开始学习。读者会立刻体会到SQL Server提供的所有好处。

读者对象

本书的读者对象是这样一些人：

- 他没有学过SQL；
- 他刚开始用SQL Server，并希望一举成功；
- 他想迅速地、尽可能多地学会使用SQL Server和T-SQL；
- 他希望学习怎样在自己的应用程序开发中使用T-SQL；
- 他希望通过使用SQL Server轻松快速地提高工作效率，而不用劳烦他人帮忙。

配套网站

本书有一个配套网站，网址是：http://forta.com/books/0672328674/。

读者可以通过该网站访问如下内容：

- 表格创建和表格填充的脚本，可用来创建书中使用的样例表；
- 在线支持论坛；
- 在线勘误（如果发现了勘误的话）；
- 或许他会感兴趣的其他书。

本书约定

本书使用不同的字体区分代码和一般正文内容，对于重要的概念也

采用特殊的字体。

键入的文本和屏幕上显示出的文本用等宽代码字体表示。如：`It looks like this to mimic the way text looks on your screen.`

一行代码最前面如果出现箭头（➥）表示该行代码较长，书中一行放不下。读者录入时需要把这一行的内容紧接着上一行输入。

说明：表示跟上下文的内容相关的一些有意思的信息。

提示：提供建议，教读者用容易的办法完成某项任务。

注意：向读者提示可能出现的问题，帮你避免不必要的麻烦。

新术语，提供新的基本词汇的清晰定义。

输入　表示读者自己键入的代码。通常出现在程序清单的旁边。

输出　表示出运行T-SQL代码后得到的结果，通常出现在程序清单之后。

分析　告诉读者这是作者对输入或输出的逐行分析。

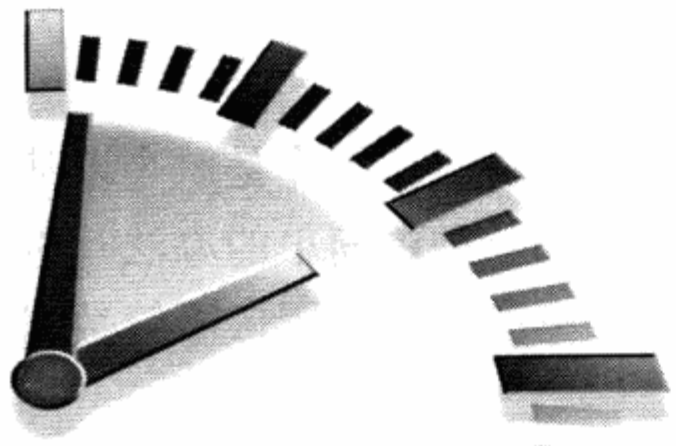
致　谢

首先，我要感谢Sams出版公司的伙伴们，他们再一次给了我灵活的自由度，让我把书写成我认为合适的样子。谢谢Mark Renfrow又一次提供深入且极有价值的反馈意见。特别感谢Loretta Yates、Damon Jordan和Mark Taber，他们勇敢地介入到整个出版过程中，使书稿经过多次改动和推迟仍然能回归正轨，继续进行。

谢谢Jon Price，他是我迄今为止有幸合作过的最认真的技术编辑之一。

最后，本书以及我的另一本书（中文版《MySQL必知必会》，人民邮电出版社出版）是应《SQL必知必会》读者的请求编写的。《SQL必知必会》收到了很多极有价值的反馈意见和建议，在此我深表谢意。本书中可以看到这些建议的成果。谢谢大家，我希望自己没有辜负大家的期望。

目　录



第1章 了解SQL

本章将介绍数据库和SQL，它们是学习T-SQL的先决条件。

1.1 数据库基础

你正在阅读本书，这表明，你需要以某种方式与数据库打交道。因此，在深入学习SQL Server及其SQL语言的T-SQL实现以前，应该对数据库及数据库技术的某些基本概念有所了解。

你可能还没有意识到，其实你自己一直在使用数据库。每当你从自己的电子邮件地址簿里查找名字时，你就在使用数据库。如果你在某个因特网搜索站点上进行搜索，也是在使用数据库。如果你在工作中登录网络，也需要依靠数据库验证自己的名字和密码。即使是在自动取款机上使用ATM卡，也要利用数据库进行PIN码验证和余额检查。

虽然我们一直都在使用数据库，但对究竟什么是数据库并不十分清楚。特别是不同的人可能会使用相同的数据库术语表示不同的事物，更加剧了这种混乱。因此，我们学习的良好切入点就是给出一张最重要的数据库术语清单，并加以说明。

基本概念回顾 下面是某些基本数据库概念的简要介绍。如果你已经具有一定的数据库经验，这可以用于复习巩固；如果你是一个数据库新手，这将给你提供一些必需的基本知识。理解数据库是掌握SQL Server和T-SQL的一个重要部分，如果有必要的话，你应该参阅一些有关数据库基础知识的书籍[①]。

① 推荐人民邮电出版社出版的由Kifer、Bernstain和Lewis合著的《数据库系统：面向应用的方法》。——编者注

1.1.1　什么是数据库

数据库这个术语的用法很多，但就本书而言，数据库是一个以某种有组织的方式存储的数据集合。理解数据库的一种最简单的办法是将其想象为一个文件柜。此文件柜是一个存放数据的物理位置，不管数据是什么以及如何组织的。

数据库（database）　保存有组织的数据的容器（通常是一个文件或一组文件）。

误用导致混淆　人们通常用数据库这个术语来代表他们使用的数据库软件。这是不正确的，它是引起混淆的根源。确切地说，数据库软件应称为数据库管理系统（DBMS）。数据库是通过DBMS创建和操纵的容器。数据库可以是保存在硬盘上的文件，但也可以不是。在大多数情况下，数据库究竟是文件还是别的什么东西并不重要，因为你并不直接访问数据库；你使用的是DBMS，它替你访问数据库。

1.1.2　表

在你将资料放入自己的文件柜时，并不是随便将它们扔进某个抽屉就完事了，而是在文件柜中创建文件，然后将相关的资料放入特定的文件中。

在数据库领域中，这种文件称为表。表是一种结构化的文件，可用来存储某种特定类型的数据。表可以保存顾客清单、产品目录，或者其他信息清单。

表（table）　某种特定类型数据的结构化清单。

这里关键的一点在于，存储在表中的数据应是一种类型的数据或一个清单。决不应该将顾客的清单与订单的清单存储在同一个数据库表中。这样做以后的检索和访问会很困难。应该创建两个表，每个清单一个表。

数据库中的每个表都有一个名字，来标识自己。此名字是唯一的，这表示数据库中没有其他表具有相同的名字。

表名 表名的唯一性由几个因素决定，包括数据库名和表名的结合。这表示，虽然在相同数据库中不能两次使用相同的表名，但在不同的数据库中却可以使用相同的表名。

表具有一些特性，这些特性定义了数据在表中如何存储，如可以存储什么样的数据，数据如何分解，各部分信息如何命名，等等。描述表的这组信息就是所谓的模式，模式可以用来描述数据库中特定的表，也可以用来描述整个数据库（和其中表的关系）。

模式（schema） 关于数据库和表的布局及特性的信息。

7

1.1.3 列和数据类型

表由列组成。列中存储着表中某部分的信息。

列（column） 表中的一个字段。所有表都是由一个或多个列组成的。

理解列的最好办法是将数据库表想象为一个网格。网格中每一列存储着一条特定的信息。例如，在顾客表中，一个列存储着顾客编号，另一个列存储着顾客名，而地址、城市、州以及邮政编码全都存储在各自的列中。

分解数据 正确地将数据分解为多个列极为重要。例如，城市、州、邮政编码应该总是独立的列。通过把它分解开，才有可能利用特定的列对数据进行排序和过滤（如，找出特定州或特定城市的所有顾客）。如果城市和州组合在一个列中，则按州进行排序或过滤会很困难。

数据库中每个列都有相应的数据类型。数据类型定义列可以存储的数据种类。例如，如果列中存储的为数字（或许是订单中的物品数），则相应的数据类型应该为数值类型。如果列中存储的是日期、文本、注释、

金额等，则应该用恰当的数据类型规定出来。

8

数据类型（datatype） 所容许的数据的类型。每个表列都有相应的数据类型，它限制（或容许）该列中存储的数据。

数据类型限制可存储在列中的数据种类（例如，防止在数值字段中录入字符值）。数据类型还帮助正确地排序数据，并在优化磁盘使用方面起重要的作用。因此，在创建表时必须特别注意选取正确的数据类型。

1.1.4　行

表中的数据是按行存储的，所保存的每个记录存储在自己的行内。如果将表想象为网格，网格中垂直的列为表列，水平行为表行。

例如，顾客表可以每行存储一个顾客。表中的行数即为记录的总数。

行（row） 表中的一个记录。

是记录还是行？ 你可能听到用户在提到行（row）时称其为数据库记录（record）。在多数情况下，这两个术语是可以互相替代的，但从技术上说，行才是正确的术语。

1.1.5　主键

表中每一行都应该有可以唯一标识自己的一列（或一组列）。一个顾客表可以使用顾客编号列，而订单表可以使用订单ID，雇员表可以使用

9

雇员ID或雇员社会保险号。

主键（primary key）[①] 一列（或一组列），其值能够唯一区分表中每个行。

唯一标识表中每行的这个列（或这组列）称为主键。主键用来表示一个特定的行。如果没有主键，更新或删除表中特定行就很困难，因为没有安全的方法保证只涉及相关的行。

① 全国科学技术名词审定委员会审定的key在数据库中的对应名词为“键码”或“码”，本书采用了已约定俗成的“键”，请读者注意。——编者注

应该总是定义主键 虽然并不总是都需要主键，但大多数数据库设计人员都应保证他们创建的每个表具有一个主键，以便于以后的数据操纵和管理。

表中的任何列都可以作为主键，只要它满足以下条件：

- 任意两行都不具有相同的主键值；
- 每个行都必须具有一个主键值（主键列不允许NULL值）。

主键值规则 这里列出的规则是由SQL Server本身强制实施的。

主键通常定义在表的一列上，但这并不是必需的，也可以一起使用多个列作为主键。在使用多列作为主键时，上述条件必须应用到构成主键的所有列，所有列值的组合必须是唯一的（但单个列的值可以不唯一）。

主键的最好习惯 除SQL Server强制实施的规则外，还应该坚持以下几个普遍认可的最好习惯：

- 不更新主键列中的值；
- 不重用主键列的值；
- 不在主键列中使用可能会更改的值。（例如，如果使用供应商名字作为主键，当该供应商发生并购或改名时，主键也得跟着变。）

还有一种非常重要的键，称为外键，我们将在第14章中介绍。

1.2 什么是SQL

SQL（发音为字母S-Q-L或sequel）是结构化查询语言（Structured Query Language）的缩写。SQL是一种专门用来与数据库通信的语言。

与其他语言（如英语以及Java和Visual Basic这样的程序设计语言）不一样，SQL只有很少的单词，这是有意而为的。设计SQL的目的是

很好地完成一项任务，即提供一种从数据库中读写数据的简单有效的方法。

SQL有如下的优点。

- SQL不是某个特定数据库供应商专有的语言。几乎所有重要的DBMS都支持SQL，所以，学会SQL就几乎能与所有数据库打交道。
- SQL简单易学。它的语句全都是由描述性很强的英语单词组成，而且这些单词的数目不多。
- SQL尽管看上去很简单，但它实际上是一种强有力的语言，灵活使用其语言元素，可以进行非常复杂和高级的数据库操作。

11

DBMS专用的SQL　虽然SQL不是一种专利语言，而且有一个标准委员会在定义可供所有DBMS使用的SQL语法，但事实是任意两个DBMS实现的SQL都不完全相同。本书讲授的SQL是T-SQL（Transact-SQL），是专门针对Microsoft SQL Server的，虽然书中讲授的多数语法也适用于其他DBMS，但这些SQL语法并不是完全可移植的。

1.3　动手实践

本书所有章节都采用可上机运行的例子来说明SQL语法，它的功能是什么，为什么起这样的作用。作者强烈建议读者试验每个例子，以便掌握T-SQL的第一手资料。

附录B描述了本书中使用的样例表，说明如何获得和安装它们。如果你还没有安装它们，请在继续学习前先看这个附录。

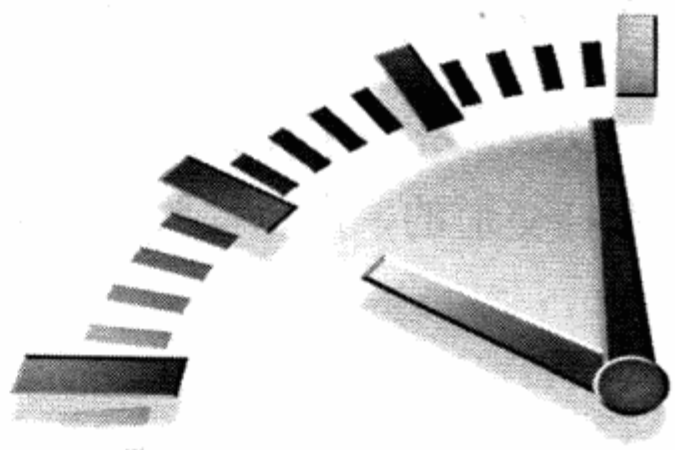
你需要SQL Server　显然，你需要有一份SQL Server才能学习本书的内容。附录A说明了如何获得SQL Server，并提供一定的入门指导。如果你还没有SQL Server，在继续学习之前，也请参阅该附录。

1.4 小结

这一章介绍了什么是SQL以及它为什么很有用。因为SQL是用来与数据库打交道的，所以，我们也复习了一些基本的数据库术语。 12

第2章

SQL Server介绍



本章将介绍什么是SQL Server，以及在SQL Server中可以应用什么工具。

2.1 什么是SQL Server

我们在前一章中介绍了数据库和SQL。正如所述，数据的所有存储、检索、管理和处理实际上是由数据库软件——DBMS（数据库管理系统）完成的。SQL Server是一种DBMS，即，它是一种数据库软件。

SQL Server已经存在很久了，在全世界被广泛安装和使用。为什么有那么多的公司和开发人员使用SQL Server？以下是一些主要原因：

- 性能——SQL Server执行很快（非常快）；
- 可信赖——某些非常重要和声望很高的公司、站点使用SQL Server，这些公司和站点都用SQL Server来处理自己的重要数据；
- 集成——SQL Server与Microsoft 的其他软件紧密集成在一起；
- 简单——SQL Server是最容易安装和使用的DBMS之一，其管理工具使服务器管理轻松又简单。

那么，为什么有些人不使用SQL Server呢？首先，SQL Server只能在Windows上运行，如果你的服务器运行的是其他操作系统（如Linux），显然你不能使用SQL Server。此外，SQL Server是一个商业产品，对于那些对不花钱的开源产品感兴趣的人来说，其他DBMS可能更有吸引力。最后，有人批评SQL Server不像某些其他DBMS那样支持高端企业特性（如集群和容错），虽然这些在SQL Server 2005中已经大为改观。

2.1.1 客户机-服务器软件

DBMS可分为两类：一类为基于共享文件系统的DBMS，另一类为基于客户机-服务器的DBMS。前者（包括诸如Microsoft Access和FileMaker）用于桌面用途，通常不用于高端或更关键的应用。

SQL Server、Oracle以及MySQL等数据库是基于客户机-服务器的数据库。客户机-服务器应用分为两个不同的部分。服务器部分是负责所有数据访问和处理的一个软件，它运行在称为数据库服务器的计算机上。

与数据文件打交道的只有服务器软件。关于数据、数据增删和数据更新的所有请求都由服务器软件完成。这些请求或更改来自运行客户机软件的计算机。客户机是与用户打交道的软件。例如，如果你请求一个按字母顺序列出的产品表，则客户机软件通过网络提交该请求给服务器软件。服务器软件处理这个请求，根据需要过滤、丢弃和排序数据，然后把结果送回到你的客户机软件。

有多少计算机？ 客户机和服务器软件可能安装在两台计算机或一台计算机上。不管它们在不在相同的计算机上，为进行所有数据库交互，客户机软件都要与服务器软件进行通信。

所有这些活动对用户都是透明而不可见的。数据存储在别的地方，或者数据库服务器为你完成这个处理，这些事实是隐藏的。你不需要直接访问数据文件。事实上，多数网络使用户不具有对数据的访问权，甚至不具有对存储数据的驱动器的访问权。

14

这样做意义何在？因为为了使用SQL Server，你需要访问运行SQL Server服务器软件的计算机和发布命令到SQL Server的客户机软件：

- 服务器软件为SQL Server DBMS，你可以在本地安装的副本上运行，也可以连接到你具有访问权的远程服务器上的一个副本；
- 客户机可以是SQL Server提供的工具、脚本语言（如Perl）、Web应用开发语言（如ASP、ASP.NET、ColdFusion、JSP和PHP）、程序设计语言（如VB、VB.NET、C、C++、C#和Java）等。

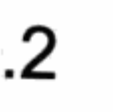
2.1.2　SQL Server版本

客户机软件稍后介绍。我们先简要介绍DBMS版本。

到本书出版时，SQL Server的最新版本为SQL Server 2005[①]（虽然许多公司还在使用SQL Server 2000及之前版本）。本书是为SQL Server 2005而写的，当然也适用于SQL Server 2000（毕竟大部分内容还是适用之前版本的）。

使用SQL Server 2005版本　如果可能，推荐使用SQL Server 2005。它不仅适用于本书各章内容（包括专门针对SQL Server 2005中引入的新特性的两章，而且还可以使用一个技术上非常先进的产品，一个起大多数先进客户机工具作用的产品）。

15



版本要求说明　如果某章要求特殊的SQL Server版本，则将在该章开始处明确说明。

2.2　SQL Server工具

如前所述，SQL Server是一个客户机-服务器DBMS，因此，为了使用SQL Server，需要有一个客户机（你需要用它来与SQL Server打交道），给SQL Server提供要执行的命令。

有许多客户机应用可供选择，但在学习SQL Server（编写和测试SQL Server脚本）时，最好是使用简单脚本执行的实用程序。具体哪种工具最理想取决于使用的SQL Server的版本。

2.2.1　SQL Server 2005

SQL Server 2005提供了一个名为Microsoft SQL Server Management Studio的高级客户机工具。这个工具可用来创建和管理数据库及表，控制数据库的访问和安全，运行优化与调整DBMS性能的向导，当然，也执行SQL语句。

---

① 目前SQL Server的最新版本为SQL Server 2008。——编者注

本地或远程 Microsoft SQL Server Management Studio可用来连接本地或远程的DBMS。只要DBMS配置为允许你连接它，你就可以连接到任何地点的数据库。

使用Microsoft SQL Server Management Studio有许多方式，以下是录入和测试SQL语句所需的基本步骤。

- 单击屏幕左上角的New Query按钮打开一个窗口，在其中录入SQL语句。
- 在输入T-SQL语句后，Microsoft SQL Server Management Studio自动做标记，标示出语句和文本（这是一个非常有价值的排错工具，因为它让你能快速找出打字错误，或者找出错误的引号等）。
- 要执行（运行）一条语句，可单击Execute按钮（其上有红色感叹号），也可以按F5或Ctrl+E执行一条语句。
- 为检验一条SQL语句在语法上是否正确（不执行它），单击Parse按钮（其上有蓝色的对勾）。
- Microsoft SQL Server Management Studio在屏幕底部显示语句结果。这些结果可以作为纯文本显示在网格中（默认行为），或者保存为一个文件。你可以单击适当的工具栏按钮在这些方式之间切换。
- 除显示语句结果外，Microsoft SQL Server Management Studio还在标记为Messages的一个辅助标签中显示状态信息（如，返回行的数目）。
- 为获得帮助，单击你需要帮助的语句并按F1。

Microsoft SQL Server Management Studio还可以用来执行保存的脚本（保存在文件中的SQL语句，如附录B中提到的样例表创建和填充脚本）。事实上，本书中使用的所有输出例子都是从Microsoft SQL Server Management Studio中抓取的（采用纯文本输出）。

其他工具 SQL Server 2005还安装有另外一整套工具和实用程序，不过，它们超出了本书的范围。

2.2.2 SQL Server 2000

SQL Server 2000提供一个名为SQL Query Analyzer的很便于使用的客户机工具。此工具可用来录入和执行SQL语句。SQL Query Analyzer可直接运行，或者从另一名为SQL Enterprise Manager的工具（用来创建和管理数据库及表，控制数据库的访问和安全等）中运行。下面是使用SQL Query Analyzer录入和测试SQL语句所需的基本步骤。

17

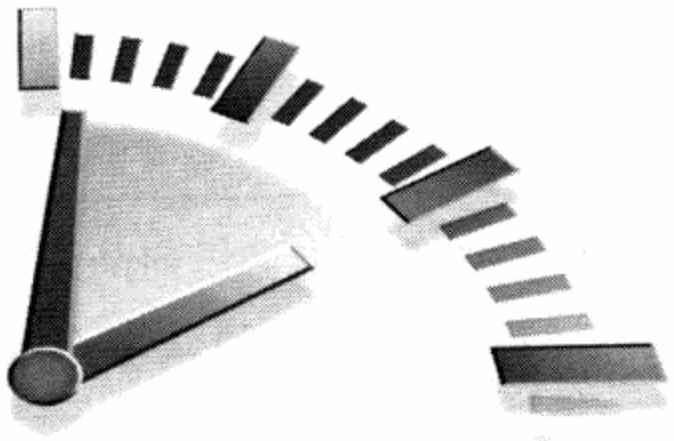
- 单击New Query按钮（工具栏上的最左边）打开一个窗口，在其中输入SQL语句。你也可以按Ctrl+N打开一个新查询窗。
- 在输入T-SQL语句后，SQL Query Analyzer自动做标记，选择语句和文本（这是一个非常有价值的排错工具，因为它让你能快速找出打字错误，或者找出错误的引号等）。
- 要执行（运行）一条语句，可单击Execute按钮（其上有绿色箭头），也可以按F5或Ctrl+E。
- 为检验一条SQL语句在语法上是否正确（不执行它），单击Parse按钮（其上有蓝色的对勾）。
- SQL Query Analyzer在屏幕底部显示语句结果。数据库检索结果显示在网格标签中，消息（如返回的行数）显示在Messages标签中。

虽然没有SQL Server 2005的Management Studio那样复杂，但SQL Query Profiler用来试验和学习T-SQL很理想。

2.3 小结

本章学习了什么是SQL Server，介绍了两个客户机实用程序（分别为SQL Server 2005的和SQL Server 2000的客户机实用程序）。

18



第3章 使用SQL Server

本章将学习如何连接和登录到SQL Server，如何执行SQL Server语句，以及如何获得数据库和表的信息。

3.1 连接

在具有可供使用的SQL Server DBMS和客户机软件之后，有必要简要讨论一下如何连接到数据库。

SQL Server与所有客户机-服务器DBMS一样，要求在能执行命令之前登录到DBMS。SQL Server可利用它自己的用户列表或利用Windows用户列表（用来开始使用Windows的登录）验证用户和登录。因此，如果对SQL Server配置得当，它可以利用你登录Windows的信息自动登录，或者提示你输入登录名和口令。

在最初安装SQL Server时，很可能会提示你输入一个管理登录（系统管理员通常为sa）和一个口令。如果你使用的是自己的本地服务器，并且只是简单地试验一下SQL Server，使用上述登录就可以了。但现实中，管理登录受到密切保护，因为对它的访问授予了创建表、删除整个数据库、更改登录和口令等完全的权限）。

为了连接到SQL Server，需要以下信息：

- 主机名（计算机名）——如果连接到本地SQL Server服务器，为localhost；
- 一个合法的用户名（如果没有使用Windows验证）；
- 用户口令（如果需要）。

如果你正在使用前一章介绍的某种客户应用，会显示一个对话框提示你这些信息。

使用其他客户机　如果你使用的客户机不是这里提到的客户机，则为了连接到SQL Server，仍然需要提供上述信息。

在连接之后，你具有你的登录名所对应的访问权。（登录、访问控制和安全可参阅第29章。）

3.2　选择数据库

一开始连接到SQL Server时，会有一个默认的数据库打开供你使用。通常，这个数据库名为master（一般说来，你不能使用它）。在你执行任意数据库操作前，需要先选择一个数据库。为此，可使用USE关键字。

关键字（keyword）　作为T-SQL语言组成部分的保留字。决不要用关键字命名一个表或列。附录E列出了SQL Server的关键字。

例如，为了使用crashcourse数据库，应该（在一个查询窗口中）输入以下内容：

输入

```
USE crashcourse;
```

输出

```
Command(s) completed successfully.
```

分析　USE语句并不返回任何结果。根据所用的客户机，它显示某种形式的通知（如上所示）。

交互式数据库选择　在SQL Server Management Studio（或SQL Query Analyzer）中，可从工具栏的下拉列表内选择某个要使用的数据库。你看不到USE命令的实际执行（虽然它已经被执行），但数据库将会改变，窗口标题栏将会反映这个改变。

记住，必须先使用USE打开数据库，才能读取其中的数据。

3.3 了解数据库和表

如果你不知道可以使用的数据库名时怎么办？这时，客户机应用如何获得显示在下拉列表中可用的数据库列表？

数据库、表、列、用户、权限等的信息被存储在数据库和表中（SQL Server使用SQL Server来存储这些信息）。这些内部表都位于master数据库中（这就是不能使用master数据库的原因），它们一般不直接访问。 21
SQL Server包括一系列的预先写入的存储过程，可用这些存储过程来显示这些信息（SQL Server从内部表中提取这些信息）。

存储过程 存储过程将在第23章中介绍。现在，视存储过程为保存在SQL Server中可以随时执行的SQL语句就可以了。

请看下面的例子：

输入

```
sp_databases;
```

输出

```
DATABASE_NAME      DATABASE_SIZE REMARKS
---------------- ------------- -------
coldfusion         9096          NULL
crashcourse        3072          NULL
forta              2048          NULL
master             4608          NULL
model              1728          NULL
msdb               5824          NULL
tempdb             8704          NULL
```

分析 sp_databases;返回可用数据库的一个列表。包含在这个列表中的可能是SQL Server内部使用的数据库（如例子中的master和tempdb）。当然，你自己的数据库列表可能看上去与这里的不一样。

为了获得一个数据库内的表的列表，使用sp_tables;，如下所示：

输入

```
sp_tables;
```
22

分析 sp_tables;返回当前选择的数据库内可用表的列表，不仅包括你自己的表，还包括各种系统表和其他条目（可能有几百个

条目)。

为了获得一个表的列表(只是表，不包括视图，也不包括系统表等)，可以使用下面语句：

输入

```
sp_tables NULL, dbo, crashcourse, "'TABLE'";
```

输出

```
TABLE_QUALIFIER TABLE_OWNER TABLE_NAME   TABLE_TYPE REMARKS
--------------- ----------- ------------ ---------- -------
crashcourse     dbo         customers    TABLE      NULL
crashcourse     dbo         orderitems   TABLE      NULL
crashcourse     dbo         orders       TABLE      NULL
crashcourse     dbo         products     TABLE      NULL
crashcourse     dbo         vendors      TABLE      NULL
crashcourse     dbo         productnotes TABLE      NULL
crashcourse     dbo         sysdiagrams  TABLE      NULL
```

分析　其中，sp_tables接受几个参数，告诉它使用哪个数据库，列出什么内容('TABLE'而不是'VIEW'或'SYSTEM TABLE')。

sp_columns可用来显示表列：

输入

```
sp_columns customers;
```

省略输出　sp_columns返回许多数据。在下面的输出中，我截短了显示，因为完全的输出会超出本书页面，每行输出可能需要占据多行的位置。

输出

```
TABLE_QUALIFIER TABLE_OWNER TABLE_NAME COLUMN_NAME  DATA_TYPE TYPE_NAME
crashcourse     dbo         customers  cust_id      4         int identity
crashcourse     dbo         customers  cust_name    -8        nchar
crashcourse     dbo         customers  cust_address -8        nchar
crashcourse     dbo         customers  cust_city    -8        nchar
crashcourse     dbo         customers  cust_state   -8        nchar
crashcourse     dbo         customers  cust_zip     -8        nchar
crashcourse     dbo         customers  cust_country -8        nchar
crashcourse     dbo         customers  cust_contact -8        nchar
crashcourse     dbo         customers  cust_email   -8        nchar
```

分析　sp_columns要求指定一个表名(这个例子中的customers)，对每个字段返回一行，行中包括字段名、数据类型、是否允许

NULL、键信息、默认值等。

什么是标识？ 列cust_id是一个标识列。某些表列需要唯一值（如，订单号、员工ID或刚才例子中的顾客ID）。在给表增加一行时，SQL Server能够自动分配下一个可用的编号，不需要手动分配唯一值（也不需要记住最后使用的是什么值）。这个功能就是所谓的标识。如果需要，它必须是用CREATE语句创建表时使用的表定义的组成部分。我们在第20章中介绍CREATE语句。

还支持大量的存储过程，包括以下几个。

- sp_server_info：用来显示广泛的服务器状态信息。
- sp_spaceused：用来显示数据库使用（和未使用）的空间量。
- sp_statistics：用来显示与数据库表有关的使用统计数据。
- sp_helpuser：用来显示可用的用户账号。
- sp_helplogins：用来显示用户登录及它们具有的权限。

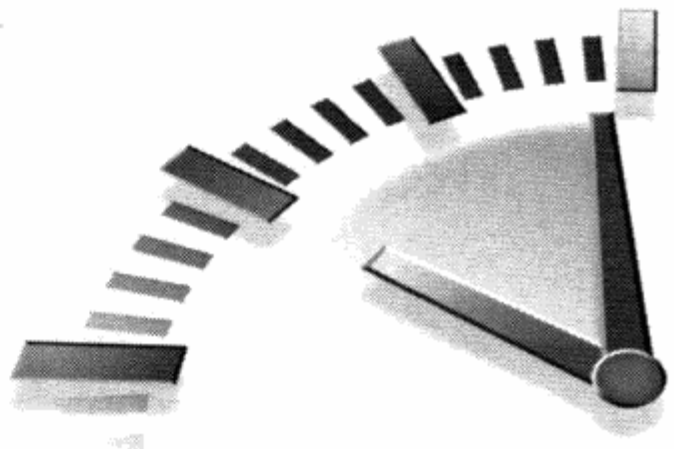
值得注意的是，客户机应用程序使用与这里相同的存储过程。显示数据库和表的交互式列表、允许交互式创建和编辑表、便于数据录入和编辑或允许管理用户账号和权限等的应用，全都使用你可以直接执行的相同的存储过程完成它们的工作。

3.4 小结

本章介绍了如何连接和登录SQL Server，如何用USE选择数据库，如何用存储过程查看SQL数据库、表和内部信息。有了这些知识，我们可以进一步深入学习所有重要的SELECT语句了。

第4章

检索数据



本章将介绍如何使用SELECT语句从表中检索一个或多个数据列。

4.1 SELECT语句

正如第1章所述，SQL语句是由简单的英语单词构成的。这些单词称为关键字，每个SQL语句都是由一个或多个关键字构成的。最常用的SQL语句大概就是SELECT语句了，它的用途是从一个或多个表中检索信息。

为了使用SELECT检索表数据，必须至少给出两条信息：想选择什么，以及从什么地方选择。

确保使用正确的数据库 为了理解和试验这些样例，要确保使用正确的数据库，即你在其中创建和填充了样例表的那个数据库。

4.2 检索单个列

我们将从简单的SQL SELECT语句开始介绍，此语句如下所示：

输入

```
SELECT prod_name
FROM products;
```

分析 上述语句利用SELECT语句从products表中检索一个名为prod_name的列。所需的列名在SELECT关键字之后给出，FROM关键字指出从其中检索数据的表名。此语句的输出如下所示：

输出

```
prod_name
--------------
.5 ton anvil
1 ton anvil
2 ton anvil
Detonator
Bird seed
Carrots
Fuses
JetPack 1000
JetPack 2000
Oil can
Safe
Sling
TNT (1 stick)
TNT (5 sticks)
```

未排序数据 如果读者自己试验这个查询，可能会发现显示输出的数据顺序与这里的不同。出现这种情况很正常。如果没有明确排序查询结果（下一章介绍），则返回的数据不会按重要性排顺序。返回数据的顺序可能是数据被添加到表中的顺序，也可能不是。只要返回相同数目的行，就是正常的。

如上的一条简单SELECT语句将返回表中所有行。数据没有过滤（过滤将得出结果集的一个子集），也没有排序。后几章将讨论这些内容。 28

结束SQL语句 多条SQL语句必须以分号(；)分隔。SQL Server（如多数DBMS）不需要在单条SQL语句后加分号。当然，如果愿意可以总是加上分号。事实上，即使不一定需要，但加上分号肯定没有坏处。

SQL语句和大小写 请注意，SQL语句不区分大小写，因此SELECT与select是相同的。同样，写成Select也没有关系。许多SQL开发人员喜欢对所有SQL关键字使用大写，而对所有列和表名使用小写，这样做使代码更易于阅读和调试。最佳方式是选定一种大小写的规定，在使用时要保持一致。

使用空白　在处理SQL语句时，其中所有空白都被忽略。SQL语句可以在一行上给出，也可以分成许多行。多数SQL开发人员认为，将SQL语句分成多行更容易阅读和调试。

4.3　检索多个列

要想从一个表中检索多个列，可以使用相同的SELECT语句。唯一的不同是必须在SELECT关键字后给出多个列名，列名之间必须以逗号分隔。

当心逗号　在选择多个列时，一定要在列名之间加上逗号，但最后一个列名后不加。如果在最后一个列名后加了逗号，将出现错误。

下面的SELECT语句从products表中检索3列：

输入

```
SELECT prod_id, prod_name, prod_price
FROM products;
```

分析　与前一个例子一样，这条语句使用SELECT语句从表products中检索数据。在这个例子中，指定了3个列名，列名之间用逗号分隔。此语句的输出如下：

输出

```
prod_id    prod_name          prod_price
---------- ------------------ ----------
ANV01      .5 ton anvil       5.99
ANV02      1 ton anvil        9.99
ANV03      2 ton anvil        14.99
DTNTR      Detonator          13.00
FB         Bird seed          10.00
FC         Carrots            2.50
FU1        Fuses              3.42
JP1000     JetPack 1000       35.00
JP2000     JetPack 2000       55.00
OL1        Oil can            8.99
SAFE       Safe               50.00
SLING      Sling              4.49
TNT1       TNT (1 stick)      2.50
TNT2       TNT (5 sticks)     10.00
```

数据表示 SQL语句一般返回原始的、无格式的数据。数据的格式化是表示问题，而不是检索问题。因此，具体的表示一般在显示该数据的应用程序中规定，如对齐和显示上面的价格值时带货币符号和逗点。一般很少像上面的输出那样显示实际检索出的原始数据（没有应用程序提供的格式）。

30

4.4 检索所有列

除了指定所需的列外（如上所述，一个或多个列），SELECT语句还可以检索所有的列而不必逐个列出它们。这可以通过在实际列名的位置使用星号（*）通配符来达到，如下所示：

输入

```
SELECT *
FROM products;
```

分析

如果给出通配符（*），则返回表中所有列。列的顺序一般是列在表定义中出现的物理顺序。但是，并不都是这个顺序，因为表模式的变化（例如，添加和删除列）会导致列的顺序被改变。

使用通配符 除非你确实需要表中的每个列，否则一般最好不要使用*通配符。虽然使用通配符可能会使你省时又省力，不用明确列出所需列，但检索不需要的列通常会降低检索和应用程序的性能。

检索未知列 使用通配符有一个大优点。由于不明确指定列名（因为星号检索每个列），所以能检索出名字未知的列。

31

4.5 检索不同的行

可以看到，SELECT语句返回所有匹配的行。但是，若你希望每个值只出现一次，该如何做呢？例如，你希望检索products表中产品的所有供应商的ID。

输入

```
SELECT vend_id
FROM products;
```

输出

```
vend_id
-----------
1001
1001
1001
1003
1003
1003
1002
1005
1005
1002
1003
1003
1003
1003
```

这个SELECT语句返回14行（即使在products表中只有4个供应商），因为在该表中有14种产品。那么，该如何检索不同的值呢？

解决方法是使用DISTINCT关键字，正如其关键字名所示，它指示SQL Server仅返回不同的值。

32

输入

```
SELECT DISTINCT vend_id
FROM products;
```

分析 SELECT DISTINCT vend_id告诉SQL Server仅返回不同（唯一）的vend_id行，因此，只有4行数据返回，如下面的输出所示。如果使用DISTINCT关键字，那么它必须直接放在列名前。

输出

```
vend_id
-----------
1001
1002
1003
1005
```

DISTINCT不只对部分列　DISTINCT关键字可以应用于所有的列，而不仅仅针对紧跟其后的一列。如果输入为SELECT DISTINCT vend_id, prod_price，除非指定的列不同，否则返回所有的行。

4.6 限制结果

SELECT语句返回所有匹配的行，可能是指定表中的每一行。为了只返回前一行或几行，可以使用TOP关键字。例如：

输入

```
SELECT TOP(5) prod_name
FROM products;
```

分析 前面的语句使用SELECT语句来检索单个列。TOP(5)指示SQL Server最多返回5行。这个语句的输出如下：

33

输出

```
prod_name
------------
.5 ton anvil
1 ton anvil
2 ton anvil
Detonator
Bird seed
```

SQL Server 6.5及更新版本 对TOP语句的支持是在SQL Server 6.5中引入T-SQL的。如果你还在使用SQL Server更早的版本，那么可以使用SET ROWCOUNT语句，如下所示：

```
SET ROWCOUNT 5;
SELECTprod_name
FROM products;
```

在SQL Server的新版本中仍然支持SET ROWCOUNT语句，但一般最好使用TOP语句。

通过添加PERCENT关键字，可以使用TOP来获得行的百分比。例如：

输入

```
SELECT TOP(25) PERCENT prod_name
FROM products;
```

分析 TOP(25) PERCENT指示SQL Server返回products表中前25%的行。这个语句的输出如下：

输出

```
prod_name
-------------
.5 ton anvil
1 ton anvil
2 ton anvil
Detonator
```

34

没有足够多的行　TOP语句中指定的要检索的行数是所检索的最大行数。如果没有足够多的行（例如，指定的是20行，而仅有14行），那么SQL Server会尽其所能地返回更多的行。

SQL Server 2005支持使用TABLESAMPLE关键字检索任意的行，例如：

输入

```
SELECT * FROM products
TABLESAMPLE (3 ROWS);
```

分析　TABLESAMPLE用于指定检索的行数，这个例子中为检索任意的3行。

输入

```
SELECT * FROM products
TABLESAMPLE (50 PERCENT);
```

分析　TABLESAMPLE用于指定检索的行数的百分比，这个例子中为检索50%的行。

值得注意的是，你无法获得你预期的确切的行数。采样由表分页产生（SQL Server使用的内部机制实际上是存储数据），且一页上的行数可以变化。

35

4.7　使用完全限定表名

目前使用的SQL示例都是通过列名来指向列的，也可以使用完全限定名（同时使用表名和列名）来指向列。例如：

输入

```
SELECT products.prod_name
FROM products;
```

这个语句与本章最前面的例子的功能一致，但这里指定了完全限定列名。

表名也可以是完全限定的，例如：

输入

```
SELECT products.prod_name
FROM crashcourse.dbo.products;
```

这个语句的功能还是与上面的例子一致（当然，假设products表位于crashcourse数据库中）。要注意表名中的dbo。完全限定表名由数据库名、表所有者名和表名组成，默认的所有者一般都是dbo（即database

owner)，因此，crashcourse.dbo.products是crashcourse数据库中products表的完全限定名。

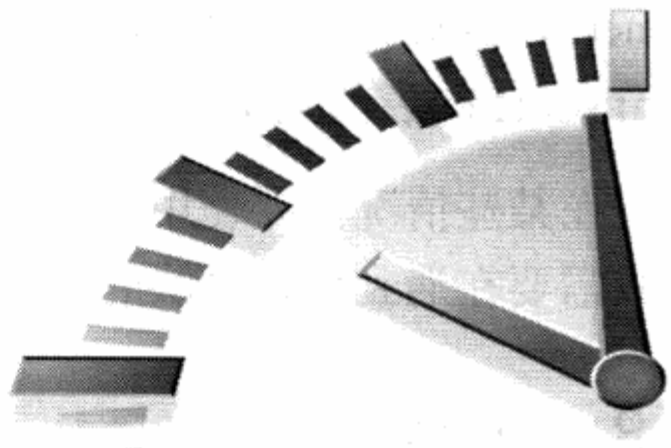
在某些情况需要使用完全限定名，这在后面章节中会看到。现在，你只需要在遇到这种语法的时候知道是怎么回事就可以了。

4.8 小结

本章学习了如何使用SQL的SELECT语句来检索单个表列、多个表列以及所有表列。下一章将讲授如何排序检索出来的数据。

第5章

排序检索数据



本章将讲授如何使用SELECT语句的ORDER BY子句，根据需要排序检索出的数据。

5.1 排序数据

正如前一章所述，下面的SQL语句返回某个数据库表的单个列。但请看其输出，数据并没有特定的显示顺序。

输入

```
SELECT prod_name
FROM products;
```

输出

```
prod_name
--------------
.5 ton anvil
1 ton anvil
2 ton anvil
Detonator
Bird seed
Carrots
Fuses
JetPack 1000
JetPack 2000
Oil can
Safe
Sling
TNT (1 stick)
TNT (5 sticks)
```


其实，检索出的数据并不是以纯粹的随机顺序显示的。如果不排序，数据一般将以它在底层表中出现的顺序显示。这可以是数据最初添加到表中的顺序。但是，如果数据后来进行过更新或删除，则此顺序将会受

到SQL Server重用回收存储空间的影响。因此，如果不明确控制的话，不能（也不应该）依赖该排序顺序。关系数据库设计理论认为，如果不明确规定排序顺序，则不应该假定检索出的数据的顺序有意义。

子句（clause） SQL语句由子句构成，有些子句是必需的，而有的是可选的。一个子句通常由一个关键字加上所提供的数据组成。子句的例子有SELECT语句的FROM子句，我们在前一章看到过这个子句。

为了明确地排序用SELECT语句检索出的数据，可使用ORDER BY子句。ORDER BY子句取一个或多个列的名字，据此对输出进行排序。请看下面的例子：

输入

```
SELECT prod_name
FROM products
ORDER BY prod_name;
```

分析 这条语句除了指示SQL Server对prod_name列以字母顺序排序数据的ORDER BY子句外，与前面的语句相同。结果如下：

输出

```
prod_name
--------------
.5 ton anvil
1 ton anvil
2 ton anvil
Bird seed
Carrots
Detonator
Fuses
JetPack 1000
JetPack 2000
Oil can
Safe
Sling
TNT (1 stick)
TNT (5 sticks)
```

