高职高专 21 世纪规划教材

数据库技术与应用
——SQL Server 2005

张建伟　主　编
梁树军　副主编

人 民 邮 电 出 版 社
北 京

目　　录

第 1 章

数据库基础知识

数据库技术已成为计算机科学的一个重要分支，是数据管理的最新技术，是计算机技术中发展最快的领域之一。许多信息系统都是以数据库为基础建立的。数据库已经成为人们存储数据、管理信息、共享资源的最先进、最常用的技术。

本章介绍数据库系统的基本概念，包括数据管理技术的发展过程、数据库系统的基本概念、数据模型及数据库系统的体系结构等。读者从中可以学习到使用数据库的原因及其重要性。本章是学习后面各章节的预备和基础。

1.1 数据库概述

数据库技术是计算机技术中发展最为迅速的领域之一，已经在科学、技术、经济、文化和军事等领域发挥着重要作用。

1.1.1 数据库、数据库管理系统与数据库系统

1. *数据库*

数据库（Database，DB），顾名思义，是存放数据的仓库。只不过这个仓库是在计算机的存储设备上，而且数据是按照一定的数据模型组织并存放在外存上的一组相关数据的集合。通常这些数据是面向一个组织、企业或部门的。例如学生成绩管理系统中，学生的基本信息、课程信息、成绩信息等都是来自学生成绩管理数据库的。

除了用户可以直接使用的数据，还有另外一种数据。它们是有关数据库的定义信息的，如数据库的名称，表的定义，数据库用户名及密码、权限等。这些数据用户不会经常使用，但是对数据库非常重要。这些数据通常存放在“数据字典（Data Dictionary）”中。数据字典是数据库管理系统中非常重要的组成部分，它是由数据库管理系统自动生成并维护的一组表和视图。数据字典是数据库管理系统工作的依据。数据库管理系统借助数据字典来理解数据库中数据的组织，并完成对数据库中数据的管理与维护。数据库用户可通过数据字典获取有用的信息，如用户创建了哪些数据库对象，这些对象是如何定义的，这些对象允许哪些用户使用等。但是，数据库用户是不能随便改动数据字典中的内容的。

在收集并抽取出一个应用所需要的大量数据之后，应将其保存起来供进一步查询和加工处理，以获得更多有用的信息。过去人们把数据存放在文件柜里，数据越来越多，从大量的

文件中查找数据就会十分困难。现在人们借助数据库，科学地保存和管理大量复杂的数据，从而能方便而又充分地利用这些宝贵的信息资源。

严格地讲，数据库是长期存储在计算机内，有组织的、大量的、可共享的数据集合。数据库中的数据按一定的数据模型组织、描述和存储，具有较小的冗余度、较高的数据独立性和易扩展性，并可为用户共享。

简而言之，数据库中的数据具有永久存储、有组织和可共享 3 个基本特点。

2. 数据库管理系统

在建立了数据库之后，下一个问题就是如何科学地组织和存储数据，如何高效地获取和维护数据。完成这个任务的是一个系统软件——数据库管理系统（Database Management System，DBMS）。

DBMS 是指数据库系统中对数据进行管理的软件系统，它是数据库系统的核心组成部分，数据库系统的一切操作，包括查询、更新及各种控制，都是通过 DBMS 进行的。DBMS 是基于数据模型的，因此可以把它看成是某种数据模型在计算机系统上的具体实现。根据所采用数据模型的不同，DBMS 可以分成网状型、层次型、关系型、面向对象型等。但在不同的计算机系统中，由于缺乏统一的标准，即使是同种数据模型的 DBMS，它们在用户接口、系统功能等方面也常常是不同的。

DBMS 把用户对数据库的操作从应用程序带到外部级、概念级，再导向内部级，进而操纵存储器中的数据。一个 DBMS 的主要目标是使数据成为一种可管理的资源。DBMS 应使数据易于为各种用户所共享，应该增进数据的安全性、完整性及可用性，并提供高度的数据独立性。

3. 数据库系统

数据库系统（Database System，DBS）是指在计算机系统中引入数据库后的系统，一般由数据库、数据库管理系统（及其开发工具）、应用系统和数据库管理员构成。应当指出的是，数据库的建立、使用和维护等工作只靠一个 DBMS 是远远不够的，还要有专门的人员来完成，这些人被称为数据库管理员（Database Administrator，DBA）。

在不引起混淆的情况下，人们常常把数据库系统简称为数据库。数据库系统组成如图 1.1 所示。数据库系统在计算机系统中的位置如图 1.2 所示。

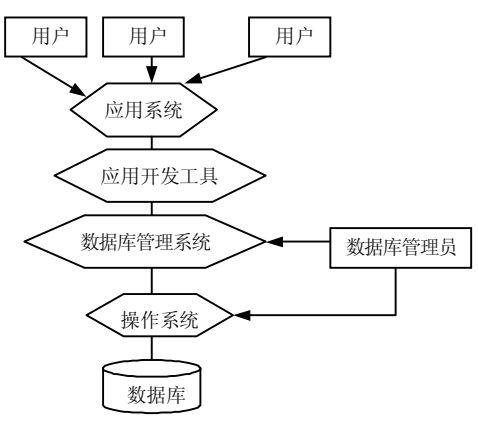


图 1.1　数据库系统

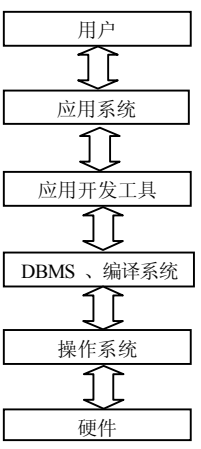


图 1.2　数据库系统在计算机系统中的地位

1.1.2　数据库技术的产生与发展

在使用计算机之后，数据处理的速度及规模都是过去人工或机械方式无法比拟的。随着数据处理量的不断增加，数据管理技术应运而生，其演变过程随着计算机硬件和软件的发展不断变化，以一个学校的教务处对学生、课程和成绩的管理为例，在没有使用计算机的时候，教务处的工作人员将学生的信息抄写在一张张的卡片上。为了方便查找，将同一个系、同一个年级、同一个班级的学生卡片存放在相邻的地方，并对不同的系、年级和班级做上标签。每门课程的信息也抄写在卡片上，将同一个专业的卡片放在一起并做上标签。每个学期的期末将同一个班的各门课的成绩单收集起来存放在档案中。

当查找一个学生的信息时，如果知道他所在的系和班级，按照标签可以很快找到该学生的卡片。如果只知道他的姓名，那么只有在所有的学生卡片中一个一个查找，需要花费很多时间。

当计算一个学生某个学期的平均成绩时，首先在档案中找到该学生所在班级这个学期的所有成绩单，从中找出该学生各门课程的成绩，再算平均成绩。统计某一门课的成绩分布时也要进行类似的处理。

总的来说，数据库技术的发展经历了以下几个阶段。

1. 人工管理阶段

计算机没有应用到数据管理领域之前，数据管理的工作是由人工完成的。这种处理方式经历了很长时间。

20 世纪 50 年代中期以前，计算机主要用于科学计算。当时的硬件状况是，外存只有纸带、卡片、磁带，没有磁盘等直接存取的存储设备；软件状况是，没有操作系统，没有管理数据的软件，因此称这一阶段的数据管理方式为人工管理数据。人工管理数据具有如下特点。

（1）数据不保存。由于当时计算机主要用于科学计算，一般不需要将数据长期保存，只是在计算某一课题时将数据输入，用完不保存。

如果要用计算机统计分析全校每一门课的成绩，就要编写统计分析程序，在运行该程序时读入相应的学生选修课程成绩单等数据，计算完成后数据和程序都不在计算机中保存。

（2）应用程序管理数据。数据需要由应用程序自行管理，没有相应的软件系统负责数据的管理工作。应用程序中不仅要规定数据的逻辑结构，而且要设计物理结构，包括存储结构、存取方法、输入方式等，因此程序员负担很重。

（3）数据不共享。数据是面向应用的，一组数据只能对应一个程序。当多个应用程序涉及某些相同的数据时，由于必须各自定义，无法互相利用、互相参照，因此程序与程序之间有大量的冗余数据。例如教务处既要统计某一门课的成绩又要分析某一个学生的成绩时，就要编写两个程序，尽管都要使用学生选修课成绩单，但是每个程序要分别定义两个成绩单数据，分别输入，分别使用，如图 1.3 所示。

（4）数据不具有独立性。数据的逻辑结构或物理结构改变后，必须对应用程序做相应的修改，这就进一步加重了程序员的负担。例如学生成绩由 5 级记分制改为百分制时，上面两个统计分析程序都要修改。

在人工管理阶段，程序与数据之间的对应关系如图 1.4 所示。

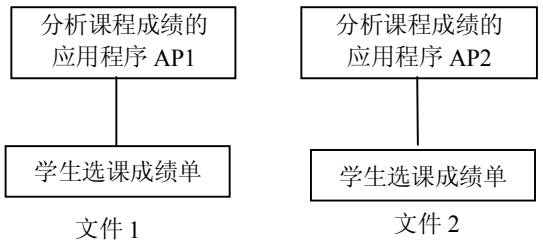


图 1.3　两个应用程序使用同一数据

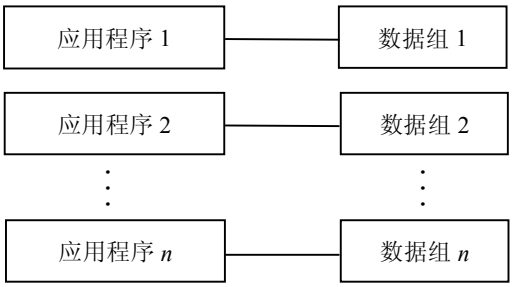


图 1.4　人工管理阶段应用程序与数据之间的对应关系

2. 文件系统阶段

20 世纪 50 年代后期到 60 年代中期，硬件方面已有了磁盘、磁鼓等直接存储设备；软件方面，操作系统中已经有了专门的数据管理软件——文件系统。可以把相关的数据组织成一个文件存放在计算机中，需要时只要提供文件名，计算机就能从文件系统中找出所要的文件，把文件中存储的数据提供给用户进行处理。

例如在学校教务处对学生学籍的管理中，为了改变查找、计算工作量大及花费时间长的被动局面，将学生卡片、课程卡片和学生学习成绩单中的内容存放到文件“Student”、“Course”、“Study”中，并对每个文件编写一组程序用于数据维护，包括增加、删除、修改和查询一条记录。在此基础上根据需要编写一些查询和报表打印程序，如根据学生的姓名、学生编号查找学生的信息，统计某学期某个学生的平均成绩，统计某门课的平均成绩等。

管理系统投入应用后，教务处的工作效率大大提高。例如，每学期末将各门课的考试成绩输入计算机以后，可以很快计算出学生的平均成绩，并打印出需要补考的学生的名单。

但是，由于数据的组织仍然是面向程序，所以仍然存在大量的数据冗余，经过一段时间的使用后，教务人员发现有时必须修改程序和文件结构才能适应工作的需要。例如，学校领导要求统计不同生源地的学生成绩，因为原来的学籍管理软件中没有实现这个功能，必须编写一段程序来实现。又如，当需要在“Student”文件中增加“个人网址”属性时，这涉及改变文件的结构，需要若干步骤才能完成。

第一步，建立一个新文件“Student-new”，其结构是在“Student”的结构中加入“个人网址”这一项。

第二步，编写一个程序将文件“Student”中的数据转存到“Student-new”中。

第三步，删除文件“Student”。

第四步，将文件“Student-new”重命名为“Student”。

这项工作到此并没有结束，因为文件中保存的是数据，不保存数据的结构，数据结构是在程序中定义的，旧“Student”文件的结构写到了所有使用它的程序中，必须一一修改这些程序以适应新的文件结构，否则程序运行就会出错。

从这个例子可以看到用文件系统管理数据的优点和不足。一般地讲，用文件系统管理数据具有如下特点。

（1）数据可以长期保存。数据可以组织成文件长期保存在计算机中反复使用。

（2）由文件系统管理数据。文件系统把数据组织成内部有结构的记录，实现“按文件名

访问，按记录进行存取”的管理技术。

文件系统使应用程序与数据之间有了初步的独立性，程序员不必过多的考虑数据存储的物理细节。例如，文件系统中可以有顺序结构文件、索引结构文件、Hash 等，数据在存储上的不同不会影响程序的处理逻辑。如果数据的存储结构发生改变，应用程序的改变会很小，节省了程序的维护工作量。但是，文件系统仍存在以下缺点。

（1）数据共享性差，冗余度大。在文件系统中，一个（或一组）文件基本上对应于一个应用（程序），即文件是面向应用的。当不同的应用（程序）使用相同的数据时，也必须建立各自的文件，而不能共享相同的数据。因此数据的冗余度大，浪费存储空间。同时，由于相同数据的重复存储、各自管理，容易造成数据的不一致性，给数据的修改和维护带来了困难。

（2）数据独立性差。文件系统中的文件是为某一特定应用服务的，文件的逻辑结构对该应用来说是优化的，因此要对现有的数据再增加一些新的应用会很困难，系统不容易扩充。一旦数据的逻辑结构发生改变，就必须修改应用程序，修改文件结构的定义。因此数据与程序之间仍缺乏独立性。文件系统阶段程序与数据之间的关系如图 1.5 所示。

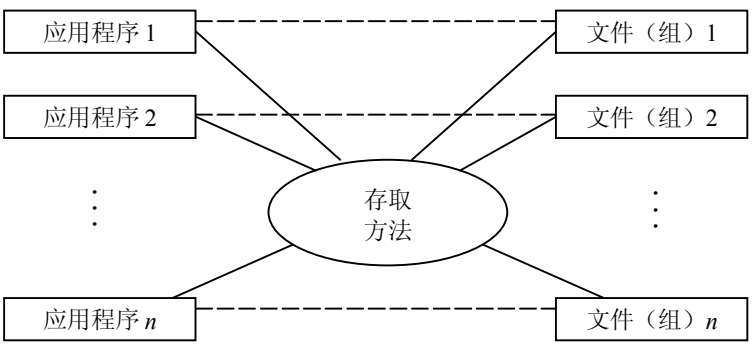


图 1.5　文件系统阶段应用程序与数据之间的对应关系

3. 数据库系统阶段

20 世纪 60 年代后期，计算机用于管理的规模越来越大，应用越来越广泛，数据量急剧增长，同时对多种应用、多种语言互相覆盖的共享数据集合的需求也越来越强烈。

这时已有大容量磁盘，硬件价格下降；软件价格则上升，为编制和维护系统软件及应用程序所需的成本相对增加。在这种背景下，以文件系统作为数据管理手段已经不能满足应用的需求。于是，为解决多用户、多应用共享数据的需求，使数据为尽可能多的应用服务，数据库技术便应运而生，出现了统一管理数据的专用软件系统——数据库管理系统。

用数据库系统来管理数据和使用文件系统相比具有明显的优点，从文件系统到数据库系统，标志着数据管理技术的飞跃。

例如，对教务管理系统，学校决定采用数据库技术，购买了一个关系数据库管理系统（RDBMS），在这个 RDBMS 之上建立一个应用系统，将教务处和学生工作处保存的学生数据进行综合设计，供全校各院系的教师和教务人员共享访问和使用。

在系统中要建立 3 个关系：学生基本表“Student”、课程基本表“Course”和学习基本表“SC”。在数据库系统中只要用 DDL 语言向 RDBMS 提交 CREATE TABLE 语句就可以了，例

如建立关系“Student”：

```
CREATE TABLE Student
                (Sno        CHAR（10） NOT NULL UNIQUE，
                Sname       CHAR（10），
                Sex         CHAR（2），
                Age         INT，
                Classno     CHAR（6），
                Dept        CHAR（12） )；
```

这条语句在数据库中建立了一个关系“Student”，用来保存学生的信息。更重要的是，RDBMS 将“Student”的结构也保存到数据库的数据字典中。

向关系中增加、删除、修改一个元组（记录）用 RDBMS 提供的语句 INSERT、DELETE、UPDATE 来完成。这些语句如何操作磁盘上的数据是由 RDBMS 来完成的，程序员不用编写专门的程序，从而节省了程序员大量的时间和精力。

因为关系“Student”的结构和数据都由 RDBMS 管理，在教务管理系统中，向关系“Student”添加“E-mail”属性时，既不需要编写转储数据的程序，也不用修改那些使用了关系“Student”的程序。

从这个例子可以看出，数据库系统阶段应用程序与数据之间的对应关系如图 1.6 所示。

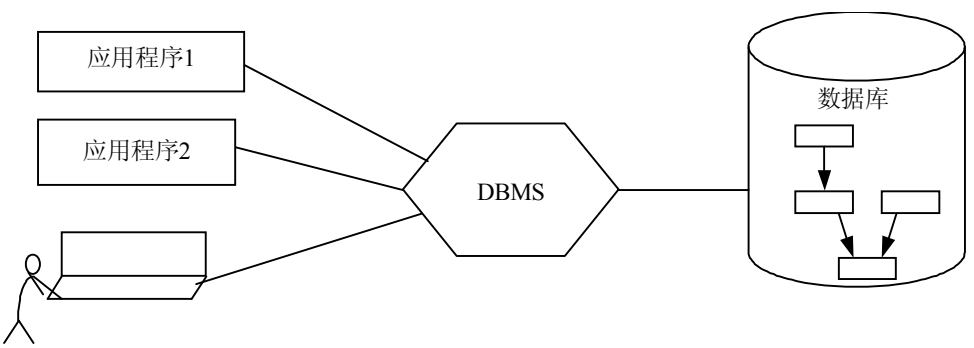


图 1.6　数据库系统阶段应用程序与数据之间的对应关系

由于数据库是以数据为中心组织数据，减少数据的冗余，提供更高的数据共享能力，同时要求程序和数据具有较高的独立性，因此当数据的逻辑结构改变时，不涉及数据的物理结构，也不影响应用程序，这样就降低了应用程序研制与维护的费用。

4. 高级数据库阶段

这一阶段的主要标志是 20 世纪 80 年代的分布式数据库系统、90 年代的对象数据库系统和 21 世纪初的网络数据库系统的出现。

（1）分布式数据库系统。在这一阶段以前的数据库系统是集中式的。在文件系统阶段，数据分散在各个文件中，文件之间缺乏联系。集中式数据库把数据库集中在一个数据库中进行管理，减少了数据冗余和不一致性，而且数据联系比文件系统更强。但集中式系统也有弱点：一是随着数据量增加，系统会变得相当庞大，操作复杂，开销大；二是数据集中存储，大量的通信都要通过主机，造成拥挤现象。随着小型计算机和微型计算机的普及，计算机网

络软件和远程通信的发展，分布式数据库系统崛起了。

分布式数据库系统主要有以下 3 个特点。

① 数据库的数据物理上分布在不同地方，但逻辑上是一个整体。

② 各个分散的数据库既可以执行局部应用（访问本地数据库），又可以执行全局应用（访问异地数据库）。

③ 各分散的计算机由数据通信网络相连。本地计算机不能单独胜任的处理任务，可以通过通信网络取得其他数据库和计算机的支持。

分布式数据库系统兼顾了集中管理和分布处理两个方面，因而有良好的性能。

（2）对象数据库系统。在数据处理领域，关系数据库的使用已相当普遍、相当出色。但是现实世界存在着许多具有更复杂数据结构的实际应用领域，已有的层次、网状、关系三种数据模型对这些应用领域都显得力不从心。如多媒体数据、多维表格数据、CAD 数据等应用问题，需要更高级的数据库技术来支持，以便管理、构造与维护大容量的持久数据，并使它们能与大型复杂程序紧密结合。对象数据库正是由于适应这种形势而发展起来的，它是面向对象的程序设计技术与数据库技术结合的产物。

对象数据库系统主要有以下两个特点。

① 对象数据库模型能完整地描述现实世界的数据结构，能表达数据间嵌套、递归等关系。

② 具有面向对象技术的封装性（把数据与操作定义在一起）和继承性（继承数据结构和操作）的特点，提高了软件的可重用性。

（3）网络数据库系统。C/S（客户机/服务器）结构的出现，使得人们可以更有效地使用计算机资源。但在网络环境中，如何隐藏各种复杂性，这就要使用中间件。中间件是网络环境中保证不同的操作系统、通信协议和 DBMS 之间进行对话、互操作的软件系统。其中涉及到数据访问的中间件，就是 20 世纪 90 年代提出的 ODBC 和 JDBC 技术。

现在，计算机网络已成为信息化社会中十分重要的一类基础设施。随着广域网（WAN）的发展，信息高速公路已发展成为采用通信手段将地理位置分散，具备自主功能的若干台计算机和数据库系统有机地连接起来组成的因特网（Internet），用于实现通信交往、资源共享或协调工作等目标。这个目标在 20 世纪末已经实现，正在对社会的发展起着巨大的推进作用。

1.1.3　数据库系统的组成与结构

在前面已经介绍了数据库系统一般由数据库、数据库管理系统（及其开发工具）、应用系统和数据库管理员构成。下面分别介绍这几部分的内容，并从不同的角度描述数据库系统的结构。

1. 数据库系统的组成

（1）硬件平台及数据库。硬件系统主要指计算机各个组成部分。鉴于数据库应用系统的需求，要求数据库主机或数据库服务器外存足够大，I/O 存取效率高，主机的吞吐量大，作业处理能力强。对于分布式数据库而言，计算机网络也是基础环境。因此，具体的硬件要求包括：

① 要有足够大的内存，存放操作系统和 DBMS 的核心模块、数据库缓冲区和应用程序；

② 有足够大的磁盘等直接存取设备存放数据库数据，有足够的光盘、磁盘、磁带等作为

数据备份介质；

③ 要求连接系统的网络有较高的数据传送率；

④ 有较强处理能力的中央处理器（CPU）来保证数据处理的速度。

（2）软件。数据库系统的软件主要包括：

① DBMS，为数据库的建立、使用和维护配置的软件；

② 支持 DBMS 运行的操作系统；

③ 与数据库通信的高级程序语言及编译系统；

④ 为特定应用环境开发的数据库应用系统。

（3）数据库管理员及其他相关人员。数据库相关人员包括数据库管理员（DBA）、系统分析员、应用程序员和普通用户，各自职责如下所述。

① 数据库管理员（Database Administrator，DBA）。数据库管理员负责管理和监控数据库系统，负责为用户解决应用中出现的系统问题。为了保证数据库能够高效正常地运行，大型数据库系统都设有专人负责数据库系统的管理和维护。其主要职责如下。

- 决定数据库中的信息内容和结构。数据库中要存放哪些信息，DBA 要参与决策。因此 DBA 必须参加数据库设计的全过程，并与用户、应用程序员、系统分析员密切合作，共同协商，做好数据库设计工作。
- 决定数据库的存储结构和存取策略。
- 监控数据库的运行（系统运行是否正常，系统效率如何），及时处理数据库系统运行过程中出现的问题。比如系统发生故障时，数据库会因此遭到破坏，DBA 必须在最短的时间内把数据库恢复到正确状态。
- 安全性管理。通过对系统的权限设置、完整性控制设置来保证系统的安全性。DBA 要负责确定各个用户对数据库的存取权限、数据的保密级别和完整性约束条件。
- 日常维护，如定期对数据库中的数据进行备份，维护日志文件等。
- 对数据库有关文档进行管理。

数据库管理员在数据库管理系统的正常运行中起着非常重要的作用。

② 系统分析员和数据库设计人员。系统分析员负责应用系统的需求分析和规范说明，和用户及 DBA 配合，确定系统的硬件、软件配置，并参与数据库系统概要设计。

③ 应用程序员。应用程序员是负责设计、开发应用系统功能模块的软件编程人员，根据数据库结构编写特定的应用程序，并进行调试和安装。

④ 用户。这里的用户是指最终用户。最终用户通过应用程序的用户接口使用数据库。常用的接口方式有浏览器、菜单驱动、表格操作、图形显示、报表等。

2. 数据库系统的结构

数据库系统的结构可以从不同的层次或角度来考察。

从数据库管理系统角度看，数据库系统通常分为三级模式，这是数据库管理系统内部的体系结构。

从数据库最终用户角度看，数据库系统的结构分为单用户结构、主从式结构、分布式结构、客户/服务器结构等。这是数据库系统外部的体系结构。

数据库系统结构是数据库的总的框架。尽管实际的数据库系统软件产品多种多样，支持不同的数据模型，使用不同的数据库语言，建立在不同的操作系统之上，但绝大多数数据库

系统在总的体系结构上都具有三级模式的结构特征。学习数据库的三级模式将有助于理解数据库设计及应用中的一些基本概念。

数据库的三级模式为外模式、概念模式和内模式，如图 1.7 所示。

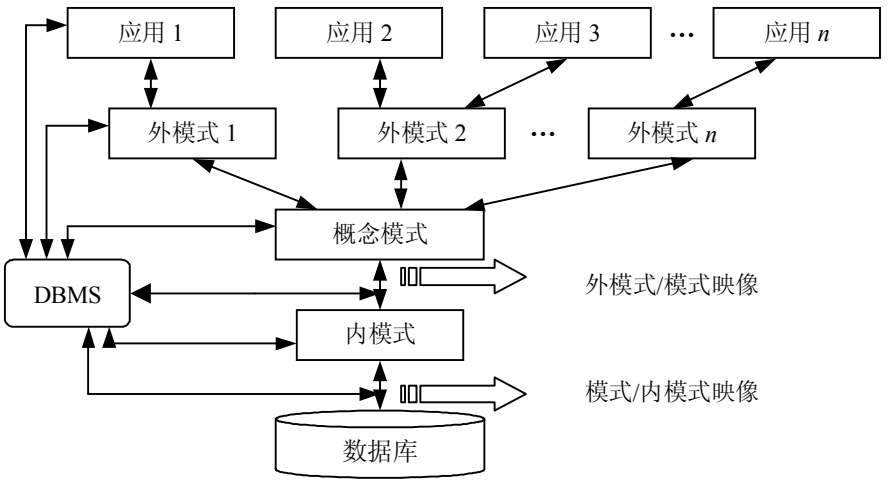


图 1.7　数据库系统结构——三级模式

（1）数据库的三级模式

① 概念模式。概念模式也称模式，是对数据库中全局数据逻辑结构的描述，是全体用户公共的数据视图。这种描述是一种抽象描述，不涉及具体硬件环境与平台，也与具体软件环境无关。

概念模式主要描述数据的概念记录类型及其关系，还包括数据间的一些语义约束。对它的描述可用 DBMS 中的 DDL 定义。

② 外模式（External Schema）。外模式也称子模式（Subschema）或用户模式，它是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用相关的数据的逻辑表示。

外模式通常是模式的子集。一个模式可以有多个外模式。由于它是各个用户的数据视图，因此如果不同的用户在应用需求、看待数据的方式、对数据保密的要求等方面存在差异，则其外模式描述就可能不同。即使对模式中的同一数据，在外模式中的结构、类型、长度、保密级别等都可以不同。另外，同一外模式也可以为某一用户的多个应用系统使用，但一个应用程序只能使用一个外模式。

外模式是保证数据库安全性的一个有力措施。每个用户只能看到和访问所对应的外模式中的数据，数据库中的其他数据是看不到的。

DBMS 提供子模式描述语言（子模式 DDL）来严格地定义子模式。

③ 内模式（Internal Schema）。内模式也称存储模式（Storage Schema），一个模式只有一个内模式。它是数据物理结构和存储方式的描述，它定义所有的内部记录类型、索引和文件的组织形式，以及数据控制方面的细节。

内部记录并不涉及到物理记录，也不涉及到设备的约束。比内模式更接近于物理存储和访问的那些软件机制是操作系统的一部分（即文件系统），如从磁盘读数据或写数据到磁盘上

的操作等。

DBMS 提供内模式描述语言（内模式 DDL，或者存储模式 DDL）来严格的定义内模式。

（2）数据库的二级映像。数据库系统的模式、内模式、外模式之间有很大的差别。为了实现用户和数据之间的透明化，DBMS 提供了两层映像：外模式/模式映像和模式/内模式映像。

有了这两层映像，用户就能逻辑地、抽象地处理数据，而不必关心数据在计算机中的具体表示方式与存储方式。

正是这两层映像保证了数据库系统中的数据能够具有较高的逻辑独立性和物理独立性。

映像实质上是一种对应关系，是指映像双方如何进行数据转换，并定义转换规则。这样就能使数据独立性得到保证。

① 外模式/模式映像。数据库的每一个外模式都有一个外模式/模式映像，它定义了该外模式与模式之间的对应关系，外模式/模式映像一般是在外模式中描述的。

模式描述的是数据的全局逻辑结构，外模式描述的是数据的局部逻辑结构。对应于同一个模式可以有任意多个外模式。对于每一个外模式，数据库系统都有一个外模式/模式映像，它定义了该外模式与模式之间的对应关系。这些映像通常包含在各自外模式的描述中。

如果模式需要进行修改，例如数据重新定义，增加新的关系、新的属性，改变属性的数据类型等，那么只需对各个外模式/模式的映像做相应的修改，使外模式尽量保持不变。而应用程序一般是依据外模式编写的，因此应用程序也不必修改，从而保证了数据与程序的逻辑独立性，简称数据的逻辑独立性。

② 模式/内模式映像。模式/内模式映像是唯一的，因为数据库只有一个模式和内模式。映像存在于模式和内模式之间。两级模式之间的数据结构可能不一致，甚至可能差别很大。模式/内模式映像定义了模式和内模式之间的对应关系，即数据全局逻辑结构与存储结构之间的对应关系。模式/内模式映像一般是在模式中描述的。当数据库的存储结构改变时（如采用了另外一种存储结构），由数据库管理员对模式/内模式映像做相应改变，可以使模式保持不变，应用程序也不必改变。这就保证了数据与程序的物理独立性，简称数据的物理独立性。

在数据库的三级模式结构中，数据库模式即全局逻辑结构是数据库的中心与关键，它独立于数据库的其他层次。因此设计数据库模式结构应首先确定数据库的逻辑模式。

数据库的内模式依赖于它的全局逻辑结构，但独立于数据库的用户视图即外模式，也独立于具体的存储设备。它是将全局逻辑结构中所定义的数据结构及其联系按照一定的物理存储策略进行组织，达到较好的时间与空间效率的目的。

数据库的外模式面向具体的应用程序，它定义在逻辑模式之上，但独立于存储模式和存储设备。当应用需求发生较大变化，相应外模式不能满足其视图要求时，该外模式就要做相应改动，所以设计外模式时应充分考虑到应用的可扩充性。

特定的应用程序是在外模式描述的数据结构上编制的，它依赖于特定的外模式，与数据库的模式和存储结构独立。不同的应用程序有时可以共用同一个外模式。数据库的二级映像保证了数据库外模式的稳定性，从底层保证了应用程序的稳定性。除非应用需求本身发生变化，否则应用程序一般不需要修改。

数据与程序之间的独立性，使得数据的定义和描述可以从应用程序中分离出去。另外，

由于数据的存取由 DBMS 管理，用户不必考虑存取路径等细节，从而简化了应用程序的编制，大大减少了应用程序的维护和修改工作。

1.1.4　数据库系统的作用与特点

1. 数据库系统的作用

数据库系统的应用，使计算机应用深入到社会的各个角落，并发挥着越来越重要的作用，具体有下列几个方面。

（1）灵活应用。数据库容易扩充以适应增加新用户的要求，同时也容易移植以适应新的硬件环境和更大的数据容量。

（2）使用简易。由于精心设计的数据库能模拟企业的运转情况，并提供该企业数据逼真的描述，使管理部门和其他使用部门能很方便地使用和理解数据库。

（3）面向用户。由于数据库反映企业的实际运转情况，因此能满足用户的基本要求，同时又为企业的信息系统奠定了基础。

（4）简便的数据控制。对数据进行集中控制，就能保证所有用户在同样的数据上操作，而且数据对所有部门具有相同的含义。数据的冗余减到最少，消除了数据的不一致性。

（5）加快应用系统开发速度。程序员和系统分析员可以集中精力于应用的逻辑方面，而不必关心数据操纵和文件设计的细节。后援和恢复问题均由系统保证。

（6）程序设计高效。数据库使系统中的程序数目减少而又不过分增加程序的复杂性。由于 DML 命令功能强，应用程序编写快，又进一步提高了程序员的生产效率。

（7）修改方便。数据独立性使得修改数据库结构时尽量不损害已有的应用程序，使程序维护的工作量大为减少。

（8）标准化。数据库方法能促进建立整个企业的数据一致性和用法的标准化。

2. 数据库系统的特点

数据库系统已经深入人类社会活动的诸多领域。社会生活中的许多工作已经越来越依赖于数据库系统了，而且使用者与日俱增，主要是因为数据库系统具有其独特的优势。

（1）面向企业或部门，以数据为中心，形成综合性的数据库为各应用共享。

（2）数据结构化，采用一定的数据模型来表示数据结构。文件系统中的文件之间不存在联系，总体上看，其数据是没有结构的；数据库中的文件是相互联系的，从全局来看，它遵循一定的结构形式（数据模型）。这是两者最大的区别。数据库正是通过文件间的联系，较好地反映了现实世界事物之间的自然联系。数据模型不仅要描述数据本身的特点，而且要描述数据之间的联系。

（3）数据冗余小、易修改、易扩充。在数据库系统中，用户不是自己建立文件，而是取数据库中的数据子集。不同的应用程序根据处理要求不同，从数据库中获取需要的数据，这样就减少了数据的重复存储，也便于增加新的数据结构，同时也利于维护数据的一致性。

（4）较高的数据独立性。数据独立性是数据库技术努力追求的目标。简单地说，就是令数据与程序无关，数据存储方式的改变不会影响应用程序。

数据库的结构分为三级：用户的逻辑结构、整体逻辑结构和物理结构。数据独立性分两级：物理数据独立性和逻辑数据独立性，其结构如图 1.8 所示。

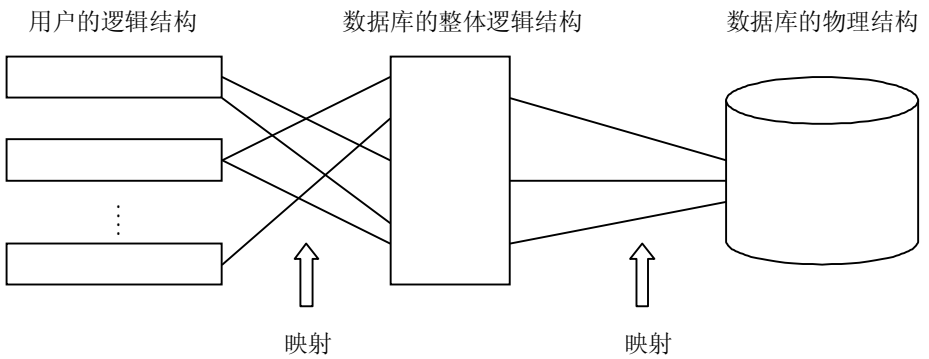


图 1.8　数据库的三级结构

当数据库物理结构（包括数据的组织和存储、存取方式、外部存储设备等）发生改变时，通过修改映射，使数据库整体逻辑结构不受影响，进而使用户的逻辑结构以及应用程序不需要改变，就称数据库达到了物理数据独立性。同样，当数据库的整体逻辑结构发生改变时，用户的逻辑结构以及应用程序不需要改变，就称数据库达到了逻辑数据独立性。

（5）为用户提供方便的用户接口。数据库管理系统作为用户与数据库的接口，提供数据库定义、运行、维护等功能。用户可方便地开发和使用数据库。

（6）对数据进行统一管理和控制，包括数据库的恢复、并发控制、数据安全性和数据完整性，从而可以保证数据库中的数据是安全的、正确的和可靠的。

1.1.5　数据模型

模型，是现实世界特征的模拟与抽象。比如一组建筑规划沙盘，精致逼真的飞机航模，都是对现实生活中的事物的描述和抽象，见到这些就会让人们联想到现实世界中的实物。

数据模型（Data Model）也是一种模型，它是现实世界数据特征的抽象。由于计算机不可能直接处理现实世界中的具体事物，因此必须把具体事物转换成计算机能够处理的数据，即首先要数字化，要把现实世界中的人、事、物、概念用数据模型这个工具来抽象、表示和加工处理。数据模型是数据库中用来对现实世界进行抽象的工具，是数据库中用于提供信息表示和操作手段的形式构架，是现实世界的一种抽象模型。

数据模型按不同的应用层次分为 3 种类型，分别是概念数据模型（conceptual data model）、逻辑数据模型（logic data model）和物理数据模型（physical data model）。

概念数据模型又称概念模型，是一种面向客观世界、面向用户的模型，与具体的数据库管理系统无关，与具体的计算机平台无关。人们通常先将现实世界中的事物抽象到信息世界，建立所谓的“概念模型”，然后再将信息世界的模型映射到机器世界，将概念模型转换为计算机世界中的模型。因此，概念模型是从现实世界到机器世界的一个中间层次。

逻辑数据模型又称逻辑模型，是一种面向数据库系统的模型，它是概念模型到计算机之间的中间层次。概念模型只有在转换成逻辑模型之后才能在数据库中得以表示。目前，逻辑模型的种类很多，其中比较成熟的有：层次模型、网状模型、关系模型、面向对象模型等。

这 4 种数据模型的根本区别在于数据结构不同，即数据之间联系的表示方式不同。

（1）层次模型用“树结构”来表示数据之间的联系。

（2）网状模型是用“图结构”来表示数据之间的联系。

（3）关系模型是用“二维表”来表示数据之间的联系。

（4）面向对象模型是用“对象”来表示数据之间的联系。

物理数据模型又称物理模型，它是一种面向计算机物理表示的模型，此模型是数据模型在计算机上的物理结构表示。

数据模型通常由 3 部分组成，也称为数据模型的三大要素，分别是数据结构、数据操纵和完整性约束。

1.1.6　概念模型

概念模型是独立于计算机系统的数据模型，它完全不涉及信息在计算机系统中的表示，只是用来描述某个特定组织所关心的信息结构。概念模型用于建立信息世界的数据模型，强调其语义表达能力，概念应该简单、清晰，易于用户理解。它是现实世界的第一层抽象，是用户和数据库设计人员之间进行交流的工具。概念模型可以看成是现实世界到机器世界的一个过渡的中间层次。

概念模型有以下特点。

（1）真实性。概念模型是对现实世界的抽象和概括，它必须真实地反映现实世界中的事物及事物之间的联系。

（2）易理解性。概念模型是独立于机器的信息结构，容易被用户理解。设计人员可以用概念模型和不熟悉计算机的用户交换意见，使用户能积极参与数据库的设计工作，保证设计工作顺利进行。

（3）易修改性。应用环境和应用需求是经常改变的，概念模型应该容易修改和扩充。

（4）易转换性。概念模型应该容易向关系、网状、层次等各种数据模型进行转换。

概念模型中最著名的是实体联系模型（Entity Relationship Model，ER 模型）。实体联系模型是 P．P．Chen 于 1976 年提出的。这个模型直接从现实世界中抽象出实体类型及实体间联系，然后用实体联系图（E-R 图）表示数据模型。设计 E-R 图的方法称为 E-R 方法。E-R 图是设计概念模型的有效工具。下面先介绍一下有关的名词术语及 E-R 图。

1．实体

现实世界中客观存在并可相互区分的事物叫做实体。实体可以是一个具体的人或物，如王伟、汽车等；也可以是抽象的事件或概念，如购买一本图书。

2．属性

实体的某一特性称为属性。如学生实体有学号、姓名、年龄、性别、系等方面的属性。属性有“型”和“值”之分，“型”即为属性名，如姓名、年龄、性别是属性的型；“值”即为属性的具体内容，如（990001，张立，20，男，计算机）；这些属性值的集合表示了一个学生实体。

3．实体型

若干个属性的型组成的集合可以表示一个实体的类型，简称实体型。如学生（学号，姓名，年龄，性别，系）就是一个实体型。

4．实体集

同型实体的集合称为实体集。如所有的学生、所有的课程等。

5. 码

能唯一标识一个实体的属性或属性集称为实体的码。如学生的学号可以作为码，学生的姓名可能有重名，不能作为学生实体的码。

6. 域

属性值的取值范围称为该属性的域。如学号的域为 6 位整数，姓名的域为字符串集合，年龄的域为小于 40 的整数，性别的域为（男，女）。

7. 联系

在现实世界中，事物内部以及事物之间是有联系的，这些联系同样也要抽象和反映到信息世界中来。在信息世界中联系将被抽象为实体型内部的联系和实体型之间的联系。

实体内部的联系通常是指组成实体的各属性之间的联系；实体之间的联系通常是指不同实体集之间的联系。

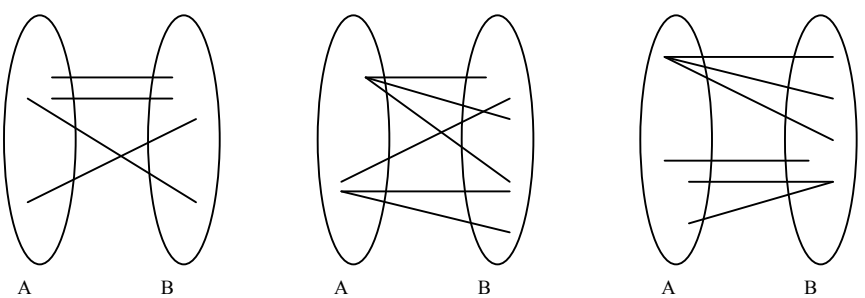
两个实体型之间的联系有如下 3 种类型。

（1）一对一联系（1∶1）。实体集 A 中的一个实体至多与实体集 B 中的一个实体相对应，反之亦然，则称实体集 A 与实体集 B 为一对一的联系。记作 1∶1。如：班级与班长，观众与座位，病人与床位。

（2）一对多联系（1∶*n*）。实体集 A 中的一个实体与实体集 B 中的多个实体相对应，而实体集 B 中的一个实体至多与实体集 A 中的一个实体相对应。记作 1∶*n*。如：班级与学生、公司与职员、省与市。

（3）多对多（*m*∶*n*）。实体集 A 中的一个实体与实体集 B 中的多个实体相对应，而实体集 B 中的一个实体与实体集 A 中的多个实体相对应。记作 *m*∶*n*。如：教师与学生,学生与课程，工厂与产品。

实际上，一对一联系是一对多联系的特例，而一对多联系又是多对多联系的特例。可以用图形来表示两个实体型之间的这 3 类联系，如图 1.9 所示。



（a）1∶1 联系　　（b）1∶*n* 联系　　（c）*m*∶*n* 联系

图 1.9　3 种联系示意图

在 E-R 图中有下面 4 个基本成分。

① 矩形框，表示实体类型（研究问题的对象）。

② 菱形框，表示联系类型（实体间的联系）。

③ 椭圆形框，表示实体类型和联系类型的属性。

相应的命名均记入各种框中。对于实体标识符的属性，在属性名下面画一条横线。

④ 直线，联系类型与其涉及的实体类型之间以直线连接，用来表示它们之间的联系，并在直线端部标注联系的种类（1∶1、1∶*n* 或 *m*∶*n*）。

下面通过例 1.1 说明设计 E-R 图的过程。

【例 1.1】为图书管理设计一个 E-R 模型。读者从图书馆借书，图书馆从出版社购书，E-R 图的具体建立过程如下。

① 首先确定实体类型。本问题有 3 个实体类型：读者、书、出版社。

② 确定联系类型。读者和书之间是 *m*∶*n* 联系，起名为“借阅”，书和出版社之间是 1∶*n* 联系，起名为“订购”。

③ 把实体类型和联系类型组合成 E-R 图。

④ 确定实体类型和联系类型的属性。实体类型读者的属性有：读者编号、读者姓名、读者年龄、性别、系别；实体类型书的属性有：书号、书名、作者、价格；实体类型出版社的属性有：出版社编号、出版社名、出版社地址。联系类型借阅的属性有借阅日期、归还日期。

⑤ 确定实体类型的键，在 E-R 图中，属于键的属性名下画一条横线。具体的 E-R 图如图 1.10 所示。

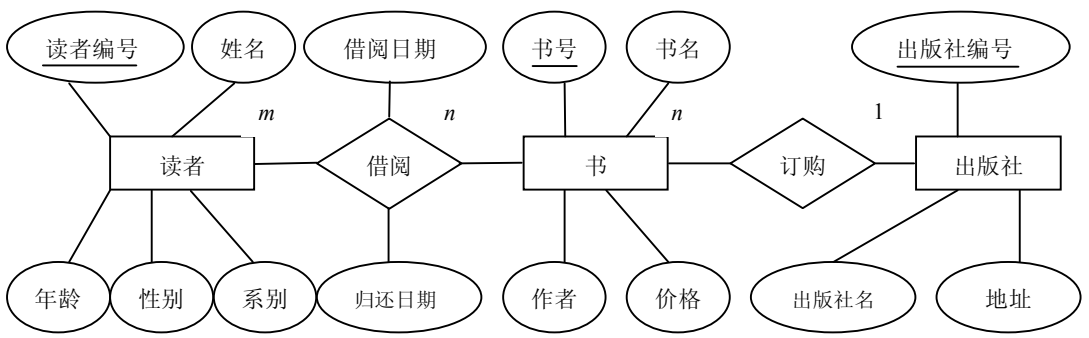


图 1.10　E-R 图实例

E-R 模型有两个明显的优点：一是接近于人的思维，容易理解；二是与计算机无关，用户容易接受。因此 E-R 模型已成为软件工程中的一个重要设计方法。但是 E-R 模型只能说明实体间语义的联系，还不能进一步说明详细的数据结构。一般遇到一个实际问题，总是先设计一个 E-R 模型，然后再把 E-R 模型转换成计算机已实现的数据模型。

1.2　关系数据库

1.2.1　关系数据库系统概述

关系数据库是因为采用关系模型而得名，它是目前数据库应用中的主流技术。

关系数据库之所以得到广泛应用，是因为它是建立在严格的数学理论基础上的，概念清晰、简单，能够用统一的结构来表示实体集合和它们之间的联系。从数据库的发展历程中可以看到，关系数据库的出现标志着数据库技术走向成熟。

关系数据库系统与非关系数据库系统的区别是，关系系统只有“表”这一种数据结构；而非关系数据库系统还有其他数据结构，对这些数据结构还有其他的操作。

本节首先介绍关系模型的基本概念，然后介绍关系的完整性以及关系代数。

1. 关系数据库的特点

关系数据库系统是基于关系模型的数据库系统，20 世纪 70 年代末以后所问世的数据库产品大多为关系模型，并逐渐替代网状模型、层次模型数据库系统而成为主流数据库系统。关系数据库系统的崛起并迅速在市场中站稳脚跟与它的优越性有关。关系数据库系统具有以下优点。

（1）数据结构简单。关系数据库系统采用统一的二维表作为数据结构，不存在复杂的内部联系，具有高度的简洁性与方便性。

（2）功能强。关系数据库系统能直接构造复杂的数据模型，特别是多联系间的联系表达，它可以一次得到一条完整记录，也可以修改数据间的联系，同时还具备一定程度的修改数据模式的能力。此外，路径选择的灵活性、存储结构的简单性都是它的优点。

（3）使用方便。关系数据库系统数据结构简单，它的使用不涉及系统内部物理结构，用户不必了解，更无须干预内部组织，所用数据语言均为非过程性语言，因此操作、使用都很方便。

（4）数据独立性高。关系数据库系统的组织、使用由于不涉及物理存储因素，不涉及过程性因素，因此数据的物理独立性很高，数据的逻辑独立性也有一定的改善。

当然，关系数据库系统也存在一些不足之处，如它对事务处理领域应用效果较好，但对非事务性应用及分析领域的应用尚显不足等。

目前关系数据库系统已经成熟，其产品全方位向纵深方向发展，主要表现在如下几个方面。

（1）可移植性。目前的产品能同时适应多个操作系统，如 SQL SERVER 2000 能适应 70 多种操作系统。

（2）标准化。数据库语言的标准化工作经过多年的努力之后，目前以 SQL 为代表的结构化查询语言已陆续被美国标准化组织 ANSI、国际标准化组织 ISO 以及我国标准化组织确定为关系数据库使用的标准化语言，从而完成了其使用的统一性，这被称为是一次关系数据库领域的革命。而其中 SQL-92 又被认为是典型的关系数据库系统语言。

（3）开发工具。由于数据库在应用中大量使用，用户需要对它直接操作，这就要求数据库不仅有数据定义、操纵与控制等操作，还需要大量用户界面生成及开发的工具软件以利于用户开发应用。因此，自 20 世纪 80 年代以来，关系数据库所提供的软件还包括大量用户界面生成软件以及开发工具。如 ORACLE Developer-2000、Microsoft 公司的 Visual Basic 以及 PowerBuilder、Delphi 等。

（4）分布式功能。由于数据库在计算机网络上的大量应用以及数据共享的要求，数据库的分布式功能已在应用中成为迫切需要，因此目前多数关系数据库系统都提供此类功能，它们的方式有数据库远程访问、客户/服务器方式、浏览器/服务器方式。

（5）开放性。现代关系数据库系统大都具有较好的开放性，能与不同的数据库、不同的应用接口结合，并能扩充与发展。一般关系数据库系统都具有通用的 ODBC 与 JDBC 接口以及快速的专用接口。

2. 关系模型的基本术语

在关系模型中，用单一的二维表结构来表示实体及实体间的关系，如图 1.11 所示。

（1）关系。一个关系对应一个二维表，二维表名就是关系名。图 1.11 中包含两个二维表，

即两个关系：学生信息关系及选课信息关系。

关系名→学生信息表　　↙属性（字段、数据项）

学号	姓名	性别	年龄	←关系模式（记录类型）
101001	王军	男	24	
101003	黄明业	男	34	←元组（记录）
103018	张华	女	35	
104024	吴林华	女	27	

↑　↑　↑

主键　外键　属性值（字段值）

＼＼

参照表↓　选课信息表

学号	课程号	成绩
101001	001	75
101003	003	80
103018	005	85
104024	002	77

图 1.11　关系模型的基本术语

（2）属性及值域。二维表中的列（字段）称为关系的属性。属性的个数称为关系的元数，又称为度。度为 *n* 的关系称为 *n* 元关系，度为 1 的关系称为一元关系，度为 2 的关系称为二元关系。关系的属性包括属性名和属性值两部分，其列名即为属性名，列值即为属性值。属性值的取值范围称为值域，每一个属性对应一个值域，不同属性的值域可以相同。

图 1.11 中，学生信息关系中有学号、姓名、性别、年龄 4 个属性，是四元关系。其中性别属性的值域是“男”和“女”，年龄属性的值域是 18～65。选课信息关系中有学号、课程号、成绩 3 个属性，是三元关系。学号“101001”就是学号属性的一个值。

（3）关系模式。二维表中的行定义（表头）、记录的类型，即对关系的描述称为关系模式，关系模式的一般形式为：

关系名（属性 1，属性 2，…，属性 n）

图 1.11 中的两个关系模式表示为：

学生信息关系（学号，姓名，性别，年龄）

选课信息关系（学号，课程号，成绩）

（4）元组。二维表中的一行，即每一条记录的值称为关系的一个元组。其中，每一个属性的值称为元组的分量。关系由关系模式和元组的集合组成。

图 1.11 中学生信息关系有以下元组：

（101001，王军，男，24）

（103018，张华，女，35）

选课信息关系有以下元组：

（101001，001，75）

（101003，003，80）

（5）键（或码）。由一个或多个属性组成。在实际使用中，有下列几种键。

① 候选键（Candidate Key）：若关系中的某一属性组的值能唯一地标识一个元组，则称该属性组为候选键。

② 主键（Primary Key）：若一个关系有多个候选键，则选定其中一个为主键。主键中包含的属性称为主属性。不包含在任何候选键中的属性称为非键属性（Non-Key attribute）。关系模型的所有属性组是这个关系模式的候选键，称为全键（All-key）。

③ 外键（Foreign Key）：设 F 是关系 R 的一个或一组属性，但不是关系 R 的键。如果 F 与关系 S 的主键相对应，则称 F 是关系 R 的外键，关系 R 称为参照关系，关系 S 称为被参照关系或目标关系。

如图 1.11 所示，在学生信息关系中，学号就是主键，在选课信息关系中，（学号，课程号）为主键，而学号称为外键。

（6）主属性与非主属性。关系中包含在任何一个候选键中的属性称为主属性，不包含在任何一个候选键中的属性称为非主属性。图 1.11 中的学生信息关系中因为（学号），（姓名）是候选键，所以学号和姓名是主属性，其他属性是非主属性。

3. 关系的性质

我们用集合的观点定义关系。关系是笛卡尔积的子集。也就是说，把关系看成一个集合，集合中的元素是元组，每个元组的属性个数均相同。如果一个关系的元组个数是无限的，称为无限关系；反之，称为有限关系。

在关系模型中对关系做了一些规范性的限制，可通过二维表格形象地理解关系的性质。

（1）关系中每个属性值都是不可分解的，即关系的每个元组分量必须是原子的。从二维表的角度讲，不允许表中嵌套表。表 1.1 就出现了这种表中再嵌套表的情况，在“学时”下嵌套“讲课”和“实验”。虽然类似的表在实际生活中司空见惯，但却不符合关系的基本定义。因为关系是从域出发定义的，每个元组分量都是不可再分的，不可能出现表中套表的现象。遇到这种情况，可对表格进行简单的等价变换，使之成为符合规范的关系。例如，可把表 1.1 改成表 1.2。这里把“学时”分成两列——“理论学时”和“实验学时”，两个属性都取自同一个域“学时”。

表 1.1　　不符合规范的表

课　　程	学　　时	
	理　　论	实　　验
数据库原理	54	10
编译原理	40	10

表 1.2　　符合规范的表

课　　程	理 论 学 时	实 验 学 时
数据库原理	54	10
编译原理	40	10
操作系统	50	12

（2）关系中不允许出现相同的元组。从语义角度看，二维表中的一行即一个元组，代表着一个实体。现实生活中不可能出现完全一样、无法区分的两个实体，因此，二维表不允许出现相同的两行。同一关系中不能有两个相同的元组存在，否则将使关系中的元组失去唯一性，这一性质在关系模型中很重要。

（3）在定义一个关系模式时，可随意指定属性的排列次序，因为交换属性顺序的先后，

并不改变关系的实际意义。例如，在定义表 1.2 所示的关系模式时，可以指定属性的次序为（课程，理论学时，实验学时），也可以指定属性的次序为（课程，实验学时，理论学时）。

（4）在一个关系中，元组的排列次序可任意交换，并不改变关系的实际意义。由于关系是一个集合，因此不考虑元组间的顺序问题。在实际应用中，常常对关系中的元组排序，这样做仅仅为了加快检索数据的速度，提高数据处理的效率。

对性质（3）和性质（4），需要再补充一点。判断两个关系是否相等，是从集合的角度来考虑的与属性的次序无关，与元组次序无关，与关系的命名也无关。如果两个关系仅仅是上述差别，在其余各方面完全相同，就认为这两个关系相等。

（5）关系模式相对稳定，关系却随着时间的推移不断变化。这是由数据库的更新操作（包括插入、删除、修改）引起的。

1.2.2　关系模式

关系模式是对关系的描述。关系模式是型，而关系是值。定义关系模式必须指明：

（1）元组集合的结构包括属性构成、属性来自的域、属性与域之间的映像关系；

（2）元组语义以及完整性约束条件；

（3）属性间的数据依赖关系集合。

关系模式可以形式化地表示为

R(U, D, dom, F)

R：　　关系名；

U：　　组成该关系的属性名集合；

D：　　属性组 U 中属性所属的域；

dom：　　属性向域的映像集合；

F：　　属性间的数据依赖关系集合。

关系模式通常可以简记为 R（U）或 R（A_1，A_2，…，A_n），其中：R 为关系名，A_1，A_2，…，A_n 为属性名。

在 1.2.1 节中详细介绍了关系的概念。关系实际上是关系模式在某一时刻的状态或内容。也就是说，关系模式是型，关系是它的值。关系模式是静态的、稳定的，而关系是动态的、随时间不断变化的，因为关系操作在不断地更新着数据库中的数据。但在实际应用中，常常把关系模式和关系统称为关系，读者可以从上下文中加以区别。

1.2.3　关系的完整性

关系模型的完整性规则是对关系的某种约束条件。关系模型中可以有 3 类完整性约束：实体完整性、参照完整性和用户定义的完整性。

1. 实体完整性（Entity Integrity）

一个基本关系通常对应现实世界的一个实体集，如学生关系对应于学生的集合。现实世界中的实体是可区分的，即它们具有某种唯一性标识。相应的，关系模型中以主键作为唯一性标识。主键中的属性即主属性，不能取空值。所谓空值就是“不知道”或“无意义”的值。如果主属性取空值，就说明存在某个不可标识的实体，即存在不可区分的实体，这与现实世界的应用环境相矛盾，因此这个实体一定不是一个完整的实体。

实体完整性规则：若属性 A 是基本关系 R 的主属性，则属性 A 不能取空值。

2. 参照完整性（Referential integrity）

现实世界中的实体之间往往存在某种联系，在关系模型中实体及实体间的联系都是用关系来描述的。这样就自然存在着关系与关系间的引用。

设 F 是基本关系 R 的一个或一组属性，但不是关系 R 的键，如果 F 与基本关系 S 的主键 Ks 相对应，则称 F 是基本关系 R 的外键（Foreign key），并称基本关系 R 为参照关系（Referencing relation），基本关系 S 为被参照关系（Referenced relation）或目标关系（Target relation）。关系 R 和 S 不一定是不同的关系。

参照完整性规则就是定义外码与主码之间的引用规则。

参照完整性规则：若属性（或属性组）F 是基本关系 R 的外键，它与基本关系 S 的主键 Ks 相对应（基本关系 R 和 S 不一定是不同的关系），则对于 R 中每个元组在 F 上的值必须为：

- 或者取空值（F 的每个属性值均为空值）；
- 或者等于 S 中某个元组的主键值。

【例 1.2】下面各种情况说明了参照完整性规则在关系中如何实现的。在关系数据库中有下列两个关系模式。

学生关系模式：S（学号，姓名，性别，年龄，班级号，系别），PK（学号）

学习关系模式：SC（学号，课程号，成绩），PK（学号，课程号），FK1（学号），FK2（课程号）

根据规则要求，关系 SC 中的“学号”值应该在关系 S 中出现。如果关系 SC 中有一个元组（S07，C04，80），而学号 S07 却在关系 S 中找不到，那么就认为在关系 SC 中引用了一个不存在的学生实体，这违反了参照完整性规则。另外，在关系 SC 中“学号”不仅是外键，也是主键的一部分，因此这里“学号”值不允许空。

3. 用户定义的完整性（User-defined integrity）

实体完整性和参照性适用于任何关系数据库系统。除此之外，不同的关系数据库系统根据其应用环境的不同，往往还需要一些特殊的约束条件。

用户定义的完整性就是针对某一具体关系数据库的约束条件，它反映某一具体应用所涉及的数据必须满足的语义要求。关系模型应提供定义和检验这类完整性的机制，以便用统一的系统的方法处理，而不是由应用程序承担这一功能。

【例 1.3】例 1.2 中的学生关系模式 S 中，学生的年龄定义为两位整数，但范围仍然太大，为此用户可以写出如下规则把年龄限制在 15～30 岁之间：

CHECK（AGE BETWEEN 15 AND 30）

1.2.4 关系数据库语言 SQL

关系数据库语言 SQL（Structured Query Language），又称为结构化查询语言，是关系数据库管理系统中最流行的数据查询和操作语言，用户可以使用 SQL 语言对数据库执行各种操作，包括数据定义、数据操纵和数据控制等与数据库有关的全部功能。

SQL 语言是在 1974 年由美国 IBM 公司的 San Jose 研究所中的科研人员 Boyce 和 Chamberlin 提出的，并于 1975～1979 年在关系数据库管理系统原型 System R 上实现了这种语言。1986 年 10 月，美国国家标准局（American National Standards Institute，ANSI）的数据

库委员会批准了 SQL 作为关系数据库语言的美国标准，同年公布了 SQL 标准文本 SQL-86。1987 年国际标准化组织（International Standards Organization，ISO）将其采纳为国际标准。1989 年公布了 SQL-89，1992 年又公布了 SQL-92（也称为 SQL2）。1999 年颁布了反 映最新数据库理论和技术的标准 SQL-99（也称为 SQL3）。

由于 SQL 语言具有功能丰富、简洁易学、使用方式灵活等突出优点，因而倍受计算机工业界和计算机用户的欢迎。尤其自 SQL 成为国际标准后，各数据库管理系统厂商纷纷推出支持 SQL 或与 SQL 接口的软件。这就使得大多数数据库均采用了 SQL 作为数据存取语言和标准接口。

但是，不同的数据库管理系统厂商开发的 SQL 并不完全相同。这些不同类型的 SQL 一方面遵循了标准 SQL 语言规定的基本操作，另一方面又在标准 SQL 语言的基础上进行了扩展，增强了功能。不同厂商的 SQL 有不同的名称，例如，Oracle 产品中的 SQL 称为 PL/SQL，Microsoft SQL Server 产品中的 SQL 称为 Transact-SQL。

1. SQL 的主要功能

SQL 的功能可以分为 3 类。

（1）数据定义功能。SQL 的数据定义功能通过数据定义语言（Data Definition Language，DDL）实现。它用来定义数据库的逻辑结构，包括基本表、视图和索引。基本的 DDL 包括 3 类，即定义、修改和删除。

（2）数据操纵功能。SQL 的数据操纵功能通过数据操纵语言（Data Manipulation Language，DML）实现。它包括数据查询和数据更新两大类操作，其中数据查询是指对数据库中的数据进行查询、统计、分组、排序等操作；数据更新包括插入、删除和修改 3 种操作。

（3）数据控制功能。数据库的控制是指数据库的安全性和完整性控制。

SQL 的数据控制功能通过数据控制语言（Data Control Language，DCL）实现，它包括对基本表和视图的授权，完整性规则的描述以及事务开始和结束等控制语句。

SQL 通过对数据库用户的授权和取消授权命令来实现相关数据的存取控制，以保证数据库的安全性。另外还提供了数据完整性约束条件的定义和检查机制，来保证数据库的完整性。

2. SQL 的特点

SQL 语言集数据查询、数据操纵、数据定义和数据控制功能于一体，语言风格统一。使用 SQL 语句就可以独立完成数据管理的核心操作。SQL 还是高度非过程化的，在对数据库进行存取操作时无需了解存取路径，大大减轻了用户负担，也有利于提高数据的独立性。另外，SQL 语言采用集合操作方式，其操作对象、操作结果均可以是元组的集合。

SQL 语言除上述特点外，还具有下列 3 个特点。

（1）SQL 具有交互式和嵌入式两种形式。交互式 SQL 能够独立地用于联机交互，直接键入 SQL 命令就可以对数据库进行操作。

嵌入式 SQL 能够嵌入到高级语言（如 C，COBOL，FORTRAN，PASCAL，PL / 1）程序中，来实现对数据库的存取操作。

无论是哪种使用方式，SQL 语言的语法结构基本一致。这种统一的语法结构的特点，为使用 SQL 提供了极大的灵活性和方便性。

（2）SQL 具有语言简洁、易学易用的特点。虽然 SQL 的语言功能极强，但其语言十分

简洁，只用了 9 个动词就完成了其核心功能。SQL 的命令动词及其功能如表 1.3 所示。另外，SQL 语言的语法简单，与英语口语的风格类似，易学易用。

表 1.3　　SQL 的命令动词

SQL 的功能	命令动词
数据定义	CREATE，DROP，ALTER
数据操纵	SELECT，INSERT，UPDATE，DELETE
数据控制	GRANT，REVOKE

（3）SQL 支持三级模式结构。SQL 支持关系数据库的三级模式结构，如图 1.12 所示。

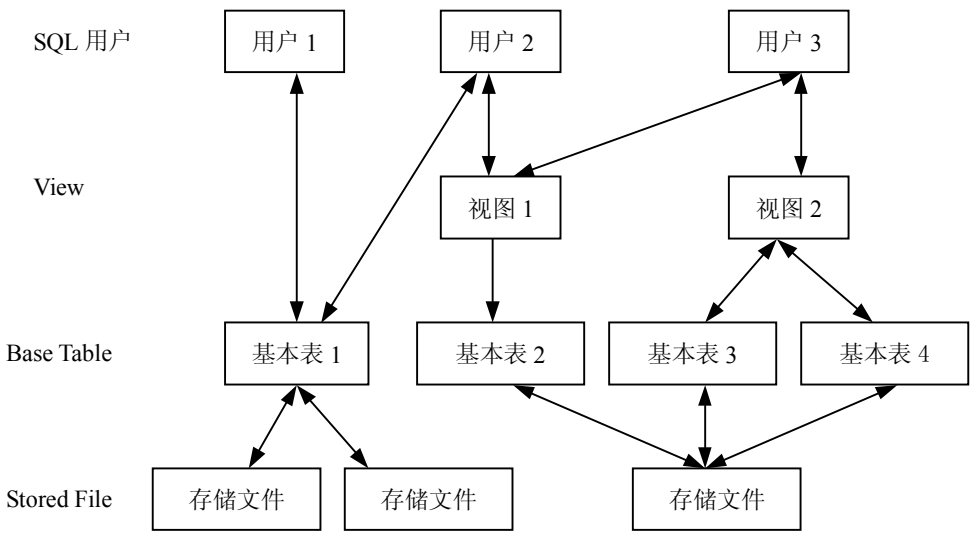


图 1.12　SQL 对关系数据库模式的支持（SQL 数据库的体系结构）

① 全体基本表（Base Table）构成了数据库的模式。基本表是本身独立存在的表，在 SQL 中一个关系就对应一个基本表。

② 视图（View）和部分基本表构成了数据库的外模式。视图是从基本表或其他视图中导出的表，它本身不独立存储在数据库中，即数据库中只存放视图的定义而不存放视图对应的数据，这些数据仍存放在导出视图的基本表中，因此视图是一个虚表。

用户可以用 SQL 语句对视图和基本表进行查询等操作。在用户看来，视图和基本表是一样的，都是关系。

视图是根据用户的需求设计的，这些视图再加上某些被用户直接使用的基本表就构成了关系数据库的外模式。SQL 支持关系数据库的外模式结构。

③ 数据库的存储文件（Stored File）和它们的索引文件构成了关系数据库的内模式。在 SQL 中，一个关系对应一个表，一个或多个基本表对应一个存储文件，一个基本表也可以对应多个存储文件，一个表可以带若干索引，索引也存放在存储文件中。每个存储文件与外部存储器上的一个物理文件对应。存储文件的逻辑结构组成了关系数据库的内模式。

关于 SQL 语言的具体操作将在后续章节中陆续介绍。

1.3　数据库的设计

有人说：一个成功的管理信息系统，是由 50%的业务+50%的软件所组成，而成功软件所占的 50%又由 25%的数据库+25%的程序所组成，笔者认为非常有道理。因此，要开发管理信息系统，数据库设计的好坏是关键。

数据库设计是指在给定的环境下，创建一个性能良好，能满足不同用户使用要求，又能被选定的 DBMS 所接受的数据模式。

从本质上讲，数据库设计乃是将数据库系统与现实世界相结合的一种过程。

人们总是力求设计出的数据库好用，但是设计数据库时既要考虑数据库的框架和数据结构，又要考虑应用程序存取数据库和处理数据。因此，最佳设计不可能一蹴而就，只能是一个反复探寻的过程。

大体上可以把数据库设计划分成以下几个阶段：需求分析阶段、概念结构设计阶段、逻辑结构设计阶段、数据库物理结构设计阶段、数据库实施阶段、数据库运行和维护阶段。如图 1.13 所示。

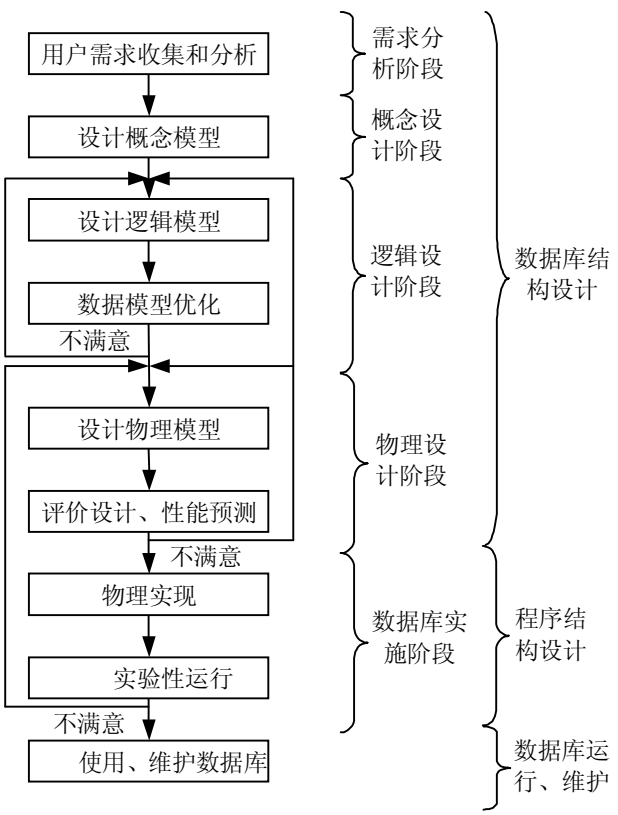


图 1.13　数据设计流程图

下面详细介绍数据库设计过程。

1.3.1 需求分析

准确地搞清楚用户需求，乃是数据库设计的关键。需求分析的好坏，决定了数据库设计的成败。

确定用户的最终需求其实是一件很困难的事。这是因为一方面用户缺少计算机知识，开始时无法确定计算机究竟能为自己做什么，不能做什么，因此无法一下子准确地表达自己的需求，他们所提出的需求往往不断地变化。另一方面设计人员缺少用户的专业知识，不易理解用户的真正需求，甚至误解用户的需求。此外新的硬件、软件技术的出现也会使用户需求发生变化。因此设计人员必须与用户不断深入地进行交流，才能逐步确定用户的实际需求。

需求分析阶段的成果是系统需求说明书，主要包括数据流图、数据字典、各种说明性表格、统计输出表、系统功能结构图等。系统需求说明书是以后设计、开发、测试和验收等过程的重要依据。

需求分析的任务是通过详细调查现实世界要处理的对象（组织、部门、企业等），充分了解原系统（手工系统或计算机系统）工作概况，明确用户的各种需求，在此基础上确定新系统的功能。新系统必须充分考虑今后可能的扩充和改变，不能仅仅按当前应用的需求来设计数据库。

需求分析的重点是调查、收集与分析用户在数据管理中的信息要求、处理要求、安全性与完整性要求。

需求分析阶段的主要任务有以下几个方面。

（1）确认系统的设计范围，调查信息需求，收集数据。分析需求调查得到的资料，明确计算机应当处理和能够处理的范围，确定新系统应具备的功能。

（2）综合各种信息包含的数据，各种数据间的关系，数据的类型、取值范围和流向。

（3）建立需求说明文档、数据字典、数据流程图。将需求调查文档化，文档既要为用户所理解，又要方便数据库的概念结构设计。需求分析的结果应及时与用户进行交流，反复修改，直到得到用户的认可。在数据库设计中，数据需求分析是对有关信息系统现有数据及数据间联系的收集和处理，当然也要适当考虑系统在将来的需求。一般需求分析包括数据流分析及功能分析。功能分析是指系统如何得到事务活动所需要的数据，在事务处理中如何使用这些数据进行处理（也叫加工），以及处理后数据流向的全过程的分析。换言之，功能分析是对所建数据模型支持的系统事务处理的分析。

数据流分析是对事务处理所需的原始数据的收集以及处理后所得数据及其流向，一般用数据流程图（DRP）来表示。在需求分析阶段，应当用文档形式整理出整个系统所涉及的数据、数据间的依赖关系、事务处理的说明和所需产生的报告，并且尽量借助于数据字典加以说明。除了使用数据流程图、数据字典，需求分析还可使用判定表、判定树等工具。

1.3.2 概念结构设计

概念结构设计是数据库设计的第二阶段，其目标是对需求说明书提供的所有数据和处理要求进行抽象与综合处理，按一定的方法构造反映用户环境的数据及其相互联系的概念模型，即用户数据模型或企业数据模型。这种概念数据模型与 DBMS 无关，是面向现实世界的数据模型，极易为用户所理解。为保证所设计的概念数据模型能正确、完全地反映用户（一个单

位）的数据及其相互联系，便于进行所要求的各种处理，在本阶段设计中可吸收用户参与和评议设计。在进行概念结构设计时，可设计各个应用的视图（View），即各个应用所看到的数据及其结构，然后再进行视图集成（View Integration），以形成一个单位的概念数据模型。形成的初步数据模型还要经过数据库设计者和用户的审查和修改，最后才能形成所需的概念数据模型。

1.3.3　逻辑结构设计

逻辑结构设计阶段的设计目标是把上一阶段得到的不被 DBMS 理解的概念数据模型转换成等价的，并为某个特定的 DBMS 所接受的逻辑模型所表示的概念模式，同时将概念结构设计阶段得到的应用视图转换成外部模式，即特定 DBMS 下的应用视图。在转换过程中要进一步落实需求说明，并使其满足 DBMS 的各种限制。逻辑结构设计阶段的结果是 DBMS 提供的数据定义语言（DDL）写成的数据模式。逻辑结构设计的具体方法与 DBMS 的逻辑数据模型有关。

1.3.4　物理结构设计

物理结构设计阶段的任务是把逻辑结构设计阶段得到的逻辑数据库在物理上加以实现。其主要内容是根据 DBMS 提供的各种手段，设计数据的存储形式和存取路径，如文件结构、索引的设计等，即设计数据库的内模式或存储模式。数据库的内模式对数据库的性能影响很大，应根据处理需求及 DBMS、操作系统和硬件的性能进行精心设计。

1.3.5　数据库的实施

数据库实施主要包括以下工作：

- 用 DDL 定义数据库结构；
- 组织数据入库；
- 编制与调试应用程序；
- 数据库试运行。

1. 定义数据库结构

确定了数据库的逻辑结构与物理结构后，就可以用选好的 DBMS 提供的数据定义语言（DDL）来严格描述数据库结构。

2. 数据装载

数据库结构建立好后，就可以向数据库中装载数据了。组织数据入库是数据库实施阶段最主要的工作。对于数据量不大的小型系统，可以用人工方式完成数据入库，其步骤如下。

（1）筛选数据。需要装入数据库中的数据通常都分散在各个部门的数据文件或原始凭证中，所以首先必须把需要入库的数据筛选出来。

（2）转换数据格式。筛选出来的需要入库的数据，其格式往往不符合数据库要求，还需要进行转换。这种转换有时可能很复杂。

（3）输入数据。将转换好的数据输入计算机中。

（4）校验数据。检查输入的数据是否有误。

对于大型系统，由于数据量大，用人工方式组织数据入库将会耗费大量人力物力，而且

很难保证数据的正确性。因此应该设计一个数据输入子系统由计算机辅助数据入库工作。

3. 编制与调试应用程序

数据库应用程序的设计应该与数据入库并行进行。在数据库实施阶段，当数据库结构建立好后，就可以开始编制与调试数据库的应用程序。调试应用程序时由于数据入库尚未完成，可先使用模拟数据。

4. 数据库试运行

应用程序调试完成，并且已有小部分数据入库后，就可以开始数据库的试运行。数据库试运行也称为联合调试，其主要工作包括以下两项内容。

（1）功能测试。实际运行应用程序，执行对数据库的各种操作，测试应用程序的各种功能。

（2）性能测试。测量系统的性能指标，分析是否符合设计目标。

1.3.6 数据库的运行和维护

数据库试运行结果符合设计目标后，数据库就可以真正投入运行了。数据库投入运行标志着开发任务的基本完成和维护工作的开始，但并不意味着设计过程的终结。由于应用环境在不断变化，数据库运行过程中物理存储也会不断变化，对数据库设计进行评价、调整、修改等维护工作是一个长期的任务，也是设计工作的继续和提高。

在数据库运行阶段，对数据库经常性的维护工作主要是由 DBA 完成的。维护工作包括：故障维护，数据库的安全性、完整性控制，数据库性能的监督、分析和改进，数据库的重组织和重构造。