38

通过非选择列进行排序 通常，ORDER BY子句中使用的列将是为显示所选择的列。但是，实际上并不一定要这样，用非检索的列排序数据是完全合法的。

5.2　按多个列排序

经常需要按不止一个列进行数据排序。例如，如果要显示雇员清单，可能希望按姓和名排序（首先按姓排序，然后在每个姓中再按名排序）。如果多个雇员具有相同的姓，这样做很有用。

为了按多个列排序，只要指定列名，列名之间用逗号分开即可（就像选择多个列时所做的那样）。

下面的代码检索3个列，并按其中两个列对结果进行排序——首先按价格，然后再按名称排序。

输入

```
SELECT prod_id, prod_price, prod_name
FROM products
ORDER BY prod_price, prod_name;
```

输出

```
prod_id    prod_price           prod_name
---------- -------------------- -------------
FC         2.50                 Carrots
TNT1       2.50                 TNT (1 stick)
FU1        3.42                 Fuses
SLING      4.49                 Sling
ANV01      5.99                 .5 ton anvil
OL1        8.99                 Oil can
ANV02      9.99                 1 ton anvil
FB         10.00                Bird seed
TNT2       10.00                TNT (5 sticks)
DTNTR      13.00                Detonator
ANV03      14.99                2 ton anvil
JP1000     35.00                JetPack 1000
SAFE       50.00                Safe
JP2000     55.00                JetPack 2000
```

重要的是理解在按多个列排序时，排序完全按所规定的顺序进行。换句话说，对于上述例子中的输出，仅在多个行具有相同的prod_price值时才对产品按prod_name进行排序。如果prod_price列中所有的值都是唯一的，则不会按prod_name排序。

5.3　指定排序方向

数据排序不限于升序排序（从A到Z）。这只是默认的排序顺序，还可以使用ORDER BY子句以降序（从Z到A）顺序排序。为了进行降序排序，

必须指定DESC关键字。

下面的例子按价格以降序排序产品（最贵的排在最前面）：

输入

```
SELECT prod_id, prod_price, prod_name
FROM products
ORDER BY prod_price DESC;
```

输出

```
prod_id    prod_price            prod_name
---------- --------------------- -------------
JP2000     55.00                 JetPack 2000
SAFE       50.00                 Safe
JP1000     35.00                 JetPack 1000
ANV03      14.99                 2 ton anvil
DTNTR      13.00                 Detonator
TNT2       10.00                 TNT (5 sticks)
FB         10.00                 Bird seed
ANV02      9.99                  1 ton anvil
OL1        8.99                  Oil can
ANV01      5.99                  .5 ton anvil
SLING      4.49                  Sling
FU1        3.42                  Fuses
FC         2.50                  Carrots
TNT1       2.50                  TNT (1 stick)
```

但是，如果打算用多个列排序怎么办？这又如何影响排序方向呢？下面的例子以降序排序产品（最贵的在最前面），然后再对产品名排序：

输入

```
SELECT prod_id, prod_price, prod_name
FROM products
ORDER BY prod_price DESC, prod_name;
```

输出

```
prod_id    prod_price            prod_name
---------- --------------------- -------------
JP2000     55.00                 JetPack 2000
SAFE       50.00                 Safe
JP1000     35.00                 JetPack 1000
ANV03      14.99                 2 ton anvil
DTNTR      13.00                 Detonator
FB         10.00                 Bird seed
TNT2       10.00                 TNT (5 sticks)
ANV02      9.99                  1 ton anvil
OL1        8.99                  Oil can
ANV01      5.99                  .5 ton anvil
SLING      4.49                  Sling
FU1        3.42                  Fuses
FC         2.50                  Carrots
TNT1       2.50                  TNT (1 stick)
```

分析 DESC关键字只应用到直接位于其前面的列名。在上例中，只对prod_price列指定DESC，对prod_name列不指定。因此，prod_price列以降序排序，而prod_name列（在每个价格内）仍然按标准的升序排序。

在多个列上降序排序　如果想在多个列上进行降序排序，必须对每个列指定DESC关键字。

与DESC相反的关键字是ASC（即ascending），在升序排序时可以指定它。但实际上，ASC没有多大用处，因为升序是默认的（如果既不指定ASC也不指定DESC，则假定为ASC）。

区分大小写和排序顺序　在对文本性的数据进行排序时，A与a相同吗？a位于B之前还是位于Z之后？这些问题不是理论问题，其答案依赖于数据库如何设置。

在字典（dictionary）排序顺序中，A被视为与a相同，这是SQL Server（大多数DBMS）的默认行为。但是，许多数据库管理员在需要时可以改变这种行为（如果你的数据库包含大量外语字符，可能必须这样做）。

这里的关键问题是，如果确实需要改变这种排序顺序，用简单的ORDER BY子句做不到。你必须请求数据库管理员的帮助。

使用ORDER BY和TOP的组合，可能会找到一列中的最大值和最小值。下面的例子表明了如何找到最贵产品的价格。

42

输入

```
SELECT TOP(1) prod_price
FROM products
ORDER BY prod_price DESC;
```

输出

```
prod_price
----------------------
55.00
```

分析 prod_price DESC确保了行是按照价格由最贵到最便宜检索的，TOP(1)指示SQL Server仅返回一行。

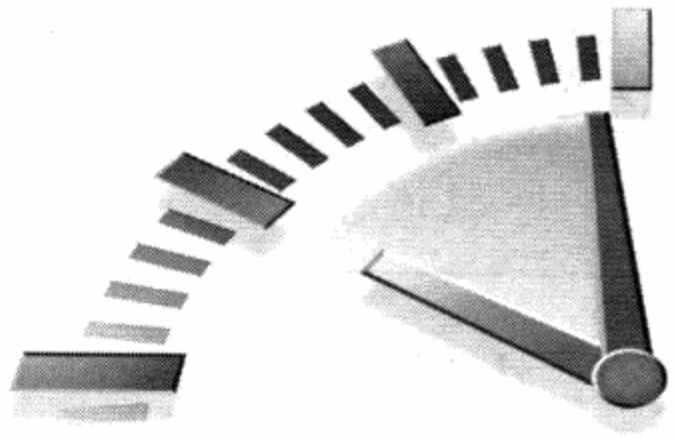
ORDER BY子句的位置 在指定一条ORDER BY子句时，应保证它位于FROM子句之后。该子句的使用次序不对将会出现错误消息。

5.4 小结

本章学习了如何用SELECT语句的ORDER BY子句对检索出的数据进行排序。这个子句必须是SELECT语句中的最后一条子句。可根据需要，利用它在一个或多个列上对数据进行排序。

第6章

过滤数据



本章将讲授如何使用SELECT语句的WHERE子句指定搜索条件。

6.1 使用WHERE子句

数据库表一般包含大量的数据，很少需要检索表中所有行。通常只会根据特定操作或报告的需要提取表数据的子集。只检索所需数据需要指定搜索条件（search criteria），搜索条件也称为过滤条件（filter condition）。

在SELECT语句中，数据根据WHERE子句中指定的搜索条件进行过滤。WHERE子句在表名（FROM子句）之后给出，如下所示：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price = 2.50;
```

分析 这条语句从products表中检索两个列，但不返回所有行，只返回prod_price值为2.50的行，如下所示：

输出

```
prod_name      prod_price
-------------- ----------
Carrots        2.50
TNT (1 stick)  2.50
```

这个例子采用了简单的相等测试：它检查一个列是否具有指定的值，据此进行过滤。但是，T-SQL允许做的事情不仅仅是相等测试。

SQL过滤与应用过滤 数据也可以在应用层过滤。为此，SQL的SELECT语句为客户机应用检索出超过实际所需的数据，然后客户机代码对返回数据进行循环，以提取出需要的行。

通常，这种实现并不令人满意。因此，对DBMS进行了优化，以便快速有效地对数据进行过滤。让客户机应用（或开发语言）处理数据库的工作极大地影响应用的性能，并且使所创建的应用完全不具备可伸缩性。此外，如果在客户机上过滤数据，服务器不得不通过网络发送多余的数据，这将导致网络带宽的浪费。

WHERE子句的位置 在同时使用ORDER BY和WHERE子句时，应该让ORDER BY位于WHERE之后，否则将会产生错误（关于ORDER BY的使用，请参阅第5章）。

6.2 WHERE子句操作符

我们在关于相等的测试时看到了第一个WHERE子句，它确定一个列是否包含特定的值。T-SQL支持很多比较操作符，表6-1列出了其中的一部分。 46

表 6-1 WHERE 子句操作符

操 作 符	说 明
=	等于
<>	不等于
!=	不等于
<	小于
<=	小于等于
!<	不小于
>	大于
>=	大于等于
!>	不大于
BETWEEN	在指定的两个值之间
IS NULL	为 NULL 值

6.2.1 检查单个值

我们已经看到了测试相等的例子。下面是另一个例子：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE prod_name = 'fuses';
```

输出

```
prod_name   prod_price
----------- ----------
Fuses       3.42
```

分析　WHERE prod_name = 'fuses'将返回值为Fuses的一行。默认情况下，T-SQL在执行匹配时不区分大小写，因此，fuses与Fuses相匹配。

47

现在来看看几个使用其他操作符的例子。第一个例子是列出价格小于10美元的所有产品：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price < 10;
```

输出

```
prod_name      prod_price
-------------- ----------
.5 ton anvil   5.99
1 ton anvil    9.99
Carrots        2.50
Fuses          3.42
Oil can        8.99
Sling          4.49
TNT (1 stick)  2.50
```

下一条语句检索价格小于等于10美元的所有产品（有2个额外的匹配）：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price <= 10;
```

输出

```
prod_name      prod_price
-------------- ----------
.5 ton anvil   5.99
1 ton anvil    9.99
Bird seed      10.00
Carrots        2.50
Fuses          3.42
Oil can        8.99
Sling          4.49
TNT (1 stick)  2.50
TNT (5 sticks) 10.00
```

48

6.2.2 不匹配检查

以下例子列出不是由供应商1003制造的所有产品：

输入

```
SELECT vend_id, prod_name
FROM products
WHERE vend_id <> 1003;
```

输出

```
vend_id     prod_name
----------- ------------
1001        .5 ton anvil
1001        1 ton anvil
1001        2 ton anvil
1002        Fuses
1005        JetPack 1000
1005        JetPack 2000
1002        Oil can
```

何时使用引号 如果仔细观察上述WHERE子句中使用的条件，会看到有的值括在单引号内（如'fuses'），而有的值未括起来。单引号用来限定字符串。如果将值与串数据类型的列进行比较，则需要限定引号。用来与数值列进行比较的值不用引号。

下面是相同的例子，其中使用!=而不是<>操作符：

输入

```
SELECT vend_id, prod_name
FROM products
WHERE vend_id != 1003;
```


6.2.3 范围值检查

为了检查某个范围的值，可使用BETWEEN操作符。其语法与其他WHERE子句的操作符稍有不同，因为它需要两个值，即范围的开始值和结束值。例如，BETWEEN操作符可用来检索价格在5美元和10美元之间，或日期在指定的开始日期和结束日期之间的所有产品。

下面的例子说明如何使用BETWEEN操作符，它检索价格在5美元和10美元之间的所有产品：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price BETWEEN 5 AND 10;
```

输出

```
prod_name          prod_price
---------------- ----------
.5 ton anvil       5.99
1 ton anvil        9.99
Bird seed          10.00
Oil can            8.99
TNT (5 sticks)     10.00
```

分析 从这个例子中可以看到，在使用BETWEEN时，必须指定两个值——所需范围的低端值和高端值。这两个值必须用AND关键字分隔。BETWEEN匹配范围中所有的值，包括指定的开始值和结束值。

50

6.2.4　空值检查

在创建表时，表设计人员可以指定其中的列是否可以不包含值。在一个列不包含值时，称其为包含空值NULL。

NULL　无值（no value），它与字段包含0、空字符串或仅仅包含空格不同。

SELECT语句有一个特殊的WHERE子句，可用来检查具有NULL值的列。这个WHERE子句就是IS NULL子句。其语法如下：

输入

```
SELECT prod_name
FROM products
WHERE prod_price IS NULL;
```

这条语句返回没有价格（空prod_price字段，不是价格为0）的所有产品，由于表中没有这样的行，所以没有返回数据。但是，customers表确实包含有具有NULL值的列，如果用户在文件中没有电子邮件地址，则cust_email列将包含NULL值：

输入

```
SELECT cust_id
FROM customers
WHERE cust_email IS NULL;
```

输出

```
cust_id
-----------
10002
10005
```

51

NULL与不匹配　你可能希望在通过过滤选择出不具有特定值的行时，也返回具有NULL值的行。但是，不行。因为未知具

有特殊的含义，数据库不知道它们是否匹配，所以在匹配过滤或不匹配过滤时不返回它们。

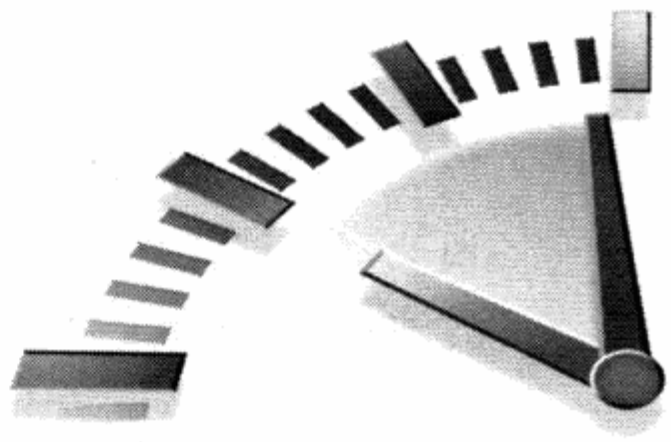
因此，在过滤数据时，一定要验证返回数据中确实给出了被过滤列具有NULL的行。

6.3 小结

本章介绍了如何用SELECT语句的WHERE子句过滤返回的数据。我们学习了如何对相等、不相等、大于、小于、值的范围以及NULL值等进行测试。

第7章

数据过滤



本章讲授如何结合WHERE子句建立功能强大的高级搜索条件。我们还将学习如何使用NOT和IN操作符。

7.1 组合WHERE子句

第6章中介绍的所有WHERE子句在过滤数据时使用的都是单一的条件。为了进行更强的过滤控制，T-SQL允许给出多个WHERE子句。这些子句可以两种方式使用，即以AND子句的方式或OR子句的方式使用。

操作符（operator） 用来联结或改变WHERE子句中的子句的关键字，也称为逻辑操作符（logical operator）。

7.1.1 AND操作符

为了通过不止一个列进行过滤，可使用AND操作符给WHERE子句附加条件。下面的代码给出了一个例子：

输入

```
SELECT prod_id, prod_price, prod_name
FROM products
WHERE vend_id = 1003 AND prod_price <= 10;
```


分析 此SQL语句检索由供应商1003制造且价格小于等于10美元的所有产品的名称和价格。这条SELECT语句中的WHERE子句包含两个条件，并且用AND关键字联结它们。AND指示SQL Server只返回满足所有给定条件的行。如果某个产品由供应商1003制造，但它的价格高于10美元，则不检索它。类似，如果产品价格小于10美元，但不是由指定供应商制造的也不被检索。这条SQL语句产生的输出如下：

输出

```
prod_id    prod_price            prod_name
---------- --------------------- --------------
FB         10.00                 Bird seed
FC         2.50                  Carrots
SLING      4.49                  Sling
TNT1       2.50                  TNT (1 stick)
TNT2       10.00                 TNT (5 sticks)
```

AND 用在WHERE子句中的关键字，用来指示检索满足所有给定条件的行。

这个例子包含一个AND子句，因此，它是由两个过滤条件所构成的。也可以使用更多的过滤条件，每个都用AND关键字分隔。

7.1.2 OR操作符

正如上述，使用AND操作符的前提是要检索的行必须满足两个条件。OR操作符指示SQL Server检索匹配任一条件的行，因此，在第一个条件满足时，不管第二个条件是否满足，相应的行都将被检索出来。

请看如下的SELECT语句：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id = 1002 OR vend_id = 1003;
```

分析 此SQL语句检索由任一个指定供应商制造的所有产品的产品名和价格。OR操作符告诉DBMS匹配任一条件而不是同时匹配两个条件。如果这里使用的是AND操作符，则没有数据返回（它将创建一条永远不可能匹配的WHERE子句）。这条SQL语句产生的输出如下：

输出

```
prod_name       prod_price
--------------- ----------
Detonator       13.00
Bird seed       10.00
Carrots         2.50
Fuses           3.42
Oil can         8.99
Safe            50.00
Sling           4.49
TNT (1 stick)   2.50
TNT (5 sticks)  10.00
```

OR　WHERE子句中使用的关键字，用来表示检索匹配任一给定条件的行。

7.1.3　计算次序

WHERE可包含任意多个AND和OR操作符。两者结合可以进行复杂和高级的过滤。

55

但是，组合AND和OR带来了一个有趣的问题。为了说明这个问题，来看一个例子。假如需要列出价格为10美元（含）以上且由1002或1003制造的所有产品。下面的SELECT语句使用AND和OR操作符的组合建立了一个WHERE子句：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id = 1002 OR vend_id = 1003 AND prod_price >= 10;
```

输出

```
prod_name          prod_price
---------------- ----------
Detonator          13.00
Bird seed          10.00
Fuses              3.42
Oil can            8.99
Safe               50.00
TNT (5 sticks)     10.00
```

分析　请看上面的结果。返回的行中有2行价格小于10美元，显然，返回的行未按预期的进行过滤。为什么会这样呢？原因在于计算的次序。T-SQL（像多数语言一样）在处理OR操作符前，优先处理AND操作符。当SQL Server看到上述WHERE子句时，它理解为由供应商1003制造的任何价格为10美元（含）以上的产品，或者由供应商1002制造的任何产品，而不管其价格如何。换句话说，由于AND在计算次序中优先级更高，操作符被错误地组合了。

此问题的解决方法是使用圆括号把相应的操作符进行明确的分组。请看下面的SELECT语句及输出：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE (vend_id = 1002 OR vend_id = 1003) AND prod_price >= 10;
```

56

输出

```
prod_name          prod_price
---------------    ----------
Detonator          13.00
Bird seed          10.00
Safe               50.00
TNT (5 sticks)     10.00
```

分析 这条SELECT语句与前一条的唯一差别是，在这条语句中，前两个条件用圆括号括了起来。因为圆括号具有较AND或OR操作符高的计算次序，SQL Server首先过滤圆括号内的OR条件。这时，SQL语句变成了选择由供应商1002或1003制造的且价格都在10美元（含）以上的任何产品，这正是我们所希望的。

在WHERE子句中使用圆括号 任何时候使用具有AND和OR操作符的WHERE子句，都应该使用圆括号明确地分组操作符。不要过分依赖默认计算次序，即使它确实是你想要的东西也是如此。使用圆括号没有什么坏处，它能消除歧义。

7.2 IN操作符

圆括号在WHERE子句中还有其他不同的用途。IN操作符用来指定条件范围，范围中的每个条件都可以进行匹配。IN取合法值的由逗号分隔的清单，全都括在圆括号中。下面的例子说明了这个操作符：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id IN (1002,1003)
ORDER BY prod_name;
```

57

输出

```
prod_name          prod_price
---------------    ----------
Bird seed          10.00
Carrots            2.50
Detonator          13.00
Fuses              3.42
Oil can            8.99
Safe               50.00
Sling              4.49
TNT (1 stick)      2.50
TNT (5 sticks)     10.00
```

分析　此SELECT语句检索供应商1002和1003制造的所有产品。IN操作符后跟由逗号分隔的合法值清单，整个清单必须括在圆括号中。

如果你认为IN操作符完成与OR相同的功能，那么你的这种猜测是对的。下面的SQL语句完成与上面的例子相同的工作：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id  = 1002 OR vend_id = 1003
ORDER BY prod_name;
```

输出

```
prod_name         prod_price
--------------- ----------
Bird seed         10.00
Carrots           2.50
Detonator         13.00
Fuses             3.42
Oil can           8.99
Safe              50.00
Sling             4.49
TNT (1 stick)     2.50
TNT (5 sticks)    10.00
```


为什么要使用IN操作符？其优点为：

- 在使用长的合法选项清单时，IN操作符的语法更清楚且更直观；
- 在使用IN时，计算的次序更容易管理（因为使用的操作符更少）；
- IN操作符一般比OR操作符清单执行更快；
- IN的最大优点是可以包含其他SELECT语句，使得能够更动态地建立WHERE子句（第13章将对此进行详细介绍）。

IN　WHERE子句中用来指定要匹配值的清单的关键字，功能与OR相当。

7.3　NOT操作符

WHERE子句中的NOT操作符有且只有一个功能，那就是否定它之后所跟的任何条件。

NOT　WHERE子句中用来否定后跟条件的关键字。

下面的例子说明NOT的使用。为了列出除1002和1003之外的所有供应商制造的产品，可编写如下的代码：

输入

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id NOT IN (1002,1003)
ORDER BY prod_name;
```

59

输出

```
prod_name      prod_price
-------------- ----------
.5 ton anvil   5.99
1 ton anvil    9.99
2 ton anvil    14.99
JetPack 1000   35.00
JetPack 2000   55.00
```

分析 这里的NOT否定跟在它之后的条件；因此，SQL Server不是匹配vend_id为1002和1003的供应商，而是匹配除1002和1003之外的其他所有供应商。

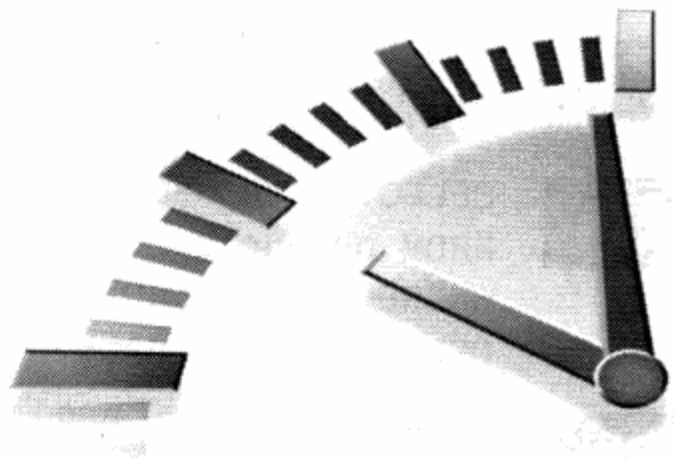
为什么使用NOT？对于这里的这种简单的WHERE子句，使用NOT确实没有什么优势。但在更复杂的子句中，NOT是非常有用的。例如，在与IN操作符联合使用时，NOT使找出与条件列表不匹配的行非常简单。

7.4 小结

本章讲授如何用AND和OR操作符组合成WHERE子句，而且还讲授了如何明确地管理计算的次序，如何使用IN和NOT操作符。

60

第8章



用通配符进行过滤

本章介绍什么是通配符、如何使用通配符以及怎样使用LIKE操作符进行通配搜索，以便对数据进行复杂过滤。

8.1 LIKE操作符

前面介绍的所有操作符都是针对已知值进行过滤的。不管是匹配一个还是多个值，测试大于还是小于已知值，或者检查某个范围的值，共同点是过滤中使用的值都是已知的。但是，这种过滤方法并不是任何时候都好用。例如，怎样搜索产品名中包含文本anvil的所有产品？用简单的比较操作符肯定不行，必须使用通配符。利用通配符可创建比较特定数据的搜索模式。在这个例子中，如果你想找出名称包含anvil的所有产品，可构造一个通配符搜索模式，找出产品名中任何位置出现anvil的产品。

通配符（wildcard） 用来匹配值的一部分的特殊字符。

搜索模式（search pattern）[①] 由字面值、通配符或两者组合构成的搜索条件。

通配符本身实际是SQL的WHERE子句中有特殊含义的字符，SQL支持几种通配符。

为在搜索子句中使用通配符，必须使用LIKE操作符。LIKE指示SQL

① 数据库中的schema（见1.1.2节）和pattern都译作“模式”，特此说明，请读者注意。——编者注

Server，后跟的搜索模式利用通配符匹配而不是直接相等匹配进行比较。

谓词 操作符何时不是操作符？答案是在它作为谓词（predicate）时。从技术上说，LIKE是谓词而不是操作符。虽然最终的结果是相同的，但应该对此术语有所了解，以免在SQL Server文献中或手册中遇到此术语时不知道。

8.1.1 百分号（%）通配符

最常使用的通配符是百分号（%）。在搜索串中，%表示匹配出现任意次数的任何字符。例如，为了找出所有以词jet起头的产品，可发布以下SELECT语句：

输入

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE 'jet%';
```

输出

```
prod_id    prod_name
---------- --------------
JP1000     JetPack 1000
JP2000     JetPack 2000
```

分析 此例子使用了搜索模式'jet%'。在执行这条子句时，将检索任意以jet起头的词。%告诉SQL Server接受jet之后的任意字符，不管它有多少字符。

62

区分大小写 根据SQL Server的配置，搜索可以是区分大小写的。如果区分大小写，'jet%'与JetPack 1000将不匹配。但是多数SQL Server安装的默认行为是不区分大小写的。

通配符可在搜索模式中任意位置使用，并且可以使用多个通配符。下面的例子使用两个通配符，它们位于模式的两端：

输入

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '%anvil%';
```

输出

```
prod_id    prod_name
---------- --------------
ANV01      .5 ton anvil
```

```
ANV02        1 ton anvil
ANV03        2 ton anvil
```

分析 搜索模式'%anvil%'表示匹配任何位置包含文本anvil的值，而不论它之前或之后出现什么字符。

通配符也可以出现在搜索模式的中间，虽然这样做不太有用。下面的例子找出以s起头以e结尾的所有产品：

输入

```
SELECT prod_name
FROM products
WHERE prod_name LIKE 's%e';
```


重要的是要注意到，除了一个或多个字符外，%还能匹配0个字符。%代表搜索模式中给定位置的0个、1个或多个字符。

注意尾空格 尾空格可能会干扰通配符匹配。例如，在保存词anvil时，如果它后面有一个或多个空格，则子句WHERE prod_name LIKE '%anvil'将不会匹配它们，因为在最后的l后有多余的字符。解决这个问题的一个简单的办法是在搜索模式最后附加一个%。一个更好的办法是使用函数（如第10章所介绍）去掉首尾空格。

注意NULL 虽然似乎%通配符可以匹配任何东西，但有一个例外，即NULL。即使是WHERE prod_name LIKE'%'也不能匹配用值NULL作为产品名的行。

8.1.2 下划线（_）通配符

另一个有用的通配符是下划线（_）。下划线的用途与%一样，但下划线只匹配单个字符而不是多个字符。

举一个例子：

输入

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '_ ton anvil%';
```

输出

```
prod_id    prod_name
---------- -------------
ANV02      1 ton anvil
ANV03      2 ton anvil
```

64

分析 此WHERE子句中的搜索模式给出了后面跟有文本的通配符。结果只显示匹配搜索模式的行：第一行中下划线匹配1，第二行中匹配2。.5 ton anvil产品没有匹配，因为搜索模式要求匹配一个通配符而不是两个。对照一下，下面的SELECT语句使用%通配符，返回三行产品：

输入

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '% ton anvil%';
```

输出

```
prod_id    prod_name
---------- -------------
ANV01      .5 ton anvil
ANV02      1 ton anvil
ANV03      2 ton anvil
```

与%能匹配0个字符不一样，_总是匹配一个字符，不能多也不能少。

8.1.3 方括号（[]）通配符

方括号（[]）通配符用来指定一个字符集，它必须匹配指定位置（通配符的位置）的一个字符。

例如，为找出所有名字以E或J起头的联系人，可如下查询：

输入

```
SELECT cust_contact
FROM customers
WHERE cust_contact LIKE '[EJ]%'
ORDER BY cust_contact;
```

65

输出

```
cust_contact
---------------
E Fudd
Jerry Mouse
Jim Jones
```

分析 此语句的WHERE子句中的模式为'[EJ]%'。此搜索模式使用了两个不同的通配符。[EJ]匹配任何以方括号中字母开头的联系人名，它也只能匹配单个字符。因此，任何多于一个字符的名字都不匹配。[EJ]之后的%通配符匹配第一个字符之后的任意数目的字符，返回所需结果。

此通配符可以用前缀字符^（脱字符号）来否定。例如，下面的查询匹配不以E或J起头的任意联系人名（与前一个例子相反）：

输入

```
SELECT cust_contact
FROM customers
WHERE cust_contact LIKE '[^EJ]%'
ORDER BY cust_contact;
```

当然，也可以使用NOT操作符得出相同的结果。^的唯一优点是在使用多个WHERE子句时简化语法：

输入

```
SELECT cust_contact
FROM Customers
WHERE NOT cust_contact LIKE '[EJ]%'
ORDER BY cust_contact;
```

66

8.2　使用通配符的技巧

正如所见，T-SQL的通配符很有用。但这种功能是有代价的，即通配符搜索的处理一般要比前面讨论的其他搜索所花时间更长。这里给出一些使用通配符要记住的技巧。

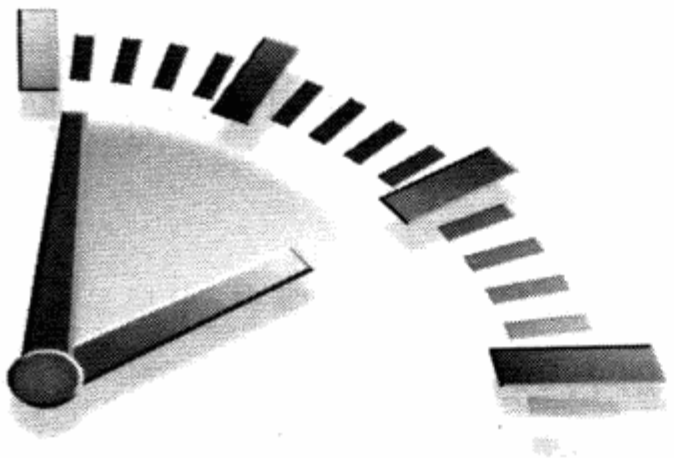
- 不要过度使用通配符。如果其他操作符能达到相同的目的，应该使用其他操作符。
- 在确实需要使用通配符时，除非绝对有必要，否则不要把它们用在搜索模式的开始处。把通配符置于搜索模式的开始处，搜索起来是最慢的。
- 仔细注意通配符的位置。如果放错地方，可能不会返回想要的数据。

总之，通配符是一种极重要和有用的搜索工具，以后我们经常会用到它。

8.3　小结

本章介绍了什么是通配符以及如何在WHERE子句中使用SQL通配符，并且还说明了通配符应该细心使用，不要过度使用。

67



第9章 创建计算字段

本章介绍什么是计算字段，如何创建计算字段以及怎样从应用程序中使用别名引用它们。

9.1 计算字段

存储在数据库表中的数据一般不是应用程序所需要的格式。下面举几个例子：

- 你想在一个字段中既显示公司名又显示公司的地址，但这两个信息一般包含在不同的表列中。
- 城市、州和邮政编码存储在不同的列中（应该这样），但邮件标签打印程序却需要把它们作为一个恰当格式的字段检索出来。
- 列数据是大小写混合的，但报表程序需要把所有数据按大写表示出来。
- 物品订单表存储物品的价格和数量，但不需要存储每个物品的总价格（用价格乘以数量即可）。为打印发票，需要物品的总价格。
- 需要根据表数据进行总数、平均数计算或其他计算。

在上述每个例子中，存储在表中的数据都不是应用程序所需要的。我们需要直接从数据库中检索出转换、计算或格式化过的数据；而不是检索出数据，然后再在客户机应用程序或报表程序中重新格式化。

这就是计算字段发挥作用的所在了。与前面各章介绍过的列不同，计算字段并不实际存在于数据库表中。计算字段是运行时在SELECT语句内创建的。

字段（field） 基本上与列（column）的意思相同，经常互换使用，不过数据库列一般称为列，而术语字段通常用在计算字段的连接上。

重要的是要注意到，只有数据库知道SELECT语句中哪些列是实际的表列，哪些列是计算字段。从客户机（如应用程序）的角度来看，计算字段的数据是以与其他列的数据相同的方式返回的。

客户机与服务器的格式 可在SQL语句内完成的许多转换和格式化工作都可以直接在客户机应用程序内完成。但一般来说，在数据库服务器上完成这些操作比在客户机中完成要快得多，因为DBMS是设计来快速有效地完成这种处理的。

9.2 拼接字段

为了说明如何使用计算字段，举一个创建由两列组成的标题的简单例子。

70

vendors表包含供应商名和位置信息。假如要生成一个供应商报表，需要在供应商的名字中按照name(location)这样的格式列出供应商的位置。

此报表需要单个值，而表中数据存储在两个列vend_name和vend_country中。此外，需要用括号将vend_country括起来，这些东西都没有存储在数据库表中。我们来看看怎样编写返回供应商名和位置的SELECT语句。

拼接（concatenate） 将值联结到一起构成单个值。

解决办法是把两个列拼接起来。在T-SQL中的SELECT语句中，可使用＋操作符来拼接两个列。

输入

```
SELECT vend_name + ' (' + vend_country + ')'
FROM vendors
ORDER BY vend_name;
```

输出

```
----------------------------------------------------
ACME                (USA                           )
Anvils R Us         (USA                           )
Furball Inc.        (USA                           )
Jet Set             (England                       )
Jouets Et Ours      (France                        )
LT Supplies         (USA                           )
```

分析 +操作符拼接串，即把多个串连接起来形成一个较长的串。上面的SELECT语句连接以下元素：

71

- 存储在vend_name列中的名字；
- 包含一个空格和一个左圆括号的串；
- 存储在vend_country列中的国家；
- 包含一个右圆括号的串。

从上述输出中可以看到，SELECT语句返回包含上述4个元素的单个列（计算字段）。但是，由于SQL Server以定长列存储数据，检索出的列全都补充空格以达到最大长度。因此，新的计算列包含附加的空格，并非正好是我们所查找到的东西。

在第8章中曾提到通过删除数据右侧多余的空格来整理数据，这可以使用T-SQL的RTrim()函数来完成，如下所示：

输入

```
SELECT RTrim(vend_name) + ' (' + RTrim(vend_country) + ')'
FROM vendors
ORDER BY vend_name;
```

输出

```
----------------------------------------------
ACME (USA)
Anvils R Us (USA)
Furball Inc. (USA)
Jet Set (England)
Jouets Et Ours (France)
LT Supplies (USA)
```

分析 RTrim()函数去掉值右边的所有空格。通过使用RTrim()，各个列都进行了整理。

72

LTrim()函数 T-SQL除了支持RTrim()（正如刚才所见，它去掉串的右边空格），还支持LTrim()（去掉串左边的空格）。为了去掉串左右两边的空格可以使用RTrim(LTrim(vend_name))。

使用别名

从前面的输出中可以看到，SELECT语句拼接地址字段工作得很好。但此新计算列的名字是什么呢？实际上它没有名字，它只是一个值。如果仅在SQL查询工具中查看一下结果，这样没有什么不好。但是，一个未命名的列不能用于客户机应用中，因为客户机没有办法引用它。

为了解决这个问题，SQL支持列别名。别名（alias）是一个字段或值的替换名。别名用AS关键字赋予。请看下面的SELECT语句：

输入

```
SELECT RTrim(vend_name) + ' (' + RTrim(vend_country) + ')' AS
➥vend_title
FROM vendors
ORDER BY vend_name;
```

输出

```
vend_title
----------------------------------------------------
ACME (USA)
Anvils R Us (USA)
Furball Inc. (USA)
Jet Set (England)
Jouets Et Ours (France)
LT Supplies (USA)
```


分析 SELECT语句本身与以前使用的相同，只不过这里的语句中计算字段之后跟了文本AS vend_title。它指示SQL Server创建一个包含指定计算的名为vend_title的计算字段。从输出中可以看到，结果与以前的相同，但现在列名为vend_title，任何客户机应用都可以按名引用这个列，就像它是一个实际的表列一样。

AS是可选的　与多数SQL实现不一样，在T-SQL中，AS关键字实际上是可选的。因此，SELECT vend_name AS VendName和SELECT vend_name VendName是相同的东西。不过，总是给出AS关键字是一个很好的主意（使你在使用别的DBMS时习惯于使用它）。

派生列 别名有时也称为派生列（derived column），不管称为什么，它们所代表的都是相同的东西。

别名还有其他用途。常见的用途包括在实际的表列名包含不符合规定的字符（如空格）时重新命名它，在原来的名字含混或容易误解时扩充它，等等。例如，如果某个表包含一个名为Last Name的列（列名中有一个空格），在SQL语句以及你的应用中使用该列将非常困难。解决办法是使用别名，如下所示：

```
SELECT [Last Name] AS LastName
```

其中，[和]用来界定列名，给Last Name起别名为LastName。

74

9.3 执行算术计算

计算字段的另一常见用途是对检索出的数据进行算术计算。举一个例子，orders表包含收到的所有订单，orderitems表包含每个订单中的各项物品。下面的SQL语句检索订单号20005中的所有物品：

输入

```
SELECT prod_id, quantity, item_price
FROM orderitems
WHERE order_num = 20005;
```

输出

```
prod_id    quantity    item_price
---------- ----------- ----------------------
ANV01      10          5.99
ANV02      3           9.99

TNT2       5           10.00
FB         1           10.00
```

item_price列包含订单中每项物品的单价。如下汇总物品的价格（单价乘以订购数量）：

输入

```
SELECT prod_id,
       quantity,
```

```
        item_price,
        quantity*item_price AS expanded_price
FROM orderitems
WHERE order_num = 20005;
```

75

输出

```
prod_id    quantity    item_price           expanded_price
---------- ----------- -------------------- --------------
ANV01      10          5.99                 59.90
ANV02      3           9.99                 29.97
TNT2       5           10.00                50.00
FB         1           10.00                10.00
```

分析 输出中显示的expanded_price列为一个计算字段，此计算为quantity*item_price。客户机应用现在可以使用这个新计算列，就像使用其他列一样。

T-SQL支持表9-1中列出的基本算术操作符。此外，圆括号可用来区分优先顺序。关于优先顺序的介绍，请参阅第7章。

表9-1　SQL算术操作符

操作符	说明
+	加
-	减
*	乘
/	除
%	取模（返回余数）

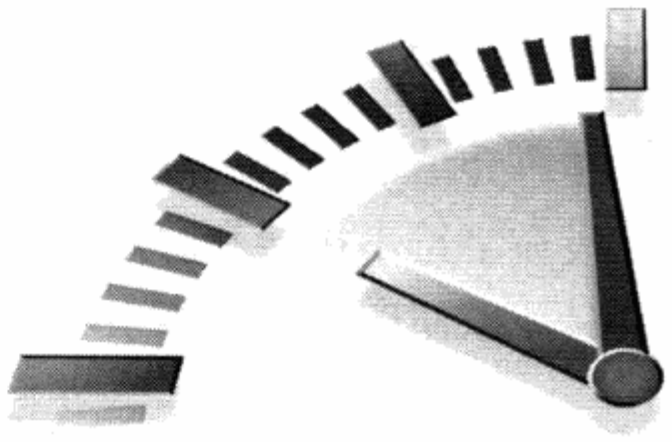
如何测试计算　SELECT提供了测试、试验函数和计算的一个很好的办法。虽然SELECT通常用来从表中检索数据，但可以省略FROM子句以便简单地访问和处理表达式。例如，SELECT 3*2;将返回6，SELECT Trim('　abc　');将返回abc，而SELECT GetDate()利用GetDate()函数返回当前日期和时间。通过这些例子，可以明白如何根据需要使用SELECT进行试验。

76

9.4 小结

本章介绍了计算字段以及如何创建计算字段。我们用例子说明了计算字段在字符串拼接和算术计算的用途。此外，还学习了如何创建和使用别名，以便应用程序能引用计算字段。

第 10 章



使用数据处理函数

本章介绍什么是函数，T-SQL支持何种函数，以及如何使用这些函数。

10.1 函数

与其他大多数计算机语言一样，SQL支持利用函数来处理数据。函数一般是在数据上执行的，它给数据的转换和处理提供了方便。

在前一章中用来去掉字符串尾空格的RTrim()就是一个函数的例子。

函数的可移植性没有SQL的可移植性高 可以在多个系统上运行的代码被认为是可移植的。大多数SQL语句是可移植的，且当SQL实现之间存在差异时，通常并不难处理。另一方面，函数是不可移植的。几乎每个主要的DBMS都会支持其他DBMS不支持的函数，这些差异有时非常重要。

为了使代码具有可移植性，许多SQL程序员不赞成使用特殊实现的功能。虽然这样做很有好处，但不总是利于应用程序的性能。如果不使用这些函数，编写某些应用程序代码会很艰难。必须利用其他方法来实现DBMS能非常有效地完成的工作。

如果你决定使用函数，应该保证做好代码注释，以便以后你（或其他人）能确切地知道所编写SQL代码的含义。

10.2 使用函数

大多数SQL实现支持以下类型的函数。

- 用于处理字符串（如删除或填充值，转换值为大写或小写）的文本函数。
- 用于在数值数据上进行算术操作（如返回绝对值，进行代数运算）的数值函数。
- 用于处理日期和时间值并从这些值中提取特定成分（例如，返回两个日期之差，检查日期有效性）的日期和时间函数。
- 返回DBMS正使用的特殊信息（如返回用户登录信息，检查版本信息）的系统函数。

10.2.1 文本处理函数

上一章中我们已经看过一个文本处理函数的例子，其中使用RTrim()函数来去除列值末尾的空格。下面是另一个例子，这次使用Upper()函数：

输入

```
SELECT vend_name, Upper(vend_name) AS vend_name_upcase
FROM vendors
ORDER BY vend_name;
```

输出

```
vend_name            vend_name_upcase
-------------------- ---------------
ACME                 ACME
Anvils R Us          ANVILS R US
Furball Inc.         FURBALL INC.
Jet Set              JET SET
Jouets Et Ours       JOUETS ET OURS
LT Supplies          LT SUPPLIES
```

80

分析 可以看到，Upper()将文本转换为大写，因此本例子中每个供应商都列出两次，第一次为vendors表中存储的值，第二次作为列vend_name_upcase转换为大写。

表10-1列出了某些常用的文本处理函数。

表10-1 常用的文本处理函数

函 数	说 明
CharIndex()	返回字符串中指定字符的位置
Left()	返回字符串左边的字符
Len()	返回字符串的长度
Lower()	将字符串转换为小写

（续）

函　数	说　明
LTrim()	去掉字符串左边的空格
Replace()	用其他特殊字符替换字符串中的字符
Right()	返回字符串右边的字符
RTrim()	去掉字符串右边的空格
Soundex()	返回字符串的SOUNDEX值
Str()	将数值转换为字符串
SubString()	返回字符串中的字符
Upper()	将字符串转换为大写

表10-1中的SOUNDEX需要做进一步的解释。SOUNDEX是一个将任何文本字符串转换为描述其语音表示的字母数字模式的算法。SOUNDEX考虑了类似的发音字符和音节，使得能对字符串进行发音比较而不是字母比较。虽然SOUNDEX不是SQL概念，但T-SQL（如多数DBMS）都提供对SOUNDEX的支持。

下面给出一个使用Soundex()函数的例子。customers表中有一个顾客Coyote Inc.，其联系名为Y.Lee。但如果这是输入错误，此联系名实际变成Y.Lie，怎么办？显然，按正确的联系名搜索不会返回数据，如下所示：

81

输入

```
SELECT cust_name, cust_contact
FROM customers
WHERE cust_contact = 'Y. Lie';
```

输出

```
cust_name                cust_contact
------------------------ ------------------
```

现在试一下使用Soundex()函数进行搜索，它匹配所有发音类似于Y.Lie的联系名：

输入

```
SELECT cust_name, cust_contact
FROM customers
WHERE Soundex(cust_contact) = Soundex('Y Lie');
```

输出

```
cust_name                cust_contact
------------------------ ------------------
Coyote Inc.              Y Lee
```

分析　在这个例子中，WHERE子句使用Soundex()函数来转换cust_contact列值和搜索字符串为它们的SOUNDEX值。因为

Y.Lee和Y.Lie发音相似，所以它们的SOUNDEX值匹配，因此WHERE子句正确地过滤出了所需的数据。

10.2.2 日期和时间处理函数

日期和时间采用具有特定内部格式的特殊数据类型存储在表中，因而能快速和有效地排序或过滤，并且节省物理存储空间。

一般而言，应用程序不使用用来存储日期和时间的格式，因此日期和时间函数总是被用来读取、统计和处理这些值。由于这个原因，日期和时间处理函数在T-SQL语言中具有重要的作用。 82

表10-2列出了一些常用的日期和时间处理函数。

表10-2 常用的日期和时间处理函数

函 数	说 明
DateAdd()	添加日期（天、周等）
DateDiff()	计算两个日期的差
DateName()	返回部分日期的字符串表示
DatePart()	返回日期的一部分（星期几、月、年等）
Day()	返回日期中的天
GetDate()	返回当前日期和时间
Month()	返回日期中的月
Year()	返回日期中的年

函数DateDiff()、DateName()和DatePart()要求把日期成分传递给它们。表10-3列出支持的日期成分。（你可以给出日期成分或日期成分的缩写。）

表10-3 所支持的日期成分和缩写

日 期 成 分	缩 写
天	dd或d
年中的天	dy或y
时	hh
毫秒	ms
分	mi或n
月份	m或mm

（续）

日期成分	缩　写
季度	q或qq
秒	ss或s
周	wk或ww
周中的天（仅用于DatePart()）	dw
年	yy或yyyy

83

例如，为了获得下订单日期为星期几，可使用DatePart()，指定weekday作为日期成分：

输入

```
SELECT order_num,
       DatePart(weekday, order_date) AS weekday
FROM orders;
```

输出

```
order_num   weekday
----------- -----------
20005       5
20006       2
20007       6
20008       2
20009       7
```

分析 DatePart()从日期中提取指定的日期成分。DatePart(weekday,order_date)返回每个order_date值的星期几表示，如别名weekday列所示。

为了返回星期几的英文单词（而不是数值），可以大致相同的方式使用DateName()函数：

输入

```
SELECT order_num,
       DateName(weekday,order_date) AS
weekday
FROM orders;
```

输出

```
order_num   weekday
----------- ------------
20005       Saturday
20006       Wednesday
20007       Sunday
20008       Wednesday
20009       Monday
```

84

分析 与DatePart()一样，DateName()接受一个日期成分作为它的第一个参数。DateName(weekday, '2005-09-01')返回

Thursday。DateName(month, '2005-08-01')返回August。

DatePart()的快捷方式 Day()、Month()和Year()分别为DatePart(day,)、DatePart(month,)和DatePart(year,)的快捷方式。

这是重新复习用WHERE进行数据过滤的一个好时机。迄今为止，我们都是用比较数值和文本的WHERE子句过滤数据，但数据经常需要用日期进行过滤。用日期进行过滤需要注意一些别的问题和使用特殊的T-SQL函数。

首先需要注意的是SQL Server使用的日期格式。无论你什么时候指定一个日期，不管是插入或更新表值还是用WHERE子句进行过滤，日期必须为已认可的格式之一。T-SQL支持以下几种日期的串表示格式：

- 2006-08-17；
- August 17, 2006；
- 20060817；
- 8/17/2006。

在这些格式中，要避免使用最后一种。（毕竟04/05/06会产生歧义，可以理解为2006年5月4号，也可以理解为2006年4月5号，还可以理解为2004年5月6号，等等。）

应该总是使用4位数字的年份 支持2位数字的年份，SQL Server处理00-49为2000-2049，处理50-99为1950-1999。虽然它们可能是打算要的年份，但使用完整的4位数字年份更可靠，因为SQL Server不必做出任何假定。

85

因此，基本的日期比较应该很简单：

输入

```
SELECT cust_id, order_num
FROM orders
WHERE order_date = '2005-09-01';
```

输出

```
cust_id     order_num
----------- -----------
10001       20005
```

分析　此SELECT语句正常，它检索出一个订单记录，一个order_date为2005-09-01的订单记录。

但是，使用WHERE order_date = '2005-09-01'可靠吗？order_date的数据类型为datetime。这种类型存储日期及时间值。样例表中的值全都具有时间值00:00:00，但实际中很可能并不总是这样。如果用当前日期和时间存储订单日期（因此你不仅知道订单日期，还知道下订单当天的时间），怎么办？比方存储的order_date值为2005-09-01 11:30:05，则WHERE order_date = '2005-09-01'失败。即使给出具有该日期的一行，也不会把它检索出来，因为WHERE匹配失败。

解决办法是指示SQL Server仅将给出的日期与列中的日期部分进行比较，而不是将给出的日期与整个列值进行比较。为此，必须使用DateDiff()函数。DateDiff()函数用来查明两个日期的区别。请看下面的例子：

输入

```
SELECT cust_id, order_num
FROM orders
WHERE DateDiff(day, order_date, '2005-09-01') = 0;
```

86

输出

```
cust_id     order_num
----------- -----------
10001       20005
```

分析　类似于前面使用过的DatePart()和DateName()，DateDiff()要求3个参数。第一个参数为要比较的日期成分。如果你想按天比较两个日期（检查它们是否同一天），则指定day。如果你想查看两个日期的月份是否相同（不管是月份中的几号），则指定month，如此等等。后两个参数是要比较的日期。DateDiff()返回两个日期之差。返回值0表示没有差别（它们匹配）。返回值5表示第一个日期比第二个日期大5（5月或5天等，有赖于指定的日期成分）。类似，-3表示第二个日期比第一个日期大3。

总是使用DateDiff()　在比较两个日期时，应该总是使用DateDiff()，而且不要假定日期是如何存储的。

在你知道了如何用日期进行相等测试后，其他操作符（第6章介绍）的使用也就很清楚了。

不过，还有一种日期比较需要说明。如果你想检索出2005年9月下的所有订单，怎么办？简单的相等测试不行，因为它也要匹配月份中的天数。有几种解决办法，其中之一为：

输入

```
SELECT cust_id, order_num
FROM orders
WHERE DateDiff(month, order_date, '2005-09-01') = 0;
```

87

输出

```
cust_id     order_num
----------- -----------
10001       20005
10003       20006
10004       20007
```

分析 其中，DateDiff()用来寻找与2005-09-01匹配的订单（那些具有相同月份的订单）。因此，2005-09-10或2005-09-30或其他2005年9月中的任意日期的作用都相同。因为指定的日期成分为month，所以如果日期在相同的月份中，则匹配（月份中的天数忽略）。

还有另外一种办法：

输入

```
SELECT cust_id, order_num
FROM orders
WHERE Year(order_date) = 2005 AND Month(order_date) = 9;
```

分析 Year()从日期（或日期时间）中返回年份。类似，Month()从日期中返回月份。因此，WHERE Year(order_date) = 2005 AND Month(order_date) = 9检索出order_date为2005年9月的所有行。

10.2.3 数值处理函数

数值处理函数仅处理数值数据。这些函数一般主要用于代数、三角或几何运算，因此没有字符串或日期-时间处理函数的使用那么频繁。

具有讽刺意味的是，在主要DBMS的函数中，数值函数是最一致最统一的函数。表10-4列出一些常用的数值处理函数。

88

表10-4 常用数值处理函数

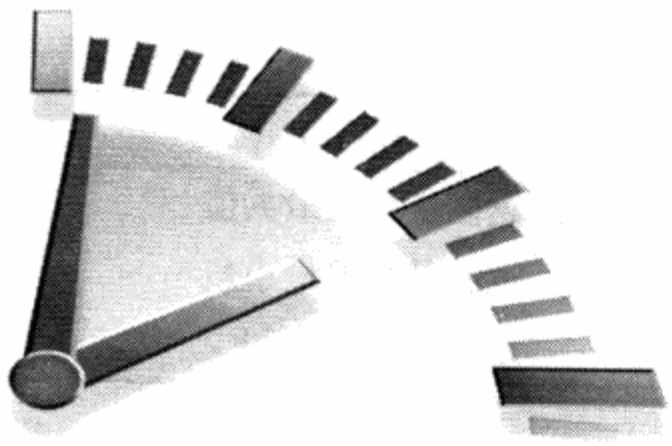
函 数	说 明
Abs()	返回一个数的绝对值
Cos()	返回一个角度的余弦
Exp()	返回一个数的指数值

（续）

函　数	说　明
Pi()	返回圆周率
Rand()	返回一个随机数
Round()	返回四舍五入为特定长度或精度的数值
Sin()	返回一个角度的正弦
Sqrt()	返回一个数的平方根
Square()	返回一个数的平方
Tan()	返回一个角度的正切

10.3　小结

本章介绍了如何使用SQL的数据处理函数，并着重介绍了日期处理函数。



第11章 汇总数据

本章介绍什么是SQL的聚集函数以及如何利用它们汇总表的数据。

11.1 聚集函数

我们经常需要汇总数据而不用把它们实际检索出来，为此SQL Server提供了专门的函数。使用这些函数，T-SQL查询可用于检索数据，以便分析和报表生成。这种类型的检索例子有以下几种。

- 确定表中行数（或者满足某个条件或包含某个特定值的行数）。
- 获得表中行组的和。
- 找出表列（或所有行或某些特定的行）的最大值、最小值和平均值。

上述例子都需要对表中数据（而不是实际数据本身）汇总。因此，返回实际表数据是浪费时间和计算资源（更不用说带宽了）。重复一遍，实际想要的是汇总信息。

为方便这种类型的检索，T-SQL给出了5个聚集函数，见表11-1。这些函数能进行上述罗列的检索。

聚集函数（aggregate function） 运行在行组上，计算和返回单个值的函数。

表11-1 SQL聚集函数

函 数	说 明
Avg()	返回某列的平均值
Count()	返回某列的行数

（续）

函　数	说　明
Max()	返回某列的最大值
Min()	返回某列的最小值
Sum()	返回某列值之和

以下说明各函数的使用。

标准偏差　T-SQL还支持一系列的标准偏差聚集函数，但本书并未涉及这些内容。

11.1.1　Avg()函数

Avg()通过对表中行数计数并计算特定列值之和，求得该列的平均值。Avg()可用来返回所有列的平均值，也可以用来返回特定列或行的平均值。

下面的例子使用Avg()返回products表中所有产品的平均价格：

输入

```
SELECT Avg(prod_price) AS avg_price
FROM products;
```

输出

```
avg_price
--------------------
16.1335
```

92

分析　此SELECT语句返回值avg_price，它包含products表中所有产品的平均价格。如第9章所述，avg_price是一个别名。

Avg()也可以用来确定特定列或行的平均值。下面的例子返回特定供应商所提供产品的平均价格：

输入

```
SELECT Avg(prod_price) AS avg_price
FROM products
WHERE vend_id = 1003;
```

输出

```
avg_price
--------------------
13.2128
```

分析　这条SELECT语句与前一条的不同之处在于它包含了WHERE子句。此WHERE子句仅过滤出vend_id为1003的产品，因此

avg_price中返回的值只是该供应商的产品的平均值。

只用于单个列 Avg()只能用来确定特定数值列的平均值，而且列名必须作为函数参数给出。为了获得多个列的平均值，必须使用多个Avg()函数。

NULL值 Avg()函数忽略列值为NULL的行。

93

11.1.2 Count()函数

Count()函数进行计数。可利用Count()确定表中行的数目或符合特定条件的行的数目。

Count()函数有两种使用方式。

- 使用Count(*)对表中行的数目进行计数，不管表列中包含的是空值（NULL）还是非空值。
- 使用Count(column)对特定列中具有值的行进行计数，忽略NULL值。

下面的例子返回customers表中客户的总数：

输入

```
SELECT Count(*) AS num_cust
FROM customers;
```

输出

```
num_cust
-----------
5
```

分析 在此例子中，利用Count(*)对所有行计数，不管行中各列有什么值。计数值在num_cust中返回。

下面的例子只对具有电子邮件地址的客户计数：

输入

```
SELECT Count(cust_email) AS num_cust
FROM customers;
```

94

输出

```
num_cust
-----------
3
```

分析　这条SELECT语句使用Count(cust_email)对cust_email列中有值的行进行计数。在此例子中，cust_email的计数为3（表示5个客户中只有3个客户有电子邮件地址）。

NULL值　如果指定列名，则指定列的值为空的行被Count()函数忽略，但如果Count()函数中用的是星号（*），则不忽略。

11.1.3　Max()函数

Max()返回指定列中的最大值。Max()要求指定列名，如下所示：

输入

```
SELECT Max(prod_price) AS max_price
FROM products;
```

输出

```
max_price
--------------------
55.00
```

95

分析　这里，Max()返回products表中最贵的物品的价格。

对非数值数据使用Max()　虽然Max()一般用来找出最大的数值或日期值，但T-SQL允许将它用来返回任意列中的最大值，包括返回文本列中的最大值。在用于文本数据时，如果数据按相应的列排序，则Max()返回最后一行。

NULL值　Max()函数忽略列值为NULL的行。

11.1.4　Min()函数

Min()的功能正好与Max()功能相反，它返回指定列的最小值。与Max()一样，Min()要求指定列名，如下所示：

输入

```
SELECT Min(prod_price) AS min_price
FROM products;
```

输出

```
min_price
---------------------
2.50
```

分析 其中Min()返回products表中最便宜物品的价格。

对非数值数据使用Min() Min()函数与Max()函数类似，T-SQL允许将它用来返回任意列中的最小值，包括返回文本列中的最小值。在用于文本数据时，如果数据按相应的列排序，则Min()返回最前面的行。

96

NULL值 Min()函数忽略列值为NULL的行。

11.1.5 Sum()函数

Sum()用来返回指定列值的和（总计）。

下面举一个例子，orderitems包含订单中实际的物品，每个物品有相应的数量（quantity）。可如下检索所订购物品的总数（所有quantity值之和）：

输入

```
SELECT Sum(quantity) AS items_ordered
FROM orderitems
WHERE order_num = 20005;
```

输出

```
items_ordered
------------
19
```

分析 函数Sum(quantity)返回订单中所有物品数量之和，WHERE子句保证只统计某个物品订单中的物品。

Sum()也可以用来合计计算值。在下面的例子中，合计每项物品的item_price*quantity，得出总的订单金额：

输入

```
SELECT Sum(item_price*quantity) AS total_price
FROM orderitems
WHERE order_num = 20005;
```

97

输出

```
total_price
--------------------
149.87
```

分析 函数Sum(item_price*quantity)返回订单中所有物品价钱之和，WHERE子句同样保证只统计某个物品订单中的物品。

在多个列上进行计算 如本例所示，利用标准的算术操作符，所有聚集函数都可用来执行多个列上的计算。

NULL值 Sum()函数忽略列值为NULL的行。

11.2 聚集不同值

以上5个聚集函数都可以如下使用：

- 对所有的行执行计算，指定ALL参数或不给参数（因为ALL是默认行为）；
- 只包含不同的值，指定DISTINCT参数。

ALL为默认 ALL参数不需要指定，因为它是默认行为。如果不指定DISTINCT，则假定为ALL。

98

下面的例子使用Avg()函数返回特定供应商提供的产品的平均价格。它与上面的SELECT语句相同，但使用了DISTINCT参数，因此平均值只考虑各个不同的价格：

输入

```
SELECT Avg(DISTINCT prod_price) AS avg_price
FROM products
WHERE vend_id = 1003;
```

输出

```
avg_price
--------------------
15.998
```

分析 可以看到，在使用了DISTINCT后，此例子中的avg_price比

较高，因为有多个物品具有相同的较低价格。排除它们提升了平均价格。

DISTINCT的使用局限 如果指定列名，则DISTINCT只能用于Count()。DISTINCT不能用于Count(*)。因此，不能使用Count(DISTINCT *)，它会产生错误。类似地，DISTINCT必须使用列名，不能用于计算或表达式。

将DISTINCT用于Min()和Max() 虽然DISTINCT从技术上可用于Min()和Max()，但这样做实际上没有价值。一个列中的最小值和最大值不管是否包含不同值都是相同的。

11.3 组合聚集函数

目前为止的所有聚集函数例子都只涉及单个函数。但实际上SELECT语句可根据需要包含多个聚集函数。请看下面的例子：

输入

```
SELECT Count(*) AS num_items,
       Min(prod_price) AS price_min,
       Max(prod_price) AS price_max,
       Avg(prod_price) AS price_avg
FROM products;
```

输出

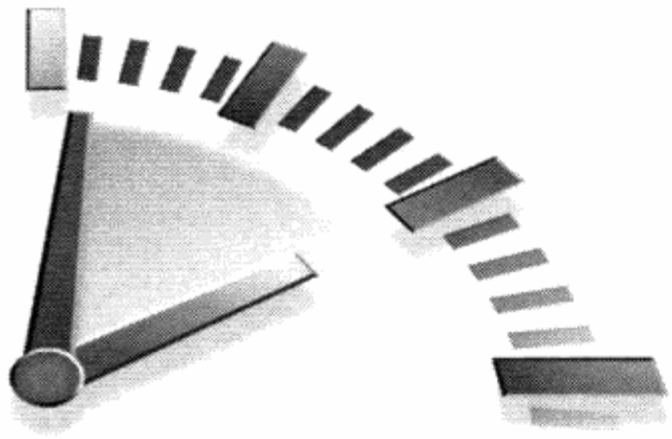
```
num_items   price_min            price_max            price_avg
----------- -------------------- -------------------- ---------
14          2.50                 55.00                16.1335
```

分析 这里用单条SELECT语句执行了4个聚集计算，返回4个值（products表中物品的数目，产品价格的最高、最低以及平均值）。

取别名 在指定别名以包含某个聚集函数的结果时，不应该使用表中实际的列名。虽然这样做并非不合法，但使用唯一的名字会使你的SQL更易于理解和使用（以及将来容易排除故障）。

11.4　小结

聚集函数用来汇总数据。SQL Server支持一系列聚集函数，可以用多种方法使用它们以返回所需的结果。这些函数是高效设计的，它们返回结果一般比你在自己的客户机应用程序中计算要快得多。



第12章 分组数据

本章将介绍如何分组数据，以便能汇总表内容的子集。这涉及两个新SELECT语句子句，分别是GROUP BY子句和HAVING子句。

12.1 数据分组

从上一章知道，SQL聚集函数可用来汇总数据。这使我们能够对行进行计数，计算和与平均数，获得最大和最小值而不用检索所有数据。

目前为止的所有计算都是在表的所有数据或匹配特定的WHERE子句的数据上进行的。提示一下，下面的例子返回供应商1003提供的产品数目：

输入

```
SELECT Count(*) AS num_prods
FROM products
WHERE vend_id = 1003;
```

输出

```
num_prods
-----------
7
```

但如果要返回每个供应商提供的产品数目怎么办？或者返回只提供单项产品的供应商所提供的产品，或返回提供10个以上产品的供应商怎么办？

这就是分组显身手的时候了。分组允许把数据分为多个逻辑组，以便能对每个组进行聚集计算。

12.2 创建分组

分组是在SELECT语句的GROUP BY子句中建立的。理解分组的最好

办法是看一个例子：

输入

```
SELECT vend_id, Count(*) AS num_prods
FROM products
GROUP BY vend_id;
```

输出

```
vend_id     num_prods
----------- -----------
1001        3
1002        2
1003        7
1005        2
```

分析 上面的SELECT语句指定了两个列，vend_id包含产品供应商的ID，num_prods为计算字段（用Count(*)函数建立）。GROUP BY子句指示SQL Server按vend_id排序并分组数据。这导致对每个vend_id而不是整个表计算num_prods一次。从输出中可以看到，供应商1001有3个产品，供应商1002有2个产品，供应商1003有7个产品，而供应商1005有2个产品。

因为使用了GROUP BY，就不必指定要计算和估值的每个组了。系统会自动完成。GROUP BY子句指示SQL Server分组数据，然后对每个组而不是整个结果集进行聚集。

102

在具体使用GROUP BY子句前，需要知道一些重要的规定。

- GROUP BY子句可以包含任意数目的列。这使得能对分组进行嵌套，为数据分组提供更细致的控制。
- 如果在GROUP BY子句中指定多个分组，数据将在最后指定的分组上进行汇总。换句话说，在建立分组时，指定的所有列都一起计算（所以不能从个别的列取回数据）。
- GROUP BY子句中列出的每个列都必须是检索列或有效的表达式（但不能是聚集函数）。如果在SELECT中使用表达式，则必须在GROUP BY子句中指定相同的表达式。不能使用别名。
- 除聚集计算语句外，SELECT语句中的每个列都必须在GROUP BY子句中给出。
- 如果分组列中具有NULL值，则NULL将作为一个分组返回。如果列中有多行NULL值，它们将分为一组。
- GROUP BY子句必须出现在WHERE子句之后，ORDER BY子句之前。

12.3 过滤分组

除了能用GROUP BY分组数据外，SQL Server还允许过滤分组，规定包括哪些分组，排除哪些分组。例如，可能想要列出至少有两个订单的所有顾客。为得出这种数据，必须基于完整的分组而不是个别的行进行过滤。

我们已经看到了WHERE子句的作用（第6章中引入）。但是，在这个例子中WHERE不能完成任务，因为WHERE过滤指定的是行而不是分组。事实上，WHERE没有分组的概念。

103

那么，不使用WHERE使用什么呢？T-SQL为此目的提供了另外的子句，那就是HAVING子句。HAVING非常类似于WHERE。事实上，目前为止所学过的所有类型的WHERE子句都可以用HAVING来替代。唯一的差别是WHERE过滤行，而HAVING过滤分组。

HAVING支持所有WHERE操作符 在第6章和第7章中，我们学习了WHERE子句的条件（包括通配符条件和带多个操作符的子句）。所学过的有关WHERE的所有这些技术和选项都适用于HAVING。它们的句法是相同的，只是关键字有差别。

实际上，在指定的GROUP BY子句用HAVING代替其中的WHERE，其效果是一样的。

那么，怎么过滤行呢？请看以下的例子：

输入

```
SELECT cust_id, Count(*) AS orders
FROM orders
GROUP BY cust_id
HAVING Count(*) >= 2;
```

输出

```
cust_id     orders
----------- -----------
10001       2
```

分析 这条SELECT语句的前3行类似于上面的语句。最后一行增加了HAVING子句，它过滤Count(*) >=2（两个以上的订单）的那些分组。

104

正如所见，这里WHERE子句不起作用，因为过滤是基于分组聚集值而不是特定行值的。

HAVING和WHERE的差别　这里有另一种理解方法，WHERE在数据分组前进行过滤，HAVING在数据分组后进行过滤。这是一个重要的区别，WHERE排除的行不包括在分组中。这可能会改变计算值，从而影响HAVING子句中基于这些值过滤掉的分组。

那么，有没有在一条语句中同时使用WHERE和HAVING子句的需要呢？事实上，确实有。假如想进一步过滤上面的语句，使它返回过去12个月内具有两个以上订单的顾客。为达到这一点，可增加一条WHERE子句，过滤出过去12个月内下过的订单。然后再增加HAVING子句过滤出具有两个以上订单的分组。

为更好地理解，请看下面的例子，它列出具有两个以上、价格为10以上的产品的供应商：

输入

```
SELECT vend_id, Count(*) AS num_prods
FROM products
WHERE prod_price >= 10
GROUP BY vend_id
HAVING Count(*) >= 2;
```

输出

```
vend_id     num_prods
----------- -----------
1003        4
1005        2
```

105

分析　这条语句中，第一行是使用了聚集函数的基本SELECT，它与前面的例子很相像。WHERE子句过滤所有prod_price至少为10的行。然后按vend_id分组数据，HAVING子句过滤计数为2或2以上的分组。如果没有WHERE子句，将会多检索出两行（供应商1002，销售的所有产品价格都在10以下；供应商1001，销售3个产品，但只有一个产品的价格大于等于10）：

输入

```
SELECT vend_id, Count(*) AS num_prods
FROM products
GROUP BY vend_id
HAVING Count(*) >= 2;
```

输出

```
vend_id     num_prods
----------- -----------
1001        3
1002        2
1003        7
1005        2
```

不适用于所有的数据类型 HAVING和GROUP BY不适用于数据类型为text、ntext和image的列中。合时才使用HAVING，而WHERE子句用于标准的行级过滤。

12.4 分组和排序

GROUP BY和ORDER BY执行不同但却相关的功能，许多用户会混淆它们。为了说明何时使用以及为何使用这两个子句，表12-1汇总了它们之间的差别。

106

表12-1 ORDER BY与GROUP BY

ORDER BY	GROUP BY
排序产生的输出	分组行。但输出可能不是分组的顺序
任意列都可以使用（甚至未选择的列也可以使用）	只可能使用选择列或表达式列，而且必须使用每个选择列表达式
不一定需要	如果与聚集函数一起使用列（或表达式），则必须使用

表12-1中列出的第一项差别极为重要。我们经常发现用GROUP BY分组的数据确实是以分组顺序输出的。但情况并不总是这样，它并不是SQL规范所要求的。此外，用户也可能会要求以不同于分组的顺序排序。仅因为你以某种方式分组数据（获得特定的分组聚集值），并不表示你需要以相同的方式排序输出。应该提供明确的ORDER BY子句，即使其效果等同于GROUP BY子句也是如此。

不要忘记ORDER BY 一般在使用GROUP BY子句时，应该也给出ORDER BY子句。这是保证数据正确排序的唯一方法。千万不要仅依赖GROUP BY排序数据。

为说明GROUP BY和ORDER BY的使用方法，请看一个例子。下面的

SELECT语句类似于前面那些例子。它检索总计订单价格大于等于50的订单的订单号和总计订单价格：

输入

```
SELECT order_num, Sum(quantity*item_price) AS ordertotal
FROM orderitems
GROUP BY order_num
HAVING Sum(quantity*item_price) >= 50;
```

107

输出

```
order_num   ordertotal
----------- --------------------
20005       149.87
20006       55.00
20007       1000.00
20008       125.00
```

为按总计订单价格排序输出，需要添加ORDER BY子句，如下所示：

输入

```
SELECT order_num, Sum(quantity*item_price) AS ordertotal
FROM orderitems
GROUP BY order_num
HAVING Sum(quantity*item_price) >= 50
ORDER BY ordertotal;
```

输出

```
order_num   ordertotal
----------- --------------------
20006       55.00
20008       125.00
20005       149.87
20007       1000.00
```

分析 在这个例子中，GROUP BY子句用来按订单号（order_num列）分组数据，以便Sum(*)函数能够返回总计订单价格。HAVING子句过滤数据，使得只返回总计订单价格大于等于50的订单。最后，用ORDER BY子句排序输出。

12.5　SELECT子句顺序

下面回顾一下SELECT语句中子句的顺序。表12-2以在SELECT语句中使用时必须遵循的次序，列出迄今为止所学过的子句。

108

表12-2　SELECT子句及其顺序

子　句	说　明	是否必须使用
SELECT	要返回的列或表达式	是
FROM	从中检索数据的表	仅在从表选择数据时使用

（续）

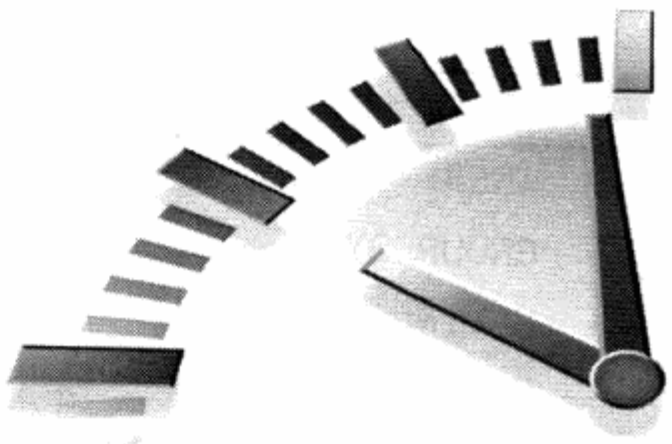
子　句	说　明	是否必须使用
WHERE	行级过滤	否
GROUP BY	分组说明	仅在按组计算聚集时使用
HAVING	组级过滤	否
ORDER BY	输出排序顺序	否

12.6 小结

在第11章中，我们学习了如何用SQL聚集函数对数据进行汇总计算。本章讲授了如何使用GROUP BY子句对数据组进行这些汇总计算，返回每个组的结果。我们看到了如何使用HAVING子句过滤特定的组，还知道了ORDER BY和GROUP BY之间以及WHERE和HAVING之间的差异。

第 13 章

使用子查询



本章介绍什么是子查询以及如何使用它们。

13.1 子查询

SELECT语句是SQL的查询。迄今为止我们所看到的所有SELECT语句都是简单查询，即从单个数据库表中检索数据的单条语句。

查询（query） 任何SQL语句都是查询。但此术语一般指SELECT语句。

SQL还允许创建子查询（subquery），即嵌套在其他查询中的查询。为什么要这样做呢？理解这个概念的最好方法是考察几个例子。

13.2 利用子查询进行过滤

本书所有章中使用的数据库表都是关系表（关于每个表及关系的描述，请参阅附录B）。订单存储在两个表中。对于包含订单编号、客户ID、订单日期的每个订单，orders表存储一行。各订单的物品存储在相关的orderitems表中。orders表不存储客户信息。它只存储客户的ID。实际的客户信息存储在customers表中。

现在，假如需要列出订购物品TNT2的所有客户，应该怎样检索？下面列出具体的步骤。

(1) 检索包含物品TNT2的所有订单的编号。

(2) 检索具有前一步骤列出的订单编号的所有客户的ID。

(3) 检索前一步骤返回的所有客户ID的客户信息。

上述每个步骤都可以单独作为一个查询来执行。可以把一条SELECT语句返回的结果用于另一条SELECT语句的WHERE子句。

也可以使用子查询来把3个查询组合成一条语句。

第一条SELECT语句的含义很明确，对于prod_id为TNT2的所有订单物品，它检索其order_num列。输出列出两个包含此物品的订单：

输入

```
SELECT order_num
FROM orderitems
WHERE prod_id = 'TNT2';
```

输出

```
order_num
-----------
20005
20007
```

下一步，查询具有订单20005和20007的客户ID。利用第7章介绍的IN子句，编写如下的SELECT语句：

输入

```
SELECT cust_id
FROM orders
WHERE order_num IN (20005,20007);
```

输出

```
cust_id
-----------
10001
10004
```

现在，把第一个查询（返回订单号的那一个）变为子查询组合两个查询。请看下面的SELECT语句：

输入

```
SELECT cust_id
FROM orders
WHERE order_num IN (SELECT order_num
                    FROM orderitems
                    WHERE prod_id = 'TNT2');
```

输出

```
cust_id
-----------
10001
10004
```

分析　在SELECT语句中，子查询总是从内向外处理。在处理上面的SELECT语句时，SQL Server实际上执行了两个操作。

首先，它执行下面的查询：

113

```
SELECT order_num FROM orderitems WHERE prod_id='TNT2'
```

此查询返回两个订单号：20005和20007。然后，这两个值以IN操作符要求的逗号分隔的格式传递给外部查询的WHERE子句。外部查询变成：

```
SELECT cust_id FROM orders WHERE order_num IN (20005,20007)
```

可以看到，输出是正确的并且与前面硬编码WHERE子句所返回的值相同。

格式化SQL　包含子查询的SELECT语句难以阅读和调试，特别是它们较为复杂时更是如此。如上所示把子查询分解为多行并且适当地进行缩进，能极大地简化子查询的使用。

现在得到了订购物品TNT2的所有客户的ID。下一步是检索这些客户ID的客户信息。检索两列的SQL语句为：

输入

```
SELECT cust_name, cust_contact
FROM customers
WHERE cust_id IN (10001,10004);
```

可以把其中的WHERE子句转换为子查询而不是硬编码这些客户ID：

输入

```
SELECT cust_name, cust_contact
FROM customers
WHERE cust_id IN (SELECT cust_id
                  FROM orders
                  WHERE order_num IN (SELECT order_num
                                      FROM orderitems
                                      WHERE prod_id = 'TNT2'));
```

114

输出

```
cust_nam             cust_contact
-------------------- --------------
Coyote Inc.          Y Lee
Yosemite Place       Y Sam
```

分析 为了执行上述SELECT语句，SQL Server实际上必须执行3条SELECT语句。最里边的子查询返回订单号列表，此列表用于其外面的子查询的WHERE子句。外面的子查询返回客户ID列表，此客户ID列表用于最外层查询的WHERE子句。最外层查询确实返回所需的数据。

可见，在WHERE子句中使用子查询能够编写出功能很强并且很灵活的SQL语句。对于能嵌套的子查询的数目没有限制，不过在实际使用时由于性能的限制，不能嵌套太多的子查询。

只能是单列 请注意，与EXISTS结合使用的子查询（稍后会讲到）只能检索单个列。企图检索多个列将返回错误。

虽然子查询一般与IN操作符结合使用，但也可以用于测试等于（=）、不等于（<>）等。

子查询和性能 这里给出的代码有效并获得所需的结果。但是，使用子查询并不总是执行这种类型的数据检索的最有效的方法。更多的论述，请参阅第14章，其中将再次给出这个例子。

13.3 作为计算字段使用子查询

使用子查询的另一方法是创建计算字段。假如需要显示customers表中每个客户的订单总数。订单与相应的客户ID存储在orders表中。

为了执行这个操作，遵循下面的步骤。

(1) 从customers表中检索客户列表。
(2) 对于检索出的每个客户，统计其在orders表中的订单数目。

正如前两章所述，可使用SELECT Count(*)对表中的行进行计数，并且通过提供一条WHERE子句来过滤某个特定的客户ID，可仅对该客户的订单进行计数。例如，下面的代码对客户10001的订单进行计数：

输入

```
SELECT Count(*) AS orders
FROM orders
WHERE cust_id = 10001;
```

为了对每个客户执行COUNT(*)计算，应该将Count(*)作为一个子查询。请看下面的代码：

输入

```
SELECT cust_name,
       cust_state,
       (SELECT Count(*)
        FROM orders
        WHERE orders.cust_id = customers.cust_id) AS orders
FROM customers
ORDER BY cust_name;
```

116

输出

```
cust_name                 cust_state orders
------------------------- ---------- ----------
Coyote Inc.               MI         2
E Fudd                    IL         1
Mouse House               OH         0
Wascals                   IN         1
Yosemite Place            AZ         1
```

分析 这条SELECT语句对customers表中每个客户返回3列：cust_name、cust_state和orders。orders是一个计算字段，它是由圆括号中的子查询建立的。该子查询对检索出的每个客户执行一次。在此例子中，该子查询执行了5次，因为检索出了5个客户。

子查询中的WHERE子句与前面使用的WHERE子句稍有不同，因为它使用了完全限定列名（在第4章中首次提到）。下面的语句告诉SQL比较orders表中的cust_id与当前正从customers表中检索的cust_id：

```
WHERE orders.cust_id = customers.cust_id
```

相关子查询（correlated subquery）　设计外部查询的子查询。

这种类型的子查询称为相关子查询。任何时候只要列名可能有多义性，就必须使用这种语法（表名和列名由一个句点分隔）。为什么这样？我们来看看如果不使用完全限定的列名会发生什么情况：

输入

```
SELECT cust_name,
       cust_state,
       (SELECT Count(*)
        FROM orders
        WHERE cust_id = cust_id) AS orders
FROM customers
ORDER BY cust_name;
```

117

输出

```
cust_name                      cust_state orders
------------------------------ ---------- ----------
Coyote Inc.                    MI         5
E Fudd                         IL         5
Mouse House                    OH         5
Wascals                        IN         5
Yosemite Place                 AZ         5
```

分析 显然，返回的结果不正确（请比较前面的结果），那么，为什么会这样呢？有两个cust_id列，一个在customers中，另一个在orders中，需要比较这两个列以正确地把订单与它们相应的顾客匹配。如果不完全限定列名，SQL Server将假定你是对orders表中的cust_id进行自身比较。而SELECT Count(*) FROM orders WHERE cust_id = cust_id;总是返回orders表中的订单总数（因为SQL Server查看每个订单的cust_id是否与本身匹配，当然，它们总是匹配的）。

虽然子查询在构造这种SELECT语句时极有用，但必须注意限制有歧义性的列名。

不止一种解决方案 正如本章前面所述，虽然这里给出的样例代码运行良好，但它并不是解决这种数据检索的最有效的方法。在后面的章节中我们还要遇到这个例子。

118

13.4 用子查询检查存在性

子查询的另一用途是与EXISTS谓词联合使用。在将EXISTS用于WHERE子句时，请看子查询返回的结果，不是看返回的数据列而是看返回的行。如果该子查询返回行，则EXISTS测试为真且WHERE子句匹配。但是，如果不返回行，则EXISTS测试为假，WHERE子句不匹配。

请看以下SELECT语句：

输入

```
SELECT cust_id, cust_name
FROM customers
WHERE cust_id IN (SELECT cust_id
                  FROM orders
                  WHERE DateDiff(month, order_date,
                                 '2005-09-01') = 0
                   AND customers.cust_id = orders.cust_id);
```

输出

```
cust_id     cust_name
----------- ------------------
10001       Coyote Inc.
10003       Wascals
10004       Yosemite Place
```

分析 此SELECT语句检索2005年9月下订单的顾客的名字和ID。与本章前面的例子类似，WHERE子句使用IN和一个子查询首先找出在指定月份下订单的顾客ID，然后再根据这些顾客ID从customers表选择所需顾客。

119

现在来看另一条SELECT语句，它返回与上一条SELECT语句完全相同的结果：

输入

```
SELECT cust_id, cust_name
FROM customers
WHERE EXISTS (SELECT *
              FROM orders
              WHERE DateDiff(month, order_date,
                             '2005-09-01') = 0
               AND customers.cust_id = orders.cust_id);
```

分析 这条WHERE子句使用EXISTS而不是IN。这条子查询与使用IN的子查询大致相同，这条WHERE子句也匹配customers和orders表中的cust_id列，使得只检索2005年9月下订单的顾客。应该注意，此子查询使用SELECT *，通常在子查询中不允许，虽然选择什么列实际上没有差别，因为它不是返回用来过滤顾客的数据，而只是判断是否存在匹配数据。

那么，使用IN还是EXISTS？多数情况下两者都可以使用，两者都允许用子查询来过滤数据，两者都可以用NOT否定以查找不匹配的行（或查找没有指定月份订单的所有顾客）。实际上，它们之间的最大的差别是性能。有时，使用EXISTS的语句的处理比使用IN的语句的处理快，这就是

为什么最好是试验一下这两种选择（以及第三种选择——使用联结，下一章介绍）的原因。 120

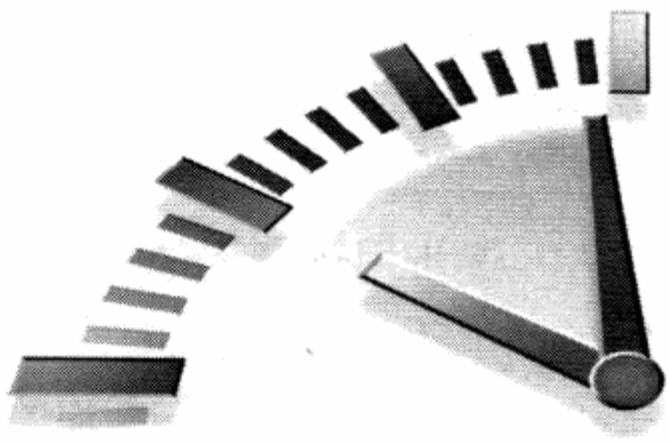
逐步用子查询建立查询 测试和调试具有子查询的查询很有技巧性，特别是对那些越来越复杂的语句更是如此。用子查询建立（和测试）查询的最保险的办法是用类似于SQL Server处理子查询的方式逐渐编写T-SQL代码。首先建立和测试最内层的查询。在已经检验被嵌入子查询正常后用硬编码数据建立和测试外部查询。然后，将被嵌入子查询嵌入外部查询，再次测试它。对增加的每个查询重复这个步骤。这样构造查询需要多花费一点时间，但能节省以后（找出查询为什么不正常）的很多时间，并且提高了查询一投入工作就正常的可能性。

13.5 小结

本章学习了什么是子查询以及如何使用它们。子查询最常见的使用是在WHERE子句的IN操作符中，以及用来填充计算列。我们举了这两种操作类型的例子。 121

第14章

联　结　表



本章将介绍什么是联结，为什么要使用联结，如何编写使用联结的SELECT语句。

14.1　联结

SQL最强大的功能之一就是能在数据检索查询的执行中联结（join）表。联结是利用SQL的SELECT能执行的最重要的操作，很好地理解联结及其语法是学习SQL的一个极为重要的组成部分。

在能够有效地使用联结前，必须了解关系表以及关系数据库设计的一些基础知识。下面的介绍并不是这个内容的全部知识，但作为入门已经足够了。

14.1.1　关系表

理解关系表的最好方法是来看一个现实世界中的例子。

假如有一个包含产品目录的数据库表，其中每种类别的物品占一行。对于每种物品要存储的信息包括产品描述和价格，以及生产该产品的供应商信息。

现在，假如有由同一供应商生产的多种物品，那么在何处存储供应商信息（如，供应商名、地址、联系方法等）呢？将这些数据与产品信息分开存储的理由如下。

- 因为同一供应商生产的每个产品的供应商信息都是相同的，对每个产品重复此信息既浪费时间又浪费存储空间。

- 如果供应商信息改变（例如，供应商搬家或电话号码变动），只需改动一次即可。
- 如果有重复数据（即每种产品都存储供应商信息），很难保证每次输入该数据的方式都相同。不一致的数据在报表中很难利用。

关键是，相同数据出现多次决不是一件好事，此因素是关系数据库设计的基础。关系表的设计就是要保证把信息分解成多个表，一类数据一个表。各表通过某些常用的值（即关系设计中的关系（relational））互相关联。

在这个例子中，可建立两个表，一个存储供应商信息，另一个存储产品信息。vendors表包含所有供应商信息，每个供应商占一行，每个供应商具有唯一的标识。此标识称为主键（primary key）（在第1章中首次提到），可以是供应商ID或任何其他唯一值。

products表只存储产品信息，它除了存储供应商ID（vendors表的主键）外不存储其他供应商信息。vendors表的主键又叫作products表的外键，它将vendors表与products表关联，利用供应商ID能从vendors表中找出相应供应商的详细信息。

外键（foreign key） 外键为某个表中的一列，它包含另一个表的主键值，定义了两个表之间的关系。

124

这样做的好处如下：

- 供应商信息不重复，从而不浪费时间和空间；
- 如果供应商信息变动，可以只更新vendors表中的单个记录，相关表中的数据不用改动；
- 由于数据无重复，显然数据是一致的，这使得处理数据更简单。

总之，关系数据可以有效地存储和方便地处理。因此，关系数据库的可伸缩性远比非关系数据库要好。

可伸缩性（scale） 能够适应不断增加的工作量而不失败。设计良好的数据库或应用程序称之为可伸缩性好（scale well）。

14.1.2 为什么要使用联结

正如所述，分解数据为多个表能更有效地存储，更方便地处理，并

且具有更大的可伸缩性。但这些好处是有代价的。

如果数据存储在多个表中，怎样用单条SELECT语句检索出数据？

答案是使用联结。简单地说，联结是一种机制，用来在一条SELECT语句中关联表，因此称之为联结。使用特殊的语法，可以联结多个表返回一组输出，联结在运行时关联表中正确的行。

125

维护引用完整性　重要的是，要理解联结不是物理实体。换句话说，它在实际的数据库表中不存在。联结由SQL Server根据需要建立，它存在于查询的执行当中。

在使用关系表时，仅在关系列中插入合法的数据非常重要。回到这里的例子，如果在products表中插入拥有非法供应商ID（即没有在vendors表中出现）的供应商生产的产品，则这些产品是不可访问的，因为它们没有关联到某个供应商。

为防止这种情况发生，可指示SQL Server只允许在products表的供应商ID列中出现合法值（即出现在vendors表中的供应商）。这就是维护引用完整性，它是通过在表的定义中指定主键和外键来实现的。（这将在第20章介绍。）

14.2　创建联结

联结的创建非常简单，规定要联结的所有表以及它们如何关联即可。请看下面的例子：

输入