本 章 小 结

本章初步讲解了数据库的基本概念，并通过对数据管理技术发展状况的介绍，阐述了数据库技术产生和发展的背景，也说明了数据库系统的优点，同时对关系数据库以及数据库设计步骤作了一些介绍，为读者学习后续课程打下良好的理论基础。

本章介绍了数据库系统的组成，使读者了解数据库系统不仅是一个计算机系统，而且是一个人机系统，人的作用特别是 DBA 的作用尤为重要。同时阐述了数据库系统的作用，以及数据库管理系统的组成。

数据库系统三级模式和二层映像的系统结构保证了数据库系统中能够具有较高的逻辑独立性和物理独立性。

本章的新概念较多，在学习过程中要注意理解，在学习后续章节时，可重新对这些概念作进一步的理解。

习　　题

一、选择题

1．数据模型有 3 个要素，其中用于描述系统静态特性的是（　　）。

A．数据结构　　B．数据操作

C．数据完整性约束　　D．数据模型

2．用树形结构来表示实体之间关系的结构数据模型称为（　　）。

A．关系模型　　B．层次模型

C．网状模型　　D．面向对象模型

3．下列实体类型的关联中，属于一对多关系的是（　　）。

A．学生与课程的选课关系　　B．部门与职工的关系

C．省与省会的关系　　D．顾客与商品的购买关系

二、填空题

1．数据库设计分别是：_____、_____、_____、_____、_____、_____。

2．数据独立性分为_____和_____两级。

3．包含在任何一个候选码中的属性叫做_____，不包含在任何候选码中的属性叫做_____。

三、简答题

1．数据库技术的发展经历了哪几个阶段？各个阶段的特点是什么？

2．阐述数据、数据库、数据库管理系统、数据库系统的概念。

3．简述使用数据库系统的优点。

第 2 章

SQL Server 2005 概述

本章主要介绍 SQL Server 2005 的特性、安装、启动及退出方法，并详细介绍如何配置 SQL Server 2005 服务器，以及如何配置 SQL Server 2005 网络环境等内容。通过本章的学习可以了解 SQL Server 2005 主要技术、新特性、新增功能；掌握安装 SQL Server 2005 的软硬件要求，安装过程及主要实用工具的使用。

2.1 SQL Server 2005 简介

2.1.1 SQL Server 2005 概述

SQL Server 2005 是一个数据库平台，用集成的商业智能工具提供企业级的数据管理。SQL Server 2005 数据库引擎为关系型数据和结构化数据提供了更安全可靠的存储功能，使用户可以构建和管理用于业务的，高可用性和高性能的数据应用程序。

Microsoft SQL Server 2005 完全重新定义了 SQL Server 的数据库平台，为小型、中型和大型机构建立其下一代 IT 基础结构提供了基石。SQL Server 2005 的核心内容介绍如下。

（1）SQL Server 数据库服务。包括核心数据库、复制和全文搜索组件。核心数据库、数据库引擎是 SQL Server 的心脏。复制通过跨越多个数据库分发数据，提高了数据的可用性，允许跨越指定的数据库服务器扩大读取的数据量。全文搜索允许用简明的语言查询存储在 SQL Server 数据表中的数据。

（2）分析服务。SQL Server 2005 为商业智能应用程序提供联机分析处理（OLAP）和数据挖掘功能。分析服务允许机构从多个数据源聚集数据（如关系数据库），并且以广泛多变的方式使用这些数据。

（3）数据集成服务。从多个数据源提取和转换数据，并移动到一个或多个目标源，提供企业数据转换和集成解决方案。这样就允许用户从异构的数据源合并数据，载入数据到数据仓库和数据市场等。

（4）通知服务。包括一个通知引擎和客户端组件，使得当一个触发事件发生的时候，产生和发送个性化、适时的信息给用户。通知能发送到无线设备，如移动电话、个人数字助理、Windows Messenger 账户和电子邮件账户。

（5）报表服务。包括报表管理器和报表服务器，提供一个创建、管理和分发报表功能的

完全的、基于服务器的平台。报表服务器是建立于标准的 IIS 和.NET 框架技术之上，允许结合 SQL Server 和 IIS 的优点来承载和处理报表。

（6）Service Broker。提供可靠的队列排序和消息传递作为数据库的一个核心组成部分。队列能用于堆栈工作（例如查询和其他请求），以及作为允许的资源来执行它们。消息传递允许数据库应用程序之间相互通信。

2.1.2　SQL Server 2005 新增强功能

在介绍 SQL Server 2005 版本的新特性之前，有必要介绍 SQL Server 的发展过程。Microsoft SQL Server（在本书中简称为 SQL Server）最初起源于 Sybase 的 SQL Server（简称 Sybase）。1989 年，Microsoft 公司与 Sybase 公司联合开发了一个用于 OS/2 版的 SQL Server。1993 年，他们将 4.2 版的 SQL Server 移植到 Windows NT 上。6.0 版推出后，这种合作关系解除了。6.5 及以后版本的 SQL Server 已经成为 Microsoft 公司的专有产品，并首次包含了“复制”的内容，Microsoft 公司通常将这一版本称为 SQL Server 变革历程中的第一代产品。在 7.0 版中，Microsoft 公司对该产品进行了彻底重写，并取得了高度成功，从而使该产品成为第一个可以用于 Windows 9x 的数据库产品（目前，SQL Server 中几乎没有保留 Sybase 的原有代码），而且重新架构了关系型服务器，实现了广泛的自动资源管理。在 2000 版中，Microsoft 公司修补了 7.0 版存在的漏洞，并增加了许多新的功能：改进了系统性能并提高了系统可伸缩性，支持 XML 语言编程功能等，支持联机分析处理（OLAP），数据压缩、转换和加载（Extract/Transformation/Load，ETL），这是 SQL Server 变革历程中的第二代产品。针对以上几个版本的缺陷与不足，Microsoft 公司历时多年，最终于 2005 年 11 月推出了 SQL Server 变革历程中的第三代产品，这就是今天的 SQL Server 2005。在这一代产品中，Microsoft 公司在开发时特别注重其高可用性、安全性，并致力于提高开发人员的效率。

SQL Server 2005 在原有 SQL Server 2000 系统的基础上增加了一些新的功能和特性，主要表现在企业级数据库管理、开发人员能力和商业智能方面。

1．企业级数据库管理

在当今的网络世界中，数据和管理数据的系统必须始终为用户可用且能够确保安全，有了 SQL Server 2005，用户和 IT 专家将从减少应用程序宕机时间、提高性能及可伸缩性、更紧密的安全控制中获益。SQL Server 2005 还提供了很多新的和改进的功能来帮助企业的 IT 团队更高效的工作。SQL Server 2005 在企业级数据管理中有以下几个关键方面的增强：

- 易管理；
- 可用性；
- 可伸缩性；
- 安全性。

（1）易管理。SQL Server 2005 能够更为简单地部署、管理和优化企业数据和分析应用程序。作为一个企业数据管理平台，SQL Server 2005 提供了唯一的管理控制台，使得数据管理人员能够在组织内的任何地方监视、管理和调控企业中所有的数据库和相关的服务。它还提供了一个可扩展的管理架构，可以更容易地用 SQL 管理对象（SMO）来编程，使得用户可以定制和扩展他们的管理环境，独立软件开发商（ISV）也能够创建附加的工具和功能来更好地扩展应用。

（2）可用性。SQL Server 2005 在高可用技术、额外的备份和恢复功能，以及复制增强上的投入使企业能够构建和部署高可用的应用系统。SQL Server 2005 在高可用上的创新有：数据镜像，故障转移集群，数据库快照和增强的联机操作，这有助于最小化宕机时间和确保企业的关键系统可用。后续章节将更为详细地介绍这些增强特性。

（3）可伸缩性。SQL Server 2005 提供了诸如表分区、快照隔离、64 位支持等方面的高级可伸缩性功能，使用户能够使用 SQL Server 2005 构建和部署最关键的应用。表和索引的分区功能显著地增强了对大型数据库的查询性能。

（4）安全性。SQL Server 2005 在数据库平台的安全模型上有了显著的增强。由于提供了更为精确和灵活的控制，数据安全更为严格。为了给企业数据提供更高级别的安全，Microsoft 公司做了相当大的投入，实现了如下特性：

- 在认证空间里强制 SQL Server login 密码策略；
- 在认证空间里可根据不同的范围上指定的权限来提供更细的粒度；
- 在安全管理空间中允许分离所有者和模式（schema）。

2. 开发人员能力

SQL Server 2005 包含了多个能显著提高开发者能力的新技术。从支持.NET Framework 到和 Visual Studio 的紧密集成，这些新特性使开发人员能够以更低的成本，更容易地创建安全、强大的数据库应用程序。SQL Server 2005 提供了一个端到端的数据库开发环境，使开发人员能够更有效地利用其已有的开发技能。本机 XML 功能也使开发人员能够创建运行在不同平台或设备上的新型应用程序。

增强开发人员能力的新技术如下：

- 扩展的语言支持；
- 改进的开发工具；
- 可扩展能力；
- 改进的数据访问；
- XML 和 Web Services；
- 应用程序 Framework。

（1）扩展的语言支持。因为通用语言运行时，公共语言运行库（CLR）被集成在数据库引擎中，所以开发人员现在可以利用多种他们熟悉的语言来开发数据库应用程序，包括：Transact-SQL、Microsoft Visual Basic.NET、Microsoft Visual C#.NET。此外，通过使用用户定义类型和函数，CLR 集成也为开发人员提供了更多的灵活性。CLR 还为快速数据库应用开放提供了使用第三方代码的选择。

（2）改进的开发工具。开发人员现在能够用一个开发工具开发 Transact-SQL、XML、Multidimensional Expressions （MDX）和 XML for Analysis （XML/A）应用。与 Visual Studio 开发环境的集成也为关键业务应用和商业智能应用提供了更有效的开发和调试环境。

（3）可扩展性。主要包括以下几个方面：

- 用户定义类型和聚合；
- SQL 管理对象（SMO）；
- 分析管理对象。

（4）改进的数据访问和 Web Services。在 SQL Server 2005 中，用户可以开发数据库层的

XML Web Services，把 SQL Server 作为一个 HTTP Listener。这对那些以 Web Services 为中心的应用程序提供了新型的数据访问功能。在 SQL Server 2005 中，可以使用 HTTP 直接访问 SQL Server，无需使用 IIS 这样的中间层 Listener。SQL Server 开放了一个 Web Services 接口，可以执行 SQL 语句和调用函数及过程，查询结果可用 XML 格式返回，并且可以利用 Visual Studio 的 Web Services 架构。

3. 商业智能

SQL Server 2005 通过在可伸缩性、数据集成、开发工具和强大的分析等方面的革新，更好地确立了 Microsoft 公司在商业智能（BI）领域的领导地位。SQL Server 2005 能够把关键的信息及时地传递到组织内员工的手中，从而实现了可伸缩的商业智能。从 CEO 到信息工作者，员工可以快速地、容易地处理数据，从而更快更好地做出决策。SQL Server 2005 全面的集成、分析和报表功能使企业能够提高他们已有应用的价值，即便这些应用是在不同的平台上。

商业智能增强体现在以下几个方面：

- 端到端的集成 BI 平台；
- 集成服务；
- 分析服务；
- 报表服务；
- Microsoft Office System 的集成。

（1）端到端的集成 BI 平台。Microsoft SQL Server 2005 是一个完整的 BI 平台，为用户提供了可用于构建典型和创新的分析应用程序所需的各种特性、工具和功能。本书会简要介绍在构建分析应用程序时将要用到的一些工具，并着重介绍一些新增功能。这些新增功能使复杂 BI 系统的构建和管理比以往更加轻松。

（2）集成服务（Integration Services）。SQL Server 2005 带来了一个全新的企业级数据整合平台。此平台具有出色的 ETL（Extract、Transform、Load，数据抽取、转换和装载）和整合能力，使得组织机构能更加容易地管理来自于不同的关系型和非关系型数据源的数据。通过 SQL Server Integration Services（SSIS），组织机构能以整体的视角去考察它们的商业运营情况，从而具有竞争优势。

（3）分析服务（Analysis Services）。在 SQL Server 2005 中，分析服务（Analysis Services）第一次提供了一个统一和集成的商业数据视图，可被用做所有传统报表、OLAP 分析、关键绩效指标（KPI）记分卡和数据挖掘的基础。

（4）报表服务（Reporting Services）。SQL Server 2005 Reporting Services 扩展了 Microsoft 公司 BI 平台，以迎合那些需要访问商业数据的信息工作者。Reporting Services 是一个基于服务器的企业级报表环境，可借助 Web Services 进行管理。报表可以用不同的格式发布，并可带多种交互和打印选项。通过把报表作为更进一步的商业智能的数据源来分发，使得复杂的分析可被更多的用户所用。

（5）和 Microsoft Office System 的集成。Reporting Services 中的报表可运行在 Microsoft SharePoint Portal Server 和 Microsoft Office System 应用程序中，可以使用 SharePoint 中的特性来订阅报表，创建新的报表和分发报表，也可以在 Word 或 Excel 中以 HTML 格式打开报表。

2.2 SQL Server 2005 的安装

2.2.1 SQL Server 2005 的版本和组件

1. SQL Server 2005 的版本

SQL Server 2005 发布了 4 个主要版本：工作组版、标准版、企业版和开发版。所有这些版本都提供服务器端和工作站端的安装。服务器端的安装包括完全版本的 SQL Server 和支持的服务。工作站端的安装包括客户端组件、工具和文档。

（1）SQL Server 2005 企业版（SQL Server 2005 Enterprise Edition）。该版本是针对大型企业的。它们需要更高的可用性以及更高级的功能和商业智能。例如，在这个版本中对 CPU 和内存数量没有限制。只要操作系统能够处理足够多的 CPU 和内存数量即可。该版本的估计零售价（只限美国地区）为 24 999 美元/CPU 或者是 13 499 美元/服务器（25 个 CAL，CAL 是 Cakewalk Application Language 的缩写，是 Cakewalk 软件支持的 MIDI 内容处理器）。Microsoft 仍然会继续支持开发人员版（Developer Edition），该版本可以让开发人员以更低廉的价格来开发 SQL Server 解决方案。开发人员版具有企业版的全部功能，但是只授权用于开发用途。大型的组织、公司一般采用企业版。企业版有如下特点。

- 无限扩展和分区，提供了非凡的性能来扩展 SQL Server 以支持大型的数据库安装。通过跨越多个服务器的水平分区表，使得可以配置一个服务器组共同工作以支持大型的网站或企业数据处理。
- 高级数据镜像，用以完成联机并行操作；高级分析工具，用以数据挖掘和全功能的 OLAP。
- 故障转移群集，允许在 Windows 2000 数据中心服务器上创建 4 个节点的群集和在 Windows 2000 高级服务器上创建两个节点的群集。使用这个功能可提供故障转移和自动恢复的支持。

（2）SQL Server 2005 标准版（SQL Server 2005 Standard Edition）。该版本在 SQL Server 2005 中具有更多的价值。例如，用户现在可以在标准版中通过使用群集、数据库镜像以及集成的 64 位支持来创建一个具有高可用性的系统。上面提到的这些特性在 SQL Server 2000 中只有企业版才提供，这导致当初许多公司不得不购买企业版（实际上它们可能使用标准版就足够了）。和 SQL Server 2005 企业版一样，2005 标准版也对内存数量没有限制，因此只要操作系统和物理硬件支持，用户可以按照自己的需求来扩展它。不过，标准版最多支持 4 个 CPU。目前在美国地区，标准版本的估计零售价为 5 999 美元/CPU 或 2 799 美元/服务器。标准版有如下特点。

- 运行在多个版本的 Microsoft Windows 操作系统上，包括 Windows 2000、Windows XP 专业版和 Windows Server 2003。
- 支持无限制的数据库大小，无限制数量的内存，用于对称多线程处理的 4 个 CPU，完全的复制发布和全文搜索。
- 提供商业智能服务，包括分析服务、报表服务、通知服务和数据转换服务。

• 提供数据库镜像、数据挖掘和数据集成服务。

（3）SQL Server 2000 和 2005 工作组版（SQL Server 2000 and 2005 Workgroup Editions）。该版本针对中小型公司，它们只需要具有有限商业智能和报告服务的数据库服务器。目前的美国地区估计零售价为 3 899 美元/CPU 或 739 美元/服务器。工作组版最多支持两个 CPU，数据库大小则没有限制。在 SQL Server 2000 工作组版中，内存容量最多为 2 GB。在 SQL Server 2005 工作组版中，内存容量则最多为 3 GB。工作组版有如下特点。

• 运行在多个版本的 Microsoft Windows 操作系统上，包括 Windows 2000、Windows XP 专业版和 Windows Server 2003。


注意

运行在 Microsoft Windows 2000 上的 SQL Server 2005 的所有版本，必须安装 Service Pack 4（SP4）或以上的系统补丁。如果使用 Windows XP 专业版，则必须安装 Service Pack 1（SP1）或以上的系统补丁。对于运行在 Windows 2000 或 Windows XP 专业版上的 SQL Server 2005 的其他要求，请参考 SQL Server 2005 联机丛书。

• 支持无限的数据库大小，高达 3 GB 的内存，用于对称多线程处理的两个 CPU，限定复制发布和全文搜索。

• 启用日志传送，这允许 SQL Server 从一个服务器向另外的服务器发送事务日志。可以使用这个功能创建备份服务器。

（4）SQL Server 2005 精简版（SQL Server 2005 Express Edition）。该版本等同于 SQL Server 中的桌面版（Desktop Edition，MSDE），不过它进行了一些增强，例如，MSDE 不提供任何类型的管理工具，而在精简版中则提供了这些工具。此外，精简版还包括了导入和导出向导以及一系列其他增强。精简版是 SQL Server 针对小型应用而推出的免费版本。它最大支持 4 GB 的数据库大小。最重要的是，该版本中删除了查询监督器，使得更多的人可以同时查询实例。

2．SQL Server 2005 的组件

（1）服务器组件。SQL Server 2005 服务器组件如表 2.1 所示。

表 2.1　　服务器组件

服务器组件	说　明
SQL Server 数据库引擎	主要用于存储、处理和保护数据的核心服务，复制，全文搜索，以及用于管理关系数据和 XML 数据的工具
Analysis Services	包括用于创建和管理联机分析处理（OLAP），以及数据挖掘应用程序的工具
Reporting Services	包括用于创建、管理和部署表格报表、矩阵报表、图形报表，以及自由格式报表的服务器和客户端组件。Reporting Services 还是一个可用于开发报表应用程序的可扩展平台
Notification Services	Notification Services 是一个平台，用于开发和部署将个性化即时信息发送给各种设备上的用户的应用程序
Integration Services	是一组图形工具和可编程对象，用于移动、复制和转换数据

（2）客户端组件。SQL Server 2005 客户端组件如表 2.2 所示。

表 2.2　　客户端组件

客户端组件	说　明
连接组件	安装用于客户端和服务器之间通信的组件，以及用于 DB-Library、ODBC 和 OLE DB 的网络库

（3）管理工具。SQL Server 2005 管理工具如表 2.3 所示。

表 2.3　　管理工具

管 理 工 具	说　　明
SQL Server Management Studio	SQL Server Management studio（SSMS）是 Microsoft SQL Server 2005 中的新组件，这是一个用于访问、配置、管理和开发 SQL Server 的所有组件的集成环境。SSMS 将 SQL Server 早期版本中包含的企业管理器、查询分析器和分析管理器的功能组合到单一环境中，为不同层次的开发人员和管理人员提供 SQL Server 访问能力
SQL Server 配置管理器	SQL Server 配置管理器为 SQL Server 服务、服务器协议、客户端协议和客户端别名提供基本配置管理
SQL Server Profiler	SQL Server Profiler 提供了图形用户界面，用于监视数据库引擎实例或 Analysis Services 实例
数据库引擎优化顾问	数据库引擎优化顾问可以协助创建索引、索引视图和分区的最佳组合

（4）开发工具。SQL Server 2005 开发工具如表 2.4 所示。

表 2.4　　开发工具

开 发 工 具	说　　明
Business Intelligence Development Studio	是用于分析服务、报表服务和集成服务解决方案的集成开发环境

（5）文档和示例。SQL Server 2005 文档和示例如表 2.5 所示。

表 2.5　　文档和示例

文档和示例	说　　明
SQL Server 联机丛书	SQL Server 2005 的技术文档
SQL Server 示例	提供数据库引擎、分析服务、报表服务和集成服务的示例代码和示例应用程序

2.2.2　安装 SQL Server 2005 的软、硬件要求

以下部分列出了运行 Microsoft SQL Server 2005 的最低硬件和软件要求。在 32 位平台上运行 SQL Server 2005 的要求与在 64 位平台上的要求有所不同。

1．*硬件和软件要求（32 位和 64 位）*

（1）监视器。SQL Server 图形工具需要 VGA（Video Graphic Array，显示绘图阵列）或更高分辨率：分辨率至少为 1 024×768 像素。

（2）定点设备。需要 Microsoft 鼠标或兼容定点设备。

（3）CD 或 DVD 驱动器。通过 CD 或 DVD 媒体进行安装时需要相应的 CD 或 DVD 驱动器。

（4）群集硬件要求。在 32 位和 64 位平台上，支持 8 节点群集安装（Microsoft Windows Server 2003 支持的最大节点数量）。

（5）网络软件要求。64 位版本的 SQL Server 2005 的网络软件要求与 32 位版本的要求相同。Windows Server 2003、Windows XP 和 Windows 2000 都具有内置网络软件。

（6）Internet 要求。32 位版本和 64 位版本的 SQL Server 2005 的 Internet 要求相同。表 2.6 列出了 SQL Server 2005 的 Internet 要求。

表 2.6　　SQL Server 2005 的 Internet 要求

组　　件	要　　求
Internet 软件	所有 SQL Server 2005 的安装都需要 Microsoft Internet Explorer 6.0 SP1 或更高版本，因为 Microsoft 管理控制台 （MMC） 和 HTML 帮助需要它。只需 Internet Explorer 的最小安装即可满足要求，并且不要求 Internet Explorer 是默认浏览器 然而，如果只安装客户端组件且不需要连接到要求加密的服务器，则 Internet Explorer 4.01（Service Pack 2）即可满足要求
Internet 信息服务（IIS）	安装 Microsoft SQL Server 2005 Reporting Services（SSRS）需要 IIS 5.0 或更高版本
ASP.NET 2.0	Reporting Services 需要 ASP.NET 2.0。安装 Reporting Services 时，如果尚未启用 ASP.NET，则 SQL Server 安装程序将启用它

2. 硬件和软件要求（32 位）

表 2.7 列出了在 32 位平台上安装和运行 SQL Server 2005 的硬件要求。

表 2.7　　32 位平台上 SQL Server 2005 的硬件要求

SQL Server 2005（32 位）	处理器类型	处理器速度	内存 （RAM）
SQL Server 2005 Enterprise Edition 4 SQL Server 2005 Developer Edition SQL Server 2005 Standard Edition	Pentium III 兼容处理器或更高速度的处理器	最低要求：600 MHz 推荐使用：1 GHz 或更高	最低要求：512 MB 推荐使用：1 GB 或更大
SQL Server 2005 Workgroup Edition	Pentium III 兼容处理器或更高速度的处理器	最低要求：600 MHz 推荐使用：1 GHz 或更高	最低要求：512 MB 推荐使用：1 GB 或更大
SQL Server 2005 Express Edition	Pentium III 兼容处理器或更高速度的处理器	最低要求：500 MHz 推荐使用：1 GHz 或更高	最低要求：192 MB 推荐使用：512 MB 或更高
具有高级服务的 SQL Server 2005 Express Edition	Pentium III 兼容处理器或更高速度的处理器	最低要求：600 MHz 推荐使用：1 GHz 或更高	最低要求：512 MB 推荐使用：1 GB 或更大

3. 硬盘空间要求（32 位和 64 位）

在安装 SQL Server 2005 的过程中，安装程序会在系统驱动器中创建临时文件。在运行安装程序之前，请验证系统驱动器中是否具有 2.0 GB 的可用磁盘空间来存储这些文件。即使在将所有 SQL Server 组件安装到非默认（系统）驱动器中时，此要求也是必需的。因为很多文件会安装到系统驱动器上，通常为驱动器 C:。

实际硬盘空间要求取决于系统配置和选择安装的应用程序和功能。表 2.8 列出了 SQL Server 2005 各组件对磁盘空间的要求。

表 2.8　　安装 SQL Server 2005 硬盘空间要求

功　　能	磁盘空间要求
数据库引擎和数据文件、复制以及全文搜索	280 MB
Analysis Services 和数据文件	90 MB
Reporting Services 和报表管理器	120 MB
Notification Services 引擎组件、客户端组件和规则组件	50 MB
Integration Services	120 MB
客户端组件	850 MB
SQL Server 联机丛书和 SQL Server 2005 Compact Edition 联机丛书	240 MB
示例和示例数据库。请注意，默认情况下不安装示例和示例数据库	410 MB

4. 操作系统要求（32 位）

表 2.9 所示为对于每种 32 位版本的 SQL Server 2005 的操作系统是否可以运行其服务器软件。

表 2.9　　支持不同 SQL Server 2005 版本的操作系统

操 作 系 统	Enterprise Edition[1]	Developer Edition	Standard Edition	Workgroup Edition	Express Edition 和具有高级服务的 Express	Evaluation Edition
Windows 2000	否	否	否	否	否	否
Windows 2000 Professional Edition SP42, 4	否	是	是	是	是	是
Windows 2000 Server SP42	是	是	是	是	是	是
Windows 2000 Advanced Server SP42	是	是	是	是	是	是
Windows 2000 Datacenter Edition SP42	是	是	是	是	是	是
嵌入式 Windows XP	否	否	否	否	否	否
Windows XP Home Edition SP2	否	是	否	否	是	否
Windows XP Professional Edition SP24	否	是	是	是	是	是
Windows XP Media Edition SP2	否	是	是	是	是	是
Windows XP Tablet Edition SP2	否	是	是	是	是	是
Windows Server 2003 Server SP16	是	是	是	是	是	是
Windows Server 2003 Enterprise Edition SP16	是	是	是	是	是	是
Windows Server 2003 Datacenter Edition SP16	是	是	是	是	是	是
Windows Server 2003 Web Edition SP1	否	否	否	否	是	否
Windows Small Business Server 2003 Standard Edition SP1	是	是	是	是	是	是
Windows Small Business Server 2003 Premium Edition SP1	是	是	是	是	是	是
Windows Vista Starter Edition	否	否	否	否	否	否
Windows Vista Home Basic Edition	否	是[7]	否	否	是[8]	是[7]
Windows Vista Home Premium Edition	否	是[7]	否	否	是[8]	是[7]
Windows Vista Ultimate Edition	是[7]	是[7]	是[7]	是[7]	是[8]	是[7]
Windows Vista Business Edition	是[7]	是[7]	是[7]	是[7]	是[8]	是[7]
Windows Vista Enterprise Edition	是[7]	是[7]	是[7]	是[7]	是[8]	是[7]
Windows Server 2003 64-Bit Itanium Datacenter Edition SP1	否	否	否	否	否	否
Windows Server 2003 64-Bit Itanium Enterprise Edition SP1	否	否	否	否	否	否
Windows Server 2003 64-Bit x64 Standard Edition SP16	WOW64[3]	WOW64[3]	WOW64[3]	WOW64[3]	WOW64[3]	WOW64[3]
Windows Server 2003 64-Bit x64 Datacenter Edition SP16	WOW64[3]	WOW64[3]	WOW64[3]	WOW64[3]	WOW64[3]	WOW64[3]

续表

操 作 系 统	Enterprise Edition[1]	Developer Edition	Standard Edition	Workgroup Edition	Express Edition 和具有高级服务的 Express	Evaluation Edition
Windows Server 2003 64-Bit x64 Enterprise Edition SP16	WOW64[3]	WOW64[3]	WOW64[3]	WOW64[3]	WOW64[3]	WOW64[3]
Windows XP x64 Professional 2003	否	WOW64[3]	WOW64[3]	WOW64[3]	WOW64[3]	否
Windows Vista Home Basic 64-Bit x64 Edition	否	WOW64[3, 7]	否	否	WOW64[3, 8]	WOW64[3,7]
Windows Vista Home Premium 64-Bit x64 Edition	否	WOW64[3,7]	否	否	WOW64[3,8]	WOW64[3,7]
Windows Vista Ultimate 64-Bit x64 Edition	否	WOW64[3,7]	WOW64[3,7]	WOW64[3,7]	WOW64[3,8]	WOW64[3,7]
Windows Vista Business 64-Bit x64 Edition	否	WOW64[3,7]	WOW64[3,7]	WOW64[3,7]	WOW64[3,8]	WOW64[3,7]
Windows Vista Enterprise 64-Bit x64 Edition	否	WOW64[3,7]	WOW64[3,7]	WOW64[3,7]	WOW64[3,8]	WOW64[3,7]

注：

（1）SQL Server 2005 Evaluation Edition 支持与 SQL Server 2005 Enterprise Edition 相同的功能集，但并非所有支持 Evaluation Edition 的操作系统都支持 SQL Server 2005 Enterprise Edition。

（2）可以从 Microsoft 网站下载 Windows 2000 SP4。必须通过原始设备制造商（OEM）来获取 Windows 2000 Datacenter Edition 的 Service Pack。

（3）这些版本的 SQL Server 2005 可以安装到 64 位服务器的 Windows on Windows （WOW64） 32 位子系统中。

（4）可以在 Windows 2000 Professional SP4 和 Windows XP SP2 上为 SQL Server 2005 Enterprise Edition 安装 Microsoft SQL Server 联机丛书、客户端工具和某些早期工具。客户端工具包括 SQL Server Management Studio 和 SQL Server 2005 软件开发包 Business Intelligence Development Studio。早期工具包括 Data Transformation Services 运行时和 SQL-DMO。

（5）如果操作系统不包含 Internet 信息服务 （IIS），则不会安装作为具有高级服务的 SQL Server Express 一部分的 Reporting Services。

（6）Windows Server 2003 R2 与 Windows Server 2003 SP1 上的 SQL Server 2005 支持相同。

（7）在 Windows Vista 上需要 SQL Server 2005 SP2。

（8）SQL Server 2005 Express Edition 在 Windows Vista 上需要 SP1。具有 Advanced Services 的 SQL Server 2005 Express Edition 在 Windows Vista 上需要 SP2。

2.2.3　SQL Server 2005 安装过程

安装 SQL Server 2005 的步骤如下。

（1）主菜单。只要载入 DVD 驱动器中的光盘，就会出现如图 2.1 所示的安装屏幕。

图 2.1　主菜单

在安装标题下选择“服务器组件、工具、联机丛书和示例”选项。

（2）终端用户许可协议。检查终端用户的许可协议（EULA），如图 2.2 所示。阅读后选择“我接受许可条款和条件”框，单击“下一步”按钮。

（3）安装必备组件，如图 2.3 所示。选择好必备组件后单击“安装”按钮。

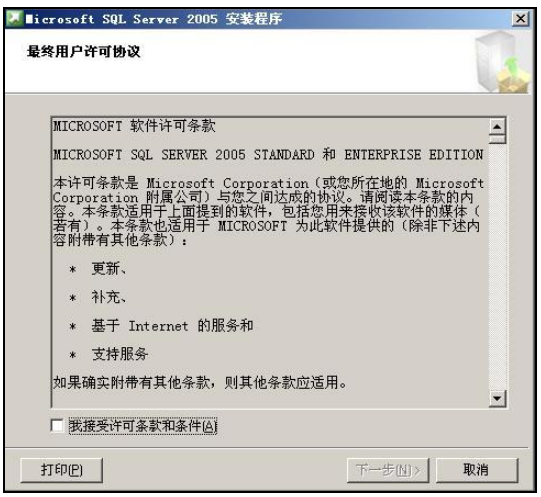


图 2.2　最终用户许可协议

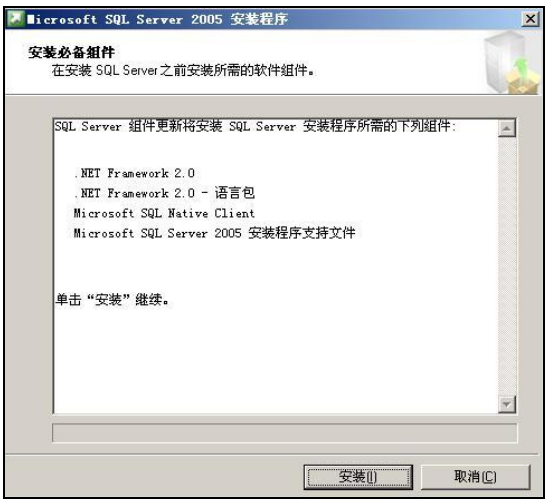


图 2.3　安装先决条件

（4）安装完必备组件后，单击如图 2.4 所示的“下一步”按钮。

（5）进入 SQL Server 2005 安装向导界面，如图 2.5 所示，单击“下一步”按钮。

图 2.4　成功安装必备组件

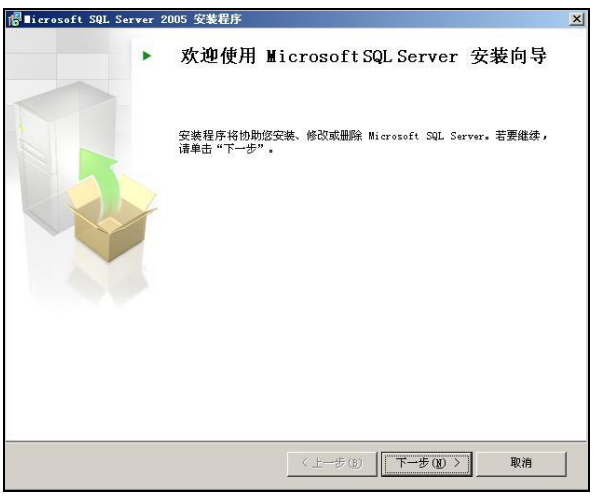


图 2.5　安装向导

（6）进入“系统配置检查”界面，检查用户系统的软硬件是否满足安装 SQL Server 2005 的要求，检查完毕后，将显示检查结果，包括成功、错误和警告信息，如图 2.6 所示。

（7）单击“下一步”按钮，进入“注册信息”界面，如图 2.7 所示。输入姓名和公司，继续 SQL Server 的安装。

图 2.6　系统配置检查结果信息

图 2.7　设置注册信息

（8）单击“下一步”按钮，进入“要安装的组件”界面，如图 2.8 所示。用户可以选择要安装的组件，并设置组件的安装路径。

（9）单击“下一步”，进入“实例名”界面，如图 2.9 所示。选中“命名实例”单选框，在文本输入框中输入“zzuli_soft”。

图 2.8　选择安装组件

图 2.9　设置实例名

注意

实例是虚拟的 SQL Server 2005 服务器，在同一台计算机上可以安装多个 SQL Server 实例，每个实例就像是一个单独的 SQL Server 2005 服务器，不同版本的 SQL Server 服务器可以通过多个实例机制运行在同一台计算机上而互不干扰，但前提是各版本的 SQL Server 按照自己的端口运行。

（10）在图 2.9 中单击“下一步”按钮，出现如图 2.10 所示的“服务账户”界面，选择“使用内置系统账户”单选框，单击“下一步”按钮。

（11）进入如图 2.11 所示的“身份验证”界面，选择“混合模式”，并输入密码，单击“下一步”按钮。

图 2.10　设置服务账户

图 2.11　身份验证

（12）进入如图 2.12 所示的“排序规则设置”界面。排序规则指定了 SQL Server 2005 数据库中字符数据的物理存储模式，以及存储和比较字符时所使用的规则，这里使用默认设置，单击“下一步”按钮。

（13）进入如图 2.13 所示的“报表服务器安装选项”界面，该界面用于指定如何安装报表服务器，使用默认设置，单击“下一步”按钮。

图 2.12　设置排序规则

图 2.13　设置报表服务器

（14）进入如图 2.14 所示的“错误和使用情况报告设置”界面。该界面用于设置是否自动将 SQL Server 2005 的错误信息和使用情况报告给 Microsoft 公司，以帮助该公司改进 SQL Server 2005 的性能和产品质量选择其一，然后单击“下一步”按钮。

（15）出现如图 2.15 所示的“准备安装”界面。用户确认安装信息后，单击“安装”按钮，开始安装 SQL Server 2005。

图 2.14　错误和使用情况报告设置

图 2.15　准备安装 SQL Server 2005

（16）图 2.16 所示的是安装的进度表，在此期间，系统会提示用户插入第 2 张安装光盘的信息，插入后单击“确定”按钮，系统继续安装。经过一段时间后，系统安装完毕，“下一步”按钮会被激活，单击“下一步”按钮。

（17）进入如图 2.17 所示的界面，单击“完成”按钮，完成 SQL Server 2005 的安装。

图 2.16　安装进度

图 2.17　完成 SQL Server 2005 安装

虽然大多数的 SQL Server 安装都使用了默认的参数，这样的话安装就是一个简单的过程，但是如果没有理解安装参数的话，也会导致困惑或者将来安全攻击方面的问题。正因为如此，下面列出了一些关键点，供安装 SQL Server 的时候考虑。

（1）只安装必要的 SQL Server 组件来限制服务的数量。这也同时减少了因为没有实现 SQL Server 必要组件而忘记打关键补丁的可能性。

（2）对于 SQL Server 服务账号，确保要选择一个拥有域内适当权限的账号。不要只是选择域管理员来运行 SQL Server 服务账号。平衡最小权限和只分配所需权限给账号的原则。与

此同时，确保给 SQL Server 服务账号分配了一个复杂的密码。针对以上各项，在进行步骤 1、2 之前确保选择了正确的参数。

（3）在步骤 3 中，选择认证模式。有两个选项，分别是 Windows 认证模式或者混合模式（Windows 和 SQL Server 认证）。在 Windows 认证模式下，只有 Windows 账号才可以拥有登录 SQL Server 的权限，在混合模式下，Windows 和 SQL Server 的账号都可以拥有登录到 SQL Server 的权限。

（4）步骤 3 中的 SQL Server 认证模式会直接影响步骤 4 中的安装过程。当选中混合模式认证的时候，就是分配系统管理员密码的时机。因为是服务账号，确保使用一个强有力的密码或者密码段，并且正确保护密码。

（5）接下来的一个关键点是在 SQL Server 安装过程中的，就是调整设置。在 SQL Server 中，Windows 和 SQL Server 的调整都是可用的。这些调整应该是基于应用程序语言支持需求的。另外在所在环境中检查当前 SQL Server 的配置也是明智的，可以以此确保能够满足特殊应用程序需求，以及与 SQL Server 的一致性。

（6）最后一个关键点就是察看所有安装的输出，以此确保过程是成功的。在把 SQL Server 发布到你的环境之前验证输出。


注意　本地安装，必须以管理员身份运行安装程序。如果通过远程共享安装 SQL Server，则必须使用对远程共享具有读取和执行权限的域账户。

2.3　SQL Server 2005 的配置和管理

2.3.1　Management Studio 概述

Management Studio 是 Microsoft SQL Server 2005 提供的一种新集成环境，用于访问、配置、控制、管理和开发 SQL Server 的所有组件。SQL Server Management Studio 将一组多样化的图形工具与多种功能齐全的脚本编辑器组合在一起，可为各种技术级别的开发人员和管理员提供对 SQL Server 的访问。

SQL Server Management Studio 将早期版本的 SQL Server 中所包含的企业管理器、查询分析器和 Analysis Manager 功能整合到单一的环境中。此外，SQL Server Management Studio 还可以和 SQL Server 的所有组件协同工作，如 Reporting Services、Integration Services、SQL Server Compact Edition 和 Notification Services。使用过早期版本的开发人员可以获得熟悉的体验，而数据库管理员可获得功能齐全的单一实用工具，其中包括易于使用的图形工具和丰富的脚本撰写功能。

1. *启动* SQL Server Management Studio

在任务栏中单击“开始”，依次指向“所有程序”、“Microsoft SQL Server 2005”，再单击“SQL Server Management Studio”，打开如图 2.18 所示的“连接到服务器”对话框。

在如图 2.18 所示的界面中的“服务器类型”中选择默认设置“数据库引擎”，输入登录名和密码后单击“连接”按钮，出现如图 2.19 所示的“Microsoft SQL Server Management Studio”

界面。

图 2.18　“连接到服务器”对话框

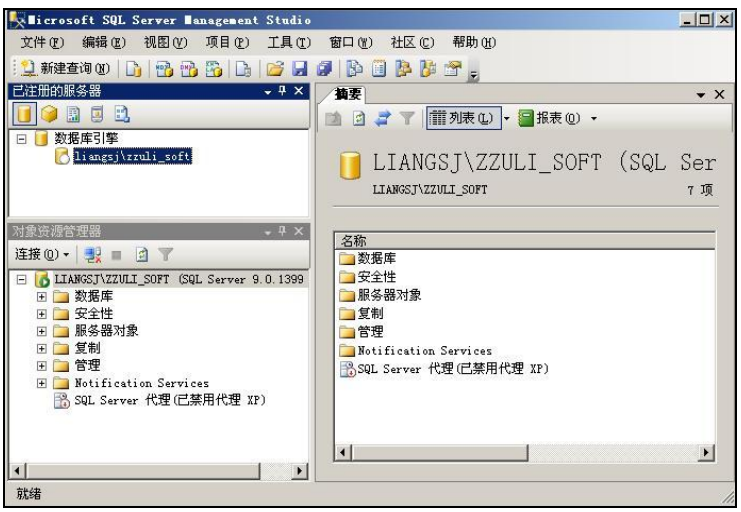


图 2.19　Microsoft SQL Server Management Studio 界面

默认情况下，Management Studio 中将显示 3 个组件窗口。

（1）已注册的服务器。窗口列出的是经常管理的服务器，可以在此列表中添加和删除服务器。如果计算机上以前安装了 SQL Server 2000 企业管理器，则系统将提示导入已注册服务器的列表。否则，列出的服务器中仅包含运行 Management Studio 的计算机上的 SQL Server 实例。如果未显示所需的服务器，请在“已注册的服务器”中右键单击 Microsoft SQL Servers，再单击“更新本地服务器注册”。

（2）对象资源管理器。该窗口是服务器中所有数据库对象的树形视图。此树形视图包括 SQL Server Database Engine、Analysis Services、Reporting Services、Integration Services 和 SQL Server Mobile 的数据库。对象资源管理器包括与其连接的所有服务器的信息。打开 Management Studio 时，系统会提示将对象资源管理器连接到上次使用的设置。可以在“已注册的服务器”组件中双击任意服务器进行连接，但无需注册要连接的服务器。

（3）文档窗口。该窗口是 Management Studio 中的最大窗口。文档窗口可能包含查询编辑器和浏览器窗口。默认情况下，将显示已与当前计算机上的数据库引擎实例连接的“摘要”页。

2. 与已注册的服务器和对象资源管理器连接

（1）连接到服务器。与已注册的服务器类似，对象资源管理器也可以连接到数据库引擎、Analysis Services、Integration Services、Reporting Services 和 SQL Server Mobile。可以注册上述任意服务器类型以便于管理。现在通过一个例子来展示如何注册“AdventureWorks”数据库。

【例 2.1】注册“AdventureWorks”数据库。

AdventureWorks 数据库是个较大的新示例数据库，可演示 Microsoft SQL Server 2005 中比较复杂的功能。AdventureWorksDW 是一个支持 SQL Server Analysis Services 的关系数据库。为了增强安全性，默认情况下不会安装示例数据库。若要安装该示例数据库，请重新运行安装光盘。

注册过程通过以下步骤完成。

① 在“已注册的服务器”工具栏上，右键单击“数据库引擎”，指向“新建”，再单击“服务器注册”。此时将打开“新建服务器注册”对话框，如图 2.20 所示。

图 2.20　新建服务器注册

② 在“服务器名称”文本框中，输入 SQL Server 实例的名称。

③ 在“已注册的服务器名称”框中，输入 AdventureWorks。

④ 在“连接属性”选项卡的“连接到数据库”列表中，选择 AdventureWorks，再单击“保存”按钮，出现如图 2.21 所示的“已注册的服务器”窗口。

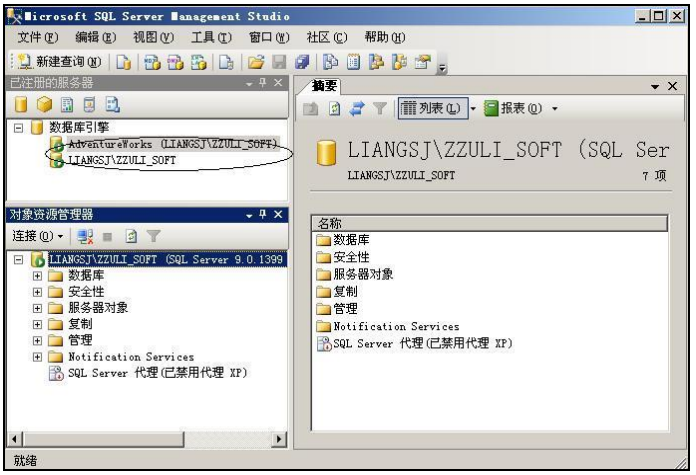


图 2.21　“已注册的服务器”窗口

（2）与对象资源管理器连接。与已注册的服务器类似，对象资源管理器也可以连接到数据库引擎、Analysis Services、Integration Services、Reporting Services 和 SQL Server Mobile。

【例 2.2】在“对象资源管理器”中注册“AdventureWorks”数据库。

注册过程如下。

① 在图 2.21 所示的对象资源管理器的工具栏上，单击“连接”按钮显示可用连接类型下拉列表，再选择“数据库引擎”。系统将打开“连接到服务器”对话框。

② 在“服务器名称”文本框中，输入 SQL Server 实例的名称。

③ 单击“选项”按钮，然后浏览各选项。

④ 单击“连接”按钮，连接到服务器。如果已经连接，则将直接返回到对象资源管理器，并将该服务器设置为焦点。

连接到 SQL Server 的某个实例时，对象资源管理器会显示外观和功能与 SQL Server 2000 企业管理器中的控制台根节点类似的信息。增强功能包括，在浏览数以千计的数据库对象时可具有更大的伸缩性。使用对象资源管理器，可以管理 SQL Server 安全性、SQL Server 代理、复制、数据库邮件以及 Notification Services。但对象资源管理器只能管理 Analysis Services、Reporting Services 和 SSIS 的部分功能。上述每个组件都有其他专用工具。

⑤ 在对象资源管理器中，展开“数据库”文件夹，然后选择“AdventureWorks”。

3. *更改环境布局*

（1）关闭、隐藏和还原组件。

① 在如图 2.21 所示的界面中，单击“已注册的服务器”右上角的关闭按钮，将其隐藏。“已注册的服务器”随即关闭。

② 在“对象资源管理器”中，单击带有“自动隐藏”工具提示的图钉按钮。“对象资源管理器”将被最小化到屏幕的左侧。

③ 在“对象资源管理器”标题栏上移动鼠标，“对象资源管理器”将重新打开。

④ 再次单击图钉按钮，使“对象资源管理器”驻留在打开的位置。

⑤ 在“视图”菜单上，单击“已注册的服务器”，对其进行还原。

（2）移动组件。

① 在如图 2.21 所示的界面中，单击“已注册的服务器”标题栏，并将其拖到文档窗口中央。该组件将取消停靠并保持浮动状态，直到将其放下。

② 将“已注册的服务器”拖到屏幕的其他位置。在屏幕的多个区域，将收到蓝色停靠信息。如果出现箭头，则表示组件放在该位置将使窗口停靠在框架的顶部、底部或一侧。将组件移到箭头处会导致目标位置的基础屏幕变暗。如果出现中心圆，则表示该组件与其他组件共享空间。如果把可用组件放入该中心，则该组件显示为框架内部的选项卡。

4. 显示文档窗口

文档窗口可以配置为显示选项卡式文档或多文档界面（MDI）环境。在选项卡式文档模式中，默认的多个文档将沿着文档窗口的顶部显示为选项卡。

（1）查看文档布局。

① 在图 2.21 所示界面中的主工具栏上，单击“数据库引擎查询”。在“连接到数据库引擎”对话框中，单击“连接”。

② 在已注册的服务器中，右键单击选择的服务器，指向“连接”，再单击“新建查询”。在这种情况下，查询编辑器将使用已注册的服务器的连接信息。

请注意各窗口如何显示为文档窗口的选项卡。

（2）显示 MDI 环境。

① 在图 2.21 所示的界面中的“工具”菜单上，单击“选项”，打开“选项”对话框，如图 2.22 所示。

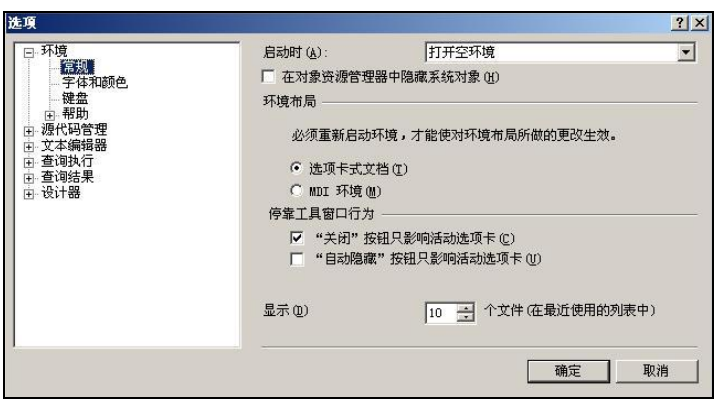


图 2.22 “选项”对话框

② 展开“环境”，再单击“常规”。

③ 在“环境布局”区域中，单击“MDI 环境”，再单击“确定”按钮。

此时，各窗口分别浮动在 Microsoft 文档窗口中。

5. 设置启动选项

SQL Server Management Studio 可配置为，当启动 Management Studio 时打开首选配置的设计界面。配置步骤如下。

① 在如图 2.21 所示的界面中的“工具”菜单上，单击“选项”。

② 展开“环境”，并单击“常规”。在“启动时”列表中，查看以下选项。

- 打开对象资源管理器。这是默认选项。
- 打开新查询窗口。选中此选项以估计 SQL Server 2005 查询分析器的行为。

- 打开对象资源管理器和新查询。
- 打开空环境。

③ 选择“首选”选项，再单击“确定”按钮。

2.3.2　注册服务器

注册服务器就是确定一台工作服务器，为客户端的各种请求提供服务。一般只有远程的数据库服务器，才需要在客户端上注册服务器，然后通过客户端对远程数据库服务器进行管理。

在 SQL Server 以前的版本中，所有的服务器都必须在逻辑上属于某个服务器组，但是，在 SQL Server 2005 中，取消了这种限制，即服务器不必属于某个服务器组。

注册服务器的步骤见例 2.1。

2.3.3　配置服务器

服务与服务器是两个不同的概念，服务器是提供服务的计算机，配置服务器主要是对内存、处理器、安全性等几个方面进行配置。由于 SQL Server 2005 服务器设置的参数比较多，这里介绍一些常用的参数。

配置 SQL Server 2005 服务器的办法：启动“SQL Server Management Studio”，在“对象资源管理器”窗口里，鼠标右键单击要配置的服务器（实例）名，在弹出的快捷菜单里单击“属性”选项，如图 2.23 所示。下面介绍各选项卡里的内容。

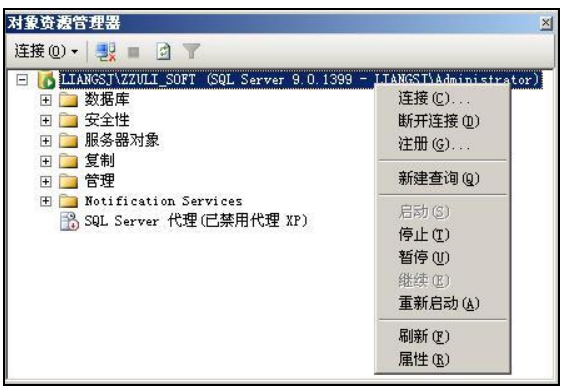


图 2.23　选择要配置的服务器

1. 配置“常规”选项卡

图 2.24 所示的是服务器属性中的“常规”选项卡，此处功能是查看服务器的属性，如服务器名、操作系统、CPU 数等。此处各项只能查看，不能修改。选项卡里有以下项目。

（1）名称：显示服务器（实例）的名称。

（2）产品：显示当前运行的 SQL Server 的版本。

（3）操作系统：显示当前运行的操作系统及版本号。

（4）平台：显示运行 SQL Server 的操作系统和硬件。

（5）版本：显示当前运行的 SQL Server 版本号。

（6）语言：显示当前的 SQL Server 实例所使用的语言。

（7）内存：显示当前服务器上的内存大小。

（8）处理器：显示当前服务器上的 CPU 数量。

（9）根目录：显示当前 SQL Server 实例所在的目录。

（10）服务器排序规则：显示当前服务器采用的排序规则。

（11）已群集化：显示是否安装了 SQL Server 2005 服务器群集。

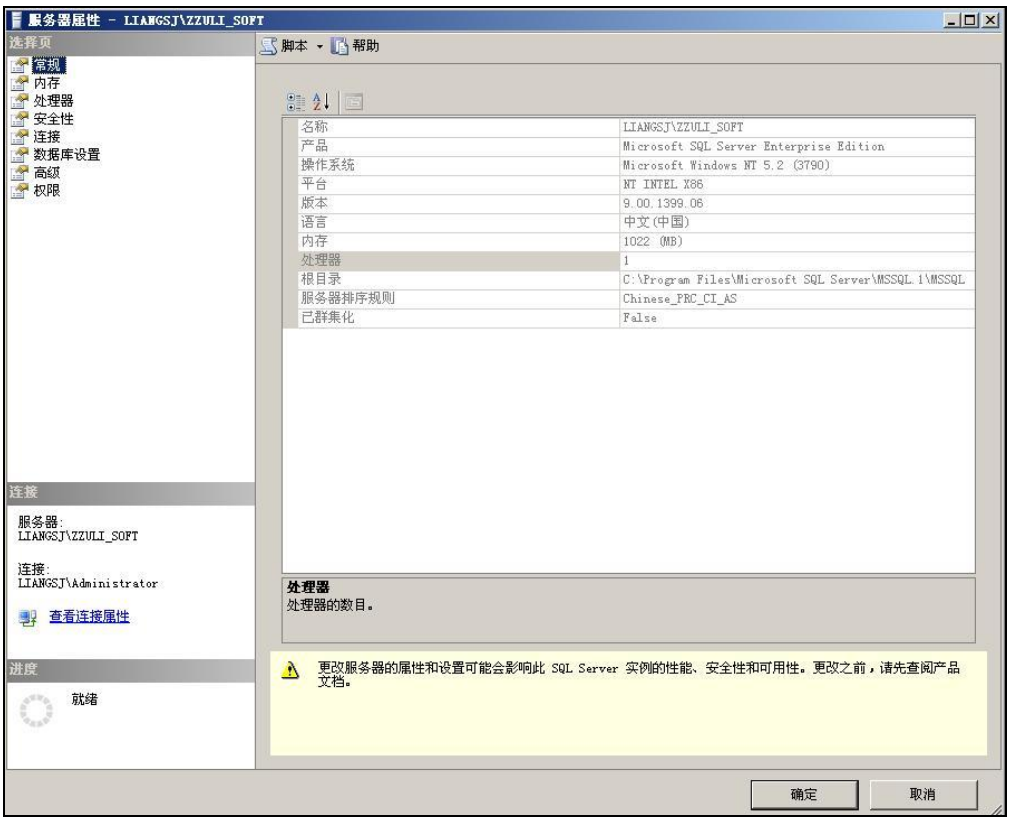


图 2.24 “常规”选项卡

2. 配置“内存”选项卡

图 2.25 所示的是服务器属性中的“内存”选项卡，选项卡里有以下项目。

（1）使用 AWE 分配内存。32 位的操作系统最多只能支持到 4GB 的内存，而大型的 SQL Server 2005 服务器的物理内存可以扩展到 64GB。如果要使用大于 4GB 的内存，就要用到 Windows 2000 和 Windows Server 2003 地址扩展插件（AWE）API 来识别和分配了。这个选项是指定 SQL Server 利用 AWE 来支持超过 4GB 以上的物理内存。如果数据库服务器的内存没有超过 4GB，就不用选择此项。

（2）最小服务器内存。该项指定分配给 SQL Server 的最小内存，低于这个值的内存是不会被释放的。要根据当前实例的大小和活动设置此值，以确保操作系统不会从 SQL Server 请求过多的内存，以免影响 SQL Server 的性能。

（3）最大服务器内存。该项指定分配给 SQL Server 的最大内存。除非知道有多少个应用程序与 SQL Server 同时运行，并且知道这些应用程序要使用多少内存，那么就可以将此项设为特定值，否则的话，就不必设置此项，让应用程序按需请求内存。

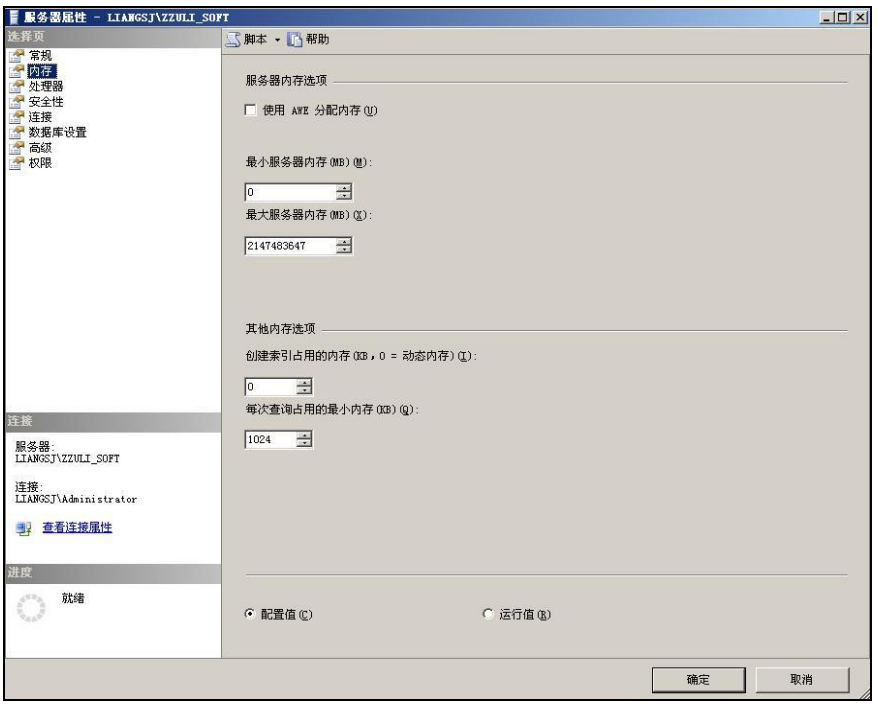


图 2.25　“内存”选项卡

（4）创建索引占用的内存。该项指定在索引创建排序过程中要使用的内存量。其值为 0 时表示由操作系统动态分配。一般情况下，此项都不需要设置，不过也可以输入 704 至 2 147 483 647 之间的值。

（5）每次查询占用的最小内存。该项指定为执行查询分配的内存量，默认为 1 024KB。如果经常执行的 SQL 查询语句涉及到排序，或查询的数据量很大的情况，可以将此值设得较大。此值的范围为 512KB 至 2 147 483 647KB 之间。

（6）配置值和运行值。配置值显示本对话框上选项的配置值，运行值查看本对话框上选项当前运行的值。在配置值修改过后，可以单击运行值来查看更改是否已经生效，如果没有生效的话，就要重新启动 SQL Server 实例了。

3. 配置“处理器”选项卡

图 2.26 所示的是服务器属性中的“处理器”选项卡，在此页中可以查看或修改 CPU 选项，一般来说，只有安装了多个处理器才需要配置此项。选项卡里有以下项目。

（1）处理器关联。为了执行多任务，Windows 2000 和 Windows Server 2003 有时会在不同的 CPU 之间移动进程和线程，对于操作系统而言，这种活动是高效的，但是对于高负荷的 SQL Server 而言，该活动会降低性能，因为每个处理器都会不断地重新加载数据。这种线程与处理器之间的关联就是“处理器关联”。如果将处理器分配给特定线程，那么就会消除处理器的重新加载需要，以及减少处理器之间的线程迁移。

（2）I/O 关联。与处理器关联类似。此项设置是否将 SQL Server 磁盘 I/O 绑定到指定的 CPU 子集。

（3）自动设置所有处理器的处理器关联掩码。此项设置是否允许 SQL Server 设置处理器关联。如果启用的话，操作系统将自动为 SQL Server 2005 分配 CPU。

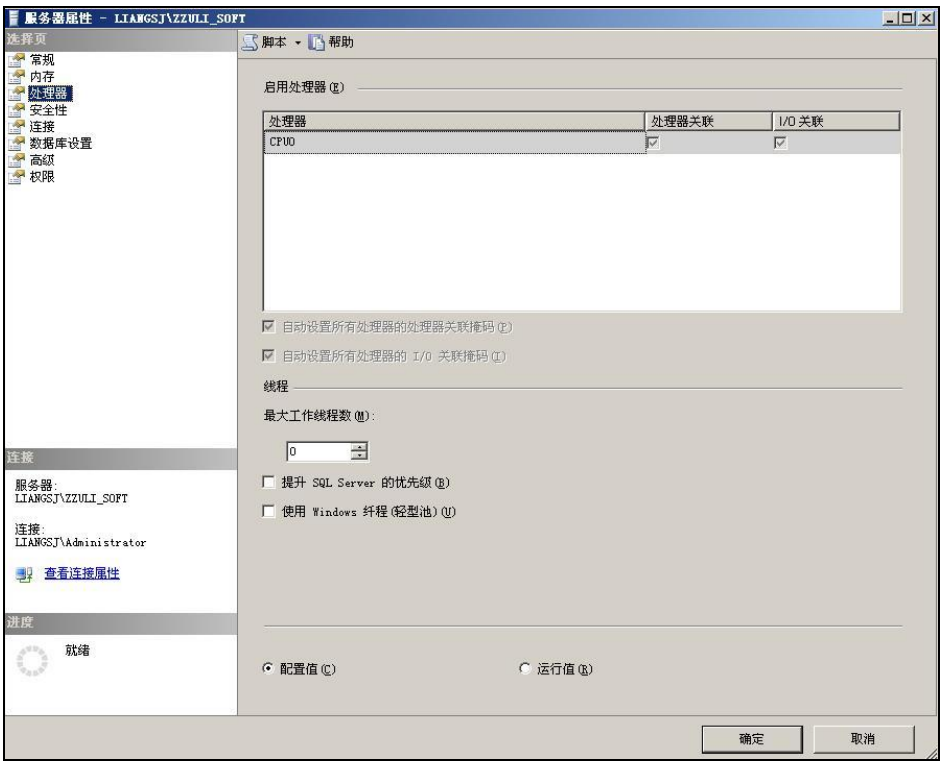


图 2.26 “处理器”选项卡

（4）自动设置所有处理器的 I/O 关联掩码。此项设置是否允许 SQL Server 设置 I/O 关联。如果启用的话，操作系统将自动为 SQL Server 2005 分配磁盘控制器。

（5）最大工作线程数。也就是允许 SQL Server 动态设置工作线程数，默认设置为 0。一般来说，此值不用修改。

（6）提升 SQL Server 的优先级。指定 SQL Server 是否应比其他进程具有优先处理的级别。如果服务器上主要运行的服务是 SQL Server 的话，可以选用此项。

（7）使用 Windows 纤程。使用 Windows 纤程代替 SQL Server 服务的线程。此选项仅适用于 Windows 2003 Server Edition。

4. 配置“安全性”选项卡

图 2.27 所示的是服务器属性中的“安全性”选项卡，可以用来查看或修改服务器的安全选项。选项卡里有以下项目。

（1）服务器身份验证。用于更改 SQL Server 2005 服务器的身份验证方式，与安装 SQL Server 2005 时的选项相同，有“Windows 身份验证模式”和“混合模式”（SQL Server 和 Windows 身份验证模式）两种。

注意

① 更改安全性配置之后需要重新启动服务。

② 如果是从“Windows 身份验证模式”改到“混合模式”的话，不会自动启用 sa 账户。如果要使用 sa 账户，要执行带有 enable 选项的 Alter Login 命令。

（2）登录审核。此项设置是否对用户登录 SQL Server 2005 服务器的情况进行审核。

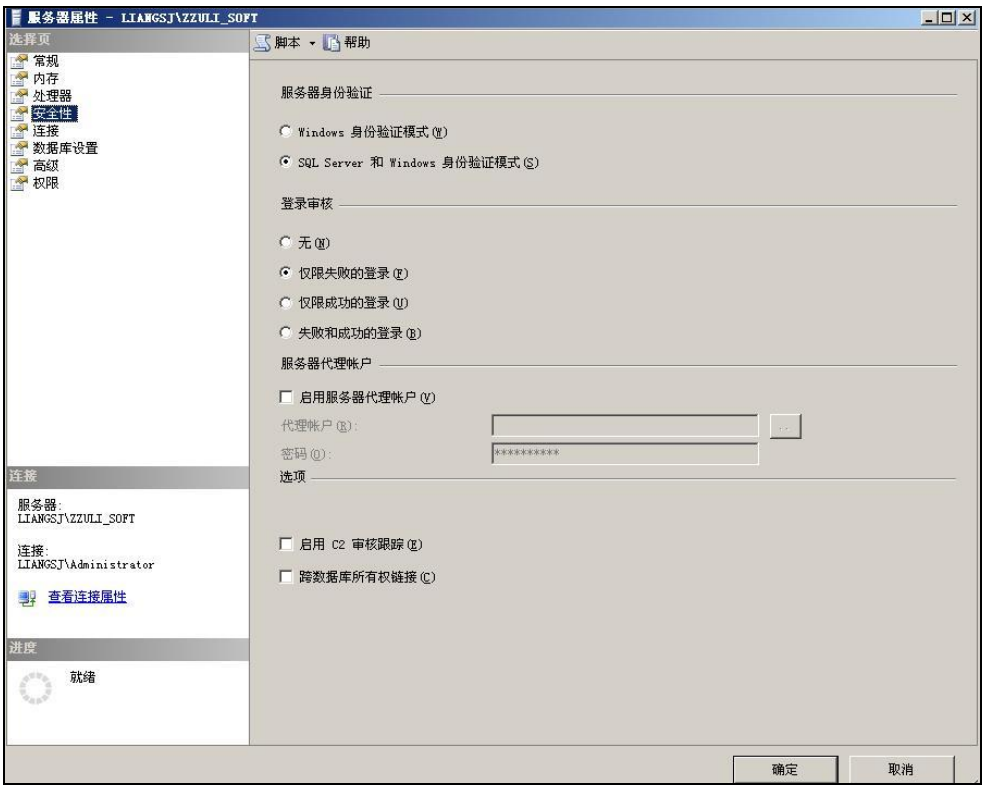


图 2.27　“安全性”选项卡

（3）服务器代理账户。指定是否启用供“xp_cmdshell”使用的账户。“xp_cmdshell”是一个 Transact-SQL 存储过程，可以生成 Windows 命令，并以字符串的形式传递和执行。在执行操作系统命令时，代理账户可以模拟登录、服务器角色和数据库角色。

（4）启用 C2 审核跟踪。C2 是一个政府安全等级，它保证系统能够保护资源并具有足够的审核能力。C2 模式允许监视对所有数据库实体的访问企图。C2 审核模式数据保存在默认实例或命名实例的“Data”目录中的某个文件内。如果审核日志文件超过了 200 MB 的大小限制，SQL Server 将创建一个新文件，关闭旧文件并将所有新的审核记录写入新文件。此过程将继续下去，直到审核数据目录已满或审核被关闭。C2 审核模式将大量事件信息保存在日志文件中，这样可能会导致日志文件迅速增大。如果保存日志数据的空间不足，SQL Server 将自行关闭。

（5）跨数据库所有权链接。选中此项将允许数据库成为跨数据库所有权链接的源或目标。

5. 配置“连接”选项卡

图 2.28 所示的是服务器属性中的“连接”选项卡，在此可以设置与连接服务器相关的属性和参数。选项卡里有以下项目。

（1）最大并发连接数。默认值为 0，表示无限制。也可以输入数字，限制 SQL Server 2005 允许的连接数。如果将此值设置过小，可能会阻止管理员进行连接，但是“专用管理员连接”始终可以连接。

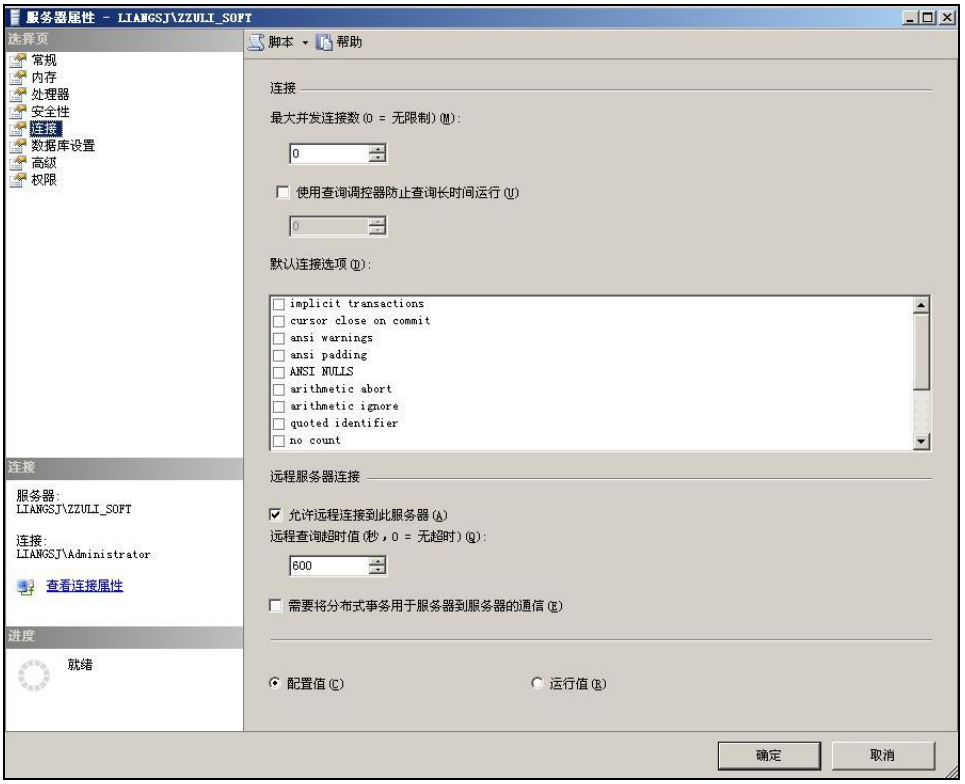


图 2.28 “连接”选项卡

（2）使用查询调控器防止查询长时间运行。为了避免使用 SQL 查询语句执行时间过长，导致 SQL Server 服务器的资源被长时间占用，可以设置此项。选择此项后输入最长的查询运行时间，超过这个时间后，会自动终止查询，以释放更多的资源。

（3）默认连接选项。默认连接的选项内容比较多，如表 2.10 所示。

（4）允许远程连接到此服务器。选中此项则允许运行 SQL Server 实例的远程服务器执行控制存储过程。远程查询超时值指定在 SQL Server 超时之前远程操作可执行的时间，默认为 600s。

（5）需要将分布式事务用于服务器到服务器的通信。选中此项则允许通过 Microsoft 分布式事务处理协调器（MS DTC）事务保护服务器到服务器的操作。

表 2.10　　默认连接选项及说明

配 置 选 项	说　　明
disable deferred constraint checking	控制执行期间或延迟的约束检查
implicit transactions	在运行一条语句时，是否隐式启动一项事务
cursor close on commit	控制执行提交操作后游标的行为
ansi warnings	控制集合警告中的截断和 NULL
ansi padding	控制固定长度的变量的填充
ansi nulls	在使用相等运算符时控制 NULL 的处理
arithmetic abort	在查询执行过程中发生溢出或被零除错误时终止查询
arithmetic ignore	在查询过程中发生溢出或被零除错误时返回 NULL

续表

配 置 选 项	说　　明
quoted identifier	计算表达式时区分单引号和双引号
no count	关闭在每个语句执行后所返回的说明，该说明表示有多少行受影响的消息
ansi null default on	更改会话的行为，使用 ANSI 兼容为空性。未显式定义为空性的新列定义为允许使用空值
ansi null default on	更改会话的行为，不使用 ANSI 兼容为空性。未显式定义为空性的新列定义为不允许使用空值
concat null yields null	当将 NULL 值与字符串连接时返回 NULL
numeric round abort	当表达式中出现失去精度的情况时生成错误
xact abort	如果 Transact-SQL 语句引发运行时错误，则回滚事务

6. 配置“数据库设置”选项卡

图 2.29 所示的是服务器属性中的“数据库设置”选项卡，选项卡里有以下项目。

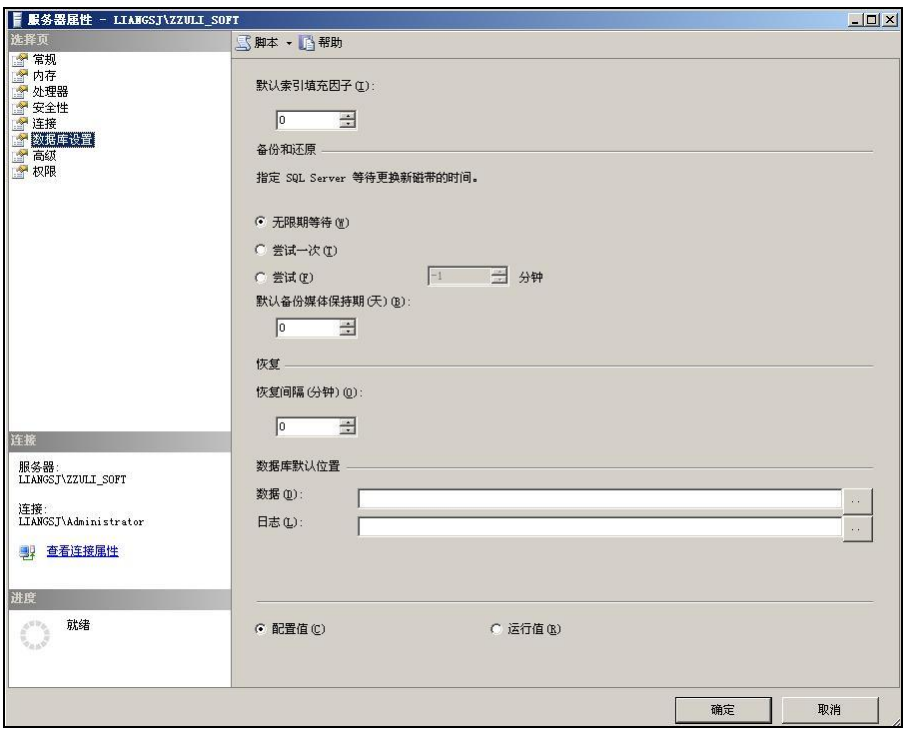


图 2.29　“数据库设置”选项卡

（1）默认索引填因子。该项的作用是指定在 SQL Server 使用数据创建新索引时对每一页的填充程度。SQL Server 2005 会为索引分配 8KB 大小的数据分页。索引的填充因子就是规定向索引页中插入索引数据最多的情况下可以占用的页面空间。例如填充因子为 60%，那么在向索引页面中插入索引数据时最多可以占用页面空间的 60%，剩下的 40%的空间留给索引的数据更新时用。当表中产生索引的数据更新时，SQL Server 2005 就会自动维护和更新索引页。由于在页填充时 SQL Server 必须花时间来拆分页，因此填充因子会影响性能。本项的默认值是 0，有效值是 0～100。

（2）备份和还原。此项主要是指定 SQL Server 2005 等待更换新磁带的时间。“无限期”

指 SQL Server 在等待新备份磁带时永不超时；“尝试一次”是指如果需要备份磁带，但它却不可用，则 SQL Server 将超时。“尝试”的分钟数是指如果备份磁带在指定的时间内不可用，SQL Server 将超时。“默认备份媒体保持期（天）”提供一个系统范围默认值，指示在用于数据库备份或事务日志备份后每一个备份媒体的保留时间。此选项可以防止在指定的日期前覆盖备份。

（3）恢复。此项用于设置每个数据库恢复时所需的最大分钟数。如果为 0 的话，是让 SQL Server 自动配置。

（4）数据库默认位置。用于指定数据文件和日志文件的默认位置。

7. 配置“高级”选项卡

图 2.30 所示为服务器属性中的“高级”选项卡，在该选项卡中可以设置服务器的并行操作行为、网络行为等。选项卡里有以下项目。

图 2.30 “高级”选项卡

（1）并行的开销阈值。本项指定一个数字，如果一个 SQL 查询语句的开销超过这个数字的话，那么就会启用多个 CPU 来并行执行高于这个数字的查询，以优化性能。开销指的是在特定硬件配置中运行串行计划估计需要花费的时间，单位为 s。

（2）查询等待值。该项指定在超时之前查询等待资源的秒数，有效值是 0～2 147 483 647。默认值是−1，含义是按估计查询开销的 25 倍计算超时值。

（3）锁。该项也指定一个数字，用于设置可用锁的最大数目，以限制 SQL Server 为锁分配的内存量。默认值为 0，也就是允许 SQL Server 根据系统要求来动态分配和释放锁。推荐

使用 SQL Server 动态地使用锁，也就是设为 0。

（4）最大并行度。该限用于设置执行并行计划时能使用的 CPU 数量，最大值为 64。如果设为 0 的话，则使用所有可用的处理器；如果设为 1 的话，则不生成并行计划。默认值为 0。

（5）网络数据包大小。设置整个网络使用的数据包的大小，单位为 Byte。默认值是 4 096Byte。如果应用程序经常执行大容量复制操作或者是发送、接收大量的 text 和 image 数据的话，可以将该值设得较大。如果应用程序接收和发送的信息量都很小，那么可以将其设为 512Byte。

（6）远程登录超时值。该项用于指定从远程登录尝试失败返回之前等待的时间。默认值为 20s，如果设为 0 的话，则允许无限期等待。此项设置影响为执行异类查询所创建的与 OLE DB 访问接口的连接。

（7）两位数年份截止。该项指定从 1 753～9 999 的一个整数，该整数表示将两位数年份解释为四位数的截止年份。

（8）默认全文语言。该项用于指定全文索引列的默认语言。全文索引数据的语言分析取决于数据的语言。默认值为服务器的语言。

（9）默认语言。该项用于指定默认情况下所有新创建的登录名使用的语言。

（10）启动时扫描存储过程。该项用于指定 SQL Server 在启动时是否扫描并自动执行存储过程。如果设为 True，则 SQL Server 在启动时将扫描并自动运行服务器上定义的所有存储过程。

（11）游标阈值。该项用于指定游标集中的行数，如果起过此行数，将异步生成游标键集。当游标为结果集生成键集时，查询优化器会估算为该结果集返回的行数。如果查询优化器估算出的返回行数大于此阈值，则将异步生成游标，使用户能够在继续填充游标的同时从该游标中提取行。否则，同步生成游标，查询将一直等待，直到返回所有行。如果设置为−1，则将同步生成所有键集，此设置适用于较小的游标集。如果设置为 0，则将异步生成所有游标键集。如果设置为其他值，则查询优化器将比较游标集中的预期行数，并在该行数超过所设置的数量时异步生成键集。

（12）允许触发器激发其他触发器。该项用于指定触发器是否可以执行启动另一个触发器的操作，也就是指定触发器是否允许递归或嵌套。

（13）最大文本复制大小。该项指定用一个 INSERT、UPDATE、WRITETEXT 或 UPDATETEXT 语句向复制列添加的 text 和 image 数据的最大值，单位为 Byte。

8. 设置“权限”选项卡

图 2.31 所示为服务器属性中的“权限”选项卡，该选项卡用于授予或撤销账户对服务器的操作权限。

在“登录名或角色”的列表框里显示的是多个可以设置权限的对象。单击“添加”按钮，可以添加更多的“登录名”和“服务器角色”到这个列表框里。单击“删除”按钮也可以将列表框中已有的登录名或角色删除。

在“显式权限”的列表框里，可以看到“登录名或角色”列表框里的对象的权限。在“登录名或角色”列表框里选择不同的对象，在“显式权限”的列表框里会有不同的权限显示。在这里也可以为“登录名或角色”列表框里的对象设置权限。

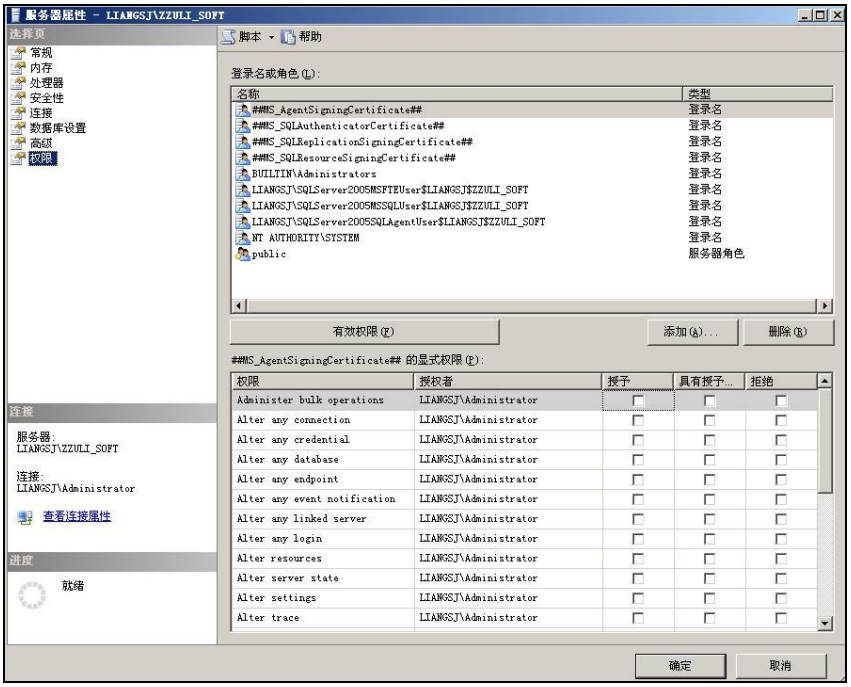


图 2.31 “权限”选项卡

2.3.4 管理服务器

管理 SQL Server 2005 服务器包括服务器的启用、关闭、暂停等。可以通过控制面板、SQL Server Management Studio 和 SQL Server 配置管理器来实现 SQL Server 2005 各种服务器的管理，下面分别进行介绍。

1. *启动服务器*

执行“开始”→“所有程序”→“Microsoft SQL Server 2005”→“配置工具”→“SQL Server Configuration Manager”命令，打开 SQL Server 配置管理器，如图 2.32 所示。

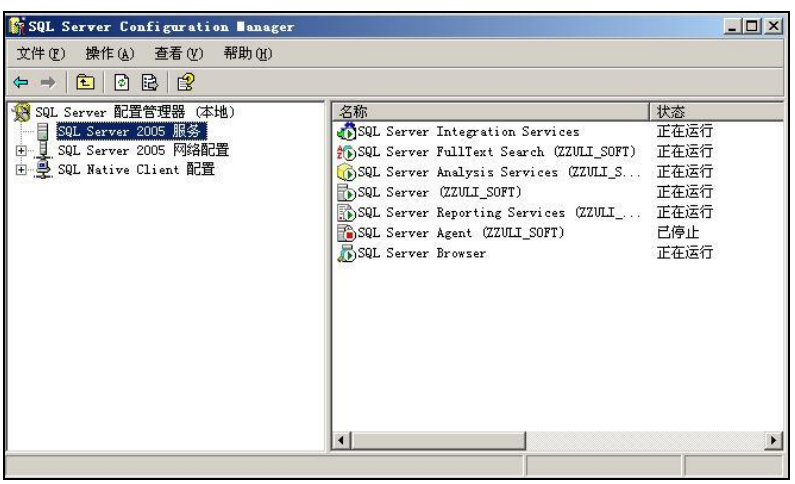


图 2.32 SQL Server 配置管理器

在如图 2.32 所示界面的左侧目录树中选择“SQL Server 2005 服务”，右侧区域将显示服

务器上所有的服务，右键单击任何一个服务，可以对它进行启动、停止、暂停和恢复等操作。

2. 暂停服务器

暂停服务器往往是在需要临时关闭服务器时进行的。服务器暂停后，已经连接的用户其操作将继续运行；新的用户将拒绝连接，除非暂停结束。

在 SQL Server 配置管理器中暂停某个服务器的方法与启动服务器相似，只需右键单击该服务器，在弹出菜单中选择“暂停”命令即可。下面介绍如何在 Management Studio 中暂停某服务器。

启动 Management Studio，在“已注册的服务器”窗口中选择某个服务器，如“LIANGSJ\ZZULI_SOFT”，右键单击，在弹出的快捷菜单中选择“暂停”命令，如图 2.33 所示。系统将弹出是否确认暂停该服务器的提示框，单击“是”按钮，将暂停该服务器。如图 2.34 所示。

图 2.33　暂停服务器

图 2.34　“是否暂停”提示框

3. 关闭服务器

关闭服务器不同于暂停服务器，关闭服务器是从内存中清除所有的 SQL Server 2005 的服务器进程，所有已连接或即将连接用户的操作将全部被禁止。而暂停服务器则是仅仅暂停对数据库的登录请求和对数据的操作。下面介绍在控制面板中关闭服务器的方法。

打开“控制面板”中的“管理工具”，在“管理工具”中打开“服务”组件，找到需要关闭的服务，右键单击该服务，选择“停止”命令即可，如图 2.35 所示。

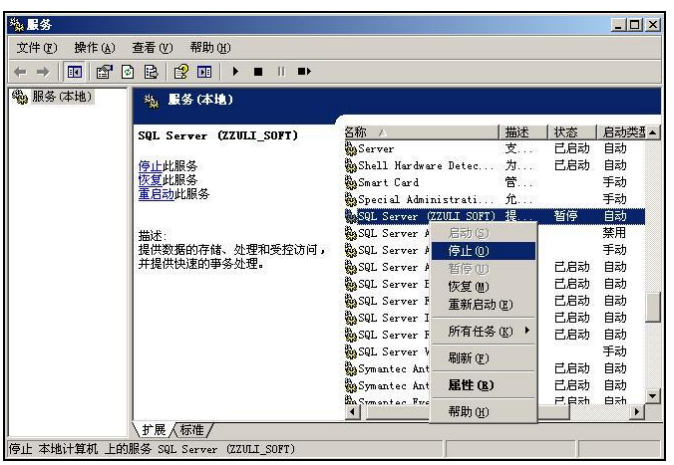


图 2.35　关闭服务器

本 章 小 结

本章详细介绍了 SQL Server 2005 的基本概况，以及新增的一些功能。SQL Server 2005 的版本比较多，本章详尽地介绍了 SQL Server 2005 的不同版本对软硬件的支持，同时详细说明了 SQL Server 2005 的安装过程。

SQL Server 2005 不同于一般的软件，有一些重要的设置对用户来说非常重要，本章详细介绍了如何注册、配置和管理 SQL Server 2005 服务器，这是使用 SQL Server 2005 的基础。

习　　题

一、选择题

1．对于大型企业，宜采用的 SQL Server 2005 版本是（　　）。

A．开发版　　B．工作组版　　C．企业版　　D．学习版

2．要配置“身份验证模式”，应在“服务器属性”窗口中的（　　）选项页中进行设置。

A．常规　　B．内存　　C．安全性　　D．高级

二、填空题

1．Microsoft SQL Server 2005_____是用于存储、处理和保护数据的核心服务。

2．SQL Server 2005 提供了两种身份验证模式，分别是 Windows 身份验证模式和_____。

三、简答题

1．SQL Server 2005 有哪些版本？有哪些主要部件？

2．SQL Server 2005 有哪些新增功能？

3．关闭和暂停 SQL Server 2005 服务器有何区别？

本 章 实 训

一、实训目的

1．熟悉 SQL Server 2005 的安装过程。

2．掌握用 Management Studio 配置和管理 SQL Server 2005。

二、实训要求

1．实训前做好上机实训的准备，针对实训内容，认真复习与本次实训有关的知识，完成实训内容的预习准备工作。

2．独立完成实训内容。

3．实训后做好实训总结，根据实训情况完成总结报告。

三、实训学时

2 学时。

四、实训内容

1．熟悉 SQL Server 2005 的安装过程。

2．注册本地和远程数据库服务器。

3．对已经注册的本地/远程数据库服务器进行配置，要求进行“常规”、“安全性”、“数据库设置”、“权限”等标签的设置。比较不同的设置对数据库服务器运行的影响。

4．管理本地/远程数据库服务器，包括“启动”、“暂停”、“停止”等选项。

五、实训思考题

1．SQL Server 2005 不同版本的安装要求有何不同？

2．在配置数据库服务器时，“Windows 身份验证”和“混合模式”有何不同？

第 3 章

创建和管理 SQL Server 2005 数据库

管理数据库及其对象是数据库的主要任务，本章将介绍使用 SQL Server 2005 创建和管理数据库的基本知识。

Management Studio 用于在脚本项目的环境中管理和设计 SQL Server。Management Studio 包括设计器、编辑器、指南和向导，可帮助用户开发、部署和维护数据库。

3.1 系统数据库概述

SQL Server 2005 包含 5 个系统数据库，如表 3.1 所示。

表 3.1　　系统数据库说明

系统数据库	说　　明
Master 数据库	记录 SQL Server 实例的所有系统级信息
Msdb 数据库	用于 SQL Server 代理计划警报和作业
Model 数据库	用作 SQL Server 实例上创建的所有数据库的模板。对 model 数据库进行的修改（如数据库大小、排序规则、恢复模式和其他数据库选项）将应用于以后创建的所有数据库
Resource 数据库	一个只读数据库，包含 SQL Server 2005 的系统对象。系统对象在物理上保留在 Resource 数据库中，但在逻辑上显示在每个数据库的 sys 架构中
Tempdb 数据库	一个工作空间，用于保存临时对象或中间结果集

SQL Server 不支持用户直接更新系统对象（如系统表、系统存储过程和目录视图）中的信息。而 SQL Server 提供了一整套管理工具，使用户可以充分管理他们的系统和数据库中的所有用户和对象，包括 Transact-SQL 脚本和存储过程。

因为 Resource 数据库是隐藏的，所以从 Management Studio 2005 中的资源管理器只能看到 4 个系统数据库。如图 3.1 所示。

图 3.1　系统数据库

3.1.1　Master 数据库

Master 数据库是“数据库的数据库”，Master 数据库记录 SQL Server 系统的所有系统级信息。这些系统信息包括：

- 实例范围的元数据（例如登录账户）、端点、链接服务器和系统配置设置；
- 其他数据库是否存在以及这些数据库文件的位置；
- SQL Server 的初始化信息。

注意　在 SQL Server 2005 中，系统对象不再存储在 Master 数据库中，而是存储在 Resource 数据库中。

3.1.2　Msdb 数据库

Management Studio 和 SQL Server Agent 使用 Msdb 数据库来存储计划信息以及与备份和恢复相关的信息，尤其是 SQL Server Agent 需要使用它来执行安排工作和警报，记录操作者等操作。

3.1.3　Model 数据库

Model 数据库用在 SQL Server 实例上创建的所有数据库的模板。因为每次启动 SQL Server 时都会创建 Tempdb，所以 Model 数据库必须始终存在于 SQL Server 系统中。

当执行 CREATE DATABASE 语句时，将通过复制 Model 数据库中的内容来创建数据库的第一部分，然后用空页填充新数据库的剩余部分。

如果修改 Model 数据库，之后创建的所有数据库都将继承这些修改。

3.1.4　Resource 数据库

Resource 数据库是只读数据库，它包含了 SQL Server 2005 中的所有系统对象。SQL Server 系统对象（例如 sys.objects）在物理上持续存在于 Resource 数据库中，但在逻辑上，它们出现在每个数据库的 sys 架构中。

Resource 数据库是隐藏的，通常应该由 Microsoft 客户服务专家来打开，用于查找问题和进行客户支持。

3.1.5　Tempdb 数据库

Tempdb 系统数据库是连接到 SQL Server 实例的所有用户都可用的全局资源，它保存所有临时表和临时存储过程。另外，它还用来满足所有其他临时存储要求，例如存储 SQL Server 生成的工作表。

每次启动 SQL Server 时，都要重新创建 Tempdb，以便系统启动时，该数据库总是空的。在断开连接时 SQL Server 会自动删除临时表和存储过程，并且在系统关闭后没有活动连接。因此 Tempdb 无法从一个 SQL Server 会话保存内容使其用于另一个会话。

3.2　创建数据库

创建新的用户数据库时，Model 数据库中的所有用户定义对象都将复制到每个新创建的

数据库中。可以向 Model 数据库中添加任何对象（如表、视图、存储过程、数据类型等），使得所有新建数据库中都包括这些对象。

3.2.1 数据库文件

每个 SQL Server 2005 数据库至少具有两个操作系统文件：一个数据文件和一个日志文件。数据文件包含数据和对象，如表、索引、存储过程和视图。日志文件包含恢复数据库中的所有事务所需的信息。为了便于分配和管理，可以将数据文件集合起来，放到文件组中。

默认情况下，在单磁盘中，数据和事务日志被放在同一文件目录下，大多数数据库在只有单个数据文件和单个事务日志文件的情况下性能良好。为安全考虑，在生产环境中，建议将数据文件和日志文件放在不同的磁盘上。

1. 数据文件

数据文件是存放数据库数据和数据库对象的文件。一个数据库可以有一个或多个数据文件。

当有多个数据文件时，有一个文件被定义为主数据文件（Primary Database File），扩展名为 mdf。它用来存储数据库的启动信息和部分或全部数据，一个数据库只能有一个主数据文件。其他数据文件被称为次数据文件（Secondary Database File），扩展名为.ndf，次要文件可用于将数据分散到多个磁盘上。如果数据库超过了单个 Windows 文件的最大长度，可以使用次要数据文件，这样数据库就能继续增长。

2. 日志文件

事务日志文件保存用于恢复数据库的日志信息。每个数据库必须至少有一个日志文件，也可以为多个。事务日志的建议文件扩展名是 .ldf。

3. 物理文件

每个数据库文件有两个名称。

（1）逻辑文件名（logical_file_name）

在所有 Transact-SQL 语句中引用物理文件时所使用的名称，在数据库中的逻辑文件名必须是唯一的。

（2）物理文件名（os_file_name）

包含目录路径的物理文件名。

4. 文件大小

必须指定数据文件和日志文件的初始大小，或采用默认大小。随着数据不断地添加到数据库，这些文件将逐渐变大。

在创建数据库时，应根据数据库中预期的最大数据量，创建尽可能大的数据文件。允许数据文件自动增长，但要有一定的限度。为此，需要指定数据文件增长的最大值，以便在硬盘上留出一些可用空间。这样便可以使数据库在添加超过预期的数据时继续增长，而不会填满磁盘驱动器。如果已经超过了初始数据文件的大小并且文件开始自动增长，则重新计算预期的数据库容量的最大值。然后，根据计划添加更多的磁盘空间。如果需要，在数据库中创建并添加更多的文件或文件组。

5. 文件组

为便于分配和管理，可以将数据文件分成文件组（日志文件不包括在文件组内）。有两种类型的文件组。

主文件组（Primary）：主文件组包含主数据文件和任何没有明确分配给其他文件组的其他文件。系统表的所有页均分配在主文件组中。

用户定义文件组：用户定义文件组是通过在 CREATE DATABASE 或 ALTER DATABASE 语句中使用 FILEGROUP 关键字指定的任何文件组。

【例 3.1】在 Master 数据库下，执行存储过程 sys.sp_helpfile，则得到结果如图 3.2 所示，逻辑文件名在“name”列，物理文件名在“filename”列，最大容量在 maxsize 列，增长容量在 growth 列，文件组在“filegroup”列。

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	master	1	D:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\master.mdf	PRIMARY	4096 KB	Unlimited	10%	data only
2	mastlog	2	D:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\mastlog.ldf	NULL	1280 KB	Unlimited	10%	log only

图 3.2　数据库文件

3.2.2　使用 Management Studio 创建数据库

在 Management Studio 下创建数据库的步骤介绍如下。

1. *新建数据库*

在对象资源管理器中的数据库节点上右键单击，如图 3.3 所示，在弹出菜单中单击“新建数据库…”，则出现“新建数据库”对话框，如图 3.4 所示。

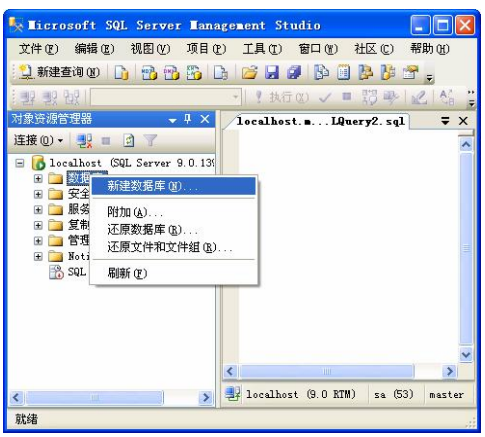


图 3.3　新建数据库弹出菜单

2. *填写“新建数据库”对话框*

在如图 3.4 所示的界面中填写如下内容。

（1）数据库名称。

（2）数据库所有者：创建数据库的用户。

（3）数据库文件。包括数据文件（一个或多个）或日志文件（一个或多个），如果需要设置多个数据文件或日志文件，单击“添加”或“删除”按钮，对文件设置如下选项。

① 逻辑文件名称。

② 文件组：每个数据库至少有两个文件（一个主文件和一个事务日志文件）和一个文件组。应尽可能的在不同的本地物理磁盘上创建文件或文件组，此外，还要将争夺空间最激烈的对象放置在不同的文件组中。

③ 初始大小：根据数据库中表所占数据估算大小设置。

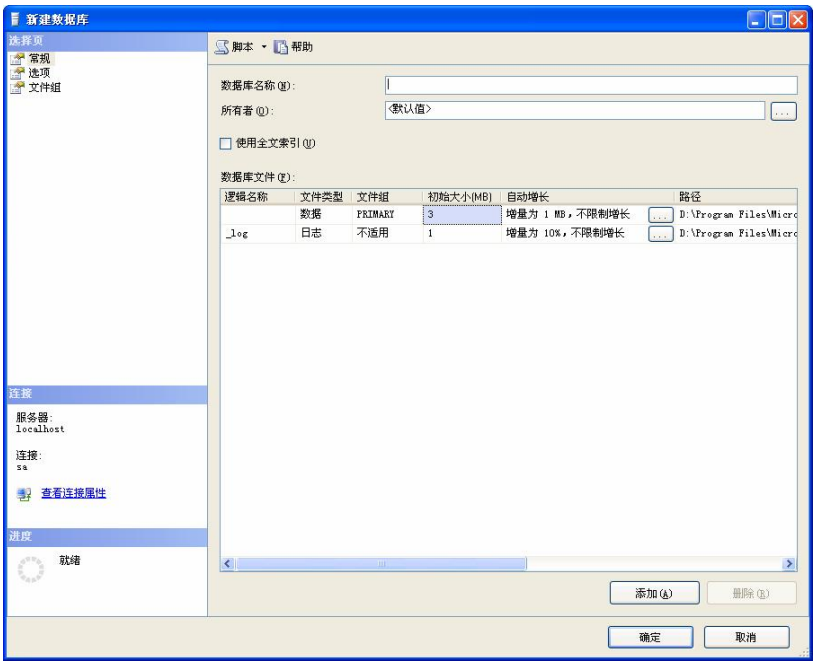


图 3.4　新建数据库对话框

④ 自动增长大小：根据数据库增长的需要来设置。

⑤ 物理路径：设置数据库文件的磁盘位置。

3．设置“选项”页

单击“选择页”中的“选项”，可以打开“选项”页，设置创建数据库的各个选项，如图 3.5 所示。

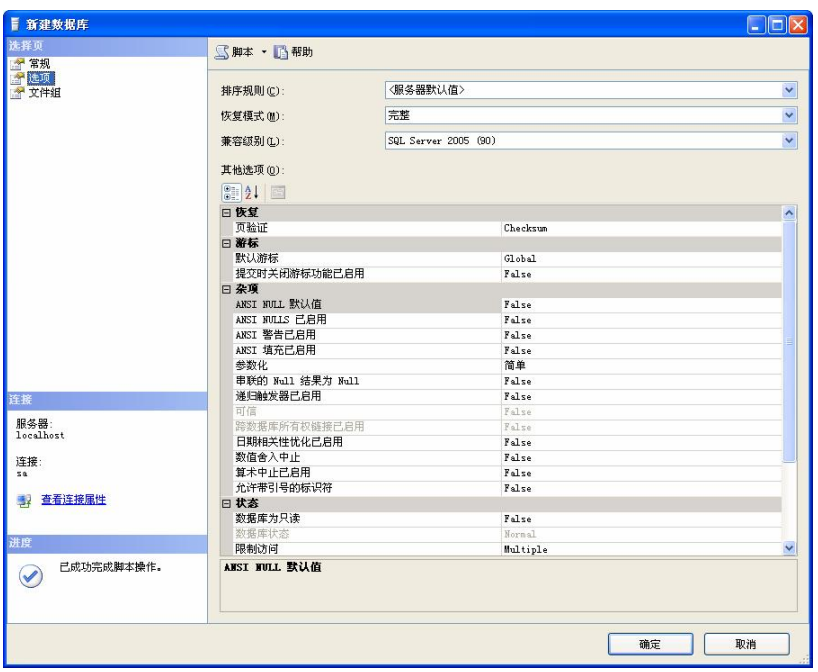


图 3.5　创建数据库选项

4. 设置文件组

单击“选择页”中的“文件组”，可以打开文件组设置页，可以在其中添加或删除文件组，如图 3.6 所示。

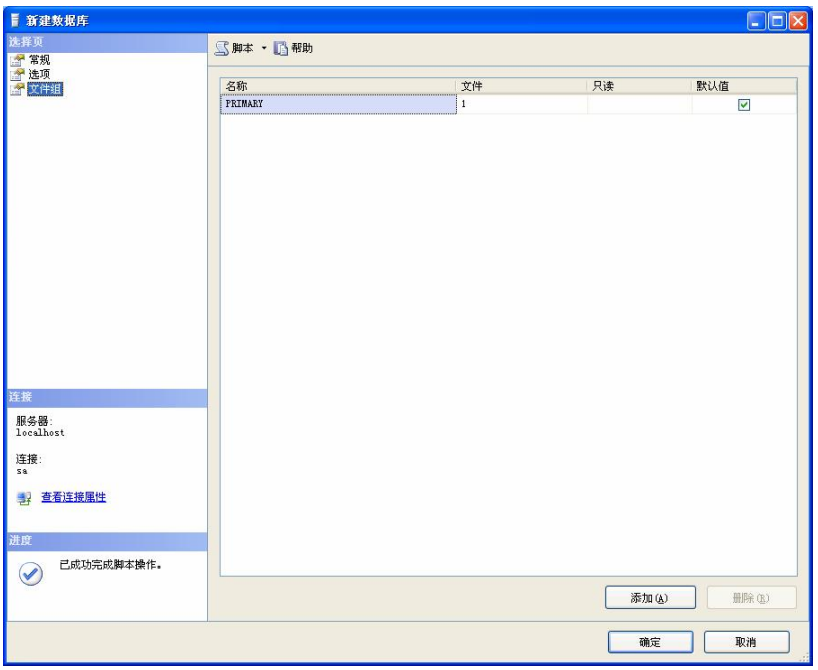


图 3.6　文件组

5. 创建完成

单击“确定”，数据库创建成功。可在对象资源管理器中看到刚创建的数据库。

3.2.3　用 SQL 命令创建数据库

使用 CREATE DATABASE，可以创建一个新数据库及存储该数据库的文件，创建一个数据库快照，或从先前创建的数据库的已分离文件中附加数据库。其创建数据库的语法格式如下：

```
CREATE DATABASE database_name
    [ ON
        [ PRIMARY ] [ <filespec> [,...n ]
        [, <filegroup> [,...n ] ]
    [ LOG ON { <filespec> [,...n ] } ]
    ]
    [ COLLATE collation_name ]
    [ WITH <external_access_option> ]
]
```

其参数说明如下。

（1）database_name：新数据库的名称。

（2）ON：指定数据库文件或文件组的明确定义。

（3）PRIMARY：指明主数据库文件或主文件组。一个数据库只能有一个主文件，如果没有指定 PRIMARY，那么 CREATE DATABASE 语句中列出的第一个文件将成为主文件。

（4）<filegroup> ：控制文件组属性。其语法格式为：

```
<filegroup> ::= FILEGROUP filegroup_name <filespec> [，...n]
```

其中<filespec>：控制文件属性。其格式如下：

```
<filespec> ::=
{
    (
    NAME = logical_file_name ，
    FILENAME = 'os_file_name'
        [, SIZE = size [ KB | MB | GB | TB ] ]
        [, MAXSIZE = { max_size [ KB | MB | GB | TB ] | UNLIMITED } ]
        [, FILEGROWTH = growth_increment [ KB | MB | GB | TB | % ] ]
    ) [,...n ]
}
```

其中有逻辑文件名（NAME），物理文件名（FILENAME），初始大小（SIZE，默认单位为 MB），可增大到的最大容量（MAXSIZE），自动增长（FILEGROWTH）。每个文件之间以逗号分隔。

（5）LOG ON：明确指定存储数据库日志的磁盘文件（日志文件）。LOG ON 后跟以逗号分隔的用以定义日志文件的 <filespec> 项列表。如果没有指定 LOG ON，将自动创建一个日志文件，其大小为该数据库的所有数据文件大小总和的 25% 或 512 KB，取两者之中的较大者。

【例 3.2】创建未指定文件的数据库。

```
USE Master
GO
IF DB_ID (N'mytest') IS NOT NULL DROP DATABASE mytest
GO
CREATE DATABASE mytest   --创建数据库
GO
SELECT name，size，size*1.0/128 AS [Size in MBs]--检查数据库文件和大小
FROM sys.master_files WHERE name = N'mytest'
```

【例 3.3】创建指定文件的数据库，数据文件、日志文件的初始尺寸，大小和增长幅度（单位都是 MB）都已指定。

```
CREATE DATABASE crm
ON
( NAME = crm_dat,
    FILENAME = 'C:\Program Files\Microsoft SQL Server
                    \MSSQL.1\MSSQL\DATA\ 'crmdat.mdf'，
```

```
    SIZE = 10,
    MAXSIZE = 50,
    FILEGROWTH = 5 )
LOG ON
( NAME = crm_log,
    FILENAME = 'C:\Program Files\Microsoft SQL Server
                        \MSSQL.1\MSSQL\DATA\ 'crmlog.ldf",
    SIZE = 5MB,
    MAXSIZE = 25MB,
    FILEGROWTH = 5MB)
```

【例 3.4】通过指定多个数据和事务日志文件创建数据库 Archive。该数据库具有 3 个 100MB 的数据文件和 2 个 100MB 的事务日志文件。主文件是列表中的第一个文件，并使用 PRIMARY 关键字显式指定。事务日志文件在 LOG ON 关键字后指定。请注意用于 FILENAME 选项中各文件的扩展名：.mdf 用于主数据文件，.ndf 用于辅助数据文件，.ldf 用于事务日志文件。

```
CREATE DATABASE Archive
ON
PRIMARY
    (NAME = Arch1,
    FILENAME = "C:\Program Files\Microsoft SQL Server
                        \MSSQL.1\MSSQL\DATA\ archdat1.mdf",
    SIZE = 100MB,
    MAXSIZE = 200,
    FILEGROWTH = 20),
    ( NAME = Arch2,
    FILENAME = "C:\Program Files\Microsoft SQL Server
                        \MSSQL.1\MSSQL\DATA\ archdat2.ndf",
    SIZE = 100MB,
    MAXSIZE = 200,
    FILEGROWTH = 20),
    ( NAME = Arch3,
    FILENAME = "C:\Program Files\Microsoft SQL Server
                        \MSSQL.1\MSSQL\DATA\ archdat3.ndf",
    SIZE = 100MB,
    MAXSIZE = 200,
    FILEGROWTH = 20)
LOG ON
   (NAME = Archlog1,
```

```
    FILENAME = "C:\Program Files\Microsoft SQL Server
                                \MSSQL.1\MSSQL\DATA\ archlog1.ldf",
    SIZE = 100MB,
    MAXSIZE = 200,
    FILEGROWTH = 20),
  (NAME = Archlog2,
    FILENAME = "C:\Program Files\Microsoft SQL Server
                                \MSSQL.1\MSSQL\DATA\ ldf",
    SIZE = 100MB,
    MAXSIZE = 200,
    FILEGROWTH = 20)
```

3.3 管理数据库

3.3.1 查看数据库属性

对象资源管理器是 Management Studio 的一个组件，可连接到数据库引擎、实例和其他服务。它提供了服务器中所有对象的视图，并具有可用于管理这些对象的用户界面。如图 3.7 所示，在打开数据库文件夹目录树后，可以选择各种数据库对象进行信息浏览。

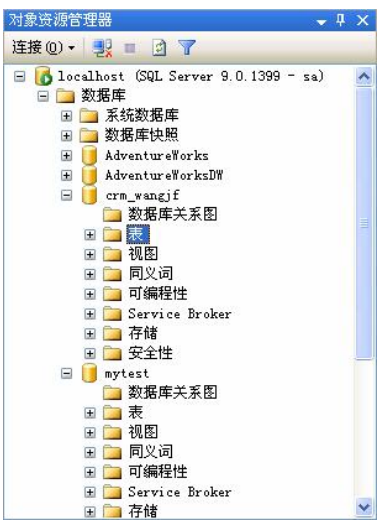


图 3.7 查看数据库

1. 使用 Management Studio 查看数据库属性

下面说明如何使用 Management Studio 中的对象资源管理器查看数据库的当前设置选项，步骤介绍如下。

在对象资源管理器中，展开“数据库”，右键单击要查看的数据库，再单击“属性”。出现数据库属性窗口，如图 3.8 所示。

在“数据库属性”对话框中，各选项页说明如下。

（1）“常规”：可以看到数据库的状态、所有者、创建时间、容量、备份、维护等属性信息。

（2）“文件”： 可以像在创建数据库时那样重新指定数据库文件和事务日志文件的名称、存储、位置、初始容量大小等属性，如图 3.4 所示。

（3）“文件组”：可以添加或删除文件组，要删除文件组，需要移动文件组的文件，如图 3.6 所示。

（4）“选项”：可以设置数据库的许多属性，包括自动、游标、恢复、杂项、状态几部分，常用的数据库选项及其说明如表 3.2 所示。使用 sp_configure 系统存储过程或者 SQL Management Studio 可以设置服务器范围。连接级设置是使用 SET 语句来指定的，详细信息参阅 SQL Server 2005 联机丛书。

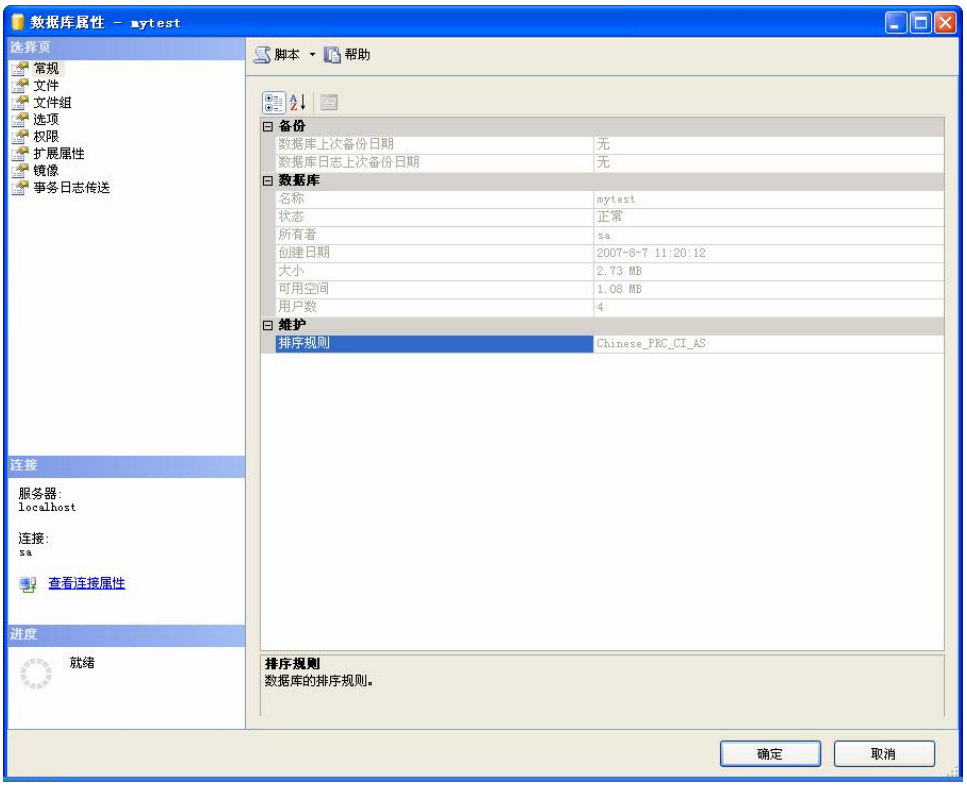


图 3.8　数据库属性

表 3.2　　数据库选项

选　　项	说　　明
自动关闭	指定在最后一个用户退出后，数据库是否完全关闭并释放资源。可取的值包括 True 和 False。如果设置为 True，则在最后一个用户注销之后，数据库会完全关闭并释放其资源
自动收缩	指定数据库文件是否可定期收缩。可取的值包括 True 和 False
默认游标	如果设置为 True，则游标声明默认为 LOCAL。设置如果为 False，则 Transact-SQL 游标默认为 GLOBAL
ANSI NULL 默认值	指定当等于 (=) 和不等于 (<>) 比较运算符用于空值时是否可以进行操作，为 True 时，可以对空值进行等于和不等于操作
允许带引号的标识符	指定在用引号时，是否可以将 SQL Server 关键字用作标识符（对象名称或变量名称）。可取的值包括 True 和 False
递归触发器已启用	指定触发器是否可以递归调用
数据库只读设置	指定数据库是否为只读。可取的值包括 True 和 False。如果设置为 True，则用户只能读取但不能修改数据或数据库对象
限制访问	指定哪些用户可以访问该数据库。多个用户、单个用户或是限制用户

（5）权限：使用“权限”页可以查看或设置数据库安全对象的权限，如图 3.9 所示。单击“添加”按钮可以将用户或角色添加到上边的表格。在上边的表格中选中一个项，然后在“显式权限”表格中可以为其设置适当的权限。详细内容请参考第 12 章中数据库权限管理的内容。

（6）扩展属性：使用扩展属性，可以向数据库对象添加自定义属性。

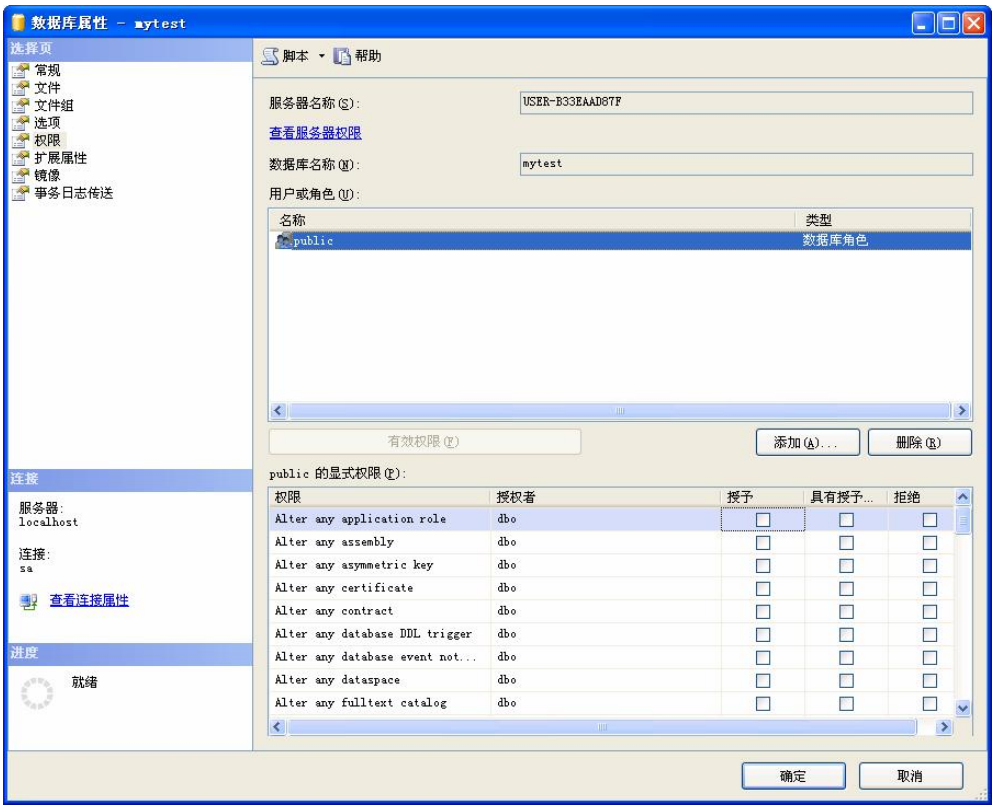


图 3.9　数据库权限设置

（7）镜像：使用镜像可以配置和修改数据库的数据库镜像属性。（注：数据库镜像包含一个数据库的两个副本，这两个副本通常驻留在不同的计算机上，称为“主体数据库”和“镜像数据库”。镜像是将对主体数据库执行的每个插入、更新或删除操作的事务日志应用到镜像数据库。）

（8）日志传送：在此页可以配置和修改数据库的日志传送属性。日志传送能够将事务日志备份从一个数据库（称为“主数据库”）发送到另一台服务器（称为“辅助服务器”）上的辅助数据库。在辅助服务器上，这些事务日志备份将还原到辅助数据库中，并与主数据库保持紧密同步。

2. 使用各种视图、系统函数和系统存储过程来查看数据库属性

表 3.3 列出了返回有关数据库、文件和文件组信息的目录视图、系统函数和系统存储过程。

表 3.3　　数据库信息相关视图、函数、存储过程

视　　图	函　　数	存储过程和其他语句
sys.databases	DATABASE_PRINCIPAL_ID	sp_databases
sys.database_files	DATABASEPROPERTYEX	sp_helpdb
sys.data_spaces	DB_ID	sp_helpfile
sys.filegroups	DB_NAME	sp_helpfilegroup
sys.allocation_units	FILE_ID	sp_spaceused
sys.master_files	FILE_IDEX	DBCC SQLPERF

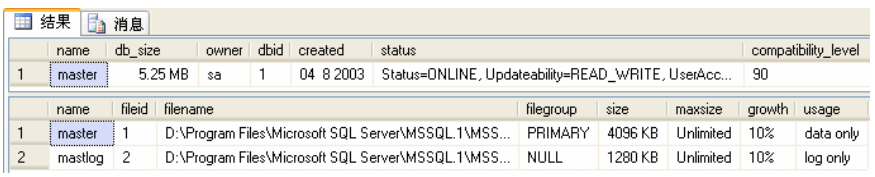
续表

视　　图	函　　数	存储过程和其他语句
sys.partitions	FILE_NAME	
sys.partition_functions	FILEGROUP_ID	
sys.partition_parameters	FILEGROUP_NAME	
sys.partition_range_values	FILEGROUPPROPERTY	
sys.partition_schemes	FILEPROPERTY	
sys.dm_db_partition_stats	fn_virtualfilestats	

【例 3.5】显示 Master 数据库信息。

```
sp_helpdb Master
```

执行结果如图 3.10 所示。



结果　消息

	name	db_size	owner	dbid	created	status	compatibility_level
1	master	5.25 MB	sa	1	04 8 2003	Status=ONLINE, Updateability=READ_WRITE, UserAcc...	90

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	master	1	D:\Program Files\Microsoft SQL Server\MSSQL.1\MSS...	PRIMARY	4096 KB	Unlimited	10%	data only
2	mastlog	2	D:\Program Files\Microsoft SQL Server\MSSQL.1\MSS...	NULL	1280 KB	Unlimited	10%	log only

图 3.10　Master 数据库信息

可以从名称上看到每个视图、函数、存储过程的功能，如 sp_helpfile 返回与当前数据库关联的文件的物理名称及属性。sp_helpfilegroup 返回与当前数据库相关联的文件组的名称及属性，具体参数请查阅 SQL Server 联机丛书。

3.3.2　修改数据库

可以在 Management Studio 中利用数据库属性的设置更改数据库各项参数，也可使用 ALTER DATABASE 命令来更改数据库，ALTER DATABASE 可以更改数据库的属性或其文件和文件组。ALTER DATABASE 的语法格式如下：

```
ALTER DATABASE database_name
{
   <add_or_modify_files>
   | <add_or_modify_filegroups>
   | <set_database_options>
   | MODIFY NAME = new_database_name
   | COLLATE collation_name
}
```

各参数简要说明如下。

（1）database_name：要修改的数据库的名称。

（2）MODIFY NAME = new_database_name：使用指定的名称 new_database_name 重命名数据库。

（3）<add_or_modify_files>：指定添加、修改或删除的数据库文件。

其语法格式为：

```
<add_or_modify_files>::=
{
   ADD FILE <filespec> [ ,...n ]
         [ TO FILEGROUP { filegroup_name | DEFAULT } ]
   | ADD LOG FILE <filespec> [ ,...n ]
   | REMOVE FILE logical_file_name
   | MODIFY FILE <filespec>
}
```

（4）<add_or_modify_filegroups>：指定添加、修改或删除的文件组。

（5）<set_database_options>：更改数据库参数。

【例 3.6】将数据库名 mytest 更改为 mytest1。

```
ALTER DATABASE mytest MODIFY NAME = mytest1
```

【例 3.7】将一个 5MB 的数据文件添加到“AdventureWorks”数据库。

```
ALTER DATABASE AdventureWorks
ADD FILE
(
    NAME = Test1dat2,
    FILENAME = 'C:\Program Files\Microsoft SQL Server
                        \MSSQL.1\MSSQL\DATA\ \t1dat2.ndf',
    SIZE = 5MB,
    MAXSIZE = 100MB,
    FILEGROWTH = 5MB
)
```

【例 3.8】删除上例中添加的数据库文件。

```
ALTER DATABASE AdventureWorks
REMOVE FILE Test1dat2
```

【例 3.9】移动数据库文件的位置。

```
ALTER DATABASE AdventureWorks
MODIFY FILE
(
    NAME = Test1dat2,
    FILENAME = N'c:\t1dat2.ndf'
)
```

【例 3.10】更改数据库文件大小。

```
ALTER DATABASE AdventureWorks
MODIFY FILE
    (NAME = test1dat3,
SIZE = 20MB
)
```

【例 3.11】向数据库添加两个日志文件。

```
ALTER DATABASE AdventureWorks
ADD LOG FILE
(
    NAME = test1log2,
    FILENAME = 'C:\Program Files\Microsoft SQL Server
                                    \MSSQL.1\MSSQL\DATA\ \test2log.ldf",
    SIZE = 5MB,
    MAXSIZE = 100MB,
    FILEGROWTH = 5MB
),
(
    NAME = test1log3,
    FILENAME = 'C:\Program Files\Microsoft SQL Server
                                    \MSSQL.1\MSSQL\DATA\ \test3log.ldf",
    SIZE = 5MB,
    MAXSIZE = 100MB,
    FILEGROWTH = 5MB
)
```

【例 3.12】更改数据库选项。

```
ALTER DATABASE AdventureWorks SET SINGLE_USER    --单用户
ALTER DATABASE AdventureWorks SET READ_ONLY --只读
ALTER DATABASE AdventureWorks    SET AUTO_SHRINK ON    --自动收缩
GO
```

3.3.3　收缩数据库

数据库在使用一段时间后，时常会出现因数据删除而造成数据库中空闲空间过多的情况，需要使用收缩的方式来缩减数据库空间。可在数据库属性选项中选择“Auto shrink”选项，使系统自动收缩数据库。也可用人工的方法来收缩。

1. 使用 Management Studio 收缩数据库

在目标数据库上右键单击，在快捷菜单中选择“任务”→“收缩”→“数据库”，出现“收缩”数据库对话框，如图 3.11 所示。

可选择“在释放未使用的空间前重新组织文件”。选择此项等效于执行具有指定目标百分比选项的 DBCC SHRINKDATABASE。如果选择此选项，用户必须指定目标百分比选项。清除此选项等效于执行具有 TRUNCATEONLY 选项的 DBCC SHRINKDATABASE。

单击“确定”开始收缩数据库，收缩结束之后可以再打开此对话框查看数据库大小。也可对单个数据库文件进行压缩，方法是在目标数据库上右键单击，在快捷菜单中选择“任务”→“收缩”→“文件”。

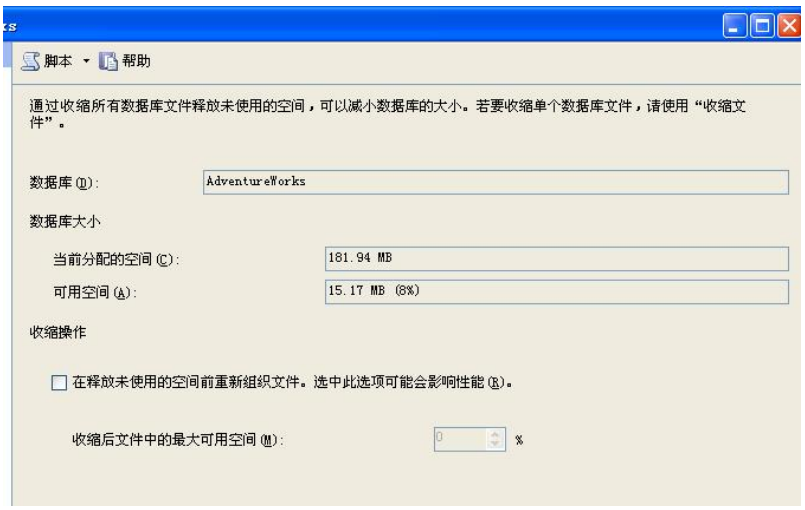


图 3.11　收缩数据库

2. 使用 Transact-SQL 命令收缩数据库

可以使用 DBCC SHRINKDATABASE 和 DBCC SHRINKFILE 命令来收缩数据库。

（1）使用 DBCC SHRINKDATABASE 对数据库进行收缩。

其语法格式为：

```
DBCC SHRINKDATABASE
 ( 'database_name' | database_id | 0
      [,target_percent ]
      [, { NOTRUNCATE | TRUNCATEONLY } ]
)[ WITH NO_INFOMSGS ]
```

各参数说明如下。

① 'database_name' | database_id | 0 ：要收缩的数据库的名称或 ID。如果指定 0，则使用当前数据库。

② target_percent：收缩后的数据库文件中可用空间百分比。

③ NOTRUNCATE：指定在数据库文件中保留所释放的文件空间。如果未指定，将所释放的文件空间释放给操作系统。

④ TRUNCATEONLY：指定数据文件中任何未使用空间被释放给操作系统，并将文件收缩到最后分配的区。

【例 3.13】将 UserDB 用户数据库中的文件减小，以使 UserDB 中的文件有 10%的可用空间。

```
DBCC SHRINKDATABASE (UserDB,10)
```

（2）使用 DBCC SHRINKFILE 对数据库中指定的文件进行收缩。

其语法格式为：

```
DBCC SHRINKFILE
(
    { 'file_name' | file_id }
```

```
    { [ , EMPTYFILE ]
    | [ [ , target_size ] [ , { NOTRUNCATE | TRUNCATEONLY } ] ]
    }
)
```

各参数说明如下。

① file_name：要收缩的文件的逻辑名称。文件名必须符合标识符规则。

② file_id：要收缩的文件的标识（ID）号。

③ target_size：文件大小（MB）。如果未指定，则文件大小减少到默认文件大小。如果指定了 target_size，则 DBCC SHRINKFILE 尝试将文件收缩到指定大小。DBCC SHRINKFILE 不会将文件收缩到小于存储文件中的数据所需要的大小。例如，如果使用 10MB 数据文件中的 7MB，则带有 target_size 为 6 的 DBCC SHRINKFILE 语句只能将该文件收缩到 7 MB，而不能收缩到 6 MB。

④ EMPTYFILE：将指定文件中的所有数据迁移到同一文件组中的其他文件。

【例 3.14】将“AdventureWorks”用户数据库中名为“TestDat1”的文件的大小收缩到 10MB。

```
DBCC SHRINKFILE (TestDat1, 10)
```

3.3.4　删除数据库

可以在 Management Studio 通过数据库右键菜单删除数据库，也可使用 DROP DATABASE 删除数据库。其语法格式为：

```
DROP DATABASE { database_name } [ ,...n ]
```

注意　无法删除系统数据库。不能删除当前正在使用（表示正在打开供任意用户读写）的数据库。只有通过还原备份才能重新创建已删除的数据库。

【例 3.15】删除数据库。

```
DROP DATABASE myTest1                --删除单个数据库
DROP DATABASE myTest2，myTest3       --删除多个数据库
```

本 章 小 结

本章介绍了系统数据库，数据库的创建与管理。强大的数据库管理功能是 SQL Server 的特点，掌握本章内容是对数据库管理员的基本要求。

习　　题

一、选择题

1．当执行 CREATE DATABASE 语句时，将通过复制（　　）数据库中的内容来创建数

据库的第一部分。

A．Master　　B．Msdb　　C．Model　　D．Tempdb

2．主数据库文件（Primary Database File）的扩展名为（　　）。

A．.mdf　　B．.ndf　　C．.ldf　　D．.pdf

3．使用 SQL 语句创建数据库时，SQL 语句中初始大小（SIZE），可增大到的最大容量（MAXSIZE）和自动增长（FILEGROWTH）的默认单位是（　　）。

A．KB　　B．MB　　C．GB　　D．TB

二、填空题

1．系统数据库包括：_____、_____、_____、_____、_____。

2．每个 SQL Server 2005 数据库至少具有两个操作系统文件：一个_____和一个_____。

3．每个数据库文件有两个名称，分别是_____和_____。

三、简答题

1．简述系统数据库的组成和每个数据库的作用。

2．阐述使用 Management Studio 创建数据库的过程。

3．为何要收缩数据库，如何收缩数据库？

本 章 实 训

一、实训目的

1．掌握查看数据库属性的方法。

2．掌握用 Management Studio 与 T-SQL 两种方法建立、修改和删除数据库。

二、实训要求

1．实训前做好上机实训的准备，针对实训内容，认真复习与本次实训有关的知识，完成实训内容的预习准备工作。

2．认真独立完成实训内容。

3．实训后做好实训总结，根据实训情况完成总结报告。

三、实训学时

2 学时。

四、实训内容

1．创建数据库 db_stu。

（1）参考 3.2.2 小节在 Management Studio 下创建数据库 db_stu。

（2）参考例 3.4 小节用 CREATE DATABASE 命令创建名为 db_stu 的数据库，并说明数据库存储的位置、数据库文件的大小和名称。

2．查看数据库属性和信息。

（1）参考 3.3.1 小节，使用 Management Studio 查看系统数据库 Model 和用户数据库 db_Stu 的属性和信息。

（2）参考表 3.2 和例 3.1，使用视图、函数、存储过程查看系统数据库 Model 信息。

3．参考例 3.6～3.12，练习修改数据库 db_stu 的各个属性。

4．参考 3.3.3 小节，使用 Management Studio 界面和 T-SQL 命令两种方法收缩数据库 db_stu。

5．使用 T-SQL 命令删除数据库。

五、实训思考题

1．创建数据库时，主要考虑哪些内容？

2．系统实际运行中，发现数据增长过快，应该如何处理？

第 4 章

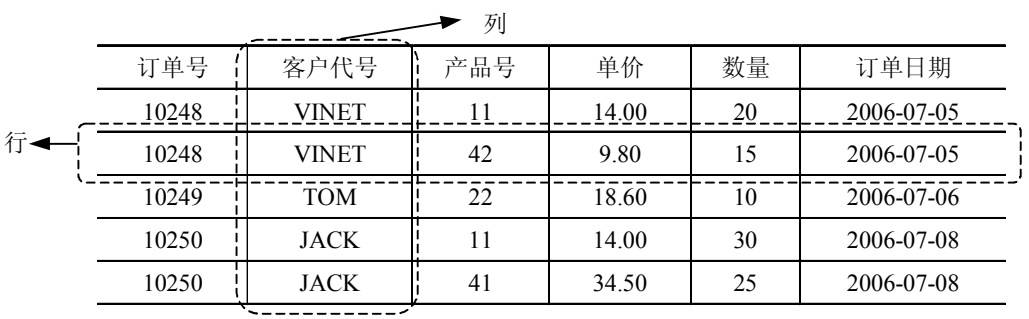
创建和管理 SQL Server 2005 数据表

表是数据库中最重要、最基本、最核心的对象，是实际存储数据的地方。其他数据库对象，如索引、视图等，都是依附于表对象而存在的。对数据库的各种管理操作，实际上主要是对数据库中表的管理操作。

本章主要讲述有关数据表的管理技术，包括表的创建、修改和删除，查看和编辑表中的数据等。通过本章的学习，读者可以理解表的特点和类型，熟练掌握创建和修改表的相关技术。

4.1 表 的 概 念

表是关系模型中表示实体的方式，是数据库存储数据的主要对象。SQL Server 数据库的表由行和列组成，行有时也称为记录，列有时也称为字段或域，如图 4.1 所示。



订单号	客户代号	产品号	单价	数量	订单日期
10248	VINET	11	14.00	20	2006-07-05
10248	VINET	42	9.80	15	2006-07-05
10249	TOM	22	18.60	10	2006-07-06
10250	JACK	11	14.00	30	2006-07-08
10250	JACK	41	34.50	25	2006-07-08

图 4.1　表的结构

在图 4.1 中，表中的每一行数据都表示了一个唯一的、完整的订单信息。表中的每一个字段都是对订单某种属性的描述。

在表中，行的顺序可以是任意的，一般按照数据插入的先后顺序存储。在使用过程中，可以使用排序语句或按照索引对表中的行进行排序。

列的顺序也可以是任意的，对于每一个表，最多可以允许用户定义 1 024 列。在同一个表中，列名必须是唯一的，即不能有名称相同的两个或两个以上的列同时存在于一个表中。但是，在同一个数据库的不同表中，可以使用相同的列名。

SQL Server 2005 把表分成了 4 种类型，即普通表、临时表、已分区表和系统表。普通表就是通常提到的数据库中存储数据的表，是最重要、最基本的表，也是本章要介绍的重点。其他几种类型的表都是有特殊用途的表，往往是在特殊应用环境下，为了提高系统的使用效率而派生出来的。

临时表，顾名思义是临时创建的、不能永久生存的表。临时表被创建之后，可以一直存储到 SQL Server 实例断开连接为止。临时表又可以分为本地临时表和全局临时表，本地临时表只对创建者是可见的，全局临时表在创建之后对所有的用户和连接都是可见的。

已分区表是将数据水平划分成多个单元的表。这些单元可以分散到数据库中的多个文件组里面，实现对单元中数据的并行访问。如果表中的数据量非常庞大，并且这些数据经常以不同的使用方式被访问，那么建立已分区表是一个有效的选择。已分区表的优点就在于可以方便地管理大型表，提高对这些表中数据的使用效率。

系统表和普通表的主要区别在于，系统表储存了有关 SQL Server 服务器的配置、数据库配置、用户和表对象的描述等系统信息。一般来说，只能由 DBA 来使用系统表。

4.2　数据表的创建

在创建表之前，需要先设计表结构，考虑该表中存储哪些数据，应该由哪些列组成，并为每一列指定所属的数据类型。确定表结构之后，就可以在 Management Studio 中进行数据表的创建。

4.2.1　在图形界面下创建数据表

在 SQL Server 2005 中，可以使用可视化工具执行有关表的操作，包括创建表，修改表结构，查看表属性信息等。本节将以“学生选课”数据库中“Student”表的创建为例，介绍如何在图形界面下完成数据表的创建。

在 Management Studio 中，创建一个名为“学生选课”的数据库（创建过程参考本书第 3 章 3.2 节）。右键单击该数据库中的“表”节点，在弹出菜单中选择“新建表”，打开表设计器，如图 4.2 所示。

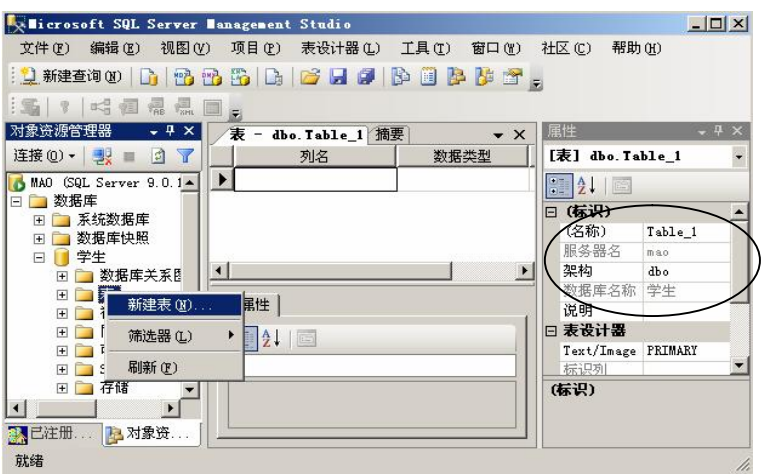


图 4.2　使用表设计器创建表

在如图 4.2 所示的结构中修改表属性，将表名“Table_1”改为“Student”，并输入表中各列的名称，数据类型，是否为空等信息。选中任意一列，在窗体的下半部分将显示该列的主要属性信息，如图 4.3 所示。

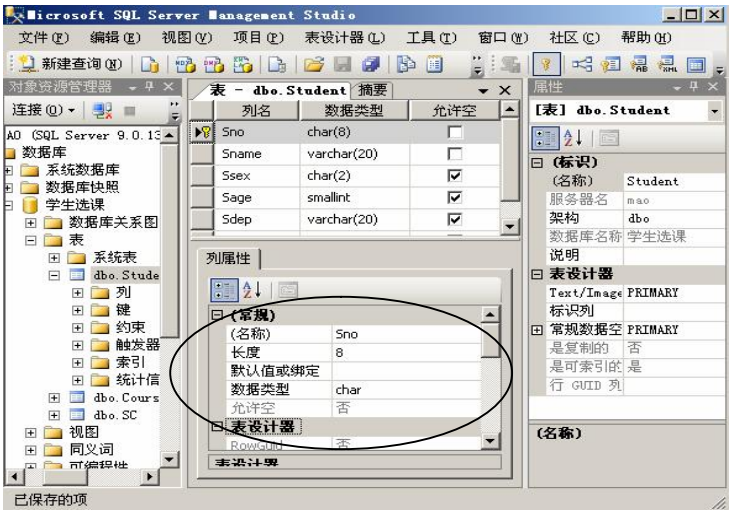


图 4.3　建立 Student 表结构

图 4.3 中创建了一个“Student”表，表中共有 5 列，每一列都由列名，数据类型，是否允许空等属性组成。“Sno”表示学生学号，类型为 char，长度为 8，不允许为空；“Sname”表示学生姓名，类型为 varchar，长度为 20，不允许为空；“Ssex”表示学生性别，类型为 char，长度为 2，允许为空；“Sage”表示学生年龄，类型为 smallint，允许为空；“Sdep”表示学生所属院系，类型为 varchar，长度为 20，允许为空。

SQL Server 中表的主要列属性如下。

1. 允许空（NULL 或 NOT NULL）

该属性定义在输入数据时指定列是否可以为空。在 SQL Server 中，列的默认属性为“允许空”，如“Student”表中的“Ssex”列和“Sage”列。当某列不允许为空时，只需要在该列的“允许空”选项中取消选择复选框就行了，如“Sno”列。

2. 标识规范（IDENTITY）

很多表中使用编号列来标识表中的记录。在插入数据时，用户并不需要指定编号列的值，只要它们互不相同就行了。在这种情况下，可以将编号列设置为标识列。设置成功后，每插入一条记录，系统都会根据增量值自动为该列生成新数据。

定义标识列需要指定两个值：种子值和增量值。表中第一行记录的 IDENTITY 列的值就是种子值，其他行的 IDENTITY 列的值是在前一行值的基础上增加一个增量值得到的。

通常将标识列的类型定义为 int 或 bigint（之所以不使用 smallint 和 tinyint，是由于它们的适用范围较小，当编号范围超过相应的适用范围时，会造成溢出）。例如，在上面创建学生表“Student”时，如果将学号列“Sno”定义为 int 型，那么就可以通过更改窗口下方的列属性将其设置为标识列，然后自动生成该列的值，如图 4.4 所示。

由于 IDENTITY 属性列的增长是单向的，所以一般情况下不能手工为设置了 IDENTITY 属

性的列添加数据。而且，如果删除了这些列中的部分数据，还会造成标识符序列空缺——已删除的标识符值是不能重用的，系统不会自动补充这部分数据值。可以通过下述方法解决这种标识符序列空缺的问题。

使用 SET IDENTITY_INSERT 语句将标识列设置为可以插入数据的状态。该语句的基本格式为：

SET IDENTITY_INSERT < 表名 > { ON | OFF }

ON 表示可以插入数据，OFF 表示拒绝插入数据。

3. 定义主键（PRIMARY KEY）

主键是表中的一列或者一组列，它的值可以唯一标识表中的一行记录。例如，每一个学生入学后都有一个学号，而且该学号和其他任何一个学生都是不同的，或者说，确定了一个学号就确定了一个学生。因此，在设计表时，可以把学号定义为主键。

以“Student”表为例，将“Sno”列定义为主键列，有 3 种可视化方法。

① 在 Management Studio 中，右键单击“Student”表，选择“修改”菜单项，打开“Student”表结构。选中主键列“Sno”，单击右键，选择“设置主键”项，如图 4.5 所示。此时“Sno”列的左侧出现了一个图标，表明主键设置成功。

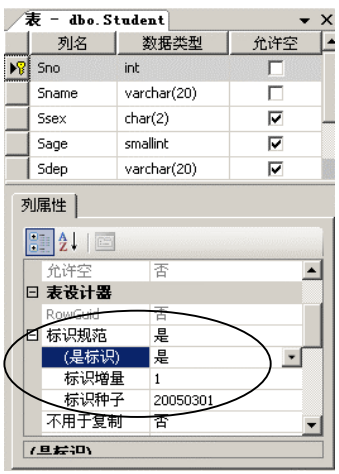


图 4.4　设置标识列

图 4.5　设置主键

② 按照①中步骤，选中“Sno”列，选择“表设计器”菜单，在打开的下拉菜单中选择“设置主键”，即可将“Sno”列设置为主键。

③ 按照①中步骤，选中“Sno”列，在工具栏上单击图标，同样可以将“Sno”列设置为主键。

当一列被设置为主键后，该列的“允许空”选项将自动取消，因为主键列必须输入数据。

当要取消某列的主键属性时，步骤和设置主键的过程完全一样，只是选项由“设置主键”变成了“移除主键”。

在表设计器中单击右键，在弹出菜单中选择“索引/键”（或者左键单击“表设计器”菜单，在下拉菜单中选择“索引/键”），打开“索引/键”对话框，在这里可以查看、创建、修改和删除键，如图 4.6 所示。

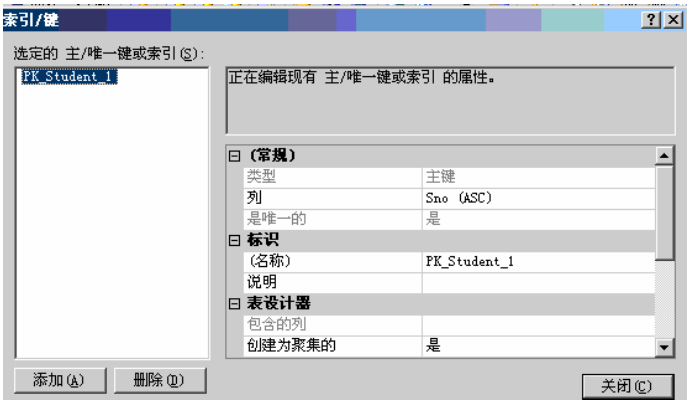


图 4.6　管理主键

表的各项组成、属性等设置完成后，单击工具栏中的“”图标，保存表的定义即可。

4.2.2　用 SQL 命令创建数据表

在 SQL Server 2005 中，既可以使用表设计器工具在图形界面下创建表，也可以使用 CREATE TABLE 语句创建表。后者是一种最强大、最灵活的创建表的方式。它的基本语法如下：

```
CREATE TABLE  表名
(
    列名 1   数据类型和长度 1   列属性 1,
    列名 2   数据类型和长度 2   列属性 2,
                    ⋮
    列名 n   数据类型和长度 n   列属性 n   )
```

由此可以看出，在 CREATE TABLE 语句中需要指出的元素与在表设计器中相同，包括表名、列名、列的数据类型以及列属性等。

在 Management Studio 中，单击工具栏上的“新建查询”，在查询窗口中输入下面的脚本命令，可以创建 4.2.1 小节中定义的“Student”表。

```
USE 学生选课
GO
CREATE TABLE Student
(
    Sno         int             PRIMARY KEY IDENTITY(20050301,1),
    Sname       varchar(20)     NOT NULL,
    Ssex        char(2),
    Sage        smallint,
    Sdept       varchar(20)
)
```

其中，USE 语句表示选择数据库；PRIMARY KEY 属性定义“Sno”字段为主键；IDENTITY（20050301，1）属性定义“Sno”列的第一行记录值为 20050301，以后每一行的“Sno”列值在此基础上依次递增，增量为 1；“NOT NULL”表示不允许“Sname”列为空。

使用同样的方法在“学生选课”数据库中创建课程表“Course”和选课表“SC”，SQL 语句如下：

```
CREATE TABLE Course
(
    Cno             char(4)             PRIMARY KEY,
    Cname           varchar(40)         NOT NULL,
    Cpno            char(4),
    Ccredit         smallint,
    FOREIGN KEY Cno REFERENCES Course(Cno)
);
CREATE TABLE SC
(
    Sno             int,
    Cno             char(4),
    Grade           smallint,
    PRIMARY KEY (Sno, Cno),
    FOREIGN KEY Sno REFERENCES Student(Sno),
    FOREIGN KEY Cno REFERENCES Course(Cno)
);
```

其中，Cno 代表课程号，Cname 代表课程名，Cpno 代表先修课课程号，Ccredit 代表学分，Grade 代表分数。“FOREIGN KEY”命令用于定义外键，“REFERENCES”用于指定外键所参考的表。

在创建表的过程中，除了在列中直接指定数据类型和属性之外，还可以对某些列进行计算。或者说，某些列的值可以不用输入，通过计算可以得到。例如，在下面创建学生成绩表“Sgrade”的示例中，就使用了一个计算列。

```
CREATE TABLE Sgrade
(
    Sno             int,
    Grade1          int,
    Grade2          int,
    Grade3          int,
    Total    AS     Grade1+Grade2+Grade3
)
```

在上面的脚本中，共创建了 5 个列，其中，“Total”列没有指定数据类型，但是它的值是 Grade1+Grade2+Grade3，也就是说该列的值是通过计算得到的。在这里，“AS”是 SQL Server 使用的关键字。

需要注意的是，一般情况下，计算列的数据并不进行物理存储，它仅仅是一个虚拟列，只用于显示。如果希望将该列的数据物理化存储，可以使用 PERSISTED 关键字。例如，下面的脚本执行后，“Total”列的值就会存储在数据库中，当该列所依赖的其他列的数据发生

改变时，该列中的数据也会自动更新。

```
CREATE TABLE Sgrade
(
    Sno             int,
    Grade1          int,
    Grade2          int,
    Grade3          int,
    Total   AS   Grade1+Grade2+Grade3      PERSISTED
)
```

4.3 数据表的修改

表创建以后，用户可以使用函数、存储过程查看有关表的各种信息，并根据需要来修改表的结构，在表中增加新列，删除已有的列或者修改已有列的属性等。

4.3.1 查看表属性

在 Management Studio 中，选中要查看的数据表，单击鼠标右键，选择“属性”，将打开“表属性”对话框，如图 4.7 所示。

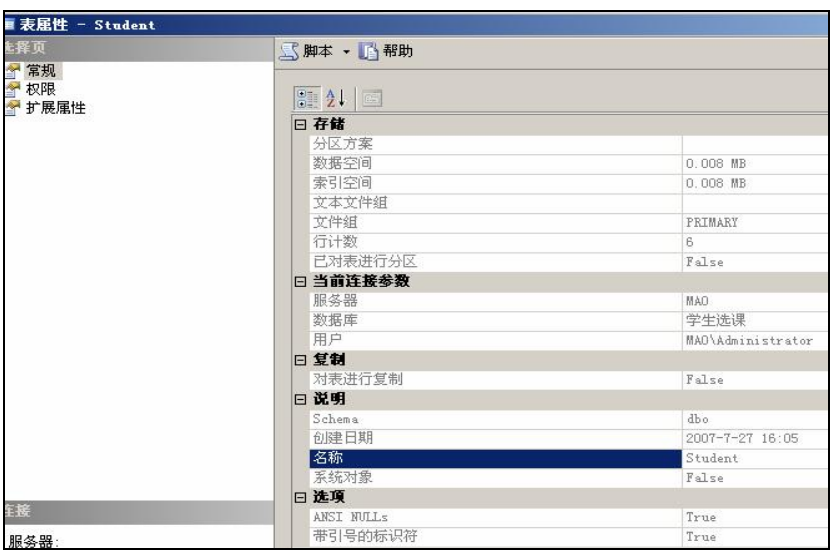


图 4.7 通过“表属性”对话框查看表属性

在该对话框中，选择左侧的“常规”项，在右侧窗口可以看到该表的相关属性信息。如表名，创建日期，存储所占的空间，所属文件组，表中记录行数，及是否已进行分区等信息。

另外，还可以通过 sp_help 存储过程来查看表结构信息。图 4.8 显示了使用 sp_help 查看“Student”表结构信息的过程。

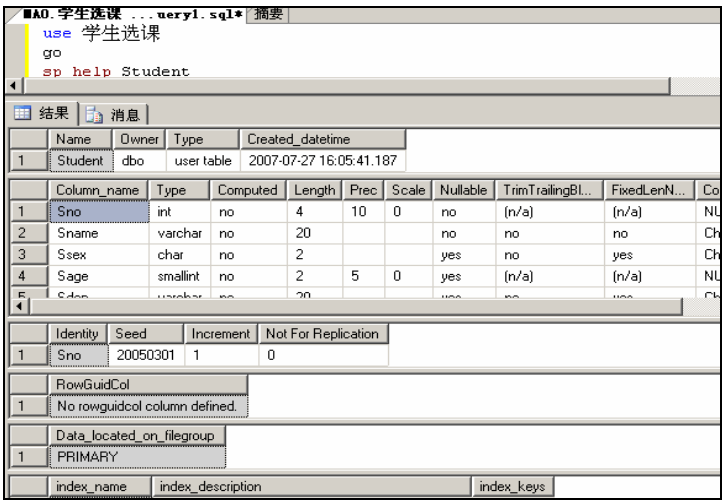


图 4.8　使用存储过程查看表结构信息

4.3.2　修改表结构

在 Management Studio 中，选中要查看的数据表，单击右键，在弹出菜单中选择“修改”，可以打开表设计器，在图形界面下修改表结构，具体方法可以参照 4.2.1 小节理解。