```
SELECT vend_name, prod_name, prod_price
FROM vendors, products
WHERE vendors.vend_id = products.vend_id
ORDER BY vend_name, prod_name;
```

输出

```
vend_name          prod_name                  prod_price
---------------- ---------------------------- ----------
ACME               Bird seed                  10.00
ACME               Carrots                    2.50
ACME               Detonator                  13.00
ACME               Safe                       50.00
```

126

```
ACME          Sling          4.49
ACME          TNT (1 stick)  2.50
ACME          TNT (5 sticks) 10.00
Anvils R Us   .5 ton anvil   5.99
Anvils R Us   1 ton anvil    9.99
Anvils R Us   2 ton anvil    14.99
Jet Set       JetPack 1000   35.00
Jet Set       JetPack 2000   55.00
LT Supplies   Fuses          3.42
LT Supplies   Oil can        8.99
```

分析 我们来考察一下此代码。SELECT语句与前面所有语句一样指定要检索的列。这里，最大的差别是所指定的两个列（prod_name和prod_price）在一个表中，而另一个列（vend_name）在另一个表中。

现在来看FROM子句。与以前的SELECT语句不一样，这条语句的FROM子句列出了两个表，分别是vendors和products。它们就是这条SELECT语句联结的两个表的名字。这两个表用WHERE子句正确联结，WHERE子句指示SQL Server匹配vendors表中的vend_id和products表中的vend_id。

可以看到要匹配的两个列以vendors.vend_id和products.vend_id指定。这里需要这种完全限定列名，因为如果只给出vend_id，则SQL Server不知道指的是哪一个（它们有两个，每个表中一个）。

完全限定列名 在引用的列可能出现二义性时，必须使用完全限定列名（用一个句点分隔的表名和列名）。如果引用一个没有用表名限制的具有二义性的列名，SQL Server将返回错误。

127

14.2.1 WHERE子句的重要性

利用WHERE子句建立联结关系似乎有点奇怪，但实际上，有一个很充分的理由。请记住，在一条SELECT语句中联结几个表时，相应的关系是在运行中构造的。在数据库表的定义中不存在能指示SQL Server如何对表进行联结的东西。你必须自己做这件事情。在联结两个表时，你实际上做的是将第一个表中的每一行与第二个表中的每一行配对。WHERE子句作

为过滤条件，它只包含那些匹配给定条件（这里是联结条件）的行。没有WHERE子句，第一个表中的每个行将与第二个表中的每个行配对，而不管它们逻辑上是否可以配在一起。

笛卡儿积（cartesian product）　由没有联结条件的表关系返回的结果为笛卡儿积。检索出的行的数目将是第一个表中的行数乘以第二个表中的行数。

为理解这一点，请看下面的SELECT语句及其输出：

输入

```
SELECT vend_name, prod_name, prod_price
FROM vendors, products
ORDER BY vend_name, prod_name;
```

输出

```
vend_name          prod_name                  prod_price
---------------- ------------------------- ---------
ACME               .5 ton anvil               5.99
ACME               1 ton anvil                9.99
ACME               2 ton anvil                14.99
ACME               Bird seed                  10.00
ACME               Carrots                    2.50
ACME               Detonator                  13.00
ACME               Fuses                      3.42
ACME               JetPack 1000               35.00
ACME               JetPack 2000               55.00
ACME               Oil can                    8.99
ACME               Safe                       50.00
ACME               Sling                      4.49
ACME               TNT (1 stick)              2.50
ACME               TNT (5 sticks)             10.00
Anvils R Us        .5 ton anvil               5.99
Anvils R Us        1 ton anvil                9.99
Anvils R Us        2 ton anvil                14.99
Anvils R Us        Bird seed                  10.00
Anvils R Us        Carrots                    2.50
Anvils R Us        Detonator                  13.00
Anvils R Us        Fuses                      3.42
Anvils R Us        JetPack 1000               35.00
Anvils R Us        JetPack 2000               55.00
Anvils R Us        Oil can                    8.99
Anvils R Us        Safe                       50.00
Anvils R Us        Sling                      4.49
Anvils R Us        TNT (1 stick)              2.50
Anvils R Us        TNT (5 sticks)             10.00
Furball Inc.       .5 ton anvil               5.99
```

128

Furball Inc.	1 ton anvil	9.99
Furball Inc.	2 ton anvil	14.99
Furball Inc.	Bird seed	10.00
Furball Inc.	Carrots	2.50
Furball Inc.	Detonator	13.00
Furball Inc.	Fuses	3.42
Furball Inc.	JetPack 1000	35.00
Furball Inc.	JetPack 2000	55.00
Furball Inc.	Oil can	8.99
Furball Inc.	Safe	50.00
Furball Inc.	Sling	4.49
Furball Inc.	TNT (1 stick)	2.50
Furball Inc.	TNT (5 sticks)	10.00
Jet Set	.5 ton anvil	5.99
Jet Set	1 ton anvil	9.99
Jet Set	2 ton anvil	14.99
Jet Set	Bird seed	10.00
Jet Set	Carrots	2.50
Jet Set	Detonator	13.00
Jet Set	Fuses	3.42
Jet Set	JetPack 1000	35.00
Jet Set	JetPack 2000	55.00
Jet Set	Oil can	8.99
Jet Set	Safe	50.00
Jet Set	Sling	4.49
Jet Set	TNT (1 stick)	2.50
Jet Set	TNT (5 sticks)	10.00
Jouets Et Ours	.5 ton anvil	5.99
Jouets Et Ours	1 ton anvil	9.99
Jouets Et Ours	2 ton anvil	14.99
Jouets Et Ours	Bird seed	10.00
Jouets Et Ours	Carrots	2.50
Jouets Et Ours	Detonator	13.00
Jouets Et Ours	Fuses	3.42
Jouets Et Ours	JetPack 1000	35.00
Jouets Et Ours	JetPack 2000	55.00
Jouets Et Ours	Oil can	8.99
Jouets Et Ours	Safe	50.00
Jouets Et Ours	Sling	4.49
Jouets Et Ours	TNT (1 stick)	2.50
Jouets Et Ours	TNT (5 sticks)	10.00
LT Supplies	.5 ton anvil	5.99
LT Supplies	1 ton anvil	9.99
LT Supplies	2 ton anvil	14.99
LT Supplies	Bird seed	10.00
LT Supplies	Carrots	2.50
LT Supplies	Detonator	13.00

129

```
LT Supplies     Fuses                3.42
LT Supplies     JetPack 1000         35.00
LT Supplies     JetPack 2000         55.00
LT Supplies     Oil can              8.99
LT Supplies     Safe                 50.00
LT Supplies     Sling                4.49
LT Supplies     TNT (1 stick)        2.50
LT Supplies     TNT (5 sticks)       10.00
```

分析 从上面的输出中可以看到，相应的笛卡儿积不是我们所想要的。这里返回的数据用每个供应商匹配了每个产品，它包括了供应商不正确的产品。实际上有的供应商根本就没有产品。

不要忘了WHERE子句　应该保证所有联结都有WHERE子句，否则SQL Server将返回比想要的数据多得多的数据。同理，应该保证WHERE子句的正确性。不正确的过滤条件将导致SQL Server返回不正确的数据。

叉联结　有时我们会听到返回称为叉联结（cross join）的笛卡儿积的联结类型。

在第13章采用两种方案得到2005年9月订购产品的客户列表，这两种方案都使用了子查询（一个使用IN，另一个使用EXISTS）。下面是第三种方案，这次使用内部联结。

输入

```
SELECT customers.cust_id, customers.cust_name
FROM customers, orders
WHERE DateDiff(month, order_date, '2005-09-01') = 0
 AND customers.cust_id = orders.cust_id;
```

输出

```
cust_id     cust_name
----------- ----------------
10001       Coyote Inc.
10003       Wascals
10004       Yosemite Place
```

14.2.2　内部联结

目前为止所用的联结称为等值联结（equijoin），它基于两个表之间的相等测试。这种联结也称为内部联结。其实，对于这种联结可以使用稍

微不同的语法来明确指定联结的类型。下面的SELECT语句返回与前面例子完全相同的数据：

输入

```
SELECT vend_name, prod_name, prod_price
FROM vendors INNER JOIN products
 ON vendors.vend_id = products.vend_id;
```

131

分析 此语句中的SELECT与前面的SELECT语句相同，但FROM子句不同。这里，两个表之间的关系是FROM子句的组成部分，以INNER JOIN指定。在使用这种语法时，联结条件用特定的ON子句而不是WHERE子句给出。传递给ON的实际条件与传递给WHERE的相同。

输出排序 WHERE语法联结和INNER JOIN语法联结返回完全相同的结果。但是，可以看到两种形式的联结以不同顺序返回结果。当然，如果指定一个ORDER BY子句，则不管使用何种语法，数据都将按规定排序。

使用哪种语法 ANSI SQL规范首选INNER JOIN语法。此外，尽管使用WHERE子句定义联结的确比较简单，但是使用明确的联结语法能够确保不会忘记联结条件，有时候这样做也能影响性能。

14.2.3 联结多个表

SQL对一条SELECT语句中可以联结的表的数目没有限制。创建联结的基本规则也相同。首先列出所有表，然后定义表之间的关系。例如：

输入

```
SELECT prod_name, vend_name, prod_price, quantity
FROM orderitems, products, vendors
WHERE products.vend_id = vendors.vend_id
  AND orderitems.prod_id = products.prod_id
  AND order_num = 20005;
```

132

输出

```
prod_name           vend_name         prod_price   quantity
------------------- ----------------- ------------ --------
.5 ton anvil        Anvils R Us       5.99         10
1 ton anvil         Anvils R Us       9.99         3
TNT (5 sticks)      ACME              10.00        5
Bird seed           ACME              10.00        1
```

分析　此例子显示编号为20005的订单中的物品。订单物品存储在orderitems表中。每个产品按其产品ID存储，它引用products表中的产品。这些产品通过供应商ID联结到vendors表中相应的供应商，供应商ID存储在每个产品的记录中。这里的FROM子句列出了3个表，而WHERE子句定义了这两个联结条件，而第三个联结条件用来过滤出订单20005中的物品。

性能考虑　SQL Server在运行时关联指定的每个表以处理联结。这种处理可能是非常耗费资源的，因此应该仔细，不要联结不必要的表。联结的表越多，性能下降越厉害。

通过为外键列有效地创建索引可以明显地改善这种性能下降。

现在可以回顾一下第13章中的例子了。该例子如下所示，其SELECT语句返回订购产品TNT2的客户列表：

输入

```
SELECT cust_name, cust_contact
FROM customers
WHERE cust_id IN (SELECT cust_id
                  FROM orders
                  WHERE order_num IN (SELECT order_num
                                      FROM orderitems
                                      WHERE prod_id = 'TNT2'));
```

133

正如第13章所述，子查询并不总是执行复杂SELECT操作的最有效的方法，下面是使用联结的相同查询：

输入

```
SELECT cust_name, cust_contact
FROM customers, orders, orderitems
WHERE customers.cust_id = orders.cust_id
  AND orderitems.order_num = orders.order_num
  AND prod_id = 'TNT2';
```

输出

```
cust_nam             cust_contact
-------------------- --------------
Coyote Inc.          Y Lee
Yosemite Place       Y Sam
```

分析 正如第13章所述，这个查询中返回数据需要使用3个表。但这里我们没有在嵌套子查询中使用它们，而是使用了两个联结。这里有3个WHERE子句条件。前两个关联联结中的表，后一个过滤产品TNT2的数据。

多做实验 正如所见，为执行任一给定的SQL操作，一般存在不止一种方法。很少有绝对正确或绝对错误的方法。性能可能会受操作类型、表中数据量、是否存在索引或键以及其他一些条件的影响。因此，有必要对不同的选择机制进行实验，以找出最适合具体情况的方法。

134

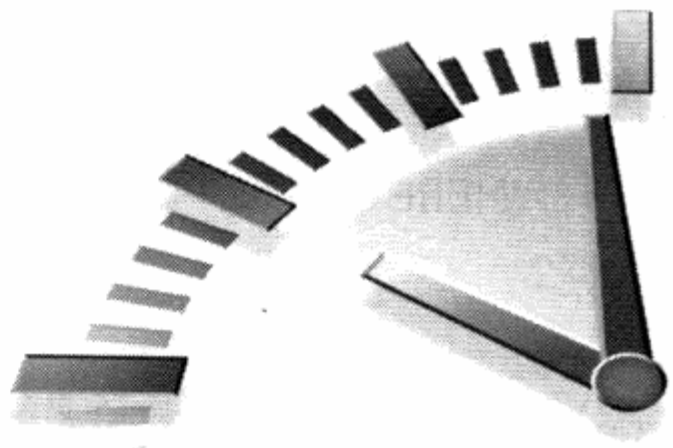
14.3 小结

联结是SQL中最重要最强大的特性，有效地使用联结需要对关系数据库设计有基本的了解。本章随着对联结的介绍讲述了关系数据库设计的一些基本知识，包括等值联结（也称为内部联结）这种最经常使用的联结形式。下一章将介绍如何创建其他类型的联结。

135

第 15 章

创建高级联结



本章将讲解另外一些联结类型（包括它们的含义和使用方法），介绍如何对被联结的表使用表别名和聚集函数。

15.1 使用表别名

第9章中介绍了如何使用别名引用被检索的表列。给列起别名的语法如下：

输入

```
SELECT RTrim(vend_name) + ' (' + RTrim(vend_country) + ')' AS
vend_title
FROM vendors
ORDER BY vend_name;
```

别名除了用于列名和计算字段外，SQL还允许给表名起别名。这样做有两个主要理由：

- 缩短SQL语句；
- 允许在单条SELECT语句中多次使用相同的表。

请看下面的SELECT语句。它与前一章的例子中所用的语句基本相同，但改成了使用别名：

输入

```
SELECT cust_name, cust_contact
FROM customers AS c, orders AS o, orderitems AS oi
WHERE c.cust_id = o.cust_id
  AND oi.order_num = o.order_num
  AND prod_id = 'TNT2';
```

分析 可以看到，FROM子句中3个表全都具有别名。customers AS c建立c作为customers的别名，等等。这使得能使用省写的c

而不是全名customers。在此例子中，表别名只用于WHERE子句。但是，表别名不仅能用于WHERE子句，它还可以用于SELECT的列表、ORDER BY子句以及语句的其他部分。

只在执行时使用 应该注意，表别名只在查询执行中使用。与列别名不一样，表别名不返回到客户机。

15.2 使用不同类型的联结

迄今为止，我们使用的只是称为内部联结或等值联结（equijoin）的简单联结。现在来看3种其他联结，它们分别是自联结、自然联结和外部联结。

15.2.1 自联结

如前所述，使用表别名的主要原因之一是能在单条SELECT语句中不止一次引用相同的表。下面举一个例子。

假如你发现某物品（其ID为DTNTR）存在问题，因此想知道生产该物品的供应商生产的其他物品是否也存在这些问题。此查询要求首先找到生产ID为DTNTR的物品的供应商，然后找出这个供应商生产的其他物品。下面是解决此问题的一种方法：

输入

```
SELECT prod_id, prod_name
FROM products
WHERE vend_id = (SELECT vend_id
                 FROM products
                 WHERE prod_id = 'DTNTR');
```

输出

```
prod_id    prod_name
---------- --------------------
DTNTR      Detonator
FB         Bird seed
FC         Carrots
SAFE       Safe
SLING      Sling
TNT1       TNT (1 stick)
TNT2       TNT (5 sticks)
```

分析　这是第一种解决方案，它使用了子查询。内部的SELECT语句做了一个简单的检索，返回生产ID为DTNTR的物品的供应商的vend_id。该ID用于外部查询的WHERE子句中，以便检索出这个供应商生产的所有物品（第13章中讲授了子查询的所有内容。更多信息请参阅该章）。

现在来看使用联结的相同查询：

输入

```
SELECT p1.prod_id, p1.prod_name
FROM products AS p1, products AS p2
WHERE p1.vend_id = p2.vend_id
  AND p2.prod_id = 'DTNTR';
```

139

输出

```
prod_id    prod_name
---------- --------------------
DTNTR      Detonator
FB         Bird seed
FC         Carrots
SAFE       Safe
SLING      Sling
TNT1       TNT (1 stick)
TNT2       TNT (5 sticks)
```

分析　此查询中需要的两个表实际上是相同的表，因此products表在FROM子句中出现了两次。虽然这是完全合法的，但对products的引用具有二义性，因为SQL Server不知道你引用的是products表中的哪个实例。

为解决此问题，使用了表别名。products的第一次出现为别名p1，第二次出现为别名p2。现在可以将这些别名用作表名。例如，SELECT语句使用p1前缀明确地给出所需列的全名。如果不这样，SQL Server将返回错误，因为分别存在两个名为prod_id、prod_name的列。SQL Server不知道想要的是哪一个列（即使它们事实上是同一个列）。WHERE（通过匹配p1中的vend_id和p2中的vend_id）首先联结两个表，然后按第二个表中的prod_id过滤数据，返回所需的数据。

用自联结而不用子查询　自联结通常作为外部语句用来替代从相同表中检索数据时使用的子查询语句。虽然最终的结果是相同的，但有时候处理联结远比处理子查询快得多。应该试一下两种方法，以确定哪一种的性能更好。

140

15.2.2 自然联结

无论何时对表进行联结，应该至少有一个列出现在不止一个表中（被联结的列）。标准的联结（前一章中介绍的内部联结）返回所有数据，甚至相同的列多次出现。自然联结排除多次出现，使每个列只返回一次。

怎样完成这项工作呢？答案是，系统不完成这项工作，由你自己完成它。自然联结是这样一种联结，其中你只能选择那些唯一的列。这一般是通过对表使用通配符（SELECT *），对所有其他表的列使用明确的子集来完成的。下面举一个例子：

输入

```
SELECT c.*, o.order_num, o.order_date,
       oi.prod_id, oi.quantity, OI.item_price
FROM customers AS c, orders AS o, orderitems AS oi
WHERE c.cust_id = o.cust_id
  AND oi.order_num = o.order_num
  AND prod_id = 'FB';
```

分析 在这个例子中，通配符只对第一个表使用。所有其他列明确列出，所以没有重复的列被检索出来。

事实上，迄今为止我们建立的每个内部联结都是自然联结，很可能我们永远都不会用到不是自然联结的内部联结。

15.2.3 外部联结

许多联结将一个表中的行与另一个表中的行相关联。但有时候会需要包含没有关联行的那些行。例如，可能需要使用联结来完成以下工作：

- 对每个客户下了多少订单进行计数，包括那些至今尚未下订单的客户；
- 列出所有产品以及订购数量，包括没有人订购的产品；
- 计算平均销售规模，包括那些至今尚未下订单的客户。

141

在上述例子中，联结包含了那些在相关表中没有关联行的行。这种类型的联结称为外部联结。

下面的SELECT语句给出一个简单的内部联结。它检索所有客户及其订单：

输入

```
SELECT customers.cust_id, orders.order_num
FROM customers INNER JOIN orders
 ON customers.cust_id = orders.cust_id;
```

外部联结语法类似。为了检索所有客户，包括那些没有订单的客户，可如下进行：

输入

```
SELECT customers.cust_id, orders.order_num
FROM customers LEFT OUTER JOIN orders
 ON customers.cust_id = orders.cust_id;
```

输出

```
cust_id     order_num
----------- -----------
10001       20005
10001       20009
10002       NULL
10003       20006
10004       20007
10005       20008
```

142

分析 类似于上一章中所看到的内部联结，这条SELECT语句使用了关键字OUTER JOIN来指定联结的类型（而不是在WHERE子句中指定）。但是，与内部联结关联两个表中的行不同的是，外部联结还包括没有关联行的行。在使用OUTER JOIN语法时，必须使用RIGHT或LEFT关键字指定包括其所有行的表（RIGHT指出的是OUTER JOIN右边的表，而LEFT指出的是OUTER JOIN左边的表）。上面的例子使用LEFT OUTER JOIN从FROM子句左边的表（customers表）中选择所有行。为了从右边的表中选择所有行，应该使用RIGHT OUTER JOIN，如下例所示：

输入

```
SELECT customers.cust_id, orders.order_num
FROM customers RIGHT OUTER JOIN orders
 ON orders.cust_id = customers.cust_id;
```

外部联结的类型 存在两种基本的外部联结形式：左外部联结和右外部联结。它们之间的唯一差别是所关联的表的顺序不同。换句话说，左外部联结可通过颠倒FROM或WHERE子句中表的顺序转换为右外部联结。因此，两种类型的外部联结可互换使用，而究竟使用哪一种纯粹是根据方便而定。

143

非ANSI的外部联结 在前一章中，我们学习了编写内部联结的两种方法：使用简单的WHERE子句和使用INNER JOIN语法。在本章中，我们看到了ANSI风格的OUTER JOIN语法而不是简化的WHERE子句外部联结。

对于使用WHERE子句的外部联结，确实存在一个简化的语法。这里提供一个例子，使你在遇到时知道它是什么（而且也让你知道为什么应该避免使用这种语法）。

下面是这个简化的外部联结：

```
SELECT customers.cust_id, orders.order_num
FROM customers, orders
WHERE customers.cust_id *= orders.cust_id;
```

*=指示SQL Server从第一个表（customers，靠近*的表）中检索所有行，并且关联到第二个表（orders，靠近=的表）的行。因此，*=创建一个左外部联结。类似，=*将创建一个右外部联结（因为*在右边）。

上述是一个较简单的语法。但这种形式的语法不是ANSI标准的成分，在未来的SQL Server版本中并不支持它。SQL Server 6.x、SQL Server 7和SQL Server 2000支持它，而且SQL Server 2005也可以支持（默认为禁用，但可以用sp_dbcmptlevel启用向后兼容来启用），但以后的版本将不再支持。

因此，为免除以后的兼容性问题，应该避免使用外部联结的这种简化的WHERE形式。这对于简化的内部联结语法（前一章所示）不适用，微软还没有表露出不支持它的意图。

还有另一种形式的外部联结需要注意，虽然很少会有人使用它。FULL OUTER JOIN用来从两个表中检索相关的行，以及从每个表中检索不相关的行（这些行对另一表的非选择列具有NULL值）。显然，用RIGHT和LEFT替换FULL后，FULL OUTER JOIN的语法与前面所示的外部联结相同。 144

15.3 使用带聚集函数的联结

正如第11章所述，聚集函数用来汇总数据。虽然至今为止聚集函数

的所有例子只是从单个表汇总数据，但这些函数也可以与联结一起使用。

为说明这一点，请看一个例子。如果要检索所有客户及每个客户所下的订单数，下面使用了Count()函数的代码可完成此工作：

输入

```
SELECT customers.cust_name,
       customers.cust_id,
       Count(orders.order_num) AS num_ord
FROM customers INNER JOIN orders
 ON customers.cust_id = orders.cust_id
GROUP BY customers.cust_name,
         customers.cust_id;
```

输出

```
cust_name                 cust_id     num_ord
------------------------- ----------- -----------
Coyote Inc.               10001       2
Wascals                   10003       1
Yosemite Place            10004       1
E Fudd                    10005       1
```

分析　此SELECT语句使用INNER JOIN将customers和orders表互相关联。GROUP BY子句按客户分组数据，因此，函数调用Count(orders.order_num)对每个客户的订单计数，将它作为num_ord返回。

145　聚集函数也可以方便地与其他联结一起使用。请看下面的例子：

输入

```
SELECT customers.cust_name,
       customers.cust_id,
       Count(orders.order_num) AS num_ord
FROM customers LEFT OUTER JOIN orders
 ON customers.cust_id = orders.cust_id
GROUP BY customers.cust_name,
         customers.cust_id;
```

输出

```
cust_name                 cust_id     num_ord
------------------------- ----------- -----------
Coyote Inc.               10001       2
Mouse House               10002       0
Wascals                   10003       1
Yosemite Place            10004       1
E Fudd                    10005       1
```

分析　这个例子使用左外部联结来包含所有客户，甚至包含那些没有下任何订单的客户。结果显示也包含了客户Mouse House，它有0个订单。

NULL值消除警告 依赖于所使用的数据库客户机，你可能会看到前面T-SQL语句生成的警告：Warning: Null value is eliminated by an aggregate or other SET operation。

这不是一条错误消息，它只是一个提示性的警告，告诉你有一行（这里为Mouse House）返回了NULL，因为没有订单。但是，因为使用了聚集函数（Count()函数），该NULL被转换为一个数（这里的0）。

146

15.4 使用联结和联结条件

在总结关于联结的这两章前，有必要汇总一下关于联结及其使用的某些要点。

- 注意所使用的联结类型。一般我们使用内部联结，但使用外部联结也是有效的。
- 保证使用正确的联结条件，否则将返回不正确的数据。
- 应该总是提供联结条件，否则会得出笛卡儿积。
- 在一个联结中可以包含多个表，甚至对于每个联结可以采用不同的联结类型。虽然这样做是合法的，一般也很有用，但应该在一起测试它们前，分别测试每个联结。这将使故障排除更为简单。

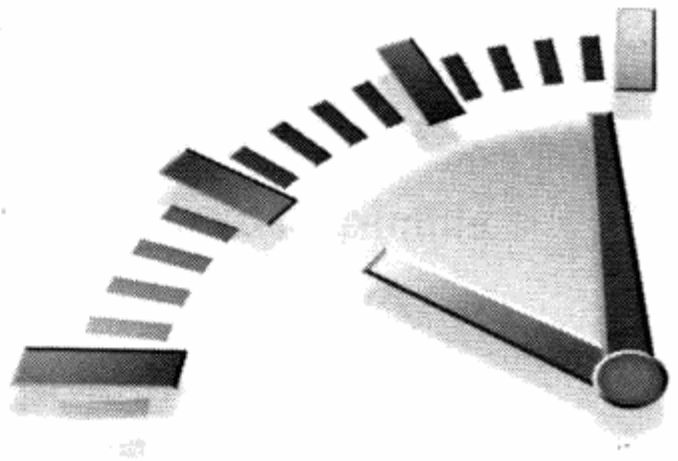
15.5 小结

本章是上一章关于联结的继续。本章从讲授如何以及为什么要使用别名开始，然后讨论不同的联结类型及对每种类型的联结使用的各种语法形式。我们还介绍了如何与联结一起使用聚集函数，以及在使用联结时应该注意的某些问题。

147

第16章

组合查询



本章讲述如何利用UNION操作符将多条SELECT语句组合成一个结果集。

16.1 组合查询

多数SQL查询都只包含从一个或多个表中返回数据的单条SELECT语句。T-SQL也允许执行多个查询（多条SELECT语句），并将结果作为单个查询结果集返回。这些组合查询通常称为并（union）或复合查询（compound query）。

有两种基本情况，其中需要使用组合查询：

- 在单个查询中从不同的表返回类似结构的数据；
- 对单个表执行多个查询，按单个查询返回数据。

组合查询和多个WHERE条件 多数情况下，组合相同表的两个查询完成的工作与具有多个WHERE子句条件的单条查询完成的工作相同。换句话说，任何具有多个WHERE子句的SELECT语句都可以作为一个组合查询给出，在以下段落中可以看到这一点。这两种技术在不同的查询中性能也不同。因此，应该试一下这两种技术，以确定对特定的查询哪一种性能更好。

16.2 创建组合查询

可用UNION操作符来组合数条SQL查询。利用UNION，可给出多条

SELECT语句，将它们的结果组合成单个结果集。

16.2.1 使用UNION

UNION的使用很简单。所需做的只是给出每条SELECT语句，在各条语句之间放上关键字UNION。

举一个例子，假如需要价格小于等于5的所有物品的一个列表，而且还想包括供应商1001和1002生产的所有物品（不考虑价格）。当然，可以利用WHERE子句来完成此工作，不过这次我们将使用UNION。

正如所述，创建UNION涉及编写多条SELECT语句。首先来看单条语句：

输入

```
SELECT vend_id, prod_id, prod_price
FROM products
WHERE prod_price <= 5;
```

输出

```
vend_id     prod_id    prod_price
---------- ---------- ----------
1003        FC         2.50
1002        FU1        3.42
1003        SLING      4.49
1003        TNT1       2.50
```

输入

```
SELECT vend_id, prod_id, prod_price
FROM products
WHERE vend_id IN (1001,1002);
```

150

输出

```
vend_id     prod_id    prod_price
---------- ---------- ----------
1001        ANV01      5.99
1001        ANV02      9.99
1001        ANV03      14.99
1002        FU1        3.42
1002        OL1        8.99
```

分析 第一条SELECT，检索价格不高于5的所有物品所有行。第二条SELECT使用IN找出供应商1001和1002生产的所有物品。

为了组合这两条语句，按如下进行：

输入

```
SELECT vend_id, prod_id, prod_price
FROM products
WHERE prod_price <= 5
UNION
SELECT vend_id, prod_id, prod_price
FROM products
WHERE vend_id IN (1001,1002);
```

输出

```
vend_id     prod_id     prod_price
----------- ---------- ----------
1001        ANV01       5.99
1001        ANV02       9.99
1001        ANV03       14.99
1002        FU1         3.42
1002        OL1         8.99
1003        FC          2.50
1003        SLING       4.49
1003        TNT1        2.50
```

151

分析　这条语句由前面的两条SELECT语句组成，语句中用UNION关键字分隔。UNION指示SQL Server执行两条SELECT语句，并把输出组合成单个查询结果集。

作为参考，这里给出使用多条WHERE子句而不是使用UNION的相同查询：

输入

```
SELECT vend_id, prod_id, prod_price
FROM products
WHERE prod_price <= 5
   OR vend_id IN (1001,1002);
```

在这个简单的例子中，使用UNION可能比使用WHERE子句更为复杂。但对于更复杂的过滤条件，或者从多个表（而不是单个表）中检索数据的情形，使用UNION可能会使处理更简单。

16.2.2　UNION规则

正如所见，并是非常容易使用的。但在进行并时有几条规则需要注意。

- UNION必须由两条或两条以上的SELECT语句组成，语句之间用关键字UNION分隔（因此，如果组合4条SELECT语句，将要使用3个UNION关键字）。
- UNION中的每个查询必须包含相同的列、表达式或聚集函数，而且各个列必须以相同的次序列出（对其他DBMS则没有这种限制，只要各个列都出现，它们以任意次序出现都可以）。
- 列数据类型必须兼容：类型不必完全相同，但必须是SQL Server可以隐含地转换的类型（例如，不同的数值类型或不同的日期

类型)。 152

如果遵守了这些基本规则或限制,则可以将并用于任何数据检索任务。

16.2.3 包含或取消重复的行

请返回到16.2.1节，考察一下所用的样例SELECT语句。我们注意到，在分别执行时，第一条SELECT语句返回4行，第二条SELECT语句返回5行。但在用UNION组合两条SELECT语句后，只返回了8行而不是9行。

UNION从查询结果集中自动去除了重复的行（换句话说，它的行为与单条SELECT语句中使用多个WHERE子句条件一样)。因为供应商1002生产的一种物品的价格也低于5，所以两条SELECT语句都返回该行。在使用UNION时，重复的行被自动取消。

这是UNION的默认行为，但是如果需要，可以改变它。事实上，如果想返回所有匹配行，可使用UNION ALL而不是UNION。

请看下面的例子：

输入

```
SELECT vend_id, prod_id, prod_price
FROM products
WHERE prod_price <= 5
UNION ALL
SELECT vend_id, prod_id, prod_price
FROM products
WHERE vend_id IN (1001,1002);
```

输出

```
vend_id     prod_id    prod_price
----------- ---------- ----------
1003        FC         2.50
1002        FU1        3.42
1003        SLING      4.49
1003        TNT1       2.50
1001        ANV01      5.99
1001        ANV02      9.99
1001        ANV03      14.99
1002        FU1        3.42
1002        OL1        8.99
```

153

分析 使用UNION ALL，SQL Server不取消重复的行。因此这里的例子返回9行，其中有一行出现两次。

UNION与WHERE　本章开始时说过，UNION几乎总是完成与多个WHERE条件相同的工作。UNION ALL为UNION的一种形式，它完成WHERE子句完成不了的工作。如果确实需要每个条件的匹配行全部出现（包括重复行），则必须使用UNION ALL而不是WHERE。

16.2.4　对组合查询结果排序

SELECT语句的输出用ORDER BY子句排序。在用UNION组合查询时，只能使用一条ORDER BY子句，它必须出现在最后一条SELECT语句之后。对于结果集，不存在用一种方式排序一部分，而又用另一种方式排序另一部分的情况，因此不允许使用多条ORDER BY子句。

下面的例子排序前面UNION返回的结果：

输入

```
SELECT vend_id, prod_id, prod_price
FROM products
WHERE prod_price <= 5
UNION
SELECT vend_id, prod_id, prod_price
FROM products
WHERE vend_id IN (1001,1002)
ORDER BY vend_id, prod_price;
```

154

输出

```
vend_id     prod_id    prod_price
----------- ---------- ----------
1001        ANV01      5.99
1001        ANV02      9.99
1001        ANV03      14.99
1002        FU1        3.42
1002        OL1        8.99
1003        FC         2.50
1003        TNT1       2.50
1003        SLING      4.49
```

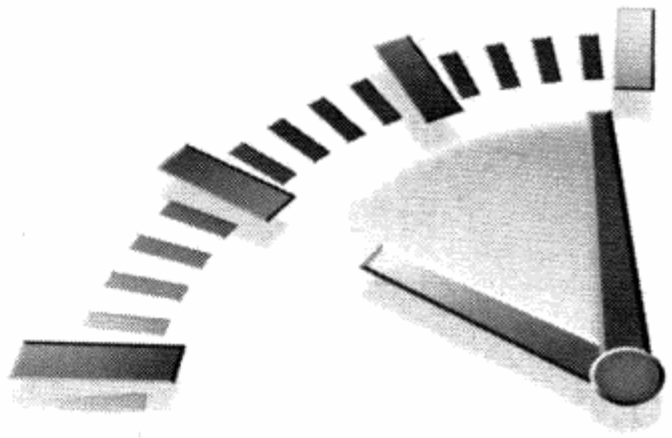
分析　这条UNION在最后一条SELECT语句后使用了ORDER BY子句。虽然ORDER BY子句似乎只是最后一条SELECT语句的组成部分，但实际上SQL Server将用它来排序所有SELECT语句返回的所有结果。

组合不同的表 为使表述比较方便，本章例子中的组合查询使用的均是相同的表。但是其中使用UNION的组合查询可以应用不同的表。

16.3 小结

本章讲授如何用UNION操作符来组合SELECT语句。利用UNION，可把多条查询的结果作为一条组合查询返回，不管它们的结果中包含还是不包含重复。使用UNION可极大地简化复杂的WHERE子句，简化从多个表中检索数据的工作。

第 17 章



全文本搜索

本章将学习如何使用SQL Server的全文本搜索功能进行高级的数据查询和选择。

17.1 理解全文本搜索

仅针对SQL Server 2005 本章介绍使用SQL Server 2005的全文本搜索。以前的SQL Server版本不支持全文本搜索，但SQL Server 2005更改和加强了这个特性，所以本章所介绍的内容不适用于以前的版本。

第8章介绍了LIKE关键字，它利用通配操作符匹配文本（和部分文本）。使用LIKE，能够查找包含特殊值或部分值的行（不管这些值位于列内什么位置）。

虽然这些搜索机制非常有用，但存在几个重要的限制。

- 性能——通配符匹配通常要求SQL Server尝试匹配表中所有行（而且这些搜索极少使用表索引）。因此，由于被搜索行数不断增加，这些搜索可能非常耗时。
- 明确控制——使用通配符匹配，很难（而且并不总是能）明确地控制匹配什么和不匹配什么。例如，指定一个词必须匹配，一个词必须不匹配，而一个词仅在第一个词确实匹配的情况下可以匹配或者可以不匹配。
- 智能化的结果——虽然基于通配符的搜索提供了非常灵活的搜

索，但它们都不能提供一种智能化的选择结果的方法。例如，一个特殊词的搜索将会返回包含该词的所有行，而不区分包含单个匹配的行和包含多个匹配的行（按照可能是更好的匹配来排列它们）。类似，一个特殊词的搜索将不会找出不包含该词但包含其他相关词的行。

所有这些限制以及更多的限制都可以用全文本搜索来解决。在使用全文本搜索时，SQL Server不需要分别查看每个行，不需要分别分析和处理每个词。SQL Server创建指定列中各词的一个索引，搜索可以针对这些词进行。这样，SQL Server可以快速有效地决定哪些词匹配（哪些行包含它们），哪些词不匹配，它们匹配的频率，等等。

17.2 设置全文本搜索

以下是进行全文本搜索的需求列表：

- 必须对相应的数据库启用全文本搜索的支持；
- 必须定义一个目录（保存全文本数据）；
- 必须对要索引的表和列建立全文本索引。

在索引之后，可使用带FREETEXT和CONTAINS谓词的SELECT具体执行搜索。

158

17.2.1 启用全文本搜索支持

在创建数据库后，在能执行全文本操作之前必须启用全文本支持。为启用全文本支持，使用sp_fulltext_database存储过程。此存储过程更新当前选择的数据库，因此应该在执行这条语句前保证使用正确的数据库：

输入

```
EXEC sp_fulltext_database 'enable';
```

分析 sp_fulltext_database接受一个指定启用（enable）还是禁用（disable）全文本支持的参数。

使用New Database对话框　如果你使用交互式的New Database对话框创建数据库，可选择Use Full-Text Indexing复选框，这样会自动执行前面提及的存储过程。

是否启用了全文本？　如果你不知道是否启用了全文本支持，只需运行此存储过程即可。如果全文本支持没有启用，此存储过程将启用它。如果全文本支持已经启用，则存储过程不做任何事件。

17.2.2　创建全文本目录

正如所述，SQL Server在一个目录（需要创建的一个文件）中存储全文本数据。单个目录可用于多个表和索引，因此，可以使用一个现存的目录。如果没有现存的目录，则应该用CREATE FULLTEXT CATALOG创建一个目录：

159

输入

```
CREATE FULLTEXT CATALOG catalog_crashcourse;
```

分析　这条命令在默认目录位置创建了一个名为catalog_crash-course的目录。为了指出具体的文件位置，可以给定IN PATH属性。

17.2.3　创建全文本索引

在创建目录之后，可定义实际的全文本索引。如下所示，用CREATE FULLTEXT INDEX创建索引：

输入

```
CREATE FULLTEXT INDEX ON productnotes(note_text)
KEY INDEX pk_productnotes
ON catalog_crashcourse;
```

分析　这条语句在productnotes表上创建一个全文本索引，索引note_text列。需要唯一标识各行的键，因此用KEY INDEX提供表的主键名pk_productnotes。最后，ON子句指定用来存储全文本数据的目录，这里用刚创建的目录。

如果需要，可以索引不止一列。为此，只需指定各个列名（逗号分隔）即可。

定义默认目录 在用CREATE FULLTEXT CATALOG来创建一个新目录时，可以指定可选的AS DEFAULT子句。这样使新创建的目录默认用于后面创建的全文本索引，这表示CREATE FULLTEXT INDEX中的ON子句可以省略。

160

在创建了全文本索引后，任何现有的表数据都将被索引，而且对于productnotes表执行的任何INSERT、UPDATE和DELETE操作都将使该索引被更新。

不要在导入数据时使用全文本索引 更新索引要花时间，虽然所花时间不多，但总还是要花时间的。而且，更新全文本索引所花时间甚至更长。如果正在把数据导入一个新表，则不应该在此时启用FULLTEXT索引。而是应该先导入所有数据，然后再修改该表，定义FULLTEXT。这使数据导入更快（而且索引所有数据所需的总时间将小于分别索引每行所需的时间之和）。

17.2.4 管理目录和索引

目录和索引可以用ALTER FULLTEXT更新，用DROP FULLTEXT删除。实际上，这两条语句很少使用，只有一个例外。如果目录或索引讹误（返回不一致的结果）或太慢，重建可能有好处。可如下进行：

输入

```
ALTER FULLTEXT CATALOG catalog_crashcourse REBUILD;
```

分析 这条语句删除和重建目录索引，有效地进行一个完全的重新索引。

如果你想了解现有目录和索引的更多细节，可使用一系列的系统视图：

输入

```
SELECT * FROM sys.fulltext_catalogs;
```

161

分析　这条语句返回当前使用目录的信息，包括物理文件位置和是否标记为默认目录。

输入

```
SELECT * FROM sys.fulltext_indexes;
```

分析　这条语句返回定义的索引信息，包括使用目录的ID，索引是否自动更新以及最后更新的开始和结束时间。

FulltextCatalogProperty()函数　可用一个名为FulltextCatalogProperty()的系统函数来获得目录的信息。FulltextCatalogProperty()接受一个目录名和要检查的属性。两个最重的属性为IndexSize和PopulateStatus(后者让你知道属性是否最新，是否当前正在建立等)。

17.3　进行全文本搜索

在数据被索引后，可以用以下两个谓词进行全文本搜索：

- FREETEXT进行简单的搜索，按意思进行匹配（不同于精确文本匹配）；
- CONTAINS进行词或短语的搜索，包括近似词、派生词以及近似词。

162　FREETEXT和CONTAINS都用于SELECT语句的WHERE子句。

17.3.1　使用FREETEXT进行搜索

FREETEXT用来搜索包含可能与指定短语意思相同（或类似）的词或短语的行。

举一个例子，下面是一个简单的LIKE通配符SELECT：

输入

```
SELECT note_id, note_text
FROM productnotes
WHERE note_text LIKE '%rabbit food%';
```

分析　这条语句在列note_text中查找短语rabbit food。没有返回行，因为该短语没有出现在任一行中。

现在进行FREETEXT全文本搜索：

输入

```
SELECT note_id, note_text
FROM productnotes
WHERE FREETEXT(note_text, 'rabbit food');
```

输出

```
note_id     note_text
----------  ----------------------------------------------
104         Quantity varies, sold by the sack load. All
            guaranteed to be bright and orange, and suitable
            for use as rabbit bait.
110         Customer complaint: rabbit has been able to detect
            trap, food apparently less effective now.
```

分析 FREETEXT(note_text, 'rabbit food')意思是“在列note_text上进行FREETEXT查找，寻找可能具有rabbit food含义的任何东西”。检索出两行，一行包含词rabbit和food，但不是联结在一起的短语，另一行包含rabbit和可以推断出food的一个上下文，虽然词food实际上没有出现。 163

可以看出，FREETEXT全文本搜索的使用非常简单。遗憾的是，这种简单性需要付出代价，FREETEXT搜索缺乏全文本搜索可能需要的更高级的控制。这就是为什么要使用另一个名为CONTAINS的谓词的原因。

其他语言的支持 默认时，FREETEXT使用默认的目录语言确定哪些词要索引，哪些词要忽略（如，通常忽略it和the等词，因为它们的出现频率会使结果失真）。为指定某种替代语言，作为第三个参数传递语言名或ID给FULLTEXT()即可。被指定的语言必须是sys.syslanguages系统表中列出的语言之一。可以用以下语句列出这些语言：

```
SELECT * FROM sys.syslanguages;
```

17.3.2 用CONTAINS进行搜索

CONTAINS用来搜索包含词、短语、部分短语、具有相同词干的词、近似搜索、同义词（用辞典搜索）等。

请看一个简单的例子：

输入

```
SELECT note_id, note_text
FROM productnotes
WHERE CONTAINS(note_text, 'handsaw');
```

输出

```
note_id     note_text
----------- ------------------------------------------
112         Customer complaint: Circular hole in safe floor
            can apparently be easily cut with handsaw.
```

164

分析 WHERE CONTAINS(note_text, 'handsaw')的意思为“在列note_text中找出词handsaw”。

CONTAINS或LIKE? WHERE CONTAINS(note_text,'handsaw')在功能上等同于LIKE note_text = '%handsaw%'。但是，CONTAINS搜索通常更快，特别是随着表的大小的增加更是如此。

CONTAINS还支持使用通配符，如下所示：

输入

```
SELECT note_id, note_text
FROM productnotes
WHERE CONTAINS(note_text, '"anvil*"');
```

输出

```
note_id     note_text
----------- ------------------------------------------
108         Multiple customer returns, anvils failing to drop
            fast enough or falling backwards on purchaser.
            Recommend that customer considers using heavier
            anvils.
```

分析 '"anvil*"'表示“匹配任何以anvil开始的词”。注意，与LIKE不一样，全文本搜索使用*（而不是%）作为通配符。通配符可

165 以在串的开始或结束处使用。

注意引号 最后这个例子中的搜索项为'"anvil*"'，它用常见的单引号括住"anvil*"（双引号）。如果你传递简单的文本给CONTAINS，则该文本用单引号括起来。如果传递通配符，则每个搜索短语必须括在外部单引号内的双引号中。不这样很可能会导致搜索不正常。

CONTAINS还支持布尔操作符AND、OR和NOT。下面举几个例子：

输入

```
SELECT note_id, note_text
FROM productnotes
WHERE CONTAINS(note_text, 'safe AND handsaw');
```

输出

```
note_id     note_text
----------- ----------------------------------------------
112         Customer complaint: Circular hole in safe floor
            can apparently be easily cut with handsaw.
```

分析 'safe AND handsaw'表示“只匹配包含safe和handsaw的行”。

输入

```
SELECT note_id, note_text
FROM productnotes
WHERE CONTAINS(note_text, 'rabbit AND NOT food');
```

166

输出

```
note_id     note_text
----------- ----------------------------------------
104         Quantity varies, sold by the sack load. All
            guaranteed to be bright and orange, and
            suitable for use as rabbit bait.
```

分析 'rabbit AND NOT food'表示“只匹配包含词rabbit和不包含词food的行”。

在搜索极长的文本时，如果搜索项在数据中相互接近，则找到匹配的可能性更大。简单的AND搜索匹配文本中任何位置的项，可用NEAR来指示全文本搜索引擎只在项互相接近时匹配它们。下面是一个例子：

输入

```
SELECT note_id, note_text
FROM productnotes
WHERE CONTAINS(note_text, 'detonate NEAR quickly');
```

输出

```
note_id     note_text
----------- ----------------------------------------------
105         Included fuses are short and have been known to
            detonate too quickly for some customers. Longer
            fuses are available (item FU1) and should be
            recommended.
```

分析 'detonate NEAR quickly'表示“只匹配包含相互靠近的词detonate和quickly的行”。

167

有时，你可能想匹配作为相同系列（基于相同词干）成分的词。例如，如果搜索vary，你还想匹配varies。显然，vary*的通配符在这里没有用，使用var*会匹配出太多的假词。这就是词尾变化匹配的用处。

下面是一个例子：

输入

```
SELECT note_id, note_text
FROM productnotes
WHERE CONTAINS(note_text, 'FORMSOF(INFLECTIONAL, vary)');
```

输出

```
note_id    note_text
---------- ----------------------------------------------
104        Quantity varies, sold by the sack load. All
           guaranteed to be bright and orange, and suitable
           for use as rabbit bait.
```

分析 'FORMSOF(INFLECTIONAL, vary)'指示全文本引擎查找与指定词(这里为vary)具有相同词干的词。因此，包含词varies的行被匹配和检索出来。

THESAURUS搜索　FORMSOF()还支持THESAURUS搜索，其中的词可以匹配同义词。为使用此功能，必须首先用词及其同义词填写一个XML辞典文件。

混合搜索类型　为保持简单和清晰，这里的各个例子使用了通配符、布尔操作符、近似搜索或词尾变化搜索（inflectional search）。实际上，你可以根据需要混合使用它们。

168

17.3.3　排序搜索结果

在进行全文本搜索时，全文本引擎使用复杂的算法来寻找你想查找的内容。它也可以给匹配赋予等级值，越接近的匹配，赋予等级值越高。

等级值通过等级函数访问，FULLTEXT搜索用FULLTEXTTABLE()函数排序，而CONTAINS搜索用CONTAINSTABLE()函数排序。这两个函数都以相同的方式使用，两者都接受搜索模式（本章中已经解释过相同的搜索模式）。

下面是一个例子：

输入

```
SELECT f.rank, note_id, note_text
FROM productnotes,
     FREETEXTTABLE(productnotes, note_text, 'rabbit food') f
WHERE productnotes.note_id=f.[key]
ORDER BY rank DESC;
```

输出

```
rank        note_id     note_text
----------  ----------  ------------------------------------
256         110         Customer complaint: rabbit has been
                        able to detect trap, food apparently
                        less effective now.
45          104         Quantity varies, sold by the sack
                        load. All guaranteed to be bright and
                        orange, and suitable for use as
                        rabbit bait.
```

分析 这个例子进行一个FREETEXT类型的搜索，它使用FREETEXTTABLE()函数而不是使用WHERE子句进行过滤，并且提供一个搜索模式，指示全文本引擎匹配包含意思为rabbit和food的词的行。 169
FREETEXTTABLE()返回别名为f的一个表（能在列选择和联结中引用它）。这个表包含名为key的一个列，它匹配被索引的表（这个例子中为productnotes）的主键，和一个名为rank的列，它是被赋予的等级值。第一行的等级为256，因为它是一个较好的匹配（它实际包含两个搜索词），而第二行的等级为45，因为它是一个较差的匹配。

此技术可以用于FULLTEXT和CONTAINS匹配，但如果进行CONTAINS匹配，应该使用函数CONTAINSTABLE()。

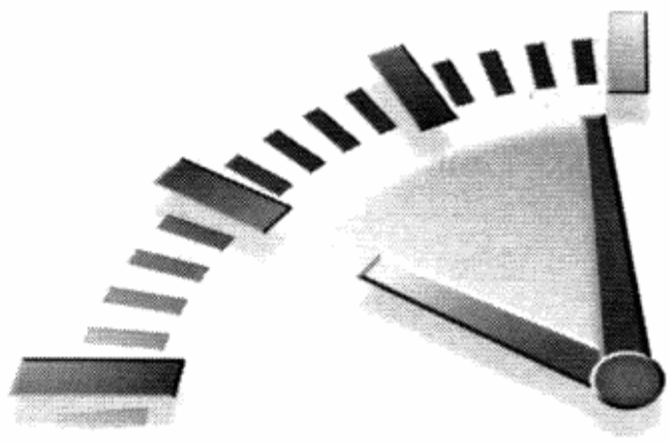
分配搜索项权重 这个例子中分配的等级值假定所有词都是同样重要，而且相关。如果不是这种情况，有的词比另外一些词更重要，则可以用ISABOUT()函数给特定的词赋予权重值。然后，全文本搜索引擎会在决定等级时使用这些权重值。

17.4 小结

本章学习了为什么使用全文本搜索，以及如何使用T-SQL的FREETEXT()和CONTAINS()函数进行这些搜索。我们还学习了如何使用布尔操作符、通配符、近似搜索、词尾变化匹配和搜索排序等内容。 170

第 18 章

插入数据



本章介绍如何利用SQL的INSERT语句将数据插入表中。

18.1 数据插入

毫无疑问，SELECT是最常使用的SQL语句了（这就是为什么前14章讲的都是它的原因）。但是，还有其他3个经常使用的SQL语句需要学习。第一个就是INSERT（下一章介绍另外两个）。

顾名思义，INSERT是用来插入（或添加）行到数据库表的。插入可以用几种方式使用：

- 插入完整的行；
- 插入行的一部分；
- 插入多行；
- 插入某些查询的结果。

下面将介绍这些内容。

插入及系统安全 可针对每个表或每个用户，利用SQL Server的安全机制禁止使用INSERT语句，这将在第29章介绍。

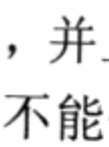
18.2 插入完整的行

把数据插入表中的最简单的方法是使用基本的INSERT语法，它要求

指定表名和被插入到新行中的值。下面举一个例子（但真正上机运行时会失败）：

输入

```
INSERT INTO Customers
VALUES(10006,
   'Pep E. LaPew',
   '100 Main Street',
   'Los Angeles',
   'CA',
   '90046',
   'USA',
   NULL,
   NULL);
```



没有输出（no output） INSERT语句一般不会产生输出。

分析 此例子插入一个新客户到customers表。存储到每个表列中的数据在VALUES子句中给出，对每个列必须提供一个值。如果某个列没有值（如上面的cust_contact和cust_email列），应该使用NULL值（假定表允许对该列指定空值）。各个列必须以它们在表定义中出现的次序填充。

那么，为什么这条INSERT语句会失败？cust_id为一个标识字段（identity field）。这表示其值由SQL Server自动分配，每当增加一个新行时，它自动按顺序使用和保存下一个未使用的数。默认时，标识字段不允许你明确指定值，SQL Server会自己完成此项工作。

172

手动指定标识字段值 如果你确实需要手动指定标识字段的值，则应该首先执行SET IDENTITY_INSERT = ON语句。假如该值的指定是合法的（没有使用过），那么后续的插入将会自动在这个新值上进行增加。

即使你使用的是不同的字段，前面这种形式的INSERT也并不安全，应该尽量避免使用。这种类型的语句高度依赖于表中列的定义次序，并且还依赖于其次序容易获得的信息。即使可得到这种次序信息，也不能保证下一次表结构变动后各个列保持完全相同的次序。因此，编写依赖于特定列次序的SQL语句是很不安全的。如果这样做，有时

难免会出问题。

编写INSERT语句的更安全（不过更烦琐）的方法如下：

输入

```
INSERT INTO customers(cust_name,
   cust_address,
   cust_city,
   cust_state,
   cust_zip,
   cust_country,
   cust_contact,
   cust_email)
VALUES('Pep E. LaPew',
   '100 Main Street',
   'Los Angeles',
   'CA',
   '90046',
   'USA',
   NULL,
   NULL);
```

173

分析　此例子完成与前一个INSERT语句完全相同的工作，但在表名后的括号里明确地给出了列名。在插入行时，SQL Server将用VALUES列表中的相应值填入列表中的对应项。VALUES中的第一个值对应于第一个指定的列名。第二个值对应于第二个列名，如此等等。

因为提供了列名，VALUES必须以其指定的次序匹配指定的列名，不一定按各个列出现在实际表中的次序。其优点是，即使表的结构改变，此INSERT语句仍然能正确工作。你会发现没有指定cust_id，不必要的列可以从列的列表和VALUES列表中简单地省略。

下面的INSERT语句填充所有列（与前面的一样），但以一种不同的次序填充。因为给出了列名，所以插入结果仍然正确：

输入

```
INSERT INTO customers(cust_name,
   cust_contact,
   cust_email,
   cust_address,
   cust_city,
   cust_state,
   cust_zip,
   cust_country)
VALUES('Pep E. LaPew',
   NULL,
   NULL,
```

```
'100 Main Street',
'Los Angeles',
'CA',
'90046',
'USA');
```

174

总是使用列的列表 一般不要使用没有明确给出列的列表的INSERT语句。使用列的列表能使SQL代码继续发挥作用，即使表结构发生了变化。

仔细地给出值 不管使用哪种INSERT语法，都必须给出VALUES的正确数目。如果不提供列名，则必须给每个表列提供一个值。如果提供列名，则必须对每个列出的列给出一个值。如果不这样，将产产生一条错误消息，相应的行插入不成功。

使用这种语法，还可以省略列。这表示可以只给某些列提供值，给其他列不提供值。（事实上你已经看到过这样的例子：当列名被明确列出时，cust_id可以省略。）

省略列 如果表的定义允许，则可以在INSERT操作中省略某些列。省略的列必须满足以下某个条件。

- 该列定义为允许NULL值（无值或空值）。
- 在表定义中给出默认值。这表示如果不给出值，将使用默认值。

如果对表中不允许NULL值且没有默认值的列不给出值，则SQL Server将产生一条错误消息，并且相应的行插入不成功。

INTO是可选的 在T-SQL中关键字INTO是可选的，因此INSERT INTO customers也可以简化为INSERT customers。实际上，为了具有更好的移植性，经常要给出INTO。

175

18.3　插入多行

INSERT插入单个行到一个表中。但是，如果你需要插入多个行怎么办？基本的INSERT语句一次只插入一行，因此需要使用多条INSERT语句。你可以一次性地提交这些插入语句，每条语句用分号分隔，如下所示：

输入

```
INSERT INTO customers(cust_name,
   cust_address,
   cust_city,
   cust_state,
   cust_zip,
   cust_country)
VALUES('Pep E. LaPew',
   '100 Main Street',
   'Los Angeles',
   'CA',
   '90046',
   'USA');
INSERT INTO customers(cust_name,
   cust_address,
   cust_city,
   cust_state,
   cust_zip,
   cust_country)
VALUES('M. Martian',
   '42 Galaxy Way',
   'New York',
   'NY',
   '11213',
   'USA');
```

一次只能插入一行值　与其他DBMS不一样，SQL Server的单条INSERT语句不支持多个VALUES子句。

176

18.4　插入检索出的数据

INSERT一般用来给表插入一个指定列值的行。但是，INSERT还存在另一种形式，可以利用它将一条SELECT语句的结果插入表中。这就是所谓的INSERT SELECT，顾名思义，它是由一条INSERT语句和一条SELECT

语句组成的。

假如你想从另一表中合并客户列表到你的customers表。不需要每次读取一行，然后再将它用INSERT插入，可以如下进行：

新例子的说明 这个例子从一个名为custnew的表中读出数据并插入customers表。为了试验这个例子，应该首先创建和填充custnew表。custnew表的结构与附录B中描述的customers表的相同。在填充custnew时，不应该使用已经在customers中使用过的cust_id值（如果主键值重复，后续的INSERT操作将会失败），或省略这列值，让SQL Server在导入数据的过程中产生新值。

输入

```
INSERT INTO customers(cust_contact,
    cust_email,
    cust_name,
    cust_address,
    cust_city,
    cust_state,
    cust_zip,
    cust_country)
SELECT cust_contact,
    cust_email,
    cust_name,
    cust_address,
    cust_city,
    cust_state,
    cust_zip,
    cust_country
FROM custnew;
```

177

分析 这个例子使用INSERT SELECT从custnew中将所有数据导入customers。SELECT语句从custnew检索出要插入的值，而不是列出它们。SELECT中列出的每个列对应于customers表名后所跟的列表中的每个列。这条语句将插入多少行有赖于custnew表中有多少行。如果这个表为空，则没有行被插入（也不产生错误，因为操作仍然是合法的）。如果这个表确实含有数据，则所有数据将被插入到customers。

INSERT SELECT中的列名　为简单起见，这个例子在INSERT和SELECT语句中使用了相同的列名。但是，不一定要求列名匹配。事实上，SQL Server甚至不关心SELECT返回的列名。它使用的是列的位置，因此SELECT中的第一列（不管其列名）将用来填充表列中指定的第一个列，第二列将用来填充表列中指定的第二个列，如此等等。这对于从使用不同列名的表中导入数据是非常有用的。

INSERT SELECT中SELECT语句可包含WHERE子句以过滤插入的数据。

插入检索数据的另一方法是使用SELECT INTO。SELECT的这个变种允许指定一个目的表，此表将用SELECT语句的结果来填充。

输入

```
SELECT cust_contact,
    cust_email,
    cust_name,
    cust_address,
    cust_city,
    cust_state,
    cust_zip,
    cust_country
INTO customersExport
FROM customers;
```

178

分析　如果使用此方法，INTO子句必须出现在列列表之后、FROM子句之前。INTO指定要创建的一个表名，此表名必须不存在（否则将产生错误）。在此语句执行之后，新创建的表将包含SELECT语句检索出的行。

在你打算创建一个包含从多个表检索出的行时，SELECT INTO非常有用。

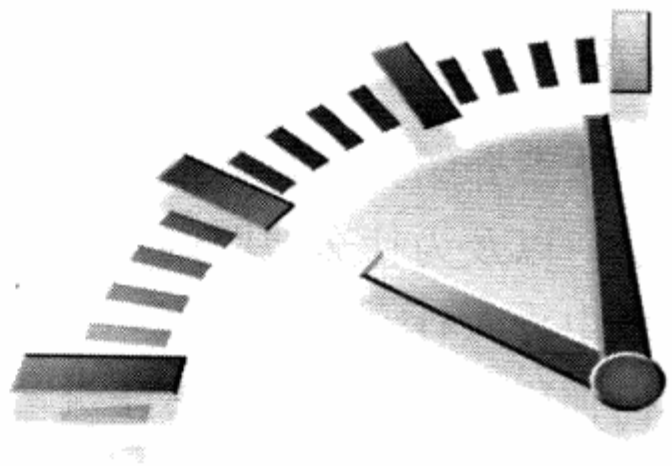
INSERT SELECT与SELECT INTO　可把INSERT SELECT视为一个导入操作，而把SELECT INTO视为一个导出操作。

更多例子　如果想看INSERT用法的更多例子，请参阅附录B中给出的样例表填充脚本，这主要用于创建本书中使用的样例表。

18.5 小结

本章介绍如何将行插入到数据库表。我们学习了使用INSERT的几种方法，以及为什么要明确使用列名，学习了如何用INSERT SELECT从其他表中导入行，如何用SELECT INTO将行导出到一个新表。下一章讲述如何使用UPDATE和DELETE进一步操纵表数据。

第 19 章



更新和删除数据

本章介绍如何利用UPDATE和DELETE语句进一步操纵表数据。

19.1 更新数据

为了更新（修改）表中的数据，可使用UPDATE语句。可采用两种方式使用UPDATE：

- 更新表中特定行；
- 更新表中所有行。

不要省略WHERE子句 在使用UPDATE时一定要注意细心。因为稍不注意，就会更新表中所有行。在使用这条语句前，请完整地阅读本节。

UPDATE与安全 可以限制和控制UPDATE语句的使用，更多内容请参见第29章。

UPDATE语句非常容易使用，甚至可以说是太容易使用了。基本的UPDATE语句由3部分组成，分别是：

- 要更新的表；
- 列名和它们的新值；
- 确定要更新行的过滤条件。

举一个简单例子。客户10005现在有了电子邮件地址，因此他的记录需要更新，语句如下：

输入

```
UPDATE customers
SET cust_email = 'elmer@fudd.com'
WHERE cust_id = 10005;
```

UPDATE语句总是以要更新的表的名字开始。在此例子中，要更新的表的名字为customers。SET命令用来将新值赋给被更新的列。如这里所示，SET子句设置cust_email列为指定的值：

```
SET cust_email = 'elmer@fudd.com'
```

UPDATE语句以WHERE子句结束，它告诉SQL Server更新哪一行。没有WHERE子句，SQL Server将会用这个电子邮件地址更新customers表中所有行，这不是我们所希望的。

更新多个列的语法稍有不同：

输入

```
UPDATE customers
SET cust_name = 'The Fudds',
    cust_email = 'elmer@fudd.com'
WHERE cust_id = 10005;
```

在更新多个列时，只需要使用单个SET命令，每个“列=值”对之间用逗号分隔（最后一列之后不用逗号）。在此例子中，更新客户10005的cust_name和cust_email列。

182

在UPDATE语句中使用子查询 UPDATE语句中可以使用子查询，使得能用SELECT语句检索出的数据更新列数据。关于子查询及使用的更多内容，请参阅第13章。

为了删除某个列的值，可设置它为NULL（假如表定义允许NULL值）。如下进行：

输入

```
UPDATE customers
SET cust_email = NULL
WHERE cust_id = 10005;
```

其中NULL用来去除cust_email列中的值。

19.2　删除数据

为了从一个表中删除（去掉）数据，使用DELETE语句。DELETE有两种用法：

- 从表中删除特定的行；
- 从表中删除所有行。

下面分别对它们进行介绍。

不要省略WHERE子句　在使用DELETE时一定要注意细心。因为稍不注意，就会错误地删除表中所有行。在使用这条语句前，请完整地阅读本节。

183

DELETE与安全　可以限制和控制DELETE语句的使用，更多内容请参见第29章。

前面说过，UPDATE非常容易使用，而DELETE更容易使用。

下面的语句从customers表中删除一行：

输入

```
DELETE FROM customers
WHERE cust_id = 10006;
```

这条语句很容易理解。DELETE FROM要求指定从中删除数据的表名。WHERE子句过滤要删除的行。在这个例子中，只删除客户10006。如果省略WHERE子句，它将删除表中每个客户。

DELETE不需要列名或通配符。DELETE删除整行而不是删除列。为了删除指定的列，请使用UPDATE语句。

删除表的内容而不是表　DELETE语句从表中删除行，甚至是删除表中所有行。但是，DELETE不删除表本身。

19.3 更新和删除的指导原则

前一节中使用的UPDATE和DELETE语句全都具有WHERE子句，这样做的理由很充分。如果省略了WHERE子句，则UPDATE或DELETE将被应用到表中所有的行。换句话说，如果执行UPDATE而不带WHERE子句，则表中每个行都将用新值更新。类似地，如果执行DELETE语句而不带WHERE子句，表的所有数据都将被删除。

184

下面是许多SQL程序员使用UPDATE或DELETE时所遵循的习惯。

- 除非确实打算更新和删除每一行，否则绝对不要使用不带WHERE子句的UPDATE或DELETE语句。
- 保证每个表都有主键（如果忘记这个内容，请参阅第14章），尽可能像WHERE子句那样使用它（可以指定各主键、多个值或值的范围）。
- 在对UPDATE或DELETE语句使用WHERE子句前，应该先用SELECT进行测试，保证它过滤的是正确的记录，以防编写的WHERE子句不正确。
- 使用强制实施引用完整性的数据库（关于这个内容，请参阅第14章），这样SQL Server将不允许删除具有与其他表相关联的数据的行。

小心使用 SQL Server没有撤销（undo）按钮。应该非常小心地使用UPDATE和DELETE，否则你会发现自己更新或删除了错误的数据。

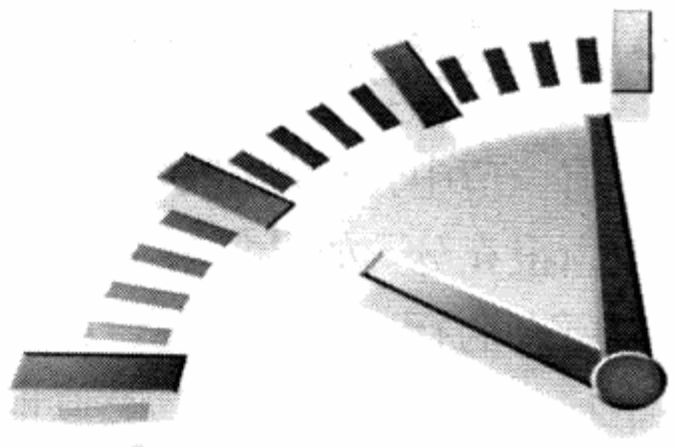
19.4 小结

我们在本章中学习了如何使用UPDATE和DELETE语句处理表中的数据。我们学习了这些语句的语法，知道了它们固有的危险性。本章中还讲解了为什么WHERE子句对UPDATE和DELETE语句很重要，并且给出了应该遵循的一些指导原则，以保证数据的安全。

185

第20章

创建和操纵表



本章讲授表的创建、更改和删除的基本知识。

20.1 创建表

T-SQL不仅用于表数据操纵，而且还可以用来执行数据库和表的所有操作，包括表本身的创建和处理。

一般有两种创建表的方法：

- 使用具有交互式创建和管理表的工具（如第2章讨论的工具）；
- 表也可以直接用SQL语句操纵。

为了用程序创建表，可使用SQL的CREATE TABLE语句。值得注意的是，在使用交互式工具时，实际上使用的是T-SQL语句。但是，这些语句不是用户编写的，界面工具会自动生成并执行相应的T-SQL语句（更改现有表时也是这样）。

另外的例子 关于表创建脚本的另外例子，请参阅本书中用来创建样例表的代码。

20.1.1 表创建基础

为利用CREATE TABLE创建表，必须给出下列信息：

- 新表的名字，在关键字CREATE TABLE之后给出；
- 表列的名字和定义，用逗号分隔。

CREATE TABLE语句也可能会包括其他关键字或选项，但至少要包括表的名字和列的细节。下面的T-SQL语句创建本书中所用的customers表：

输入

```
CREATE TABLE customers
(
  cust_id      INT         NOT NULL IDENTITY(1,1),
  cust_name    NCHAR(50)   NOT NULL ,
  cust_address NCHAR(50)   NULL ,
  cust_city    NCHAR(50)   NULL ,
  cust_state   NCHAR(5)    NULL ,
  cust_zip     NCHAR(10)   NULL ,
  cust_country NCHAR(50)   NULL ,
  cust_contact NCHAR(50)   NULL ,
  cust_email   NCHAR(255) NULL ,
  PRIMARY KEY (cust_id)
);
```

分析 从上面的例子中可以看到，表名紧跟在CREATE TABLE关键字后面。实际的表定义（所有列）括在圆括号之中。各列之间用逗号分隔。这个表由9列组成。每列的定义以列名（它在表中必须是唯一的）开始，后跟列的数据类型（关于数据类型的解释，请参阅第1章。此外，附录D列出了SQL Server支持的数据类型）。表的主键可以在创建表时用PRIMARY KEY关键字指定，这里，指定cust_id列作为主键列。整条语句由右圆括号后的分号结束（现在先忽略IDENTITY关键字，后面会对它进行介绍）。

188

语句格式化 可回忆一下，以前说过T-SQL语句中忽略空格。语句可以在一个长行上输入，也可以分成许多行。它们的作用都相同。这允许你以最适合自己的方式安排语句的格式。前面的CREATE TABLE语句就是语句格式化的一个很好的例子，它被安排在多个行上，其中的列定义进行了恰当的缩进，以便阅读和编辑。以何种缩进格式安排T-SQL语句没有规定，但我强烈推荐采用某种缩进格式。

处理现有的表 在创建新表时，指定的表名必须不存在，否则将出错。如果要防止意外覆盖已有的表，SQL Server要求首先手工删除该表（请参阅后面的段落），然后再重建它，而不是简单地用创建表语句覆盖它。

20.1.2　使用NULL值

第6章中说过，NULL值就是没有值或缺值。允许NULL值的列也允许在插入行时不给出该列的值。不允许NULL值的列不接受该列没有值的行，换句话说，在插入或更新行时，该列必须有值。

每个表列或者是NULL列，或者是NOT NULL列，这种状态在创建时由
189 表的定义规定。请看下面的例子：

输入

```
CREATE TABLE orders
(
  order_num  INT      NOT NULL IDENTITY(1,1),
  order_date DATETIME NOT NULL ,
  cust_id    INT      NOT NULL ,
  PRIMARY KEY (order_num)
);
```

分析　这条语句创建本书中所用的orders表。orders包含3个列，分别是订单号、订单日期、客户ID。所有3个列都需要，因此每个列的定义都含有关键字NOT NULL。这将会阻止插入没有值的列。如果试图插入没有值的列，将返回错误，且插入失败。

理解NULL　不要把NULL值与空串相混淆。NULL值是没有值，它不是空串。如果指定''（两个单引号，其间没有字符），这在NOT NULL列中是允许的。空串是一个有效的值，它不是无值。NULL值用关键字NULL而不是空串指定。

20.1.3　主键再介绍

正如所述，主键值必须唯一，即表中的每个行必须具有唯一的主键值。如果主键使用单个列，则它的值必须唯一。如果使用多个列，则这些列的组合值必须唯一。

迄今为止我们看到的CREATE TABLE例子都是用单个列作为主键。其中主键用以下的类似的语句定义：

```
PRIMARY KEY (vend_id)
```

为创建由多个列组成的主键，应该以逗号分隔的列表给出各列名，

如下所示：

190

输入

```
CREATE TABLE orderitems
(
  order_num  INT        NOT NULL ,
  order_item INT        NOT NULL ,
  prod_id    NCHAR(10) NOT NULL ,
  quantity   INT        NOT NULL ,
  item_price MONEY      NOT NULL ,
  PRIMARY KEY (order_num, order_item)
);
```

orderitems表包含orders表中每个订单的细节。每个订单有多项物品，但每个订单任何时候都只有一个第1项物品，一个第2项物品，如此等等。因此，订单号（order_num列）和订单物品（order_item列）的组合是唯一的，从而适合作为主键，其定义为：

```
PRIMARY KEY (order_num, order_item)
```

主键可以在创建表时定义（如这里所示），或者在创建表之后定义（本章稍后讨论）。

主键和NULL值 第1章介绍过，主键为其值唯一标识表中每个行的列。主键中只能使用不允许NULL值的列。允许NULL值的列不能作为唯一标识。

20.1.4 使用IDENTITY

我们再来看看customers和orders表。customers表中的顾客由列cust_id唯一地标识，每个顾客一个唯一的编号。类似，orders表中的每个订单都有一个唯一的订单号，存储在order_num中。

191

这些编号除它们是唯一的以外没有别的特殊意义。在增加一个新顾客或新订单时，需要一个新的顾客ID或订单号。这些编号可以任意，只要它们是唯一的即可。

显然，使用的最简单的编号是下一个编号，下一个大于当前最大编号的编号。例如，如果cust_id的最大编号为10005，则插入表中的下一个顾客可以具有等于10006的cust_id。

简单吗？不见得。你怎样确定下一个要使用的值？当然，你可以使用SELECT语句得出最大的数（使用第11章介绍的Max()函数），然后对它加1。但这样做并不可靠（你需要找出一种办法来保证，在你执行SELECT和INSERT两条语句之间没有其他人插入行，对于多用户应用，这种情况是很可能出现的），而且效率也不高（执行额外的T-SQL操作肯定不是理想的办法）。

这就是IDENTITY发挥作用的地方了。请看以下代码行（用来创建customers表的部分CREATE TABLE语句）：

```
cust_id      INT         NOT NULL IDENTITY(1,1),
```

IDENTITY告诉SQL Server，每当增加一行时本列自动增量。每次执行一个INSERT操作时，SQL Server自动对该列增量，给该列赋予下一个可用的值。这样给每个行分配一个唯一的cust_id，用作主键值。

IDENTITY需要知道开始值[所谓的种子（seed）]和每次生成新值时的增量值。IDENTITY(1, 1)表示“从种子1开始，每次生成新编号的增量为1”。如果从种子100开始，增量10，则可以使用IDENTITY(100, 10)。如果不提供种子和增量值，则默认使用(1,1)。

192　每个表只允许一个IDENTITY列，而且IDENTITY列一般用作主键。

覆盖IDENTITY　指定为IDENTITY的列中需要使用特定的值吗？你可以在INSERT语句中指定特定的值，但首先需要SET IDENTITY_INSERT为ON，如第18章所述（相应的例子请参阅本书中使用的表填充脚本）。

确定IDENTITY值　让SQL Server（通过IDENTITY）生成主键的一个缺点是你不知道这些值是什么值。

考虑这个场景：你正在增加一个新订单。这要求在orders表中建立一行，然后在orderitms表中对订购的每项物品建立一行。order_num在orderitems表中与订单细节一起存储。这就是为什么orders表和orderitems表为相互关联的表的原因。这显然要求你在插入orders行之后、插入orderitems

行之前知道生成的order_num。

那么，如何在使用IDENTITY列时获得这个值呢？可通过引用@@IDENTITY系统函数获得这个值，如下所示：

```
SELECT @@IDENTITY AS newId;
```

此语句返回最后生成的IDENTITY值作为newId，然后可以在后面的T-SQL语句中使用它。

20.1.5 指定默认值

如果在插入行时没有给出值，SQL Server允许指定默认值。默认值在CREATE TABLE语句的列定义中用关键字DEFAULT指定。 193

请看下面的例子：

输入

```
CREATE TABLE orderitems
(
  order_num  INT         NOT NULL ,
  order_item INT         NOT NULL ,
  prod_id    NCHAR(10)   NOT NULL ,
  quantity   INT         NOT NULL  DEFAULT 1,
  item_price MONEY       NOT NULL ,
  PRIMARY KEY (order_num, order_item)
);
```

分析 这条语句创建orderitems表，它包含构成订单的各项（订单本身存储在orders表中）。quantity列存放订单中每个物品的数量。在此例子中，给列描述增加DEFAULT 1，它指示SQL Server，如果不给出数量则使用数量1。

前一例子使用一个固定值作为给定的DEFAULT。你也可以使用T-SQL函数作为默认值，如下面的例子所示：

输入

```
CREATE TABLE orders
(
  order_num  INT      NOT NULL  IDENTITY(1,1),
  order_date DATETIME NOT NULL  DEFAULT GetDate(),
  cust_id    INT      NOT NULL ,
  PRIMARY KEY (order_num)
);
```

分析　订单被存储在orders表中，而订单日期存储在order_date列。此列的默认值为GetDate()，这是一个T-SQL函数，它返回当前系统日期。这样，如果插入行时不给出明确的order_date值，则
194 自动使用当前日期。

顺便说一下，虽然名为GetDate()，但它实际上返回的是当前系统的日期和时间。如果你想查看GetDate()究竟返回什么样的值，请执行下面的SELECT语句：

输入
```
SELECT GetDate();
```

使用DEFAULT而不是NULL值　许多数据库开发人员喜欢使用DEFAULT值而不是NULL列，特别是对用于计算或数据分组的列更是如此。

20.2　更新表

为更新表定义，可使用ALTER TABLE语句。但是，在理想状态下，当表中存储数据以后，该表就不应该再被更新。在表的设计过程中需要花费大量时间来考虑，以便后期不对该表进行大的改动。

为了使用ALTER TABLE更改表结构，必须给出下面的信息：

- 在ALTER TABLE之后给出要更改的表名（该表必须存在，否则将出错）；
- 所做更改的列表。

下面的例子给表添加一个列：

输入
195
```
ALTER TABLE vendors
ADD vend_phone CHAR(20);
```

分析　这条语句给vendors表增加一个名为vend_phone的列，必须明确其数据类型。

删除刚刚添加的列，可以这样做：

输入
```
ALTER TABLE vendors
DROP COLUMN vend_phone;
```

ALTER TABLE的一种常见用途是定义外键。下面是用来定义本书中的表所用的外键的代码：

输入

```
ALTER TABLE orderitems
ADD CONSTRAINT fk_orderitems_orders FOREIGN KEY (order_num)
REFERENCES orders (order_num);

ALTER TABLE orderitems
ADD CONSTRAINT fk_orderitems_products FOREIGN KEY (prod_id)
REFERENCES products (prod_id);

ALTER TABLE orders
ADD CONSTRAINT fk_orders_customers FOREIGN KEY (cust_id)
REFERENCES customers (cust_id);

ALTER TABLE products
ADD CONSTRAINT fk_products_vendors FOREIGN KEY (vend_id)
REFERENCES vendors (vend_id);

ALTER TABLE productnotes
ADD CONSTRAINT fk_productnotes_products FOREIGN KEY (prod_id)
REFERENCES products (prod_id);
```

分析 这里，由于要更改5个不同的表，使用了5条ALTER TABLE语句。每条语句定义了一个外键。如果单个表上需要多个外键，这些外键可以使用单条ALTER TABLE语句来定义。

复杂的表结构更改一般需要手动删除过程，它涉及以下步骤：

- 用新的列布局创建一个新表；
- 使用INSERT SELECT和SELECT INTO语句（关于这条语句的详细介绍，请参阅第18章）从旧表复制数据到新表。如果有必要，可使用转换函数和计算字段；
- 检验包含所需数据的新表；
- 重命名旧表（如果确定，可以删除它）；
- 用旧表原来的名字重命名新表；
- 根据需要，重新创建触发器、存储过程、索引和外键。

小心使用ALTER TABLE 使用ALTER TABLE要极为小心，应该在进行改动前做一个完整的备份（模式和数据的备份）。数据库表的更改不能撤销，如果增加了不需要的列，可能不能

> 删除它们。类似地，如果删除了不应该删除的列，可能会丢失该列中的所有数据。

20.3　删除表

删除表（删除整个表而不是其内容）非常简单，使用DROP TABLE语句即可：

197

输入

```
DROP TABLE customers2;
```

分析　这条语句删除customers2表（假设它存在）。删除表没有确认，也不能撤销，执行这条语句将永久删除该表。

20.4　重命名表

没有重命名表的T-SQL语句，但可以用SQL Server提供的一个名为sp_rename的存储过程来完成此工作：

输入

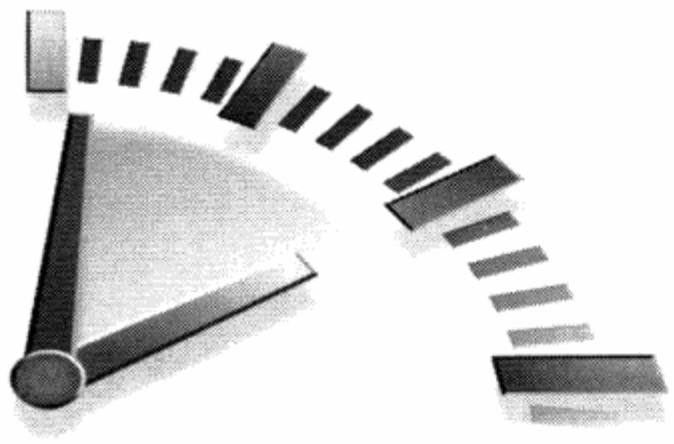
```
EXEC sp_rename 'customers2', 'customers';
```

分析　sp_rename可用来重命名各种对象，包括表。这个例子将表customers2重命名为customers。

20.5　小结

本章介绍了几条新SQL语句。CREATE TABLE用来创建新表，ALTER TABLE用来更改表列（或其他诸如约束或索引等对象），而DROP TABLE用来完整地删除一个表。这些语句必须小心使用，并且应在做了备份后使用。同时，还介绍了标识字段、定义主键和外键，以及其他重要的表和列选项。

198



第21章 使用视图

本章将介绍视图究竟是什么，它们怎样工作，何时使用它们。我们还将看到如何利用视图简化前面章节中执行的某些SQL操作。

21.1 视图

视图是虚拟的表。与包含数据的表不一样，视图只包含使用时动态检索数据的查询。

理解视图的最好方法是看一个例子。第14章中用下面的SELECT语句从3个表中检索数据：

输入

```
SELECT cust_name, cust_contact
FROM customers, orders, orderitems
WHERE customers.cust_id = orders.cust_id
  AND orderitems.order_num = orders.order_num
  AND prod_id = 'TNT2';
```

此查询用来检索订购了某个特定产品的客户。任何需要这个数据的人都必须理解相关表的结构，并且知道如何创建查询和对表进行联结。为了检索其他产品（或多个产品）的相同数据，必须修改最后的WHERE子句。

现在，假如可以把整个查询包装成一个名为productcustomers的虚拟表，则可以如下轻松地检索出相同的数据：

输入