右键单击表名，在弹出菜单中选择“重命名”，可以更改表名。

上述两种操作都是在图形方式下完成的，另外一种很常用的方法是使用 ALTER TABLE 语句修改表结构，包括添加列、修改列名及列属性、删除列等。

1. 向表中添加列

使用 ALTER TABLE 语句向表中添加列的基本语法为：

```
ALTER TABLE 表名 ADD 列名 数据类型和长度 列属性
```

【例 4.1】在表“Student”中新增加一列入学年月，列名为“Sentrance”，数据类型为 datetime，允许空值。

执行以下语句：

```
ALTER TABLE Student ADD Sentrance datetime
```

执行完毕，表“Student”中将新增一列“Sentrance”。

以下有一些需要注意的地方。

（1）当向表中新增一列时，最好为该列定义一个默认约束，使该列有一个默认值。这一点可以使用关键字 DEFAULT 来实现。

（2）如果增加的新列没有设置默认值，并且表中已经有了其他数据，那么必须指定该列允许空值，否则，系统将产生错误信息。

【例 4.2】要求在表“Student”中添加一列入学年月，列名为“Sentrance”，数据类型为 datetime，并且该列不能为空。

如果此时执行下面的 SQL 语句，系统将会报错，如图 4.9 所示。

```
USE 学生选课
GO
ALTER TABLE Student ADD Sentrance datetime NOT NULL
```

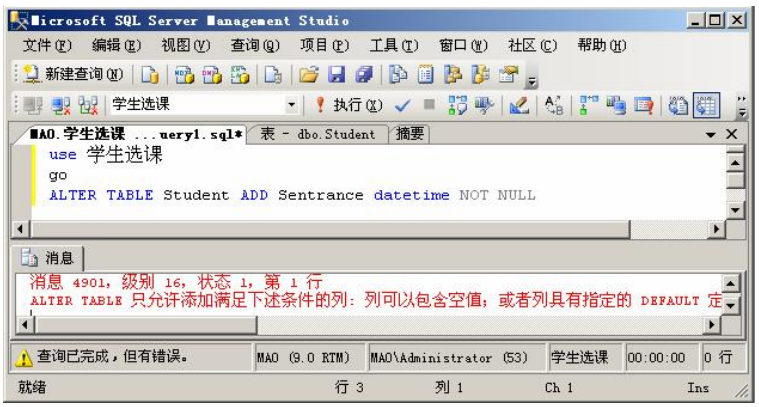


图 4.9　向表中添加非空数据列示例

纠正的方法就是给新增的列设置默认约束，将上述 SQL 命令改为：

```
USE 学生选课
GO
ALTER TABLE Student ADD Sentrance datetime NOT NULL DEFAULT('2005-09-01')
```

该命令执行后，会在“Student”中添加“Sentrance”列，并将每行数据的“Sentrance”值赋为“2005-09-01”。

2. 修改列属性

使用 ALTER TABLE 语句修改列属性的基本语法为：

```
ALTER TABLE 表名 ALTER COLUMN 列名 新数据类型和长度 新列属性
```

例如，上例中创建的“Sentrance”列是 datetime 类型，并且不允许为空。现在要将该列改为 smalldatetime 类型，并且允许为空。SQL 语句如下：

```
ALTER TABLE Student ALTER COLUMN Sentrance smalldatetime NULL
```

3. 删除列

使用 ALTER TABLE 语句删除列的基本语法为：

```
ALTER TABLE 表名 DROP COLUMN 列名
```

例如，要删除“Student”表中的“Sentrance”列，可以执行下面的 SQL 语句：

```
ALTER TABLE Student DROP COLUMN Sentrance
```

4. 修改列名和表名

可以使用 sp_rename 存储过程对表和表中的列进行重命名，重命名的基本语法为：

```
sp_rename 原对象名, 新对象名
```

例如，如果想将“Student”表改名为“StudentInfo”，可以执行以下 SQL 语句：

```
sp_rename Student, StudentInfo
```

4.3.3　删除数据表

删除表就是将表中的数据和表的结构从数据库中永久性地移除。也就是说，表一旦被删除，就无法恢复，除非还原数据库。因此，执行此操作时应该慎重。

在 Management Studio 中，选中要删除的数据表，单击右键，在弹出菜单中选择“删除”，

将弹出“删除对象”对话框，如图 4.10 所示。单击“确定”按钮，选中的表就从数据库中被删除了。

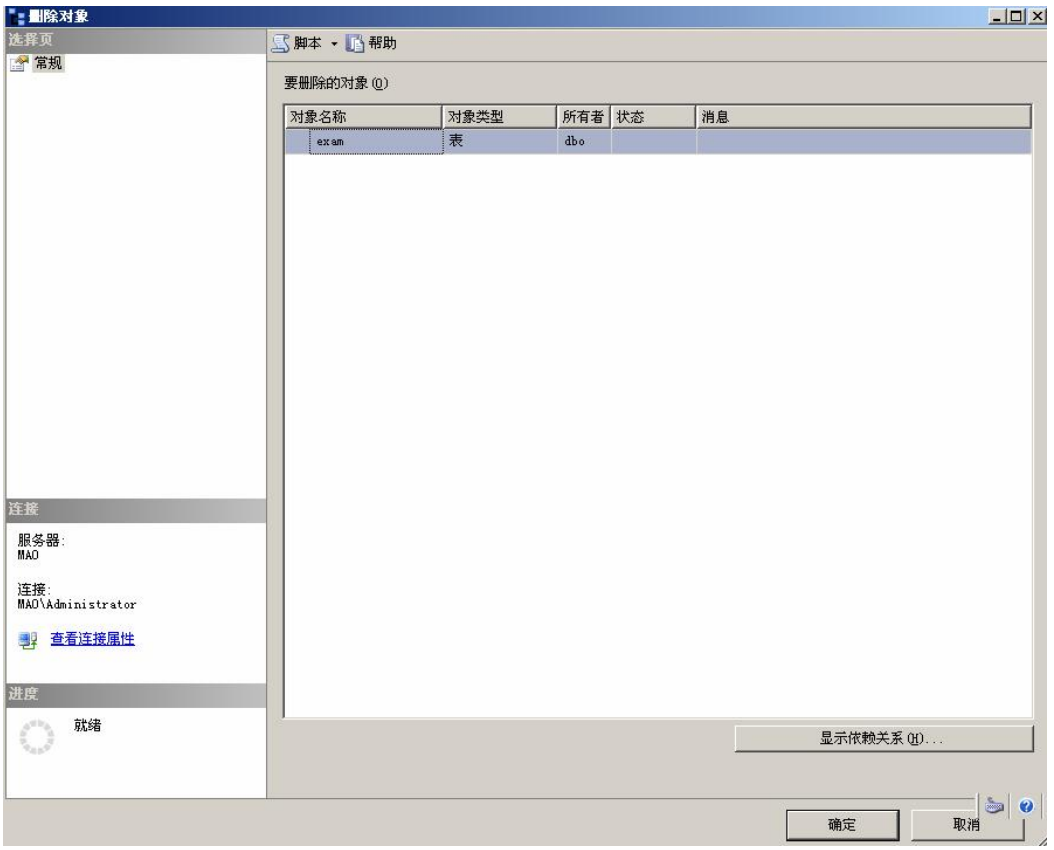


图 4.10　确认删除表

也可以使用 DROP TABLE 语句来完成数据表的删除。该语句的语法非常简单，如下所示：

DROP TABLE 表名

【例 4.3】删除“学生选课”数据库中的“Exam”表。

语句如下：

```
USE 学生选课
GO
DROP TABLE Exam
```

在使用 DROP TABLE 语句删除数据表时，需要注意以下几点。

（1）DROP TABLE 语句不能删除系统表。

（2）DROP TABLE 语句不能删除正被其他表中的外键约束参考的表。当需要删除这种有外键约束参考的表时，必须先删除外键约束，然后才能删除该表。

（3）当删除表时，属于该表的约束和触发器也会自动被删除。如果重新创建该表，必须重新创建相应的规则、约束和触发器等。

（4）DROP TABLE 语句可以一次性删除多个表，表之间用逗号分开。

4.4 添加和修改表数据

表创建以后，往往只是一个没有数据的空表。因此，向表中输入数据可能是创建表之后首先要执行的操作。无论表中是否有数据，都可以根据需要向表中添加数据。当表中的数据不合适或者出现了错误时，可以更新表中的数据。如果表中的数据不再需要了，则可以删除这些数据。本节将详细描述如何添加、更新和删除表中数据。

4.4.1 手工添加表数据

在 Management Studio 中，选中要查看的数据表，单击右键，在弹出菜单中选择“打开表”，可以打开查询表数据的窗口，该窗口显示了表中已经存储的数据，数据列表的最后有一个空行，如图 4.11 所示。

图 4.11 编辑表数据

插入数据时，将光标定位在空白行某个字段的编辑框中，就可以输入新数据。编辑完成后，选中其他行，即可完成数据的插入。

注意 在编辑表中数据的过程中，可能会遇到这样一个问题，就是无法输入中文。这种情况出现在表的第一列为标识列，并且打开表时表中已经存在数据时。此时，光标定位在第一条记录的标识列中，再将光标定位到其他的编辑框，将无法输入中文。解决方法是在定义表结构时，不要将标识列定义为表的第一列，或者使用 SQL 命令进行数据的插入（这一点将在 4.4.3 小节具体介绍）。

4.4.2 查看表记录

在 Management Studio 中，选中要查看的数据表，单击右键，在弹出菜单中选择“打开表”，打开查询表数据的窗口，就可以查看该表中现已存储的数据。

另外，用户还可在查询窗口中使用 SELECT 命令查看一个或多个表中的数据。SELECT 是一个非常灵活和强大的命令，它的具体功能和使用将在第 5 章进行详细介绍。

4.4.3　用 INSERT 语句插入数据

在 4.4.1 节中讲述了如何在图形窗口下插入数据，这种方式比较直观，容易理解和操作，但是存在一定缺陷，比如前面提到的中文输入问题。但是如果使用 INSERT 语句来进行数据插入的话，就可以避免发生类似的问题。

INSERT 语句的基本格式如下：

INSERT INTO 表名　(列名 1，列名 2, …, 列名 n)

VALUES　(值 1, 值 2，…, 值 n)

其中，INSERT 子句指定要插入的数据表名，并且可以同时指定表的列名称；VALUES 子句指定要插入的数据。

例如，在图 4.11 中，无法用中文输入学生的姓名、性别等信息，但是用户可以使用下面的语句完成数据的插入，如图 4.12 所示。

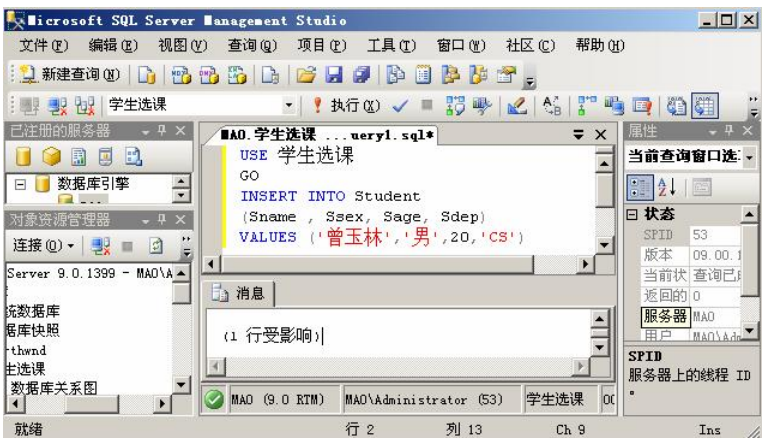


图 4.12　使用 INSERT INTO 插入一条记录

```
USE 学生选课
GO
INSERT INTO Student
(Sname , Ssex, Sage, Sdep)
VALUES ('曾玉林', '男', 20, 'CS')
```

有的读者也许会问，在“Student”表中，不是还有一个“Sno”列吗？为什么插入的时候没有指定这一列的值呢？这是因为“Sno”是标识列，它的值由系统自动计算并保存在相应的记录中。

另外，当完全按照表中列的存储顺序来设置 VALUES 子句中的值时，可以在 INSERT INTO 子句中省略列名，如图 4.13 所示。

注意　向表中插入数据时，数字数据可以直接插入，但是字符数据和日期数据要用英文单引号引起来，不然系统就会提示错误。

另外，一般情况下，使用 INSERT 语句一次只能插入一行数据，但是如果在 INSERT 语句中包含了 SELECT 语句，就可以一次插入多行数据了。使用 SELECT 语句插入数据的基本

语法形式为：

INSERT INTO 表名 （列名 1, 列名 2, …, 列名 *n*）

SELECT 语句

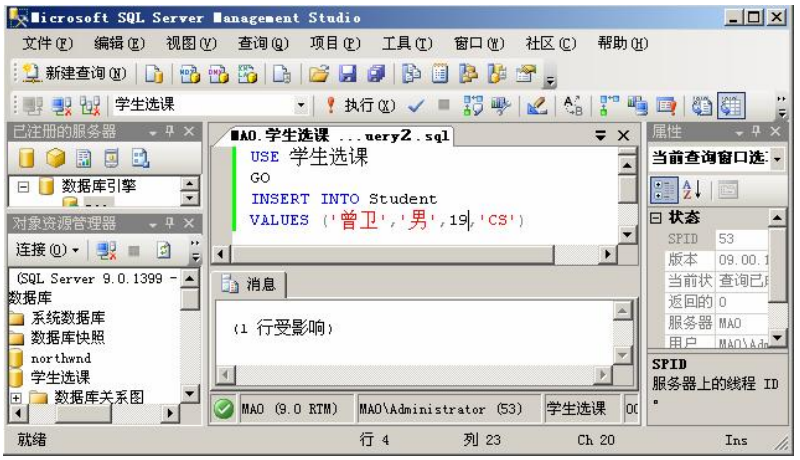


图 4.13　按照表中列存储顺序插入记录

如图 4.14 所示的示例中，使用 INSERT INTO 形式向“Exam”表中插入了从“Student”表中检索出来的数据。

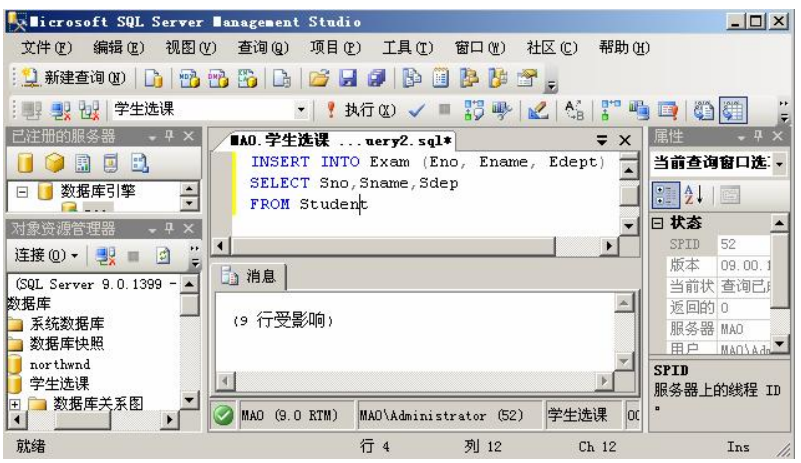


图 4.14　使用 INSERT INTO 形式插入多行数据

使用 INSERT INTO 形式插入多行数据时，需要注意下面两点：

（1）要插入的数据表必须已经存在；

（2）要插入数据的表结构必须和 SELECT 语句的结果集兼容，也就是说，二者的列的数量和顺序必须相同，列的数据类型必须兼容。

4.4.4　用 UPDATE 语句更新数据

可以使用 UPDATE 语句更新表中已经存在的数据，该语句既可以一次更新一行数据，也可以一次更新多行数据，甚至可以一次更新表中的全部数据行。

UPDATE 语句的基本语法如下所示：

```
UPDATE 表名
SET 列名 1＝值 1, 列名 2＝值 2, …, 列名 n＝值 n
WHERE 更新条件表达式
```

当执行 UPDATE 语句时，如果使用了 WHERE 子句，则指定表中所有满足 WHERE 子句条件的行都将被更新，如果没有指定 WHERE 子句，则表中所有的行都将被更新。

【例 4.4】将学生表“Student”中“刘尘”所属的学院由“CS”改为“SS”。

语句如下：

```
USE 学生选课
GO
UPDATE Student SET Sdep = 'SS'
WHERE Sname = '刘尘'
```

执行结果如图 4.15 所示，有一行记录被更新。

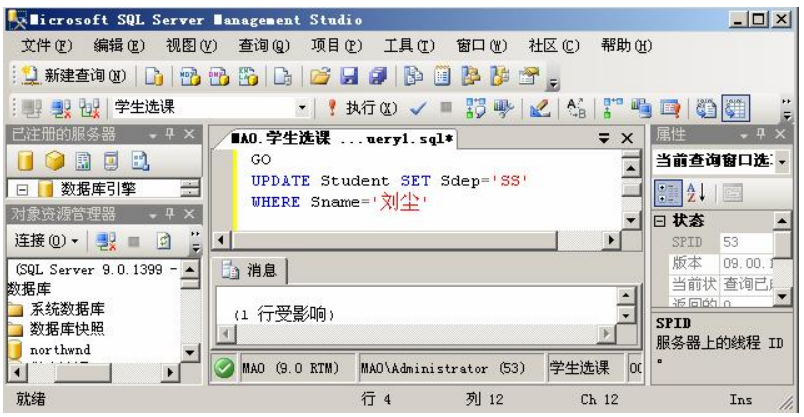


图 4.15　更新数据表数据

更新数据时，每个列既可以被直接赋值，如例 4.3，也可以通过计算得到新值。

【例 4.5】将所有学生的年龄增加 1 岁。

```
USE 学生选课
GO
UPDATE Student SET Sage=Sage+1
```

4.4.5　用 DELETE 语句删除数据

当表中的数据不再需要的时候，可以将其删除。一般情况下，可以使用 DELETE 语句删除表中的数据。该语句可以从一个表中删除一行或多行数据。

使用 DELETE 语句删除数据的基本语法形式如下：

```
DELETE FROM 表名
WHERE 条件表达式
```

在 DELETE 语句中，如果使用了 WHERE 子句，表示从指定的表中删除满足 WHERE 子句条件的数据行。如果没有使用 WHERE 子句，则表示删除指定表中的全部数据。

【例 4.6】删除“Exam”表中姓名为“刘尘”的数据记录。

```
USE 学生选课
GO
DELETE FROM Exam
WHERE Ename = '刘尘'
```

执行结果如图 4.16 所示。

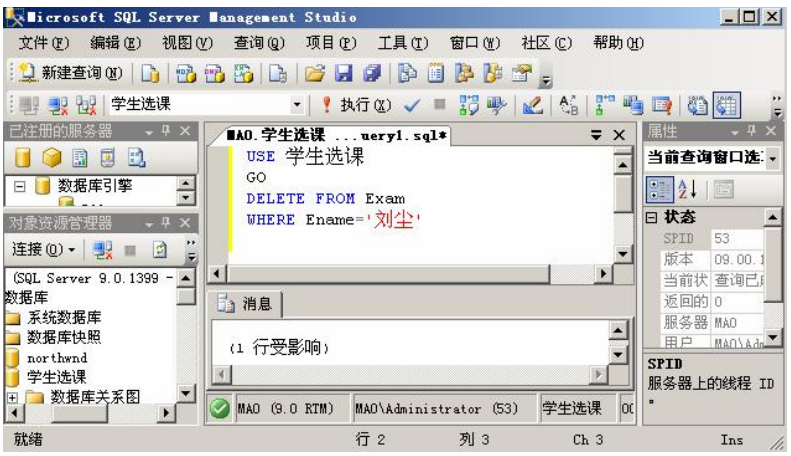


图 4.16　删除满足条件的数据记录

如果要删除“Exam”表中的所有数据，直接执行 DELETE FROM Exam 或者 DELETE Exam 就可以了。

注意　在删除数据时需要注意的是，DELETE 语句只是删除表中存储的数据，表结构依然存在于数据库中。如果需要删除表结构，应该使用前面介绍的 DROP TABLE 语句。

在删除表中的全部数据时，还可以使用 TRUNCATE TABLE 语句。该语句和 DELETE 语句都可以将表中的数据全部删除，但是，两条语句又有不同的特点。一般情况下，当使用 DELETE 语句删除数据时，被删除的数据记录在日志中。而当使用 TRUNCATE TABLE 语句删除数据时，系统将立即释放表中数据和索引所占的空间，并且这种数据变化不被记录在日志中。

本 章 小 结

表是数据库中最核心、最重要的一个内容，它负责存储数据库中的数据。本章介绍了有关表的基本概念和 SQL Server 2005 中常见的数据表类型，然后在此基础上重点介绍了数据表的管理技术，详细讲解了如何在图形方式下和 SQL 命令方式下进行表的创建、修改、删除操作，以及表中数据的添加、更新和删除操作。

创建、更改、删除表的 SQL 命令分别为：CREATE TABLE、ALTER TABLE、DROP TABLE。向表中插入数据的命令为 INSERT INTO，修改表中数据的命令为 UPDATE，删除表中数据的

命令为 DELETE。各命令的语法形式都在文中进行了详细的介绍。

本章涉及的 SQL 命令，需要读者在学习过程中多上机进行练习，做到熟练掌握。

习　　题

一、选择题

1．下面有关 INSERT…SELECT 语句的描述中，哪些是正确的（　　）。

A．新建一个表　　B．语法不正确

C．一次可以插入多行数据　　D．必须向已有的表中插入数据

2．假设列中的数据变化规律如下，请问哪种情况可以使用 IDENTITY 列定义（　　）。

A．1，2，3，4，5，…　　B．10，20，30，40，50，…

C．1，1，2，3，5，8，13，…　　D．5，10，15，20，25，30，…

3．设关系数据库中一个表 S 的结构为 S（SN，CN，grade），其中 SN 为学生名，CN 为课程名，二者均为字符型；grade 为成绩，数值型，取值范围为 0～100。若要把“张二的化学成绩 80 分”插入 S 中，则可用（　　）。

A．ADD
　　INTO S
　　VALUES（'张二', '化学', '80'）

B．INSERT
　　INTO S
　　VALUES（'张二', '化学', '80'）

C．ADD
　　INTO S
　　VALUES（'张二', '化学', 80）

D．INSERT
　　INTO S
　　VALUES（'张二', '化学', 80）

4．下列关于 DROP TABLE 语句的描述正确的是（　　）。

A．可以删除任何表

B．一次只能删除一个表

C．执行该语句时，只删除表中数据，表结构依然存在于数据库中

D．执行该语句时，将删除表结构

二、填空题

1．在 Microsoft SQL Server 2005 系统中，可以把表分成 4 种类型，即_____、_____、_____、_____。

2．NULL 表示_____，而不是没有或 0。

3．向指定的表中插入数据时，如果希望为某个列指定空值，可以使用_____关键字或_____关键字。

三、简答题

1．什么是标识列？它有什么作用？

2．表分为哪几种类型？

3．表创建以后，表中列的数据类型是否可以再修改？

本 章 实 训

一、实训目的

1．理解表的基本概念和在数据库中的作用。

2．掌握创建表的方法。

3．掌握修改表和删除表的方法。

4．掌握向表中添加、更新、删除数据的方法。

二、实训要求

1．实训前做好上机实训的准备，针对实训内容，认真复习与本次实训有关的知识，完成实训内容的预习准备工作。

2．认真独立完成实训内容。

3．实训后做好实训总结，根据实训情况完成总结报告。

三、实训学时

6 学时。

四、实训内容

1．练习创建和修改表。

首先创建“MySql”数据库，然后在“MySql”数据库中创建一个“Student”表，并练习修改该表的结构。

（1）启动 Microsoft SQL Server Management Studio 工具，新建一个查询。

（2）使用 CREATE TABLE 语句创建图 4.3 所示的“Student”表，并手工添加几行数据记录。

（3）使用 ALTER TABLE…ADD COLUMN 语句修改“Student”表，为该表新增一列“Sentrance”（入学日期），列的数据类型为 datetime 型，允许为空。

（4）使用 ALTER TABLE…ALTER COLUMN 语句修改“Student”表，将“Sentrance”列的属性置为 NOT NULL。

（5）使用 ALTER TABLE…DROP COLUMN 语句删除“Sentrance”列。

2．练习使用 INSERT 语句插入数据。

（1）启动 Microsoft SQL Server Management Studio 工具，新建一个查询。

（2）打开实训内容 1 创建的“Student”表。

（3）使用 INSERT 语句在“Student”表中插入表 4.1 所示的 2 条记录。

表 4.1　　Student 表记录

学　号	姓　名	性　别	年　龄	所 属 院 系
20060901	白浩然	男	20	SS
20070901	崔蓝	女	19	CS

（4）新建一个表“Stu1”，将“Student”表中的数据一次性地插入到“Stu1”中。

（5）分析上面两条 INSERT 语句执行的结果。

3．练习使用 UPDATE 语句。

（1）启动 Microsoft SQL Server Management Studio 工具，新建一个查询。

（2）使用 UPDATE 语句更新“Student”表中的数据，将所有学生的年龄增加 1 岁。

（3）使用 UPDATE 语句更新“Student”表中的数据，将所属院系为“CS”的字段记录改为“计算机学院”。

（4）查看数据更新后的结果。

4．练习使用删除语句。

（1）删除表“Stu1”中的所有“CS”学院的学生数据。

（2）将表“Stu1”从数据库中删除。

五、实训思考题

1．当使用 INSERT 语句向表中插入部分列数据时，应该注意什么？

2．DELETE 语句和 DROP TABLE 语句的作用分别是什么？

第 5 章

数据查询

查询数据是使用数据库的最基本，也是最重要的方式。在 Microsoft SQL Server 2005 系统中，可以使用 SELECT 语句执行数据查询的操作。该语句具有非常灵活的使用方式和丰富的功能，它既可以完成简单的单表查询，也可以完成复杂的连接查询和嵌套查询。

本章将以“学生选课”数据库为例，在“Student”表、“Course 表”和“SC”表的基础上讲述有关数据的查询技术，包括基本查询和高级查询（在一般数据查询的基础上，对查询结果进行分组、统计等操作）。通过本章的学习，读者可以比较全面地了解使用 SELECT 语句进行数据查询的技术。

5.1 SELECT 语句解析与简单 SQL 语句

如果希望查看表中的数据，可以使用 SELECT 语句来完成任务。SELECT 语句有 3 个基本的组成部分：SELECT 子句、FROM 子句和 WHERE 子句。其一般格式为：

```
SELECT [ ALL | DISTINCT ] < 目标列表达式 > [，<目标列表达式>] …
FROM  < 表名或视图名 > [, < 表名或视图名 > ] …
[ WHERE  < 条件表达式 >]
[ GROUP BY  < 列名 1 > [ HAVING < 条件表达式 > ] ]
[ ORDER BY  < 列名 2 > [ ASC | DESC ] ]
```

SELECT 子句用于指定将要查询的列名称，FROM 子句指定将要查询的对象（表或视图），WHERE 子句指定数据应该满足的条件。一般情况下，SELECT 子句和 FROM 子句是必不可少的，WHERE 子句是可选的。如果没有使用 WHERE 子句，那么表示无条件地查询所有的数据。

如图 5.1 的示例中，使用了一个简单的 SELECT 语句查询“Student”表中的所有数据。在这个示例中，SELECT 子句后面的*表示所有列，FROM 子句后面的表名称是要查询的对象，没有使用 WHERE 子句，表示查询所有的数据。

如果 SELECT 语句中有 GROUP 子句，则将查询结果按照< 列名 1 >的值进行分组，将该属性列值相等的记录作为一个组。如果 GROUP 子句带有 HAVING 短语，则只有满足指定条件的组才会输出。

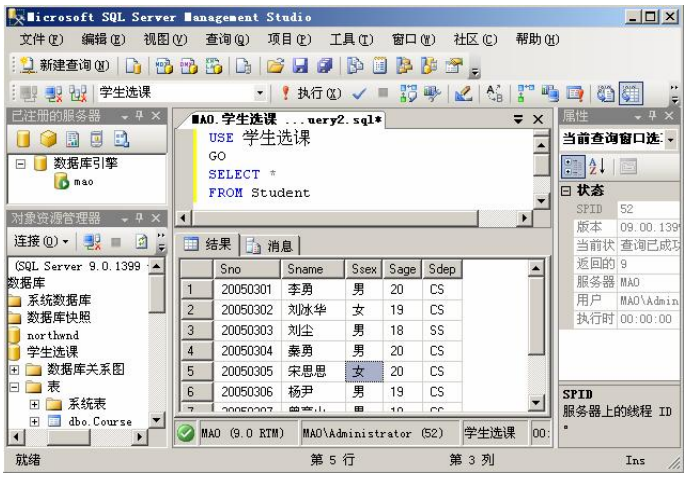


图 5.1　简单的 SELECT 查询语句示例

如果有 ORDER 子句，则结果还要按照< 列名 2 >的值进行升序或降序排列后再输出。对于每一个子句的使用，将在后面的章节中详细介绍。

5.2　SELECT 子句查询

在很多情况下，用户可能只对表中一部分属性列的值感兴趣，这时可以在 SELECT 子句的<目标列表达式>中指定要查询的属性列。在图 5.2 的示例中，显示了查询全体学生的学号和姓名的过程。

图 5.2　查询指定数据列示例

另外，SELECT 子句的<目标列表达式>不仅可以是表中的属性列，也可以是表达式，或者说，可以是经过计算的值。可以在 SELECT 关键字后面的列表中使用运算符和函数，包括算术运算符、数学函数、字符串函数、日期和时间函数以及系统函数等。

算术运算符（包括+、−、*、/ 和%）可以用在各种数值列上，数值列的数据类型可以是

int、smallint、tinyint、float、real、money 或 smallmoney。

【例 5.1】查询全体学生的姓名及其出生年份。

```
SELECT Sname, 2007-Sage
FROM Student
```

查询结果如图 5.3 所示。

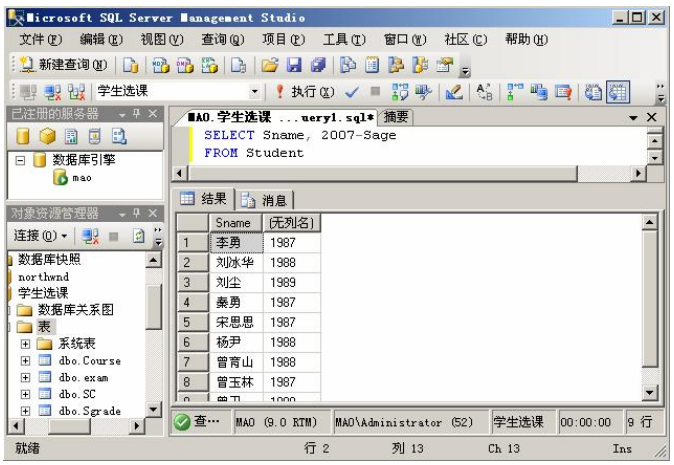


图 5.3　查询计算列的值

从图 5.3 中可以看出，经计算得到的列是没有列名的（显示为“无列名”）。可以通过指定别名定义查询列的列标题。如可以将例 5.1 中的查询改为：

```
SELECT Sname NAME, 2007-Sage BIRTHDAY
FROM Student
```

这里分别为“Sname”列和“2007-Sage”列定义了别名“NAME”和“BIRTHDAY”。语句执行后，在结果集的列标题上显示的就是“NAME”和“BIRTHDAY”了。关于别名查询在 5.9 节还会有更详细的介绍。

数学函数返回参加运算的数据的数值，如下面的检索语句，分别使用了求圆周率的 PI 函数、求正弦值的 SIN 函数、求指数的 EXP 函数、求幂值的 POWER 函数等。

```
SELECT PI(), SIN(PI()/2), COS(PI()/4), EXP(10), POWER(10,2)
```

更多函数的使用请参照联机丛书。

在 SELECT 子句中，可以通过使用 ALL 或 DISTINCT 关键字来控制查询结果集的显示。ALL 关键字表示检索所有的数据，包括重复的数据行，DISTINCT 关键字表示仅仅显示不重复的数据行，对于重复的数据行，则只显示一次。默认使用 ALL 关键字。

【例 5.2】查询选修了课程的学生学号。

```
SELECT Sno
FROM SC
```

执行上面的 SQL 语句，结果如图 5.4 所示。由图中可以看到，结果集中包含了许多重复的行。这时因为默认使用了 ALL 关键字。如果想去掉重复行，可以指定 DISTINCT 关键字，此时执行的结果如图 5.5 所示。

```
SELECT DISTINCT Sno
FROM SC
```

结果　消息

	Sno
1	20050301
2	20050303
3	20050302
4	20050303
5	20050306
6	20050307
7	20050304
8	20050306
9	20060308

图 5.4　使用了 ALL 的查询

结果　消息

	Sno
1	20050301
2	20050302
3	20050303
4	20050304
5	20050306
6	20050307
7	20060308

图 5.5　使用了 DISTINCT 的查询

5.3　条 件 查 询

本节主要介绍包含 WHERE 子句的简单查询，WHERE 子句指定要搜索的数据行的条件，也就是说，只有满足 WHERE 子句条件的数据行才会出现在结果集中。根据 WHERE 子句中条件表达式的不同，该类查询又可分为确定查询、模糊查询和带查找范围的查询。

5.3.1　确定查询

在 WHERE 子句中，确定查询指的是使用比较运算符、列表、合并以及取反等运算方式进行的条件查询。

比较运算符是搜索条件中最常用的。用于比较大小的运算符一般包括：

=（等于），>（大于），<（小于），>=（大于等于），<=（小于等于），!=或<>（不等于）。

【例 5.3】查询“Student”表中所有年龄大于 19 岁的学生信息。

```
SELECT * FROM Student
WHERE Sage > 19
```

或者

```
SELECT * FROM Student
WHERE NOT Sage <= 19
```

查询结果如图 5.6 所示。

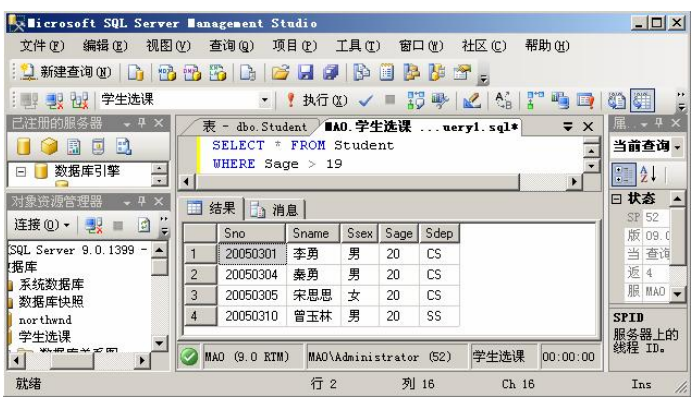


图 5.6　使用“>”运算符查询

【例 5.4】查询所有计算机学院（“CS”）的学生姓名和学号。

```
SELECT Sno, Sname
FROM Student
WHERE Sdep = 'CS'
```

如果想查询所有非计算机学院的学生名单，则可以使用：

```
SELECT Sno, Sname
FROM Student
WHERE Sdep <> 'CS'
```

在 WHERE 子句中，还可以使用逻辑运算符把若干个查询条件合并起来，组成较复杂的查询条件。这些逻辑运算符包括 AND、OR 和 NOT。

AND 运算符表示只有在所有的条件都为真时，才返回真。OR 运算符表示只要有一个条件为真，就可以返回真。NOT 运算符表示取反。当一个 WHERE 子句同时包含多个逻辑运算符时，其优先级从高到低依次为 NOT、AND、OR。用户也可以用括号改变优先级。

【例 5.5】查询“Student”表中所有男生或者年龄大于 19 岁的学生姓名和年龄。

```
SELECT Sname, Sage
FROM Student
WHERE Ssex = '男' OR Sage > 19
```

查询结果如图 5.7 所示。

图 5.7　带有逻辑运算符的查询

5.3.2　模糊查询

通常在查询字符数据时，提供的查询条件并不是十分精确，如查询条件仅仅是包含或类似某种样式的字符。这种查询称为模糊查询，在 WHERE 子句中，可以使用 LIKE 关键字实现这种灵活的查询。

LIKE 关键字用于搜索与特定字符串匹配的字符数据。LIKE 关键字后面可以跟一个列值的一部分而不是完整的列值。其基本语法形式为：

[NOT] LIKE '匹配字符串' [ESCAPE '<转换字符>']

其中，方括号中的内容是可选的，例如如果 LIKE 关键字前面有 NOT 关键字，表示该条

件取反。ESCAPE 子句用于指定转义字符。匹配字符串可以是一个完整的字符串，也可以包含通配符%、_、[]、[^]，这 4 种通配符的含义如表 5.1 所示。

表 5.1　　LIKE 子句中的通配符

通　配　符	含　　义
%	代表任意长度（长度可以为 0）的字符串
_	代表任意单个字符
[]	指定范围或集合中的任意单个字符
[^]	不在指定范围内或集合中的任意单个字符

需要强调的是，带有通配符的字符串必须使用单引号引起来。下面是一些带有通配符的示例。

- LIKE 'AB%'：返回以“AB”开始的任意字符串。
- LIKE '%ABC'：返回以“ABC”结束的任意字符串。
- LIKE '%ABC%'：返回包含“ABC”的任意字符串。
- LIKE '_AB'：返回以“AB”结束的 3 个字符的字符串。
- LIKE '[ACE]%'：返回以“A”、“C”或“E”开始的任意字符串。
- LIKE '[A-Z]ing'：返回 4 个字符长的字符串，结尾是“ing”，第 1 个字符的范围是从 A～Z。
- LIKE 'L[^a]%'：返回以“L”开始、第 2 个字符不是“a”的任意字符串。

下面通过例子来说明 LIKE 子句的使用。

【例 5.6】查询所有姓刘的学生的姓名、学号和性别。

```
SELECT Sno, Sname, Ssex
FROM Student
WHERE Sname LIKE '刘%'
```

查询结果如图 5.8 所示。

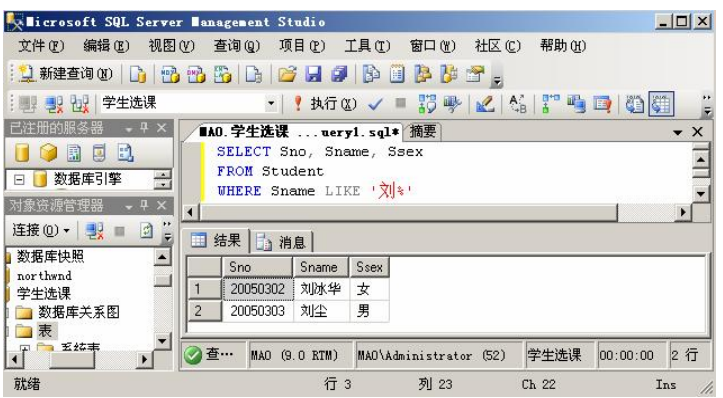


图 5.8　含通配符“%”的查询

【例 5.7】查询所有不姓刘的学生姓名和学号。

```
SELECT Sno, Sname
FROM Student
WHERE Sname NOT LIKE '刘%'
```

【例 5.8】查询姓名中第二个字为“勇”字的学生姓名和学号。

```
SELECT Sno, Sname
FROM Student
WHERE Sname LIKE '_勇%'
```

查询结果如图 5.9 所示。

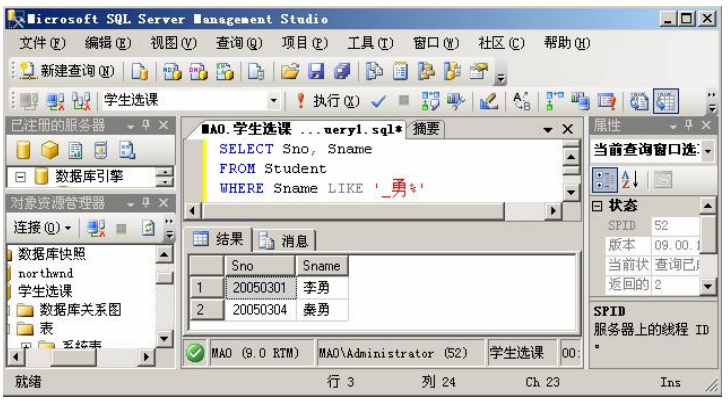


图 5.9　含通配符“_”和“%”的查询

如果用户要查询的字符串本身就含有通配符，这时就要使用 ESCAPE 关键字，对通配符进行转义。

【例 5.9】查询“DB_Design”课程的课程号和学分。

```
SELECT Cno, Ccredit
FROM Course
WHERE Cname LIKE 'DB\_Design' ESCAPE '\'
```

ESCAPE '\'表示“\”为转义字符，这样匹配字符串中紧跟在“\”后面的字符“_”就不再具有通配符的含义，而是转义为普通的“_”字符处理。

5.3.3　带查找范围的查询

谓词 BETWEEN…AND 和 NOT BETWEEN…AND 可以用来查找属性值在或不在指定范围内的元组。其中，BETWEEN 后是范围的下限，AND 后是范围的上限。

【例 5.10】查询年龄在 19～22 岁之间的学生的姓名、年龄和所属院系。

```
SELECT Sname, Sage, Sdep
FROM Student
WHERE Sage BETWEEN 19 AND 22
```

查询结果如图 5.10 所示。

与 BETWEEN…AND 相对的谓词是 NOT BETWEEN…AND

【例 5.11】查询年龄不在 19～22 岁之间的学生的姓名、年龄和所属院系。

```
SELECT Sname, Sage, Sdep
FROM Student
WHERE Sage NOT BETWEEN 19 AND 22
```

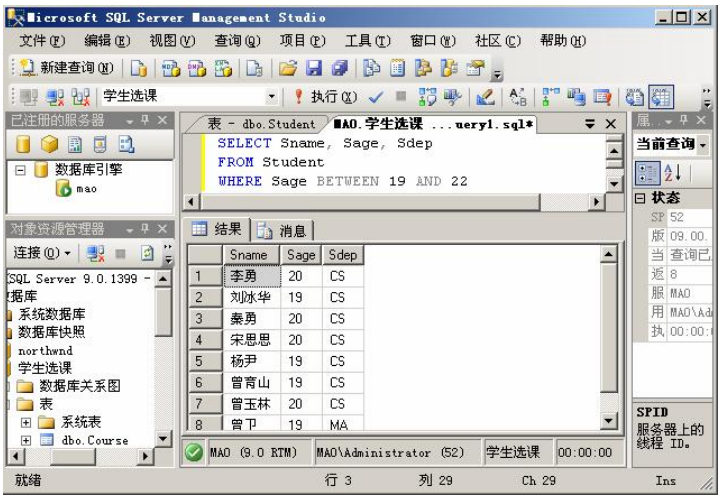


图 5.10　使用 BETWEEN…AND 进行某个范围内的查询

5.4　嵌 套 查 询

在 SQL 语言中，一个 SELECT…FROM…WHERE 语句称为一个查询块。将一个查询块嵌套在另一个查询块的 WHERE 子句或 HAVING 短语的条件中的查询称为嵌套查询。

例如：

```
SELECT Sname
FROM Student
WHERE Sno IN
            ( SELECT Sno
              FROM SC
              WHERE Cno = '2' ) ;
```

在本例中，下层查询块“SELECT Sno FROM SC WHERE Cno = '2'”是嵌套在上层查询块“SELECT Sname FROM Student WHERE Sno IN”的 WHERE 子句中的。上层查询块称为外层查询或者父查询，下层查询块称为内层查询或子查询。

当查询语句比较复杂，不容易理解，或者一个查询依赖于另外一个查询结果时，就可以使用子查询。

SQL 语言允许多层嵌套查询，即一个子查询中还可以嵌套其他子查询。在使用子查询时，需要注意以下几点。

- 子查询必须使用圆括号括起来。
- 子查询中不能单独使用 ORDER BY 子句，如果子查询中使用了 ORDER BY 子句，则 ORDER BY 子句必须与 TOP 子句同时出现。
- 嵌套查询一般的求解方法是由里向外，即每个子查询要在上一级查询处理之前求解，子查询的结果用于建立其父查询要使用的查找条件。

5.4.1 带 IN 的嵌套查询

在嵌套查询中，子查询的结果往往是一个集合，所以谓词 IN 是嵌套查询中最常使用的谓词。其主要使用方式为：

```
WHERE < 条件表达式 >
IN（子查询）
```

【例 5.12】查询与“刘尘”在同一个院系学习的学生信息。

先分步完成此查询，然后再构造嵌套查询。

（1）确定“刘尘”所在的系名。

```
SELECT Sdep
FROM Student
WHERE Sname='刘尘'
```

查找结果为 SS，如图 5.11 所示。

（2）查找所有在 SS 系学习的学生学号、姓名、系别信息。查找结果如图 5.12 所示。

图 5.11　查询刘尘所在系名称

图 5.12　查询在 SS 系学习的学生信息

```
SELECT Sno, Sname, Sdep
FROM Student
WHERE Sdep='SS'
```

将第 1 步查询嵌入到第 2 步查询中，构造出嵌套查询：

```
SELECT Sno, Sname, Sdep
FROM Student
WHERE Sdep IN
            ( SELECT Sdep
              FROM Student
              WHERE Sname='刘尘' )
```

【例 5.13】查询选修了课程名为“数据库原理”的学生学号和姓名。

本查询涉及学号、姓名和课程名 3 个属性。学号和姓名存放在“Student”表中，课程名存放在“Course”表中，这两个表之间没有直接的联系，所以需要通过“SC”表建立二者之间的联系。因此本查询实际上涉及 3 个关系。

```
SELECT Sno, Sname          --③最后在“Student”表中取出相应学
FROM Student               --生的“Sno”和“Sname”
```

```
WHERE Sno IN
      ( SELECT Sno                  --②然后在“SC”表中找出选修了 2 号
        FROM SC                     --课程的学生学号
        WHERE Cno IN
              ( SELECT Cno          --①首先在“Course”表中找出“数据
                FROM Course         --库原理”的课程号，结果为 2
                WHERE Cname='数据库原理' ) )
```

查询结果如图 5.13 所示。

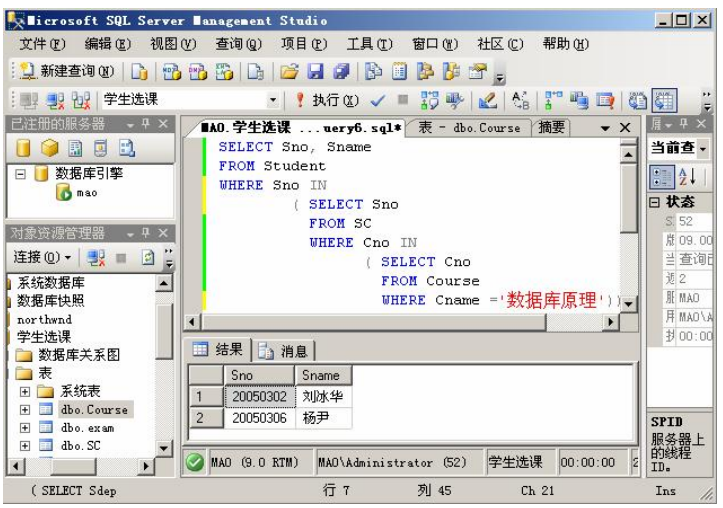


图 5.13　查询选修了“数据库原理”课程的学生信息

由例 5.12 和例 5.13 可以看出，当查询涉及多个关系（表）时，用嵌套查询逐步求解，层次清楚，易于构造，具有结构化程序设计的特点。

5.4.2　带比较运算符的嵌套查询

带有比较运算符的子查询是指父查询与子查询之间用比较运算符进行连接。当用户能确切知道内层查询返回的是单值时，可以用=、>、<、>=、<=、!=或<>等比较运算符。

例如，在例 5.12 中，由于一个学生只可能在一个系学习，也就是说子查询的结果是一个值，因此可以用“=”代替“IN”，其 SQL 语句如下：

```
SELECT Sno, Sname, Sdep
FROM Student
WHERE Sdep =
            ( SELECT Sdep
              FROM Student
              WHERE Sname='刘尘' )
```

需要注意的是，子查询一定要跟在比较运算符之后，下列写法是错误的：

```
SELECT Sno, Sname, Sdep
FROM Student
```

```
WHERE ( SELECT Sdep
        FROM Student
        WHERE Sname= '刘尘' ) = Sdep
```

例 5.12 和例 5.13 中的各个子查询都只执行一次，其查询结果用于父查询。子查询的条件不依赖于父查询，这类子查询称为不相关子查询。不相关子查询是较简单的一类子查询。如果子查询的查询条件依赖于父查询，这类子查询称为相关子查询。例 5.14 就是一个相关子查询的例子。

【例 5.14】找出每个学生超过他选修课程平均成绩的课程号。

```
SELECT Sno, Cno
FROM SC x
WHERE Grade > =   ( SELECT AVG (Grade)
                    FROM SC y
                    WHERE y.Sno = x.Sno )
```

“x”是表“SC”的别名，又称为元组变量，可以用来表示“SC”的一个元组。内层查询是求一个学生所有选修课程平均成绩的，至于是哪个学生的平均成绩要看参数“x.Sno”的值，而该值是与父查询相关的，因此这类查询称为相关子查询。

5.4.3 带 ANY 或 ALL 的嵌套查询

子查询返回单值时，可以用比较运算符，但返回多值时，要用 ANY 或 ALL 谓词修饰符。而使用 ANY 或 ALL 谓词时，必须同时使用比较运算符。其语义见表 5.2。

表 5.2　　使用 ANY 或 ALL 带比较运算符时的语义

谓词运算符组合	语　义
>ANY	大于子查询结果中的某个值
>ALL	大于子查询结果中的所有值
<ANY	小于子查询结果中的某个值
<ALL	小于子查询结果中的所有值
>=ANY	大于等于子查询结果中的某个值
>=ALL	大于等于子查询结果中的所有值
<=ANY	小于等于子查询结果中的某个值
<=ALL	小于等于子查询结果中的所有值
=ANY	等于子查询结果中的某个值
=ALL	等于子查询结果中的所有值（通常没有实际意义）
!=(或< >)ANY	不等于子查询结果中的某个值
!= (或< >)ALL	不等于子查询结果中的任何一个值

【例 5.15】查询其他系中比计算机系某一学生年龄小的学生姓名、年龄和所属院系。

```
SELECT Sname, Sage, Sdep
FROM Student
WHERE Sage < ANY ( SELECT Sage
                   FROM Student
```

```
                    WHERE Sdep = 'CS' )
AND Sdep != 'CS'
```

查询结果如图 5.14 所示。

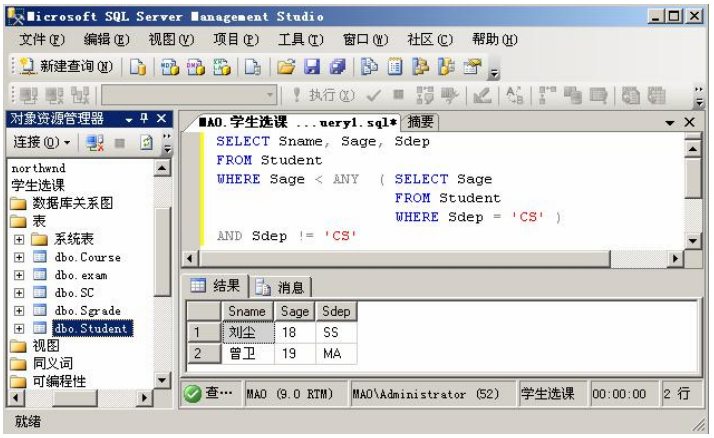


图 5.14　其他系比计算机系某一学生年龄小的学生信息

系统执行此查询时，首先处理子查询，找出计算机系中所有学生的年龄，构成一个集合（20，19）。然后处理父查询，找出所有非计算机系且年龄小于 20 或者 19 的学生。

本查询也可以用聚集函数来实现。首先用子查询找出计算机系学生的最大年龄（20），然后在父查询中查询所有非计算机系且年龄小于 20 的学生。SQL 语句如下：

```
SELECT Sname, Sage, Sdep
FROM Student
WHERE Sage < ( SELECT MAX(Sage)
               FROM Student
               WHERE Sdep = 'CS' )
AND Sdep != 'CS'
```

【例 5.16】查询其他系中比计算机系所有学生年龄都小的学生姓名、年龄和所属院系。

```
SELECT Sname, Sage, Sdep
FROM Student
WHERE Sage < ALL ( SELECT Sage
                   FROM Student
                   WHERE Sdep = 'CS' )
AND Sdep != 'CS'
```

本查询同样也可以用聚集函数来实现，SQL 语句如下：

```
SELECT Sname, Sage, Sdep
FROM Student
WHERE Sage < ( SELECT MIN(Sage)
               FROM Student
               WHERE Sdep = 'CS' )
AND Sdep != 'CS'
```

5.4.4 带 EXISTS 的嵌套查询

EXISTS 代表存在。带有 EXISTS 谓词的子查询不返回任何数据，只产生逻辑真值“True”或逻辑假值“False”。

【例 5.17】查询所有选修了 1 号课程的学生姓名。

本查询涉及“Student”和“SC”关系。可以在“Student”表中依次取每个元组的“Sno”值，再用这个值去检查“SC”关系。若“SC”中存在这样的元组，其“Sno”值等于此“Student.Sno”值，并且其课程号“Cno”为 1，则取“Student.Sname”送入结果集中。SQL 语句如下：

```
SELECT Sname
FROM Student
WHERE EXISTS
            ( SELECT *
              FROM SC
              WHERE Sno = Student.Sno AND Cno = '1' )
```

查询结果如图 5.15 所示。

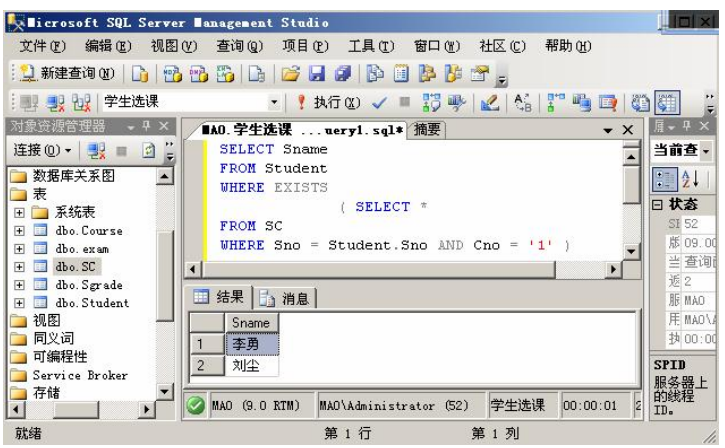


图 5.15　选修了 1 号课程的学生姓名信息

使用存在量词 EXISTS 后，若内层查询结果非空，则外层的 WHERE 子句返回真值，否则，返回假值。

由 5.4 节前面的几个例子可以看到，只有当使用了 EXISTS 关键字时，才在子查询的列表达式中使用星号（*）代替所有的列名。这是因为当使用 EXISTS 关键字时，子查询不返回数据，而是判断子查询是否存在数据，这时给出列名没有实际意义。

例 5.17 中子查询的查询条件依赖于外层父查询的“Sno”属性值，因此是相关子查询。求解相关子查询不能像求解不相关子查询那样，一次将子查询求解出来，然后求解父查询。由于内层查询与外层查询相关，因此必须反复求值。从概念上讲，相关子查询的一般处理过程如下。

首先取外层查询表（Student）中的第一个元组，根据它与内层查询相关的属性值（Sno 值）处理内层查询，如果 WHERE 子句返回值为真，则取外层查询中该元组放入结果表；然后再取外层查询表（Student）中的下一个元组；重复这一过程，直到外层查询表全部检

查完为止。

与 EXISTS 谓词相对应的是 NOT EXISTS 谓词。使用存在谓词 NOT EXISTS 后，若内层查询结果为空，则外层的 WHERE 子句返回真值，否则返回假值。

【例 5.18】查询没有选修 1 号课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE NOT EXISTS
            ( SELECT *
              FROM SC
              WHERE Sno = Student.Sno AND Cno = '1' )
```

【例 5.19】查询选修所有课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE NOT EXISTS
      (SELECT *
       FROM Course
         WHERE NOT EXISTS
              ( SELECT *
                FROM SC
                WHERE Sno = Student.Sno AND Cno = Course.Cno)
```

一些带有 EXISTS 或 NOT EXISTS 谓词的子查询不能被其他形式的子查询等价替换，但所有带 IN 谓词、比较运算符、ANY 和 ALL 谓词的子查询都能用 EXISTS 谓词的子查询等价替换。例如，带有 IN 谓词的例 5.12 可以用如下带 EXISTS 谓词的子查询替换：

```
SELECT Sno, Sname, Sdep
FROM Student S1
WHERE EXISTS
            ( SELECT *
              FROM Student S2
              WHERE S2.Sdep = S1.Sdep AND S2.Sname = '刘尘' )
```

由于带 EXISTS 谓词的相关子查询只关心内层查询是否有返回值，并不需要查具体值，因此其效率并不一定低于不相关子查询，有时效率更高。

5.5　集 合 查 询

SELECT 查询语句的结果集往往是一个包含了多行数据（元组）的集合。在数学领域中，集合之间可以进行并、交、差等运算。在 Microsoft SQL Server 2005 中，两个查询语句之间也可以进行集合运算，主要包括并操作 UNION、交操作 INTERSECT 和差操作 EXCEPT。需要注意的是，在进行集合运算时，所有查询语句中的列的数量和顺序必须相同，且数据类型

必须兼容。

下面通过几个示例来讲述如何执行集合运算。

5.5.1 并操作

UNION 运算符表示并集运算，结果集中包含了执行并操作后得到的所有数据。

【例 5.20】查询选修了课程 1 或者选修了课程 2 的学生学号。

本查询实际上就是查询选修了课程 1 的学生集合与选修了课程 2 的学生集合的并集。

```
SELECT Sno
FROM SC
WHERE Cno = '1'
UNION ALL
SELECT Sno
FROM SC
WHERE Cno = '2'
```

查询结果如图 5.16 所示。在图中可以看到，结果集中包含了重复数据，这是由于查询语句中使用了 ALL 关键字的缘故。如果没有使用 ALL 关键字，则结果集中不会出现重复值。

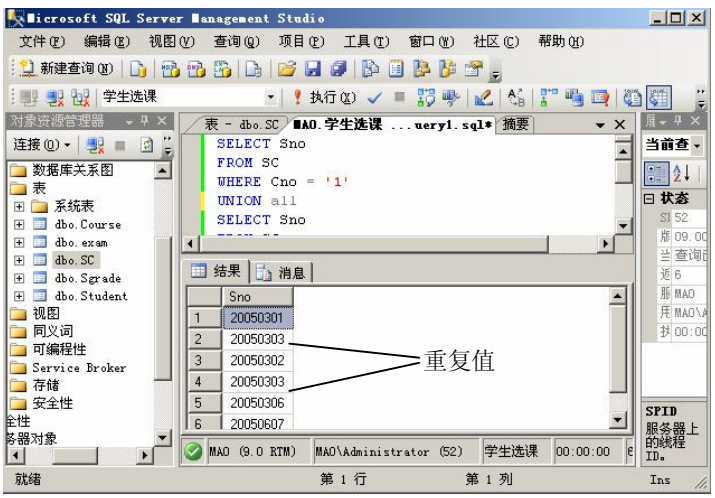


图 5.16　使用 UNION 运算符示例

5.5.2 交操作

【例 5.21】查询既选修了课程 1 又选修了课程 2 的学生学号。

本查询实际上就是查询选修了课程 1 的学生集合与选修了课程 2 的学生集合的交集。SQL 语句如下：

```
SELECT Sno
FROM SC
WHERE Cno = '1'
INTERSECT
SELECT Sno
```

```
FROM SC
WHERE Cno = '2'
```

如果要查询既选修了课程 1 又选修了课程 2 的学生学号和姓名信息的话，又该如何实现呢？留给读者自己思考。

5.5.3　差操作

【例 5.22】查询计算机系的学生与年龄不大于 19 岁的学生的差集。

```
SELECT *
FROM Student
WHERE Sdep = 'CS'
EXCEPT
SELECT *
FROM Student
WHERE Sage <= 19
```

先查询得到计算机系所有学生信息的结果集，再从中减去（差）所有年龄不大于 19 岁的学生信息集合，即是计算机系的学生与年龄不大于 19 岁的学生的差集。

本例等价于查询计算机系中年龄大于 19 岁的学生信息，也可以用下面的 SQL 语句实现：

```
SELECT *
FROM Student
WHERE Sdep = 'CS' AND Sage > 19
```

5.6　连 接 查 询

在设计表时，为了提高表的设计质量，经常把相关的数据分散在不同的表中。但是，在实际使用时，往往需要同时从两个或两个以上表中检索数据，并且每一个表中的数据仍以单独的列出现在结果集中。实现从两个或两个以上表中检索数据且结果集中出现的列来自于多个表的检索操作称为连接技术，或者说，连接技术是指对两个或两个以上表中数据执行乘积运算的技术。

连接查询是关系数据库中最主要的查询，包括交叉连接、内连接、外连接 3 种。

连接运算与集合运算是不同的。在集合运算的结果集中，列的数量不发生变化，只是行的数量可能变化。但是在连接运算的结果集中，除了行的数量可能发生变化以外，列的数量也经常会变化。

连接可以在 SELECT 语句的 FROM 子句或 WHERE 子句中建立，在 FROM 子句中建立连接有助于将连接操作与 WHERE 子句中的搜索条件区分开。所以，在 Transact-SQL 中推荐使用这种方法。

在 FROM 子句中指定连接条件的语法格式为：

```
SELECT < 目标列表达式 >
FROM < 表 1 > 连接类型 < 表 2 >
[ ON (连接条件) ]
```

其中连接类型可以是交叉连接（CROSS JOIN）、内连接（INNER JOIN）、外连接；ON子句指出连接条件，它由被连接表中的列和比较运算符、逻辑运算符等构成。

在 WHERE 子句中指定连接条件的基本格式为：

SELECT < 目标列表达式 >

FROM < 表 1 > , < 表 2 >

[WHERE (连接条件)]

连接可以对同一个表操作，也可以对多表操作，对同一个表操作的连接又称做自连接。

5.6.1 交叉连接查询

交叉连接也称为笛卡儿乘积，它返回两个表中所有数据行的全部组合，即交叉连接结果集中的数据行数等于两个表的数据行数的乘积。

交叉连接使用关键字 CROSS JOIN 来创建，并且不带 WHERE 子句。例如，对“Student”表和“Course”表执行交叉连接，如图 5.17 所示，由于“Student”表中有 9 行数据，Course 表中有 5 行数据，因此结果集中包含了 45 行数据。

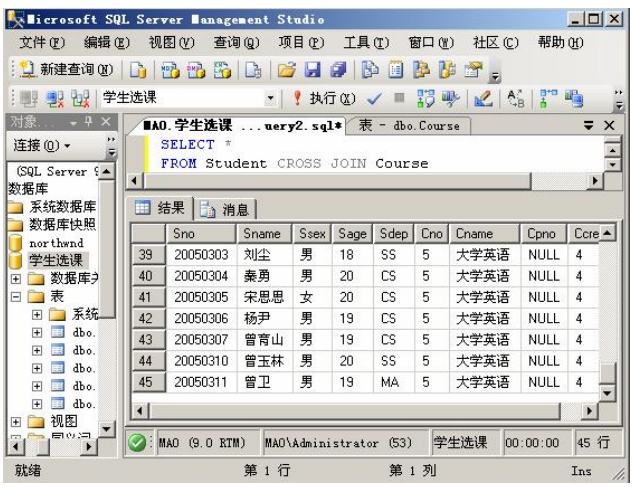


图 5.17 交叉连接查询

在实际的应用中，交叉连接的使用是比较少的，但是它是理解外连接和内连接的基础。

5.6.2 内连接查询

内连接（INNER JOIN）使用比较运算符进行表间某（些）列数据的比较操作，并列出表中与连接条件相匹配的数据行。根据所使用的比较方式不同，内连接又分为等值连接、不等值连接和自然连接 3 种。

1. 等值与非等值连接查询

连接查询中用来连接两个表的条件称为连接条件或连接谓词，它的一般格式为：

表名 1.列名 1 比较运算符 表名 2.列名 2

可以使用的比较运算符有：>、>、=、<、<=、!=（或<>），还可以使用 BETWEEN…AND 之类的谓词。当连接运算符为等号（=）时，称为等值连接，而使用其他比较运算符则称为非等值连接。

【例 5.23】查询每个学生选修课程的情况。

学生情况存放在“Student”表中，学生选课情况存放在“SC”表中，所以本查询实际上涉及“Student”与“SC”两个表。这两个表之间的联系是通过公共属性“Sno”实现的。

方法 1：在 FROM 子句中指定连接。

```
SELECT Student.*, SC.*
FROM Student INNER JOIN SC
ON Student.Sno = SC.Sno
```

方法 2：在 WHERE 子句中指定连接。

```
SELECT Student.*, SC.*
FROM Student, SC
WHERE Student.Sno = SC.Sno
```

查询结果如图 5.18 所示。

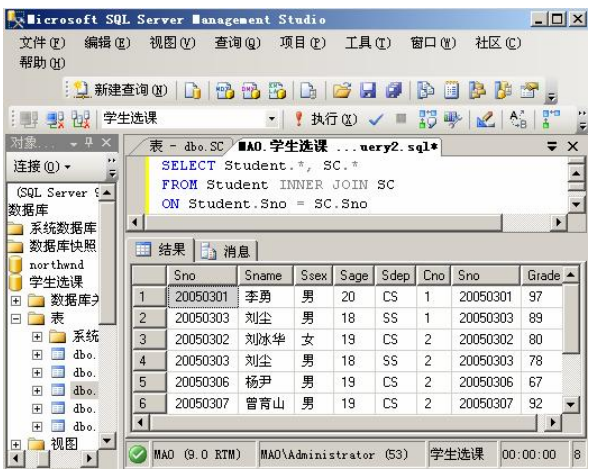


图 5.18　等值连接查询

本例中，SELECT 子句、ON 子句与 WHERE 子句中的属性名前都加上了表名前缀，这是为了避免混淆，因为在“Student”表和“SC”表中都有属性“Sno”。如果属性名在参加连接的各表中是唯一的，则可以省略表名前缀。

SQL Server 2005 系统执行该连接操作的一种可能的过程如下。

- 首先，在“Student”表中找到第一个元组，然后从头开始扫描“SC”表，逐一查找与“Student”第一个元组的“Sno”相等的“SC”元组；
- 找到后就将“Student”中的第一个元组与该元组拼接起来，形成结果集中的一个元组；
- “SC”全部查找完，再找“Student”中的第 2 个元组，然后再从头开始扫描“SC”表，逐一查找满足连接条件的元组，找到后就将“Student”中的第 2 个元组与该元组拼接起来，形成结果集中的一个元组；
- 重复以上操作，直到“Student”表中的全部元组都处理完毕。

一般情况下，不等值连接没有多大意义，不单独使用，通常和等值连接一起组成复合条件，共同完成一组查询。

【例 5.24】查询每个学生选修课程成绩大于 80 分的情况。

学生情况存放在“Student”表中，学生选课情况存放在“SC”表中，所以本查询实际上涉及“Student”与“SC”两个表。这两个表之间的联系是通过公共属性“Sno”实现的。

方法 1：在 FROM 子句中指定连接

```
SELECT Student.*, SC.*
FROM Student INNER JOIN SC
ON Student.Sno = SC.Sno AND SC.Grade >80
```

方法 2：在 WHERE 子句中指定连接

```
SELECT Student.*, SC.*
FROM Student, SC
WHERE Student.Sno = SC.Sno AND SC.Grade >80
```

查询结果如图 5.19 所示。

图 5.19　不等值连接查询

2. 自然连接查询

从图 5.18、图 5.19 所示的结果可以看出，对两个表进行等值、不等值连接查询后，在结果集中出现了重复列“Sno”。如果在进行等值连接时目标列不使用“*”而使用选择列，从而把结果集中重复的属性列去掉，就构成了自然连接。

例如，在例 5.24 中，查询每个学生选修课程的情况，并去掉重复列。可以使用下面的 SQL 语句：

```
SELECT Student.Sno, Sname, Sage, Sdep, Cno, Grade
FROM Student INNER JOIN SC
ON Student.Sno = SC.Sno
```

同样，也可以在 WHERE 子句中指定连接条件进行查询：

```
SELECT Student.Sno, Sname, Sage, Sdep, Cno, Grade
FROM Student, SC
WHERE Student.Sno = SC.Sno
```

查询结果如图 5.20 所示。

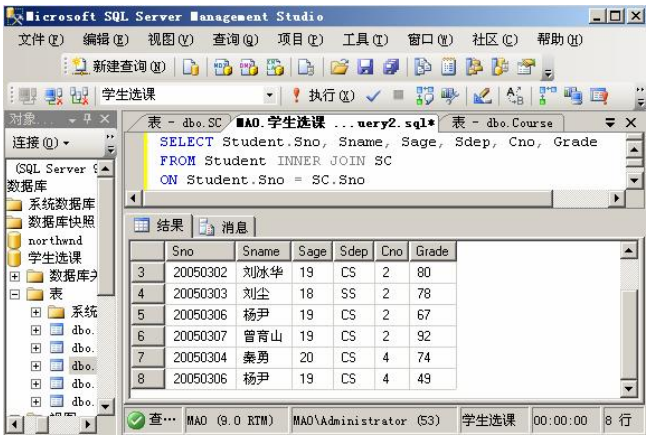


图 5.20　自然连接查询

3. 自连接查询

连接不仅可以在表之间进行，也可以使一个表同其自身进行连接，这种连接称为自连接（SELF JOIN），相应的查询称为自连接查询。

【例 5.25】查找每一门课程的间接先修课（即先修课的先修课）。

要查询每一门课程的间接先修课，在“课程”表“Course”关系中，只有每门课的直接先修课信息，而没有先修课的先修课。要得到这个信息，必须先对一门课程找到其先修课，再按此先修课的课程号，查找它的先修课程，这相当于将“Course”表与其自身连接后，取第一个副本的课程号与第二个副本的先修课号作为目标列中的属性。具体写 SQL 语句时，为清楚起见，可以为“Course”表取两个别名，一个是“FIRST”，另一个是“SECOND”，也可以在考虑问题时就把“Course”表想成是两个完全一样的表，一个是“FIRST”表，另一个是“SECOND”表。完成该查询的 SQL 语句为：

```
SELECT FIRST.Cno, SECOND. Cpno
FROM Course FIRST INNER JOIN Course SECOND
ON FIRST.Cpno = SECOND.Cno
```

查询结果如图 5.21 所示。

图 5.21　自连接查询

该语句也可以写成：

```
SELECT FIRST.Cno, SECOND. Cpno
FROM Course FIRST, Course SECOND
WHERE FIRST.Cpno = SECOND.Cno
```

5.6.3 外连接查询

在内连接操作中，只有满足连接条件的元组才能作为结果输出，如在例 5.21 的结果集中没有关于 20050305、20050310、20050311 这几个学生的信息，原因在于他们没有选课，在“SC”表中没有相应的元组。但是有时需要以“Student”表为主体列出每个学生的基本情况及其选课情况，若某个学生没有选课，则只输出其基本情况信息，其选课信息为空值即可，这时可以使用外连接（OUTER JOIN）。

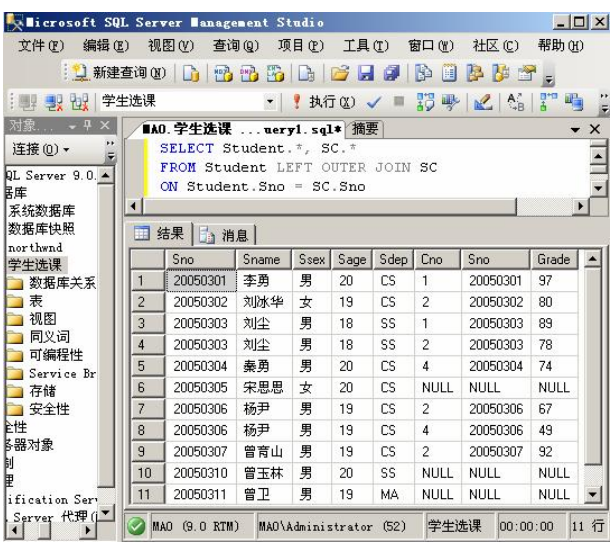
在 Microsoft SQL Server 2005 系统中，可以使用 3 种外连接关键字，即 LEFT OUTER JOIN，RIGHT OUTER JOIN 和 FULL OUTER JOIN。LEFT OUTER JOIN 表示左外连接，结果集中将包含满足搜索条件的所有数据和第一个连接表中不满足条件的数据（结果集中对应第二个表中的数据为 NULL）。RIGHT OUTER JOIN 表示右外连接，结果集中将包含满足搜索条件的所有数据和第二个连接表中不满足条件的数据（结果集中对应第一个表中的数据为 NULL）。FULL OUTER JOIN 表示全外连接，它综合了左外连接和右外连接的特点，两个表中不满足条件的数据都出现在结果集中，结果集中这些数据在另外一个表中的对应值是 NULL。

【例 5.26】查询所有学生选修课程的情况，包括没有选修课程的学生。

本例和例 5.23 不同的地方就在于，例 5.23 只需输出有选修课程的学生信息，而本例却必须输出全部学生信息。因此，必须使用外连接才能实现本例查询。SQL 语句如下：

```
SELECT Student.*, SC.*
FROM Student LEFT OUTER JOIN SC
ON Student.Sno = SC.Sno
```

在该查询语句中，使用了左外连接。所以“Student”表中的数据将全部输出，而表中不满足查询条件的数据记录在对应的“SC”表中都用 NULL 表示，结果如图 5.22 所示。



	Sno	Sname	Ssex	Sage	Sdep	Cno	Sno	Grade
1	20050301	李勇	男	20	CS	1	20050301	97
2	20050302	刘冰华	女	19	CS	2	20050302	80
3	20050303	刘尘	男	18	SS	1	20050303	89
4	20050303	刘尘	男	18	SS	2	20050303	78
5	20050304	秦勇	男	20	CS	4	20050304	74
6	20050305	宋思思	女	20	CS	NULL	NULL	NULL
7	20050306	杨尹	男	19	CS	2	20050306	67
8	20050306	杨尹	男	19	CS	4	20050306	49
9	20050307	曾育山	男	19	CS	2	20050307	92
10	20050310	曾玉林	男	20	SS	NULL	NULL	NULL
11	20050311	曾卫	男	19	MA	NULL	NULL	NULL

图 5.22　左外连接、右外连接查询

本例也可以用右外连接来完成。这时只需要把“Student”表和“SC”表的位置调换一下即可。相应的 SQL 语句如下所示：

```
SELECT Student.*, SC.*
FROM SC RIGHT OUTER JOIN Student
ON Student.Sno = SC.Sno
```

该语句的执行结果同上述左外连接完全一样。

【例 5.27】查询所有学生选修课程的情况，以及所有课程的选课记录。

要求显示学生选修课程的情况和所有的选课记录，可以使用全外连接来实现。SQL 语句如下：

```
SELECT Student.Sno, Sname, Ssex, Sage, Sdep, Cno, Grade
FROM Student FULL OUTER JOIN SC
ON Student.Sno = SC.Sno
```

查询结果如图 5.23 所示。

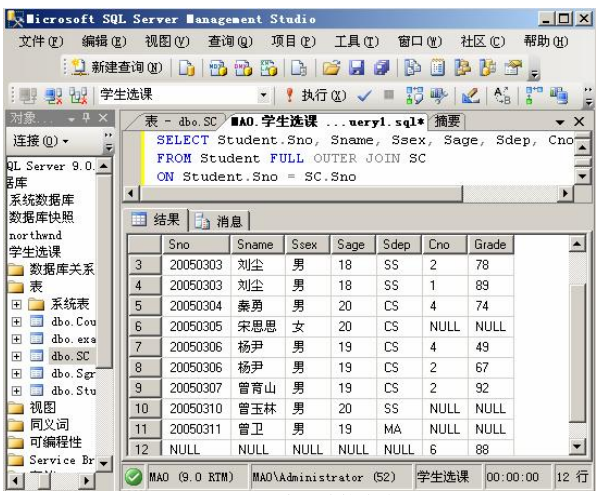


图 5.23　全外连接查询

5.7　排 序 查 询

使用 ORDER BY 子句可以按一个或多个属性列对数据进行排序。ORDER BY 子句通常位于 WHERE 子句后面，默认的排序方式有 2 种，升序和降序，分别使用关键字 ASC 和 DESC 来指定。其中，ASC 表示升序，DESC 表示降序，缺省值为升序。当排序列包含空值（NULL）时，若使用 ASC 关键字，则排序列为空值的元组最后显示；若使用 DESC 关键字，则排序列为空值的元组最先显示。