```
SELECT cust_name, cust_contact
FROM productcustomers
WHERE prod_id = 'TNT2';
```

这就是视图的作用。productcustomers是一个视图，作为视图，

它不包含任何列或数据，它包含的是一个SQL查询（与上面用以正确联结表的相同的查询）。

21.1.1　为什么使用视图

我们已经看到了视图应用的一个例子。下面是视图的一些常见应用。

- 重用SQL语句。
- 简化复杂的SQL操作。在编写查询后，可以方便地重用它而不必知道它的基本查询细节。
- 使用表的组成部分而不是整个表。
- 保护数据。可以给用户授予表的特定部分的访问权限而不是整个表的访问权限。
- 更改数据格式和表示。视图可返回与底层表的表示和格式不同的数据。

在视图创建之后，可以用与表基本相同的方式利用它们。可以对视图执行SELECT操作，过滤和排序数据，将视图联结到其他视图或表，甚至能添加和更新数据（添加和更新数据存在某些限制。关于这个内容稍后还要做进一步的介绍）。

重要的是知道视图仅仅是用来查看存储在别处的数据的一种设施。视图本身不包含数据，因此它们返回的数据是从其他表中检索出来的。
200 在添加或更改这些表中的数据时，视图将返回改变过的数据。

21.1.2　视图的规则和限制

下面是关于视图创建和使用的一些最常见的规则和限制。

- 与表一样，视图必须唯一命名（不能给视图取与别的视图或表相同的名字）。
- 对于可以创建的视图数目没有限制。
- SQL Server视图可能只能包含不多于1 024个列。
- 为了创建视图，必须具有足够的访问权限。这些限制通常由数据库管理人员授予。
- 视图可以嵌套，即可以利用从其他视图中检索数据的查询来构造

一个视图。

- ORDER BY不可以用在视图中，但可以用在从视图中检索数据的SELECT语句里。
- 视图不能索引，也不能有关联的触发器或默认值。
- 视图可以和表一起使用。例如，编写一条联结表和视图的SELECT语句。

21.2 使用视图

在理解什么是视图（以及管理它们的规则及限制）后，我们来看一下视图的创建。

- 视图用CREATE VIEW语句来创建。
- 用DROP删除视图，其语法为DROP VIEW viewname;。
- 更新视图时，先使用DROP语句，然后再使用CREATE VIEW语句。 201

21.2.1 利用视图简化复杂的联结

视图的最常见的应用之一是隐藏复杂的SQL，这通常都会涉及联结。请看下面的例子：

输入

```
CREATE VIEW productcustomers AS
SELECT cust_name, cust_contact, prod_id
FROM customers, orders, orderitems
WHERE customers.cust_id = orders.cust_id
  AND orderitems.order_num = orders.order_num;
```

分析 这条语句创建一个名为productcustomers的视图，它联结三个表，以返回已订购了任意产品的所有客户的列表。如果执行SELECT * FROM productcustomers，将列出订购了任意产品的客户。

为检索订购了产品TNT2的客户，可如下进行：

输入

```
SELECT cust_name, cust_contact
FROM productcustomers
WHERE prod_id = 'TNT2';
```

输出

```
cust_nam             cust_contact
-------------------  ------------
Coyote Inc.          Y Lee
Yosemite Place       Y Sam
```

分析 这条语句通过WHERE子句从视图中检索特定数据。在SQL

Server处理此查询时，它将指定的WHERE子句添加到视图查询中的已有
202 WHERE子句中，以便正确过滤数据。

可以看出，视图极大地简化了复杂SQL语句的使用。利用视图，可一次性编写基础的SQL，然后根据需要多次使用。

创建可重用的视图　创建不受特定数据限制的视图是一种好办法。例如，上面创建的视图返回所有产品的客户而不仅仅是生产TNT2的客户。扩展视图的范围不仅使得它能被重用，而且甚至更有用。这样做不需要创建和维护多个类似视图。

21.2.2　用视图重新格式化检索出的数据

如上所述，视图的另一常见用途是重新格式化检索出的数据。下面的SELECT语句（来自第9章）在单个组合计算列中返回供应商名和位置：

输入

```
SELECT RTrim(vend_name) + ' (' + RTrim(vend_country) + ')'
       AS vend_title
FROM vendors
ORDER BY vend_name;
```

输出

```
vend_title
----------------------------------------------------
ACME (USA)
Anvils R Us (USA)
Furball Inc. (USA)
Jet Set (England)
Jouets Et Ours (France)
LT Supplies (USA)
```

203 现在，假如经常需要这个格式的结果。不必在每次需要时执行联结，创建一个视图，每次需要时使用它即可。为把此语句转换为视图，可按如下进行：

输入

```
CREATE VIEW vendorlocations AS
SELECT RTrim(vend_name) + ' (' + RTrim(vend_country) + ')'
       AS vend_title
FROM vendors;
```

分析　这条语句使用与以前的SELECT语句相同的查询创建视图。为了检索出以创建所有邮件标签的数据，可如下进行：

输入

```
SELECT *
FROM vendorlocations
ORDER BY vend_title;
```

输出

```
vend_title
--------------------------------------------------
ACME (USA)
Anvils R Us (USA)
Furball Inc. (USA)
Jet Set (England)
Jouets Et Ours (France)
LT Supplies (USA)
```

21.2.3 用视图过滤不想要的数据

视图对于应用普通的WHERE子句也很有用。例如，可以定义customeremaillist视图，它过滤没有电子邮件地址的客户。为此目的，可使用下面的语句：

204

输入

```
CREATE VIEW customeremaillist AS
SELECT cust_id, cust_name, cust_email
FROM customers
WHERE cust_email IS NOT NULL;
```

分析 显然，在发送电子邮件到邮件列表时，需要排除没有电子邮件地址的用户。这里的WHERE子句过滤了cust_email列中具有NULL值的那些行，使他们不被检索出来。

现在，可以像使用其他表一样使用视图customeremaillist。

输入

```
SELECT *
FROM customeremaillist;
```

输出

```
cust_id     cust_name           cust_email
----------- ------------------- -------------------
10001       Coyote Inc.         ylee@coyote.com
10003       Wascals             rabbit@wascally.com
10004       Yosemite Place      sam@yosemite.com
10005       E Fudd              elmer@fudd.com
```

WHERE子句与WHERE子句 如果从视图检索数据时使用了一条WHERE子句，则两组子句（一组在视图中，另一组是传递给视图的）将自动组合。

21.2.4 使用视图与计算字段

205

视图对于简化计算字段的使用特别有用。下面是第9章中介绍的一条SELECT语句。它检索某个特定订单中的物品，计算每种物品的总价格：

输入

```
SELECT prod_id,
       quantity,
       item_price,
       quantity*item_price AS expanded_price
FROM orderitems
WHERE order_num = 20005;
```

输出

```
prod_id    quantity    item_price           expanded_price
---------- ----------- -------------------- -------------
ANV01      10          5.99                 59.90
ANV02      3           9.99                 29.97
TNT2       5           10.00                50.00
FB         1           10.00                10.00
```

为将其转换为一个视图，如下进行：

输入

```
CREATE VIEW orderitemsexpanded AS
SELECT order_num,
       prod_id,
       quantity,
       item_price,
       quantity*item_price AS expanded_price
FROM orderitems;
```

为检索订单20005的详细内容（上面的输出），如下进行：

输入

206

```
SELECT *
FROM orderitemsexpanded
WHERE order_num = 20005;
```

输出

```
order_num prod_id    quantity    item_price  expanded_price
--------- ---------- ----------- ----------- -------------
20005     ANV01      10          5.99        59.90
20005     ANV02      3           9.99        29.97
20005     TNT2       5           10.00       50.00
20005     FB         1           10.00       10.00
```

可以看到，视图非常容易创建，而且很好使用。正确使用，视图可极大地简化复杂的数据处理。

21.2.5 更新视图

迄今为止的所有视图都是用SELECT语句使用的。然而，视图的数据

能否更新？答案为视情况而定。

通常，视图是可更新的（即，可以对它们使用INSERT、UPDATE和DELETE）。更新视图将更新其基表（可以回忆一下，视图本身没有数据）。如果你对视图增加或删除行，实际上是对其基表增加或删除行。

但是，并非所有视图都是可更新的。基本上可以说，如果SQL Server不能正确地确定被更新的基数据，则不允许更新（包括插入和删除）。这实际上意味着，如果视图定义中有以下操作，则不能进行视图的更新：

- 多个基表；
- 分组（使用GROUP BY和HAVING）；
- 联结；
- 子查询；
- 并；
- 聚集函数（Min()、Count()、Sum()等）；
- DISTINCT；
- 导出（计算）列。

207

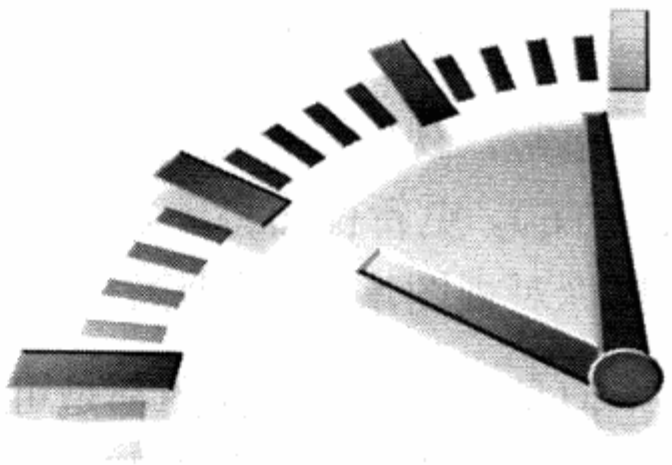
换句话说，本章许多例子中的视图都是不可更新的。这听上去好像是一个严重的限制，但实际上不是，因为视图主要用于数据检索。

21.3 小结

视图为虚拟的表。它们包含的不是数据而是根据需要检索数据的查询。视图提供了一种SQL Server的SELECT语句层次的封装，可用来简化数据处理以及重新格式化基础数据或保护基础数据。

208

第22章



T-SQL程序设计

虽然T-SQL不是一种通用的程序设计语言，但它确实支持变量和条件处理等程序设计结构。因为后面几章中要使用这些结构，所以本章将介绍这些T-SQL程序设计概念。

22.1 理解T-SQL程序设计

迄今为止使用的所有SQL语句都是独立的语句：SELECT语句检索数据，ALTER TABLE语句对表进行更改，等等。但是，有的数据检索更为复杂，常常涉及多条语句、多个处理和混合的数据处理操作。

T-SQL不是一种通用的程序设计语言，但它支持某些基本的程序设计思想和概念，可以用于数据操作处理。这些思想和概念一般在使用简单的SQL语句时不用，但在使用存储过程（见第23章）、游标（见第24章）、触发器（见第25章）等时经常使用。

因此，值得简要地试验一下这些功能，特别是以下功能：

- 使用变量；
- 执行条件处理；
- 重复处理（循环）。

22.2 使用变量

在计算机程序设计中，变量是一个存储值的有名字的对象。虽然变量的使用和性能随程序设计语言的不同而不同，但在这些程序设计语言

中有一个相当普遍的能力，即为后续使用而定义变量和存储值。

T-SQL变量具有特殊的规则和要求：

- 所有T-SQL变量名必须以@开始，局部变量用@为前缀，而全局变量（由SQL Server自身大量使用，一般你个人不用）用@@为前缀；
- 在T-SQL变量可以使用之前，必须使用DECLARE语句声明它们；
- 在声明一个变量时，必须指定它的数据类型；（关于数据类型的解释，请参见第1章。此外，附录D列出了SQL Server支持的数据类型。）
- 可用多条DECLARE语句定义多个变量，也可以在单条DECLARE语句中定义多个变量（变量之间用逗号分隔）；
- 不存在对变量“解除声明”的方法，这表示变量在被声明后直到处理完成前都会一直存在。

22.2.1 声明变量

下面的例子演示DECLARE的用法：

输入

```
DECLARE @age INT;
DECLARE @firstName CHAR(20), @lastName CHAR(20);
```

210

分析 这个例子声明了3个变量。第一条DECLARE语句声明一个名为@age的INT类型（整数类型）的变量，第二条DECLARE语句声明两个CHAR(20)类型（长为20个字符的串）的变量。请注意，第二条DECLARE语句要求在被声明的变量之前有一个逗号。

22.2.2 给变量赋值

变量在刚声明后不包含值（它们实际包含的是NULL）。为给变量赋值，可使用SET语句，如下所示：

输入

```
DECLARE @age INT;
DECLARE @firstName CHAR(20), @lastName CHAR(20);

SET @lastName='Forta';
SET @firstName='Ben';
SET @age=21;
```

分析　这个例子声明如前所示的相同的3个变量，然后使用3条SET语句给它们赋值（我的名字和年龄，多希望是这样）。

赋予默认值　许多T-SQL开发人员发现在声明变量后，用默认值或初始值初始化它们很有好处。

也可以用SELECT给变量赋值，如下所示：

211

输入

```
SELECT @age=21;
```

实际上，除非被赋予变量的值是SELECT操作的结果（在这种情况下显然使用的是SELECT），否则是用SET来把这些值赋予变量。

SET或SELECT?　在使用SET或SELECT赋予变量值之间有一个重要的差别。SET只设置单个变量，要赋值给多个变量必须使用多条SET语句。而SELECT可用来在单条语句中给多个变量赋值。

22.2.3　查看变量内容

在使用变量时，经常需要查看它们的内容。查看变量内容的最简单的办法是输出它们，有两种方法输出变量内容。可以用SELECT来检索变量值，如下所示：

输入

```
DECLARE @age INT;
DECLARE @firstName CHAR(20), @lastName CHAR(20);

SET @lastName='Forta';
SET @firstName='Ben';
SET @age=21;

SELECT @lastName, @firstName, @age
```

输出

```
------------------  ------------------  ----------
Forta               Ben                 21
```

212

分析　正如所见，SELECT可返回变量值，但不返回变量名本身。

使用别名 通过使用AS关键字（见第9章），可在返回变量值的SELECT语句中使用别名。

T-SQL也支持PRINT语句，用来显示消息以及返回的结果。下面是一个例子：

输入

```
DECLARE @age INT;
DECLARE @firstName CHAR(20), @lastName CHAR(20);

SET @lastName='Forta';
SET @firstName='Ben';
SET @age=21;

PRINT @lastName + ', ' + @firstName;
PRINT @age;
```

输出

```
Forta                , Ben
21
```

分析 PRINT输出文本。在这个例子中，第一行是由3个串[一个变量、静态（固定）文本和另一个变量]构成的一个串。第二行打印一个数值变量。

把变量值转换为串 如果你想把年龄输出显示为Age: 21，怎么办？你需要把串Age:与变量@age拼接起来，但用来拼接串的简单的+连接操作会失败，因为@age是一个数值而不是一个串。为解决此问题，需要把@age转换为一个串，以便能连接它，如下所示：

```
PRINT 'Age: ' + Convert(CHAR, @age);
```

213

使用调试程序 还有一种查看变量内容的方法。SQL Server支持使用逐行执行SQL代码的调试程序，允许你在处理中查看变量的内容。SQL Server 2000和SQL Server 2005都支持调试程序的使用，不过这两个版本中的调试程序本身（以及如何使用它们）的变化很大。

SQL Server Debugger的内容超出了本书的范围，读者可以参阅SQL Server本身所包含的文档。

22.2.4　在T-SQL语句中使用变量

在学习了声明、填充和查看变量内容后，我们来看一个如何使用变量的特殊例子。

假如你需要执行两个查询，一个查询返回某个特定顾客的信息，另一个查询返回该顾客所下的订单。这要求两条SELECT语句，如下所示：

输入

```
SELECT cust_name, cust_email
FROM customers
WHERE cust_id = 10001;

SELECT order_num, order_date
FROM orders
WHERE cust_id = 10001
ORDER BY order_date;
```

输出

```
cust_name                      cust_email
------------------------------ ----------------
Coyote Inc.                    ylee@coyote.com

order_num   order_date
----------- -----------------------
20005       2005-09-01 00:00:00.000
20009       2005-10-08 00:00:00.000
```


分析　这个批处理例子很简单，使用两条SELECT语句，因此返回两组结果。

批处理（batch processing）　批是一起提交给SQL Server处理的一组SQL语句。

注意，这两条SELECT语句都在它们的WHERE子句中使用了一个cust_id值。为了对另一个顾客执行相同的查询，需要更改两个WHERE子句。显然，这有可能引发错误。很可能某人更改了一条WHERE子句而没有更改另一条WHERE子句，这会导致返回不正确的数据。在这个简单的例子中，为进行一个不同的搜索，代码中只有两个地方需要更改。想像一个更复杂的例子，其中许多地方用到顾客ID。显然，随着代码长度和复杂性的增加，出错的机会也会增加。

处理前面例子的更好办法是只定义顾客ID一次，从而进行不同搜索只需一次更改。请看下面的例子：

输入

```
-- Define @cust_id
DECLARE @cust_id INT;
SET @cust_id = 10001;

-- Get customer name and e-mail
SELECT cust_name, cust_email
FROM customers
WHERE cust_id = @cust_id;

-- Get customer order history
SELECT order_num, order_date
FROM orders
WHERE cust_id = @cust_id
ORDER BY order_date;
```

215

输出

```
cust_name                      cust_email
------------------------------ ----------------
Coyote Inc.                    ylee@coyote.com

order_num   order_date
----------- -----------------------
20005       2005-09-01 00:00:00.000
20009       2005-10-08 00:00:00.000
```

分析 在这个例子中，使用了相同的两条SELECT语句。但这次首先定义和填充了一个名为@cust_id的局部变量。然后，两条SELECT语句在它们的WHERE子句中以WHERE cust_id = @cust_id使用了该变量。在SQL Server进行处理时，变量@cust_id中的值将被用来构造最终的WHERE子句。

使用变量时不用单引号 如第6章所述，SQL语句中使用的串总是括在单引号内。但在SQL语句中使用变量时，变量名不用单引号括起来，即使把它们作为串使用也是如此。在给串变量赋值时需要使用单引号，但在实际使用变量时不应该使用单引号。

注释代码 你可能已经注意到上面的例子包含以--开始的代码行。这些行是注释（包含在SQL代码中、被SQL Server忽略的消息），它们帮助解释代码的用途。由于SQL语句复杂性的增加，如果以后有人需要理解代码完成什么工作以及为什么要完成这些工作，阅读这些嵌入注释非常有帮助。

216

22.3　使用条件处理

条件处理是在程序代码中做出判断的一种方法，它根据所做出的决策执行某种活动。与大多数程序设计语言一样，T-SQL允许开发人员编写在运行时做出决策的代码，编写这种代码采用的是IF语句。

我们从一个基本的例子开始。想像你正在编写一条SQL语句，它必须处理营业订单（或许更改值，或许复制行到另一表）。此SQL代码需要定期运行，但它所完成的工作视当天是工作日（整天开门营业并处理订单）还是周末（不处理订单）而定。

使用T-SQL的日期和时间函数（第10章介绍）很容易知道当天是周几。GetDate()返回当前日期和时间，DatePart()返回日期成分（几号、星期几、月份等）。为知道当天是周几，可使用DatePart(dw, GetDae())。

下面是根据今天是否为周日，设置一个变量为0或1的一条简单的IF语句：

输入

```
-- Define variables
DECLARE @open BIT

-- Open for business today?
IF DatePart(dw, GetDate()) = 1
   SET @open = 0
ELSE
   SET @open = 1

-- Output
SELECT @open AS OpenForBusiness
```

217

输出

```
OpenForBusiness
---------------
1
```

分析　其中，声明了一个名为@open的局部变量，其数据类型为BIT，只能包含0（假、关闭、否）或1（真、打开、是）。此IF语句将周中的当前日期（由DatePart(dw, GetDate())返回）与1（星期天）比较，如果为真则设置@open为0（今天是星期天），为假（今天不是星期天）则设置@open为1。最后，用SELECT语句返回@open。现实中@open可能用于后续的处理（而不仅仅是简单地返回）。

ELSE是可选的 这里的例子使用一个ELSE子句来定义IF测试返回FALSE时处理的代码。ELSE的使用是可选的，许多IF语句没有ELSE子句。

当然，此代码有一个缺点，因为它只检查今天是否为星期天。因此，如果今天是星期六，则@open将被错误地设置为1。为处理这个问题，需要给IF语句增加一个OR语句：

输入

```
-- Define variables
DECLARE @dow INT
DECLARE @open BIT

-- Get the day of week
SET @dow = DatePart(dw, GetDate());

-- Open for business today?
IF @dow = 1 OR @dow = 7
   SET @open = 0
ELSE
   SET @open = 1

-- Output
SELECT @open AS OpenForBusiness
```

218

分析 这个版本的SQL代码中有两处改动。现在IF语句中使用了一个OR操作符，使得1（星期天）或7（星期六）都与测试匹配。此外，不是在IF语句中获得当前日期为周几，而是声明一个名为@dow的局部变量（用于周几）并用正确的值填充它。为理解这个更改，请看以下可替代的代码：

```
IF DatePart(dw, GetDate()) = 1 OR DatePart(dw, GetDate()) = 7
```

为了不在IF语句中两次获取当前日期为周几，代码中进行了处理，并且把结果保存在一个变量中。

T-SQL在IF语句中支持AND和OR操作符，并且支持圆括号（用来定义计算的次序）。关于AND、OR以及用圆括号定义计算次序，请参阅第7章。

22.4 语句编组

正如所见，IF用来条件性地处理直接跟在其后的内容。在前面的例

子中，如果满足IF或ELSE条件，则处理单条语句。但是，如果必须执行多条语句怎么办？请看下面的例子：

输入

```
-- Define variables
DECLARE @dow INT
DECLARE @open BIT, @process BIT

-- Get the day of week
SET @dow = DatePart(dw, GetDate());

-- Open for business today?
IF @dow = 1 OR @dow = 7
   SET @open = 0
   SET @process = 0
ELSE
   SET @open = 1
   SET @process = 1
```

219

分析　如果执行此代码，将产生一个错误。为什么？因为一旦处理IF，第一个SET或者处理（满足IF条件）或者忽略（不满足IF条件）。但不管怎么样，第二个SET语句总要执行，与IF处理无关（即使代码缩排使它看上去有关）。错误实际上是由ELSE语句引起的，因为SQL Server认为它是一个没有与IF关联的ELSE。

为解决此问题，需要两个新关键字，BEGIN和AND。请看下面的例子：

输入

```
-- Define variables
DECLARE @dow INT
DECLARE @open BIT, @process BIT

-- Get the day of week
SET @dow = DatePart(dw, GetDate());

-- Open for business today?
IF @dow = 1 OR @dow = 7
   BEGIN
      SET @open = 0
      SET @process = 0
   END
ELSE
   BEGIN
      SET @open = 1
      SET @process = 1
   END
```

分析 其中，BEGIN和END用来定义代码块。现在，当IF或ELSE被处理时，处理包括在BEGIN和END之间的整个块而不是仅仅处理后面的语句。

BEGIN和END是很重要的语句，如后面各章节所述，它们不仅仅与IF联合使用。

220

缩排代码 前面例子中的代码使用了两层缩排，第一层排出IF或ELSE执行的代码，第二层清晰地定义了每个BEGIN和END块的内容。不存在专门用于如何缩排的严格规定，你可以采用这里的风格，也可以采用有助于更好地组织和阅读你的代码的任意风格。

22.5 使用循环

SQL语句是顺序处理的，一次一条，每条处理一次。与其他程序设计语言一样，T-SQL支持循环，能够根据需要重复一个代码块。在T-SQL中，循环由WHILE语句完成。

WHILE和游标 WHILE经常与游标结合使用，如第24章所述。

下面是一个演示如何使用WHILE的例子（虽然不太典型且没有多大用处）：

输入

```
DECLARE @counter INT

SET @counter=1

WHILE @counter <= 10
BEGIN
   PRINT @counter
   SET @counter=@counter+1
END
```

221

输出

```
1
2
3
4
```

```
5
6
7
8
9
10
```

分析　这个例子定义一个名为@counter的局部变量并用值1初始化。WHILE循环测试，看@counter是否小于等于10，只要这个条件为真，就用PRINT显示相应的数值，然后增量。

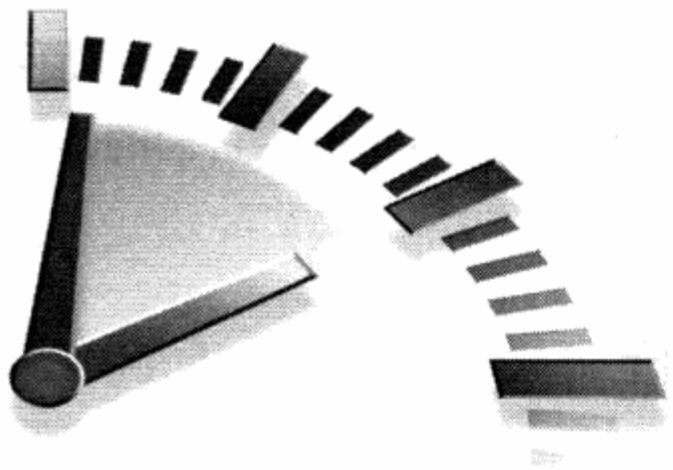
WHILE和BEGIN/END　与IF一样，WHILE只重复跟在它后面的单条语句。为了重复多行代码，可使用BEGIN和END限定该代码块，如上面例子所示。

经常与WHILE结合使用的另外两条语句为：

- BREAK立即退出当前WHILE循环（或IF）；
- CONTINUE在循环起始处重新开始处理。

22.6　小结

T-SQL支持基本的程序设计结构，包括变量、条件处理和循环。就本身来说，它们不是那么有用，但在与其他SQL Server特性结合使用时，它们非常重要，如以后章节所述。



第23章 使用存储过程

本章介绍什么是存储过程，为什么要使用存储过程以及如何使用存储过程，并且介绍创建和使用存储过程的基本语法。

23.1 存储过程

迄今为止，使用的大多数SQL语句都是针对一个或多个表的单条语句。并非所有操作都这么简单，经常会有一个完整的操作需要多条语句才能完成。例如，考虑以下的情形。

- 为了处理订单，需要核对以保证库存中有相应的物品。
- 如果库存有物品，这些物品需要预定以便不将它们再卖给别的人，并且要减少可用物品数量以反映正确的库存量。
- 库存中没有的物品需要订购，这需要与供应商进行某种交互。
- 关于哪些物品入库（并且可以立即发货）和哪些物品退订，需要通知相应的客户。

这显然不是一个完整的例子，它甚至超出了本书中所用样例表的范围，但足以帮助表达我们的意思了。执行这个处理需要针对许多表的多条T-SQL语句。此外，需要执行的具体语句及其次序也不是固定的，它们可能会（和将）根据哪些物品在库存中哪些不在而变化。

那么，怎样编写此代码？可以单独编写每条语句，并根据结果有条件地执行另外的语句。在每次需要这个处理时（以及每个需要它的应用中）都必须做这些工作。

可以创建存储过程。存储过程简单来说，就是为以后的使用而保存的一条或多条T-SQL语句的集合。可将其视为批文件，虽然它们的作用不仅限于批处理。

23.2　为什么要使用存储过程

既然我们知道了什么是存储过程，那么为什么要使用它们呢？有许多理由，下面列出一些主要的理由。

- 通过把处理封装在容易使用的单元中，简化复杂的操作（正如前面例子所述）。
- 由于不要求反复建立一系列处理步骤，这保证了数据的完整性。如果所有开发人员和应用程序都使用同一（试验和测试）存储过程，则所使用的代码都是相同的。

  这一点的延伸就是防止错误。需要执行的步骤越多，出错的可能性就越大。防止错误保证了数据的一致性。

- 简化对变动的管理。如果表名、列名或业务逻辑（或别的内容）有变化，只需要更改存储过程的代码。使用它的人员甚至不需要知道这些变化。

  这一点的延伸就是安全性。通过存储过程限制对基础数据的访问减少了数据讹误（无意识的或别的原因所导致的数据讹误）的机会。

224

- 提高性能。因为使用存储过程比使用单独的SQL语句要快。
- 存在一些只能用在单个请求中的T-SQL语言元素和SQL Server特性，存储过程可以使用它们来编写功能更强更灵活的代码。（在下一章的例子中可以看到。）

换句话说，使用存储过程有3个主要的好处，即简单、安全、高性能。显然，它们都很重要。不过，在将SQL代码转换为存储过程前，也必须知道它的一些缺陷。

- 一般来说，存储过程的编写比基本SQL语句复杂，编写存储过程需要更高的技能，更丰富的经验。
- 你可能没有创建存储过程的安全访问权限。许多数据库管理员限

制存储过程的创建权限，允许用户使用存储过程，但不允许他们创建存储过程。

尽管有这些缺陷，存储过程还是非常有用的，并且应该使用。

不能编写存储过程？你依然可以使用 SQL Server将编写存储过程的安全和访问与执行存储过程的安全和访问区分开来。这是好事情。即使你不能（或不想）编写自己的存储过程，也仍然可以在适当的时候执行别的存储过程。

23.3 使用存储过程

使用存储过程需要知道如何执行（运行）它们。存储过程的执行远比其定义更经常遇到，因此，我们将从执行存储过程开始介绍。然后再介绍创建和使用存储过程。

23.3.1 执行存储过程

SQL Server过程用EXECUTE语句执行。EXECUTE使用存储过程的名字以及需要传递给它的任意参数。请看以下例子（在创建这个过程前不能实际执行它）：

输入

```
EXECUTE productpricing @cheap OUTPUT,
                       @expensive OUTPUT,
                       @average OUTPUT
```

分析 其中执行名为productpricing的存储过程，它计算并返回产品的最低、最高和平均价格。

存储过程可以显示结果，也可以不显示结果，稍后会讲到。

EXECUTE或EXEC EXECUTE可以简写为EXEC，EXECUTE和EXEC的功能完全相同。

23.3.2 创建存储过程

正如所述，编写存储过程并不是微不足道的事情。为给你提供创建

存储过程涉及什么内容的一个直观印象，请看一个例子——一个返回产品平均价格的存储过程。以下是其代码：

输入

```
CREATE PROCEDURE productpricing AS
BEGIN
   SELECT Avg(prod_price) AS priceaverage
   FROM products;
END;
```

226

分析 此存储过程名为productpricing，用CREATE PROCEDURE productpricing AS语句定义。BEGIN和END语句用来限定存储过程体，过程体本身仅是一个简单的SELECT语句（使用第11章介绍的Avg()函数）。

在SQL Server处理这段代码时，它创建一个新的存储过程productpricing。没有返回数据，因为这段代码没有调用此存储过程，这里只是为以后使用而创建它。

那么，如何使用这个存储过程？如下所示：

输入

```
EXECUTE productpricing;
```

输出

```
priceaverage
--------------------
16.1335
```

分析 EXECUTE productpricing();执行刚创建的存储过程并显示返回的结果。

23.3.3　删除存储过程

存储过程在创建之后，被保存在服务器上以供使用，直至被删除。删除命令（类似于第20章所介绍的语句）从服务器中删除存储过程。

为删除刚创建的存储过程，可使用以下语句：

227

输入

```
DROP PROCEDURE productpricing;
```

分析 这条语句删除刚创建的存储过程。

23.3.4　使用参数

productpricing只是一个简单的存储过程，它简单地显示SELECT语

句的结果。一般，存储过程并不显示结果，而是把结果返回给你指定的变量。

以下是productpricing的修改版本（如果不先删除此存储过程，则不能再次创建它）：

输入

```
CREATE PROCEDURE productpricing
   @price_min MONEY OUTPUT,
   @price_max MONEY OUTPUT,
   @price_avg MONEY OUTPUT

AS
BEGIN
   SELECT @price_min = Min(prod_price)
   FROM products;
   SELECT @price_max = Max(prod_price)
   FROM products;
   SELECT @price_avg = Avg(prod_price)
   FROM products;
END;
```

分析

此存储过程接受3个参数：@price_min存储产品的最低价格，@price_max存储产品的最高价格，@price_avg存储产品的平均价格。每个参数必须具有自己指定的类型（这里使用MONEY）。关键字OUTPUT指出相应的参数用来从存储过程传出一个值（返回给调用者）。没有OUTPUT，变量只能用来向存储过程传递值。存储过程的代码如上封闭在BEGIN和END语句内，它们是一系列SELECT语句，用来检索值，然后保存到相应的变量。

228

变量必须以@开头 正如在第22章所讨论的一样，所以变量名必须以@开头。

参数的数据类型 存储过程的参数允许的数据类型与表中使用的数据类型相同。附录D列出了这些类型。

为调用此修改过的存储过程，必须指定3个变量名，如下所示：

输入

```
DECLARE @cheap MONEY
DECLARE @expensive MONEY
DECLARE @average MONEY
```

```
EXECUTE productpricing @cheap OUTPUT,
                       @expensive OUTPUT,
                       @average OUTPUT
```

分析 由于此存储过程要求3个参数，因此必须正好传递3个参数，不多也不少。因此，3个参数被传递给这条EXECUTE语句，而且，由于这些变量被用作参数，它们必须用DECLARE事先声明。这3个变量将存储过程返回的结果，所以每个变量必须作为OUTPUT传递（否则，将不返回它们的值）。这些变量不需要与存储过程本身中的接收变量同名，确实，在这个例子中它们也不同名。

在调用时，这条语句并不显示任何数据。它返回以后可以显示（或在其他处理中使用）的变量。

229

为了显示检索出的产品平均价格，可如下进行：

输入

```
SELECT @cheap;
```

输出

```
Cheap
---------------------
2.50
```

为了获得3个值，可使用以下语句：

输入

```
SELECT @cheap, @expensive, @average;
```

输出

```
Cheap    Expensive  Average
-------  ---------  -------
2.50     55.00      16.1335
```

下面是另一个例子。这次，这个例子传递参数给存储过程，并返回OUTPUT参数。ordertotal接受一个订单号并返回该订单的合计：

输入

```
CREATE PROCEDURE ordertotal
   @order_num INT,
   @order_total MONEY OUTPUT

AS
BEGIN
   SELECT @order_total = Sum(item_price*quantity)
   FROM orderitems
   WHERE order_num = @order_num;
END;
```

230

分析 @order_num用来传递一个值给存储过程，因此不需要OUTPUT。@order_total定义为OUTPUT，因为要从存储过程

返回合计值。SELECT语句使用这两个参数，WHERE子句使用@order_num选择正确的行，而@order_total存储计算出的合计值。

为调用这个新存储过程，可使用以下语句：

输入

```
DECLARE @order_total MONEY
EXECUTE ordertotal 20005, @order_total OUTPUT
SELECT @order_total
```

输出

```
-------------------
149.87
```

分析 必须给ordertotal传递两个参数：第一个参数为订单号，第二个参数为包含计算出来的合计的变量名。其中的一个参数（订单号）是静态值（不是变量）。

为了得到另一个订单的合计显示，需要再次调用存储过程，然后重新显示变量：

输入

```
DECLARE @order_total MONEY
EXECUTE ordertotal 20009, @order_total OUTPUT
SELECT @order_total
```

23.3.5 建立智能存储过程

迄今为止使用的所有存储过程基本上都是封装简单的T-SQL语句。虽然它们全都是有效的存储过程例子，但它们所能完成的工作你直接用这些被封装的语句就能完成（如果说它们还能带来更多的东西，那就是使事情更复杂）。只有在存储过程内包含业务规则和智能处理时，它们的真正威力才能显现出来。 231

考虑这个场景：你需要获得与以前一样的订单合计，但需要对合计增加营业税，不过只针对某些顾客（或许是你所在州中那些顾客）。那么，你需要做几件事情：

- 获得合计（与以前一样）；
- 把营业税有条件地添加到合计；
- 返回合计（带或不带税）。

存储过程的完整工作如下：

输入

```
-- Name: ordertotal
-- Parameters: @order_num    = order number
--             @taxable      = 0 if not taxable, 1 if taxable
--             @order_total  = order total variable

CREATE PROCEDURE ordertotal
   @order_num INT,
   @taxable BIT,
   @order_total MONEY OUTPUT
AS
BEGIN

   -- Declare variable for total
   DECLARE @total MONEY;
   -- Declare tax percentage
   DECLARE @taxrate INT;
   -- Set tax rate (adjust as needed)
   SET @taxrate = 6;

   -- Get the order total
   SELECT @total = Sum(item_price*quantity)
   FROM orderitems
   WHERE order_num = @order_num
   -- Is this taxable?
   IF @taxable = 1
      -- Yes, so add taxrate to the total
      SET @total=@total+(@total/100*@taxrate);

   -- And finally, save to output variable
   SELECT @order_total = @total;

END;
```


如果需要，首先删除　在你能保存存储过程的这个新版本之前，可能需要删除已有的存储过程。

分析　此存储过程有很大的变动。首先，增加了注释（如第22章所讲，前面放置--）。添加了另外一个参数@taxable，它是BIT类型的（如果要增加税则为1，否则为0）。在存储过程体中，用DECLARE语句定义了两个局部变量，这个例子中的@taxrate被设置为6%。SELECT语句已经改变，因此其结果存储到@total（局部变量）而不是@order_total。IF语句检查@taxable是否为真，如果为真，则用一条SET语句增加营业税到局部变量@total。最后，用另一SELECT语句将@total（它或许增

加或许不增加营业税）保存到@order_total。

这显然是一个更高级，功能更强的存储过程。为试验它，请用以下两条语句：

输入

```
DECLARE @order_total MONEY
EXECUTE ordertotal 20005, 0, @order_total OUTPUT
SELECT @order_total
```

233

输出

```
---------------------
149.87
```

输入

```
DECLARE @order_total MONEY
EXECUTE ordertotal 20005, 1, @order_total OUTPUT
SELECT @order_total
```

输出

```
---------------------
158.8622
```

分析 这两个EXECUTE调用的唯一区别就是第二个参数：第一个EXECUTE传递0（假），第二个EXECUTE传递1（真）。这样使得很便于将营业税有条件地加到订单合计上。

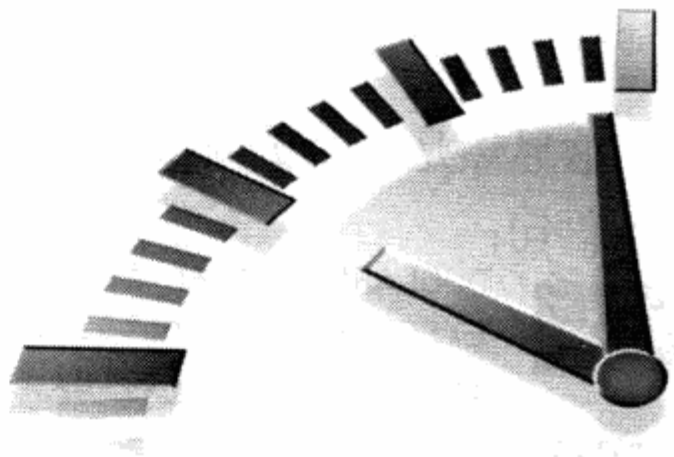
23.4 小结

本章介绍了什么是存储过程以及为什么要使用存储过程。我们介绍了存储过程的执行和创建的语法以及使用存储过程的一些方法。存储过程经常与游标操作一起使用，下一章我们将学习游标。

234

第24章

使用游标



本章将讲授什么是游标以及如何使用游标。

24.1 游标

由前几章可知，T-SQL检索操作返回一组称为结果集的行。这组返回的行都是与SQL语句相匹配的行（零行或多行）。简单地使用SELECT语句，例如，没有办法得到第一行、下一行或前10行，也不存在每次一行地处理所有行的简单方法（相对于成批地处理它们）。

有时，需要在检索出来的行中前进或后退一行或多行。这就是使用游标的原因。游标（cursor）是一个存储在SQL Server上的数据库查询，它不是一条SELECT语句，而是被该语句检索出来的结果集。在存储了游标之后，应用程序可以根据需要滚动或浏览其中的数据。

游标主要用于交互式应用，其中用户需要滚动屏幕上的数据，并对数据进行浏览或做出更改。

24.2 使用游标

使用游标涉及几个明确的步骤。

- 在能够使用游标前，必须声明（定义）它。这个过程实际上没有检索数据，它只是定义要使用的SELECT语句。
- 一旦声明后，必须打开游标以供使用。这个过程用前面定义的SELECT语句把数据实际检索出来。

- 对于填有数据的游标，根据需要取出（检索）各行。
- 当取出需要的数据后时，必须关闭游标。
- 最后，必须删除游标。

在声明游标后，可根据需要频繁地打开和关闭游标（直到它被删除）。在游标打开后，可根据需要频繁地执行取操作。

24.2.1 创建和删除游标

游标用DECLARE语句（在第22章讨论）创建。DECLARE命名游标，并定义相应的SELECT语句，根据需要带WHERE和其他子句。当不再使用游标时，必须用DEALLOCATE删除它。

隐含的DEALLOCATE DEALLOCATE实际上可以省略，在此情况下，当游标超出定义范围时，SQL Server将会自动删除游标。

例如，下面的语句用检索所有订单号的SELECT语句定义一个名为orders_cursor的游标：

输入

```
DECLARE orders_cursor CURSOR
FOR
SELECT order_num FROM orders ORDER BY order_num;

DEALLOCATE orders_cursor;
```

236

分析 DECLARE语句用来定义和命名游标，在这里，为orders_cursor。此游标什么也没有做，就立即被DEALLOCATE删除。

只能DECLARE一次 在游标被声明后，不能再次声明它，即使所使用的DECLARE语句与前面使用的一样也不行。为更改游标，必须首先用DEALLOCATE删除它，然后再用DECLARF声明它。

在游标被定义之后，就可以打开它了。

24.2.2 使用游标

游标用OPEN语句来打开：

输入

```
OPEN orders_cursor;
```

分析 在处理OPEN语句时执行查询，存储检索出的数据以供浏览和滚动。

游标处理完成后，应当使用如下语句关闭游标：

输入

```
CLOSE orders_cursor;
```

分析 CLOSE释放游标使用的所有内存和内部资源，因此在每个游标不再需要时都应该关闭。

237

在一个游标关闭后，如果没有重新打开，则不能使用它。但是，使用声明过的游标不需要再次声明，用OPEN语句打开它就可以了（只要没有使用DEALLOCATE删除这个游标）。

下面是前一例子的修改版本：

输入

```
-- Define the cursor
DECLARE orders_cursor CURSOR
FOR
SELECT order_num FROM orders ORDER BY order_num;

-- Open cursor (retrieve data)
OPEN orders_cursor;

-- Close cursor
CLOSE orders_cursor

-- And finally, remove it
DEALLOCATE orders_cursor;
```

分析 这个存储过程声明、打开和关闭一个游标，然后删除这个游标，但对检索出的数据什么也没做。

24.2.3 使用游标数据

在一个游标被打开后，可以使用FETCH语句分别访问它的每一行。FETCH指定检索什么数据，检索出来的数据存储在什么地方。它还向前移动游标中的内部行指针，使下一条FETCH语句检索下一行（不重复读取同一行）。

238

第一个例子从游标中检索单个行（第一行）：

输入

```
-- Local variables
DECLARE @order_num INT;

-- Define the cursor
DECLARE orders_cursor CURSOR
FOR
SELECT order_num FROM orders ORDER BY order_num;

-- Open cursor (retrieve data)
OPEN orders_cursor;

-- Perform the first fetch (get first row)
FETCH NEXT FROM orders_cursor INTO @order_num;

-- Close cursor
CLOSE orders_cursor

-- And finally, remove it
DEALLOCATE orders_cursor;
```

分析 其中FETCH用来检索当前行的order_num列（将自动从第一行开始）到一个名为@order_num的局部声明的变量中。对检索出的数据不做任何处理。

取什么 这个例子中的FETCH语句使用FETCH NEXT取下一行。这是最常使用的FETCH，但是，还可以使用别的FETCH选项。它们包括检索前一行的FETCH PRIOR，检索第一行和最后一行的FETCH FIRST和FETCH LAST，取从顶端开始的特定行数的行的FETCH ABSOLUTE，取从当前行开始的特定行数的行的FETCH RELATIVE。

在下一个例子中，循环检索数据，从第一行到最后一行：

239

输入

```
-- Local variables
DECLARE @order_num INT;

-- Define the cursor
DECLARE orders_cursor CURSOR
FOR
SELECT order_num FROM orders ORDER BY order_num;

-- Open cursor (retrieve data)
```

```
OPEN orders_cursor;

-- Perform the first fetch (get first row)
FETCH NEXT FROM orders_cursor INTO @order_num;

-- Check @@FETCH_STATUS to see if there are any more rows
-- to fetch.
WHILE @@FETCH_STATUS = 0
BEGIN
   -- This is executed as long as the previous fetch succeeds.
   FETCH NEXT FROM orders_cursor INTO @order_num;
END

-- Close cursor
CLOSE orders_cursor

-- And finally, remove it
DEALLOCATE orders_cursor;
```

分析 与前一个例子一样，此代码使用FETCH检索当前order_num并把它放入一个名为@order_num的已声明的变量。但与前一个例子不一样，这里的FETCH后跟WHILE循环，因此它反复重复。循环何时终止？每次使用FETCH，名为@@FETCH_STATUS的内部函数都要获得一个状态码。如果FETCH成功，@@FETCH_STATUS返回0，否则返回一个负值。这样，WHILE @@FETCH_STATUS=0时WHILE继续循环。

240

如果一切正常，你可以在循环内放入任意需要的处理（在BEGIN语句之后，下一条FETCH语句之前）。

为了把这些内容组织起来，下面给出我们的游标样例的更进一步修改的版本，这次对取出的数据进行某种实际的处理：

输入

```
-- Local variables
DECLARE @order_num INT;
DECLARE @order_total MONEY;
DECLARE @total MONEY;

-- Initialize @total
SET @total=0;

-- Define the cursor
DECLARE orders_cursor CURSOR
FOR
SELECT order_num FROM orders ORDER BY order_num;
```

```
-- Open cursor (retrieve data)
OPEN orders_cursor;

-- Perform the first fetch (get first row)
FETCH NEXT FROM orders_cursor INTO @order_num;

-- Check @@FETCH_STATUS to see if there are any more rows
-- to fetch.
WHILE @@FETCH_STATUS = 0
BEGIN
   -- Get this order total (including tax)
   EXECUTE ordertotal @order_num, 1, @order_total OUTPUT

   -- Add this order to the total
   SET @total = @total + @order_total

   -- Get next row
   FETCH NEXT FROM orders_cursor INTO @order_num;
END

-- Close cursor
CLOSE orders_cursor

-- And finally, remove it
DEALLOCATE orders_cursor;

-- And finally display calculated total
SELECT @total AS GrantTotal;
```

241

分析 在这个例子中，我们声明了名为@order_total的一个变量（存储每个订单的合计）和名为@total的另一个变量（存储所有订单的合计）。FETCH像以前一样取每个@order_num，然后用EXECUTE执行一个存储过程（前一章中创建的存储过程），计算每个订单的扣营业税的合计（结果存储在@order_total中）。每当检索出一个@order_total，就用SET语句将它加到@total上。最后用一条SELECT返回总计。

这样，我们就得到了游标、逐行处理以及存储过程执行的一个完整的工作样例。

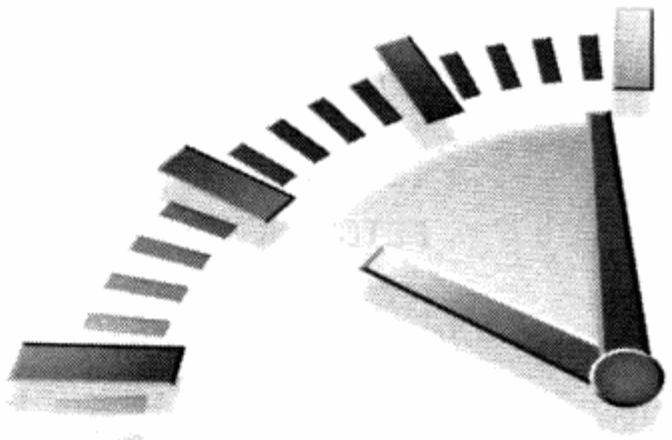
24.3 小结

本章介绍了什么是游标以及为什么要使用游标，举了演示基本游标使用的例子，并且讲解了对游标结果进行循环以及逐行处理的技术。

242

第25章

使用触发器



本章学习什么是触发器，为什么要使用触发器以及如何使用触发器。本章还介绍创建和使用触发器的语法。

25.1 理解触发器

T-SQL语句在需要时被执行，存储过程也是如此。但是，如果你想要某条语句（或某些语句）在事件发生时自动执行，怎么办呢？下面是几个例子：

- 每当增加一个顾客到某个数据库表时，都检查其电话号码格式是否正确，州的缩写是否为大写；
- 每当订购一个产品时，都从库存数量中减去订购的数量；
- 无论何时删除一行，都在某个存档表中保留一个副本。

所有这些例子的共同之处是它们都需要在某个表更改发生时自动处理。这确切地说就是触发器。触发器是SQL Server响应以下任意语句而自动执行的一条T-SQL语句（或括在BEGIN和END语句之间的一组语句）：

- DELETE；
- INSERT；
- UPDATE。

其他T-SQL语句不支持触发器。

表和视图 表和视图支持触发器（但临时表不支持）。

25.1.1 创建触发器

在创建触发器时，需要给出3个信息：

- 唯一的触发器名；
- 触发器关联的表；
- 触发器应该响应的活动（DELETE、INSERT或UPDATE）。

触发器用CREATE TRIGGER语句创建。下面是一个简单的例子：

输入

```
CREATE TRIGGER newproduct_trigger ON products
AFTER INSERT
AS
SELECT 'Product added';
```

分析 CREATE TRIGGER用来创建名为newproduct_trigger的新触发器。触发器定义为AFTER INSERT，所以此触发器将在INSERT语句成功执行后执行。文本Product added将对每个插入的行显示一次。

为了测试这个触发器，使用INSERT语句添加一行或多行到products；你将看到对每个成功的插入，显示Product added消息。

触发器按每个表每个事件每次地定义，每个表每个事件每次只允许一个触发器。因此，每个表最多支持3个触发器（INSERT、UPDATE和DELETE各一个触发器）。

244

每个触发器多个事件 单个触发器可与多个事件关联，因此，如果你需要对INSERT和UPDATE操作执行一个触发器，可定义它为AFTER INSERT，UPDATE。

INSTEAD OF触发器 多数触发器为AFTER触发器，它们在事件发生后执行。SQL Server支持另一种类型的触发器，名为INSTEAD OF触发器，如果定义，这种触发器不由原来的T-SQL语句触发。例如，如果你想不允许删除行，可创建一个INSTEAD OF触发器，用更新行使其不活动（或者通过在这些行中设置一个标志）的T-SQL语句来替代特定表上的DELETE。本章不介绍INSTEAD OF触发器。

25.1.2　删除触发器

现在，删除触发器的语法应该很明显了。为了删除一个触发器，可使用DROP TRIGGER语句，如下所示：

输入

```
DROP TRIGGER newproduct_trigger;
```

分析　上面的代码删除触发器newproduct_trigger。

更新触发器　可使用ALTER TRIGGER更新触发器，或者可以删除触发器然后再重新建立。

245

25.1.3　启用和禁用触发器

有时需要执行T-SQL语句而不执行所定义的触发器。SQL Server允许你禁用触发器，然后根据需要再启用它们，而不是删除触发器以后再重建。

为禁用触发器，使用DISABLE TRIGGER语句，如下所示：

输入

```
DISABLE TRIGGER newproduct_trigger ON products;
```

为了重新启用触发器，使用ENABLE TRIGGER语句，如下所示：

输入

```
ENABLE TRIGGER newproduct_trigger ON products;
```

25.1.4　确定触发器的任务

触发器非常有用，功能非常强。但它们可能改变SQL Server的行为方式，可能执行你不知道的代码。因为多数触发器是悄悄地执行的（不提供反馈），要在任何给定时刻确定是否有触发器正在执行可能很困难。

为解决这个问题，可使用内建的存储过程SP_HELPTRIGGER：

输入

```
SP_HELPTRIGGER products;
```

分析　SP_HELPTRIGGER取一个表名并返回触发器的一个列表（如果定义有触发器的话），用标志指出触发器的类型。如果在创建了上述的newproduct触发器后执行此代码，ISINSERT和ISAFTER两者都

将为1，因为我们创建了AFTER INSERT触发器。

246

25.2 使用触发器

在有了前面的基础知识后，我们现在来看所支持的每种触发器类型以及它们的差别。

25.2.1 INSERT触发器

INSERT触发器在INSERT语句执行之后执行。在INSERT触发器代码内，可引用一个名为INSERTED的虚拟表，访问被插入的行。

下面举一个例子（一个真正有用的例子）。IDENTITY列具有SQL Server自动赋予的值。第20章建议了几种确定新生成值的方法，但下面是一种更好的方法：

输入

```
CREATE TRIGGER neworder_trigger ON orders
AFTER INSERT
AS
SELECT @@IDENTITY AS order_num;
```

分析 此代码创建一个名为neworder_trigger的触发器，它在表orders上执行AFTER INSERT。在插入一个新订单到orders表时，SQL Server生成一个新订单号并保存到order_num中。触发器从@@IDENTITY取得这个值并返回它。对于orders的每次插入使用这个触发器将总是返回新的订单号。

为测试这个触发器，试着插入一下新行，如下所示：

输入

```
INSERT INTO orders(order_date, cust_id)
VALUES(GetDate(), 10001);
```

247

输出

```
order_num
------------
20010
```

分析 orders包含3个列。order_date和cust_id必须给出，order_num由SQL Server自动生成，而现在order_num还自动被返回。

25.2.2　DELETE触发器

DELETE触发器在DELETE语句执行之后执行。在DELETE触发器代码内，可以引用一个名为DELETED的虚拟表访问被删除的行。

下面的例子演示使用DELETE保存将要被删除的行到一个存档表中：

输入

```
CREATE TRIGGER deleteorder_trigger ON orders
AFTER DELETE
AS
BEGIN
   INSERT INTO orders_archive(order_num, order_date, cust_id)
   SELECT order_num, order_date, cust_id FROM DELETED;
END;
```

分析　此触发器在从表orders中删除行时执行。它使用INSERT SELECT语句保存DELETED中的行到一个名为orders_archive的存档表中。（为实际使用这个例子，需要用与orders相同的列创建一个名为orders_archive的表。）

248

多语句触发器　正如所见，触发器deleteorder_trigger使用BEGIN和END语句标记触发器体。这在此例子中并不是必需的，不过也没有害处。使用BEGIN END块的好处是触发器能容纳多条SQL语句（在BEGIN END块中一条挨着一条）。BEGIN END将在第22章介绍。

25.2.3　UPDATE触发器

UPDATE触发器在UPDATE语句执行之后执行。在UPDATE触发器代码中，你可以引用名为DELETED的虚拟表访问以前的值（UPDATE语句之前），引用名为INSERTED的虚拟表访问新更新的值。

下面的例子保证州名缩写总是大写（不管UPDATE语句中给出的是大写还是小写）：

输入

```
CREATE TRIGGER vendor_trigger ON vendors
AFTER INSERT, UPDATE
```

```
AS
BEGIN
   UPDATE vendors
   SET vend_state=Upper(vend_state)
   WHERE vend_id IN (SELECT vend_id FROM INSERTED);
END;
```

分析 此触发器在INSERT、UPDATE之后执行。每当行被插入或更新时，vend_state中的值都用Upper(vend_state)替换。 249

25.2.4 关于触发器的进一步介绍

在结束本章之前，我们再介绍一些使用触发器时需要记住的重点。

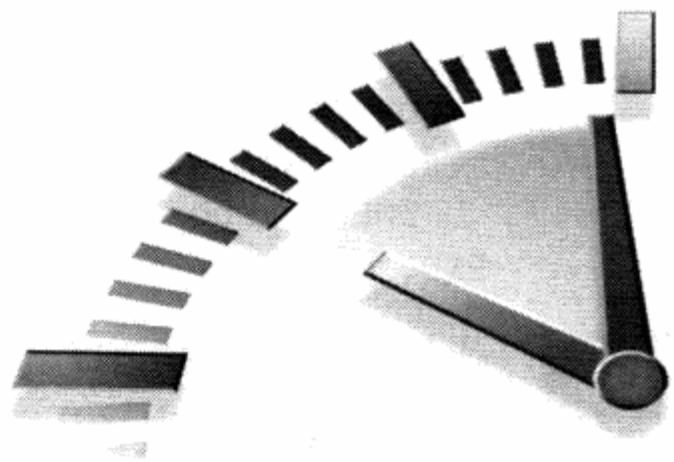
- 创建触发器可能需要特殊的安全访问权限。但是，触发器的执行是自动的。如果INSERT、UPDATE或DELETE语句能够执行，则相关的触发器也能执行。
- 应该用触发器来保证数据的一致性（大小写、格式等）。在触发器中执行这种类型的处理的优点是它总是进行这种处理，而且是透明地进行，与客户机应用无关。
- 触发器的一种非常有意义的使用是创建审计跟踪。使用触发器，把更改（如果需要，甚至还有之前和之后的状态）记录到另一个表非常容易。
- 触发器可以像多数T-SQL语句一样调用存储过程。

25.3 小结

本章介绍了什么是触发器以及为什么要使用触发器，列举了几个用于INSERT、DELETE和UPDATE操作的触发器例子。 250

第 26 章

管理事务处理



本章介绍什么是事务处理以及如何利用COMMIT和ROLLBACK语句来管理事务处理。

26.1 事务处理

事务处理（transaction processing）可以用来维护数据库的完整性，它保证成批的T-SQL操作要么完全执行，要么完全不执行。

正如第14章所述，关系数据库设计把数据存储在多个表中，使数据更容易操纵、维护和重用。不用深究如何以及为什么进行关系数据库设计，在某种程度上说，设计良好的数据库模式都是关联的。

前面章中使用的orders表就是一个很好的例子。订单存储在orders和orderitems两个表中：orders存储实际的订单，而orderitems存储订购的各项物品。这两个表使用称为主键（参阅第1章）的唯一ID互相关联。这两个表又与包含客户和产品信息的其他表相关联。

给系统添加订单的过程如下：

(1) 检查数据库（customers表）中是否存在相应的客户，如果不存在，添加他/她；

(2) 检索客户的ID；

(3) 添加一行到orders表，把它与客户ID关联；

(4) 检索orders表中赋予的新订单ID；

(5) 对于订购的每个物品在orderitems表中添加一行，通过检索出

来的ID把它与orders表关联（以及通过产品ID与products表关联）。

现在，假如由于某种数据库故障（如超出磁盘空间、安全限制、表锁等）阻止了这个过程的完成。数据库中的数据会出现什么情况？

如果故障发生在添加客户之后及orders表添加之前，不会有什么问题。某些客户没有订单是完全合法的。在重新执行此过程时，所插入的客户记录将被检索和使用。可以有效地从出故障的地方开始执行此过程。

但是，如果故障发生在orders行插入之后及orderitems行添加之前，怎么办呢？现在，数据库中有一个空订单。

更糟的是，如果系统在添加orderitems行之中出现故障。结果是数据库中存在不完整的订单，而且你还不知道。

如何解决这种问题？这里就需要使用事务处理了。事务处理是一种机制，用来管理必须成批执行的T-SQL操作，以保证数据库不包含不完整的操作结果。利用事务处理，可以保证一组操作不会中途停止，它们或者作为整体执行，或者完全不执行（除非明确指示）。如果没有错误发生，整组语句提交给（写到）数据库表。如果发生错误，则进行回退（撤销）以恢复数据库到某个已知且安全的状态。

因此，请看相同的例子，这次我们说明过程如何工作：

(1) 检查数据库中是否存在相应的客户，如果不存在，添加他/她；
(2) 提交客户信息；
(3) 检索客户的ID；
(4) 添加一行到orders表；
(5) 如果在添加行到orders表时出现故障，回退；
(6) 检索orders表中赋予的新订单ID；
(7) 对于订购的每项物品，添加新行到orderitems表；
(8) 如果在添加新行到orderitems时出现故障，回退所有添加的orderitems行和orders行。

252

在使用事务和事务处理时，有几个关键词汇反复出现。下面是关于

事务处理需要知道的几个术语：

- **事务（transaction）**指一组SQL语句；
- **回退（rollback）**指撤销指定SQL语句的过程；
- **提交（commit）**指将未存储的SQL语句结果写入数据库表；
- **保留点（savepoint）**指事务处理中设置的临时占位符（placeholder），你可以对它发布回退（与回退整个事务处理不同）。

26.2　控制事务处理

既然我们已经知道了什么是事务处理，下面讨论事务处理的管理中所涉及的问题。

管理事务处理的关键在于将SQL语句组分解为逻辑块，并明确规定数据何时应该回退，何时不应该回退。

253　T-SQL语句使用下面的方式标识事务处理块的开始：

输入

```
BEGIN TRANSACTION;
```

可以有选择地重新定义事务的名字。在同时处理多个事务的时候，这是十分有用的，可以明确定义回退时要提交的事务。

26.2.1　使用ROLLBACK

T-SQL的ROLLBACK命令用来回退（撤销）T-SQL语句，请看下面的语句（试着读懂这些语句，我知道这段代码有点难）：

输入

```
-- What is in orderitems?
SELECT * FROM orderitems;
-- Start the transaction
BEGIN TRANSACTION;
-- Delete all rows from orderitems
DELETE FROM orderitems;
-- Verify that they are gone
SELECT * FROM orderitems;
-- Now rollback the transaction
ROLLBACK;
```

```
-- And the deleted rows should all be back
SELECT * FROM orderitems;
```

分析 这个例子从显示ordertotals表的内容开始。首先执行一条SELECT以显示该表不为空。然后开始一个事务处理，用一条DELETE语句删除orderitems中的所有行。另一条SELECT语句验证orderitems确实为空。这时用一条ROLLBACK语句回退BEGIN TRANSACTION之后的所有语句，最后一条SELECT语句显示该表不为空。 254

显然，ROLLBACK只能在一个事务处理内使用（在执行一条BEGIN TRANSACTION命令之后）。

可以回退哪些语句？ 事务处理用来管理INSERT、UPDATE和DELETE语句。不能回退SELECT语句（回退SELECT语句也没有多少必要），也不能回退CREATE或DROP操作。事务处理中可以使用这些语句，但进行回退时，它们不被撤销。

26.2.2 使用COMMIT

一般的T-SQL语句都是直接针对数据库表执行和编写的。这就是所谓的自动提交（auto commit），即提交（写或保存）操作是自动进行的。

但是，在事务处理块中，提交不会隐含地进行。为进行明确的提交，使用COMMIT语句，如下所示：

输入

```
BEGIN TRANSACTION;
DELETE FROM orderitems WHERE order_num = 20010;
DELETE FROM orders WHERE order_num = 20010;
COMMIT;
```

分析 在这个例子中，从系统中完全删除订单20010。因为涉及更新两个数据库表orders和orderitems，所以使用事务处理块来保证订单不被部分删除。最后的COMMIT语句仅在不出错时写出更改。如果第一条DELETE起作用，但第二条失败，则DELETE不会提交。（实际上，它是被自动撤销的。）

这个例子在几个语句后使用了单个COMMIT。更复杂的例子通常在需 255

要时（且仅在需要时）使用多个ROLLBACK和COMMIT语句写出更改。

隐含的事务处理关闭　在执行一条COMMIT或ROLLBACK语句后，事务处理自动关闭（以后的更改将隐含地提交）。

26.2.3　使用保留点

简单的ROLLBACK和COMMIT语句就可以写入或撤销整个事务处理。但是，只是对简单的事务处理才能这样做，更复杂的事务处理可能需要部分提交或回退。

例如，前面描述的添加订单的过程为一个事务处理。如果发生错误，只需要返回到添加orders行之前即可，不需要回退到customers表（如果存在的话）。

为了支持回退部分事务处理，必须能在事务处理块中合适的位置放置占位符。这样，如果需要回退，可以回退到某个占位符。

这些占位符称为保留点。可以使用SAVEPOINT语句来创建一个保留点：

输入

```
SAVE TRANSACTION delete1;
```

保留点的名字　一般，每个保留点都取标识它的唯一名字，以便在回退时，SQL Server知道要回退到何处。然而，事实上保留点的名字可以重复使用，在这种情况下，SQL Server回退到该名字的最近一个保留点。

256

可以使用下面的语句回退到本例给出的保留点：

输入

```
ROLLBACK TRANSACTION delete1;
```

可以使用下面的语句回退到事务的开始：

输入

```
ROLLBACK TRANSACTION;
```

保留点越多越好 可以在T-SQL代码中设置任意多的保留点，越多越好。为什么呢？因为保留点越多，你就越能按自己的意愿灵活地进行回退。

26.2.4 更改自动提交的行为

正如所述，默认的SQL Server行为是自动提交所有更改。换句话说，任何时候你执行一条T-SQL语句，该语句实际上都是针对表执行的，而且所做的更改立即生效。为指示SQL Server不自动提交更改，需要使用以下语句：

输入

```
SET IMPLICIT_TRANSACTIONS ON;
```

257

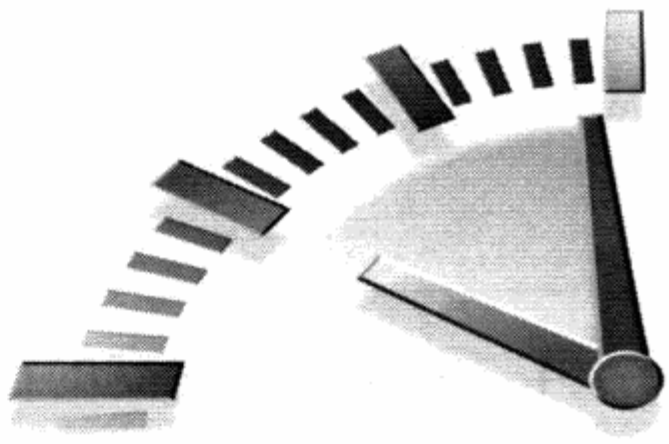
分析 IMPLICIT_TRANSACTIONS设置决定是否不需要COMMIT语句就自动提交更改。设置IMPLICIT_TRANSACTIONS为ON指示SQL Server不自动提交更改（直到IMPLICIT_TRANSACTIONS被设置为OFF为止）。

26.3 小结

本章介绍了事务处理是必须完整执行的SQL语句块。我们学习了如何使用COMMIT和ROLLBACK语句对何时写数据、何时撤销进行明确的管理。还学习了如何使用保留点对回退操作提供更强大的控制。

258

第27章



使用XML

本章学习如何从关系数据生成合式的XML，还学习XML数据类型及其应用。

27.1 SQL Server的XML支持

仅针对SQL Server 2005 本章介绍SQL Server 2005引入的功能，它们在以前的SQL Server版本中不能使用。

XML已经成了一个标准的机制，可通过它交换、分发和持久化存储（persist）数据。虽然SQL Server是一个关系DBMS，且关系数据与层次结构的XML数据很不相同，但是，经常有很充分的理由需要按XML获得SQL Server数据，并且在SQL Server表中存储XML数据。

关于SQL Server的XML支持，有3个主要方面值得注意：

- 使用SELECT检索数据，作为合式的XML返回检索出的数据；
- 在数据库表的特定列内存储合式的XML；
- 能够根据特定XML元素的内容搜索数据。

仅是基础 本章并不打算讲授XML本身。XML（包括使用XML数据的XPath、搜索XML数据的XQuery等的用法）是一个很广泛的内容，也是许多书籍的主题。如果你需要使用XML数据，应该花一定的时间更好地了解XML的数据结构以及如何使用它们。

27.2 检索为XML数据

保存在SQL Server表中的关系数据可检索为合式的XML，以便为XML客户机或你选择的应用系统的使用做好准备。为了检索为XML数据，可使用带FOR XML子句的SELECT。

下面举一个例子：

输入

```
SELECT vend_id, RTrim(vend_name) AS vend_name
FROM vendors
ORDER BY vend_name
FOR XML AUTO;
```

输出

```
<vendors vend_id="1003" vend_name="ACME" />
<vendors vend_id="1001" vend_name="Anvils R Us" />
<vendors vend_id="1004" vend_name="Furball Inc." />
<vendors vend_id="1005" vend_name="Jet Set" />
<vendors vend_id="1006" vend_name="Jouets Et Ours" />
<vendors vend_id="1002" vend_name="LT Supplies" />
```

分析 FOR XML指示SQL Server生成XML的输出。在生成XML时，通常需要定义所需XML的形式，但AUTO通过基于列和使用的顺序创建输出，简化了这项工作。这里，每个供应商作为独立的XML元素列出，表名vendors用作元素名，两个被选择的列用作属性。

260

下面是一个稍微复杂一点的例子：

输入

```
SELECT cust_name, orders.order_num, products.prod_id,
       prod_name
FROM customers, orders, orderitems, products
WHERE customers.cust_id=orders.cust_id
 AND orders.order_num=orderitems.order_num
 AND orderitems.prod_id=products.prod_id
ORDER BY cust_name, orders.order_num, products.prod_id
FOR XML AUTO;
```

输出

```
<customers cust_name="Coyote Inc.">
  <orders order_num="20005">
    <products prod_id="ANV01" prod_name=".5 ton anvil" />
    <products prod_id="ANV02" prod_name="1 ton anvil" />
    <products prod_id="FB" prod_name="Bird seed" />
    <products prod_id="TNT2" prod_name="TNT (5 sticks)" />
  </orders>
  <orders order_num="20009">
    <products prod_id="ANV03" prod_name="2 ton anvil" />
```

```
    <products prod_id="FB" prod_name="Bird seed" />
    <products prod_id="OL1" prod_name="Oil can" />
    <products prod_id="SLING" prod_name="Sling" />
  </orders>
</customers>
<customers cust_name="E Fudd">
  <orders order_num="20008">
    <products prod_id="FC" prod_name="Carrots" />
  </orders>
</customers>
<customers cust_name="Wascals">
  <orders order_num="20006">
    <products prod_id="JP2000" prod_name="JetPack 2000" />
  </orders>
</customers>
<customers cust_name="Yosemite Place">
  <orders order_num="20007">
    <products prod_id="TNT2" prod_name="TNT (5 sticks) " />
  </orders>
</customers>
```

261

分析　其中，联结4个表，返回顾客、每个顾客所下的订单和每个订单中的产品。ORDER BY子句中定义的列定义了生成的XML的形式，生成了包含一个或多个<orders>标记的顶层标记<customers>，<orders>标记又包含一个或多个<products>标记。

虽然AUTO可以生成所需的XML，但使用RAW格式可以进行更复杂一点的控制，如下所示：

输入

```
SELECT vend_id AS id, RTrim(vend_name) AS name
FROM vendors
ORDER BY vend_name
FOR XML RAW('vendor'), ROOT('vendors'), ELEMENTS;
```

输出

```
<vendors>
  <vendor>
    <id>1003</id>
    <name>ACME</name>
  </vendor>
  <vendor>
    <id>1001</id>
    <name>Anvils R Us</name>
  </vendor>
  <vendor>
    <id>1004</id>
    <name>Furball Inc.</name>
```

```
  </vendor>
  <vendor>
    <id>1005</id>
    <name>Jet Set</name>
  </vendor>
  <vendor>
    <id>1006</id>
    <name>Jouets Et Ours</name>
  </vendor>
  <vendor>
    <id>1002</id>
    <name>LT Supplies</name>
  </vendor>
</vendors>
```

262

分析 这个例子演示了几种有用的技术。ELEMENTS关键字导致列作为子元素嵌入，而不是作为标记属性嵌入。RAW允许指定行标记名，而ROOT用来指定顶层（根）标记名。此外，列别名用来明确地控制生成的元素名。

为了进行更强的控制，可使用EXPLICIT方式。EXPLICIT方式完全依赖你形成XML的形式，这样做允许你混合使用元素和属性，提供额外的嵌套层次等。下面是一个例子：

输入

```
SELECT 1 AS tag,
       NULL AS parent,
       vend_id AS [vendor!1!id],
       RTrim(vend_name) AS [vendor!1!name!ELEMENT]
FROM vendors
ORDER BY vend_name
FOR XML EXPLICIT, ROOT('vendors');
```

输出

```
<vendors>
  <vendor id="1003">
    <name>ACME</name>
  </vendor>
  <vendor id="1001">
    <name>Anvils R Us</name>
  </vendor>
  <vendor id="1004">
    <name>Furball Inc.</name>
  </vendor>
  <vendor id="1005">
    <name>Jet Set</name>
  </vendor>
  <vendor id="1006">
```

```
    <name>Jouets Et Ours</name>
  </vendor>
  <vendor id="1002">
    <name>LT Supplies</name>
  </vendor>
</vendors>
```

263

分析 如果使用EXPLICIT，则SELECT语句必须定义一个tag和一个parent，然后才是被处理的列。每个列必须为tagname!tagid!attributename格式。此外，每个列可以定义可选的属性，就像vend_name的情形一样（vend_name被重命名为name并定义为一个ELEMENT）。

XPath为另一种选择　XPath是一种用来访问XML数据的语言。在生成XML时，为了使能力与简单性理想结合，你可能希望使用FOR XML PATH方式，它允许定义XPath表达式来构造输出。遗憾的是，XPath的介绍超出了本书的范围。

27.3　存储XML数据

XML数据是包含严格控制的嵌套标记的文本。但是，XML仍然仅仅是一个文本串。因此，如果你需要将XML数据保存到一个表中，可以使用以某种文本数据类型（首选变长数据类型）定义的列。

但是，这样做并不理想。首先，如果列允许存储文本，那么甚至非XML文本也可以存储在表中，这样可能会严重地妨碍后续的XML处理。此外，处理和检索作为串存储的XML数据很不理想，因为SQL Server不能区分XML和存储在其中的数据。

由于这些原因以及其他一些原因，SQL Server 2005引入了一个新数据类型——XML。以下是使用这种数据类型的几点好处：

- XML列仅接受合式的XML，非XML的其他内容不能存储在XML类型的列中；
- 可针对某种XML模式（一个文档，它描述特定XML规范的格式）检验XML列中的数据；

- 可使用XQuery搜索来搜索XML列中的数据。

264

XQuery XQuery是一种用于XML的查询语言。你可以这样认为：SQL用于关系数据库，而XQuery用于XML数据。本书不介绍XQuery。

下面是具有XML数据类型的列的一个表的例子：

输入

```
CREATE TABLE MyXMLTable
(
   id   INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
   data XML  NOT NULL
);
```

分析 正如第20章所述，CREATE TABLE用来创建数据库的新表。这条CREATE TABLE语句创建具有两个列的一个简单的表，其中第二个列为XML类型。

在创建表之后，可使用INSERT将行添加到表中。但是，要注意保证添加的XML在语法上是正确的。

输入

```
INSERT INTO MyXMLTable(data)
VALUES(
'<state abbrev="CA">
  <city name="Los Angeles" />
  <city name="San Francisco" />
</state>');

INSERT INTO MyXMLTable(data)
VALUES(
'<state abbrev="IL">
  <city name="Chicago" />
</state>');
```

265

```
INSERT INTO MyXMLTable(data)
VALUES(
'<state abbrev="NY">
  <city name="New York" />
 </state>');
```

分析 这里给表添加了3行，每个值都作为一个文本串。因为相应的文本是语法上合法的XML，插入成功。

一般来说，把包含XML的串转换为实际的XML数据类型是一种很好

的办法。下面举一个例子：

输入

```
INSERT INTO MyXMLTable(data)
VALUES(
Cast('<state abbrev="CA">
  <city name="Los Angeles" />
  <city name="San Francisco" />
</state>' AS XML));

INSERT INTO MyXMLTable(data)
VALUES(
Cast('<state abbrev="IL">
  <city name="Chicago" />
</state>' AS XML));

INSERT INTO MyXMLTable(data)
VALUES(
Cast ('<state abbrev="NY">
  <city name="New York" />
 </state>' AS XML));
```

分析

这个例子插入相同的数据，但首先使用Cast()函数把每个串转换为XML。

266

使用XML模式　正如所述，XML模式是一个文档，它定义特定的XML格式。XML模式可在SQL Server中用CREATE XML SCHEMA来定义。在定义之后，可将这些模式与XML数据类型的表列相关联。这样，不管什么时候存储XML数据，SQL Server都不仅保证它是合式的，而且还针对模式检验它以保证只存储合法的XML。

27.4　XML数据的搜索

XML数据的搜索要求使用XQuery。XML字段可用以下XML数据类型的方法（或函数）访问：

- exist用来检查某个XQuery表达式是否存在；
- modify用来修改XML内容；
- nodes用来把XML记录分解为多个SQL行；
- query用来从XML记录提取特定的元素；
- value用来从一个XML记录返回SQL类型。

XQuery不在本书中介绍。但是，为了给你提供基于XQuery的操作的概念，下面举两个例子：

输入

```
SELECT data.query('/state/city')
FROM MyXMLTable;
```

分析 这个例子从前面存储的XML数据中提取城市名。/state/city表示“找出一个<state>标记，然后在其中找出名为<city>的子标记”。显然，如果使用简单的串处理，这种类型的数据提取将会更复杂。

267

输入

```
SELECT *
FROM MyXMLTable
WHERE data.exist('/state/city[@name="Chicago"]') = 1;
```

分析 这个例子寻找包含具有名为Chicago的子<city>标记的<state>标记的行。前面插入的行中只有一行匹配这个表达式，因此只返回它。

正如所见，一旦数据作为XML数据存储（在一个XML数据类型的列中），就可以用XPath和XQuery来执行非常复杂的数据操纵了。

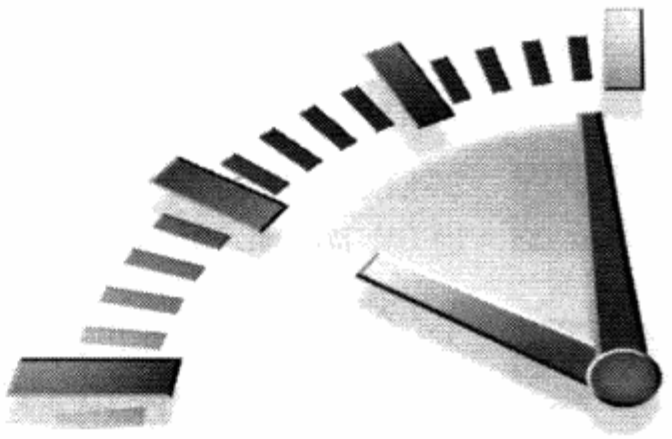
更多知识 为了解更多的XPath知识，请访问http://www.w3.org/TR/xpath。为了解更多的XQuery知识，请访问http://www.w3. org/XML/Query/。

27.5 小结

本章学习如何基于所选择的数据生成XML输出。我们还向读者介绍了XML数据类型，举了如何存储和访问XML数据的例子。

268

第28章



全球化和本地化

本章介绍SQL Server处理不同字符集和语言的基础知识，介绍如何在你的T-SQL代码中使用不同的字符集。

28.1 字符集和校对顺序

数据库表被用来存储和检索数据。不同的语言和字符集需要以不同的方式存储和检索。因此，SQL Server需要适应不同的字符集（不同的字母和字符），适应不同的排序和检索数据的方法。

在讨论多种语言和字符集时，将会遇到以下重要术语。

- 字符集：字母和符号的集合。
- 编码：某个字符集成员的内部表示。
- 校对：规定字符如何比较的指令。

校对为什么重要 排序英文文本很容易吗？或许未必。考虑词APE、apex和Apple。它们处于相同的排序顺序吗？这有赖于你是否想区分大小写。使用区分大小写的校对顺序，这些词有一种排序方式，使用不区分大小写的校对顺序有另外一种排序方式。这不仅影响排序（如用ORDER BY排序数据），还影响搜索（例如，寻找apple的WHERE子句是否能找到APPLE）。在使用诸如法文à或德文ō这样的字符时，情况更复杂，在使用非基于拉丁文的字符集（日文、希伯来文、俄文等）时，情况更为复杂。

28.2 使用校对顺序

SQL Server支持众多的字符集。为查看服务器所支持字符集的完整列表，使用特殊的fn_helpcollations()，如下所示：

输入

```
SELECT * FROM fn_helpcollations();
```

分析 fn_helpcollations返回所有可用的校对顺序列表，每个都有名字和描述。可以使用WHERE子句来过滤这个列表，找出你要查找的校对顺序。

可以看到，某些字符集具有不止一种校对顺序。例如，Latin1 General有许多变种，而且许多变种出现两次，一次区分大小写（由_CS表示），一次不区分大小写（由_CI表示）。

270

在安装SQL Server时，定义一个默认的校对顺序。你可以使用ServerProperty()函数查看默认的校对顺序，如下所示：

输入

```
SELECT ServerProperty('Collation') AS Collation;
```

输出

```
Collation
-----------------------------
SQL_Latin1_General_CP1_CI_AS
```

分析 这条语句显示当前默认的服务器校对顺序。（你自己的服务器显示的校对顺序可能与这里显示的不同。）

可以回忆一下第1章的内容，在创建一个数据库时，为此数据库定义一个默认的校对顺序。除非明确地指定，否则此校对顺序与服务器的默认校对顺序相同。

为了确定对特定数据库定义的校对顺序，可使用服务器函数DatabasePropertyEX()：

输入

```
SELECT DatabasePropertyEX('Crash Course', 'Collation')
       AS Collation;
```

分析 此语句显示指定数据库的默认校对顺序。

为了获得指定校对顺序的更多信息，可使用前面提过的fn_helpcollations()，如下所示：

271

输入

```
-- Get collation name
DECLARE @Collation VARCHAR(100);
SELECT @Collation = CONVERT(VARCHAR(100),
                            ServerProperty('Collation'))

-- Get description
SELECT description
FROM fn_helpcollations()
WHERE name = @Collation;
```

分析 如下所示，这个例子首先获得服务器的校对顺序，然后使用fn_helpcollations()获得校对顺序的描述。

实际上，字符集很少是服务器范围（甚至数据库范围）的设置。不同的表列可能需要不同的字符集，而且可以在创建表时指定。

为了给表指定字符集和校对，可使用带额外子句的CREATE TABLE（参见第20章的讨论）：

输入

```
CREATE TABLE mytable
(
   column1   INT,
   column2   VARCHAR(10) COLLATE Hebrew_CI_AI
);
```

分析 这条语句创建了一个两列的表，并为其中一列指定了一个校对顺序。

更改校对顺序　可以在表创建后用ALTER TABLE更改校对顺序。

272

28.3　区分大小写

校对顺序的一个非常重要的用途是决定搜索和排序时是否区分大小写。默认的校对顺序通常使搜索和排序不区分大小写。如果某个特定的列总是需要区分大小写的搜索和排序，则可以定义该列使用区分大小写的校对顺序：

输入

```
CREATE TABLE mytable
(
   column1   INT,
```

```
    column2   VARCHAR(10) COLLATE SQL_Latin1_General_CP1_CS_AS
);
```

如果你需要用与表创建时不同的校对顺序排序特定的SELECT语句的结果，可以使用如下的SELECT语句：

输入

```
SELECT * FROM customers
ORDER BY cust_name COLLATE SQL_Latin1_General_CP1_CS_AS;
```

分析 这条SELECT语句使用COLLATE指定了另一个校对顺序（在此例子中为一个区分大小写的校对顺序）。这显然会影响结果排序的顺序。

偶然区分大小写 刚才给出的SELECT语句演示了一个有用的技术，它在一个一般不区分大小写的表上进行区分大小写的搜索。当然，反之也是可行的。

也可以在WHERE子句中使用另一种校对顺序，更改区分大小写的搜索。下面是一个简单的通配符搜索：

输入

```
SELECT cust_id, cust_name
FROM customers
WHERE cust_name LIKE '%E%'
```

输出

```
cust_id     cust_name
----------- --------------
10001       Coyote Inc.
10002       Mouse House
10004       Yosemite Place
10005       E Fudd
10006       Pep E. LaPew
10007       Pep E. LaPew
```

分析 这条SELECT语句检索cust_name中有一个E的所有行，不管E是大写还是小写。

下面是一个区分大小写的搜索：

输入

```
SELECT cust_id, cust_name
FROM customers
WHERE cust_name COLLATE SQL_Latin1_General_CP1_CS_AS LIKE '%E%'
```

输出

```
cust_id     cust_name
----------- --------------
10005       E Fudd
```

```
10006        Pep E. LaPew
10007        Pep E. LaPew
```

274

分析 这条SELECT使用COLLATE指定一个区分大小写的校对顺序，因此仅检索出cust_name中有大写E的行。

SELECT语句的其他COLLATE子句 除了如这里所示的用在ORDER BY子句中外，COLLATE还可以与GROUP BY、HAVING、聚集函数、别名等一起使用。

28.4 使用Unicode

统一字符编码（Unicode）是一种机制，通过它引用和存储不同的字符集（特别是非基于拉丁文的字符集）。在SQL Server中使用Unicode文本需要满足以下条件：

- 为了成功地在SQL Server表中存储这些字符，列必须为可以存储Unicode文本的某种类型；
- 应该使用函数的Unicode版本，如使用NChar()而不是Char()；
- 任何时候将Unicode文本传递给一条T-SQL语句，都必须指明它为Unicode的。

SQL Server定义了3种特殊数据类型，专门用来存储Unicode文本，它们分别是：NCHAR（CHAR的对应Unicode类型）、NVARCHAR（VARCHAR的对应Unicode类型）和NTEXT（TEXT的对应Unicode类型）。

Unicode数据类型有助于未来的发展 即使你目前认为不需要非基于拉丁文的文本，也可以用Uncode类型定义文本列，以便能在以后需要时支持这些语言。

在列被定义为支持Unicode文本之后，你仍然需要告诉SQL Server以Unicode使用相应的文本，而且每当你传递串给T-SQL时都必须这样做。

275

考虑以下几个例子：

输入

```
INSERT INTO mytable(column1, column2)
VALUES(1000, 'שלום');
SELECT * FROM mytable;
```

输出

```
column1     column2
----------- ----------
1000        ????
```

分析 这条INSERT语句插入具有非拉丁文（这个例子中为希伯来文）文本的一行到前面创建的mytable表。但在检索时，此正文显示为????。换句话说，SQL Server不承认它是Unicode，因此没有正确存储它。

以下是正确的方法：

输入