【例 5.28】查询选修了 2 号课程的学生学号及其成绩，并按分数降序输出结果。

```
SELECT Sno, Grade
FROM SC
WHERE Cno = '2'
ORDER BY Grade DESC
```

当基于多个属性对数据进行排序时，出现在 ORDER BY 子句中的列的顺序是非常重要

的，因为系统是按照排序列的顺序进行排序的。如果第一个属性相同，则依据第二个属性排序，如果第二个属性相同，则依据第三个属性排序，依此类推。另外，在执行多列排序时，每一个列都可以指定是升序还是降序。

【例 5.29】查询全体学生信息，查询结果按所在系的系名升序排列，同一个系中的学生按年龄降序排列。

```
SELECT * FROM Student
ORDER BY Sdep, Sage DESC
```

在该查询中，系统先按照“Sdep”进行升序排序（关键字 ASC 省略），然后对于“Sdep”相同的元组再按照“Sage”进行降序排序，查询结果如图 5.24 所示。

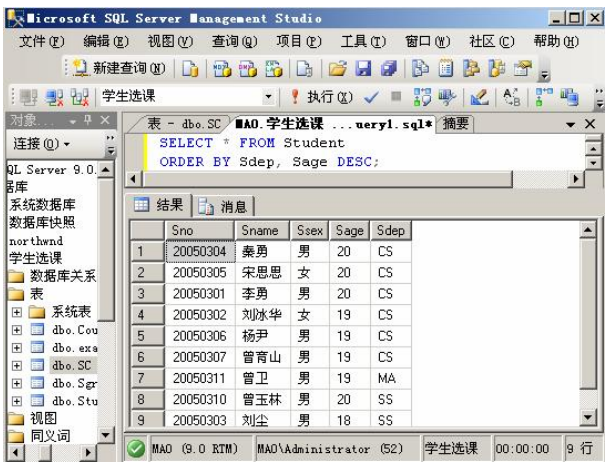


图 5.24　排序查询

5.8　显示部分记录的 TOP 查询

当在查询语句中使用了 ORDER BY 子句时，还经常在 SELECT 子句中使用 TOP 关键字。TOP 关键字表示仅在结果集中从前向后列出指定数量的数据行。如果在使用 TOP 关键字的 SELECT 语句中没有使用排序子句，则只是随机地返回指定数量的数据行。

使用 TOP 关键字的基本语法有两种。

- TOP（n）：从前向后返回 *n* 行数据；
- TOP（n）PERCENT：按照百分比返回指定数量的数据行。

【例 5.30】按序号查询班内前 5 个学生的信息。

```
SELECT TOP (5) *
FROM Student
ORDER BY Sno
```

本查询先将结果集中的数据按照“Sno”升序排序，然后取出前 5 个输出显示。

【例 5.31】查询选课成绩排名在前 30%的学生学号、姓名和成绩。

```
SELECT TOP (30) PERCENT SC.Sno, Sname, Grade
FROM SC, Student
```

```
WHERE SC.Sno = Student.Sno
ORDER BY Grade DESC
```

本查询先通过连接查询得到所有学生的选课记录，然后将得到的记录按照“Grade”降序排序，再从排序后的记录里取前 30%进行输出。输出结果如图 5.25 所示。

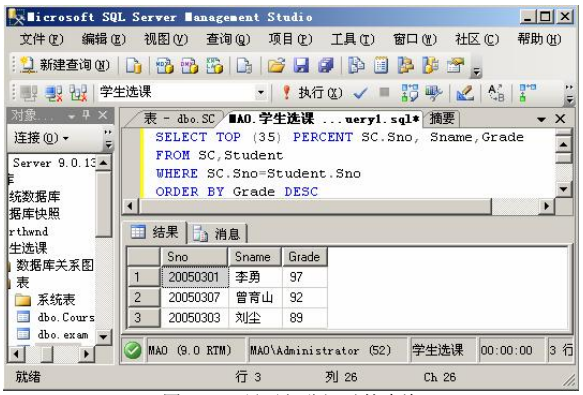


图 5.25　显示部分记录的查询

有的读者或许会问，如果设定列出 5 行数据，但是第 6 行、第 7 行甚至更多行的数据如果和第 5 行一样的话，那么这些行的数据是否会显示？例如在例 5.31 中，结果集列出了 3 行数据，假设第 4 行数据中“Grade”值也是 89 的话，那不是丢失了一部分可以使用的信息吗？要解决这个问题很简单，只需要在 TOP 子句后使用 WITH TIES 子句就可以了。例如在例 5.31 中，如果将 SELECT 语句改为：

```
SELECT TOP (30) PERCENT WITH TIES SC.Sno, Sname, Grade
FROM SC, Student
WHERE SC.Sno=Student.Sno
ORDER BY Grade DESC
```

就能避免出现上述问题。

5.9　统计函数与别名查询

在 SELECT 语句中使用统计函数，可以得到很多有用的信息。在 Microsoft SQL Server 2005 中主要包括 5 类统计函数（又称聚集函数）。

- 计数：

```
COUNT ( [ DISTINCT | ALL ] * )
COUNT ( [ DISTINCT | ALL ] )
```

- 计算总和：

```
SUM ( [ DISTINCT | ALL ] )
```

- 计算平均值：

```
AVG ( [ DISTINCT | ALL ] )
```

- 求最大值：

```
MAX ( [ DISTINCT | ALL ] )
```

- 求最小值：

MIN ([DISTINCT | ALL])

其中，DISTINCT 指明在计算时取消指定列中的重复值，只处理唯一值；而 ALL 短语则指明不取消重复值。缺省情况下为 ALL。

【例 5.32】查询学生总人数。

```
SELECT COUNT (*)
FROM Student
```

【例 5.33】查询选修了课程的学生人数。

由于一个学生可能选择多门课程，所以在计算时要避免一个学生重复计数。可以使用 DISTINCT 关键字对学生学号“Sno”进行限定，保证同一个学号只计数一次。

```
SELECT COUNT ( DISTINCT Sno)
FROM SC
```

【例 5.34】计算选修了 1 号课程的学生的平均成绩。

```
SELECT AVG(Grade)
FROM SC
WHERE Cno = '1'
```

默认情况下，在数据检索结果中显示出来的列标题就是在定义表时使用的列名称，如果要显示的列在查询表中没有定义，如例 5.34 使用统计函数计算得到的列，在显示的时候是没有标题名的（标题上显示“无列名”）。为了方便用户对结果集的理解，可以在检索过程中根据需要改变显示的列标题，实际上就是为指定的列定义一个别名。

定义别名有两种方法：一种是使用等号（=），一种是使用 AS 关键字。

- 使用等号时，其语法形式为：新标题=列名；
- 使用 AS 关键字时，其形式为：列名 [AS] 新标题，AS 关键字可以省略。

这里要注意的是，使用等号和 AS 关键字时，新标题和列名的顺序是不同的。

【例 5.35】查询选修 1 号课程的学生的最高分数。

```
SELECT 选修 1 号课程的最高分数 = MAX(Grade)
FROM SC
WHERE Cno= '1'
```

该语句为结果集中的列定义了一个新标题“选修 1 号课程的最高分数”，结果如图 5.26 所示。

图 5.26　别名查询

本查询还可以写成：

```
SELECT   MAX(Grade)   选修 1 号课程的最高分数
FROM SC
WHERE Cno= '1'
```

如果结果集中列名称很长，或者列名称不具有描述意义，或者只有表达式而没有列名称，

这时使用别名改变列标题可以为用户提供很大的方便。

5.10　分 组 查 询

使用 GROUP BY 子句可以将查询结果按属性进行分组，属性值相等的为一组。这样做的目的是为了细化统计函数的作用对象，如果未对查询结果分组，集函数将作用于整个查询结果；而对查询结果分组后，集函数将分别作用于每个组。

【例 5.36】查询所有的课程号及相应的选课人数。

```
SELECT Cno, COUNT(Sno) AS  选课人数
FROM SC
GROUP BY Cno
```

该语句对查询结果先按“Cno”的值分组，所有具有相同“Cno”值的元组作为一组，然后对每一组使用函数 COUNT 计数，以求得该组的学生人数。

查询结果如图 5.27 所示。

图 5.27　分组查询

如果分组后还要求按一定的条件对这些组进行筛选，最终只输出满足指定条件的组，则可以使用 HAVING 短语指定筛选条件。

【例 5.37】查询选修了 2 门以上课程的学生学号。

```
SELECT Sno
FROM SC
GROUP BY Sno
HAVING COUNT(*) >2
```

这里先用 GROUP BY 子句按“Sno”值进行分组，再用统计函数 COUNT 对每一组计数。HAVING 短语给出了输出结果的条件，只有满足这个条件（即元组个数>2）的组才会输出显示。

在使用 GROUP BY 和 HAVING 子句的过程中，要注意以下几点。

- GROUP BY 子句的作用对象是查询的中间结果集，按照指定的一列或多列值进行分组，值相等的为一组。因此，使用 GROUP BY 子句后，SELECT 子句的列名列表中只能出现分组属性和集函数。
- 因为 HAVING 子句是作为 GROUP BY 子句的条件出现的，所以 HAVING 子句必须与 GROUP BY 子句同时出现，并且必须出现在 GROUP BY 子句之后。
- GROUP BY 子句可以包含表达式。
- 在 HAVING 子句中的列只返回一个值。

【例 5.38】查询有 3 门以上课程是 90 分以上的学生学号及（90 分以上的）课程数。

```
SELECT Sno, COUNT(*)
FROM SC
WHERE Grade > 90
GROUP BY Sno
HAVING COUNT(*) > 3
```

HAVING 短语与 WHERE 子句的区别在于作用对象不同。WHERE 子句作用于基表或视图，从中选择满足条件的元组。HAVING 短语作用于组，从中选择满足条件的组。

以上介绍了使用 SELECT 语句进行数据查询的命令和操作，这些命令需要读者认真练习并加以掌握。更多命令信息请参阅联机从书。

本 章 小 结

SQL 语言的数据查询功能是丰富的、也是复杂的。本章介绍了基本的 SQL 查询语句，使用这些语句可以完成大部分的数据查询操作，包括基本数据查询、条件查询、嵌套查询、集合查询、连接查询、排序查询、统计函数、别名查询、分组查询、TOP 查询等，为读者学习数据库编程和进行数据库操作打下坚实的基础。

SQL 语句查询数据的基本格式为：

SELECT [ALL | DISTINCT] < 目标列表达式 >[, < 目标列表达式 >] …
FROM < 表名或视图名 >[, < 表名或视图名 >] …
[WHERE < 条件表达式 >]
[GROUP BY < 列名 1 > [HAVING < 条件表达式 >]]
[ORDER BY < 列名 2 > [ASC | DESC]]

本章所讲内容是使用数据库的主要方法和手段，读者需要加强练习，并熟练掌握。

习 题

一、选择题

1．在 SELECT 子句中关键字（ ）用于消除重复项。

A．AS B．DISTINCT C．TOP D．PERCENT

2．要使用模糊查询来从数据库中查找与某一数据相关的所有元组信息，可使用（ ）关键字。

A．AND B．OR C．ALL D．LIKE

3．下面关于分组技术的描述哪一种是正确的（ ）？

A．SELECT 子句中的非统计列必须出现在 GROUP BY 子句中

B．SELECT 子句中的非统计列可以不出现在 GROUP BY 子句中

C．SELECT 子句中的统计列必须出现在 GROUP BY 子句中

D．SELECT 子句中的统计列可以不出现在 GROUP BY 子句中

4．有关 SELECT colA colB FROM table_name 语句，下面哪种说法是正确的（ ）？

A．该语句不能正常执行，因为出现了语法错误

B．该语句可以正常执行，因为 colA 是 colB 的别名

C．该语句可以正常执行，因为 colB 是 colA 的别名

D．该语句可以正常执行，colA 和 colB 是两个不同的别名

5．下面有关 ESCAPE 子句的说法，哪些是正确的（ ）？

A．ESCAPE 子句后面的字符是转义符，该字符不能出现在匹配条件中

B．ESCAPE 子句后面的字符是转义符，该字符用于将匹配条件中的通配符转换为正常字符

C．ESCAPE 子句后面的字符是转义符，该字符可以作为通配符使用

D．ESCAPE 子句后面的字符是转义符，该字符可以出现在匹配条件中

二、填空题

1．SELECT 语句中必不可少的两个子句是_____、_____。

2．LIKE 子句中可以使用的 4 个通配符分别是_____、_____、_____、_____。

3．交叉连接也被称为笛卡儿乘积，返回两个表的乘积。可以使用_____关键字。

三、简答题

1．内连接和外连接的区别在哪里？

2．有几种改变列标题的方法？

本 章 实 训

一、实训目的

掌握数据查询的基本方法，能够熟练使用 SELECT 语句完成表中数据的简单和复杂查询。

二、实训要求

1．实训前做好上机实训的准备，针对实训内容，认真复习与本次实训有关的知识，完成实训内容的预习准备工作。

2．认真独立完成实训内容。

3．实训后做好实训总结，根据实训情况完成总结报告。

三、实训学时

8 学时。

四、实训内容

练习使用 SELECT 语句查询“Student”表、“SC”表和“Course”表中的数据。

启动 Microsoft SQL Server Management Studio 工具，打开“学生选课”数据库，新建一个查询。在查询窗口中完成以下查询操作。

1．查询选修了课程的学生学号（取消重复值）。

2．查询每个学生选修课程的门数，并为结果集中的目标列指定新的列标题（学号，课程数）。

3．查询计算机系所有学生的信息。

4．查询学生姓名中包含“华”字的学生姓名、年龄。

5．查询姓“陈”并且年龄大于 20 岁的学生姓名和所属院系。

6．将所有学生信息按照选课成绩进行排序。

7．查询选课超过一门的学生信息。

8．找出选课成绩最高的学生姓名、学号和成绩。

9．找出选课平均成绩前 3 名的学生姓名和学号。

10．设计两个查询语句，对查询结果进行并、交、差操作，并分析执行结果。

五、实训思考题

1．分析和比较连接操作与子查询操作的结果。

2．LIKE 子句可以使用哪几种通配符？这些通配符有什么作用？

第 6 章

Transact-SQL 语言

Transact-SQL 是 Microsoft SQL Server 实现的 ANSI SQL 的加强版语言，它提供了标准的 SQL 命令，另外还对 SQL 命令做了许多扩充，提供类似 Basic、Pascal、C 等第三代语言的基本功能如变量说明、程序流程控制语言、功能函数等。Transact-SQL 语言的分类如下：

- 变量说明；
- 数据定义语言（DDL，Data Definition Language）；
- 数据操纵语言（DML，Data Manipulation Language）；
- 数据控制语言（DCL，Data Control Language）；
- 流程控制语言（Flow Control Language）；
- 内嵌函数；
- 其他命令。

上述分类语言中，数据定义语言（DDL）、数据操纵语言（DML）、数据控制语言（DCL）在其他各章讲述，本章重点讨论变量说明、流程控制、内嵌函数和其他命令。

在 SQL Server 2005 中，可在 SQL Server Management Studio 中调试和运行 SQL 语句，如图 6.1 所示。

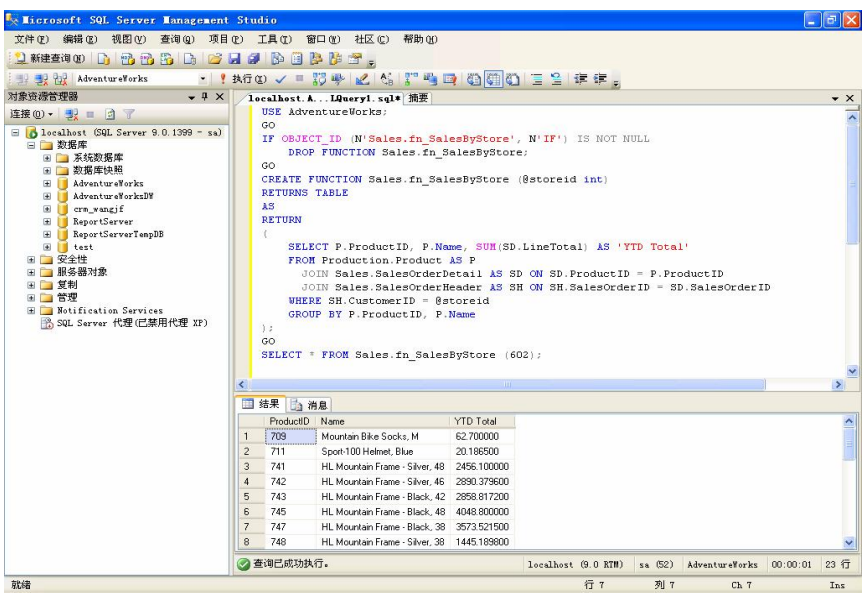


图 6.1　SQL Server 2005 Management Studio

6.1 数 据 类 型

在 SQL Server 2005 中，每个列、局部变量、表达式和参数都具有一个相关的数据类型。数据类型是一种属性。SQL Server 2005 中的数据类型归纳类别如表 6.1 所示。

表 6.1　　SQL Server 2005 提供的数据类型分类

类　　别	数 据 类 型
精确数字	bigint、int、smallint、tinyint、bit 和 decimal、numeric、money、smallmoney
近似数字	float、real
日期和时间	datetime、smalldatetime
字符串	char、text、varchar
Unicode 字符串	nchar、ntext、nvarchar
二进制	binary、image、varbinary
其他	cursor、timestamp、sql_variant、uniqueidentifier、table、xml

注意　（1）SQL Server 2005 引入了新的数据类型 xml;
（2）cursor 数据类型是唯一不能分配给表列的系统数据类型，它只能用于变量和存储过程参数。

注意　在 Microsoft SQL Server 的未来版本中将删除 ntext、text 和 image 数据类型。请避免在开发工作中使用这些数据类型，考虑修改当前使用这些数据类型的应用程序。改用 nvarchar(max)、varchar(max)和 varbinary(max)。如表 6.2 所示。

表 6.2　　大值数据类型与早期版本的大型对象的对应

大值数据类型	早期版本中的大型对象
varchar(max)	text
nvarchar(max)	ntext
varbinary(max)	image

6.1.1 精确数字类型

1. 整数类型

使用整型的精确数字类型如表 6.3 所示。

表 6.3　　使用整型的精确数字类型

数 据 类 型	范　　围	存　　储
bigint	−2^63 (−9 223 372 036 854 775 808)～2^63−1 (9 223 372 036 854 775 807)	8 Byte
int	−2^31 (−2 147 483 648)～2^31−1 (2 147 483 647)	4 Byte
smallint	−2^15 (−32 768)～2^15−1 (32 767)	2 Byte
tinyint	0～255	1 Byte

int 数据类型是 SQL Server 2005 中的主要整型数据类型，bigint 数据类型用于整数值可能超过 int 数据类型支持范围的情况。实际使用中，要根据所存储数据的最大范围来选择。

2. 逻辑数据类型

bit 是可以取值为 1、0 或 NULL 的整型数据类型。

SQL Server 2005 优化了 bit 列的存储。如果表中的列为 8bit 或更少，则这些列作为 1Byte 存储。如果列为 9 到 16bit，则这些列作为 2Byte 存储，依此类推。可将只有两个值的数据定义为 bit 类型。

3. decimal 和 numeric

decimal 和 numberic 为带固定精度和小数位数的数值数据类型。使用最大精度时，有效值从−10^38+1～10^38−1，decimal 在 SQL-92 中的同义词为 dec 和 dec(p, s)。numeric 在功能上等价于 decimal。

定义样式为：decimal[(p[, s])]和 numeric[(p[, s])]。

- p（精度）：最多可以存储的十进制数字的总位数，包括小数点左边和右边的位数。该精度必须是从 1 到最大精度 38 之间的值。
- s（小数位数）：小数点右边的最大位数，小数位数必须是从 0～p 之间的值。

如 decimal(15,5)，表示共有 15 位数，其中整数 10 位，小数 5 位。从表 6.4 可以看出，存储字节数为 9。

表 6.4　　decimal 数据类型对应字节数

精　　度	存储字节数
1～9	5
10～19	9
20～28	13
29～38	17

4. money 和 smallmoney

货币数据类型用于存储货币值，在使用货币数据类型时，应在数据前加上货币符号系统才能辨识其为哪国的货币，如果不加货币符号，则默认为“￥”。

money 和 smallmoney 代表货币或货币值的数据类型，数据类型精确到它们所代表的货币单位的万分之一。表 6.5 为货币数据类型和范围以及存储的字节数。

表 6.5　　货币数据类型

数 据 类 型	范　　围	存　　储
money	−922 337 203 685 477.5808～922 337 203 685 477.580 7	8 Byte
smallmoney	−214 748.3648～214 748.364 7	4 Byte

6.1.2　近似数字类型

用于表示浮点数值数据的类型为大数值数据类型。浮点数据为近似值，因此，并非数据类型范围内的所有值都能精确地表示。浮点数值类型的范围及存储大小如表 6.6 所示。

表 6.6　　浮点数值类型

数 据 类 型	范　　围	存　　储
float	−1.79E + 308～−2.23E−308、0 以及 2.23E−308～1.79E + 308	取决于 *n* 的值
real	−3.40E + 38～−1.18E−38、0 以及 1.18E−38～3.40E + 38	4 Byte

用法为 float[(n)]，其中 *n* 为用于存储 float 数值尾数的位数，以科学记数法表示。*n* 的范围为[1,53]，SQL Server 2005 将 *n* 视为下列两个可能值之一。如果 1<= *n*<= 24，则将 *n* 视为 24；如果 25<= *n*<= 53，则将 *n* 视为 53。如果省略 *n*，则默认为 53，如表 6.7 所示。

表 6.7　　float 存储大小和精度的关系

n 的取值	精　　度	存 储 大 小
1～24	7 位数	4 Byte
25～53	15 位数	8 Byte

Real 相当于 float(24)，而 double 相当于 float(53)。

【例 6.1】使用如下语句创建表。

```
create table testfloat(
    col1 float(20),
    col2 float(53),
    col3 float
)
```

执行之后，查看列的信息如图 6.2 所示，col1 指定 *n* 小于 24，故定义为 real 类型。

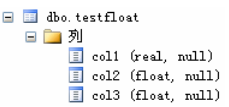


图 6.2　float 例子

6.1.3　日期和时间类型

有 datetime 和 smalldatetime 两种用于表示某天的日期和时间的数据类型，如表 6.8 所示。

表 6.8　　日期数据类型

数 据 类 型	范　　围	精　确　度
datetime	1753 年 1 月 1 日～9999 年 12 月 31 日	3.33 ms
smalldatetime	1900 年 1 月 1 日～2079 年 6 月 6 日	1 min

SQL Server 2005 用 8Byte 的整数内部存储 datetime 数据类型的值。第一个 4Byte 存储“基础日期”（即 1900 年 1 月 1 日）之前或之后的天数。另外一个 4Byte 存储天的时间（以午夜后经过的毫秒数表示）。表 6.9 为 datetime 舍入示例。

表 6.9　　Datetime 舍入示例

示　　例	舍入后的示例
01/01/98 23:59:59.999	1998-01-02 00:00:00.000
01/01/98 23:59:59.995,01/01/98 23:59:59.996, 01/01/98 23:59:59.997 或 01/01/98 23:59:59.998	1998-01-01 23:59:59.997

续表

示　　例	舍入后的示例
01/01/98 23:59:59.992,01/01/98 23:59:59.993 或 01/01/98 23:59:59.994	1998-01-01 23:59:59.993
01/01/98 23:59:59.990 或 01/01/98 23:59:59.991	1998-01-01 23:59:59.990

smalldatetime 数据类型存储天的日期和时间，但精确度低于 datetime。SQL Server 2005 将 smalldatetime 值存储为 2 个 2Byte 的整数。第一个 2Byte 存储 1900 年 1 月 1 日后的天数。另外一个 2Byte 存储午夜后经过的分钟数。将等于或小于 29.998 s 的 smalldatetime 值向下舍入到最接近的分钟数；将等于或大于 29.999 s 的值向上舍入到最接近的分钟数。

【例 6.2】执行下列 SQL 语句。

```
SELECT CAST('2003-05-08 12:35:29.109' AS datetime);
SELECT CAST('2003-05-08 12:35:29.102' AS datetime);
SELECT CAST('2003-05-08 12:35:29.998' AS smalldatetime);
SELECT CAST('2003-05-08 12:35:29.999' AS smalldatetime);
```

结果如图 6.3 所示。

SQL Server 可识别下列格式中用单引号（'）括起来的日期和时间：

- 字母日期，如“April 15, 1998”；
- 数值日期格式，如“4/15/1998”；
- 未分隔的字符串格式，如“19981207”指 1998 年 12 月 7 日。

注意　可以使用 set dateformat 来设置输入 datetime 或 smalldatetime 数据的日期部分（月/日/年）的顺序。

【例 6.3】执行以下 SQL 语句。

```
SET DATEFORMAT mdy;          -- Set date format to month, day, year
DECLARE @datevar DATETIME;
SET @datevar = '12/31/1998';
SELECT @datevar AS DateVar;
SET DATEFORMAT dmy;          -- Set date format to day, month, year
DECLARE @datevar DATETIME;
SET @datevar = '31/12/1998';
SELECT @datevar AS DateVar;
```

执行结果如图 6.4 所示。

图 6.3　时间舍入例子

图 6.4　设置日期和时间格式

6.1.4 字符数据类型

字符数据类型是使用最多的数据类型，它可以用来存储各种字母、数字符号、特殊符号。一般情况下，使用字符类型数据时须在其前后加上单引号（’）或双引号（”）。以下介绍字符串和 Unicode 字符串。

1．char

char 数据类型的定义形式为

char[（n）]，

表示长度为 *n* Byte（若不指定 *n* 值则系统默认值为 1），*n* 的取值范围为 1～8 000，存储大小是 *n* Byte。若输入数据的字符数小于 *n* 则系统自动在其后添加空格来填满设定好的空间，若输入的数据过长，将会截掉其超出部分。

2．nchar

nchar 数据类型的定义形式为

nchar[（n）]。

它与 char 类型相似，不同的是 nchar 数据类型中 *n* 的取值为 1～4 000，因为 nchar 类型采用 Unicode 标准字符集，Characterset Unicode 标准规定每个字符占用 2Byte 的存储空间，所以它比非 Unicode 标准的数据类型多占用一倍的存储空间。

使用 Unicode 标准的好处是使用 2Byte 做存储单位，一个存储单位的容纳量就大大增加了，可以将全世界的语言文字都囊括在内，在一个数据列中就可以同时出现中文、英文、法文、德文等，而不会出现编码冲突。

Unicode 字符串常量必须以大写字母 N 为前缀。

3．varchar

varchar 数据类型的定义形式为 varchar[(n | max)]。它与 char 类型相似，*n* 的取值也为 1～8 000。若输入的数据过长，将会截掉其超出部分。max 指示最大存储大小是 $2^{31}-1$Byte。不同的是 varchar 数据类型具有变动长度的特性，存储大小是输入数据的实际长度加 2Byte。

4．nvarchar

nvarchar 数据类型的定义形式为 nvarchar[(n | max)]，为可变长度 Unicode 字符数据类型。*n* 值在 1～4 000 之间（含）。max 指示最大存储容量为 2^31−1Byte。存储大小是所输入字符个数的 2 倍+2Byte。

一般情况下，有如下建议。

（1）如果支持多语言，请考虑使用 nchar 或 nvarchar 数据类型。

（2）对于 char(nchar)和 varchar(nvarchar)的区别，建议如下：

① 如果列数据项的大小一致，则使用 char(nchar)，由于 char(nchar)数据类型长度固定，因此它比 varchar(nvarchar)类型的处理速度快；

② 如果列数据项的大小差异较大，则使用 varchar(nvarchar)；

③ 如果列数据项大小相差很大，而且大小超过 8 000 Byte，使用 varchar(max)或 nvarchar (max)。

5．text

用于存储大型非 Unicode 字符，最大长度为 $2^{31}-1$ 个字符，存储大小是所输入字符个数的 2 倍（以字节为单位），在实际应用时需要视硬盘的存储空间而定。

text 类型将在未来版本中删除，请使用 varchar（max）代替。

6．ntext

用于存储大型 Unicode 字符，最大长度为 $2^{30}-1$ 个字符。ntext 类型将在未来版本中删除，请使用 nvarchar（max）代替。

6.1.5　二进制数据类型

二进制数据类型用来存储固定长度或可变长度的 binary 数据类型。

1．binary

binary 数据类型用于存储固定长度二进制数据，其定义形式为 binary [(n)]，其中 *n* 是从 1～8 000。存储大小为 n Byte。

在输入数据时必须在数据前加上字符“0x”作为二进制标识，如要输入“abc”，则应输入“0xabc”。若输入的数据过长将会截掉其超出部分，若输入的数据位数为奇数，则会在起始符号“0x”后添加一个 0，如上述“0xabc”会被系统自动变为“0x0abc”。

2．varbinary

varbinary 数据类型用于存储可变长度二进制数据，其定义形式为 varbinary [(n | max)]，其中 *n* 是从 1～8 000。max 指示最大的存储大小为 2^31−1 Byte。存储大小为所输入数据的实际长度+ 2 Byte。

对于 binary 和 varbinary 的使用，建议如下：

（1）如果列数据项的大小一致，则使用 binary；

（2）如果列数据项的大小差异相当大，则使用 varbinary；

（3）当列数据条目超出 8 000 Byte 时，使用 varbinary(max)。

3．image

image 存储长度大量可变的二进制数据，从 0～2^31−1 (2 147 483 647) Byte。

它通常用来存储图形等 OLE（Object Linking and Embedding，对象连接和嵌入）对象。在输入数据时同 binary 数据类型一样，必须在数据前加上字符“0x”作为二进制标识。

6.1.6　其他类型

1．游标 cursor

在数据库开发过程中，经常需要从某一结果集中逐一读取每条记录，那么如何解决这种问题呢？游标为我们提供了一种极为优秀的解决方案，关于游标的详细内容，请参阅本书第 10 章。

2．timestamp

公开数据库中自动生成的唯一二进制数字的数据类型。TIMESTAMP 通常用作给表行加版本戳的机制。存储大小为 8 Byte。

一个表只能有一个 timestamp 列。每次修改或插入包含 timestamp 列的行时，就会在 timestamp 列中插入增量数据库时间戳值。这一属性使 timestamp 列不适合作为键使用，尤其是不能作为主键使用。

在 CREATE TABLE 或 ALTER TABLE 语句中，不必为 timestamp 数据类型指定列名，如 CREATE TABLE EXAMPLETABLE (PRIKEY INT PRIMARY KEY, TIMESTAMP);

另外，使用 SELECT @@DBTS，可以返回数据库的时间戳。

3．sql_variant

一种数据类型，用于存储 SQL Server 2005 支持的各种数据类型（不包括 text、ntext、image、timestamp 和 sql_variant）的值。

4．table

一种特殊的数据类型，用于存储结果集以进行后续处理。table 主要用于临时存储一组行，这些行是作为表值函数的结果集返回的。

使用 DECLARE @local_variable (Transact-SQL)来声明类型为 table 的变量。

5．xml

存储 xml 数据的数据类型。可以在列中或者 xml 类型的变量中存储 xml 实例。xml 数据类型方法有：query()，value()，exists()，modify()，nodes()。

6.1.7 用户自定义类型

可以从基本数据类型创建别名数据类型。这使程序员能更容易地理解该数据类型的用途。如 CREATE TYPE birthday FROM datetime NULL，接下来就可以使用 birthday 来定义数据对象了。

6.2 变　　量

在 Transact-SQL 中，变量分为局部变量（Local Variable）和全局变量（Global Variable）。局部变量由用户定义和维护，而全局变量由系统定义和维护。

6.2.1 局部变量

局部变量的作用范围仅在程序内部，通常用来储存从表中查询到的数据，或当作程序执行过程中的暂存变量。

局部变量必须以@开头，而且必须先用 DECLARE 命令声明后才可使用，其声明形式如下：

```
DECLARE  @变量名 变量类型 [, @变量名 变量类型…]
```

其中变量类型可以是 SQL Server 2005 支持的所有数据类型，也可以是用户自定义的数据类型。

在 Transact-SQL 中不能像在一般的程序语言中，使用“变量=变量值”来给变量赋值，必须使用 SELECT 或 SET 命令来设定变量的值，其语法如下：

```
SELECT @局部变量 = 变量值
SET @局部变量 = 变量值
```

【例 6.4】执行下列 SQL 语句。

```
USE AdventureWorks;
GO
DECLARE @EmpID int;
SET @EmpID = 100;
SELECT *  FROM humanresources.employee  WHERE EmployeeID = @EmpID;
```

示例数据库“AdventureWorks”默认是不安装的，必须安装之后才能运行本代码。

变量也可以作为存储过程参数，参数是用于在存储过程和执行该存储过程的批处理或脚本之间传递数据的对象。参数分为输入参数和输出参数。

【例 6.5】用@PruductID 作为输入参数。

```
USE AdventureWorks;
GO
CREATE PROCEDURE ParmSample
@PruductID int
AS
SELECT Name,SellStartDate FROM production.product
WHERE PruductID = @PruductID
GO
EXEC ParmSample @PruductID = 322
GO
```

6.2.2　全局变量

全局变量是 SQL Server 系统内部使用的变量，其作用范围并不局限于某一程序，而是任何程序均可随时调用。全局变量通常存储一些 SQL Server 的配置设定值和效能统计数据。用户可在程序中用全局变量来测试系统的设定值或 Transact-SQL 命令执行后的状态值。

例如，@@VERSION 表示返回当前安装的 SQL Server 的日期、版本、处理器。而@@CONNECTIONS 表示自 SQL Server 最近一次启动以来，连接或企图连接到 SQL Server 的连接数目。执行语句 SELECT @@CONNECTIONS 即可返回其值。

注意　全局变量不是由用户的程序定义的，它们是在服务器定义的，所以只能使用预先说明及定义的全局变量。引用全局变量时，必须以@@开头。局部变量的名称不能与全局变量的名称相同，否则会在应用中出错。

6.3　运算符及表达式

6.3.1　运算符

1. 算术运算符

算术运算符用于对两个表达式执行数学运算，这两个表达式是数值数据类型的算术运算符主要有：+（加）、－（减）、*（乘）、/（除）、%（取余）。

其中+（加）和－（减）运算符也可用于对 datetime 和 smalldatetime 类型的值进行算术运算。

2. 赋值运算符

=(等号) 是唯一的 Transact-SQL 赋值运算符。

3. 位运算符

位运算符的操作数可以是整数或二进制字符串数据类型类别中的任何数据类型（image

数据类型除外），但两个操作数不能同时是二进制数据类型中的某种数据类型。位运算符有：&（位与）、|（位或）、^（位异或）。

4. 比较运算符

比较运算符测试两个表达式是否相同。比较运算符可以用于除了 text、ntext 或 image 数据类型外的所有类型。

比较运算符有：=（等于）、>（大于）、<（小于）、>=（大于等于）、<=（小于等于）、<>（不等于）、!=（不等于）、!<（不小于）、!>（不大于）。返回值为 True、False、UNKNOWN 3 种。

5. 逻辑运算符

逻辑运算符对某些条件进行测试，以获得其真实情况。逻辑运算符和比较运算符一样，返回带有 True 或 False 的 Boolean 数据类型。逻辑运算符的含义如表 6.10 所示。

表 6.10　　逻辑运算符表

运　算　符	含　　义
ALL	如果所有表达式都为 True，那么就为 True
AND	如果两个条件表达式都为 True，那么就为 True
ANY	只要有一个表达式为 True，那么就为 True
BETWEEN	如果操作数在某个范围之内，那么就为 True
EXISTS	如果子查询包含一些行，那么就为 True
IN	如果操作数等于表达式列表中的一个，那么就为 True
LIKE	如果操作数与一种模式相匹配，那么就为 True
NOT	对任何其他布尔运算符的值取反
OR	如果两个条件表达式中的一个为 True，那么就为 True
SOME	如果在所有表达式中，有些为 True，那么就为 True

6. 字符串串联运算符

+（加号）是字符串串联运算符，可以用它将字符串串联起来，其他所有字符串操作都使用字符串函数如 SUBSTRING 进行处理。

7. 一元运算符

一元运算符有：+（正）、－（负）、～（位非）。

+（正）和－（负）运算符可以用于 numeric 数据类型的表达式。～（位非）运算符只能用于整数数据类型的表达式。

8. 运算符的优先级

运算符优先级从高到低如表 6.11 所示。当一个表达式中的两个运算符有相同的运算符优先级时，将按照它们在表达式中的位置对其从左到右进行求值。

表 6.11　　运算符的优先级

级　　别	运　算　符
1	～（位非）
2	*（乘）、/（除）、%（取模）
3	+（正）、－（负）、+（加）、（+ 连接）、－（减）、&（位与）
4	=,>、<、>=、<=、<>、!=、!>、!<（比较运算符）
5	^（位异或）、|（位或）

续表

级　别	运　算　符
6	NOT
7	AND
8	ALL、ANY、BETWEEN、IN、LIKE、OR、SOME
9	=（赋值）

6.3.2　表达式

表达式是标识符、值和运算符的组合，可以对其求值以获取结果。例如，可以将表达式用作要在查询中检索的数据的一部分，也可以用作查找满足一组条件的数据时的搜索条件。

表达式可以是下列任何一种：常量、函数、列名、变量、子查询、CASE、NULLIF 或 COALESCE，还可以用运算符对这些实体进行组合以生成表达式。

【例 6.6】下面查询中使用了多个表达式。employeeid、year(birthdate)、sickleavehours 和 5 都是表达式。

```
SELECT employeeid,year(birthdate),sickleavehours*5
FROM humanresources.employee    WHERE employeeid = 1
```

6.3.3　注释符

注释是程序代码中不执行的文本字符串（也称为备注）。注释通常用于记录程序名、作者姓名和主要代码更改的日期。注释可用于描述复杂的计算或解释编程方法。暂不需要执行的代码也可注释掉。

在 Transact-SQL 中可使用两类注释符。

--（双连字符）

可以注释从双连字符到行尾的代码。

/* ... */（正斜杠-星号字符对）

开始注释对（/*）与结束注释对（*/）之间的所有内容均视为注释，因此可以跨越多行进行注释。

6.3.4　通配符

通配符主要用于 LIKE 运算符中，用来比较字符串是否与指定模式相匹配，请参阅本书 5.3.2 小节“模糊查询”。

6.4　控制语句和批处理

Transact-SQL 提供了控制语句，用于控制 Transact-SQL 语句、语句块和存储过程的执行流。控制流语言使用与其他高级程序设计语言相似。控制流语句不能跨越多个批处理或存储过程。

6.4.1 IF...ELSE

其语法如下：

```
IF 条件表达式
    {SQL 语句或语句组}
[ELSE
    {SQL 语句或语句组}]
```

IF 指定 Transact-SQL 语句的执行条件。条件表达式的值为 True，则执行 IF 和条件表达式之后的 SQL 语句，如果条件表达式的值为 False，并且有 ELSE 语句，则执行 ELSE 后的 SQL 语句。IF 语句允许嵌套使用。

【例 6.7】查看雇员表中编号为 1 的员工的性别。

```
USE AdventureWorks;
GO
DECLARE @gender char
SELECT @Gender =gender FROM humanresources.employee
WHERE employeeid = 1
IF @Gender = 'M'    print N'男'    ELSE    print N'女'
```

6.4.2 BEGIN...END

其语法如下：

```
BEGIN
    {SQL 语句或语句组}
END
```

BEGIN…END 包括一系列的 Transact-SQL 语句，将在 BEGIN…END 内的所有程序视为一个单元执行。BEGIN…END 允许嵌套使用。

6.4.3 WHILE…CONTINUE…BREAK

只要 WHILE 后的条件为真，循环执行 SQL 语句。其语法如下：

```
WHILE 条件表达式
BEGIN
    {SQL 语句或语句块}
    BREAK
    {SQL 语句或语句块}
    CONTINUE
    {SQL 语句或语句块}
END
```

当 WHILE 命令之后的条件表达式值为 True 时，重复执行 SQL 语句或语句块。CONTINUE 命令可以让程序跳过 CONTINUE 命令之后的语句，回到 WHILE 循环内的第一行命令，BREAK 命令则让程序跳出当前 WHILE 循环，将执行出现在 END 关键字后的内容。

WHILE 语句允许嵌套。在循环嵌套时，BREAK 只能跳出当前 WHILE 循环，从而进入

外层 WHILE 循环。

【例 6.8】执行如下步骤：

① 如果产品的平均标价小于$300，将价格乘 2；

② 然后选择最高价格。如果最高价格大于$500，则 WHILE 循环结束；

③ 重复执行第 1 步；

④ 最后退出 WHILE 循环，并打印一条消息。

```
USE AdventureWorks;
GO;
WHILE (SELECT AVG(ListPrice) FROM Production.Product) < $300
BEGIN
    UPDATE Production.Product
        SET ListPrice = ListPrice * 2
    SELECT MAX(ListPrice) FROM Production.Product
    IF (SELECT MAX(ListPrice) FROM Production.Product) > $500
        BREAK
    ELSE
        CONTINUE
END
PRINT 'Too much for the market to bear';
```

6.4.4　CASE

根据不同条件表达式返回对应的结果，如果哪个条件都不满足，则返回 ELSE 分支的结果。CASE 具有两种格式：

- 简单 CASE 函数将某个表达式与一组简单表达式进行比较以确定结果；
- CASE 搜索函数计算一组布尔表达式以确定结果。

两种格式都支持可选的 ELSE 参数。

1. 简单 CASE 函数

简单 CASE 函数格式为

```
CASE {输入表达式}
    WHEN {值表达式} THEN {结果表达式}
    ...
    ELSE {结果表达式}
END
```

【例 6.9】如果是男员工洗漱费为 80，女员工为 100，程序如下。

```
USE adventureworks;
GO;
SELECT employeeid,washFee =
    CASE gender
        WHEN 'M' then 80
```

```
        WHEN 'F' then 100
    END
FROM humanresources.employee
```

2. 搜索函数表达式

搜索函数表达式的语法格式为：

```
CASE
    WHEN 条件表达式 THEN 结果表达式
    ...
    WHEN 条件表达式 THEN 结果表达式
    [ELSE 结果表达式]
END
```

【例 6.10】根据交税类型设定不动的税率增长率。

```
USE AdventureWorks;
GO
Update Sales.SalesTaxRate
SET TaxRate =
    CASE
        WHEN TaxType=1 THEN TaxRate*1.1
        WHEN TaxType=2 THEN TaxRate*1.2
        WHEN TaxType=3 THEN TaxRate*1.3
END
```

6.4.5 RETURN

RETURN 语法格式为：

```
RETURN [整数值]
```

RETURN 语句无条件终止查询、存储过程或批处理。在存储过程或批处理中，RETURN 语句后面的语句都不执行。

可以指定整数值返回。如果未指定值，默认返回 0。一般情况下，没有发生错误时返回值 0。任何非 0 值都表示有错误发生。

【例 6.11】删除表中指定时间段的数据，当开始时间大于结束时间时，程序返回。此存储过程要求删除的表中有“reporttime”字段表示时间。

```
CREATE PROCEDURE dbo.sp_deleteData
    @tablename      varchar(40),
    @begindatetime smalldatetime,
    @enddatetime    smalldatetime,
AS
BEGIN
    DECLARE @asSQL varchar(300)
    IF(@begindatetime > @endDateTime )
```

```
        RETURN 1;
    SELECT @asSQL=' delete FROM '+ @tablename
                + ' WHERE '
      + ' reporttime >= '"+CONVERT(char(20),@beginDateTime,20)+""
                    + ' AND reporttime < '"+CONVERT(char(20),@enddatetime,20)+""
    EXEC(@asSQL)
END
```

6.4.6　批处理

批处理是一个 SQL 语句集，这些语句一起提交并作为一个组来执行。批处理结束的符号是“GO”。由于批处理中的多个语句是一起提交给 SQL Server 的，所以可以节省系统开销。

CREATE DEFAULT、CREATE PROCEDURE、CREATE RULE、CREATE TRIGGER 和 CREATE VIEW 语句不能在批处理中与其他语句组合使用。批处理必须以 CREATE 语句开始，所有跟在 CREATE 后的其他语句将被解释为第一个 CREATE 语句定义的一部分。

- 在同一个批处理中不能既绑定到列又使用规则或默认。
- 在同一个批处理中不能删除一个数据库对象又重建它。
- 在同一个批处理中不能改变一个表再立即引用其新列。

脚本是一系列顺序提交的批处理。

6.4.7　其他命令

1．BACKUP 和 RESTORE

备份和恢复数据库。详情请参阅本书“数据库的备份还原与数据传输”的章节。

2．USE

指定当前数据库。

3．EXECUTE（EXEC）

执行 SQL 字符串或存储过程。

【例 6.12】执行字符串或字符串变量。

```
EXEC ('USE AdventureWorks; GO; SELECT EmployeeID, Title FROM HumanResources.Employee;');
```

【例 6.13】执行存储过程。

隐式传递变量：EXEC dbo.uspGetEmployeeManagers 6

显式传递变量：EXEC dbo.uspGetEmployeeManagers @EmployeeID = 6;

4．PRINT

打印字符串，其他类型变量需要先使用 CONVERT 转换为字符串才能使用 PRINT。

【例 6.14】PRINT N'This message was printed on '

```
        + RTRIM(CAST(GETDATE() AS NVARCHAR(30))) + N'.';
```

5．SHUTDOWN

关闭数据库服务器，其语法为：

```
SHUTDOWN [ WITH NOWAIT ]。
```

WITH NOWAIT：可选参数，不对每个数据库执行检查点操作，在尝试终止全部用户进

程后即退出。服务器重新启动时，将针对未完成事务执行回滚操作。

6.5 常 用 函 数

SQL Server 2005 提供了许多内置函数，同时也允许创建用户定义函数。函数类型如表 6.12 所示，本章介绍其中最常用的函数。

表 6.12　　函数类型

行 集 函 数	返回可在 SQL 语句中像表引用一样使用的对象
聚合函数	对一组值进行运算，但返回一个汇总值
排名函数	对分区中的每一行均返回一个排名值
标量函数	对单一值进行运算，然后返回单一值

对于标量函数，分为如下类型，如表 6.13 所示。

表 6.13　　标量函数分类

配 置 函 数	返回当前配置信息
游标函数	返回游标信息
日期和时间函数	对日期和时间输入值执行运算，返回字符串、数字或日期和时间值
数学函数	基于作为函数的参数提供的输入值执行运算，返回数字值
元数据函数	返回有关数据库和数据库对象的信息
安全函数	返回有关用户和角色的信息
字符串函数	对字符串输入值执行运算，返回一个字符串或数字值
系统函数	执行运算后返回 SQL Server 实例中有关值、对象和设置的信息
系统统计函数	返回系统的统计信息

6.5.1 聚合函数

聚合函数对一组值执行计算，并返回单个值。除了 COUNT 以外，聚合函数都会忽略空值。聚合函数只能在以下位置作为表达式使用。

（1）SELECT 语句的选择列表（子查询或外部查询）。

（2）COMPUTE 或 COMPUTE BY 子句。

（3）HAVING 子句。

Transact-SQL 提供下列聚合函数，如表 6.14 所示。

表 6.14　　聚合函数

函　　数	说　　明
SUM、AVG	返回表达式中所有值的和、求平均值
MAX、MIN	返回表达式中所有值的最大值、最小值
COUNT、COUNT_BIG	返回表达式中所有值的个数，COUNT 返回 int 型，COUNT_BIG 返回 bigint 型值
STDDV、STDEVP	返回表达式中所有值的标准偏差、总体标准偏差
VAR、VARP	返回指定表达式中所有值的方差、总体方差

【例 6.15】统计计算机考试成绩。

总分：SELECT sum(Grade) FROM Stu_Test WHERE subject = '计算机'

平均分：SELECT Avg(Grade) FROM Stu_Test WHERE subject = '计算机'

最高分：SELECT MAX(Grade) FROM Stu_Test WHERE subject = '计算机'

人数：SELECT COUNT(Grade) FROM Stu_Test WHERE subject = '计算机'

标准偏差：SELECT STDEV(Grade) FROM Stu_Test WHERE subject = '计算机'

如果要按照班级进行统计，则在以上 SQL 语句结尾添加

```
GROUP BY class
```

如果只统计计算机总分大于 3 000 的班级。则为：

```
SELECT SUM(Grade)
FROM Stu_Test WHERE subject = '计算机'
GROUP BY class
HAVING SUM(Grade)>3000
```

6.5.2　标量函数

1. 数学函数

数学函数对所有数值类型进行操作。数学函数的返回值是 6 位小数，如果使用出错，则返回 NULL 并显示警告信息。

（1）三角函数 SIN（正弦）、COS（余弦）、TAN（正切）、COT（余切）。

```
SIN ( float_expression )
```

参数为弧度，返回类型为浮点数。

（2）三角函数 ASIN（反正弦）、ACOS（反余弦）、ATAN（反正切）、ATAN2（反余切）。

```
ASIN ( float_expression )
```

参数为浮点数，返回类型为弧度。

（3）角度转换函数 RADIANS（角度→弧度）、DEGREES（弧度→角度）。

```
RADIANS ( numeric_expression )
```

输入角度，返回弧度值。

```
DEGREES ( numeric_expression ),
```

输入弧度，返回角度值。

（4）幂函数 SQRT，SQUARE、EXP、LOG，LOG10，POWER。

```
SQRT ( float_expression )
```

返回表达式的平方根。

```
SQUARE ( float_expression )
```

返回表达式的平方。

```
POWER ( numeric_expression , y )
```

返回 numeric_expression 的 y 次方。

```
EXP ( float_expression )
```

返回自然对数 e 的 float_expression 次方。

```
LOG ( float_expression )
```

返回以自然对数 e 为底的 float_expression 的对数值。

LOG10 (float_expression)

返回以 10 为底的 float_expression 的对数值。

（5）符号函数 ABS，SIGN。

ABS (numeric_expression)

返回指定数值表达式的绝对值（正值）。

SIGN (numeric_expression)

返回指定表达式的正号（+1）、零（0）或负号（−1）。

（6）近似函数 ROUND、CEILING、FLOOR。

FLOOR (numeric_expression)

返回小于或等于指定数值表达式的最大整数。

CEILING (numeric_expression)

返回大于或等于指定数值表达式的最小整数。

ROUND (numeric_expression , length [,function])

将数值表达式舍入到指定的长度或精度，length 实际是精度的意思，精确到小数点后 length 位，如果 length 为负值，则表示精确到小数点前。省略 function 或使用 0 值（默认）时，将对 numeric_expression 进行舍入。当 function 指定为非 0 值时，将对 numeric_expression 进行截断，如表 6.15 所示。

表 6.15　　近似函数

表　达　式	值
Ceiling(−3.5)	−3
Ceiling(3.5)	4
Floor(−3.5)	−4
Floor(3.5)	3
ROUND(123.9994, 3)	123.999
ROUND(123.9995, 3)	124.000 0
ROUND(748.58, −1)	750.00
ROUND(748.58, −2)	700.00
ROUND(150.75, 0)	151.00
ROUND(150.75, 0, 2)	150

【例 6.16】使用 SQL 语句来求以上表达式的值。

```
SELECT CEILING(-3.5),CEILING(3.5),
       FLOOR(-3.5),FLOOR(3.5),
       ROUND(123.9994, 3),ROUND(123.9995, 3),
       ROUND(748.58, -1),ROUND(748.58, -2),
       ROUND(150.75, 0),ROUND(150.75, 0, 2)
```

（7）其他函数 RAND、PI。RAND 语法格式为：

RAND ([seed])

返回从 0 到 1 之间的随机 float 值。可以指定 seed 值，对于指定的 seed 值，RAND 产生

的值相同。

对于一个连接，RAND 的所有后续调用将基于首次 RAND 调用生成结果。

【例 6.17】产生 10 个 0～3 之间的随机整数。

```
DECLARE @counter smallint
SET @counter = 1
WHILE @counter <= 10
BEGIN
SELECT round(RAND()*4,0) Random_Number
    SET @counter = @counter + 1
END
```

PI 的语法格式为：

```
PI()
```

返回圆周率π的常量值。如：SELECT PI()*2.5*2，返回半径为 2.5 的圆的周长。

2. 日期函数

日期函数用来操作 datetime 和 smalldatetime 类型的数据。

（1）当前日期函数 GETDATE、GETUTCDATE。其语法格式为：

```
GETDATE ( )
```

按照 SQL Server 标准内部格式返回当前系统日期和时间的 datetime 值。

在查询中，日期函数可用于 SELECT 语句的选择列表或 WHERE 子句。在设计报表时，GETDATE 函数可用于在每次生成报表时打印当前日期和时间。GETDATE 对于跟踪活动也很有用，如记录事务在某一帐户上发生的时间。

GETUTCDATE 返回当前 UTC 时间（格林尼治标准时间）的 datetime 值。

【例 6.18】执行下列 SQL 语句。

返回系统日期：SELECT GETDATE()

使用系统日期创建表：

```
CREATE TABLE STUDENT(
            stu_ID char(11) NOT NULL,
            STU_name varchar(40) NOT NULL,
            Enter_Date    datetime DEFAULT GETDATE(),
)
```

（2）日期部分值函数 DAY、MONTH、YEAR。

```
DAY ( date )
```

返回一个整数，表示指定日期的“天”部分。

```
MONTH ( date )
```

返回表示指定日期的“月”部分的整数。

```
YEAR ( date )
```

返回表示指定日期的“年”部分的整数。

【例 6.19】执行下列 SQL 语句。

```
SELECT YEAR('2007-8-1'),MONTH('2007-8-1'),DAY('2007-8-1')
```

执行结果为 2007，8，1。

（3）日期部分操作函数 DATENAME、DAAEPART、DATEADD、DATEDIFF。

DATENAME (datepart ,date)

返回指定日期的部分字符串。

DATEPART (datepart , date)

返回指定日期的部分值。

DATEADD (datepart , number, date)

给指定日期的部分加上一个时间间隔。

DATEDIFF (datepart , startdate , enddate)

返回两个指定日期的部分间隔值。

其中 date、startdate、enddate 表示 datetime 或 smalldatetime 型值，或日期格式的字符串。如果输入字符串，需将其放入引号中。

参数 datepart 指定要返回的日期部分的参数，其含义如表 6.16 所示。

表 6.16　　datepart 含义

日 期 部 分	缩　　写
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
Hour	hh
minute	mi, n
second	ss, s
millisecond	ms

【例 6.20】返回当前系统日期的年、月、日、星期。

```
SELECT DATEPART(yy,getdate()),DATEPART(mm,getdate()),DATEPART(dd,getdate()),
        DATEPART(dw,getdate())
```

返回当前日期的小时字符串

```
SELECT DATENAME(hour, getdate())
```

返回当前日期 49 天之后的日期

```
SELECT dateadd(dd,49,getdate())
```

计算你出生之后的天数，假设你的生日为“1986-1-1 3:00:00”

```
SELECT datediff(dd,'1981-1-1 3:00:00',getdate())
```

3．字符串函数

字符串函数对字符串输入值执行操作，并返回字符串或数值。

（1）长度函数 LEN。返回指定字符串表达式的字符（而不是字节）数，其中不包含尾随空格，用于 varchar、varbinary、text、image、nvarchar 和 ntext 数据类型。NULL 的 DATALENGTH 的结果是 NULL。LEN 函数和 DATALENGTH 函数的功能一样。

其语法格式如下：

LEN (string_expression)

【例 6.21】求每个学生姓名的最大长度。

```
SELECT max(len(stuname)) as NAME_MAX_LEN from student
```

（2）子串函数 LEFT、RIGHT、SUBSTRING。这 3 个函数求指定字符串的子串，如表 6.17 所示。

表 6.17　　求子串

表　达　式	值
LEFT('abcdefg',2)	'ab'
RIGHT('abcdefg',3)	'efg'
SubString（'abcdefg',3,2)	'de'

其语法格式如下。

```
LEFT ( character_expression , integer_expression )
```

返回字符串中从左边开始指定个数的字符。

```
RIGHT ( character_expression , integer_expression )
```

返回字符串中从右边开始指定个数的字符。

```
SUBSTRING ( expression ,start , length )
```

返回字符表达式、二进制表达式、文本表达式或图像表达式的一部分。

（3）字符串转换函数 ASCII、UNICODE、CHAR、NCHAR、LOWER、UPPER、STR。字符转换函数如表 6.18 所示，其语法格式如下。

表 6.18　　字符串转换函数

表　达　式	值
ascii('a')	97
Unicode(N'数据库')	25968
char(98)	'b'
nchar(99)	'c'
upper('aBc')	'ABC'
lower('DeF')	'abc'
Str(123.45)	'123'
str(123.45,10,1)	'123.5'
str(123.45,5,1)	'123.5'
str(123.45,2,1)	'**'

```
ASCII ( character_expression )
```

返回字符表达式中最左侧的字符的 ASCII 值。

```
UNICODE ( 'ncharacter_expression' )
```

返回输入表达式的第一个字符的整数值。

CHAR (integer_expression)

将 int 型 ASCII 代码转换为字符。

```
NCHAR ( integer_expression )
```

返回具有指定的整数代码的 Unicode 字符。

```
LOWER ( character_expression )
```

返回指定字符串的小写。

UPPER (character_expression)

返回指定字符串的大写。

STR (float_expression [, length [, decimal]])

返回由数字数据转换来的字符数据。其中 length 表示转换后的总长度（包括小数点、符号、数字以及空格），缺省为 10。decimal 表示小数点之后的位数，缺省为 0，即缺省没有小数位。length 长度太小时，则转化为“**”。

（4）去空格函数 LTRIM、RTRIM。

LTRIM (character_expression)

去掉字符串左边空格。

RTRIM (character_expression)

去掉字符串右边空格。

如果要去掉两边的空格，则需要将两个函数嵌套。

【例 6.22】去掉字符串两边的空格。

```
SELECT rtrim(ltrim('     hello     '))
```

结果为：

```
'hello'
```

（5）字符串替换函数 REPLACE。其语法格式为：

REPLACE (string_expression1 ,string_expression2 , string_expression3)

用 string_expression3 替换 string_expression1 中所有 string_expression2 的匹配项。

【例 6.23】执行下列 SQL 语句。

```
SELECT REPLACE('abcdefghicde','cde','xxx')
```

结果为：

```
abxxxfghixxx
```

（6）字符串比较函数 CHARINDEX、PATINDEX。语法格式为：

CHARINDEX (expression1 ,expression2 [, start_location])

表示在 expression2 中，从第 start_location 个字符开始，寻找第一次出现 expression1 的位置并返回。

【例 6.24】执行下列 SQL 语句。

```
SELECT CHARINDEX('ef','abcdefghefhi')
```

结果为 5。

```
SELECT CHARINDEX('ef','abcdefghefhi',6)
```

由于是从 6 个开始找，则将找到第 2 个'ef'所在位置，则结果为 9。

PATINDEX 的语法格式为：

PATINDEX ('%pattern%' , expression)

返回指定表达式中某模式第一次出现的起始位置。可以进行模糊查找（字符串前后加'%'）。

CHARINDEX 和 PATINDEX 的区别在于：前者可以指定开始查找位置，后者不能，只能从开始查找；前者不可以使用通配符'%'进行模糊查找，后者可以。

（7）其他函数。

- SPACE

语法格式为：

SPACE (integer_expression)

产生指定个数空格的字符串。

- REPLICATE

语法格式为：

REPLICATE (character_expression ,integer_expression)

产生指定个数和字符的字符串。

- REVERSE

语法格式为：

REVERSE (character_expression)

返回字符表达式的逆向表达式。

表 6.19 所示为其他字符节转换函数。

表 6.19　　字符串转换函数

表　达　式	值
SPACE(5)	'　　'
REPLICATE('数',3)	'数数数'
REVERSE('Hello')	'olleH'

4. 数据类型转换函数

在一般情况下 SQL Server 会自动完成数据类型的转换，例如，可以直接将字符数据类型或表达式与 datatime 数据类型或表达式比较。当表达式中用 integer、smallint 或 tinyint 时，SQL Server 也可将 integer 数据类型或表达式转换为 smallint 数据类型或表达式，这称为隐式转换。如果不能确定 SQL Server 是否能完成隐式转换或者使用了不能隐式转换的其他数据类型，就需要使用数据类型转换函数做显式转换了，此类函数有 CAST 和 CONVERT。

其语法格式为：

```
CAST ( expression AS data_type [ (length ) ])
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

将 expression 转换为 data_type。length 主要用于转换为字符串、二进制类型时指定长度。style 在日期转换为字符串时指定格式，或是数值转换为字符串时指定格式。具体格式请参阅 SQL Server 联机丛书。

【例 6.25】执行下列 SQL 语句。

```
SELECT getdate(),
            CONVERT (char(12),    getdate()),
            CONVERT (char(24), getdate(),100),
CONVERT (char(12),    getdate(),112)
```

执行结果如图 6.5 所示。

5. 系统函数

系统函数可以访问 SQL Server 2005 系统表中的信息。建议使用系统函数来获得系统信息，而不要直接查询系统表，因为不同版本 SQL Server 的系统表可能会有较大差别。

图 6.5　CONVERT

（1）带@@的函数。某些 Transact-SQL 系统函数的名称以两个@开头（全局变量也是以@@开头），但它们不是全局变量，而且其功能与变量的功能不同。其语法的使用遵循函数的规则，如表 6.20 所示。

表 6.20　　带@@的系统函数

函　数	说　明
@@ERROR	上一个语句的错误号。如没有错误返回 0
@@IDENTITY（函数）	最后插入的标识值
@@ROWCOUNT	受上一语句影响的行数
@@TRANCOUNT	当前连接的活动事务数

（2）判断格式是否有效的函数

有效则返回 1，否则 0，见表 6.21。

表 6.21　　判断格式是否有效的函数

函　数	说　明
ISDATE	确定输入表达式是否为有效日期
ISNUMERIC	确定表达式是否为有效的数值类型

（3）ISNULL：指定的替换值替换 NULL。语法格式：

ISNULL (check_expression , replacement_value)。

【例 6.26】判断重量是否为空，如果为空，则返回 50。

```
SELECT AVG(ISNULL(Weight, 50))  FROM Production.Product;
```

（4）APP_NAME。返回当前会话的应用程序名称（如果进行了设置）。

【例 6.27】在不同环境下执行 SELECT app_name()，在 SQL Management Studio 新建查询执行之后返回“Microsoft SQL Server Management Studio-查询”。而在 SQLCMD 则返回“SQLCMD”。

（5）数据库、主机、对象、登录名和用户函数。下列每对系统函数在给定标识符（ID）时返回名称，在给定名称时返回 ID。

- 数据库：DB_ID 和 DB_NAME。
- 主机：HOST_ID 和 HOST_NAME。
- 对象：OBJECT_ID 和 OBJECT_NAME，需要一个或两个参数。
- 登录名：SUSER_ID 和 SUSER_NAME（或 SUSER_SID 和 SUSER_SNAME）。
- 用户：USER_ID 和 USER_NAME。

以上除对象函数外，其余函数参数如果省略，则返回当前数据库、主机、登录名、用户。

【例 6.28】在“AdvertureWorks”数据库中，执行下列 SQL 语句。

```
SELECT DB_ID(), DB_NAME(),
       HOST_ID(),HOST_NAME(),
       SUSER_ID(),SUSER_NAME(),
       USER_ID(),USER_NAME()
```

执行结果如图 6.6 所示。

10	AdventureWorks	1060	USER-B33EAAD87F	1	sa	1	dbo

图 6.6　执行结果

【例 6.29】使用 OBJECT_ID 检查指定对象是否存在，通常用在创建对象前。

```
IF OBJECT_ID (N'dbo.AWBuildVersion', N'U') IS NOT NULL
DROP TABLE dbo.AWBuildVersion;
GO;
CREATE TABLE dbo.AWBuildVersion ... --创建表
GO
```

6.6　用户自定义函数

除了使用系统提供的函数外，用户还可以根据需要自定义函数。用户自定义函数有如下优点：模块化程序设计；执行速度更快；减少了网络流量。但是在用户自定义函数中不能更改数据，仅用于返回信息。

根据返回类型，用户自定义函数可以分为以下两种。

（1）标量函数

只返回单个数据值。函数体语句定义在 BEGIN...END 语句内，其中包含了带有返回值的 Transact-SQL 命令。返回类型可以是除 text、ntext、image、cursor 和 timestamp 外的任何数据类型。

（2）表值函数

返回 table 数据类型，可以看作一个临时表。

根据函数体 SQL 语句构成，用户自定义函数又可分为以下两种。

（1）内联函数

RETURN 子句在括号中包含单个 SELECT 语句，一般用作表值函数。RETURNS 子句只包含关键字 table，不必定义返回变量的格式，不用 BEGIN...END 包含。

（2）多语句函数

在 BEGIN...END 语句块中定义的函数体包含一系列 T-SQL 语句。

【例 6.30】下例为创建标量函数，当输入成绩时，返回成绩的等级。

```
CREATE FUNCTION dbo.GetGradeLevel(@grade int)
RETURNS char
AS
BEGIN
    DECLARE @strLevel char
    SET @strLevel = CASE
        WHEN @grade>=90 THEN 'A'
        WHEN @grade>=80 THEN 'B'
        WHEN @grade>=70 THEN 'C'
        WHEN @grade>=60 THEN 'D'
        ELSE   'e'
```

```
    END
    RETURN(@strLevel)
END
--下面为调用语句，输入 90，查看结果
SELECT dbo.GetGradeLevel(90)
```

用户自定义函数相比于存储过程，用户自定义函数比存储过程灵活，代码精简；在用户自定义函数中不能更改数据，而存储过程可以，但这并不是用户自定义函数的目的。

本 章 小 结

本章着重介绍了 Transact-SQL 语言的基本概念，包括数据类型、变量、表达式、常用函数及其使用方法，并介绍了用户自定义函数的类型和实现。

Transact-SQL 语言需要大量的实践，才能熟练运用。

习　　题

一、选择题

1．定义学生姓名，适宜使用的类型为（　　）。

A．int　　B．real　　C．VarChar(30)　　D．VarChar(MAX)

2．下列运算符号优先级最高的是（　　）。

A．AND　　B．ALL　　C．NOT

3．下面那个不是数学函数中的近似函数（　　）。

A．ROUND　　B．FIX　　C．CEILING　　D．FLOOR

二、填空题

1．整数类型包括_____、_____、_____、_____。

2．求字符串子串的函数有_____、_____、_____。

3．变量分为_____和_____。

三、简答题

1．简述数据类型 char 或 varchar 的区别，数据类型 varchar 和 nvarchar 的区别。

2．举例说明 CASE 语句两种格式。

3．聚合函数有哪些？举例说明其应用。

本 章 实 训

一、实训目的

1．掌握 Transact-SQL 语言的各个要素：数据类型、变量和常量、运算符、控制语句、常用函数。

2．熟悉用户自定义函数。

二、实训要求

1．实训前做好上机实训的准备，针对实训内容，认真复习与本次实训有关的知识，完成实训内容的预习准备工作。

2．认真独立完成实训内容。

3．实训后做好实训总结，根据实训情况完成总结报告。

三、实训学时

2 学时。

四、实训内容

1．创建一个职工表“workers”，包括以下字段：职工号（5 位数字），姓名，性别，出生日期，身高（单位：m）、入职日期，工资，部门，个人简历（大概 300 字）。试考虑每个字段所用类型，并在机器上实现。

2．写脚本查询“workers”表中年龄大于 40 的职工的平均工资。

3．以下脚本计算 1+2+3+…+100 的和，并使用 PRINT 显示计算结果。

```
DECLARE @I int，@sum int，@csum char(10)
SELECT @I=1，@sum=0
WHILE   @I<=________
    BEGIN
        SELECT @sum = __________
        SELECT @I=@I+1
    END
    SELECT @csum=convert(char(10),@sum)
    __________'1+2+3+……+100=' + @csum
```

试填空，并上机实验。

4．写脚本返回今天的日期和 100 天之后的日期。

5．上机调试例 6.32，并测试其运行结果。

五、实训思考题

1．Transact-SQL 有哪些语法要素？

2．举例说明 varchar(max)、nvarchar(max)、varbinary(max) 3 种数据类型的用途。

第 7 章

视图

如果说一张表像一个房间的话，那么视图就像是房间的窗户——即使不进入房间，也可以通过窗户看到房间里的部分布局。但是，视图又与窗户不同，通过视图不仅能看到一个表中的数据，还可以看到多个表中的数据，甚至可以通过视图更改表中的数据。

本章将介绍视图的基本概念、类型和特点，以及创建、修改视图的基本方法。通过本章的学习，读者可以了解如何使用图形化工具和 SQL 命令创建、修改视图，以及如何通过视图更改数据。

7.1 视图的作用和基本类型

视图是查看数据库中表数据的一种方式。它提供了存储预定义的查询语句作为数据库中的对象供以后使用的能力。视图是一种逻辑对象，也是一种虚拟表。除非是索引视图，否则视图不占用物理存储空间。

在视图中被查询的表称为视图的基表。大多数的 SELECT 语句都可以用在视图的创建中。视图包括以下内容。

（1）基表中列的子集或行的子集。也就是说，视图可以是基表的一部分。

（2）两个或多个基表的联合。视图是对多个基表进行联合检索的 SELECT 语句。

（3）两个或多个基表的连接。视图是通过对若干个基表的连接生成的。

（4）基表的统计汇总。视图不仅仅是基表的投影，还可以是经过对基表的各种复杂运算而得到的结果。

（5）另外一个视图的子集。视图既可以基于表，也可以基于另外一个视图。

（6）视图和基表的混合。在视图的定义中，视图和基表可以起到同样的作用。

从技术上来讲，视图是 SELECT 语句的存储定义。可以在视图中定义一个或多个表的 1 024 列，但所能定义的行数是没有限制的。

使用视图有很多优点，如使得查询简单化，提高数据的安全性，掩码数据库的复杂性以及为了向其他应用程序输出而重新组织数据等。

（1）查询简单化。如果一个查询非常复杂，跨越多个数据表，那么可以通过将这个复杂查询定义为视图，从而简化用户对数据的访问。用户只需要查询视图即可。

（2）提高数据的安全性。视图创建了一种可控的环境，即数据表中的一部分数据允许访

问，而另一部分则不允许访问。那些没有必要的、敏感的或不适合的数据都从视图中排除了，用户只能查询和修改视图中显示的数据。可以为用户只授予访问视图的权限，而不授予访问数据表的权限，从而提高数据库的安全性能。

（3）掩码数据库的复杂性。视图把数据库设计的复杂性与用户的使用方式分开。在设计数据库和数据表时，因为种种因素，通常命名都是十分复杂和难以理解，而视图可以将那些难以理解的列替换成数据库用户容易理解和接受的名称，从而为用户的使用提供极大便利。

（4）为向其他应用程序输出而重新组织数据。可以创建一个基于连接两个或多个表的复杂查询的视图，然后把视图中的数据引出到另外一个应用程序中，以便对这些数据进行进一步的分析和使用。

在 Microsoft SQL Server 2005 系统中，可以把视图分成 3 种类型，即标准视图、索引视图和分区视图。一般情况下的视图都是标准视图，它是一个虚拟表，不占用物理存储空间。如果用户希望提高多行数据的视图性能，可以创建索引视图。索引视图是被物理化的视图，它包含经过计算的物理数据，因此占用一定的存储空间。通过使用分区视图，可以连接一台或多台服务器成员表中的分区数据，使得这些数据看起来就像是来自同一个表中。

7.2　视图的创建

和表的创建类似，在 Microsoft SQL Server 2005 中，可以通过两种方式完成视图的创建：使用图形化工具和使用 SQL 命令。

7.2.1　在图形界面下创建视图

打开 Microsoft SQL Server Management Studio 环境的“对象资源管理器”（如果该窗口关闭，选中“视图”菜单，选择“对象资源管理器”选项，即可打开该窗口），打开指定的服务器实例，选中“数据库”节点，再打开指定的数据库，如“学生选课”数据库。右键单击“视图”节点，从弹出菜单中选择“新建视图”命令，则出现如图 7.1 所示的“添加表”对话框。

图 7.1　“添加表”对话框

从“添加表”对话框中选择将要定义视图的基表。一个视图可以有多个基表。选中一个表（如“Course”表）后，单击“添加”按钮，就为视图选择了一个基表。基表选择结束之后，将显示如图 7.2 所示的定义视图的对话框。

在图 7.2 所示的对话框中可以完成视图的定义操作。在基表中选择视图中要显示的列，这里选中“Course”表中的“Cname”列，“SC”表中的“Grade”列，“Student”表中的“Sname”

列、“Ssex”列和“Sdep”列。

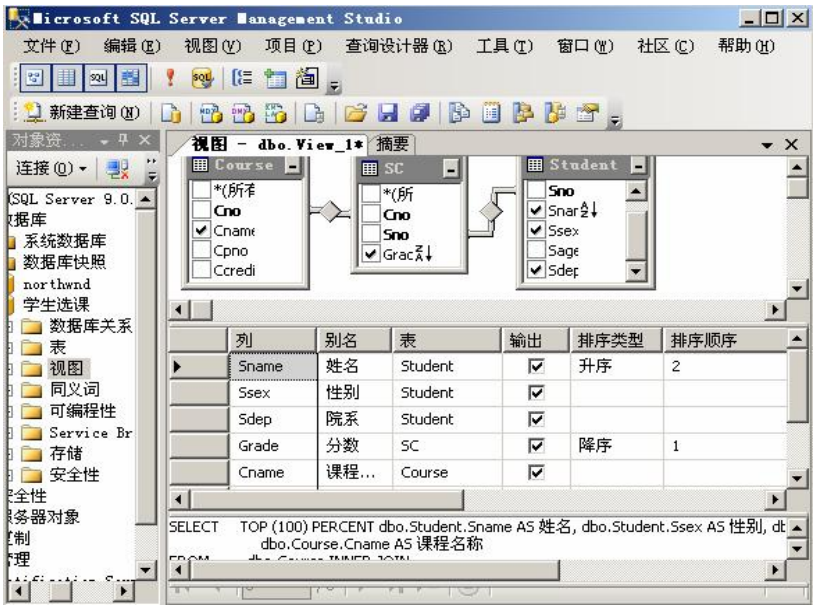


图 7.2　定义视图

在相应的列选择区可以为选中的列设置别名，排序类型，排序顺序和筛选条件等。这里为各个列取定别名，并将结果以“Grade”降序为第一排序依据，以“Sname”升序为第二排序依据。随着设置的变化，语句区的 SQL 语句也在自动更新。

单击工具栏的“保存”按钮，输入视图名称，保存视图的定义。

单击“执行”按钮，在结果区将显示视图的执行结果，如图 7.3 所示。

	姓名	性别	院系	分数	课程名称
▶	李勇	男	CS	97	计算机应用基础
	曾育山	男	CS	92	数据库原理
	刘尘	男	SS	89	计算机应用基础
	刘冰华	女	CS	80	数据库原理
	刘尘	男	SS	78	数据库原理
	秦勇	男	CS	74	高等数学
	杨尹	男	CS	67	数据库原理
	杨尹	男	CS	49	高等数学

图 7.3　视图执行结果

7.2.2　用 SQL 语句创建视图

除了使用图形化工具定义视图，还可以使用 CREATE VIEW 语句。

需要注意的是，只能在当前的数据库中创建视图，视图的名称必须符合命名规则，因为视图和表在某种程度上是类似的，因此应该使用一种能与表区别开的命名机制，使人容易分辨出表和视图。一般情况下，选择在视图名称前使用 vw_作为前缀。

使用 CREATE VIEW 语句创建视图的基本语法为：

CREATE VIEW < 视图名 > [(< 列名 > [, < 列名 > …])]

AS < 子查询 >

[WITH CHECK OPTION]

视图的内容就是子查询指定的内容。子查询可以是任意复杂的 SELECT 语句，但通常不允许含有 ORDER BY 子句。如果包含 ORDER BY 子句的话，必须同时包含 TOP 子句。

WITH CHECK OPTION 表示对视图进行 UPDATE、INSERT 和 DELETE 操作时要保证更新、插入或删除的行满足视图定义中的谓词条件（即子查询中的条件表达式）。

组成视图的属性列名要么全部省略要么全部指定。如果省略了视图的各个属性列名，则表示该视图由子查询中 SELECT 子句的目标列中的各个字段组成。但是对于下列情况，必须在视图定义中指定列的名称。

- 视图中有从算术表达式、内置函数或常量派生出的列。
- 视图中两列或多列具有相同名称（通常由于视图定义包含连接，而来自两个或多个不同表的列具有相同的名称）。
- 希望视图中的列名与它的源列名不同，这时也可以在视图中重命名列。无论重命名与否，视图列都会继承其源列的数据类型。

【例 7.1】建立计算机系（CS）学生的视图。

```
CREATE VIEW vw_CSStudent
AS
SELECT Sno, Sname, Sage
FROM Student
WHERE Sdep = 'CS'
```

本查询省略了视图“vw_CSStudent”的列名，隐含为由子查询中 SELECT 子句中的 3 个列组成。

【例 7.2】建立计算机系（CS）学生的视图，并要求进行修改和插入操作时仍需保证该视图只有 CS 系的学生。

```
CREATE VIEW vw_CSStudent
AS
SELECT Sno, Sname, Sage
FROM Student
WHERE Sdep = 'CS'
WITH CHECK OPTION
```

由于在定义“vw_CSStudent”时加上了 WITH CHECK OPTION 子句，所以以后对该视图进行插入、修改和删除操作时，系统会自动加上“Sdep = 'CS'”的条件。

视图不仅可以建立在单表上，也可以建立在多个基表上。

【例 7.3】建立所有学生选修课程及其成绩的视图。

```
CREATE VIEW vw_StuGrade1 ( Sno, Sname, Cname, Sdep, Grade )
AS
SELECT Student.Sno, Sname, Cname, Sdep, Grade
FROM Student INNER JOIN SC ON Student.Sno = SC.Sno
            INNER JOIN Course ON SC.Cno = Course.Cno
```

由于视图“vw_StuGrade1”的属性列中包含了“Student”表与“SC”表的同名列“Sno”，所以在视图名后面必须指明视图的各个属性列名，而不能像例 7.1、例 7.2 那样，将视图中的列名省略。

视图除了可以建立在一个或多个基表上，还可以建立在一个或多个已定义好的视图上，

或建立在基本表与视图上。

【例 7.4】建立选修课成绩在 85 分以上的学生的视图。

```
CREATE VIEW vw_StuGrade2
AS
SELECT Sno, Sname, Cname, Grade
FROM vw_StuGrade1
WHERE Grade > 85
```

还可以用带有聚集函数、GROUP BY 子句的查询来定义视图。

【例 7.5】为每个学生及其平均成绩建立一个视图。

```
CREATE VIEW vw_AvgGrade (Sno, Gavg)
AS
SELECT Sno, AVG ( Grade )
FROM SC
GROUP BY Sno
```

在创建视图的过程中，用户应注意以下几个问题。

- 只能在当前数据库中创建视图。但是，如果使用分布式查询定义视图，则新视图所引用的表和视图可以存在于其他数据库中，甚至其他服务器上。
- 视图名称必须遵循标识符的规则，且对每个用户必须为唯一。此外，该名称不得与该用户拥有的任何表的名称相同。
- 可以在其他视图和引用视图的过程之上建立视图。
- 定义视图的查询不可以包含 ORDER BY、COMPUTE、COMPUTE BY 子句或 INTO 关键字。
- 不能在视图上进行全文索引定义。
- 不能创建临时视图，也不能在临时表上创建视图。
- 不能删除参与到用 SCHEMABINDING 子句创建的视图中的表或视图，除非该视图已被删除或更改而不再具有架构绑定。另外，如果参与具有架构绑定的视图的表执行 ALTER TABLE 语句影响视图定义，则这些语句将失败。
- 不能对视图执行全文查询，但是如果查询所引用的表被配置为支持全文索引，就可以在视图定义中包含全文查询。

既然定义视图的查询不可以包含 ORDER BY 子句，那么如果想建立一个按照平均成绩降序排序的视图该怎么做呢？仍以例 7.5 为例，如果想让视图按照平均成绩降序输出数据，只需同时使用 ORDER BY 子句和 TOP 子句。SQL 语句如下：

```
CREATE VIEW vw_AvgGradeDesc
AS
SELECT TOP (100 ) PERCENT *
FROM vw_AvgGrade
ORDER BY Gavg DESC
```

本查询在视图“vw_AvgGrade”的基础上，建立了一个新的视图，并将学生平均成绩按照降序排列。

7.3　视图的修改

如果使用 SELECT * 语句创建了一个视图，然后又修改了基表的结构，例如增加了一个新列，则这个新列不会自动出现在该视图中。为了能在视图中看到这个新列，必须修改视图。只有对该视图经过修改之后，新增加的列才能反映到视图中。

使用 ALTER VIEW 语句可以修改视图的定义。当用该语句修改视图时，视图原有的权限不会发生变化。

例如修改例 7.3 中的视图定义，将视图中的“Sno”列去掉，并为每个列重新命名，可以用下面的 SQL 语句实现：

```
ALTER VIEW vw_StuGrade1 ( 姓名, 课程名, 所属院系, 分数 )
AS
SELECT Sname, Cname, Sdep, Grade
FROM Student INNER JOIN SC ON Student.Sno = SC.Sno
             INNER JOIN Course ON SC.Cno = Course.Cno
```

7.4　通过视图查询数据

视图定义好后，用户就可以像对基本表一样对视图进行查询了。

【例 7.6】在计算机系学生的视图中找出年龄小于 20 岁的学生。

```
SELECT Sno, Sname, Sage
FROM vw_CSStudent
WHERE Sage < 20
```

系统执行对视图的查询时，首先进行有效性检查，以确认查询中涉及到的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义，把定义好的子查询和用户的查询结合起来，转换成等价的对基本表的查询。例如，本例的查询就相当于执行了下面的 SQL 语句：

```
SELECT Sno, Sname, Sage
FROM Student
WHERE Sdep = 'CS' AND Sage < 20
```

7.5　通过视图更新数据

由于视图是不实际存储数据的虚表，因此无论在什么时候更新视图的数据，实际上都是在修改视图的基表中的数据。在利用视图更新基表中的数据时，应该注意以下几个问题。

- 创建视图的 SELECT 语句中如果包含 GROUP BY 子句，则不能修改。
- 更新基于两个或两个以上基表的视图时，每次修改数据只能影响其中的一个基表，也

就是说，不能同时修改视图所基于的两个或两个以上的数据表。

- 不能修改视图中没有定义的基表中的列。
- 不能修改通过计算得到值的列、有内置函数的列和有统计函数的列。

更新的基本操作包括插入（INSERT）、删除（DELETE）和修改（UPDATE）。

【例 7.7】向计算机学生视图“vw_CSStudent”中插入一个新的学生记录，其中学号为 20060301，姓名为白皓，年龄为 19 岁。

SQL 语句如下：

```
SET IDENTITY_INSERT Student ON;
INSERT INTO vw_CSStudent (Sno, Sname, Sage)
VALUES ( 20060301, '白皓', 19 )
```

因为“Sno”列在“Student”表中是标识列，所以先用 SET IDENTITY_INSERT 语句将该表的“IDENTITY”属性设置为可以插入数据，然后再使用 INSERT 语句（注意，此时不能省略视图列属性，必须使用列的列表指明要更新的视图中的列）进行数据的插入。

语句执行后查看“Student”表中的信息，可以发现多了一条学号为 20060301 的学生记录。

【例 7.8】将计算机系学生视图中学号为 20050304 的学生年龄改为 19。

```
UPDATE vw_CSStudent
SET Sage = 19
WHERE Sno = 20050304
```

语句执行后查看“Student”表中的信息，可以发现学号为 20050304 的学生年龄已经变为 19。

【例 7.9】删除计算机系学生视图“vw_CSStudent”中学号为 20060301 的记录。

```
DELETE FROM vw_CSStudent
WHERE Sno = 20060301
```

语句执行后查看“Student”表中的信息，可以发现学号为 20060301 的学生记录已被删除。

为了防止用户通过视图对数据进行增加、删除、修改时，有意无意地对不属于视图范围内的基本表数据进行操作，可在定义视图的时候加上 WITH CHECK OPTION 子句。这样在视图上进行增、删、改操作时，系统会检查视图定义中的条件，若不满足条件，则拒绝执行该操作。

7.6 视图的删除

可以通过执行 DROP VIEW 语句把视图的定义从数据库中删除。该语句的基本语法为：

DROP VIEW < 视图名 >

删除一个视图，就是删除其定义和赋予它的全部权限。删除基本表后，由该表导出的所有视图以及由这些视图导出的视图都将失效，但并不能自动删除，必须使用 DROP VIEW 语句显式地删除。

【例 7.10】删除视图“vw_AvgGrade”。

```
DROP VIEW vw_AvgGrade
```

执行该语句后，“vw_AvgGrade”视图的定义将从数据字典中删除。由“vw_AvgGrade”视图导出的“vw_AvgGradeDesc”视图定义虽然仍在数据字典中，但是该视图已无法使用了，因此应该同时删除：

```
DROP VIEW vw_AvgGradeDesc
```

本 章 小 结

本章介绍了视图的基本概念、类型和优点等，并通过示例详细讲述了视图的创建、修改和删除操作，以及如何通过视图完成数据的查询和更新。视图是用户更灵活地管理和使用数据库中数据的工具，通过使用视图可以更方便用户的使用查询，同时提高了系统安全性。

标准视图是一种虚拟表，它只保存了视图的定义，并不实际存储数据。对视图的管理可以通过两种方式来完成：图形化方式和 SQL 语句。

使用 SQL 语句创建视图的命令是 CREATE VIEW，修改视图的命令是 ALTER VIEW，删除视图的命令是 DELETE VIEW。通过视图查询和修改数据的过程实际上是对基本表查询和修改的过程。

本章涉及到的各种 SQL 语句和命令需要读者多加练习，并熟练掌握。

习　　题

一、选择题

1．下面关于视图特征的描述，哪些是正确的？（　　）

A．ORDER BY 子句可以出现在 CREATE VIEW 语句中。

B．ORDER BY 子句不可以出现在 CREATE VIEW 语句中。

C．GROUP BY 子句可以出现在 CREATE VIEW 语句中。

D．GROUP BY 子句不可以出现在 CREATE VIEW 语句中。

2．以下各种视图中，会占用一部分存储空间的是（　　）。

A．标准视图　　B．索引视图　　C．分区视图

3．通过视图更改数据表时，下列说法不正确的是（　　）。

A．在利用视图向数据表插入数据时，必须确保视图中没有定义的列允许空值，否则将提示出错信息。

B．通过视图，用户可以更新数据表中的任何列值。

C．用户只能访问视图中定义的字段，对于视图中没有定义的数据表中的字段，用户不能通过视图访问。

D．通过视图修改数据表时，数据表中不能包含统计函数，且 SELECT 语句中不能包含 GROUP BY 子句。

二、填空题

1．Microsoft SQL Server 2005 系统提供的视图类型包括_____、_____、_____。

2．视图是从_____中导出的表，数据库中实际存放的是视图的_____。

三、简答题

1．什么是视图？视图有什么作用？

2．什么情况下必须在视图的定义中指定列的名称？

3．视图的基表可以是哪些对象？

本 章 实 训

一、实训目的

1．理解视图的基本概念和作用。

2．掌握创建视图和修改视图的方法。

3．掌握通过视图查询和更新数据的方法。

4．掌握删除视图的方法。

二、实训要求

1．实训前做好上机实训的准备，针对实训内容，认真复习与本次实训有关的知识，完成实训内容的预习准备工作。

2．认真独立完成实训内容。

3．实训后做好实训总结，根据实训情况完成总结报告。

三、实训学时

4 学时。

四、实训内容

1．为学生基本信息建立一个视图，视图中包含学生学号、姓名、所属院系。

2．为学生选课信息建立一个视图，视图中包含学生学号、姓名、所属院系、课程名、选课分数。

3．为所有学生建立一个平均成绩视图，并按成绩降序排序。

4．在学生基本信息视图中增加“学生年龄”列。

5．向学生基本信息视图中增加一条学生记录。

6．通过学生基本信息视图将所有学生的年龄增加一岁。

7．如果要删除平均成绩低于 50 分的学生选课信息，能通过学生平均成绩视图实现吗？为什么？

8．如果要删除选课成绩为空的学生记录，能通过学生选课视图实现吗？为什么？

9．删除学生基本信息视图。

五、实训思考题

1．创建视图时，应该注意哪些问题？

2．在利用视图更新数据时，要注意哪些问题？

第 8 章

索引

本章主要介绍索引的基本概念，并详细介绍如何创建、修改和删除索引，以及如何对索引进行优化。通过本章的学习，可以了解索引的基本知识，掌握使用图形界面和代码创建、修改和删除索引的方法，以及使用数据库引擎优化顾问对索引进行优化的方法。在进行数据库编程时，建立有效的索引，可以显著提高访问数据的效率。

8.1 索 引 概 述

索引是一个单独的、物理的数据库结构，是为了加速对表中的数据行的检索而创建的一种分散存储结构。索引是针对一个表而建立的。每个索引页面中的行都含有逻辑指针，指向数据库中表的物理位置，以便加速检索物理数据。

与书的索引一样，数据库中的索引使用户可以快速找到表或索引视图中的特定信息。索引包含从表或视图中一个或多个列生成的键，以及映射到指定数据的存储位置的指针。创建设计良好的索引支持查询，可以显著提高数据库查询和应用程序的性能。索引可以减少为返回查询结果集而必须读取的数据量。索引还可以强制表中的行具有唯一性，从而确保表数据的完整性。

索引是与表或视图关联的磁盘上的结构，可以加快从表或视图中检索行的速度。索引包含由表或视图中的一列或多列生成的键。这些键存储在一个结构（B 树）中，使 SQL Server 可以快速有效地查找与键值关联的行。

表或视图可以包含以下类型的索引。

1. 聚集索引

聚集索引（clustered index，也称聚类索引、簇集索引）根据数据行的键值在表或视图中排序和存储这些数据行。索引定义中包含聚集索引列。每个表只能有一个聚集索引，因为数据行本身只能按一个顺序排序。

只有当表包含聚集索引时，表中的数据行才按照排序顺序存储。如果表具有聚集索引，则该表称为聚集表。如果表没有聚集索引，则其数据行存储在一个称为堆的无序结构中。

2. 非聚集索引

非聚集索引（nonclustered index，也称非聚类索引、非簇集索引）具有独立于数据行的结构。非聚集索引包含非聚集索引键值，并且每个键值项都有指向包含该键值的数据行的指针。

从非聚集索引中的索引行指向数据行的指针称为行定位器。行定位器的结构取决于数据

页是存储在堆中还是在聚集表中。对于堆，行定位器是指向行的指针；对于聚集表，行定位器是聚集索引键。

实际上，可以把索引理解为一种特殊的目录。下面举例来说明聚集索引和非聚集索引的区别。

其实，我们的汉语字典的正文本身就是一个聚集索引。例如，要查“安”字，就会很自然地翻开字典的前几页，因为“安”的拼音是“an”，而按照拼音排序汉字的字典是以英文字母“a”开头并以“z”结尾的，那么“安”字就排在字典的前部。如果翻完了所有以“a”开头的部分仍然找不到这个字，那么就说明该字典中没有这个字。同样，如果查“张”字，也会很自然地翻到字典最后的部分，因为“张”的拼音是“zhang”。也就是说，字典的正文部分本身就是一个目录，不需要再去查其他目录来找需要的内容。把这种正文内容本身就是按照一定规则排列的目录称为“聚集索引”。

如果认识某个字，可以快速地查到这个字。但也可能会遇到不认识的字，不知道它的发音。这时候，就不能按照刚才的方法找到要查的字，而需要根据“偏旁部首”查要找的字，然后根据这个字后的页码直接翻到某页找到要找的字。但结合“部首目录”和“检字表”查到的字的排序并不是真正的正文的排序方法，例如查“张”字，可以看到在查部首之后的检字表中“张”的页码是 672 页，检字表中“张”的上面是“驰”字，但页码却是 63 页，“张”的下面是“弩”字，页码是 390 页。很显然，这些字并不是真正的分别位于“张”字的上下方，看到的连续的“驰、张、弩”3 个字实际上就是它们在非聚集索引中的排序，是字典正文中的字在非聚集索引中的映射。可以通过这种方式来找到需要的字，但它需要两个过程：先找到目录中的结果，然后再翻到指定的页码。把这种目录纯粹是目录，正文纯粹是正文的排序方式称为“非聚集索引”。

通过以上例子，可以理解到什么是“聚集索引”和“非聚集索引”。进一步引申，每个表只能有一个聚集索引，因为目录只能按照一种方法进行排序。

聚集索引和非聚集索引都可以是唯一的。这意味着任意两行都不能有相同的索引键值。另外，索引也可以不是唯一的，即多行可以共享同一键值。

在 SQL Server 2005 中，每当修改了表数据后，都会自动维护表或视图的索引。

由上面的介绍可以看出，索引有以下优点。

（1）加快检索速度。

（2）加速对连接表查询以及执行排序或分组操作的查询。

（3）如果在创建索引时定义了唯一性，则唯一性强制执行。

（4）查询优化器依赖于索引起作用。

索引可以加速检索，但是它会消耗硬盘空间并增加开销和维护成本。索引有以下缺点。

（1）创建索引要花费时间并占有存储空间。

（2）降低数据维护的速度。在已建索引的列上维护数据时，SQL Server 会更新相关的索引。维护索引需要时间和资源。

索引设计不佳和缺少索引是影响数据库和应用程序性能的主要障碍，因此设计索引应该注意以下几个原则：

- 在高选择性的列上创建索引；
- 如果索引包含多个列，则应考虑列的顺序。用于连接或搜索条件的列放在前面，其他列基于非重复级别进行排序；
- 不要在包括大量重复数据的列上创建索引；

• 不能将 ntext、text、image、varchar(max)、nvarchar(max) 和 varbinary(max) 数据类型的列指定为索引键列。

8.2　索引的操作

8.2.1　在图形界面下创建索引

（1）打开 Microsoft SQL Server Manager Studio。

（2）在左侧的对象资源管理器中，依次展开“数据库|Northwind|表”，就可以看到已存在的表。

（3）选择要创建索引的表，如“Employees”。单击该表左侧的“+”号，然后选择索引，单击右键，在弹出菜单中选择“新建索引”命令，如图 8.1 所示。

（4）在弹出的“新建索引”对话框（如图 8.2 所示）中输入索引的名称，设置索引的类型。

（5）在对话框中单击“添加”按钮，将弹出“选择列”对话框，如图 8.3 所示。选择要添加到索引的表列，并点击“确定”按钮关闭该对话框。

（6）单击选择页中“选项”、“包含性列”和“存储”，可分别对它们的属性进行设置。然后单击“确定”按钮，将新建一个索引。

图 8.1　新建索引

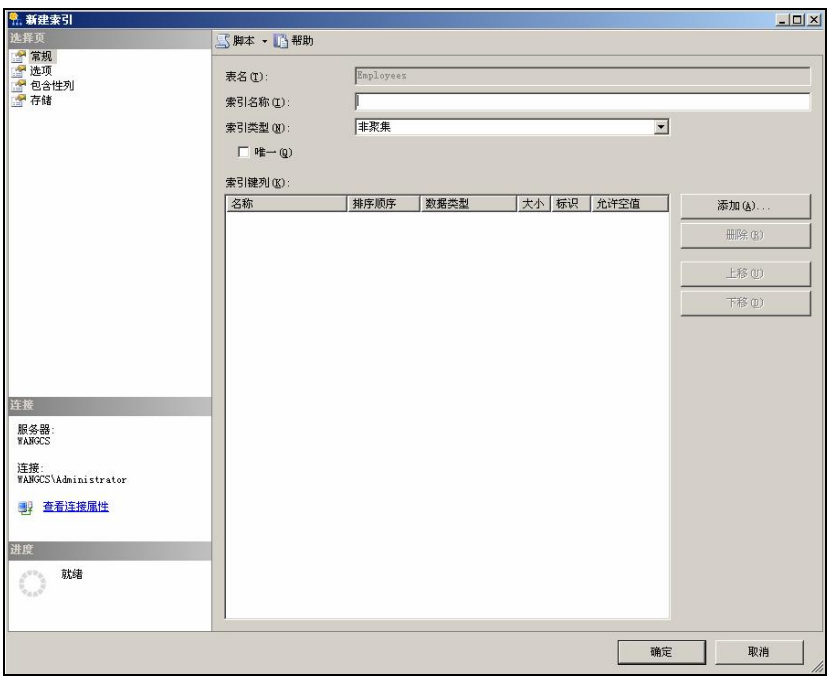


图 8.2　新建索引

图 8.3 “选择列”对话框

8.2.2 用 SQL 语句创建索引

可以使用 CREATE INDEX 语句来创建索引。创建索引时，要考虑以下事项和原则。

（1）使用 CREATE TABLE 或 ALTER TABLE 对列定义 PRIMARY KEY 或 UNIQUE 约束时，SQL Server 2005 自动创建唯一索引。和标准创建索引的方法相比，应优先考虑使用定义 PRIMARY KEY 或 UNIQUE 约束的方法来创建索引。

（2）必须是表的所有者，才能执行 CREATE INDEX 语句。

（3）尽量在较小的列上定义索引，因为较小的索引比具有较大键值的索引更有效率。

（4）根据唯一性选择列，这样每一个键值可以标识少量行。

CREATE INDEX 命令的语法格式如下：

```
CREATE [UNIQUE][CLUSTERED|NONCLUSTERED] INDEX index_name
ON {TABLE|VIEW}(column [ASC|DESC][,…n])
[INCLUDE ( column_name [ ,...n ] ) ]
```

参数说明如下。

- UNIQUE

为表或视图创建唯一索引。唯一索引不允许不同的行具有相同的索引键值。视图的聚集索引必须唯一。唯一索引中使用的列应设置为 NOT NULL，因为在创建唯一索引时，会将不同行的空值视为重复值。

- CLUSTERED

创建索引时，键值的逻辑顺序决定表中对应行的物理顺序。一个表或视图只允许有一个聚集索引。

具有唯一聚集索引的视图称为索引视图。为一个视图创建唯一聚集索引会在物理上具体化该视图。必须先为视图创建唯一聚集索引，然后才能为该视图定义其他索引。

在创建任何非聚集索引之前创建聚集索引。创建聚集索引时会重新生成表中现有的非聚集索引。

如果没有指定 CLUSTERED，则创建非聚集索引。

- NONCLUSTERED

创建一个指定表的逻辑排序的索引。对于非聚集索引，数据行的物理排序独立于索引排

序。每个表最多可包含 249 个非聚集索引。默认值为 NONCLUSTERED。

对于索引视图，只能为已定义唯一聚集索引的视图创建非聚集索引。

- index_name

索引的名称。索引名称在表或视图中必须是唯一的，但在数据库中不必唯一。索引名称必须符合标识符的规则。

- [ASC | DESC]

确定特定索引列的升序或降序排序方向。默认值为 ASC。

【例 8.1】创建唯一索引。

在“Employees”表的“EmployeeID”列创建唯一索引。

```
USE Northwind;
GO
CREATE UNIQUE INDEX IX_Employees_EmployeeID
ON dbo.Employees(EmployeeID);
GO
```

【例 8.2】创建聚集索引。

在“Employees”表的“LastName”列上创建聚集索引。

```
USE Northwind;
GO
CREATE CLUSTERED INDEX IX_LastName ON Employees(LastName);
GO
```

【例 8.3】创建简单非聚集索引。

在“Employees”表的“LastName”列上创建非聚集索引。

```
USE Northwind;
GO
CREATE NONCLUSTERED INDEX IX_LastName ON Employees(LastName);
GO
```

【例 8.4】创建简单非聚集组合索引。

在“Employees”表的“LastName”、“FirstName”列上创建非聚集组合索引。

```
USE Northwind;
GO
CREATE NONCLUSTERED INDEX IX_Employees_Name
ON dbo.Employees(LastName,FirstName);
GO
```

【例 8.5】使用 IGNORE_DUP_KEY 选项。

本示例首先在该选项设置为 ON 时，在临时表中插入多行，然后在该选项设置为 OFF 时执行相同操作，以演示 IGNORE_DUP_KEY 选项的影响。在 IGNORE_DUP_KEY 设置为 OFF 时，执行多行 INSERT 语句将导致出现重复值。表中的行计数会返回插入的行数。

```
USE Northwind;
GO
```

```
CREATE TABLE #Test (C1 int, C2 nchar(50));
GO
CREATE UNIQUE INDEX AK_Index ON #Test (C1)
    WITH (IGNORE_DUP_KEY = ON);
GO
INSERT INTO #Test VALUES (1, N'Eastern');
INSERT INTO #Test SELECT * FROM Region;
GO
SELECT COUNT(*)AS [Number of rows] FROM #Test;
GO
DROP TABLE #Test;
GO
```

下面是第二个 INSERT 语句的结果。

```
已忽略重复的键。
Number of rows
--------------
4
```

注意 从“Region”表中插入的、不违反唯一性约束的行将成功插入，会发出警告并忽略重复行，但不会回滚整个事务。

下面再次执行相同语句，但将 IGNORE_DUP_KEY 设置为 OFF。

```
USE Northwind;
GO
CREATE TABLE #Test (C1 int, C2 nchar(50));
GO
CREATE UNIQUE INDEX AK_Index ON #Test (C1)
    WITH (IGNORE_DUP_KEY = OFF);
GO
INSERT INTO #Test VALUES (1, N'Eastern');
INSERT INTO #Test SELECT * FROM Region;
GO
SELECT COUNT(*)AS [Number of rows] FROM #Test;
GO
DROP TABLE #Test;
GO
```

下面是第二个 INSERT 语句的结果。

```
消息 2601，级别 14，状态 1，第 2 行
不能在具有唯一索引“AK_Index”的对象“dbo.#Test”中插入重复键的行。
语句已终止。
```

```
Number of rows
--------------
1
```

注意　即使“Region”表中只有一行违反 UNIQUE 索引约束，也不会将其中任何一行插入该表。

【例 8.6】使用 DROP_EXISTING 删除和重新创建索引。

本示例使用 DROP_EXISTING 选项在“Products”表的“ProductID”列上删除并重新创建现有索引。

```
USE Northwind;
GO
CREATE NONCLUSTERED INDEX IX_Products_ProductID
    ON dbo.Products(ProductID)
    WITH (DROP_EXISTING = ON);
GO
```

【例 8.7】为视图创建索引。

本示例将创建一个视图并为该视图创建索引。

```
USE Northwind;
GO
IF OBJECT_ID('dbo.VEmpOrders') IS NOT NULL
  DROP VIEW dbo.VEmpOrders;
GO
CREATE VIEW dbo.VEmpOrders WITH SCHEMABINDING
AS
SELECT O.EmployeeID,
SUM(OD.Quantity) AS TotalQty,
COUNT_BIG(*) AS Cnt
FROM dbo.Orders AS O
  JOIN dbo.[Order Details] AS OD
    ON OD.OrderID = O.OrderID
GROUP BY O.EmployeeID;
GO
CREATE UNIQUE CLUSTERED INDEX idx_uc_empid ON dbo.VEmpOrders(EmployeeID);
GO
```

【例 8.8】创建具有包含性（非键）列的索引。

本示例创建具有一个键列（PostalCode）和 4 个非键列（AddressLine1、AddressLine2、City、StateProvinceID）的非聚集索引，然后执行该索引覆盖的查询。若要显示查询优化器选择的索引，执行查询前请在 SQL Server Management Studio 中的“查询”菜单上选择“显示实际执行计划”。

```
USE Northwind;
GO
CREATE NONCLUSTERED INDEX IX_Employees_PostalCode
    ON Employees (PostalCode)
    INCLUDE (Address, City, Region, Country);
GO
SELECT Address, City, Region, Country, PostalCode
FROM Employees
WHERE PostalCode BETWEEN N'98000' and N'99999';
GO
```

8.2.3 修改索引

在 Microsoft SQL Server 2005 中，可以通过使用 SQL Server Management Studio 中的 ALTER INDEX 语句或对象资源管理器执行常规索引维护任务。

ALTER INDEX 的语法如下：

```
ALTER INDEX { index_name | ALL }
  ON <object>
  { REBUILD
    [ [ WITH ( <rebuild_index_option> [ ,...n ] ) ]
     | [ PARTITION = partition_number
         [ WITH ( <single_partition_rebuild_index_option>
             [ ,...n ] )
         ]
       ]
    ]
  | DISABLE
  | REORGANIZE
    [ PARTITION = partition_number ]
    [ WITH ( LOB_COMPACTION = { ON | OFF } ) ]
  | SET ( <set_index_option> [ ,...n ] )
  }
```

1. 禁用索引

禁用索引可防止用户访问该索引。对于聚集索引，禁止索引还可防止用户访问基础表数据。禁用后，索引定义保留在元数据中，非聚集索引的索引统计信息仍保留。对视图来说，禁用非聚集索引或聚集索引，索引数据会被物理删除。禁用表的聚集索引可以防止对数据的访问，数据仍保留在表中，但在删除或重新生成索引之前，无法对这些数据执行 DML 操作。

【例 8.9】禁用索引。

```
USE Northwind;
GO
ALTER INDEX LastName ON dbo.Employees DISABLE;
GO
```

【例 8.10】启用索引。

```
USE Northwind;
GO
ALTER INDEX LastName ON dbo.Employees REBUILD;
GO
```

2. 重新组织和重新生成索引

无论何时对基础数据执行插入、更新或删除操作，SQL Server 2005 都会自动维护索引。随着操作的增多，这些修改可能会导致索引中的信息分散在数据库中（含有碎片）。碎片多的索引会降低查询性能，导致应用程序响应缓慢。可以通过重新组织索引或重新生成索引来修复索引碎片。

重新组织索引是重新进行物理排序，从而对表或视图的聚集索引和非聚集索引进行碎片整理，提高索引扫描的性能。重新生成索引将删除原索引并创建一个新索引。

【例 8.11】重新组织表“Employees”的索引“LastName”。

```
USE Northwind;
GO
ALTER INDEX LastName ON dbo.Employees REORGANIZE;
GO
```

【例 8.12】重新生成表“Employees”的索引。

```
USE Northwind;
GO
ALTER INDEX ALL ON dbo.Employees REBUILD;
GO
```

3. 重命名索引

重命名索引将用提供的新名称替换当前的索引名称。指定的名称在表或视图中必须是唯一的。

使用 sp_rename 重命名索引，语法如下：

```
sp_rename 'object_name' , 'new_name' [ ,'object_type']
```

【例 8.13】重命名索引。

```
EXEC SP_RENAME 'dbo.Employees.LastName','LName','INDEX';
```

4. 索引的维护

通过使用 SQL Server Management Studio 中的对象资源管理器执行常规索引维护任务。

首先打开 SQL Server Management Studio，在对象资源管理器中选择一个数据表。对表的所有索引进行修改操作，选择索引并单击鼠标右键，如图 8.4 所示。对单个索引进行修改操作，选择一个索引并单击右键，如图 8.5 所示。选择弹出菜单中的相应菜单命令执行修改操作。

图 8.4　修改所有索引

图 8.5　修改单个索引

8.2.4　删除索引

当一个索引不再需要时，可以将其从数据库中删除，以回收它当前使用的磁盘空间。这样数据库中的任何对象都可以使用此回收的空间。

使用 DROP INDEX 语句可以删除表或视图的索引。在删除索引时，需要考虑以下事项。

- 必须先删除 PRIMARY KEY 或 UNIQUE 约束，才能删除约束使用的索引。通过修改索引，实质上可以删除并重新创建 PRIMARY KEY 或 UNIQUE 约束使用的索引，而无需重新创建约束。
- 删除视图或表时，将自动删除视图或表中的索引。
- 在删除聚集索引时，表中的所有非聚集索引都会被自动重建。

DROP INDEX 语句的语法如下：

```
DROP INDEX index_name ON {table|view}[,…n]
```

【例 8.14】从“Employees”表中删除“LastName”索引。

```
USE Northwind;
GO
DROP INDEX LastName ON dbo.Employees;
GO
```

在创建 PRIMARY KEY 或 UNIQUE 约束时创建的索引不能使用 DROP INDEX 来删除。可以使用 ALTER TABLE DROP CONSTRAINT 语句将其删除。删除约束时，作为约束的一部分而创建的索引也将被删除。

【例 8.15】从“Employees”表中删除“PK_Employees”索引。

```
USE Northwind;
GO
ALTER TABLE dbo.Employees
DROP CONSTRAINT PK_Employees;
GO
```

8.3　索引优化向导

为数据库项目创建正确索引并不简单，需要考虑诸多因素：

- 数据库的数据模型；
- 表中数据的数量和分布；
- 对数据库执行哪些查询；
- 查询发生的频率；
- 数据更新的频率。

为了帮助用户设计索引，SQL Server 提供了一个称为“数据库引擎优化顾问”的工具。

数据库引擎优化顾问是 Microsoft SQL Server 2005 中的新工具，使用该工具可以优化数据库，提高查询处理的性能。数据库引擎优化顾问检查指定数据库中处理查询的方式，然后建议如何通过修改物理设计结构（如索引、索引视图和分区）来改善查询处理性能。

它取代了 Microsoft SQL Server 2000 中的索引优化向导，并提供了许多新增功能。例如，数据库引擎优化顾问提供了两个用户界面：图形用户界面（GUI）和 dta 命令提示实用工具。使用 GUI 可以方便快捷地查看优化结果；而使用 dta 实用工具则可以轻松地将数据库引擎优化顾问功能并入脚本中，从而实现自动优化。此外，数据库引擎优化顾问可以接受 XML 输入，该输入可对优化过程进行更多控制。

作为 SQL Server 2005 中的新增项，数据库引擎优化顾问提供了一种基于图形用户界面（GUI）的方式来查看优化会话和优化建议报表。

先打开数据库引擎优化顾问图形用户界面（GUI）。第一次使用时，必须由 sysadmin 固定服务器角色的成员来启动数据库引擎优化顾问，以初始化应用程序。初始化后，db_owner 固定数据库角色的成员便可使用数据库引擎优化顾问来优化他们拥有的数据库。

8.3.1　使用数据库引擎优化顾问 GUI

（1）打开 SQL Server Management Studio。打开“新建查询”窗口并更改数据库上下文为“Northwind”。

（2）在 SQL Server Management Studio 的查询编辑器中，输入以下 SQL 语句。将该文件保存为 MyScript.sql。

```
SELECT    Orders.CustomerID, [Order Details].OrderID
FROM    [Order Details] INNER JOIN
      Orders ON [Order Details].OrderID = Orders.OrderID
WHERE ([Order Details].ProductID = '77')
```

（3）启动数据库引擎优化顾问。在 SQL Server Management Studio 中，在“工具”菜单上选择“数据库引擎优化顾问”，或者在 Windows 界面任务栏的“开始”菜单上，依次指向“所有程序”、“Microsoft SQL Server 2005”和“性能工具”，再单击“数据库引擎优化顾问”。

（4）在“连接到服务器”对话框中，查看默认设置，再单击“连接”。默认情况下，数据库引擎优化顾问将打开如图 8.6 所示的配置。

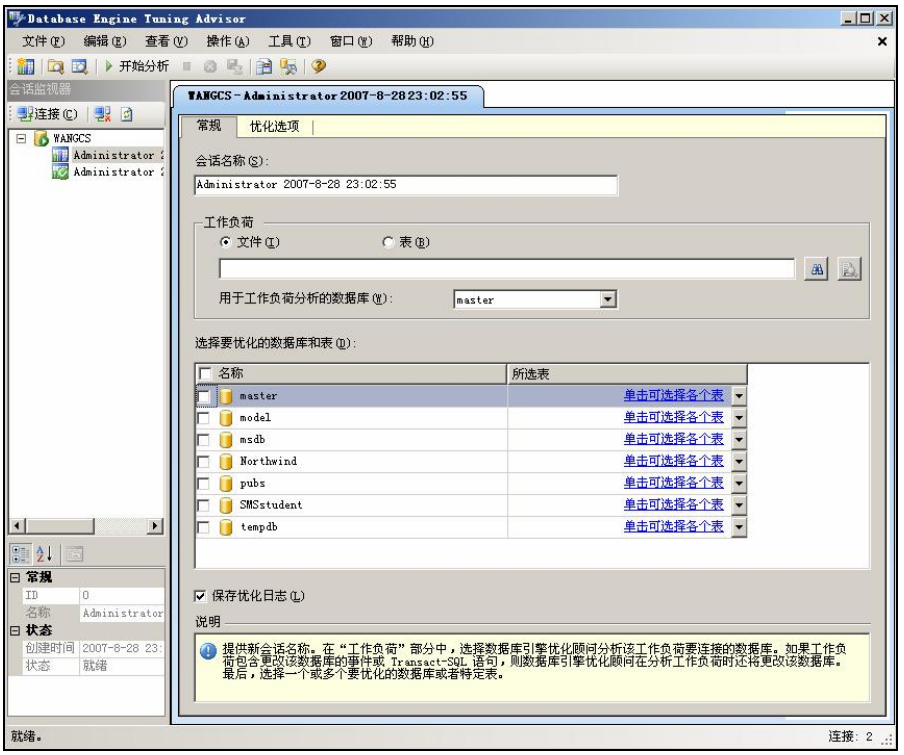


图 8.6　数据库引擎优化顾问

第一次打开时，数据库引擎优化顾问 GUI 中将显示两个主窗格。

① 左窗格包含会话监视器，其中列出已对此 Microsoft SQL Server 实例执行的所有优化会话。打开数据库引擎优化顾问时，在窗格顶部将显示一个新会话，可在相邻窗格中对此会话命名。第一次打开时，仅列出默认会话，这是数据库引擎优化顾问为用户自动创建的会话。对数据库进行优化后，连接的 SQL Server 实例的所有优化会话都将在新会话下面列出。可右键单击优化会话以对其重命名、关闭、删除或克隆。如果在列表中单击右键，则可按照名称、状态或创建时间对会话排序，或创建新会话。在此窗格的底部将显示选定优化会话的详细信息。可以选择使用“按分类顺序”按钮，以显示按类别分组的详细信息；也可使用“按字母顺序”按钮，在按字母排序的列表中显示详细信息。可以通过将右窗格边框拖动到窗口的左侧来隐藏会话监视器。若要再次查看，请将窗格边框重新拖动回右侧。利用会话监视器可以查看以前的优化会话，或使用这些会话来创建具有类似定义的新会话。还可以使用会话监视器来评估优化建议，详细信息请参阅使用会话监视器评估优化建议。

② 右窗格包含“常规”和“优化选项”选项卡。在此可以定义数据库引擎优化会话。在“常规”选项卡中，键入优化会话的名称，指定要使用的工作负荷文件或表，并选择要在该会话中优化的数据库和表。工作负荷是对要优化的一个或多个数据库执行的一组 Transact-SQL 语句。优化数据库时，数据库引擎优化顾问使用跟踪文件、跟踪表、Transact-SQL 脚本或 XML 文件作为工作负荷输入。在“优化选项”选项卡上，可以选择希望数据库引擎优化顾问在分析过程中考虑的物理数据库设计结构（索引或索引视图）和分区策略。在此选项卡上，还可以指定数据库引擎优化顾问优化工作负荷使用的最大时间，默认情况下为一个小时。

（5）在数据库引擎优化顾问 GUI 右窗格的“会话名称”中，输入“MySession”。

（6）针对“工作负荷”选择“文件”，再单击“查找工作负荷文件”按钮，以查找在步骤 1 中保存的 MyScript.sql 文件。

（7）在“用于工作负荷分析的数据库”列表中选择“Northwind”，或在“选择要优化的数据库和表”网格中选择“Northwind”，使“保存优化日志”保持选中状态。“用于工作负荷分析的数据库”指定数据库引擎优化顾问在优化工作负荷时连接到的第一个数据库。

（8）单击“优化选项”选项卡。不必为本练习设置任何优化选项，但请花些时间来查看默认的优化选项。按 F1 键可查看该选项卡的帮助。单击“高级选项”可查看其他的优化选项。在“高级优化选项”对话框中单击“帮助”，以了解有关此处所显示的优化选项的信息。单击“取消”关闭“高级优化选项”对话框，并保留选中默认选项。

（9）在工具栏上，单击“开始分析”按钮。在数据库引擎优化顾问分析工作负荷时，可以监视“进度”选项卡上的状态。优化完成后，“建议”选项卡随即显示。

（10）在“建议”选项卡上，使用页面底部的滚动条可以查看所有“索引建议”列，列出的是数据库引擎优化顾问建议删除或创建的数据库对象（索引或索引视图）。滚动到最右边的列，并单击“定义”，数据库引擎优化顾问将显示“SQL 脚本预览”窗口，如图 8.7 所示。从中可以查看创建或删除该行中的数据库对象的 Transact-SQL 脚本。单击“关闭”按钮，关闭预览窗口。

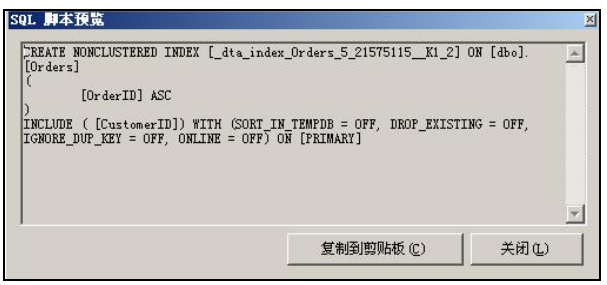


图 8.7　“SQL 脚本预览”窗口

（11）在“索引建议”窗格中右键单击网格。在出现的菜单中，可以选择或取消选择建议。还可以使用此菜单更改网格文本的字体。

选中页面底部的“显示现有对象”，可以查看“Northwind”数据库中当前存在的所有数据库对象。如果未选中此选项，则数据库引擎优化顾问将仅显示已为其生成建议的对象。使用页面右侧的滚动条可以查看所有对象。

（12）单击“操作”菜单中的“保存建议”，将所有建议保存到一个 Transact-SQL 脚本中。将脚本命名为 MySessionRecommendations.sql。

在 SQL Server Management Studio 的查询编辑器中打开 MySessionRecommendations.sql 脚本进行查看。通过在查询编辑器中执行脚本，可将建议应用于“Northwind”示例数据库。但现在不要执行该操作。不运行该脚本，直接在查询编辑器中将其关闭。

另外，也可以单击数据库引擎优化顾问“操作”菜单中的“应用建议”选项来应用建议。

（1）在数据库引擎优化顾问的“建议”选项卡上，清除“索引建议”网格中列出数据库对象的某些行。在“操作”菜单上，单击“评估建议”。数据库引擎优化顾问将创建一个新的

优化会话，从中可以评估“MySession”原有建议的子集。

（2）输入新的“会话名称”“EvaluateMySession”，然后单击工具栏上的“开始分析”按钮。可以对新的优化会话重复步骤 10 和步骤 11 以查看其建议。

（3）单击“报告”选项卡，在“优化摘要”窗格中，可以查看有关此优化会话的信息。使用滚动条可以查看所有窗格内容。请注意查看“预期的提高百分比”和“建议使用的空间”信息。

（4）在“优化报告”窗格中，单击“选择报告”列表中的“语句开销报告”。如果需要更多空间查看报表，则将“会话监视器”窗格边框拖动到左侧。对数据库中的表执行的每个 Transact-SQL 语句都会有相关的性能开销。通过对表中经常访问的列创建有效的索引，可以降低此性能开销。报告显示了在工作负荷中执行语句的原有开销与实现优化建议后的开销相比，提高百分比的估计值。


注意　报告中包含的信息量取决于工作负荷的长度和复杂性。

（5）右键单击网格区域中的“语句开销报告”窗格，再单击“导出到文件”按钮，将报告另存为 MyReport。文件名后会自动附加 .xml 扩展名。可以在常用的 XML 编辑器或 SQL Server Management Studio 中打开 MyReport.xml 以查看报告内容。

（6）单击“选择报告”列表中的其他报告，了解相关内容。

8.3.2　使用 dta 命令提示实用工具优化一个简单的工作负荷

（1）在命令提示符下，定位到存储 MyScript.sql 文件的目录。

（2）在命令提示符下，键入以下命令，然后按 Enter 键运行该命令并启动优化会话。


注意　实用工具分析命令时区分大小写：

```
dta -S YourServerName\YourSQLServerInstanceName -E -D Northwind -if MyScript.sql -s
MySession2 -of MySession2OutputScript.sql -ox MySession2Output.xml -fa IDX_IV -fp NONE -fk NONE
```

-S：指定安装了“Northwind”数据库的服务器和 SQL Server 实例的名称。

-E：指定要使用可信连接来连接实例，使用 Windows 域帐户连接时可使用该设置。

-D：指定要优化的数据库。

-if：指定工作负荷文件。

-s：指定会话名称。

-of：指定该工具要将 Transact-SQL 建议脚本写入其中的文件。

-ox：指定该工具要将 XML 格式的建议脚本写入其中的文件。

最后 3 个开关指定如下优化选项。

-fa IDX_IV：指定数据库引擎优化顾问应该只考虑添加索引（包括聚集和非聚集索引）和索引视图。

-fp NONE：指定分析时不考虑分区策略。

-fk NONE：指定数据库引擎优化顾问进行建议时不必保留数据库中现有物理设计结构。

也可以在命令提示符下键入以下命令查看完整参数信息。

dta -?

（3）数据库引擎优化顾问完成了优化工作负荷后，将显示一个消息指示优化会话已完成。若要查看优化结果，可以使用 Microsoft SQL Server Management Studio 打开 MySession2 OutputScript.sql 和 MySession2Output.xml 文件。此外，也可以在数据库引擎优化顾问 GUI 中打开“MySession2”优化会话并查看其建议和报告，执行的方式与查看优化建议和查看优化报表中执行的方式相同。

本 章 小 结

本章首先介绍了 SQL Server 2005 索引的基本概念，然后介绍了索引的创建、修改、删除等基本操作。最后介绍了使用数据库引擎优化顾问对索引进行优化，以及使用命令进行优化的方法。

习　　题

1．什么是索引？

2．索引有哪些类型？类型之间的区别是什么？

3．创建索引的目的是什么？

本 章 实 训

一、实训目的

1．掌握使用企业管理器创建并维护索引的步骤与方法。

2．掌握使用 Transact-SQL 语句创建与管理索引。

3．熟悉系统自动索引的创建。

4．理解 CREATE INDEX 选项的使用。

5．了解查询性能信息的获取方法。

二、实训要求

1．能认真独立完成实验内容。

2．实验前做好上机实验的准备，针对实验内容，认真复习与本次实验有关的知识，完成实验内容的预习准备工作。

3．验后做好实验总结，根据实验情况完成实验报告、总结报告。

三、实训学时

2 学时。

四、实训内容

分别在图形界面和查询分析器中实现如下操作。

1．在数据库“Northwind”的“Customers”表的“CompanyName”列上创建不唯一、非聚集索引“IX_CompanyName”

2．使用 DROP_EXISTING 删除和重新创建索引“IX_CompanyName”

3．删除索引“IX_CompanyName”

4．创建一个具有包含性列的索引，并对比创建前后查询执行的效率。

5．使用下面的 SELECT 语句作为工作负荷，用数据库引擎优化顾问产生一个索引优化建议，验证其索引建立的有效性。

```
SELECT Address, City, Region, Country, PostalCode
FROM Customers
WHERE PostalCode BETWEEN N'98000' and N'99999';
```

五、实训思考题

在一个表中可以有多个聚簇索引吗？为什么？

第 9 章

存储过程

本章主要介绍存储过程的基本概念，并详细介绍如何创建、修改、删除和执行存储过程，以及重命名存储过程。通过本章的学习，可以了解存储过程的基本知识，掌握使用图形界面和 T-SQL 语句创建、修改和删除存储过程的方法。

9.1 存储过程概述

在大型数据库系统中，存储过程具有很重要的作用。存储过程是 SQL 语句和流程控制语句的集合。存储过程在运行时生成执行方式，所以，以后对其再运行时其执行速度很快。SQL Server 2005 不仅提供了用户自定义存储过程的功能，而且还提供了许多可作为工具使用的系统存储过程。

9.1.1 存储过程的基本概念

存储过程（Stored Procedure）是一组为了完成特定功能的 SQL 语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。

SQL Server 支持 3 种类型的存储过程。

1. 系统存储过程

SQL Server 2005 中的许多管理活动都是通过一种特殊的存储过程执行的，这种存储过程被称为系统存储过程。例如，sys.sp_changedbowner 就是一个系统存储过程。从物理意义上讲，系统存储过程存储在源数据库中，并且带有 sp_前缀。从逻辑意义上讲，系统存储过程出现在每个系统定义数据库和用户定义数据库的 sys 构架中。在 SQL Server 2005 中，可将 GRANT、DENY 和 REVOKE 权限应用于系统存储过程。

2. 用户定义的存储过程

存储过程是指封装了可重用代码的模块或例程。存储过程可以接收输入参数，向客户端返回表格或标量结果和消息，调用 DDL 和 DML 语句，然后返回输出参数。在 SQL Server 2005 中，存储过程有两种类型：Transact-SQL 或 CLR。

Transact-SQL 存储过程是指保存了的 Transact-SQL 语句集合，可以接收和返回用户提供的参数。例如，存储过程中可能包含根据客户端应用程序提供的信息，在一个或多个表中插入新行所需的语句。存储过程也可能从数据库向客户端应用程序返回数据。例如，电子商务 Web 应用程序可能使用存储过程根据联机用户指定的搜索条件返回有关特定产品的信息。

CLR 存储过程是指对 Microsoft .NET Framework 公共语言运行时（CLR）方法的引用，可以接收和返回用户提供的参数。它在.NET Framework 程序集中是作为类的公共静态方法实现的。

3. 扩展存储过程

扩展存储过程允许使用编程语言（如 C 语言）创建自己的外部例程。扩展存储过程是指 Microsoft SQL Server 的实例可以动态加载和运行的 DLL。扩展存储过程直接在 SQL Server 实例的地址空间中运行，可以使用 SQL Server 扩展存储过程 API 进行编程。

9.1.2 存储过程的优点

存储过程有很多优点。

- 与其他应用程序共享应用程序逻辑，因而确保了数据访问和修改的一致性。存储过程可以封装业务功能，在存储过程中可以在同一位置改变封装的业务规则和策略，所有的客户端可以使用相同的存储过程来确保数据访问和修改的一致性。
- 防止数据库中表的细节暴露给用户。如果一组存储过程支持用户需要执行的所有业务功能，用户就不必直接访问表。
- 提供安全机制。即使是没有访问存储过程引用的表或视图的权限的用户，也可以被授权执行该存储过程。
- 改进性能。如果某一操作包含大量的 Transact-SQL 代码或被多次执行，那么存储过程要比批处理的执行速度快很多。因为存储过程是预编译的，在首次运行一个存储过程时，查询优化器对其进行分析、优化，并给出最终被存在系统表中的执行计划。而批处理的 Transact-SQL 语句在每次运行时都要进行编译和优化，因此速度相对要慢一些。
- 减少网络流量。用户可以通过发送一个单独的语句实现一个复杂的操作，而不需要在网络上发送大量的 Transact-SQL 代码，这样减少了在服务器和客户机之间传递的请求的数量。

9.2 存储过程的创建与执行

几乎所有可以写成批处理的 Transact-SQL 代码都可以用来创建存储过程。存储过程的设计规则包括以下内容。

- CREATE PROCEDURE 定义自身可以包括任意数量和类型的 SQL 语句，但以下语句除外。不能在存储过程的任何位置使用下面这些语句。

CREATE AGGREGATE，
CREATE DEFAULT，
CREATE 或 ALTER FUNCTION，
CREATE 或 ALTER PROCEDURE，
SET PARSEONLY，
SET SHOWPLAN_TEXT，
USE database_name。
CREATE RULE，
CREATE SCHEMA，
CREATE 或 ALTER TRIGGER，
CREATE 或 ALTER VIEW，
SET SHOWPLAN_ALL，
SET SHOWPLAN_XML，

- 数据库对象均可在存储过程中创建。可以引用在同一存储过程中创建的对象，只要引用时已经创建了该对象即可。

- 可以在存储过程内引用临时表。
- 如果在存储过程内创建本地临时表，则临时表仅为该存储过程而存在；退出该存储过程后，临时表将消失。
- 如果执行的存储过程要调用另一个存储过程，则被调用的存储过程可以访问由调用它的存储过程创建的所有对象，包括临时表在内。
- 如果执行对远程 Microsoft SQL Server 2005 实例进行更改的远程存储过程，则不能回滚这些更改。远程存储过程不参与事务处理。
- 存储过程中的参数的最大数目为 2 100。
- 存储过程中的局部变量的最大数目仅受可用内存的限制。
- 根据可用内存的不同，存储过程最大可达 128 MB。

9.2.1　在图形界面下创建存储过程

在图形界面下创建存储过程步骤如下。

（1）打开 Microsoft SQL Server Management Studio，并连接数据库。

（2）在对象资源管理器中，依次展开“数据库|Northwind|可编程性”，选中“存储过程”单击鼠标右键，选择“新建存储过程”，如图 9.1 所示。

图 9.1　新建存储过程

（3）系统将在查询编辑器中打开存储过程模版，如图 9.2 所示。

（4）在模版中输入存储过程的名称，设置相应的参数。也可以通过菜单“查询|指定模版参数的值”进行设置，如图 9.3 所示。

```
-- ================================================
-- Template generated from Template Explorer using:
-- Create Procedure (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the procedure.
-- ================================================
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
    -- Add the parameters for the stored procedure here
    <@Param1, sysname, @p1> <Datatype_For_Param1, , int> = <Default_Value_For_Param1, , 0>,
    <@Param2, sysname, @p2> <Datatype_For_Param2, , int> = <Default_Value_For_Param2, , 0>
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
END
GO
```

图 9.2　系统存储过程模版

图 9.3　指定模版参数的值

（5）“指定模版参数的值”窗口的前 3 行分别是创建人、创建时间、描述，是对存储过程进行注释。从第 4 行开始，分别指定存储过程名称、参数名称、数据类型、参数的缺省值。设置完成后，如图 9.4 所示。

图 9.4　指定参数值后的窗口

（6）单击“确定”按钮，查询编辑器中代码如下：

```
-- ============================================
-- Template generated from Template Explorer using:
-- Create Procedure (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the procedure.
-- ============================================
SET ANSI_NULLS ON
```

```
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:            张三
-- Create date: 2007-8-1
-- Description:       由订单编号获得订单详细内容
-- ============================================
CREATE PROCEDURE GetDetailsbyID
    -- Add the parameters for the stored procedure here
    @orderID int = 0,
    @p2 int = 0
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT @orderID, @p2
END
GO
```

（7）删除掉参数@p2，并编写相应的 SQL 语句。SQL 语句如下：

```
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT [Order Details].*
  FROM [Order Details]
  WHERE (OrderID = @orderID)
END
```

（8）单击工具栏上的执行按钮 执行(X) 来创建存储过程，如没有错误，消息框中则显示“命令已成功完成”。

9.2.2　用 SQL 语句创建存储过程

使用 CREATE PROCEDURE 语句可以创建存储过程。其语法如下：

```
CREATE PROC [ EDURE ] procedure_name [ ; number ]
        [ { @parameter data_type }
        [ VARYING ] [ = default ] [ OUTPUT ]
        ] [ ,...n ]
[ WITH{ RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[ FOR REPLICATION ]
AS sql_statement [ ...n ]
```

各参数含义如下。

（1）procedure_name

新存储过程的名称。过程名必须符合标识符规则，且对于数据库及其所有者必须唯一。

要创建局部临时过程，可以在 procedure_name 前面加一个编号符 (#procedure_name)；要创建全局临时过程，可以在 procedure_name 前面加两个编号符(##procedure_name)。完整的名称（包括#或##）不能超过 128 个字符。指定过程所有者的名称是可选的。

（2）;number

可选的整数，用来对同名的过程分组，以便用一条 DROP PROCEDURE 语句即可将同组的过程一起删除。例如，名为 orders 的应用程序使用的过程可以命名为 orderproc;1、orderproc;2 等。DROP PROCEDURE orderproc 语句将删除整个组。如果名称中包含定界标识符，则数字不应包含在标识符中，只在 procedure_name 前后使用适当的定界符。

（3）@parameter

过程中的参数。在 CREATE PROCEDURE 语句中可以声明一个或多个参数。用户必须在执行过程时提供每个声明参数的值（除非定义了该参数的默认值）。存储过程最多可以有 2 100 个参数。

使用@符号作为第一个字符来指定参数名称。参数名称必须符合标识符的规则。每个过程的参数仅作用于该过程本身；相同的参数名称可以用在其他过程中。默认情况下，参数只能代替常量，而不能代替表名、列名或其他数据库对象的名称。

（4）data_type

参数的数据类型。所有数据类型（包括 text、ntext 和 image）均可以用作存储过程的参数。不过，cursor 数据类型只能用于 OUTPUT 参数。如果指定的数据类型为 cursor，则必须同时使用 VARYING 和 OUTPUT 关键字。

说明：对于可以是 cursor 数据类型的输出参数，没有最大数目的限制。

（5）VARYING

指定作为输出参数支持的结果集（由存储过程动态构造，内容可以变化）。仅适用于游标参数。

（6）default

参数的默认值。如果定义了默认值，不必指定该参数的值即可执行过程。默认值必须是常量或 NULL。如果过程将对该参数使用 LIKE 关键字，那么默认值中可以包含通配符（%、_、[]和[^]）。

（7）OUTPUT

表明参数是返回参数。该选项的值可以返回给 EXEC[UTE]。使用 OUTPUT 参数可将信息返回给调用过程。text、ntext 和 image 参数可用作 OUTPUT 参数。使用 OUTPUT 关键字

的输出参数可以是游标占位符。

（8）*n*

表示最多可以指定 2 100 个参数的占位符。

（9）{RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}

RECOMPILE 表明 SQL Server 不会缓存该过程的计划，该过程将在运行时重新编译。在使用非典型值或临时值而不希望覆盖缓存在内存中的执行计划时，使用 RECOMPILE 选项。

ENCRYPTION 表示 SQL Server 加密 syscomments 表中包含 CREATE PROCEDURE 语句文本的条目。使用 ENCRYPTION 可防止将过程作为 SQL Server 复制的一部分发布。

（10）FOR REPLICATION

指定不能在订阅服务器上执行为复制创建的存储过程。使用 FOR REPLICATION 选项创建的存储过程可用作存储过程筛选，且只能在复制过程中执行。本选项不能和 WITH RECOMPILE 选项一起使用。

（11）AS

指定过程要执行的操作。

（12）sql_statement

过程中要包含的任意数目和类型的 Transact-SQL 语句，但有一些限制。

（13）*n*

表示此过程可以包含多条 Transact-SQL 语句的占位符。

注意　存储过程的最大容量为 128MB。

【例 9.1】只返回单一记录集的存储过程。

查询表“Categories”的内容的存储过程。

```
USE Northwind;
GO
CREATE PROC GETCATEGORIES
AS
SELECT * FROM Categories
```

【例 9.2】在存储过程中使用输入参数。

创建名为“SalesbyYear”的存储过程，返回在两个指定日期之间的所有销售额。

```
USE Northwind;
GO
CREATE procedure SalesbyYear
        @Beginning_Date DateTime,
        @Ending_Date DateTime
AS
IF @Beginning_Date IS NULL OR @Ending_Date IS NULL
BEGIN
  RAISERROR('NULL values are not allowed',14,1)
  RETURN
```

```
END
SELECT Orders.ShippedDate,
        Orders.OrderID,
        "Order Subtotals".Subtotal,
        DATENAME(yy,ShippedDate) AS Year
FROM Orders INNER JOIN "Order Subtotals"
  ON Orders.OrderID = "Order Subtotals".OrderID
WHERE Orders.ShippedDate Between @Beginning_Date And @Ending_Date;
GO
```

【例 9.3】在存储过程中使用输出参数。

创建 CountProductsInCategory 存储过程。此过程有一个输入参数@CatID 和一个输出参数@CatName，并返回该分类中产品的个数。

```
USE Northwind;
GO
CREATE PROCEDURE dbo.CountProductsInCategory
    @CatID int,
    @CatName nvarchar(15) OUTPUT
AS
  DECLARE @ProdCount int
  SELECT @CatName = Categories.CategoryName,
        @ProdCount = COUNT(Products.ProductID)
  FROM Categories INNER JOIN Products
    ON Categories.CategoryID = Products.CategoryID
  WHERE (Categories.CategoryID = @CatID)
  GROUP BY Categories.CategoryName
  RETURN @ProdCount;
GO
```

【例 9.4】使用带有通配符参数的简单过程。

以下存储过程只从视图中返回指定的一些雇员（提供名和姓）及其职务和部门名称。此存储过程模式与所传递的参数相匹配；或者，如果未提供参数，则使用预设的默认值（以字母 D 打头的姓）。

```
USE Northwind;
GO
CREATE PROCEDURE usp_GetEmployees
    @lastname varchar(40) = 'D%',
    @firstname varchar(20) = '%'
AS
    SELECT LastName, FirstName, Title, Address,City
    FROM dbo.Employees
```

```
    WHERE FirstName LIKE @firstname
        AND LastName LIKE @lastname;
GO
```

【例 9.5】使用 WITH RECOMPILE 选项。

如果为过程提供的参数不是典型的参数，并且新的执行计划不应被缓存或存储在内存中，则 WITH RECOMPILE 子句会很有用。

```
USE Northwind;
GO
CREATE PROCEDURE dbo.GetOrders
  @odate AS DATETIME
WITH RECOMPILE
AS
SELECT OrderID, CustomerID, EmployeeID, OrderDate
FROM dbo.Orders
WHERE OrderDate >= @odate;
GO
```

【例 9.6】使用 WITH ENCRYPTION 选项。

使用 WITH ENCRYPTION 选项可阻止返回存储过程的定义。

```
USE Northwind;
GO
CREATE PROCEDURE dbo.usp_encrypt_this
WITH ENCRYPTION
AS
    SELECT EmployeeID, LastName, FirstName, Title,
        TitleOfCourtesy, BirthDate
    FROM Employees;
GO
```

【例 9.7】典型示例。

以下示例创建一个 usp_userLogin 存储过程，该过程根据输入的 Loginid 和 Password，判断是否为一个合法用户，并返回错误原因。

```
USE Northwind;
GO
CREATE TABLE userLogin
(loginid varchar(50),
 username varchar(20),
 password varchar(20),
 allowlogin bit);
GO
CREATE PROCEDURE [dbo].[usp_userLogin]
```

```
    @loginid varchar(50),
    @password varchar(50),
    @reason varchar(50) output
AS
    select username from userLogin where    LoginID = @loginid
    if (@@RowCount<1)
        begin
            set    @reason ='不存在此用户'
        end
    else
    begin
        SELECT username
        FROM userLogin
        WHERE (LoginID = @loginid) AND (Password = @password )
        if (@@RowCount<1)
            begin
                set    @reason ='口令错误'
            end
        else
            begin
                SELECT username
                        FROM userLogin
                    WHERE (LoginID = @loginid) AND
                            (Password = @password and AllowLogin=1)
                if (@@RowCount<1)
                        begin
                            set    @reason ='该用户已禁用'
                        end
                else
                    begin
                        set    @reason ='成功'
                    end
                end
            end
    RETURN
GO
```

9.2.3 存储过程的执行

建立一个存储过程以后，可以使用 EXECUTE 语句来执行这个存储过程。EXECUTE 语

句的语法如下：

```
[ { EXEC | EXECUTE } ]
  {
   [ @return_status = ]
   { procedure_name [ ;number ] | @procedure_name_var }
    [ [ @parameter = ] { value | @variable [ OUTPUT ]|[ DEFAULT ]}]
   [ ,...n ]
   [ WITH RECOMPILE ]
  }
```

在这个语法格式中，@return_status 用于保存存储过程的返回状态。使用 EXECUTE 语句之前，这个变量必须在批处理、存储过程或函数中声明过。当执行与其他同名存储过程处于同一分组中的存储过程时，应当指定此存储过程在组内的标识号。@参数名给出在 CREATE PROCEDURE 语句中定义的过程参数。在以“@参数名＝值”格式使用时，参数名称和常量不一定按照 CREATE PROCEDURE 语句中定义的顺序出现。@变量是用来保存参数或者返回参数的变量。OUTPUT 关键字指定存储过程必须返回一个参数。DEFAULT 关键字用于提供参数的默认值。WITH RECOMPILE 子句指定强制编译新的计划，建议尽量少使用该选项，因为它会消耗较多的系统资源。

使用 EXECUTE 语句时应注意以下几点。

（1）EXECUTE 语句可以用于执行系统存储过程、用户定义存储过程或扩展存储过程，同时支持 Transact-SQL 批处理内的字符串的执行。

（2）如果 EXECUTE 语句是批处理的第一条语句，那么省略 EXECUTE 关键字也可以执行该存储过程。

（3）向存储过程传递参数时，如果使用“@参数＝值”的形式，则可以按任意顺序来提供参数，还可以省略那些已经提供默认值的参数。一旦以“@参数＝值”形式提供了一个参数，就必须按这种形式提供后面所有的参数。如果不是以“@参数＝值”形式来提供参数，则必须按照 CREATE PROCEDURE 语句中给出的顺序提供参数。

（4）虽然可以省略已提供默认值的参数，但只能截断参数列表。例如，如果一个存储过程有 5 个参数，则可以省略第 4 个和第 5 个参数，但不能跳过第 4 个参数而仍然包含第 5 个参数，除非以“@参数=值”形式提供参数。

（5）如果在建立存储过程时定义了参数的默认值，那么下列情况下将使用默认值：执行存储过程时未指定该参数的值；将 DEFAULT 关键字指定为该参数的值。

（6）如果在存储过程中使用了带 Like 关键字的参数名称，则提供的默认值必须是常量，并且可以包含%、_、[]、[^]通配符。

【例 9.8】执行简单存储过程。

执行例 9.1 创建的存储过程“GETCATEGORIES”。

```
USE Northwind;
GO
EXECUTE dbo.GETCATEGORIES;
GO
```

【例 9.9】通过参数名传递值。

执行例 9.2 创建的存储过程“SalesbyYear”。

```
USE Northwind;
GO
EXECUTE dbo.SalesbyYear
      @Beginning_Date='1998-5-1',
      @Ending_Date='1998-5-6';
GO
```

【例 9.10】通过定位传递值。

执行例 9.2 创建的存储过程“SalesbyYear”。

```
USE Northwind;
GO
EXECUTE dbo.SalesbyYear '1998-5-1','1998-5-6';
GO
```

【例 9.11】使用输出参数返回值。

执行例 9.3 创建的存储过程“CountProductsInCategory”。

```
USE Northwind;
GO
DECLARE @ProductCount int,
            @CategoryName nvarchar(15)
EXECUTE @ProductCount=dbo.CountProductsInCategory '1',@CategoryName OUT
print N'该编号对应的分类名称是'+cast(@CategoryName as varchar(20))+
N', 共包括'+cast(@ProductCount as varchar(20))+N'中产品。'
GO
```

【例 9.12】执行带有通配符参数的存储过程。

执行例 9.4 创建的存储过程“usp_GetEmployees”。不指定参数，则使用默认参数值执行。

```
USE Northwind;
GO
EXECUTE usp_GetEmployees
GO
```

结果如下：

```
LastName     FirstName   Title                    Address              City
---------- --------- --------------------   -----------------        -------
Davolio        Nancy       Sales Representative   507-20th Ave.E.Apt.2A   Seattle
Dodsworth   Anne          Sales Representative   Houndstooth Rd.          London
```

指定参数值，第一个参数使用“%”，第二个参数使用“A%”。

```
USE Northwind;
GO
EXECUTE usp_GetEmployees '%','A%'
```

```
GO
```

结果如下：

```
LastName    FirstName    Title                       Address              City
--------- ---------- -------------------- ------------------ -------- --------------- ---------------
Fuller      Andrew       Vice President, Sales 908 W. Capital Way    Tacoma
Dodsworth Anne           Sales Representative    7 Houndstooth Rd.    London
```

9.3　修改存储过程

如果需要更改存储过程中的语句或参数，可以删除并重新创建该存储过程，也可以更改该存储过程。删除并重新创建存储过程时，与该存储过程关联的所有权限都将丢失。更改存储过程时，将更改过程或参数定义，但为该存储过程定义的权限将保留，并且不会影响任何相关的存储过程或触发器。还可以修改存储过程以加密其定义或使该过程在每次执行时都重新编译。

可以使用 ALTER PROCEDURE 语句修改存储过程，其语法如下：

```
ALTER PROC[EDURE] procedure_name [;number]
[ { @parameter data_type }
[ VARYING ] [ = default ] [ OUTPUT ]
] [ ,...n ]
[ WITH{ RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[ FOR REPLICATION ]
AS sql_statement [ ...n ]
```

【例 9.13】修改存储过程。

修改存储过程“SalesbyYear”，使返回的结果集按照“Orders.ShippedDate”进行排序。

```
USE Northwind;
GO
ALTER procedure [dbo].[SalesbyYear]
      @Beginning_Date DateTime, @Ending_Date DateTime
AS
IF @Beginning_Date IS NULL OR @Ending_Date IS NULL
BEGIN
   RAISERROR('NULL values are not allowed',14,1)
   RETURN
END
SELECT Orders.ShippedDate,
        Orders.OrderID,
        "Order Subtotals".Subtotal,
        DATENAME(yy,ShippedDate) AS Year
FROM Orders INNER JOIN "Order Subtotals"
```

```
    ON Orders.OrderID = "Order Subtotals".OrderID
WHERE Orders.ShippedDate Between @Beginning_Date And @Ending_Date
Order By Orders.ShippedDate;
GO
```

9.4 重命名存储过程

在 Microsoft SQL Server Management Studio 的对象资源管理器中重命名存储过程很简单，选择要重命名的存储过程，单击右键，在弹出的菜单中选择“重命名”命令，就可以修改了。

使用系统存储过程 sp_rename 也可以重命名存储过程。其语法如下：

```
sp_rename 'object_name' , 'new_name' [ ,'object_type' ]
```

【例 9.14】将“Northwind”数据库中存储过程“SalesByCategory”重命名为“NewName”。

```
USE Northwind;
GO
EXEC sp_rename 'dbo.SalesByCategory','NewName','PROCEDURE';
GO
```

建议不要使用此语句来重命名存储过程，而应当先删除该对象，然后使用新名称重新创建该对象。

9.5 删除存储过程

不再需要存储过程时可将其删除。如果另一个存储过程调用某个已被删除的存储过程，Microsoft SQL Server 2005 将在执行调用过程时显示一条错误消息。但是，如果定义了具有相同名称和参数的新存储过程来替换已被删除的存储过程，那么引用该过程的其他过程仍能成功执行。例如，如果存储过程“proc1”引用存储过程“proc2”，而“proc2”已被删除，但又创建了另一个名为“proc2”的存储过程，现在“proc1”将引用这一新存储过程。“proc1”也不必重新创建。

使用 DROP PROCEDURE 语句来删除用户定义的存储过程。其语法如下：

```
DROP PROCEDURE {procedure}[,…n]
```

【例 9.15】将“Northwind”数据库中存储过程“SalesByCategory”删除。