```
INSERT INTO mytable(column1, column2)
VALUES(1000, N'שלום');
SELECT * FROM mytable;
```

输出

```
column1     column2
----------- ----------
1000        שלום
```

分析 这条INSERT语句插入相同文本，但用N作为串的前缀，告诉SQL Server把后跟的文本视为Unicode对待。这样，数据被正确存储，且后面的SELECT检索出正确的数据。

276

请记住N前缀 任何时候使用Unicode串都必须指定N前缀，不管它是传递给INSERT或UPDATE的文本，还是传递给WHERE子句中使用的串都是如此。

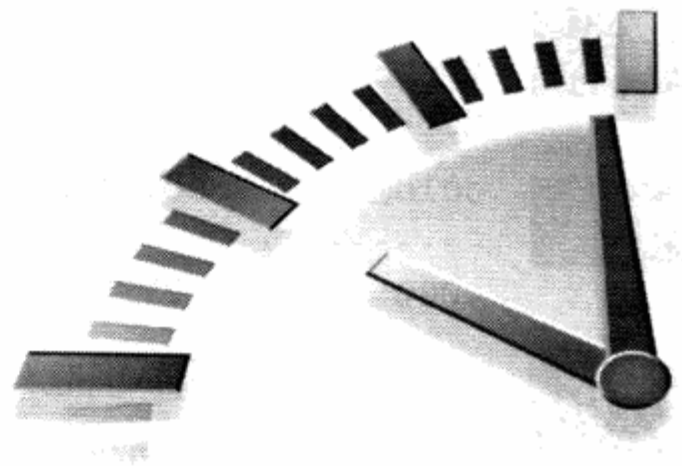
28.5 小结

本章学习了字符集和校对顺序的知识，还学习了如何为特定的表和列定义字符集和校对顺序，如何在需要时使用另一种校对顺序，以及如何处理Unicode文本。

277

第29章

安全管理



数据库服务器通常包含关键的数据，确保这些数据的安全和完整需要利用访问控制。本章将学习SQL Server的访问控制和用户管理。

29.1 访问控制

SQL Server服务器的安全基础是：用户应该对他们需要的数据具有适当的访问权，既不能多也不能少。换句话说，用户不能对过多的数据具有过多的访问权。

考虑以下内容：

- 多数用户只需要对表进行读和写，但少数用户甚至需要能创建和删除表；
- 某些用户需要读表，但可能不需要更新表；
- 你可能想允许用户添加数据，但不允许他们删除数据；
- 某些用户（管理员）可能需要处理用户账号的权限，但多数用户不需要；
- 你可能想让用户通过存储过程访问数据，但不允许他们直接访问数据；
- 你可能想根据用户登录的地点限制对某些功能的访问。

这些都只是例子，但有助于说明一个重要的事实，即你需要给用户提供他们所需的访问权，且仅提供他们所需的访问权。这就是所谓的访问控制，管理访问控制需要创建和管理用户账号。

使用管理工具 SQL Server工具（在第2章中描述）提供了一个图形用户界面，可用来管理用户及账号权限。这些工具在内部利用本章介绍的语句，使你能交互地、方便地管理访问控制。

回忆一下第3章的内容，我们知道，为了执行数据库操作，需要登录SQL Server。SQL Server创建一个名为sa（为System Administrator）的用户账号，它对整个SQL Server具有完全的控制。你可能已经在本书各章的学习中使用sa进行过登录，在对非现实的数据库试验SQL Server时，这样做很好。不过在现实世界的日常工作中，决不能使用sa。应该创建一系列的账号，有的用于管理，有的供用户使用，有的供开发人员使用，等等。

防止无意的错误 重要的是注意到，访问控制的目的不仅仅是防止用户的恶意企图。数据梦魇更为常见的是无意识错误的结果，如错打T-SQL语句，在不合适的数据库中操作或其他一些用户错误。通过保证用户不能执行他们不应该执行的语句，访问控制有助于避免这些情况的发生。

不要使用sa 应该严肃对待sa登录的使用。仅在绝对需要时使用它（或许在你不能登录其他管理账号时使用）。不应该在日常的SQL Server操作中使用sa。

280

29.2 管理用户

Windows还是SQL Server? SQL Server支持两种形式的登录和账号。它可以使用自己的用户和账号列表，或者启用由基础的Windows操作系统（或Windows域，如果服务器为域的成分）管理的账号。

如本节所述，用户管理仅适用于SQL Server登录，不适用于

> Windows登录。如果使用的是Windows登录，则账号创建、删除、口令更改等都应该利用Windows管理工具进行。但是，访问控制和权限应该在SQL Server中进行。

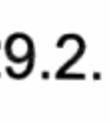
SQL Server用户账号和信息存储在内部SYS数据库中。你一般不需要直接访问SYS数据库和表。SQL Server为你提供了一系列操纵SYS表的存储过程。

例如，为了获得登录的一个列表，可以从SYS表中检索数据，或者使用sp_helplogins存储过程：

输入

```
EXEC sp_helplogins;
```

分析　sp_helplogins返回两个结果集。第一个列出每个登录及其信息，第二个列出与登录关联的用户。



列出特定的登录　sp_helplogins列出所有登录。为了显示某个具体的登录，可将登录名作为参数传递给此存储过程。

281

29.2.1　创建用户账号

为了创建一个新用户登录，使用CREATE LOGIN语句，如下所示：

输入

```
CREATE LOGIN BenF WITH PASSWORD = 'P@$$w0rd';
```

分析　CREATE LOGIN创建一个新用户登录。在创建用户账号时不一定需要口令，不过这个例子给出了一个口令。

如果你再次列出用户账号，将会在输出中看到新账号。

SQL Server 2005之前　CREATE LOGIN语句是在SQL Server 2005中引入的。如果你使用的是SQL Server的早期版本，则应该代之以使用sp_addlogin存储过程，如下所示：

```
EXEC sp_addlogin 'BenF', ' P@$$w0rd ';
```

29.2.2 删除用户账号

为了删除一个用户登录（以及相关的权限），使用DROP LOGIN语句，如下所示：

输入
```
DROP LOGIN BenF;
```

29.2.3 启用和禁用账号

为了禁用一个账号（不删除它），使用ALTER LOGIN，如下所示： 282

输入
```
ALTER LOGIN BenF DISABLE;
```

为了启用一个账号，如下进行：

输入
```
ALTER LOGIN BenF ENABLE;
```

29.2.4 重命名登录

为了重新命名一个登录，使用ALTER TABLE，如下所示：

输入
```
ALTER LOGIN BenF WITH NAME = BenForta;
```

29.2.5 更改口令

为了更改用户口令，使用ALTER LOGIN，如下所示：

输入
```
ALTER LOGIN BenF WITH PASSWORD = 'n3w p@$$w0rd';
```

29.3 管理访问权限

在创建一个登录后，你需要指定它对数据库、表和功能具有什么样的访问权。完成这些工作需要设置和管理访问权限。

参阅SQL Server文档 可授予权限的完整列表超出了本书的范围。本书不给出任何一种SQL Server版本的权限的完整列表。SQL Server 2005极大地增加了可授予的权限以及权限的详细程度。关于这里所介绍的基础知识，请参阅SQL Server文档。 283

29.3.1　设置访问权限

为设置权限，使用GRANT语句。GRANT要求你至少给出以下信息：

- 要授予的权限；
- 被授予访问权限的数据库或表；
- 用户名。

以下例子给出GRANT的用法：

输入

```
GRANT CREATE TABLE TO BenF;
```

分析　此GRANT允许用户使用CREATE TABLE。通过只授予CREATE TABLE访问权限，用户BenF能够创建新表，但不能修改或删除已有的表。

允许授出被授予的权限　在GRANT时，你可以指定WITH GRANT OPTION子句，从而允许用户把相同的访问权限授予别的用户。

29.3.2　删除访问权限

GRANT的反操作为REVOKE，用它来撤销特定的权限。下面举一个例子：

输入

```
REVOKE CREATE TABLE FROM BenF;
```

分析　这条REVOKE语句取消刚赋予用户BenF的CREATE TABLE访问权限。

284

GRANT和REVOKE可在几个层次上控制访问权限：

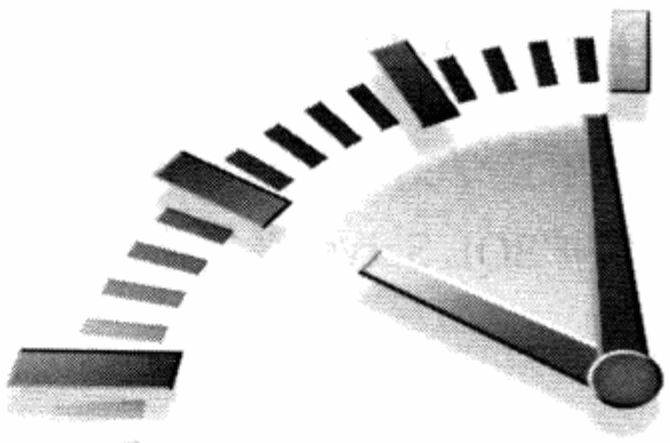
- 整个服务器；
- 整个数据库；
- 特定的表；
- 特定的列；
- 特定的存储过程。

29.4 小结

本章学习了通过管理用户登录、赋予用户特殊的权限进行访问控制和保护SQL Server服务器。

第30章

改善性能



本章将复习与SQL Server性能有关的某些要点。

30.1 改善性能

数据库管理员把他们生命中的相当一部份时间花在了调整、试验以改善DBMS性能之上。在诊断应用的滞缓现象和性能问题时，性能不良的数据库（以及数据库查询）通常是最常见的祸因。

可以看出，下面的内容并不能完全决定SQL Server的性能。我们只是想回顾一下前面各章的重点，提供进行性能优化探讨和分析的一个出发点。

- 首先，SQL Server（与所有DBMS一样）具有特定的硬件建议。在学习和研究SQL Server时，使用任何旧的计算机作为服务器都可以。但对用于生产的服务器来说，应该坚持遵循这些硬件建议。
- 一般来说，关键的生产DBMS应该运行在自己的专用服务器上。
- SQL Server是用一系列的默认设置预先配置的，从这些设置开始通常是很好的。但过一段时间后你可能需要调整内存分配、缓冲区大小等。

使用SQL Server 2005调整顾问 SQL Server 2005带有一个名为Database Engine Tuning Advisor的很好的实用程序，它能从Microsoft SQL Server Management Studio（在Tools菜单下）中执行。这个实用程序可以分析你的数据库和表，做出关于索引、分区等的建议。这个工具是专门为不想了解SQL Server复杂的内部结构的用户设计的。应该充分利用它。

- SQL Server一个多用户多线程的DBMS，换言之，它经常同时执行多个任务。如果这些任务中的某一个执行缓慢，则所有请求都会执行缓慢。你可以使用Windows Systems Monitor监视SQL Server的磁盘以及内存使用、关键事件的更改等。
- 总是有不止一种方法编写同一条SELECT语句。应该试验联结、并、子查询等，找出最佳的方法。
- 在SQL Server处理T-SQL语句时，SQL Server试图优化T-SQL，把请求适当地分解为更小的请求，使用索引，等等。理解SQL Server所做的工作，能够确定批量语句或存储过程的特定部分所花的处理时间，这些对于优化性能极为重要。SQL Server能报告已提交的SQL语句使用的执行计划。这个选项可在Microsoft SQL Server Management Studio（如果使用的是SQL Server 2005）和Enterprise Manager（如果使用的是SQL Server的早期版本）中得到。
- 一般来说，存储过程执行得比一条一条地执行其中的各条SQL Server语句快。
- 应该总是使用正确的数据类型。
- 决不要检索比需求还要多的数据。换言之，不要用SELECT *（除非你真正需要每个列）。

288

- 必须索引数据库表以改善数据检索的性能。确定索引什么不是一件微不足道的任务，需要分析使用的SELECT语句以找出重复的WHERE和ORDER BY子句。如果一个简单的WHERE子句返回结果所花的时间太长，则可以断定其中使用的列（或几个列）就是需要索引的对象。
- 你的SELECT语句中有一系列复杂的OR条件吗？通过使用多条

SELECT语句和连接它们的UNION语句，你能看到极大的性能改进。

- 索引改善数据检索的性能，但损害数据插入、删除和更新的性能。如果你有一些表，它们收集数据且不经常被搜索，则在有必要之前不要索引它们。（索引可根据需要添加和删除。）
- LIKE很慢。一般来说，最好是使用FREETEXT或CONTAINS进行全文本搜索。
- 数据库是不断变化的实体。一组优化良好的表一会儿后可能就面目全非了。由于表的使用和内容的更改，理想的优化和配置也会改变。
- 最重要的规则就是，每条规则在某些条件下都会被打破。

浏览文档　与SQL Server一起安装的SQL Server文档有许多提示和技巧，而且很便于搜索（这些搜索还可以自动包含联机资源）。一定要查看这些非常有价值的资料。

30.2　小结

本章回顾了与SQL Server性能有关的某些提示和说明。当然，这只是一小部分，不过，既然你已经完成了本书的学习，你应该能试验和掌握自己觉得最适合的内容。

289

附录A

SQL Server和T-SQL入门

如果你是SQL Server和T-SQL的初学者，本附录是一些需要知道的基本知识。

A.1 你需要什么

为使用SQL Server和学习本书中各章的内容，你需要访问SQL Server服务器和客户机应用（用来访问服务器的软件）副本。

你不一定需要自己安装SQL Server副本，但需要访问服务器。基本上有下面两种选择。

- 获得某个（或许是你的公司或许是商用的或院校的）SQL Server服务器的访问权。为使用这个服务器，你需要得到一个服务器账号（一个登录名和一个口令）。
- 下载SQL Server Express的一个免费副本，安装在你自己的计算机上（SQL Server只运行在Windows上）。

如果条件允许，安装一个本地服务器 为了得到完全的控制，包括访问你使用别人的SQL Server服务器可能得不到授权的命令和特性，你应该安装自己的本地服务器。即使你的最终生产DBMS不使用你自己的服务器，你也能从对服务器必须提供的所有功能具有完全的无约束的访问中受益。

不管是否使用本地服务器，你都需要客户机软件（用来实际运行SQL Server命令的程序）。最好选择使用SQL Server所带的客户机程序：

- 如果使用的是SQL Server 2005，可使用SQL Server Management Studio；
- 如果使用的是SQL Server的早期版本，可使用SQL Enterprise Manager（它包含一个名为Query Analyzer的工具，可在其中执行T-SQL查询）。

A.2　获得软件

为了解更多的SQL Server知识，请访问http://www.microsoft.com/sql/，此页面包含试用软件和其他下载的链接。SQL Server 2005的试用软件版本完全是产品化的，不过只能运行180天。SQL Server 2005 Express缺少某些高级功能，但它的下载和使用是完全免费的。

SQL Server Express　虽然SQL Server Express缺少SQL Server商业版本中的某些更高级的功能，但这不会影响你对本书的学习。本书中所有章节都将使用SQL Server Express。

A.3　安装软件

SQL Server的安装很简单，安装向导会指导你完成安装过程，它包括以下选项：

292

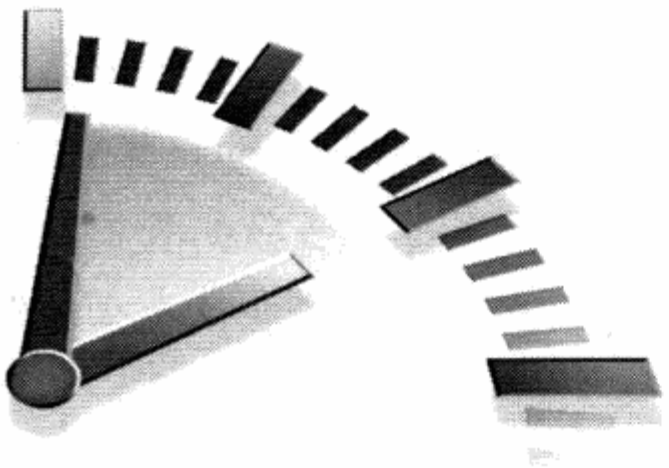
- 设置安装位置（一般采用默认就可以）；
- 选择sa用户的口令；
- 安装文档（强力推荐）；
- 选择许多其他选项（一般都可以采用默认值）。

A.4　各章准备

第3章说明在安装了SQL Server后如何登录和退出服务器，如何执行命令。

本书各章将使用真实的SQL Server语句和真实的数据。附录B描述了本书中使用的样例表，说明了如何获得和使用表创建和填充的脚本。

293



附录B 样例表

编写SQL语句需要对基础数据库的设计有良好的理解。不知道什么信息存储在什么表中，表之间如何相互关联以及行内数据如何分解，是不可能编写出高效的SQL的。

建议你实际试验本书中每章的每个例子。各章都使用相同的一组数据文件。为帮助你更好地理解这些例子和掌握各章介绍的内容，本附录描述了所用的表、表之间的关系以及如何获得它们。

B.1 样例表

本书中使用的样例表为一个想象的随身物品推销商使用的订单录入系统，这些随身物品可能是你喜欢的卡通人物需要的（是的，卡通人物，没人规定学习T-SQL必须沉闷地学）。这些表用来完成以下几个任务：

- 管理供应商；
- 管理产品目录；
- 管理顾客列表；
- 录入顾客订单。

要完成这几个任务需要作为关系数据库设计成分的紧密联系的6个表。以下几节描述各个表。

简化的例子　这里使用的表并不完整。现实中的订单录入系统必须记录这里没有包含的大量其他数据(如,报酬和记账信息、发货跟踪信息等)。不过，这些表演示了你在多数安装中会遇到的各种数据的组织和关系。你可以把这些方法和技术应用到自己的数据库中。

表的描述

以下介绍6个表以及每个表中的列。

表的列出顺序　6个表之所以要用这里的次序列出是因为它们之间的依赖关系。因为products表依赖于vendors表，所以先列出vendors，其他表的列出也有类似的关系。

vendors表

vendors表存储销售产品的供应商。每个供应商在这个表中有一个记录，供应商ID（vend_id）列用来匹配产品和供应商。

表B-1　vendors表列

列	说　明
vend_id	唯一的供应商ID
vend_name	供应商名
vend_address	供应商的地址
vend_city	供应商的城市
vend_state	供应商的州
vend_zip	供应商的邮政编码
vend_country	供应商的国家

296

- 所有表都应该有主键。这个表使用vend_id作为主键。vend_id为一个标识字段。

products表

products表包含产品目录，每行一个产品。每个产品有唯一的ID

（prod_id列），通过vend_id（供应商的唯一ID）关联到它的供应商。

表B-2 products表的列

列	说 明
prod_id	唯一产品ID
vend_id	产品供应商ID（关联到vendors表中的vend_id）
prod_name	产品名
prod_price	产品价格
prod_desc	产品描述

- 所有表都应该有一个主键，这个表用prod_id作为其主键。
- 为实施引用完整性，应该在vend_id上定义一个外键，关联到vendors的vend_id。

customers表

customers表存储所有顾客的信息。每个顾客有唯一的ID（cust_id列）。

表B-3 customers表的列

列	说 明
cust_id	唯一数值的顾客ID
cust_name	顾客名
cust_address	顾客的地址
cust_city	顾客的城市
cust_state	顾客的州
cust_zip	顾客的邮政编码
cust_country	顾客的国家
cust_contact	顾客的联系名
cust_email	顾客的联系email地址

297

- 所有表都应该定义主键，这个表将使用cust_id作为它的主键。cust_id是一个标识字段。

orders表

orders表存储顾客订单（但不是订单细节）。每个订单唯一地编号（order_num列）。订单用cust_id列（它关联到customer表的顾客唯一

ID）与相应的顾客关联。

表B-4　orders表列

列	说　明
order_num	唯一订单号
order_date	订单日期
cust_id	订单顾客ID（关系到customers表的cust_id）

- 所有表都应该定义主键，这个表使用order_num作为它的主键。order_num是一个标识字段。
- 为实施引用完整性，应该在cust_id上定义一个外键，关联到customers的cust_id。

orderitems表

orderitems表存储每个订单中的实际物品，每个订单的每个物品占一行。对orders中的每一行，orderitems中有一行或多行。每个订单物品由订单号加订单物品（第一个物品、第二个物品等）唯一标识。订

298 单物品通过order_num列（关联到orders中订单的唯一ID）与它们相应的订单相关联。此外，每个订单项包含订单物品的产品ID（它关联物品到products表）。

表B-5　orderitems表的列

列	说　明
order_num	订单号（关联到orders表的order_num）
order_item	订单物品号（在某个订单中的顺序）
prod_id	产品ID（关联到products表的prod_id）
quantity	物品数量
item_price	物品价格

- 所有表都应该有主键，这个表使用order_num和order_item作为其主键。
- 为实施引用完整性，应该在order_num上定义外键，关联它到orders的order_num，在prod_id上定义外键，关联它到products的prod_id。

productnotes表

productnotes表存储与特定产品有关的注释。并非所有产品都有相关的注释，而有的产品可能有许多相关的注释。

表B-6 productnotes表的列

列	说 明
note_id	唯一注释ID
prod_id	产品ID（对应于products表中的prod_id）
note_date	增加注释的日期
note_text	注释文本

- 所有表都应该有主键，这个表应该使用note_id作为其主键。
- 为了实现引用完整性，应该在prod_id上定义外键，关联它到products表的prod_id。

299

B.2 创建样例表

为了学习各个例子，需要一组填充了数据的表。所需要获得和运行的一切东西都可以在http://www.forta.com/books/0672328674/上找到。

此网页包含两个可以下载的SQL脚本文件。

- create.sql包含创建6个数据库表（包括所有主键和外键约束）的T-SQL语句。
- populate.sql包含用来填充这些表的INSERT语句。

仅对于SQL Server 可下载的.sql文件中的SQL语句是DBMS专用的，它们仅用于Microsoft SQL Server。

这两个脚本用SQL Server 2000和SQL Server 2005进行了广泛的测试，但没有用更早的SQL Server版本进行测试。

在下载了脚本后，可用它们创建和填充本书各章所用的表。以下是要遵循的步骤。

(1) 创建一个名为crashcourse的新数据库（或者其他任意名称，

为安全考虑，不要使用已有的数据库名称）。如果使用的是SQL Server 2005，最简单的办法是使用Microsoft SQL Server Management Studio（第2章中描述）；如果使用的是SQL Server 2000，可以用SQL Enterprise Manager。

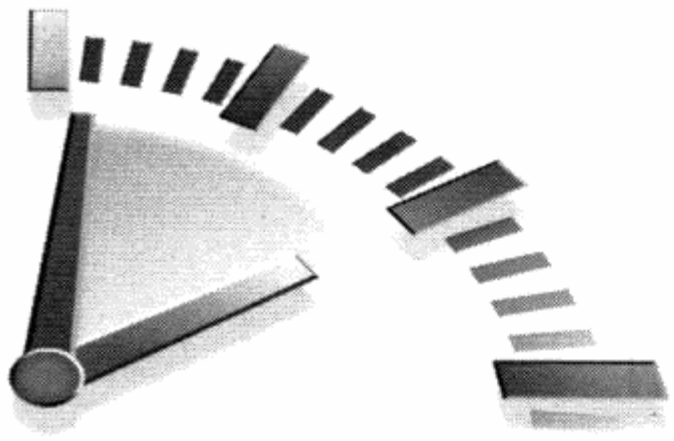
(2) 保证选择新数据库（用USE命令或从下拉列表中选择数据库）。如
300 果使用的是Microsoft SQL Server Management Studio，就可以在这个工具的内部完成；如果使用的是SQL Server 2000，可以用SQL Query Analyzer。

(3) 执行create.sql脚本。你可以简单地复制和粘贴此文件的整个内容到查询窗，或者可以使用File菜单选项直接打开create.sql。（如果使用SQL Server 2005，你应该能在Microsoft SQL Server Management Studio中双击具有.sql扩展名的文件来打开它们。）

(4) 重复前面的步骤，用populate.sql文件填充各个新表。

这样之后就做好了准备。

创建，然后填充　必须在运行表填充脚本之前运行表创建脚本。一定要查看这些脚本返回的错误消息。如果创建脚本失败，
301 则在进行表填充之前需要解决可能存在的问题。



附录C

T-SQL语句的语法

为帮助读者在需要时找到相应语句的语法，本附录列出了最常使用的T-SQL语句的语法。每条语句以简要的描述开始，然后给出它的语法。为增加方便性，还给出对讲授相应语句的章的交叉引用。

在阅读语句语法时，应该记住以下约定。

- |符号用来指出几个选择中的一个，因此，NULL | NOT NULL表示或者给出NULL或者给出NOT NULL。
- 包含在方括号中的关键字或子句（如[like this]）是可选的。
- 既没有列出所有的T-SQL语句，也没有列出每一条子句和选项。

C.1 BEGIN TRANSACTION

BEGIN TRANSACTION用于开始一个新的事务处理块，详细信息请参阅第26章。

输入

```
BEGIN TRANSACTION;
```


C.2 ALTER TABLE

ALTER TABLE用来更新已存在表的模式。为了创建新表，应该使用CREATE TABLE。详细信息请参阅第20章。

输入

```
ALTER TABLE tablename
(
    ADD         column          datatype    [NULL|NOT NULL]
                                            [CONSTRAINTS],
```

```
        ALTER       column columns   datatype    [NULL¦NOT NULL]
                                                 [CONSTRAINTS],
        DROP        column,
        ...
    );
```

C.3 COMMIT TRANSACTION

COMMIT TRANSACTION用来向数据库中写入事务。详细信息请参阅第26章。

输入

```
COMMIT TRANSACTION;
```

C.4 CREATE INDEX

CREATE INDEX用于在一个或多个列上创建索引。详细请参阅第20章。

输入

```
CREATE INDEX indexname
ON tablename (column [ASC¦DESC], ...);
```


C.5 CREATE LOGIN

CREATE LOGIN用子向系统中添加新的用户账户。详细请参阅第29章。

输入

```
CREATE LOGIN loginname;
```

C.6 CREATE PROCEDURE

CREATE PROCEDURE用于创建存储过程。详细信息请参阅第23章。

输入

```
CREATE PROCEDURE procedurename
[parameters]
AS
BEGIN
...
END;
```

C.7 CREATE TABLE

CREATE TABLE用于创建新数据库表。为更新已经存在的表的结构，使用ALTER TABLE。详细信息请参阅第20章。

输入

```
CREATE TABLE tablename
(
        column    datatype    [NULL¦NOT NULL]    [CONSTRAINTS],
        column    datatype    [NULL¦NOT NULL]    [CONSTRAINTS],
        ...
);
```

305

C.8　CREATE VIEW

CREATE VIEW用来创建一个或多个表上的新视图。详细信息请参阅第21章。

输入

```
CREATE VIEW viewname
AS
SELECT ...;
```

C.9　DELETE

DELETE从表中删除一行或多行。详细信息请参阅第19章。

输入

```
DELETE FROM tablename
[WHERE ...];
```

C.10　DROP

DROP永久地删除数据库对象（表、视图、索引等）。详细信息请参阅第20、21、23章和第24章。

输入

```
DROP DATABASE¦INDEX¦LOGIN¦PROCEDURE¦TABLE¦TRIGGER¦VIEW
    itemname;
```

C.11　INSERT

INSERT给表增加一行。详细信息请参阅第18章。

306

输入

```
INSERT INTO tablename [(columns, ...)]
VALUES(values, ...);
```

C.12　INSERT SELECT

INSERT SELECT插入SELECT的结果到一个表。详细信息请参阅第18章。

输入

```
INSERT INTO tablename [(columns, ...)]
SELECT columns, ... FROM tablename, ...
[WHERE ...];
```

C.13　ROLLBACK TRANSACTION

ROLLBACK TRANSACTION用于撤销一个事务处理块。详细信息请参阅第26章。

输入

```
ROLLBACK [savepointname];
```

C.14　SAVE TRANSACTION

SAVEPOINT TRANSACTION为使用ROLLBACK语句设立保留点。详细信息请参阅第26章。

307

输入

```
SAVE TRANSACTION sp1;
```

C.15　SELECT

SELECT用于从一个或多个表（视图）中检索数据。更多的基本信息，请参阅第4、5和6章（第4～17章都与SELECT有关）。

输入

```
SELECT columnname, ...
FROM tablename, ...
[WHERE ...]
[UNION ...]
[GROUP BY ...]
[HAVING ...]
[ORDER BY ...];
```

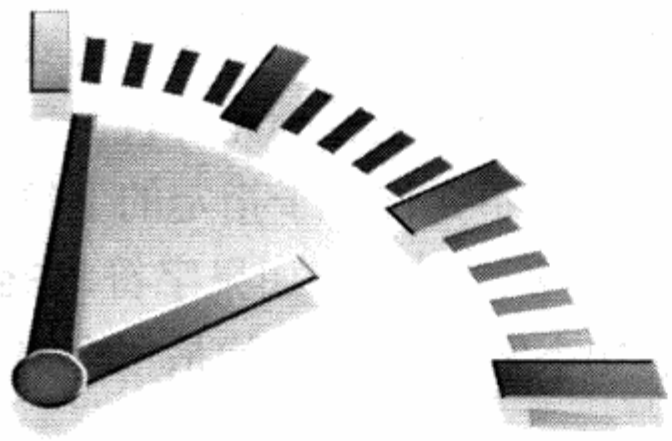
C.16　UPDATE

UPDATE更新表中一行或多行。详细信息请参阅第19章。

输入

```
UPDATE tablename
SET columnname = value, ...
[WHERE ...];
```

308



附录D

T-SQL数据类型

正如第1章所述，数据类型是定义列中可以存储什么数据以及该数据实际怎样存储的基本规则。

数据类型用于以下目的。

- 数据类型允许限制可存储在列中的数据的类型。例如，数值数据类型列只能接受数值。
- 数据类型允许在内部更有效地存储数据。可以用一种比文本字符串更简洁的格式存储数值和日期时间值。
- 数据类型允许变换排序顺序。如果所有数据都作为字符串处理，则1位于10之前，而10又位于2之前（字符串以字典顺序排序，从左边开始比较，一次一个字符）。作为数值数据类型，数值才能正确排序。

在设计表时，应该特别重视所用的数据类型。使用错误的数据类型可能会严重地影响应用程序的功能和性能。更改已填充数据列的数据类型不是一件小事（而且这样做可能会导致数据丢失）。

本附录虽然不是关于数据类型及其如何使用的完整教程，但解释了主要的T-SQL数据类型及其用途。

D.1 字符串数据类型

最常用的数据类型是字符串数据类型，它们存储字符串，如名字、地址、电话号码、邮政编码等。有两种基本的字符串类型，分别为定长字符串和变长字符串（参见表D-1）。

定长字符串是接受固定字符长度的数据类型，其长度是在创建表时指定的。例如，名字列可允许30个字符，而社会安全号列允许11个字符（允许的字符数目中包括两个破折号）。定长列不允许多于指定的字符数目。它们分配的存储空间与指定的字符一样多。因此，如果字符串Ben存储到30个字符的名字字段，则存储的是30个字节。CHAR就是一个定长字符串类型。

变长字符串存储可变长度的文本。TEXT就是一个变长字符串类型。

既然变长数据类型这样灵活，为什么还要使用定长数据类型？是因为性能。SQL Server排序和操作定长列远比排序和操作变长列快得多。此外，SQL Server不允许对变长列（或一个列的可变部分）进行索引。这也会极大地影响性能。

表D-1　字符串数据类型

数据类型	说　明
CHAR	1～8 000个字符的定长字符串，其长度必须在创建时指定，否则SQL Server认为CHAR(1)
NCHAR	1～4 000个字符的定长Unicode字符串，其长度必须在创建时指定，否则SQL Server认为NCHAR(1)
NTEXT	最大1 073 741 823个字符的变长Unicode文本
NVARCHAR	最大4 000个字符的变长Unicode文本
TEXT	最大2 147 483 647个字符的变长文本
VARCHAR	最大8 000个字符的变长文本

310

使用引号　不管使用何种形式的字符串数据类型，字符串值都必须括在引号内（通常单引号更好）。

当数值不是数值时　你可能会认为电话号码和邮政编码应该存储在数值字段中（毕竟，数值字段只存储数值数据），但是，这样做却是不可取的。如果在数值字段中存储邮政编码01234，则保存的将是数值1234，实际上丢失了一位数字。

需要遵守的基本规则是：如果数值是计算（求和、平均等）

中使用的数值，则应该存储在数值数据类型列中。如果作为字面量字符串（可能只包含数字）使用，则应该保存在字符串数据类型列中。

D.2　数值数据类型

数值数据类型存储数值。SQL Server支持多种数值数据类型，每种存储的数值具有不同的取值范围。显然，支持的取值范围越大，所需存储空间越多。此外，有的数值数据类型支持使用十进制小数点（和小数），而有的则只支持整数。表D-2列出了常用的SQL Server数值数据类型。 311

表D-2　数值数据类型

数据类型	说　明
BIT	单个二进制位字段，或者为0或者为1
BIGINT	整数值，支持从–9 223 372 036 854 775 808～9 223 372 036 854 775 807的数
DECIMAL（DEC或NUMERIC）	精度可变的浮点值
FLOAT	变长字节浮点值
INT（或INTEGER）	整数类型，支持从–2 147 483 648～2 147 483 647的数
MONEY	精确到小数点后4位的货币值，支持从–922 337 203 685 477.5808～922 337 203 685 477.5807的数
REAL	4字节浮点值
SMALLINT	整数类型，支持从–32 768～32 767的数
SMALLMONEY	精确到小数点后4位的货币值，支持从–214 748.3648～214 748.3647的数
TINYINT	整数值，支持从0～255的数

不使用引号　与字符串不一样，数值从不括在引号内。

D.3　日期和时间数据类型

312

SQL Server使用专用数据类型存储日期和时间值（见表D-3）。

表D-3　日期和时间数据类型

数据类型	说　明
DATETIME	存储从1753年1月1日到9999年12月31日的日期
SMALLDATETIME	存储从1900年1月1日到2079年6月6日的日期

D.4　二进制数据类型

二进制数据类型可用于存储任何数据（甚至二进制信息），如图像、多媒体、字处理文档等（参见表D-4）。

表D-4　二进制数据类型

数据类型	说　明
BINARY	定长二进制数据，最多8 000个字符
VARBINARY	变长二进制数据，最多8 000个字符
VARBINARY(max)	变长二进制数据，超过8 000个字符

D.5 其他数据类型

除了之前列出的数据类型，SQL Server还支持几种数值专用的数据类型（参见表D-5）。

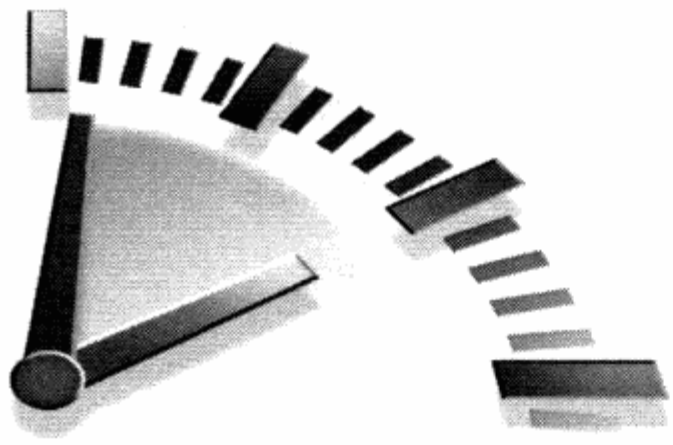
表D-5　其他数据类型

数据类型	说　明
CURSOR	包含到游标的引用
TABLE	临时表
UNIQUEIDENTIFIER	16字节GUID格式的唯一标识符
XML	合式的XML数据

313

使用中的数据类型　如果想看使用不同数据类型的例子，请参看附录B中样例表的表创建脚本。

314



附录E

T-SQL保留字

SQL的T-SQL实现由关键字组成，关键字是一些用于执行SQL操作的特殊词。在命名数据库、表、列和其他数据库对象时，一定不要使用这些关键字。因此，这些关键字是一定要保留的。本附录列出所有的T-SQL保留字（自SQL Server 2005以后的版本），包括ODBC保留字以及微软公司为将来使用而保留的词。

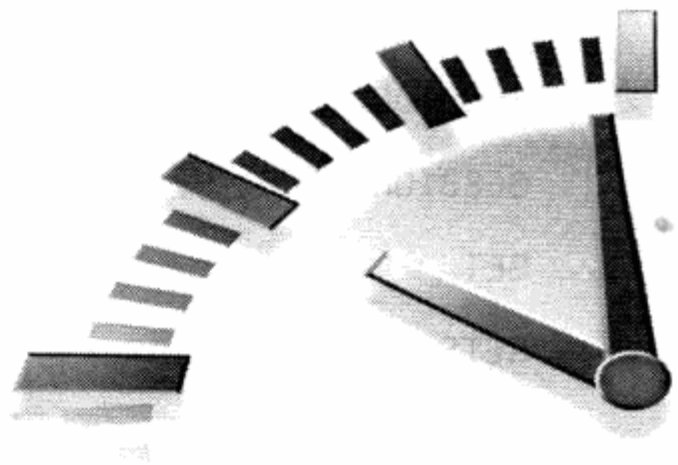
ABSOLUTE	ASSERTION	BULK
ACTION	AT	BY
ADA	AUTHORIZATION	CALL
ADD	AVG	CASCADE
ADMIN	BACKUP	CASCADED
AFTER	BEFORE	CASE
AGGREGATE	BEGIN	CAST
ALIAS	BETWEEN	CATALOG
ALL	BINARY	CHAR
ALLOCATE	BIT	CHAR_LENGTH
ALTER	BIT_LENGTH	CHARACTER
AND	BLOB	CHARACTER_LENGTH
ANY	BOOLEAN	CHECK
ARE	BOTH	CHECKPOINT
ARRAY	BREADTH	CLASS
AS	BREAK	CLOB
ASC	BROWSE	CLOSE

CLUSTERED
COALESCE
COLLATE
COLLATION
COLUMN
COMMIT
COMPLETION
COMPUTE
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS
CONSTRUCTOR
CONTAINS
CONTAINSTABLE
CONTINUE
CONVERT
CORRESPONDING
COUNT
CREATE
CROSS
CUBE
CURRENT
CURRENT_DATE
CURRENT_PATH
CURRENT_ROLE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
CYCLE
DATA
DATABASE
DATE
DAY
DBCC
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DENY
DEPTH
DEREF
DESC
DESCRIBE
DESCRIPTOR
DESTROY
DESTRUCTOR
DETERMINISTIC
DIAGNOSTICS
DICTIONARY
DISCONNECT
DISK
DISTINCT
DISTRIBUTED
DOMAIN
DOUBLE
DROP
DUMMY
DUMP
DYNAMIC
EACH
ELSE
END
END-EXEC
EQUALS
ERRLVL
ESCAPE
EVERY
EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXIT
EXTERNAL
EXTRACT
FALSE
FETCH
FILE
FILLFACTOR
FIRST
FLOAT

FOR	IN	LEVEL
FOREIGN	INCLUDE	LIKE
FORTRAN	INDEX	LIMIT
FOUND	INDICATOR	LINENO
FREE	INITIALIZE	LOAD
FREETEXT	INITIALLY	LOCAL
FREETEXTTABLE	INNER	LOCALTIME
FROM	INOUT	LOCALTIMESTAMP
FULL	INPUT	LOCATOR
FULLTEXTTABLE	INSENSITIVE	LOWER
FUNCTION	INSERT	MAP
GENERAL	INT	MATCH
GET	INTEGER	MAX
GLOBAL	INTERSECT	MIN
GO	INTERVAL	MINUTE
GOTO	INTO	MODIFIES
GRANT	IS	MODIFY
GROUP	ISOLATION	MODULE
GROUPING	ITERATE	MONTH
HAVING	JOIN	NAMES
HOLDLOCK	KEY	NATIONAL
HOST	KILL	NATURAL
HOUR	LANGUAGE	NCHAR
IDENTITY	LARGE	NCLOB
IDENTITY_INSERT	LAST	NEW
IDENTITYCOL	LATERAL	NEXT
IF	LEADING	NO
IGNORE	LEFT	NOCHECK
IMMEDIATE	LESS	NONCLUSTERED

NONE
NOT
NULL
NULLIF
NUMERIC
OBJECT
OCTET_LENGTH
OF
OFF
OFFSETS
OLD
ON
ONLY
OPEN
OPENDATASOURCE
OPENQUERY
OPENROWSET
OPENXML
OPERATION
OPTION
OR
ORDER
ORDINALITY
OUT
OUTER
OUTPUT
OVER
OVERLAPS
PAD
PARAMETER
PARAMETERS
PARTIAL
PASCAL
PATH
PERCENT
PLAN
POSITION
POSTFIX
PRECISION
PREFIX
PREORDER
PREPARE
PRESERVE
PRIMARY
PRINT
PRIOR
PRIVILEGES
PROC
PROCEDURE
PUBLIC
RAISERROR
READ
READS
READTEXT
REAL
RECONFIGURE
RECURSIVE
REF
REFERENCES
REFERENCING
RELATIVE
REPLICATION
RESTORE
RESTRICT
RESULT
RETURN
RETURNS
REVOKE
RIGHT
ROLE
ROLLBACK
ROLLUP
ROUTINE
ROW
ROWCOUNT
ROWGUIDCOL
ROWS
RULE
SAVE
SAVEPOINT
SCHEMA
SCOPE
SCROLL
SEARCH
SECOND
SECTION
SELECT

SEQUENCE
SESSION
SESSION_USER
SET
SETS
SETUSER
SHUTDOWN
SIZE
SMALLINT
SOME
SPACE
SPECIFIC
SPECIFICTYPE
SQL
SQLCA
SQLCODE
SQLERROR
SQLEXCEPTION
SQLSTATE
SQLWARNING
START
STATE
STATEMENT
STATIC
STATISTICS
STRUCTURE
SUBSTRING
SUM
SYSTEM_USER
TABLE
TEMPORARY
TERMINATE
TEXTSIZE
THAN
THEN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TOP
TRAILING
TRAN
TRANSACTION
TRANSLATE
TRANSLATION
TREAT
TRIGGER
TRIM
TRUE
TRUNCATE
TSEQUAL
UNDER
UNION
UNIQUE
UNKNOWN
UNNEST
UPDATE
UPDATETEXT
UPPER
USAGE
USE
USER
USING
VALUE
VALUES
VARCHAR
VARIABLE
VARYING
VIEW
WAITFOR
WHEN
WHENEVER
WHERE
WHILE
WITH
WITHOUT
WORK
WRITE
WRITETEXT
YEAR
ZONE



索　引

索引中的页码为英文原书的页码、与书中边栏的页码一致。

符号

A

B

C

D

E

F

G

H

I

J

K

L

M

P

Q

R

S

T

U

V

W

X-Z