```
USE Northwind;
GO
DROP PROCEDURE dbo.SalesByCategory
GO
```

本 章 小 结

本章主要介绍存储过程的基本概念，并详细介绍如何创建、修改、删除和执行存储过程，

以及重命名存储过程。通过本章的学习，可以了解存储过程的基本知识，掌握使用图形界面和代码创建、修改和删除存储过程的方法。

习　　题

一、填空题

1．存储过程是_____，可分为 3 类，分别是_____，_____，_____。

2．创建存储过程的命令是_____。

二、简答题

1．简述存储过程的优点。

2．简述重命名存储过程的方法。

本 章 实 训

一、实训目的

1．掌握使用向导创建存储过程并更新相应数据。

2．掌握使用 Transact-SQL 语句创建一个存储过程并验证。

3．掌握创建和执行带参数的存储过程。

4．熟练使用系统存储过程、系统函数和企业管理器查看存储过程的定义。

二、实训要求

1．实训前做好上机实训的准备，针对实训内容，认真复习与本次实训有关的知识，完成实训内容的预习准备工作。

2．能认真独立完成实训内容。

3．实训后做好实训总结，根据实训情况完成总结报告。

三、实训学时

2 学时。

四、实训内容

1．创建存储过程“OverdueOrders”，列出“Northwind”数据库中过期未付的订单。

2．创建一个名为“AddCustomer”的存储过程，该存储过程向“Northwind”数据库中的“Customers”表插入雇员信息。

3．创建一个名为“AdjustPrice”的存储过程，该存储过程实现对“NorthWind”数据库中的“Products”表中产品的价格进行调整。

4．分别执行“AddCustomer”、“AdjustPrice”两个存储过程，参数的传递分别使用按参数名传递和按位传递两种方法。

5．使用“AlterProcedure”修改“OverdueOrders”存储过程，选择部分列，并按照“RequiredDate”列进行升序排序。

6．使用 sp_rename 将存储过程“AdjustPrice”重命名为“ModifyProductPrice”。

7．删除“AddCustomer”存储过程。

8．使用 sp_help、sp_helptext、sp_depends 分别查看存储过程的信息。

9．修改“OverdueOrders”，使用 WITH ENCRYPTION 选项进行加密，使用 sp_helptext 查看该存储过程。

五、实训思考题

1．存储过程的类型有哪些？分别有什么特征？

2．如何创建一个存储过程？试述存储过程在程序设计中的作用。

3．假设必须修改数据库中的一个存储过程，同时有几个用户被授予执行这个存储过程的权限，请问执行哪个语句可以实现修改，但又不影响现有的权限？

第 10 章 触发器和游标

触发器实际上是一种特殊类型的存储过程，它是在执行某些特定的 T-SQL 语句时可以自动执行的一种存储过程。游标是一种处理数据的方法，它可对结果集中的记录进行逐行处理，提供了一种数据操作的灵活手段。

本章主要介绍触发器的特点、作用，触发器的创建、修改和删除；游标的特点、作用以及游标的声明、打开、关闭和释放等内容。

10.1 触发器概述

10.1.1 触发器的概念

触发器实际上就是一种特殊类型的存储过程，它在执行到一定操作时自动触发执行。在 SQL Server 2005 之前的版本中，触发器是针对数据表的特殊的存储过程，当这个表发生了 INSERT、UPDATE 或 DELETE 操作时，如果该表有对应操作的触发器，这个触发器就会自动激活执行。在 SQL Server 2005 中，触发器有了更进一步的功能，在数据表（库）发生 CREATE、ALTER 和 DROP 操作时，如果有对应操作的触发器，也会自动激活执行。

10.1.2 触发器的功能

SQL Server 2005 提供了两种方法来保证数据的有效性和完整性：约束（CHECK）和触发器（TRIGGER）。约束直接设置于数据表内，只能实现一些比较简单的功能操作，如实现字段有效性和唯一性的检查，自动填入默认值，确保字段数据不重复（即主键），确保数据表对应的完整性（即外键）等功能；触发器是针对数据表（库）的特殊的存储过程，它在指定的表中的数据（或表结构）发生改变时自动生效，并可以包含复杂的 T-SQL 语句，用于处理各种复杂的操作。将触发器和触发它的语句作为可在触发器内回滚的单个事务对待。如果检测到错误（如磁盘空间不足），则整个事务自动回滚。

触发器的常用功能如下。

（1）完成更复杂的数据约束：触发器可以实现比约束更为复杂的数据约束。例如：CHECK 约束只能根据逻辑表达式或同一个表中的另一列来验证列值，如果应用程序要求根据另一个表中

的列来验证列值，则必须使用触发器。

（2）检查 SQL 所做的操作是否允许：触发器可以检查 SQL 所做的操作是否被允许。例如：在学校班级表里，如果要删除一条班级记录，在删除记录时，触发器可以检查该班级的学生人数是否为 0，如果不为 0 则取消该删除操作。

（3）修改其他数据表里的数据：当一个 SQL 语句对数据表进行操作的时候，触发器可以根据该 SQL 语句的操作情况来对另一个数据表进行操作。例如：修改某一个学生的某一门课程的成绩时，触发器可以自动修改学生总成绩表中该学生的总成绩。

（4）调用更多的存储过程：约束是不能调用存储过程的，但触发器本身就是一种存储过程，而存储过程是可以嵌套使用的，所以触发器也可以调用一个或多个存储过程。

（5）返回自定义的错误信息：约束只能通过标准的系统错误信息来传递错误信息，如果应用程序要求使用（或能从中获益）自定义信息和较为复杂的错误处理，则必须使用触发器。

（6）更改原本要操作的 SQL 语句：触发器可以修改原本要操作的 SQL 语句。例如，原来的 SQL 语句是要删除数据表里的记录，但该数据表里的记录是重要记录，不允许删除的，那么触发器可以不执行该语句。

（7）防止数据表结构被更改或数据表被删除：为了保护已经建好的数据表，触发器可以在接收到以 DROP 或 ALTER 开头的 SQL 语句后，不对数据表结构做任何操作。

10.1.3 触发器的类型

在 SQL Server 2005 中，根据激活触发器执行的 T-SQL 语句类型，可以把触发器分为两类：一类是 DML 触发器，一类是 DLL 触发器。

1. DML 触发器

DML 触发器是当数据库服务器中发生数据操纵语言（DML）事件时执行的存储过程。

2. DDL 触发器

DDL 触发器是在响应数据定义语言（DDL）事件时执行的存储过程。DDL 触发器一般用于执行数据库中的管理任务，如审核和规范数据库操作，防止数据库表结构被修改等。

10.2 DML 触发器

10.2.1 DML 触发器的类型

在 SQL Server 2005 中，根据触发的时机可以把 DML 触发器划分为两种类型。

1. AFTER 触发器

这类触发器是在记录已经改变之后，才会被激活执行，它主要是用于记录变更后的处理或检查，一旦发现错误，也可以用 ROLLBACK TRANSACTION 语句来回滚本次的操作。

2. INSTEAD OF 触发器

这类触发器一般是用来取代原本要进行的操作，在记录变更之前发生的，它并不去执行原来 SQL 语句里的操作（INSERT、UPDATE、DELETE），而是去执行触发器本身所定义的操作。

10.2.2　DML 触发器的工作原理

在 SQL Server 2005 里，为每个 DML 触发器都定义了两个特殊的表，一个是 Inserted 表，一个是 Deleted 表。这两个表是建在数据库服务器的内存中的，是由系统管理的逻辑表，而不是真正存储在数据库中的物理表。对于这两个表，用户只有读取的权限，没有修改的权限。

这两个表的结构与触发器所在数据表的结构是完全一致的，当触发器的工作完成之后，这两个表也将从内存中删除。

Inserted 表里存放的是更新前的记录：对于插入记录操作来说，Inserted 表里存放的是要插入的数据；对于更新记录操作来说，Inserted 表里存放的是要更新的记录。

Deleted 表里存放的是更新后的记录：对于更新记录操作来说，Deleted 表里存放的是更新前的记录；对于删除记录操作来说，Deleted 表里存入的是被删除的旧记录。

下面来看一下 DML 触发器的工作原理。

1. AFTER 触发器的工作原理

AFTER 触发器是在记录变更之后才被激活执行的。以删除记录为例，当 SQL Server 接收到一个要执行删除操作的 SQL 语句时，SQL Server 先将要删除的记录存放在删除表里，然后把数据表里的记录删除，再激活 AFTER 触发器，执行 AFTER 触发器里的 SQL 语句。执行完毕之后，删除内存中的删除表，退出整个操作。

2. INSTEAD OF 触发器的工作原理

INSTEAD OF 触发器与 AFTER 触发器不同。AFTER 触发器是在 INSERT、UPDATE 和 DELETE 操作完成后才激活的，而 INSTEAD OF 触发器，是在这些操作进行之前就激活了，并且不再去执行原来的 SQL 操作，而去运行触发器本身的 SQL 语句。

10.2.3　创建 DML 触发器的注意事项

（1）CREATE TRIGGER 语句必须是批处理中的第一个语句，该语句后面的所有语句都被解释为 CREATE TRIGGER 语句定义的一部分。

（2）创建 DML 触发器的权限默认分配给表的所有者，且不能将该权限转给其他用户。

（3）DML 触发器为数据库对象，其名称必须遵循标识符的命名规则。

（4）虽然 DML 触发器可以引用当前数据库以外的对象，但只能在当前数据库中创建 DML 触发器。

（5）虽然 DML 触发器可以引用临时表，但不能对临时表或系统表创建 DML 触发器。不应引用系统表，而应使用信息架构视图。

（6）对于含有用 DELETE 或 UPDATE 操作定义的外键的表，不能定义 INSTEAD OF DELETE 和 INSTEAD OF UPDATE 触发器。

（7）虽然 TRUNCATE TABLE 语句类似于不带 WHERE 子句的 DELETE 语句（用于删除所有行），但它并不会触发 DELETE 触发器，因为 TRUNCATE TABLE 语句没有记录。

（8）WRITETEXT 语句不会触发 INSERT 或 UPDATE 触发器。

10.2.4　创建 AFTER 触发器

下面通过实例来说明如何创建一个简单的触发器。该触发器的作用是：在学生表中插入

一条记录后，发出“你已经成功添加了一个学生信息”的提示。分析该触发器的作用不难发现，要建立的触发器类型是：AFTER INSERT 类型。

1. 创建 AFTER 触发器的步骤

（1）启动“SQL Server Management Studio”，在“对象资源管理器”下选择“数据库”，定位到“student”数据库，展开其下的“表”树型目录，找到“dbo.学生表”，并选中其下的“触发器”项，如图 10.1 所示。

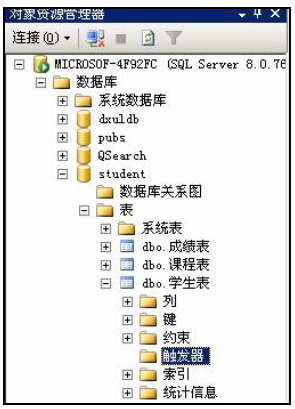


图 10.1　在“SQL ServerManagement Studio”中定位到“触发器”

（2）右键单击“触发器”，在弹出的快捷菜单中选择“新建触发器”选项，弹出“查询编辑器”对话框。在“查询编辑器”的编辑区里，SQL Server 已经预写入了一些建立触发器相关的 SQL 语句，如图 10.2 所示。

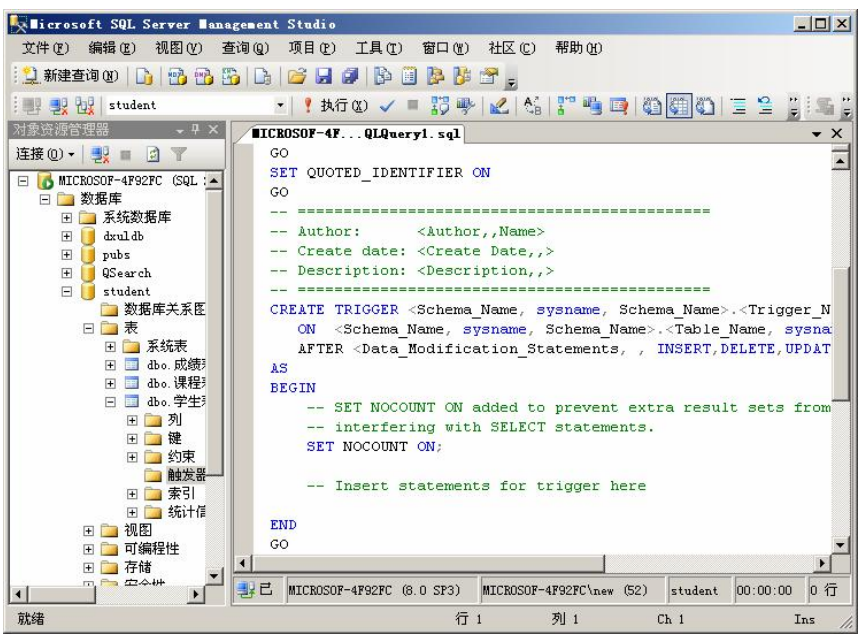


图 10.2　建立触发器的“查询编辑器”对话框

（3）修改“查询编辑器”里的代码，将从“CREATE”开始到“GO”结束的代码改为以下代码：

```
CREATE TRIGGER  学生_Insert
        ON  学生表
        AFTER INSERT
AS
BEGIN
    print '你已经成功添加了一个学生信息'
END
GO
```

（4）单击工具栏中的“分析”按钮，检查一下语法是否有错，如图 10.3 所示。如果在下面的“结果”对话框中出现“命令已成功完成”，则表示没有语法错误。

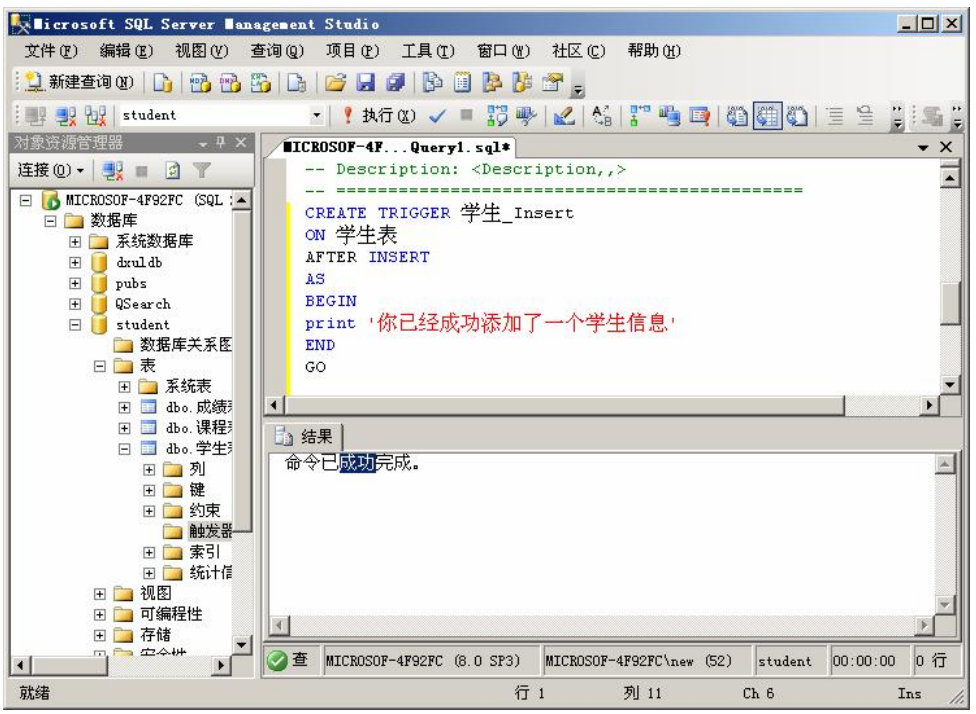


图 10.3　检查语法

（5）语法检查无误后，单击“执行”按钮，生成触发器。

（6）关掉“查询编辑器”对话框，返回到如图 10.1 所示的界面，接着单击右键，选择“刷新”菜单项，然后展开“触发器”，可以看到刚才建立的“学生_Insert”触发器，如图 10.4 所示。

图 10.4　查看建好的触发器

建立 AFTER UPDATE 触发器、AFTER DELETE 触发器和建立 AFTER INSERT 触发器的步骤一致。只要把上面的 SQL 语句中的 INSERT 分别改为 UPDATE 和 DELETE 即可，如下所示。

```
--建立 AFTER UPDATE 类型的触发器
CREATE TRIGGER  学生_UPDATE
    ON  学生表
    AFTER UPDATE
AS
BEGIN
    print '你已经成功修改了一个学生信息'
END
GO
--建立 AFTER DELETE 类型的触发器
CREATE TRIGGER  学生_DELETE
    ON  学生表
    AFTER DELETE
AS
BEGIN
    print '你已经成功删除了一个学生信息'
END
GO
```

可以把上述两段代码复制到“查询编辑器”里面，进行测试。

2. 测试触发器功能

触发器创建后，能否正常工作需要进行测试。下面就来测试一下刚建好的 AFTER INSERT 触发器的功能。

（1）在“Management Studio”里新建一个查询，在弹出的“查询编辑器”对话框里输入以下代码：

```
INSERT INTO 学生表(sno,sname) VALUES('200701001','张三')
```

（2）单击“执行”按钮，可以看到“消息”对话框里显示出提示信息：“你已经成功添加了一个学生信息”，如图 10.5 所示。可以看到，刚建好的 AFTER INSERT 触发器已经被激活，并运行成功了。

如果在“查询编辑器”里执行的不是一个 INSERT 语句，而是一个 UPDATE 或 DELETE 语句的话，AFTER INSERT 触发器将不会被激活。在“查询编辑器”中输入以下语句：

```
UPDATE  学生表 SET sname = '李四' where sno = '200701001'
```

单击“执行”按钮，在“消息”对话框里只显示了“(1 行受影响)”的提示，而没有“你已经成功添加了一个学生信息”的提示，如图 10.6 所示，说明 UPDATE 语句不能激活 AFTER INSERT 触发器。

DELETE 语句也不能激活 AFTER INSERT 触发器，可以把以下代码复制到“查询编辑器”中进行测试。

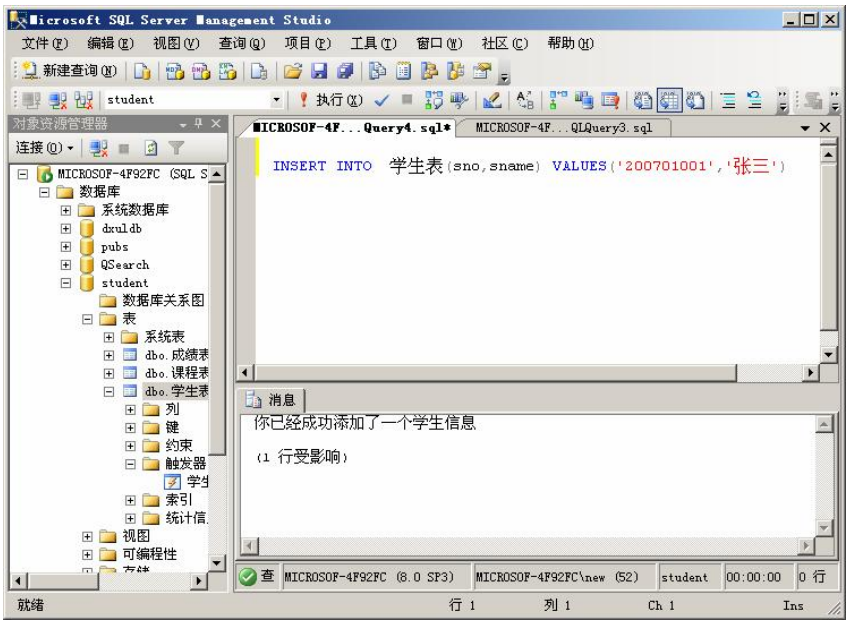


图 10.5　触发器运行结果

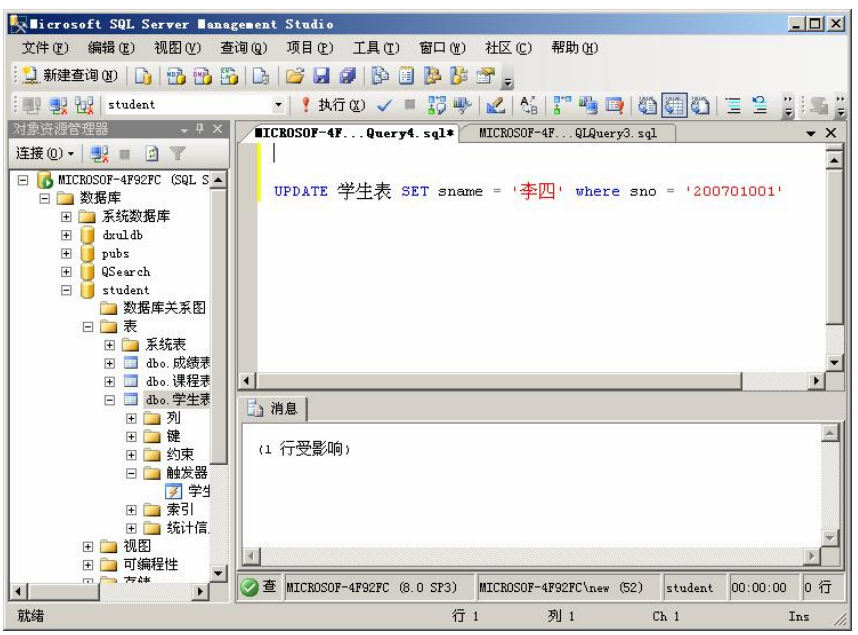


图 10.6　测试 UPDATE 语句能否激活 AFTER INSERT 触发器的运行结果

```
DELETE FROM 学生表 where sno = '200701001'
```

3. 创建 AFTER 触发器的 T-SQL 语句

创建 AFTER 触发器的语法代码如下：

```
CREATE TRIGGER <trigge_name>
    ON [<schema name>.]<table or view name>
        [WITH ENCRYPTION|EXECUTE AS <CALLER|SELF|<user>>]
        {FOR|AFTER}
        {[INSERT][,][UPDATE]>[,]<[DELETE]}
        [WITH APPEND]
        [NOT FOR REPLICATION]
AS
<<sql statements>|EXTERNAL NAME <assembly method specifier>>
```

主要参数说明如下。

trigger_name：触发器的名称，必须遵循标识符规则，且不能以“#”或“##”开头。

schema_name：触发器所属架构的名称。

table|view：指定触发器所在的数据表或视图。

注意　只有 INSTEAD OF 触发器才能建立在视图上，并且设置为 WITH CHECK OPTION 的视图也不允许建立 INSTEAD OF 触发器。

WITH ENCRYPTION：对 CREATE TRIGGER 语句的文本进行加密。使用 WITH ENCRYPTION 可以防止将触发器作为 SQL Server 复制的一部分进行发布。

EXECUTE AS：用于执行该触发器的安全上下文。

AFTER：指定 DML 触发器仅在触发 SQL 语句中指定的所有操作都已成功执行时才被激发。仅指定 FOR 关键字时，则 AFTER 为默认值。

{[DELETE][,][INSERT][,][UPDATE]}：指定数据修改语句。这些语句可在 DML 触发器对表或视图进行尝试时激活该触发器。至少指定一个选项。在触发器定义中允许使用上述选项的任意顺序组合。

WITH APPEND：指定再添加一个现有类型的触发器。

【例 10.1】修改学生表中的数据时，下述触发器将向客户端显示一条消息。

```
CREATE TRIGGER 学生_update
   ON 学生表
   AFTER UPDATE
AS
BEGIN
    RAISERROR ('注意：有人修改学生表的数据',16,10)
END
GO
```

【例 10.2】删除学生表中的记录时，下述触发器将删除成绩表中和该生有关的记录。

```
CREATE TRIGGER 学生_delete
   ON 学生表
```

```
  AFTER DELETE
AS
BEGIN
    DELETE FROM 成绩表
    WHERE sno in (SELECT sno from deleted)
END
GO
```

10.2.5　创建 INSTEAD OF 触发器

INSTEAD OF 触发器与 AFTER 触发器的工作流程是不一样的。AFTER 触发器是在 SQL Server 服务器接收到执行 SQL 语句请求之后，先建立临时的 Inserted 表和 Deleted 表，然后更改数据，最后才激活触发器的。而 INSTEAD OF 触发器是在 SQL Server 服务器接收到执行 SQL 语句请求后，先建立临时的 Inserted 表和 Deleted 表，然后就触发 INSTEAD OF 触发器。至于该 SQL 语句是插入数据、更新数据还是删除数据，就一概不管了，把执行权交给 INSTEAD OF 触发器，由它去完成之后的操作。

1. INSTEAD OF 触发器的使用范围

INSTEAD OF 触发器可以同时在数据表和视图中使用，通常在以下几种情况下，建议使用 INSTEAD OF 触发器。

（1）数据禁止修改：数据库的某些数据是不允许修改的，为了防止这些数据被修改，可以用 INSTEAD OF 触发器来跳过修改记录的 SQL 语句。

（2）数据修改后，有可能要回滚的 SQL 语句，可以使用 INSTEAD OF 触发器：在修改数据之前判断回滚条件是否成立，如果成立就不再进行修改数据操作，避免在修改数据之后再回滚操作，从而减轻服务器负担。

（3）在视图中使用触发器：因为 AFTER 触发器不能在视图中使用，如果要在视图中使用触发器，就只能用 INSTEAD OF 触发器。

（4）用个人方式去修改数据：如不满意 SQL 直接的修改数据的方式，可用 INSTEAD OF 触发器来控制数据的修改方式和流程。

2. 创建简单的 INSTEAD OF 触发器

创建 INSTEAD OF 触发器的语法代码如下：

```
CREATE TRIGGER <trigge_name>
    ON [<schema name>.]<table or view name>
        [WITH ENCRYPTION|EXECUTE AS <CALLER|SELF|<user>>]
        {INSTEAD OF}
        {[INSTEAD][,][UPDATE]>[,]<[DELETE]}
        [WITH APPEND]
        [NOT FOR REPLICATION]
AS
<<sql statements>|EXTERNAL NAME <assembly method specifier>>
```

分析上述语法代码可以发现，创建 INSTEAD OF 触发器与创建 AFTER 触发器的语法几

乎一样，只是简单地把 AFTER 改为 INSTEAD OF。

【例 10.3】当有人试图修改学生表中的数据，利用下述触发器可以跳过修改数据的 SQL 语句（防止数据被修改），并向客户端显示一条消息。

```
CREATE TRIGGER  学生_update
   ON 学生表
   INSTEAD OF UPDATE
AS
BEGIN
      RAISERROR ('对不起，学生表的数据不允许修改',16,10)
END
GO
```

10.2.6　查看 DML 触发器

查看已经设计好的 DML 触发器有两种方式：一种是通过 Management Studio 来查看，一种是利用系统存储过程来查看。

1. 在 SQL Server Management Studio 中查看触发器

（1）启动“SQL Server Management Studio”，在“对象资源管理器”下选择“数据库”，定位到“student”数据库，展开其下的“表”树型目录，找到“dbo.学生表”，并选中其下的“触发器”项，如图 10.7 所示。

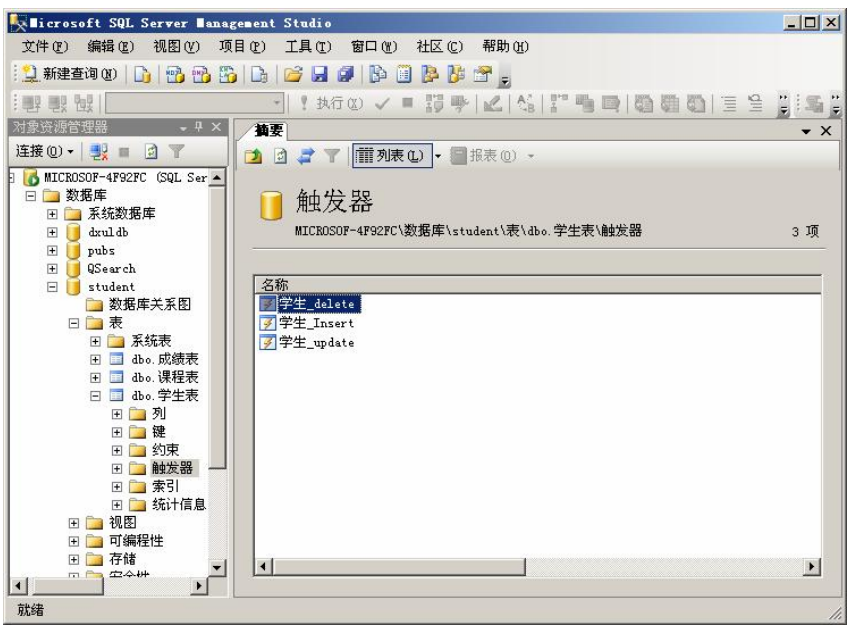


图 10.7　触发器列表

（2）单击“触发器”，在右边窗格的“摘要”对话框里，可以看到已经建好的该数据表的触发器列表。如果在点击“触发器”后，右边没有显示“摘要”对话框，可以单击菜单栏上的“视图”菜单，选择“摘要”选项，打开“摘要”对话框。如果在“摘要”对话框里没有看到已建好的触发器列表，可在“摘要”对话框里右键单击空白处，在弹出的快捷菜单中选择“刷新”选项，刷新对话框后，即可看到触发器列表。

（3）双击要查看的触发器名，弹出“查询编辑器”对话框。对话框里显示的是该触发器的内容，如图 10.8 所示。

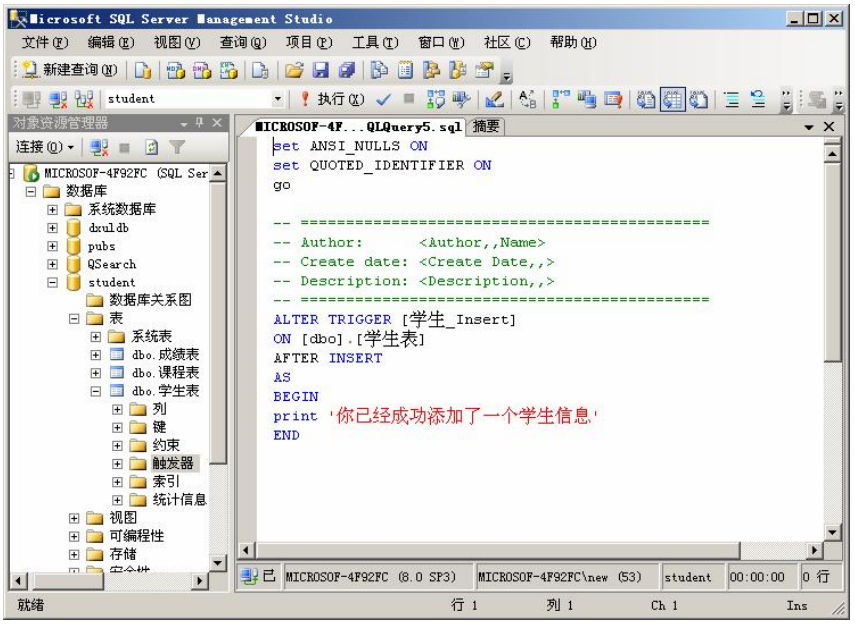


图 10.8　触发器内容

2. 通过系统存储过程查看触发器

SQL Server 2005 提供了两个可以查看触发器内容的系统存储过程。

（1）sp_help。系统存储过程 sp_help 可以了解触发器名称、类型、创建时间等基本信息，其语法格式为：

sp_help '触发器名'

如 sp_help '学生_insert'

运行结果如图 10.9 所示，可以看到触发器“学生_insert”的基本情况。

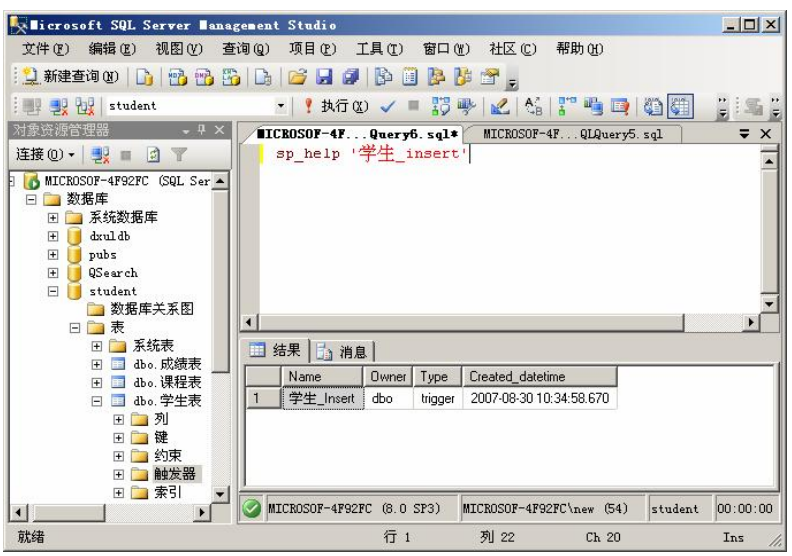


图 10.9　查看触发器基本情况

（2）sp_helptext。系统存储过程 sp_helptext 可以查看触发器的文本信息，其语法格式为：

```
sp_helptext '触发器名'
```

如

```
sp_helptext '学生_insert'
```

运行结果如图 10.10 所示，可以看到触发器“学生_insert”的具体文本内容。

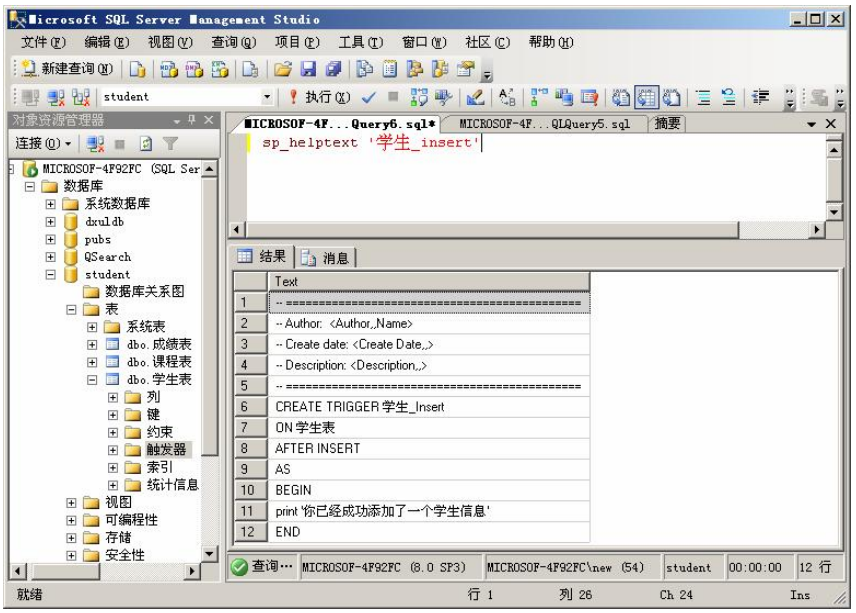


图 10.10　查看触发器具体文本内容

10.2.7　修改 DML 触发器

（1）按照 10.2.6 小节介绍的“在 SQL Server Management Studio 中查看触发器”的方法打开如图 10.8 所示的“查询编辑器”对话框，对话框中显示的就是创建触发器的代码。

（2）在“查询编辑器”对话框里修改触发器的代码。修改之后，单击“执行”按钮即可运行。如果只要修改触发器的名称的话，也可以使用存储过程 sp_rename，其语法格式为：

```
sp_rename '旧触发器名','新触发器名'
```

修改触发器的语法代码如下：

```
ALTER TRIGGER <trigge_name>
        ON {table|view}
            [WITH ENCRYPTION|EXECUTE AS <CALLER|SELF|<user>>]
            {FOR|AFTER|INSTEAD OF}
            {[INSTERT][,][UPDATE]>[,]<[DELETE]}
            [NOT FOR REPLICATION]
AS
<<sql statements>|EXTERNAL NAME <assembly method specifier>>
```

分析上述语法可以发现，修改触发器语法中涉及的主要参数和创建触发器的主要参数几乎一样，在此不再赘述。

10.2.8　删除 DML 触发器

（1）按照 10.2.6 小节介绍的“在 SQL Server Management Studio 中查看触发器”的方法打开如图 10.7 所示的“触发器列表”对话框。

（2）右键单击要删除的触发器，在弹出快捷菜单中选择“删除”选项，此时将会弹出如图 10.11 所示的“删除对象”对话框，在该对话框中单击“确定”按钮，删除操作完成。

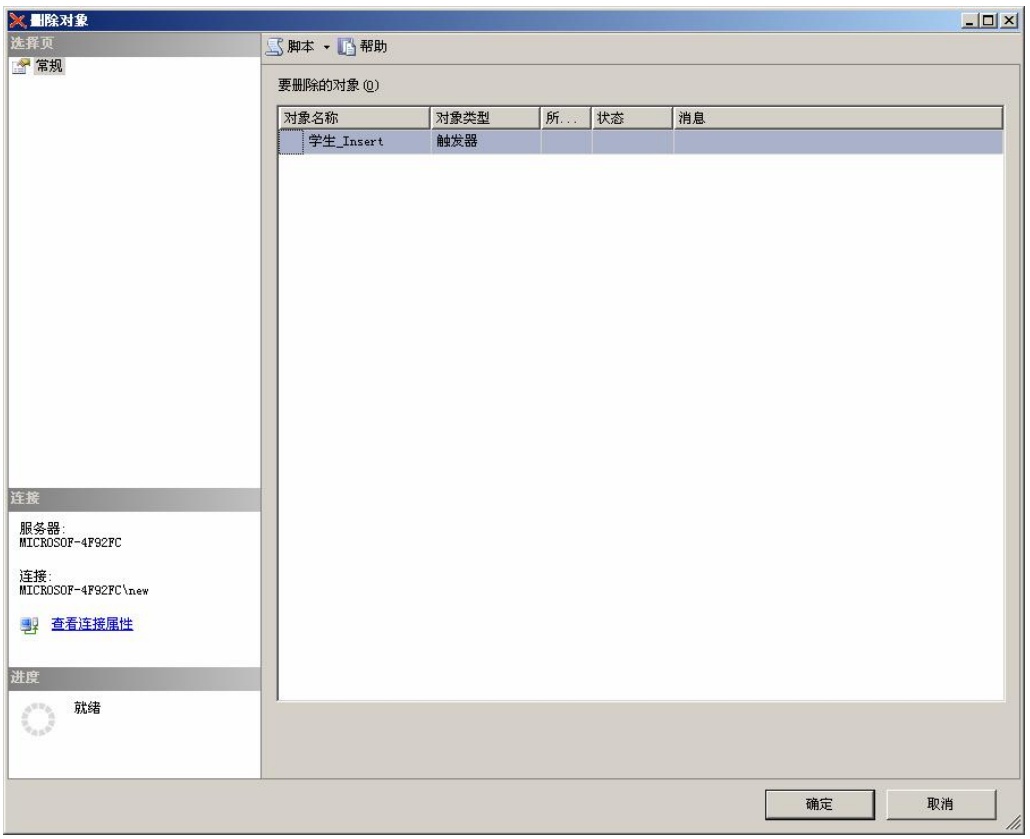


图 10.11　“删除对象”对话框

用 SQL 语句也可删除触发器，删除触发器的语法代码如下所示：

```
Drop Trigger 触发器名
```

10.2.9　禁用与启用 DML 触发器

禁用触发器与删除触发器不同。禁用触发器时，仍会为数据表定义该触发器，只是在执行 INSERT、UPDATE 或 DELETE 语句时，不会执行触发器中的操作。

1. 禁用 DML 触发器

（1）按照 10.2.6 节介绍的“在 SQL Server Management Studio 中查看触发器”的方法打开如图 10.7 所示的“触发器列表”对话框。

（2）右键单击其中一个触发器，在弹出的快捷菜单中选择“禁用”选项，即可禁用该触发器，如图 10.12 所示。

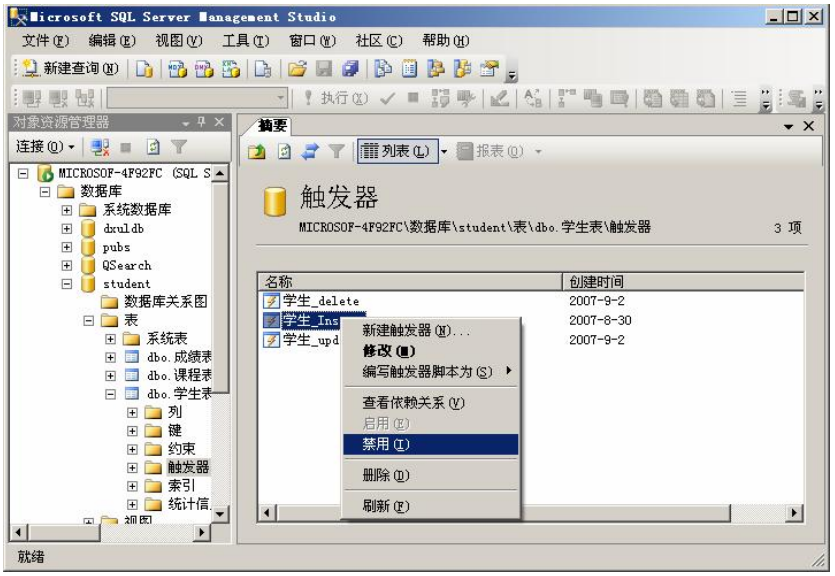


图 10.12 “禁用触发器”对话框

使用 ALTER TABLE 语句也可以禁用 DML 触发器，其语法如下：

ALTER TABLE 数据表名

DISABLE TRIGGER 触发器名或 ALL

如果要禁用所有触发器，用“ALL”来代替触发器名。

2. 启用 DML 触发器

启用触发器与禁用触发器类似，在如图 10.12 所示的弹出的快捷菜单中选择“启用”选项即可。使用 ALTER TABLE 语句也可以启用触发器，其语法如下：

ALTER TABLE 数据表名

ENABLE TRIGGER 触发器名或 ALL

如果要启用所有触发器，用“ALL”来代替触发器名。

10.3 DDL 触发器

DDL 触发器是 SQL Server 2005 新增的一个触发器类型。像常规触发器一样，DDL 触发器将激发存储过程以响应事件。但与 DML 触发器不同的是，DDL 触发器不会为响应针对表或视图的 UPDATE、INSERT 或 DELETE 语句而激发。相反，DDL 触发器会为响应多种数据定义语言（DDL）语句而激发。这些语句主要是以 CREATE、ALTER 和 DROP 开头的语句。DDL 触发器可用于管理任务，如审核和控制数据库操作。

一般来说，在以下几种情况下可以使用 DDL 触发器。

（1）防止数据库架构进行某些修改。

（2）防止数据库或数据表被误操作而删除。

（3）希望数据库发生某种情况以响应数据库架构中的更改。

（4）要记录数据库架构中的更改或事件。

仅在运行触发 DDL 触发器的 DDL 语句后，DDL 触发器才会激发。DDL 触发器无法作为 INSTEAD OF 触发器使用。

10.3.1　创建 DDL 触发器

创建 DDL 触发器的语法代码如下：

```
CREATE TRIGGER <trigge_name>
      ON {ALL SERVER|DATABASE}
          [WITH <ddl_trigger_option>[,...n]]
          {FOR|AFTER}{event_type|event_group}[,...n]
AS
   {sql_statement[;][...n]|EXTERNAL NAME<method specifier >[;]}
<ddl_trigger_option>::=
      [ENCRYPTION]
      [EXECUTE AS Clause]
<method_specifier>::=
      assembly_name.class_name.method_name
```

主要参数说明如下。

trigger_name：触发器的名称，必须遵循标识符规则，但不能以“#”或“##”开头。

DATABASE：将 DDL 触发器的作用域应用于当前数据库。如果指定了此参数，则只要当前数据库中出现 event_type 或 event_group，就会激发该触发器。

ALL SERVER：将 DDL 触发器的作用域应用于当前服务器。如果指定了此参数，则只要当前服务器中的任何位置上出现 event_type 或 event_group，就会激发该触发器。

event_type：执行之后将导致激发 DDL 触发器的 Transact-SQL 语言事件的名称。

event_group：预定义的 Transact-SQL 语言事件分组的名称。

其他参数在前面章节中已经说明，在此不再赘述。

下面通过示例来说明如何建立 DDL 触发器。

【例 10.4】建立用于保护“student”数据库中的数据表不被删除的触发器。

具体操作步骤如下。

（1）启动“SQL Server Management Studio”，在“对象资源管理器”下选择“数据库”，定位到“student”数据库。

（2）单击“新建查询”按钮，在弹出的“查询编辑器”的编辑区里输入以下代码：

```
CREATE TRIGGER disable_drop_table
   ON DATABASE
   FOR DROP_TABLE
AS
BEGIN
      RAISERROR ('对不起，student 数据库中的表不能删除',16,10)
END
GO
```

（3）单击“执行”按钮，生成触发器。

10.3.2 测试 DDL 触发器功能

测试 DDL 触发器的功能的具体操作步骤如下。

（1）启动“SQL Server Management Studio”，在“对象资源管理器”下选择“数据库”，定位到“student”数据库。

（2）单击“新建查询”按钮，在弹出的“查询编辑器”的编辑区里输入以下代码：

```
DROP TABLE 成绩表
```

（3）单击“执行”按钮，运行结果如图 10.13 所示。

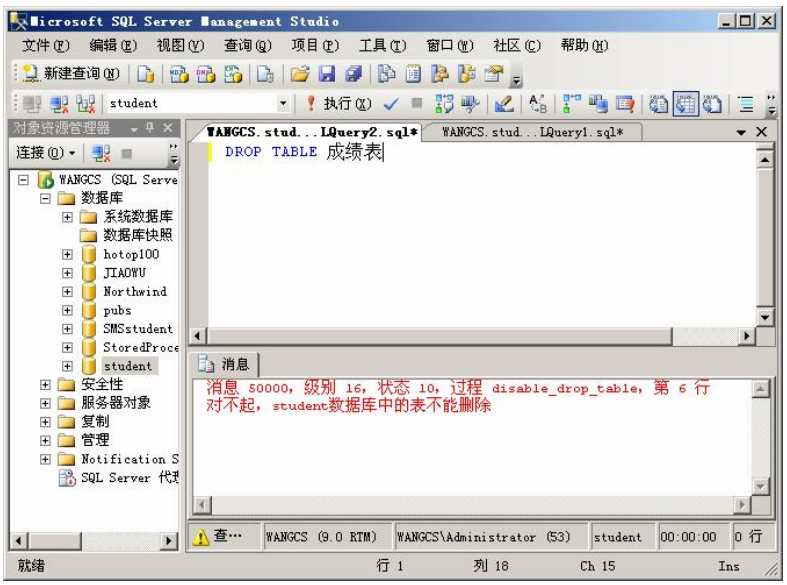


图 10.13 测试“删除数据表”结果

10.3.3 查看和修改 DDL 触发器

DDL 触发器有两种，一种是作用在当前 SQL Server 服务器上的，一种是作用在当前数据库中的。这两种 DDL 触发器在 Management Studio 中所在的位置是不同的。

1. 作用在当前 SQL Server 服务器上的 DDL 触发器所在位置

选择所在的 SQL Server 服务器，定位到“服务器对象”中的“触发器”，在“摘要”对话框里就可以看到所有作用在当前 SQL Server 服务器上的 DDL 触发器。

2. 作用在当前数据库中的 DDL 触发器所在位置

在 SQL Server 服务器上，通过“数据库”选择所在数据库，然后定位到“可编程性”中的“数据库触发器”，在摘要对话框里就可以看到所有的当前数据库中的 DDL 触发器。

右键单击触发器，在弹出的快捷菜单中选择“编写数据库触发器脚本为”→“CREATE 到”→“新查询编辑器对话框”，然后在“查询编辑器”对话框里可以看到该触发器的内容。

在 Management Studio 中如果要修改 DDL 触发器内容，就只能先删除该触发器，再重新建立一个 DDL 触发器。

虽然在 Management Studio 中没有直接提供修改 DDL 触发器的对话框，但在“查询编辑

器”对话框里依然可以用 SQL 语句来进行修改。

10.4　游 标 概 述

在 SQL Server 2005 数据库系统开发中，执行 SELECT 语句可进行查询并返回满足条件的所有记录，这一完整的记录集称为结果集。由于应用程序并不总是能将整个结果集作为一个单元来有效地处理，因此往往需要一种机制，便于每次处理结果集中的一条或一部分记录。游标就能够提供这种机制，对结果集中的部分记录进行处理，不但允许定位在结果集的特定记录上，还可以从结果集的当前位置检索若干条记录，并支持对结果集中当前记录进行数据修改。

10.4.1　游标概念及特点

在 SQL Server 数据库中，游标是一个比较重要的概念，它总是与一条 T-SQL 选择语句相关联。游标是一种处理数据的方法，它可对结果集中的记录进行逐行处理，可将游标视作一种指针，用于指向处理结果集中任意位置的数据。就本质而言，游标提供了一种对从表中检索出的数据进行操作的灵活手段。由于游标由结果集和结果集中指向特定记录的游标位置组成，因此当决定对结果集进行处理时，必须声明一个指向该结果集的游标。

游标具有如下特点。

- 允许定位在结果集的特定行上。
- 可以从结果集的当前位置检索一行或一部分行。
- 支持对结果集中当前位置的行进行数据修改。
- 为由其他用户对显示在结果集中的数据库数据所做的更改，提供不同级别的可见性支持。
- 提供脚本、存储过程和触发器中用于访问结果集中的数据的 Transact-SQL 语句。

10.4.2　游标分类

SQL Server 2005 中的游标可分为 3 类：Transact-SQL 游标、API 服务器游标和客户端游标。

1. Transact-SQL 游标

Transact-SQL 游标是由 SQL Server 服务器实现的游标，主要用于存储过程、触发器和 Transact-SQL 脚本中，它们使结果集中的内容可用于其他 Transact-SQL 语句。

2. API 服务器游标

API 服务器游标在服务器上实现，并由 API 游标函数进行管理。当应用程序调用 API 游标函数时，游标操作由 OLE DB 访问接口或 ODBC 驱动程序传送给服务器。

3. 客户端游标

客户端游标，即在客户端实现的游标。在客户端游标中，将使用默认结果集把整个结果集高速缓存在客户端上，所有的游标操作都针对此客户端高速缓存来执行。客户端游标将不使用 Microsoft SQL Server 2005 的任何服务器游标功能。客户端游标仅支持静态游标。

由于 Transact-SQL 游标和 API 服务器游标用于服务器端，所以被称为服务器游标，也被称为后台游标。本章主要讲述服务器游标。

10.5 游标的声明和应用

10.5.1 声明游标

SQL Server 2005 提供了两种声明游标的方式：一种是 SQL-92 语法，另一种是 T-SQL 扩充语法。但这两种声明形式不能混合使用，只能选择其中一种来进行游标的声明。当在 CURSOR 关键词之前指定 SCROLL 或 INSENSITIVE 关键词时，则在 CURSOR 与 FOR select-statement 之间就不能使用任何关键词；若在 CURSOR 与 FOR select-statement 之间指定了关键词，就无法在 CURSOR 关键词之前指定 SCROLL 或 INSENSITIVE。

下面们将分别针对这两个语法来说明。

1. 使用 SQL-92 语法来声明 CURSOR

使用 SQL-92 语法来声明 CURSOR 的语法代码如下：

```
DECLARE cursor_name [INSENSITIVE][SCROLL] CURSOR
FOR select_statement
[FOR {READ ONLY|UPDATE [OF column_name[,...n]]}]
```

主要参数说明如下。

cursor_name：CURSOR（游标）的名称。

INSENSITIVE：定义一个游标，以创建由该游标使用的数据的临时复本。对游标的所有请求都从 Tempdb 中的这一临时表中得到应答；因此，在对该游标进行提取操作时，返回的数据中不反映对基表所做的修改，并且该游标不允许修改。如果省略 INSENSITIVE，则已提交的（任何用户）对基表的删除和更新都反映在后面的提取中。

SCROLL：指定所有的提取选项（FIRST、LAST、PRIOR、NEXT、RELATIVE、ABSOLUTE）均可用。如果未指定 SCROLL，则 NEXT 是唯一支持的提取选项。

select_statement：定义游标结果集的标准 SELECT 语句。在游标声明的 select_statement 内不允许使用关键字 COMPUTE、COMPUTE BY、FOR BROWSE 和 INTO。

READ ONLY：禁止通过该游标进行更新。在 UPDATE 或 DELETE 语句的 WHERE CURRENT OF 子句中不能引用游标。该选项优于要更新的游标的默认功能。

UPDATE [OF column_name[,...n]]：定义游标中可更新的列。如果指定了 OF column_name [,...n]，则只允许修改列出的列。如果指定了 UPDATE，但未指定列的列表，则可以更新所有列。

2. T-SQL 扩充语法来声明 CURSOR

使用 T-SQL 扩充语法来声明 CURSOR 的语法代码如下：

```
DECLARE cursor_name CURSOR
[LOCAL|GLOBAL]
[FORWARD_ONLY|SCROLL][STATIC|KEYSET|DYNAMIC|FAST_FORWARD]
[READ_ONLY|SCROLL_LOCKS|OPTIMISTIC]
[TYPE_WARNING]
FOR select_statement[FOR UPDATE[ OF column_name [,...n ]]]
```

主要参数说明如下。

cursor_name：CURSOR（游标）的名称。

LOCAL：指定对于创建的批处理、存储过程或触发器，该游标的作用域是局部的。该游标名称仅在这个作用域内有效。在批处理、存储过程、触发器或存储过程的 OUTPUT 参数中，该游标可由局部游标变量引用。OUTPUT 参数用于将局部游标传递回调用的批处理、存储过程或触发器。批处理、存储过程或触发器可在存储过程终止后为游标变量分配参数使其引用游标。除非 OUTPUT 参数将游标传递回来，否则游标将在批处理、存储过程或触发器终止时隐式释放。如果 OUTPUT 参数将游标传递回来，则游标在最后引用它的变量释放或离开作用域时释放。

GLOBAL：指定该游标的作用域是全局的。在由连接执行的任何存储过程或批处理中，都可以引用该游标。该游标仅在断开连接时隐式释放。

FORWARD_ONLY：指定游标只能向前滚动。FETCH NEXT 是唯一支持的提取选项。如果在指定 FORWARD_ONLY 时不指定 STATIC、KEYSET 和 DYNAMIC 关键字，则游标作为 DYNAMIC 游标进行操作。如果 FORWARD_ONLY 和 SCROLL 均未指定，则除非指定 STATIC、KEYSET 或 DYNAMIC 关键字，否则游标默认为 FORWARD_ONLY。STATIC、KEYSET 和 DYNAMIC 游标默认为 SCROLL。

STATIC：定义一个游标，以创建由该游标使用的数据的临时复本。对游标的所有请求都从 Tempdb 中的这一临时表中得到应答。因此，在对该游标进行提取操作时，返回的数据中不反映对基表所做的修改，并且该游标不允许修改。

KEYSET：指定当游标打开时，游标中行的成员身份和顺序已经固定。对行进行唯一标识的键集内置在 Tempdb 内一个称为“keyset”的表中。

DYNAMIC：定义一个游标，以反映在滚动游标时对结果集内的各行所做的所有数据更改。行的数据值、顺序和成员身份在每次提取时都会更改。动态游标不支持 ABSOLUTE 提取选项。

FAST_FORWARD：指定启用了性能优化的 FORWARD_ONLY、READ_ONLY 游标。如果指定了 SCROLL 或 FOR_UPDATE，则不能同时指定 FAST_FORWARD。

READ_ONLY：禁止通过该游标进行更新。在 UPDATE 或 DELETE 语句的 WHERE CURRENT OF 子句中不能引用游标。该选项优于要更新的游标的默认功能。

SCROLL_LOCKS：指定通过游标进行的定位更新或删除一定会成功。将行读取到游标中，以确保这些行对随后的修改可用，这时，Microsoft SQL Server 将锁定这些行。如果指定了 FAST_FORWARD，则不能指定 SCROLL_LOCKS。

OPTIMISTIC：指定如果行自从被读入游标以来已得到更新，则通过游标进行的定位更新或定位删除不会成功。当将行读入游标时，SQL Server 不会锁定行，SQL Server 会使用 timestamp 列值的比较，如果表没有 timestamp 列，则使用校验和值，以确定将行读入游标后是否已修改该行。如果已修改该行，则尝试进行的定位更新或删除将失败。如果指定了 FAST_FORWARD，则不能指定 OPTIMISTIC。

TYPE_WARNING：指定如果游标从请求的类型隐式转换为另一种类型，则向客户端发送警告消息。

select_statement：定义游标结果集的标准 SELECT 语句。在游标声明的 select_statement 内不允许使用关键字 COMPUTE、COMPUTE BY、FOR BROWSE 和 INTO。

FOR UPDATE [OF column_name [,...n]]：定义游标中可更新的列。如果提供了 OF column_name [,...n]，则只允许修改列出的列。如果指定了 UPDATE，但未指定列的列表，则除非同时指定了 READ_ONLY 选项，否则可以更新所有的列。

【例 10.5】声明一个名称为学生_Cursor 的游标。

```
DECLARE 学生_Cursor CURSOR FOR
SELECT sNo, sName
FROM 学生表;
```

10.5.2 打开游标

当打开 CURSOR 时，会先执行 SELECT 语句，接着 CURSOR 会执行。当 CURSOR 执行完毕，此时 CURSOR 的指针会指到第一条记录的前面。如果在 CURSOR 内，任意行的大小超过了 SQL Server 表的最大行限制时，执行 OPEN 语句就会失败。如果以 KEYSET 选项声明了游标，OPEN 会创建临时表存放索引键集，临时表存放在 Tempdb 数据库中。打开游标的语法代码如下：

```
OPEN {{[GLOBAL] cursor_name}|cursor_variable_name}
```

主要参数说明如下。

GLOBAL：指定 cursor_name 是全局游标。

cursor_name：已声明的游标的名称。如果全局游标和局部游标都使用 cursor_name 作为其名称，那么如果指定了 GLOBAL，则 cursor_name 是全局游标；否则 cursor_name 是局部游标。

cursor_variable_name：游标变量的名称。该变量引用一个游标。

【例 10.6】打开一个名称为“学生_Cursor”的游标。

```
OPEN 学生_Cursor;
```

10.5.3 从游标中提取记录

声明一个游标并成功地打开该游标之后，就可以使用 FETCH 语句从该游标中提取特定的记录，其语法代码格式如下：

```
FETCH
  [[NEXT|PRIOR|FIRST|LAST|ABSOLUTE{n|@nvar}|RELATIVE{n|@nvar}]
  FROM]{{[GLOBAL]cursor_name}|@cursor_variable_name}
  [INTO@variable_name[,...n]]
```

主要参数说明如下。

NEXT：返回紧跟当前行之后的结果行，并将当前行递增为结果行。如果 FETCH NEXT 为对游标的第一次提取操作，则返回结果集中的第一行。NEXT 为默认的游标提取选项。

PRIOR：返回紧邻当前行的前面的结果行，并且当前行递减为结果行。如果 FETCH PRIOR 为对游标的第一次提取操作，则没有行返回，并将游标置于第一行之前。

FIRST：返回游标中的第一行并将其作为当前行。

LAST：返回游标中的最后一行并将其作为当前行。

ABSOLUTE {n|@nvar}：如果 n 或@nvar 为正数，则返回从游标开始的第 *n* 行，并将返

回行变成新的当前行。如果 n 或@nvar 为负数，则返回从游标末尾开始的第 *n* 行，并将返回行变成新的当前行。如果 n 或@nvar 为 0，则不返回行。n 必须是整数常量，@nvar 的数据类型必须为 smallint、tinyint 或 int。

RELATIVE{n|@nvar}：如果 n 或@nvar 为正数，则返回从当前行开始的第 *n* 行，并将返回行变成新的当前行。如果 n 或@nvar 为负数，则返回当前行之前的第 *n* 行，并将返回行变成新的当前行。如果 n 或@nvar 为 0，则返回当前行。在对游标完成第一次提取时，如果在将 n 或@nvar 设置为负数或 0 的情况下指定 FETCH RELATIVE，则不返回行。n 必须是整数常量，@nvar 的数据类型必须为 smallint、tinyint 或 int。

GLOBAL：指定 cursor_name 是全局游标。

cursor_name：要从中进行提取的，打开的游标名称。如果同时存在具有以 cursor_name 作为名称的全局和局部游标，若指定了 GLOBAL，则 cursor_name 是指全局游标，如果未指定 GLOBAL，则指局部游标。

@cursor_variable_name：游标变量名，引用要从中进行提取操作的打开的游标。

INTO @variable_name[,...n]：允许将提取操作的列数据放到局部变量中。列表中的各个变量从左到右与游标结果集中的对应列相关联。各变量的数据类型必须与相应的结果集列的数据类型匹配，或是结果集列数据类型所支持的隐式转换。变量的数目必须与游标选择列表中的列数一致。

执行游标语句后，可通过@@FETCH_STATUS 全局变量返回游标当前的状态。在每次使用 FETCH 从游标中读取数据时都应该检查该变量，以确定上次 FETCH 操作是否成功，进而决定如何进行下一步处理。@@FETCH_STATUS 全局变量有 3 个不同的返回值。

0：FETCH 语句执行成功。

−1：FETCH 语句执行失败或者行数据超出游标结果集的范围。

−2：提取的数据不存在。

【例 10.7】从一个已经被打开的游标（学生_Cursor）中逐行提取记录。

```
FETCH NEXT FROM 学生_Cursor
WHILE @@FETCH_STATUS = 0
BEGIN
   FETCH NEXT FROM 学生_Cursor
END
```

10.5.4　关闭游标

通过一个游标完成提取记录或修改记录的操作以后，应当使用 CLOSE 语句关闭该游标，以释放当前的结果集并解除定位于该游标的记录行上的游标锁定。使用 CLOSE 语句关闭该游标之后，该游标的数据结构仍然存储在系统中，可以通过 OPEN 语句重新打开。但关闭后不允许提取和定位更新，直到游标重新打开。CLOSE 语句必须在一个打开的游标上执行，而不允许在一个仅仅声明的游标或一个已经关闭的游标上执行。关闭游标的语法代码格式如下：

```
CLOSE {{[GLOBAL] cursor_name}|cursor_variable_name}
```

主要参数说明如下。

GLOBAL：指定 cursor_name 是指全局游标。

cursor_name：打开的游标的名称。

cursor_variable_name：与打开的游标关联的游标变量的名称。

【例 10.8】关闭一个已经打开的游标（学生_Cursor）。

```
CLOSE 学生_Cursor;
```

10.5.5 释放游标

关闭一个游标以后，其数据结构仍然存储在系统中。为了将该游标占用的资源全部归还给系统，还需要使用 DEALLOCATE 语句删除游标引用，让 SQL Server 释放组成该游标的数据结构。释放游标的语法代码格式如下：

```
DEALLOCATE {{[GLOBAL] cursor_name}|cursor_variable_name}
```

主要参数说明如下。

GLOBAL：指定 cursor_name 是全局游标。

cursor_name：声明的游标的名称。

cursor_variable_name：cursor 变量的名称。

【例 10.9】释放一个名称为“学生_Cursor”的游标。

```
DEALLOCATE 学生_Cursor;
```

10.5.6 游标的应用

前面们介绍了如何声明游标，打开游标，从游标中提取数据以及关闭和释放游标的方法。下面们将通过两个应用实例使读者更深刻地理解游标的原理与应用。

【例 10.10】建立一个名称为“学生_Cursor”的游标，通过该游标逐行浏览学生表中的记录。

步骤如下。

（1）启动“SQL Server Management Studio”，在“对象资源管理器”下选择“数据库”，定位到“student”数据库。

（2）单击“新建查询”按钮，在弹出的“查询编辑器”的编辑区里输入以下代码：

```
--声明游标
DECLARE 学生_Cursor CURSOR FOR
SELECT * FROM 学生表;
--打开游标
OPEN 学生_Cursor;
--提取数据
FETCH NEXT FROM 学生_Cursor
--判断 FETCH 是否成功
WHILE @@FETCH_STATUS = 0
BEGIN
    --提取下一行
    FETCH NEXT FROM 学生_Cursor
END
--关闭游标
```

```
CLOSE 学生_Cursor;
--释放游标
DEALLOCATE 学生_Cursor;
```

（3）单击“执行”按钮，运行结果如图 10.14 所示。

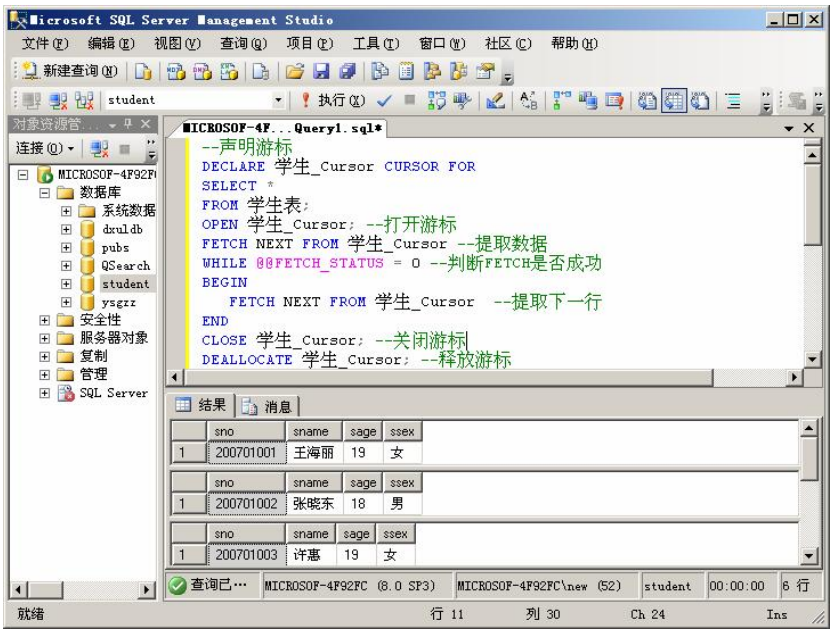


图 10.14　通过游标逐行浏览“学生表”中的记录

【例 10.11】建立一个名称为“修改年龄_Cursor”的游标，通过该游标修改“学生表”中 sno = '200701003'的年龄值。

步骤如下。

（1）启动“SQL Server Management Studio”，在“对象资源管理器”下选择“数据库”，定位到“student”数据库。

（2）单击“新建查询”按钮，在弹出的“查询编辑器”的编辑区里输入以下代码：

```
DECLARE @xh varchar(10)，@age int
DECLARE 修改年龄_Cursor CURSOR FOR
    SELECT sno,sage FROM 学生表 WHERE sno = '200701003';
OPEN 修改年龄_Cursor;
FETCH NEXT FROM 修改年龄_Cursor INTO @xh ,@age;
PRINT '修改前：' + @xh+'同学年龄为：' + convert(varchar,@age);
UPDATE 学生表 SET sage = sage + 1
    WHERE current of 修改年龄_Cursor;
CLOSE 修改年龄_Cursor;
OPEN 修改年龄_Cursor;
FETCH NEXT FROM 修改年龄_Cursor INTO@xh, @age;
PRINT '修改后：' + @xh + '同学年龄为：' + convert(varchar,@age)
```

```
CLOSE 修改年龄_Cursor;
DEALLOCATE 修改年龄_Cursor;
```

（3）单击“执行”按钮，运行结果如图 10.15 所示。

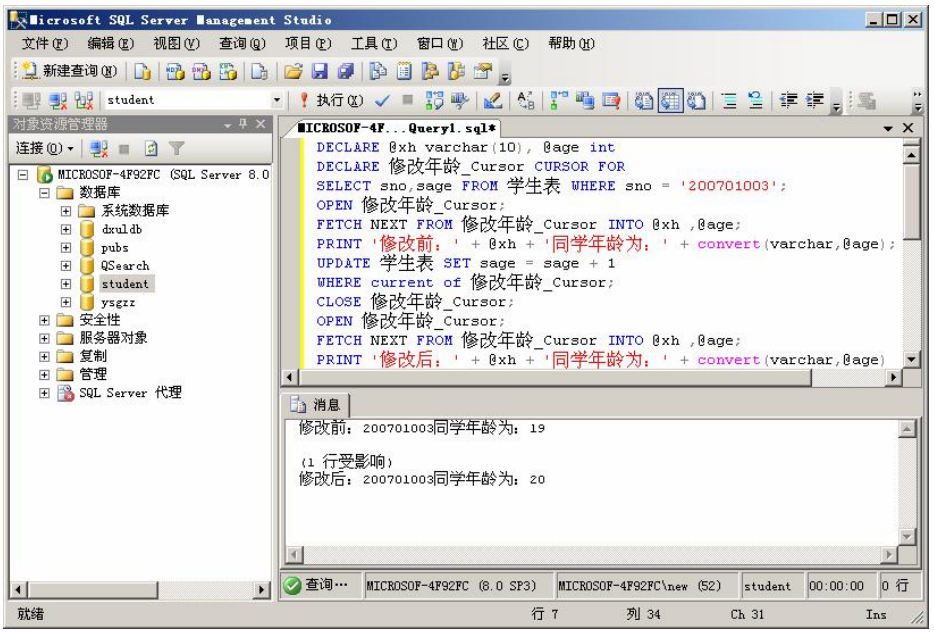


图 10.15　通过游标修改学生表中的数据

本 章 小 结

本章首先介绍了触发器的特点、作用和类型，然后图文并茂地介绍了创建 DML 和 DDL 触发器的方法和步骤，接着介绍了游标的特点、作用，最后详细描述了游标的声明、打开、数据提取、关闭和释放以及游标的应用。

习　　题

一、选择题

1．以下关于触发器的说法中错误的是（　　）。

A．在一个表上只能建立一个触发器。

B．一个触发器只能由一种数据操作来引发。

C．一个触发器只能作用于一个表。

D．所有 T-SQL 语句都可以用在触发器中。

2．以下不会引起 DML 触发器执行的语句是（　　）。

A．INSERT 语句　　B．DELETE 语句

C．DROP 语句　　D．UPDATE 语句

二、填空题

1．触发器是一类特殊的存储过程，其特殊性在于它并不需要_____，而是在对表或视图发出 T-SQL 语句时自动执行。

2．在触发器中可以使用两个特殊的临时表：即 Inserted 表和 Deleted 表，前者用于保存_____的记录，后者用于保存_____的记录。

3．使用 FETCH 语句从游标中提取记录时，使用以下参数返回的记录是

NEXT：_____。

PRIOR：_____。

FIRST：_____。

LAST：_____。

三、简答题

1．DDL 触发器和 DML 触发器有什么区别？

2．AFTER 触发器和 INSTEAD OF 触发器有什么区别？

3．使用游标访问数据包括哪些步骤?

4．关闭游标和释放游标分别用什么语句来完成？这两种操作有什么区别？

本 章 实 训

一、实训目的

1．了解触发器的基本概念，理解触发器的工作原理。

2．掌握创建和测试触发器的方法。

3．掌握修改和删除触发器的方法。

4．了解游标的基本概念，理解游标的作用。

5．掌握创建和使用游标的方法。

6．掌握关闭和释放游标的方法。

二、实训要求

1．实训前做好上机实训的准备，针对实训内容，认真复习与本次实训有关的知识，完成实训内容的预习准备工作。

2．能认真独立完成实训内容。

3．实训后做好实训总结，根据实训情况完成总结报告。

三、实训学时

2 学时。

四、实训内容

现有学生成绩数据库（XSCJDB），包括学生表“XS”（学号、姓名、性别、年龄、系别）、课程表“KC”（课程号、课程名、学分数、学时数）和成绩表“CJ”（学号、课程号、成绩），以下实验均在此数据库上完成。

1．创建触发器“trigger_1”，实现当修改学生表“XS”中的数据时，显示提示信息“学生表的数据被修改了”。

2．在学生库中创建触发器“trigger_2”，实现如下功能：当在学生表“XS”中删除一条学生信息后，自动删除该学生在成绩表“CJ”中的信息。

3．创建触发器“trigger_3”，实现如下功能：当修改学生表“XS”中的某个学生的学号时，对应成绩表“CJ”中的学号也自动修改。

4．对已创建的触发器“trigger_1”进行修改，实现当修改学生表“XS”中的数据时，显示提示信息“学生情况表中×××号学生记录被修改了”。

5．删除学生表“XS”上的触发器“trigger_1”。

6．创建游标“cursor_1”，实现如下功能：查询所有计算机系的男学生信息，并且提取所有行。

7．关闭并释放游标“cursor_1”。

五、实训思考题

1．如何使用触发器，实现学生成绩数据库（XSCJDB）中的数据表不被删除？

2．如何使用游标，实现逐行提取成绩表“CJ”中的记录？

第11章

数据库的备份还原与数据传输

数据库备份与还原组件和数据转换服务是SQL Server的重要组成部分。它们是数据库管理员维护数据库的重要工具。本章主要介绍数据库备份基础、备份设备、备份与还原类型、数据库的分离与附加以及数据的导入与导出。

11.1 数据库备份基础

SQL Server 2005 系统提供了内置的安全性和数据保护机制，以防止非法登录者或非授权用户对 SQL Server 数据库或数据造成破坏。但对于合法用户的误操作、存储媒体受损或SQL Server服务出现崩溃性出错等因素，则需要通过数据库的备份与还原来应对。

备份和还原组件是SQL Server的重要组成部分，为存储在SQL Server 数据库中的关键数据提供重要的保护手段。

数据库中的数据损失或被破坏主要由于以下原因。

1．存储介质故障

倘若保存有数据库文件的存储介质——磁盘驱动器出现彻底崩溃，而用户又未曾进行过数据库备份，则可能造成数据的丢失。

2．服务器崩溃故障

再好的系统硬件、再稳定的软件也存在漏洞与不足之处。倘若数据库服务器彻底瘫痪，将面临重建系统的窘境。如果事先进行了完善的备份，则可迅速地完成系统的恢复性重建工作，并将数据灾难造成的损失降到最低程度。

3．用户错误操作

倘若用户有意或无意地在数据库上进行了大量非法操作(诸如误删除某些重要的数据库、表格等信息)，则数据库系统将面临难以使用和管理的境地。重新恢复最好的方法是备份数据信息，使系统回归到可靠、稳定、一致的状态并再度工作。

4．计算机病毒

破坏性病毒会破坏系统软件、硬件和数据。

5．自然灾害

自然灾害，如火灾、洪水或地震等，会造成极大的破坏，会损坏计算机系统及其数据，导致数据库系统不能正常工作或造成数据的丢失。

备份是对 SQL Server 数据库或事务日志进行复制。数据库备份记录了在进行备份这一操作时，数据库中所有数据的状态。如果数据库受损，这些备份文件将在数据库恢复时用来恢复数据库。

11.2 备 份 设 备

在进行数据库备份之前首先必须创建备份设备。备份设备用来存储数据库事务日志、数据文件或文件组的存储介质，可以是硬盘、磁带等。

11.2.1 物理设备与逻辑设备

SQL Server 使用物理设备名称或逻辑设备名称标识备份设备。

物理备份设备是操作系统用来标识备份设备的名称。例如，磁盘设备名称为 d:\pubs.bak，或者磁带设备名称为\\TAPE0。

逻辑备份设备是用来标识物理备份设备的别名或公用名称。逻辑设备名称永久地存储在 SQL Server 内的系统表中。使用逻辑备份设备的优点是引用它比引用物理设备名称简单。例如，逻辑设备名称可以是 pubs_Backup，而物理设备名称则是 d:\pubs.bak.

注意 在实施数据库备份或还原时，既可以使用物理设备名又可以使用逻辑备份设备名。

11.2.2 创建与管理备份设备

1. 创建备份设备

创建备份设备的步骤如下。

（1）启动“SQL Server Management Studio”，在“对象资源管理器”窗口里展开“服务器对象”树型目录，鼠标右键单击“备份设备”，如图 11.1 所示。

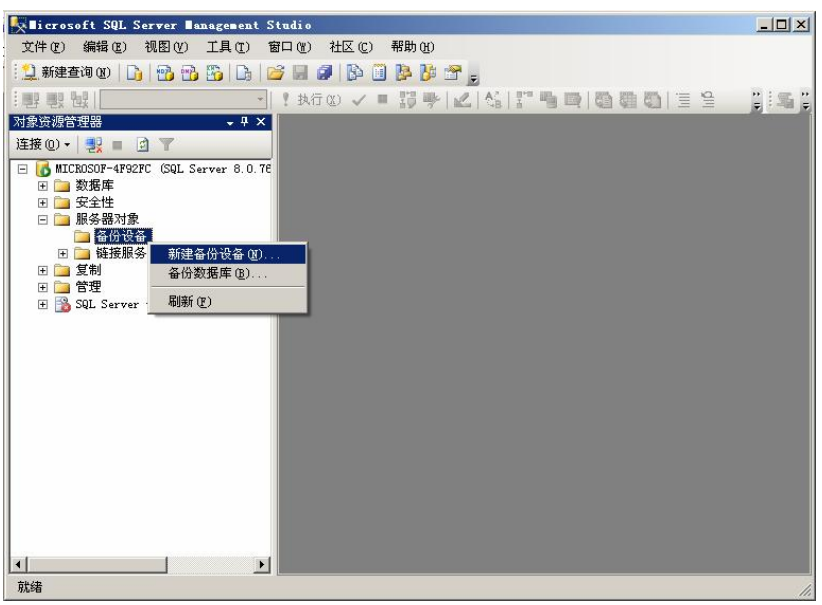


图 11.1　从“SQL Server Management Studio”中选择“备份设备”，鼠标右键单击

（2）在弹出的快捷菜单里选择“新建备份设备”选项，弹出如图 11.2 所示的“新建备份设备”对话框。

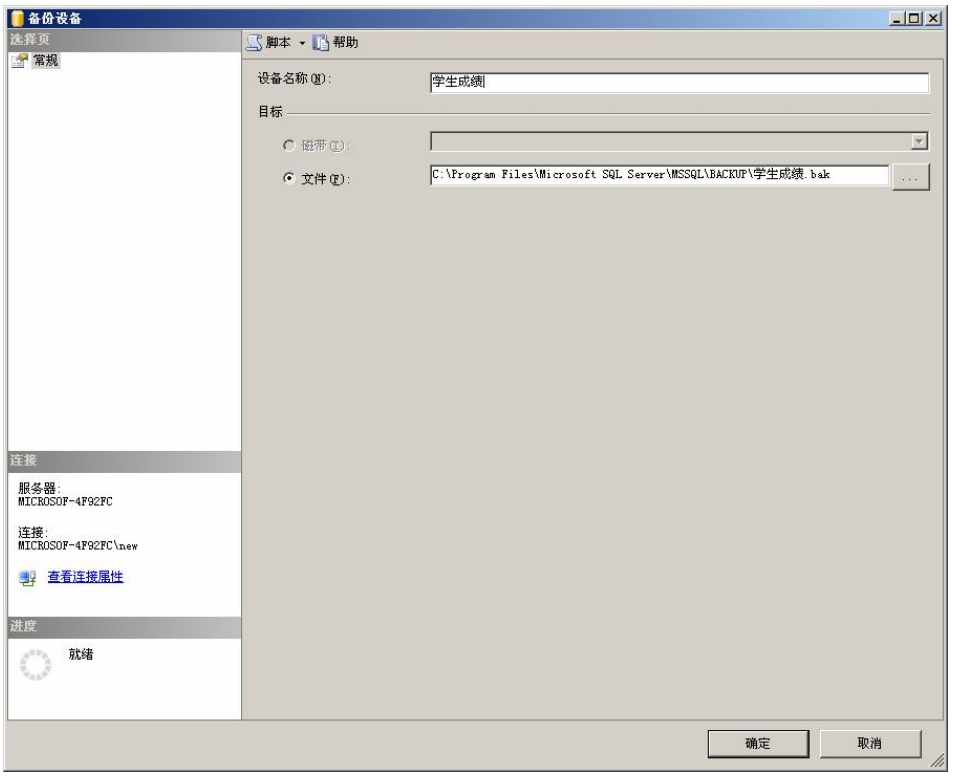


图 11.2　“新建备份设备”对话框

（3）在“设备名称”文本框里输入备份设备的名称。

（4）在“文件”文本框里输入备份设备的路径和文件名。由此可见，SQL Server 2005 中的备份设备事实上也只是一个文件而已。

（5）设置完毕后，单击“确定”按钮，开始创建备份设备操作。

SQL Server 2005 还提供了一个名为“sp_addumpdevice”的存储过程，可以创建数据库备份设备，其语法代码如下：

```
sp_addumpdevice [ @devtype = ] 'device_type'
,[ @logicalname = ] 'logical_name'
,[ @physicalname = ] 'physical_name'
```

主要参数说明如下。

@devtype：设备类型，可以支持的值为 disk 和 tape，disk 为磁盘文件；tape 为 Windows 支持的任何磁带设备。

@logicalname：备份设备的逻辑名称，相当于图 11.1 中的“设备名称”。

@physicalname：备份设备的物理名称，相当于图 11.1 中的“文件”。

【例 11.1】创建一个名为“学生成绩”的磁盘备份设备。

```
sp_addumpdevice 'disk','学生成绩','d:\学生成绩.bak'
```

2. 删除备份设备

删除数据库中备份设备的步骤如下。

（1）启动“SQL Server Management Studio”，在“对象资源管理器”窗口里展开“服务器对象”树型目录，再展开“备份设备”树型目录，鼠标右键单击要删除的备份设备名，如图 11.3 所示。

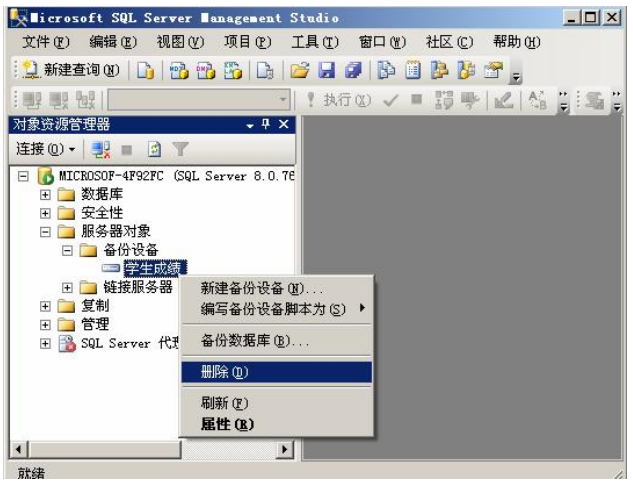


图 11.3　从“SQL Server Management Studio”中选择要删除的备份设备，鼠标右键单击

（2）在弹出的快捷菜单里选择“删除”选项，弹出如图 11.4 所示“删除对象”对话框，在该对话框里单击“确定”按钮开始执行删除备份设备操作。

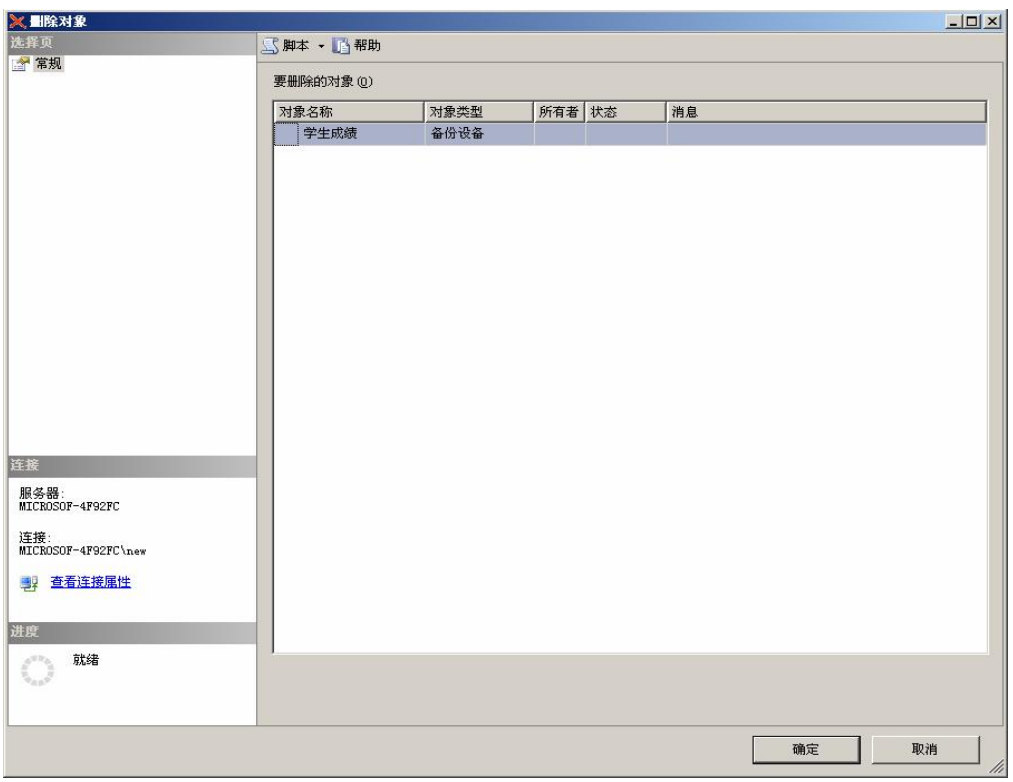


图 11.4　“删除对象”对话框

SQL Server 2005 还提供了一个名为“sp_dropdevice”的存储过程，可以删除库备份设备，其语法代码如下：

```
sp_dropdevice [ @logicalname = ] 'device'
    [,[ @delfile = ] 'delfile']
```

主要参数说明如下。

@logicalname：表示备份设备的逻辑名称。

@delfile：表示物理备份设备文件。

【例 11.2】删除名为“学生成绩”的备份设备。

```
sp_dropdevice '学生成绩'
```

11.3　数据库备份

SQL Server 2005 提供了 4 种数据库备份方法：

- 完全备份；
- 差异备份；
- 日志备份；
- 数据文件或文件组备份。

11.3.1　完全备份

完全备份指的是备份整个数据库的所有内容，包括事务日志。完全备份需要比较大的存储空间来存储备份文件，备份时间也比较长。还原完全备份时，由于需要从备份文件中提取大量数据，因此还原操作也需要较长的时间。

完全备份是所有备份方法中，还原数据库最简单的一种。正是由于完全备份还原的简单性，所以在实际应用中，它也被用得最广泛。

下面以备份“pubs”数据库为例，介绍数据库完全备份的实现方法。

1. 通过 SQL Server Management Studio 实现完全备份

（1）启动“SQL Server Management Studio”，在“对象资源管理器”窗口里展开“数据库”目录，鼠标右键单击“pubs”，在弹出的快捷菜单里选择“任务”，如图 11.5 所示。

（2）单击菜单项“备份”，弹出如图 11.6 所示“备份数据库”对话框。

（3）在“备份类型”，下拉列表框里选择“完整”。

（4）在图 11.6 所示对话框里单击“选项”标签，弹出如图 11.7 所示的“选项”对话框，根据需要设置以下各种选项。

① 是否覆盖媒体：选择“追加到现有备份集”单选框，则不覆盖现有备份集，将数据库备份追加到备份集里，同一个备份集里可以有多个数据库备份信息；如果选择“覆盖所有现有备份集”单选框，则将覆盖现有备份集，以前在该备份集里的备份信息将无法重新读取。

② 是否检查媒体集名称和备份集过期时间：如果需要可以选择“检查媒体集名称和备份集过期时间”复选框，来要求备份操作验证备份集的名称和过期时间；在“媒体集名称”文本框里可以输入要验证的媒体集名称。

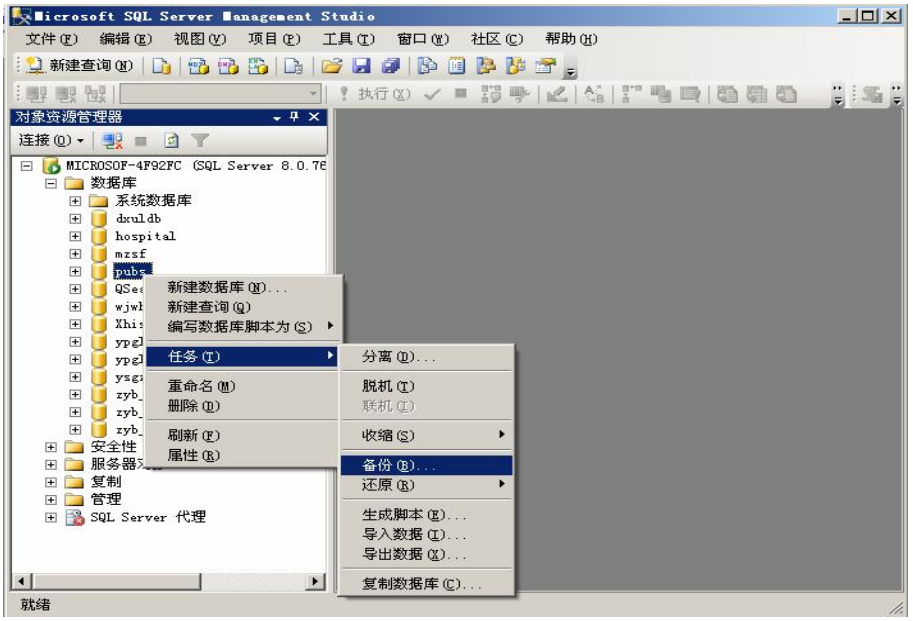


图 11.5　鼠标右键单击“pubs”数据库，选择“任务”

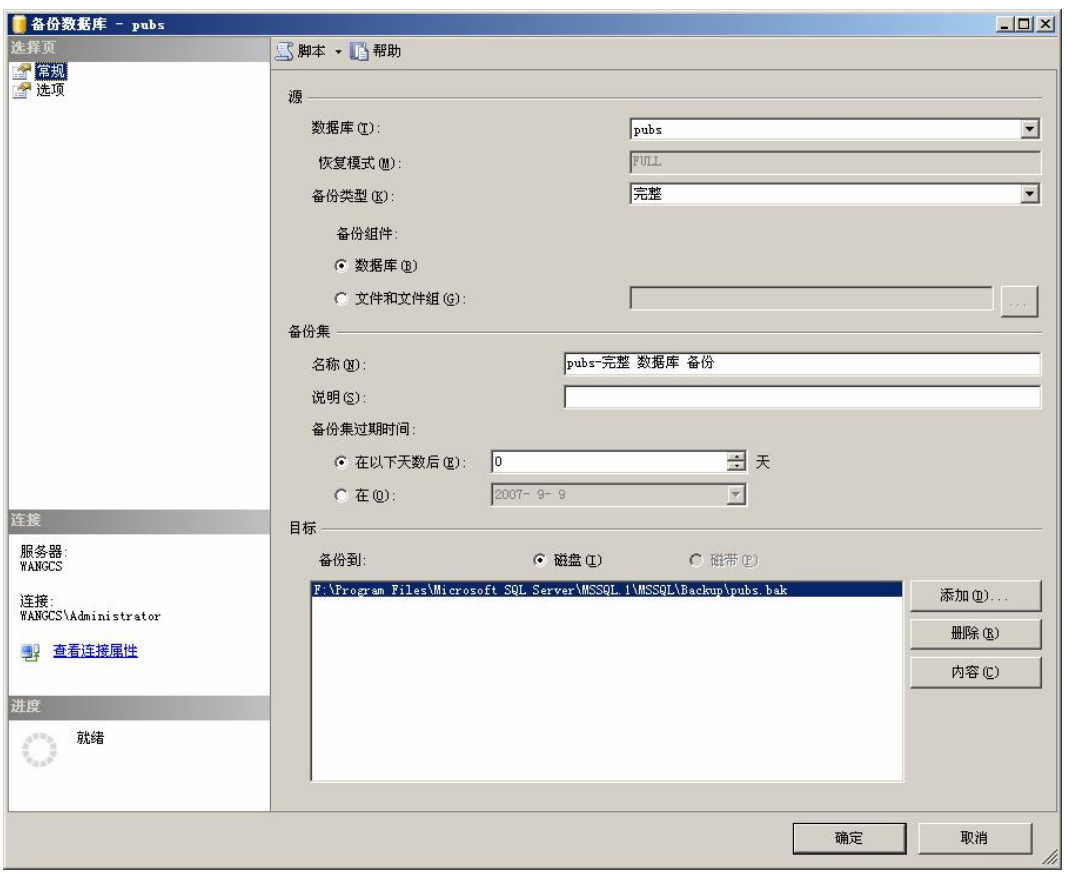


图 11.6　“备份数据库”对话框

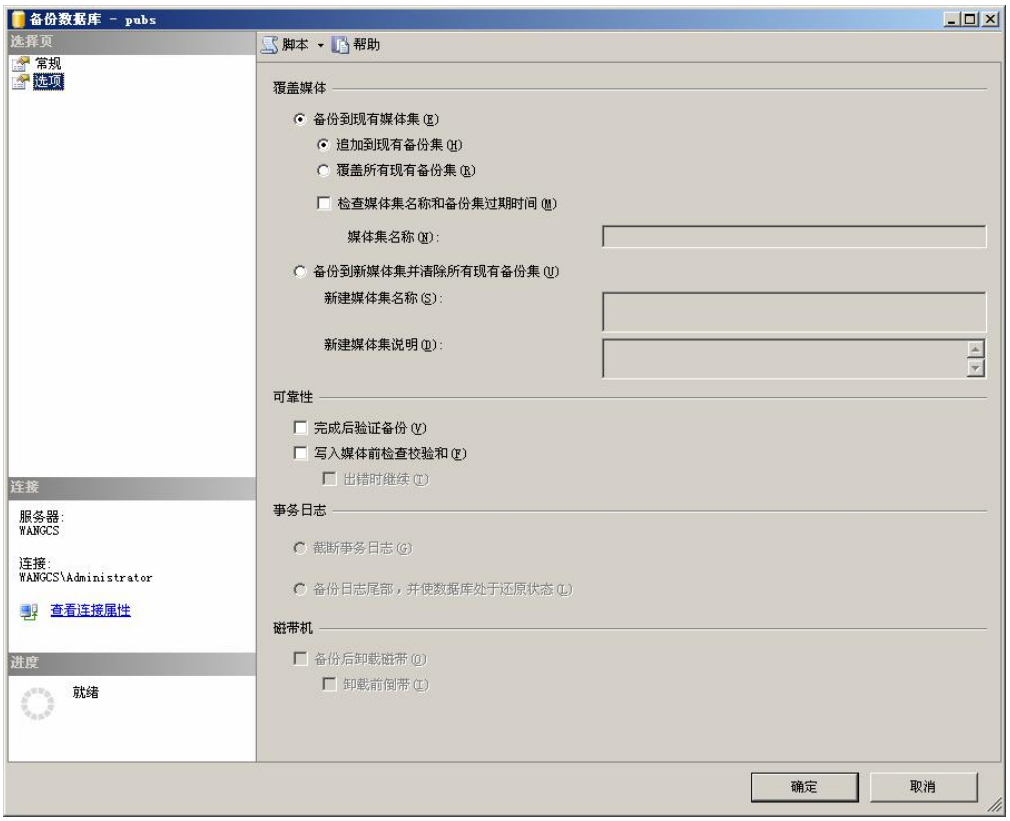


图 11.7　“选项”对话框

③ 是否使用新媒体集：选择“备份到新媒体集并清除所有现在备份集”可以清除以前的媒体集，并使用新的媒体集备份数据库。在“新建媒体集名称”文本框里输入媒体集的新名称，在“新建媒体集说明”文本框里输入新建媒体集的说明。

④ 设置数据库备份的可靠性：选择“完成后验证备份”复选框将会验证备份集是否完整以及所有卷是否都可读；选择“写入媒体前检查校验和”复选框将会在写入备份媒体前验证校验和，如果选中此项，可能会增大工作负荷，并降低备份操作的备份吞吐量。

（5）单击“确定”按钮，SQL Server 2005 开始执行备份操作。

2．使用 Transact-SQL 语句进行完全备份

完全备份语法代码如下：

```
BACKUP DATABASE { database_name | @database_name_var }
TO < backup_device > [ ,...n ]
[ [ MIRROR TO < backup_device > [ ,...n ] ] [ ...next-mirror ] ]
[ WITH
  [ BLOCKSIZE = { blocksize | @blocksize_variable } ]
  [ [ , ] { CHECKSUM | NO_CHECKSUM } ]
   [ [ , ] { STOP_ON_ERROR | CONTINUE_AFTER_ERROR } ]
   [ [ , ] DESCRIPTION = { 'text' | @text_variable } ]
   [ [ , ] DIFFERENTIAL ]
```

```
    [ [ , ] EXPIREDATE = { date | @date_var }
    | RETAINDAYS = { days | @days_var } ]
    [ [ , ] PASSWORD = { password | @password_variable } ]
    [ [ , ] { FORMAT | NOFORMAT } ]
    [ [ , ] { INIT | NOINIT } ]
    [ [ , ] { NOSKIP | SKIP } ]
    [ [ , ] MEDIADESCRIPTION = { 'text' | @text_variable } ]
    [ [ , ] MEDIANAME = { media_name | @media_name_variable } ]
    [ [ , ] MEDIAPASSWORD = { mediapassword | @mediapassword_variable } ]
    [ [ , ] NAME = { backup_set_name | @backup_set_name_var } ]
    [ [ , ] { NOREWIND | REWIND } ]
    [ [ , ] { NOUNLOAD | UNLOAD } ]
    [ [ , ] RESTART ]
    [ [ , ] STATS [ = percentage ] ]
    [ [ , ] COPY_ONLY ]
]
```

主要参数说明如下。

database_name：数据库名。

@database_name_var：数据库名称变量。

<backup_device>：备份设备名称。

MIRROR TO：表示备份设备组是包含 2～4 个镜像服务器的镜像媒体集中的一个镜像。若要指定镜像媒体集，则针对第一个镜像服务器设备使用 TO 子句，后跟最多 3 个 MIRROR TO 子句。

BLOCKSIZE：用字节数来指定物理块的大小，支持的大小为 512、1 024、2 048、4 096、8 192、16 384、32 768 和 65 536（64 K）Byte。

BUFFERCOUNT：指定用于备份或还原操作的 I/O 缓冲区总数。可以指定任何正　整数。

CHECKSUM | NO_CHECKSUM：是否启用校检和。

STOP_ON_ERROR | CONTINUE_AFTER_ERROR：校检和失败时是否还继续备份 操作。

DESCRIPTION：此次备份数据的说明文字内容。

DIFFERENTIAL：只做差异备份，如果没有该参数，则做完整备份。

EXPIREDATE：指定备份集到期和允许被覆盖的日期。

RETAINDAYS：指定可以覆盖该备份媒体集必须经过的天数。

PASSWORD：为备份集设置密码，如果为备份集定义了密码，则必须提供此密码才能对该备份集执行还原操作。

FORMAT | NOFORMAT：指定创建或不创建新的媒体集。

INIT：指定覆盖所有备份集，但是保留媒体标头。如果指定了 INIT，将覆盖该设备上所有现有的备份集。

NOINIT：表示备份集将追加到指定的媒体集上，以保留现有的备份集。

NOSKIP|SKIP：指定是否在覆盖媒体上的所有备份集之前先检查它们的过期日期。

MEDIADESCRIPTION：指定媒体集的自由格式文本说明，最多为 255 个字符。

MEDIANAME：指定整个备份媒体集的媒体名称。

MEDIAPASSWORD：为媒体集设置密码。MEDIAPASSWORD 是一个字符串。如果为媒体集定义了密码，则在该媒体集上创建备份集之前必须提供此密码。另外，从该媒体集执行任何还原操作时也必须提供该密码。

NAME：指定备份集的名称。名称最长可达 128 个字符。

REWIND：指定 SQL Server 将释放和重绕磁带。

NOREWIND：指定在备份操作之后 SQL Server 使磁带一直处于打开状态。

UNLOAD：指定在备份完成后自动重绕并卸载磁带。

NOUNLOAD：指定在备份操作之后磁带将继续加载在磁带机中。

RESTART：在 SQL Server 2005 该参数已经失效，在以前版本中，表示现在要做的备份是要继续前次被中断的备份作业。

STATS：该参数可以让 SQL Server 每备份一定百分比的数据就显示备份进度信息。

COPY_ONLY：指定此备份不影响正常的备份序列。

【例 11.3】将数据库“pubs”的数据完全备份到文件 c:\pubs.bak 中。

```
BACKUP DATABASE pubs TO DISK = 'c:\pubs.bak'
```

【例 11.4】将数据库“pubs”的数据完全备份到名为“设备 1”的备份设备上。

```
BACKUP DATABASE pubs TO  设备 1
```

11.3.2　差异备份

差异备份，是指只备份自上次完全备份后，发生了更改的数据。由于差异备份是备份完全备份后发生了更改的数据，因此在做差异备份前，必须至少有一次完全备份。

由于差异备份仅包含了完全备份后发生了更改的数据，因此仅使用差异备份文件无法还原数据。要还原差异备份，必须先还原差异备份前的最近一次完全备份，然后在此基础上还原差异备份。

差异备份生成的备份文件大小和备份需要的时间，取决于最近一次完全备份后，数据变化的多少。数据变化越多，备份处理需要的时间越长，备份文件越大。当然，如果仅仅是大量删除数据，则差异备份生成的备份文件不会很大，备份时间也不会太长。

1. 通过 SQL Server Management Studio 实现差异备份

（1）按照完全备份中的步骤，打开如图 11.6 所示的“备份数据库”对话框。

（2）在“备份类型”下拉列表框里选择“差异”。

（3）根据需要设置其他选项。

（4）单击“确定”按钮，SQL Server 2005 开始执行备份操作。

2. 使用 Transact-SQL 语句进行差异备份

差异备份语法同完全备份的语法，在此不再赘述。

【例 11.5】将数据库“pubs”的差异数据备份到文件 c:\pubs.bak 中。

```
BACKUP DATABASE pubs TO DISK = 'c:\pubs.bak' DIFFERENTIAL
```

11.3.3 事务日志备份

日志备份，是指备份自上次备份后对数据库执行的所有事物的一系列记录。所谓的上次备份，可以是完全备份、差异备份或者日志备份。日志备份前，至少有一次完全备份。还原日志备份的时候，必须先还原完全备份。如果完全备份后，在要还原的日志备份前做过差异备份，则还要还原差异备份，然后再按照日志备份的先后顺序，依次还原各日志备份。

由于日志备份仅备份自上次备份后对数据库执行的所有事务的一系列记录（可以简单地理解为自上次备份以来的数据变化），所以它生成的备份文件小，备份需要的时间也短，对SQL Server 服务性能的影响也小，适于经常备份。但是其还原过程最繁琐，不但要先还原日志备份之前做的完全备份和差异备份（如果有的话），在还原日志备份时，还必须依照日志备份的时间顺序依次还原所有的日志备份。

1. 通过 SQL Server Management Studio 实现事务日志备份

（1）按照完全备份中的步骤，打开如图 11.6 所示的“备份数据库”对话框。

（2）在“备份类型”下拉列表框里选择“事务日志”。

（3）根据需要设置其他选项。

（4）单击“确定”按钮，SQL Server 2005 开始执行备份操作。

2. 使用 Transact-SQL 语句进行事务日志备份

事务日志备份语法代码如下：

```
BACKUP LOG { database_name | @database_name_var }
TO < backup_device > [ ,...n ]
[ [ MIRROR TO < backup_device > [ ,...n ] ] [ ...next-mirror ] ]
[ WITH
  [ BLOCKSIZE = { blocksize | @blocksize_variable } ]
  [ [ , ] { CHECKSUM | NO_CHECKSUM } ]
   [ [ , ] { STOP_ON_ERROR | CONTINUE_AFTER_ERROR } ]
   [ [ , ] DESCRIPTION = { 'text' | @text_variable } ]
   [ [ , ] DIFFERENTIAL ]
   [ [ , ] EXPIREDATE = { date | @date_var }
   | RETAINDAYS = { days | @days_var } ]
   [ [ , ] PASSWORD = { password | @password_variable } ]
   [ [ , ] { FORMAT | NOFORMAT } ]
   [ [ , ] { INIT | NOINIT } ]
   [ [ , ] { NOSKIP | SKIP } ]
   [ [ , ] MEDIADESCRIPTION = { 'text' | @text_variable } ]
   [ [ , ] MEDIANAME = { media_name | @media_name_variable } ]
   [ [ , ] MEDIAPASSWORD = { mediapassword | @mediapassword_variable } ]
   [ [ , ] NAME = { backup_set_name | @backup_set_name_var } ]
   [ [ , ] { NOREWIND | REWIND } ]
```

```
    [ [ , ] { NOUNLOAD | UNLOAD } ]
    [ [ , ] RESTART ]
    [ [ , ] STATS [ = percentage ] ]
    [ [ , ] COPY_ONLY ]
]
```

从以上代码可以看出，事务日志与完整备份的代码大同小异，只是将 BACKUP BATABASE 改为了 BACKUP LOG。

【例 11.6】将数据库“pubs”的事务日志备份到文件 c:\pubs_log.bak 中。

```
BACKUP LOG pubs TO DISK = 'c:\pubs_log.bak'
```

11.3.4　文件和文件组备份

如果在创建数据库时，为数据库创建了多个数据库文件或文件组，可以使用文件和文件组备份方式。使用文件和文件组备份方式可以只备份数据库中的某些文件。该备份方式在数据库文件非常庞大的时候十分有效。由于每次只备份一个或几个文件或文件组，可以分多次来备份数据库，避免大型数据库备份的时间过长。另外，由于只备份其中一个或多个数据文件，那么当数据库里的某个或某些文件损坏时，可以只还原损坏的文件或文件组。

数据文件和文件组的还原操作在 4 种备份方法中是最复杂的。对于操作者而言，不但要熟练地掌握备份和还原方法，还必须清楚数据库的文件结构，否则还原操作往往会失败。

1. 通过 SQL Server Management Studio 实现文件和文件组备份

（1）按照完全备份中的步骤，打开如图 11.6 所示的“备份数据库”对话框。

（2）选中“文件和文件组”单选框，此时会弹出如图 11.8 所示的“选择文件和文件组”对话框。在该对话框里可以选择要备份的文件和文件组，选择完毕后单击“确定”按钮返回。

（3）所有选项设置完毕后单击“确定”，开始执行备份操作。

图 11.8　“选择文件和文件组”对话框

2. 使用 Transact-SQL 语句进行文件和文件组备份

文件和文件组备份语法代码如下：

```
BACKUP DATABASE { database_name | @database_name_var }
<file_or_filegroup> [ ,...f ]
TO < backup_device > [ ,...n ]
[[ MIRROR TO < backup_device > [ ,...n ] ] [ ...next-mirror ] ]
[WITH
[BLOCKSIZE = { blocksize | @blocksize_variable } ]
```

```
[[ , ] { CHECKSUM | NO_CHECKSUM } ]
[[ , ] { STOP_ON_ERROR | CONTINUE_AFTER_ERROR } ]
[[ , ] DESCRIPTION = { 'text' | @text_variable } ]
[[ , ] DIFFERENTIAL ]
[[ , ] EXPIREDATE = { date | @date_var }
  | RETAINDAYS = { days | @days_var } ]
[[ , ] PASSWORD = { password | @password_variable } ]
[[ , ] { FORMAT | NOFORMAT } ]
[[ , ] { INIT | NOINIT } ]
[[ , ] { NOSKIP | SKIP } ]
[[ , ] MEDIADESCRIPTION = { 'text' | @text_variable } ]
[[ , ] MEDIANAME = { media_name | @media_name_variable } ]
[[ , ] MEDIAPASSWORD = { mediapassword | @mediapassword_variable } ]
[[ , ] NAME = { backup_set_name | @backup_set_name_var } ]
[[ , ] { NOREWIND | REWIND } ]
[[ , ] { NOUNLOAD | UNLOAD } ]
[[ , ] RESTART ]
[[ , ] STATS [ = percentage ] ]
[[ , ] COPY_ONLY ]
]
--Specifying a file or filegroup
<file_or_filegroup> :: =
  {
  FILE = { logical_file_name | @logical_file_name_var }
  |
  FILEGROUP = { logical_filegroup_name | @logical_filegroup_name_var }
  | READ_WRITE_FILEGROUPS
}
```

从以上代码可以看出，文件和文件组的备份与完全备份的代码大同小异，不同的是，在“TO <backup_device>”之前多了一句“<file_or_filegroup>”。该语法块里有如下参数。

FILE：给一个或多个包含在数据库备份中的文件命名。

FILEGROUP：给一个或多个包含在数据库备份中的文件组命名。

READ_WRITE_FILEGROUPS：指定部分备份，包括主文件组和所有具有读写权限的辅助文件组。创建部分备份时需要此关键字。

【例 11.7】将数据库“pubs”中的“pubs_Data”文件备份到到名为“设备 2”的备份设备上。

```
BACKUP DATABASE pubs FILE='pubs_Data' TO '设备2'
```

11.4　数据库还原

11.4.1　数据库还原方式介绍

数据库还原方式有 4 种。

1．完全备份的还原

无论是完全备份、差异备份还是事务日志备份的还原，第一步都要先做完全备份的还原。完全备份的还原只需要还原完全备份文件即可。

2．差异备份的还原

差异备份的还原一共需要两个步骤，第一步还原完全备份，第二步还原差异备份。

3．事务日志备份的还原

还原事务日志备份的步骤比较多，因为事务日志备份相对而言会做得比较频繁一些。其步骤是先还原完全备份，然后按时间先后顺序依次还原差异备份，最后依次还原每一个事务日志备份。

4．文件和文件组备份的还原

通常只有数据库中某个文件或文件组损坏了才会使用这种还原模式。

11.4.2　数据库还原

1．通过 SQL Server Management Studio 进行数据库还原

（1）进行数据库完全备份、差异备份和事务日志备份还原的步骤如下。

① 启动“SQL Server Management Studio”，展开“对象资源管理器”树型目录，鼠标右键单击“数据库”，在弹出的快捷菜单里选择“还原数据库”，弹出如图 11.9 所示的“还原数据库”对话框。

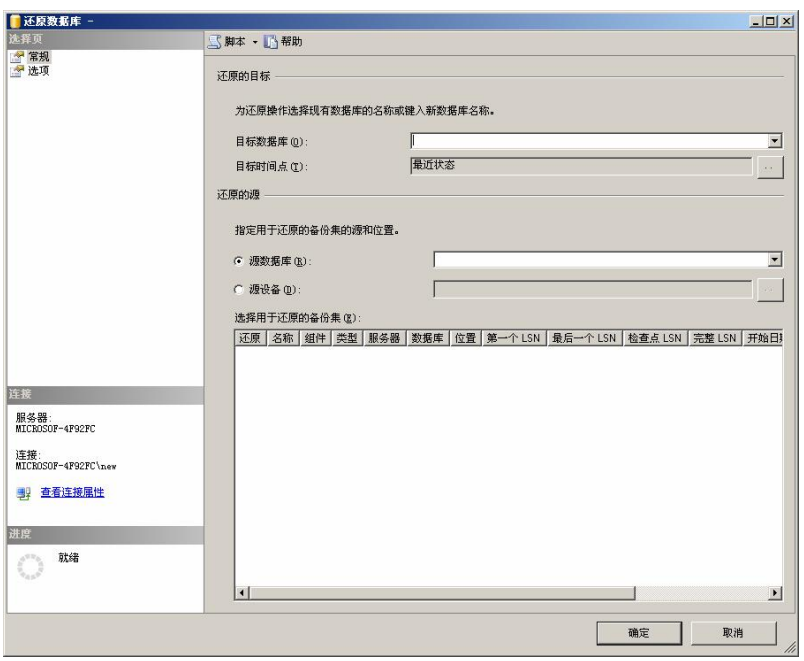


图 11.9　“还原数据库”对话框

② 在“目标数据库”下拉列表框里可以选择或键入要还原的数据库名。

③ 如果备份文件或备份设备里的备份集很多，还可以选择“目标时间点”，只要有事务日志备份支持，就可以还原到某个时刻的数据库状态。在默认情况下该项为“最近状态”。

④ 在“还原的源”区域里，指定用于还原的备份集的源和位置。

如果选择“目标数据库”单选框，则从 Msdb 数据库中的备份历史记录里查得可用的备份，并显示在“选择用于还原的备份集”区域里。此时不需要指定备份文件的位置或指定备份设备，SQL Server 会自动根据备份记录来找到这些文件。

如果选择“源设备”单选框，则要指定还原的备份文件或备份设备。点击“…”按钮，弹出如图 11.10 所示“指定备份”对话框。在“备份媒体”下拉列表框里可以选择是备份文件还是备份设备，选择完毕后单击“添加”按钮，将备份文件或备份设备添加进来，之后，返回到如图 11.9 所示对话框。

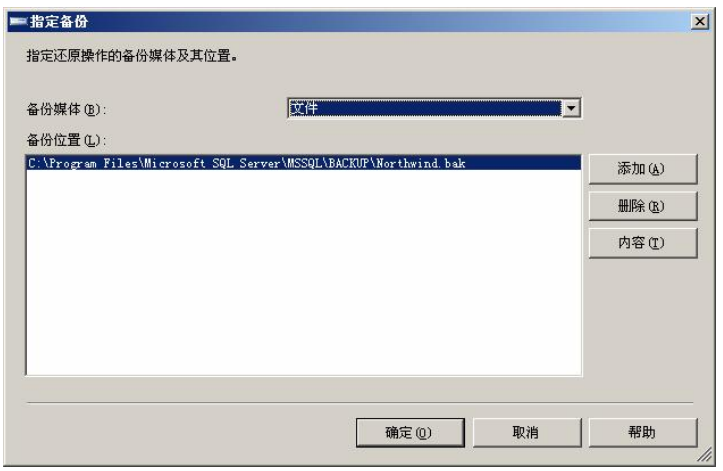


图 11.10 “指定备份”对话框

“选择用于还原的备份集”区域：在该区域里列出了所有可用的备份集。

- 如果“目标时间点”为“最近状态”，“还原的源”为“源数据库”，则该区域显示的是最后一次完整备份到现在的所有可用备份集。
- 如果“目标时间点”为“最近状态”，“还原的源”为“源设备”，则该区域显示的是备份文件或备份设备里的所有可用备份集。
- 如果“目标时间点”为指定的时间，“还原的源”为“源数据库”，则该区域显示的是从指定的时间前一个完整备份到目前为止的所有非完整备份。
- 如果“目标时间点”为指定的时间，“还原的源”为“源设备”，则该区域显示的是备份文件或备份设备里，指定时间之后第一个完整备份到目前为止的所有备份。

在“选择用于还原的备份集”里可以选择完整备份、差异备份或事务日志备份。SQL Server 2005 十分智能，如果选择差异备份，系统会自动将上一个完整备份选上；如果选日志备份，系统也会自动将上一个完整备份以及所需要的差异备份和日志备份都选上。换句话说，只要选择想恢复到的那个备份集即可，系统会自动选上要恢复到这个备份集所需的其他备份集。

⑤ 如果没有其他的需要，可以单击“确定”按钮进行还原操作，也可以在图 11.9 所示对话框里选择“选项”标签，进入如图 11.11 所示的“选项”对话框。

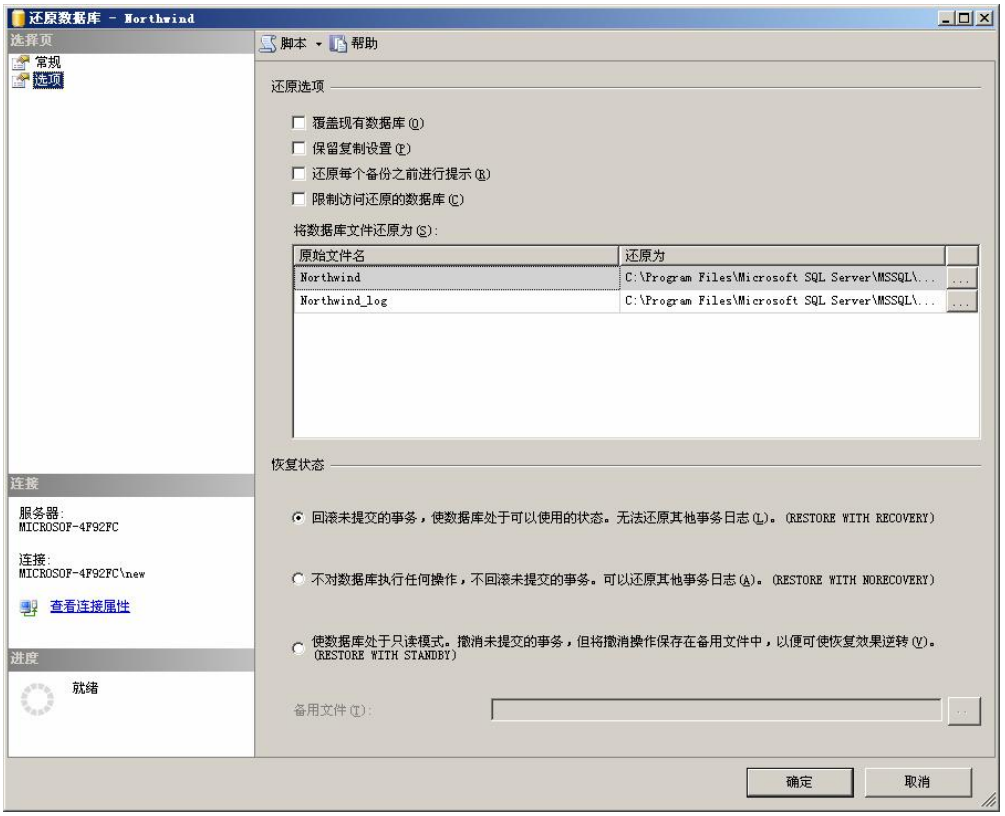


图 11.11　“选项”对话框

⑥ 在图 11.11 所示“选项”对话框可以设置如下选项。

- 还原选项

如果选择了“覆盖现有数据库”，则会覆盖所有现有数据库以及相关文件，包括已存在的同名的其他数据库或文件。

如果选择了“保留复制设置”，则会将已发布的数据库还原到创建该数据库的服务器之外的服务器上，保留复制设置。

如果选择了“还原每个备份之前进行提示”，则在还原每个备份设备前都会要求确认。

如果选择了“限制访问还原的数据库”，则使还原的数据库仅供 db_owner、dbcreator 或 sysadmin 的成员使用。

- 将数据库文件还原为

在该区域里可以更改目的文件的路径和名称。

- 恢复状态

如果选择了“回滚未提交的事务，使数据库处于可以使用状态。无法还原其他事务日志”单选框，则数据库在还原后进入可正常使用的状态，并自动恢复尚未完成的事务。如果本次还原是还原的最后一次操作，可以选择该项。

如果选择了“不对数据库执行任何操作，不回滚未提交的事务。可以还原其他事务日志”单选框，则在还原后数据库仍然无法正常使用，也不恢复未完成的事务操作，但可再继续还原事务日志备份或差异备份，使数据库能恢复到最接近目前的状态。

如果选择了“使数据库处于只读模式。撤消未提交的事务，但将撤消操作保存在备用文件中，以便可使恢复效果逆转”单选框，则在还原后恢复未完成事务的操作，并使数据库处于只读状态。为了可再继续还原后的事务日志备份，还必须指定一个还原文件来存放被恢复的事务内容。

⑦ 单击“确定”按钮，开始执行还原操作。

（2）进行数据库文件和文件组备份的还原的步骤如下。

① 启动“SQL Server Management Studio”，展开“对象资源管理器”树型目录，鼠标右键单击“数据库”，在弹出的快捷菜单里选择“还原文件和文件组”，弹出如图 11.12 所示的“还原文件和文件组”对话框。

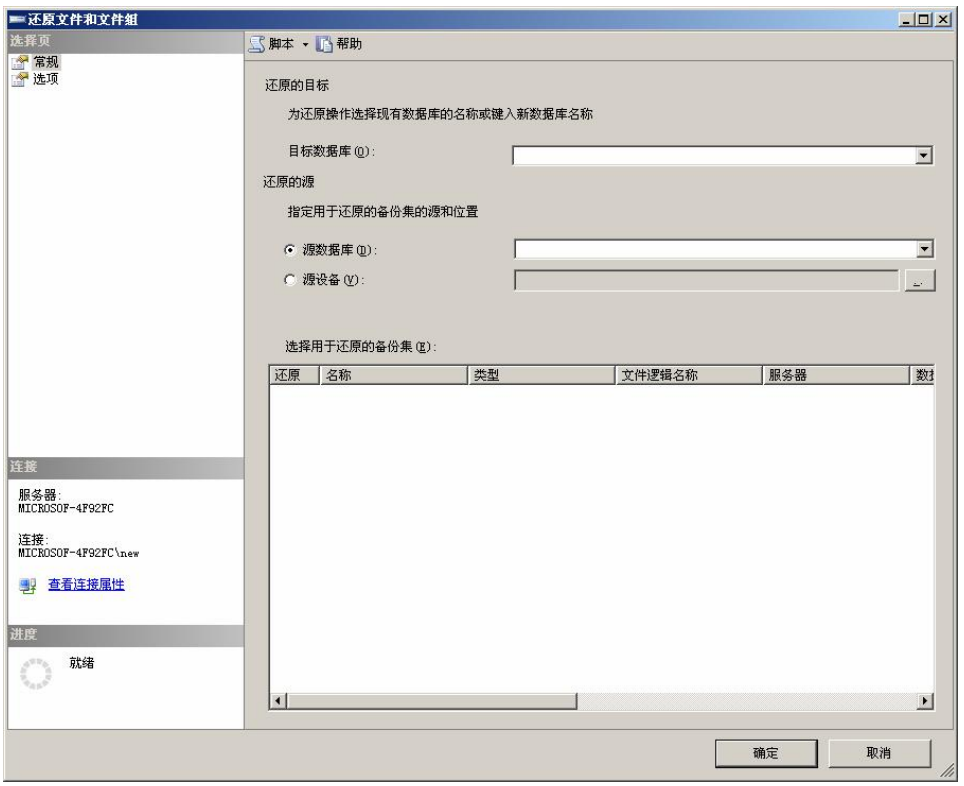


图 11.12 “还原文件和文件组”对话框

② 在图 11.12 所示的“还原文件和文件组”对话框里，可以设置以下选项。

- “目标数据库”：在该下拉列表框里可以选择或输入要还原的数据库名。
- “还原的源”：在该区域里可以选择要用来还原的备份文件或备份设备，用法与还原数据库完整备份中的用法一样，在此不再赘述。
- “选择用于还原的备份集”：在该区域里可以选择要还原的备份集。

③ 选择完毕后可以单击“确定”按钮开始执行还原操作，也可以选择“选项”进行进一步设置。

注意 在进行完文件和文件组备份之后，还必须进行一次事务日志备份，否则无法还原文件和文件组备份。

2．使用 Transact-SQL 语句进行数据库备份还原

T-SQL 语言里提供了 RESTORE DATABASE 语句来恢复数据库备份，用该语句可以恢复完全备份、差异备份、文件和文件组备份。如果要还原事务日志备份可以用 RESTORE LOG 语句。虽然 RESTORE DATABASE 语句可以恢复完全备份、差异备份、文件和文件组备份，但是在恢复完整备份、差异备份与文件和文件组备份的语法上有一定的出入，下面分别介绍几种类型备份的还原方法。

（1）还原完整备份。还原完整备份的语法如下：

```
RESTORE DATABASE { database_name | @database_name_var }
[ FROM <backup_device> [ ,...n ] ]
[ WITH
  [ [ , ] { CHECKSUM | NO_CHECKSUM } ]
  [ [ , ] { STOP_ON_ERROR | CONTINUE_AFTER_ERROR } ]
  [ [ , ] FILE = { file_number | @file_number } ]
  [ [ , ] KEEP_REPLICATION ]
  [ [ , ] MEDIANAME = { media_name | @media_name_variable } ]
  [ [ , ] MEDIAPASSWORD = { mediapassword |@mediapassword_variable } ]
  [[,] MOVE 'logical_file_name' TO 'operating_system_file_name' ][ ,...n ]
  [ [ , ] PASSWORD = { password | @password_variable } ]
  [ [ , ] { RECOVERY | NORECOVERY | STANDBY = {standby_file_name
        | @standby_file_name_var }
  }]
  [ [ , ] REPLACE ]
  [ [ , ] RESTART ]
  [ [ , ] RESTRICTED_USER ]
  [ [ , ] { REWIND | NOREWIND } ]
  [ [ , ] STATS [ = percentage ] ]
  [ [ , ] {STOPAT = { date_time | @date_time_var }
  |STOPATMARK = { 'mark_name' | 'lsn:lsn_number' }
      [ AFTER datetime ]
    |STOPBEFOREMARK = { 'mark_name' | 'lsn:lsn_number' }
      [ AFTER datetime ]
  } ]
  [ [ , ] { UNLOAD | NOUNLOAD } ]
]
[;]
<backup_device>
::=
{
  { logical_backup_device_name |@logical_backup_device_name_var }
```

```
    | { DISK | TAPE } = { 'physical_backup_device_name' |
      @physical_backup_device_name_var }
}
```

其中大多数参数在介绍备份数据时已经介绍过了，下面介绍前面没有提到的参数。

ENABLE_BROKER：启动 Service Broker 以便消息可以立即发送。

ERROR_BROKER_CONVERSATIONS：发生错误时结束所有会话，并产生一个错误指出数据库已附加或还原。此时 Service Broker 将一直处于禁用状态，直到此操作完成，再重新启用。

KEEP_REPLICATION：将复制设置为与日志传送一同使用。设置该参数后，在备用服务器上还原数据库时，可防止删除复制设置。该参数不能与 NORECOVERY 参数同时使用。

MOVE:将逻辑名指定的数据文件或日志文件还原到指定的位置,相当于图 11.11 中所示的“将数据库文件还原为”功能。

NEW_BROKER:使用该参数会在原数据库和备份数据库中都创建一个新的 service_ broker_guid 值，并通过清除结束所有会话端点。此时 Service Broker 已启用，但未向远程会话端点发送消息。

RECOVERY：回滚未提交的事务，使数据库处于可以使用状态，但无法还原其他事务日志。

NORECOVERY：不对数据库执行任何操作，不回滚未提交的事务，可以还原其他事务日志。

STANDBY:使数据库处于只读模式，撤消未提交的事务，但将撤消操作保存在备用文件中，以便可使恢复效果逆转。

standby_file_name | @standby_file_name_var：指定一个允许撤消恢复效果的备用文件或变量。

REPLACE：覆盖所有现有数据库以及相关文件，包括已存在的同名的其他数据库或文件。

RESTART：指定 SQL Server 应重新启动被中断的还原操作。RESTART 从中断点重新启动还原操作。

RESTRICTED_USER：还原后的数据库仅供 db_owner、dbcreator 或 sysadmin 的成员使用。

STOPAT：将数据库还原到在指定的日期和时间的状态。

STOPATMARK：恢复为已标记的事务或日志序列号（LSN）。恢复中包括带有已命名标记或 LSN 的事务，仅当该事务最初于实际生成事务时已获得提交，才可进行本次提交。

TOPBEFOREMARK：恢复为已标记的事务或日志序列号。恢复中不包括带有已命名标记或 LSN 的事务，在使用 WITH RECOVERY 时，该事务将回滚。

【例 11.8】用名为“c:\pubs.bak”的完全备份文件来还原“pubs”数据库。

```
USE master
RESTORE DATABASE pubs FROM DISK = 'c:\pubs.bak'
```

【例 11.9】用名为“设备 1”的备份设备来还原“pubs”数据库。

```
USE master
RESTORE DATABASE pubs FROM 设备 1
```

（2）还原差异备份。还原差异备份的语法与还原完整备份的语法是一样的，只是在还原差异备份时，必须要先还原完整备份再还原差异备份，因此还原差异备份必须要分两步完成。完整备份与差异备份数据可能在同一个备份文件或备份设备中，也有可能是在不同的备份文件或备份设备中。如果在同一个备份文件或备份设备中,则必须要用 file 参数来指定备份集。无论是备份集是否在同一个备份文件（备份设备）中，除了最后一个还原操作，其他所有还

原操作都必须要加上 NORECOVERY 或 STANDBY 参数。

【例 11.10】用名为“pub 完全备份设备”的备份设备来还原“pubs”数据库的完全备份，再用名为“pub 差异备份设备”的备份设备来还原差异备份。

```
USE Master
RESTORE DATABASE pubs
    FROM pub 完全备份设备 NORECOVERY
GO
RESTORE DATABASE pubs
    FROM pub 差异备份设备
GO
```

（3）还原事务日志备份。SQL Server 2005 中已经将事务日志备份看成和完整备份、差异备份一样的备份集。因此，还原事务日志备份也可以和还原差异备份一样，只要知道它在备份文件或备份设备里的顺序即可。

与还原差异备份相同，还原事务日志备份必须要先还原之前的完整备份，除了最后一个还原操作，其他所有还原操作都必须要加上 NORECOVERY 或 STANDBY 参数。

【例 11.11】用名为“pub 完全备份设备”的备份设备来还原“pubs”数据库的完全备份，再用名为“pub 日志备份设备”的备份设备来还原日志备份。

```
USE Master
RESTORE DATABASE pubs
    FROM pub 完全备份设备 NORECOVERY
GO
RESTORE LOG pubs
    FROM pub 日志备份设备
GO
```

（4）还原文件和文件组备份。还原文件和文件组备份也可以使用 RESTORE DATABASE 语句，但是必须要在数据库名与 FROM 之间加上“FILE”或“FILEGROUP”参数来指定要还原的文件或文件组。

【例 11.12】用名为“pub 完全备份设备”的备份设备，还原“pubs”数据库的文件和文件组，再用名为“pub 日志备份设备”来还原事务日志备份。

```
USE Master
RESTORE DATABASE pubs
    FILEGROUP = 'PRIMARY'
    FROM pub 完全备份设备 NORECOVERY
GO
RESTORE LOG pubs
    FROM pub 日志备份设备
GO
```

11.5　数据库的分离和附加

Microsoft SQL Server 2005 允许分离数据库的数据和事务日志文件，然后将其重新附加到另一台服务器，甚至同一台服务器上。分离数据库将从 SQL Server 中删除数据库，但是不改变组成该数据库的数据和事务日志文件。这些数据和事务日志文件可以用来将数据库附加到任何 SQL Server 实例上，包括从中分离该数据库的服务器。这时数据库的使用状态与它分离时的状态完全相同。

注意　不能分离 Master、Model、Tempdb 等系统数据库。

11.5.1　分离数据库

分离数据库的操作步骤如下。

（1）启动“SQL Server Management Studio”，在“对象资源管理器”窗口里展开“数据库”树型目录，鼠标右键单击“pubs”数据库，在弹出的快捷菜单里选择“任务”，如图 11.13 所示。

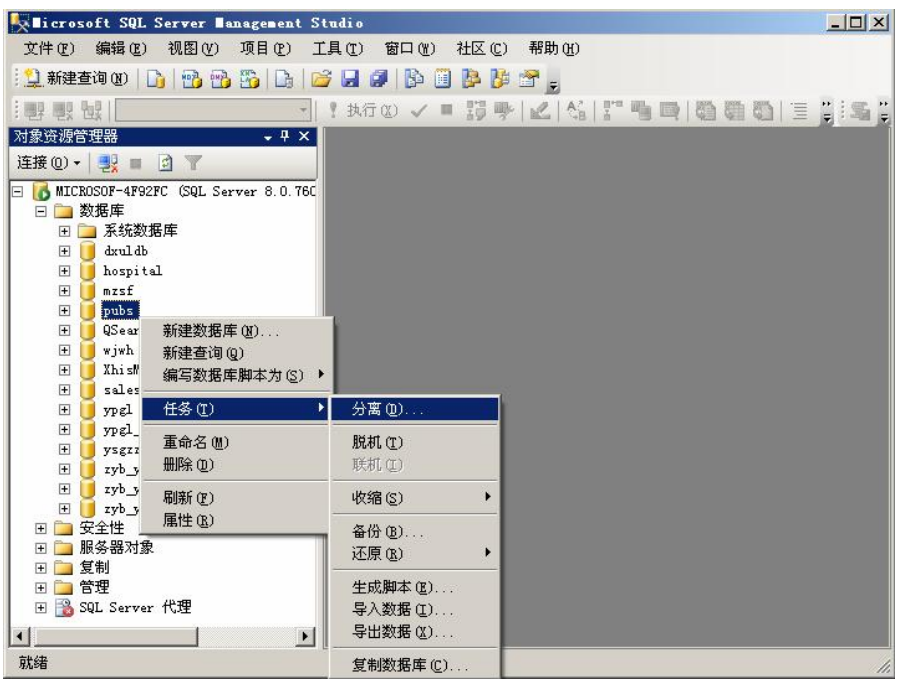


图 11.13　鼠标右键单击“pubs”数据库，选择“任务”

（2）单击“分离”菜单项，弹出如图 11.14 所示的“分离数据库”对话框。图中数据库“pubs”的状态为就绪，表示可以分离。

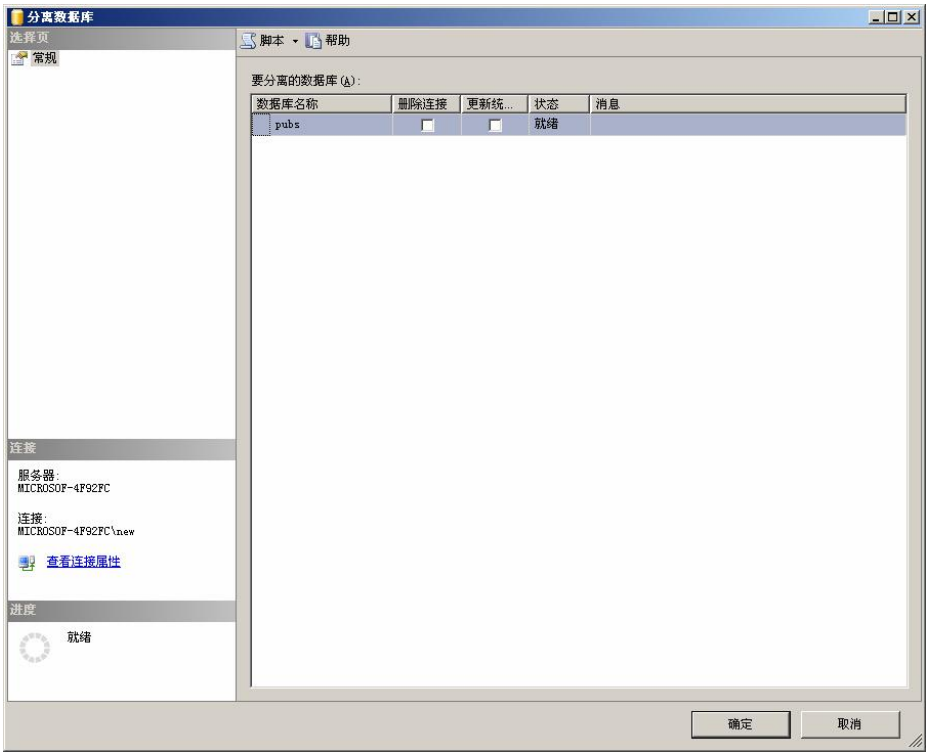


图 11.14　“分离数据库”对话框

（3）单击“确定”按钮，开始执行分离数据库操作。

11.5.2　附加数据库

附加数据库的操作步骤如下。

（1）启动“SQL Server Management Studio”，展开“对象资源管理器”树型目录，鼠标右键单击“数据库”节点，如图 11.15 所示。

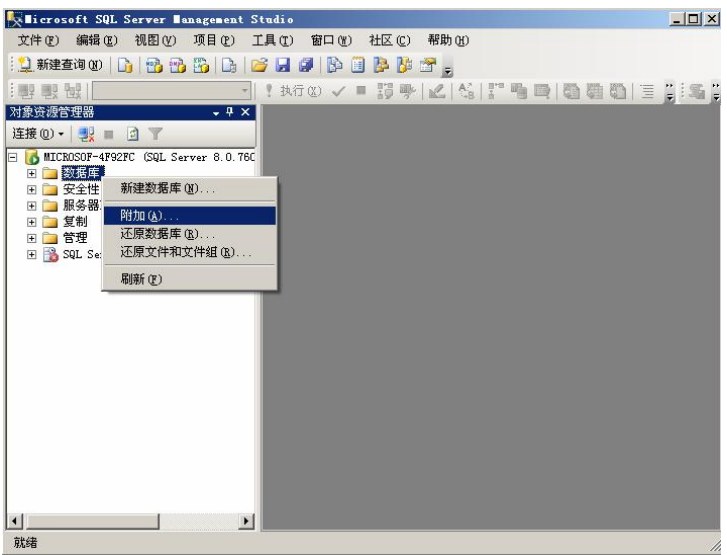


图 11.15　鼠标右键单击“数据库”节点

（2）单击“附加”菜单项，弹出如图 11.16 所示的“附加数据库”对话框。

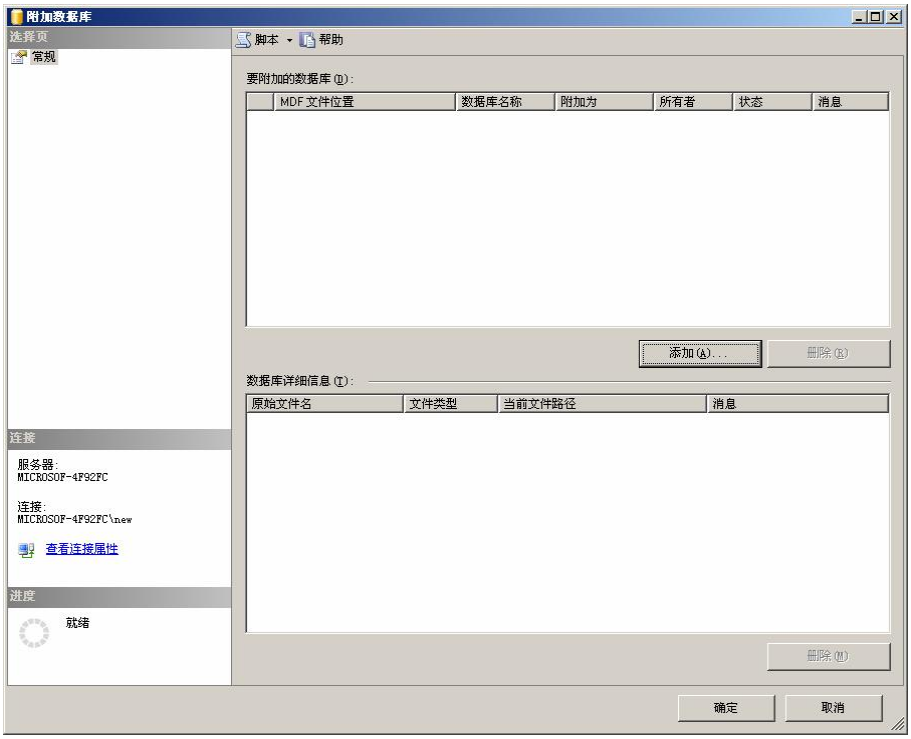


图 11.16 “附加数据库”对话框

（3）单击“添加”按钮，弹出如图 11.17 所示的“定位数据库文件”对话框。

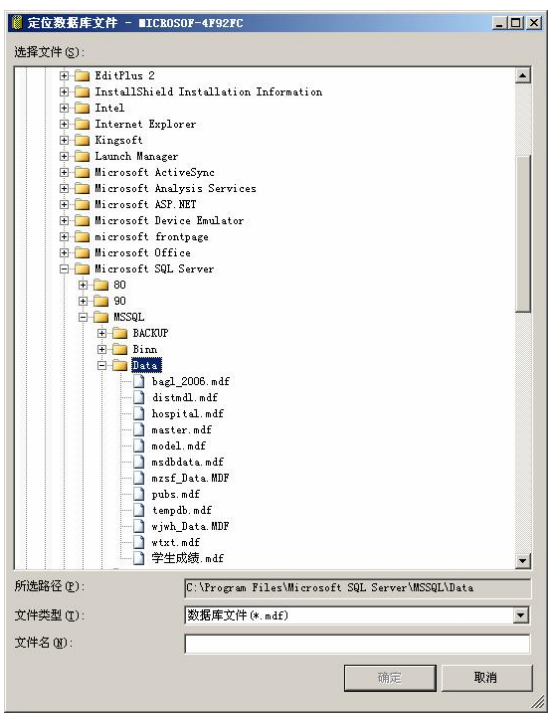


图 11.17 “定位数据库文件”对话框

（4）选择要附加的数据库的 MDF 文件，本例选择“pubs.mdf”，然后单击“确定”按钮，返回“附加数据库”对话框。

（5）单击“附加数据库”对话框中的“确定”按钮，开始执行附加数据库操作。

（6）附加完成后，展开“数据库”树型目录，可以查看刚才附加的数据库，如图 11.18 所示。

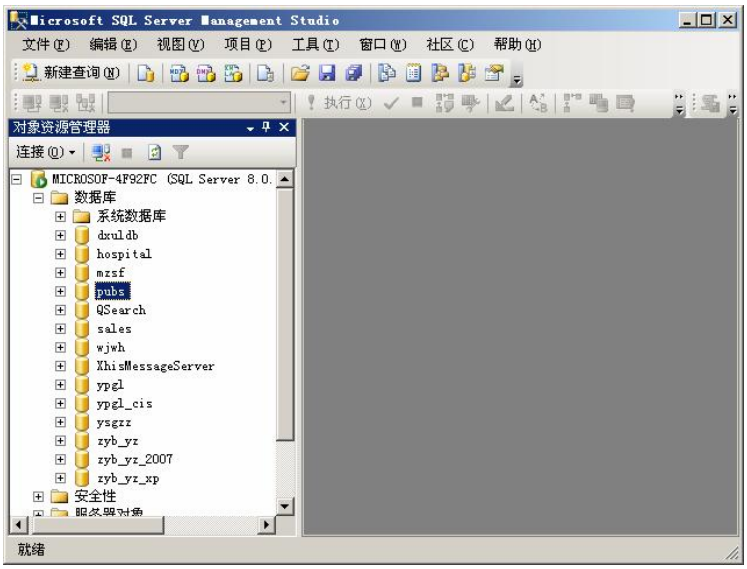


图 11.18　“数据库”树型目录

11.6　数据导入与导出

SQL Server 2005 提供了一个数据导入与导出工具，这是一个向导程序，用于在不同的 SQL Server 服务器之间，以及 SQL Server 与其他类型的数据库（如 Access、Foxpro 等）或数据文件（如文本文件等）之间进行数据交换。下面将介绍如何利用数据导入与导出工具实现 SQL Server 2005 与 Access 数据库的数据交换。

11.6.1　数据导出

将一个 SQL Server 数据库中的数据导入一个 Access 数据库时，后者必须是一个已经存在的数据库。在执行导入操作之前，首先在 Access 中建立一个文件名为“学生成绩.mdb”的空白数据库，以便接收来自 SQL Server 数据库中的数据。

导出数据的操作步骤如下。

（1）启动“SQL Server Management Studio”，在“对象资源管理器”窗口里展开“数据库”树型目录，鼠标右键单击“xscj”数据库，在弹出的快捷菜单里选择“任务”，如图 11.19 所示。

（2）单击“导出数据”菜单项，弹出如图 11.20 所示的“SQL Server 导入和导出向导”欢迎界面。

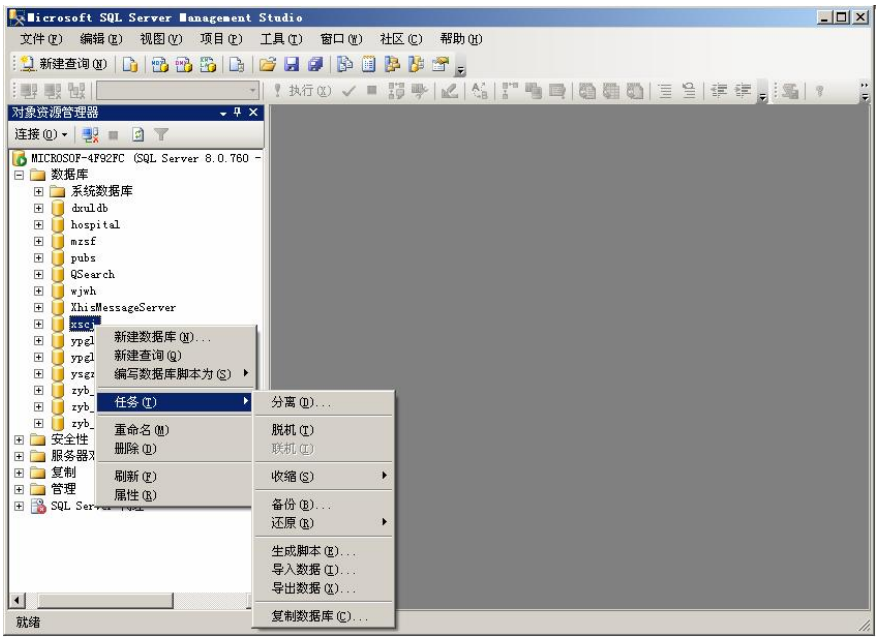


图 11.19　鼠标右键单击“xscj”数据库，选择“任务”

图 11.20　“SQL Server 导入和导出向导”欢迎界面

（3）单击图 11.20 中的“下一步”按钮，弹出如图 11.21 所示的“选择数据源”对话框。

在“数据源”下拉列表框中选择“Microsoft OLE DB Provider for SQL Server”。

在“服务器”框中选择或输入服务器的名称。

“服务器的登录方式”可以选择使用 Windows 身份验证模式，也可以选择使用 SQL Server 身份验证模式。如果选择了后一种方式，还需要在“用户名”文本框中输入登录时使用的用户账户名称，然后在“密码”文本框中输入登录密码。

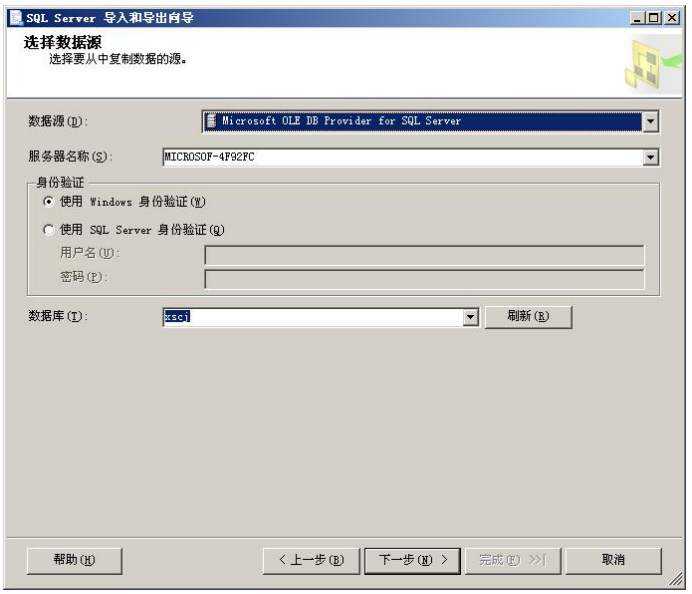


图 11.21　“选择数据源”对话框

（4）单击图 11.21 所示的“下一步”按钮，弹出如图 11.22 所示的“选择目标”对话框。在“目的”下拉列表框中选择目的数据库的格式。

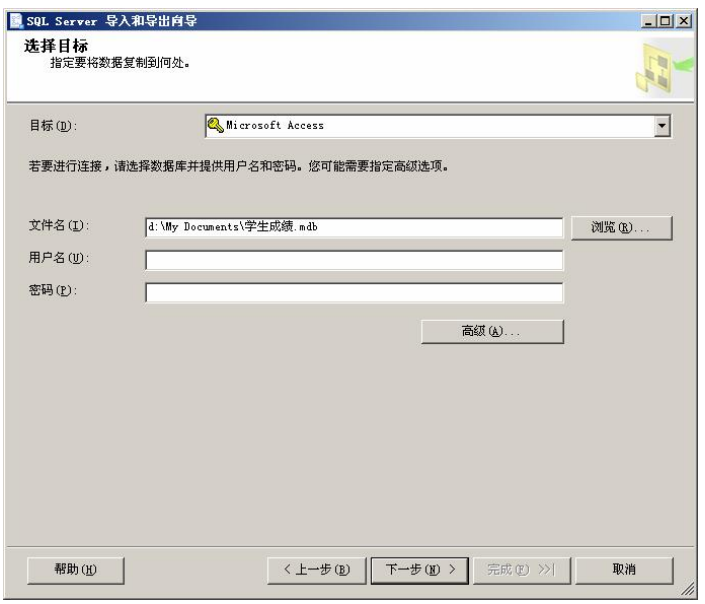


图 11.22　“选择目标”对话框

在“文件名”文本框中输入目的数据库的文件名和路径（这个文件必须已经存在），也可以单击文本框右边的“浏览”按钮，然后从磁盘上选择一个 Access 数据库，使其文件名和路径出现在此文本框中。在本例中，所选样的 Access 数据库文件名为“学生成绩.mdb”。

如果需要登录到目标数据库，那么分别在“用户名”和“密码”文本框中输入登录用户名和密码。

如果需要对目标数据库 OLE DB 驱动程序的进程选项进行设置，单击“高级”按钮，然后

在“高级连接属性”对话框中设置有关选项。

（5）单击图 11.22 所示的“下一步”按钮，弹出如图 11.23 所示的“指定表复制或查询”对话框。

图 11.23 “指定表复制或查询”对话框

选择整个表或部分数据进行复制。若要把整个源表全部复制到目标数据库中，选择“从源数据库复制表和视图”选项；若只想使用一个查询将指定数据复制到目标数据库中，选择“用一条查询指定要传输的数据”选项。

（6）单击图 11.23 所示的“下一步”按钮，弹出如图 11.24 所示的“选择源表和源视图”对话框。

图 11.24 “选择源表和源视图”对话框

选择源表。在图 11.24 中列出了源数据库中包含的表，可以从中选择一个或多个表作为源表，为此在“源”列中选定相应的复选框即可。选择一个源表以后，就会在“目标”列中显示出目标表的名称，默认与源表名称相同，也可以更改为其他名称。

（7）单击图 11.24 所示的“下一步”按钮，弹出如图 11.25 所示的“保存并执行包”对话框，选中“立即执行”复选框。

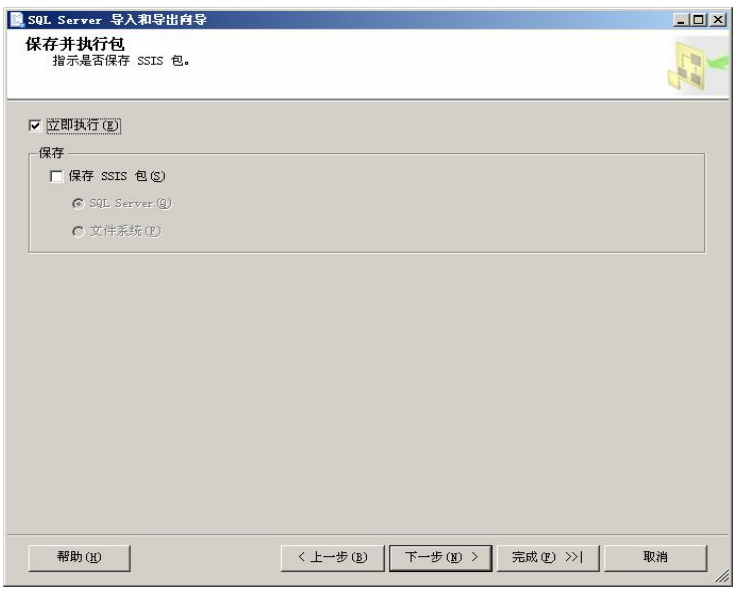


图 11.25　“保存并执行包”对话框

（8）单击图 11.25 所示的“下一步”按钮，弹出如图 11.26 所示的“完成该向导”对话框。

图 11.26　“完成该向导”对话框

（9）单击图 11.26 所示的“完成”按钮，开始执行数据导出操作。

通过以上操作，SQL Server 数据库中的源表被导入 Access 目标数据库中。在 Access 打开目标数据库，便可以查看这些表，如图 11.27 所示。

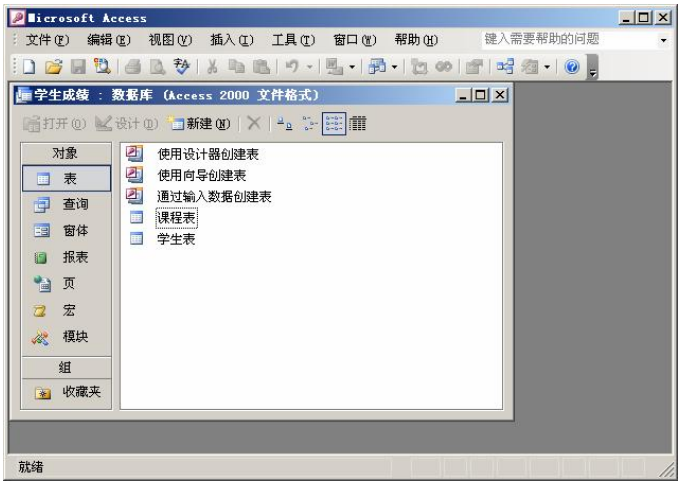


图 11.27　在 Access 中查看导入目标数据库中的表

11.6.2　数据导入

为了说明导入数据的操作，首先在 SQL Server 2005 中创建一个名为“Sales”的数据库，然后将 Access 数据库“商品销售管理”中的数据导入“Sales”数据库中。

1．导入数据的操作步骤

（1）启动“SQL Server Management Studio”，在“对象资源管理器”窗口里展开“数据库”树型目录，鼠标右键单击“Sales”数据库，选中“任务”菜单项，在弹出的快捷菜单里单击“导入数据”，弹出如图 11.20 所示的“SQL Server 导入和导出向导”欢迎界面。

（2）单击图 11.20 中的“下一步”按钮，弹出如图 11.28 所示的“选择数据源”对话框。

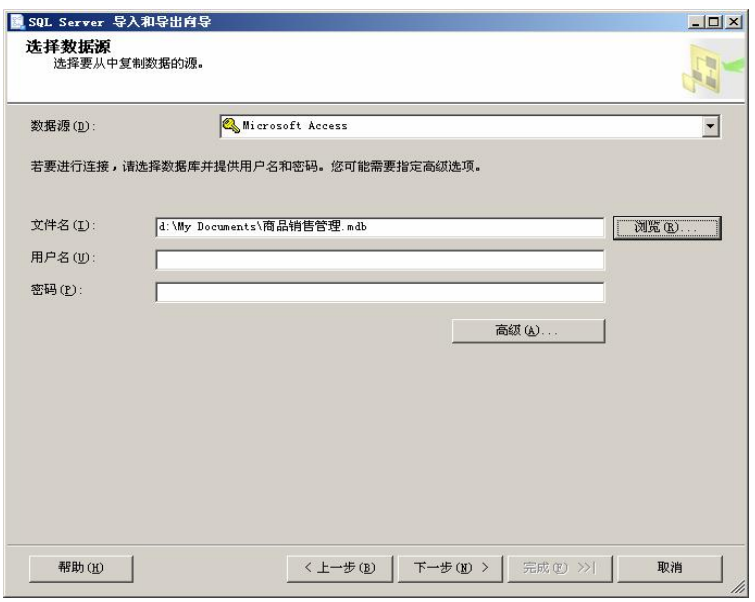


图 11.28　“选择数据源”对话框

在“数据源”下拉列表框中选择“Microsoft Access”。

在“文件名”文本框中输入源数据库的文件名和路径，本例的文件名和路径“d:\My Documents\商品销售管理.mdb”。

如果要登录到源数据库，分别在“用户名”和“密码”文本框中输入登录用户名和所用密码。

（3）单击图 11.28 所示的“下一步”按钮，弹出如图 11.29 所示的“选择目标”对话框。

图 11.29　“选择目标”对话框

在“目标”下拉列表框中选择“Microsoft OLE DB Provider for SQL Server”。

在“服务器”框中选择或输入服务器的名称。

“服务器的登录方式”可以选择使用 Windows 身份验证模式，也可以选择使用 SQL Server 身份验证模式。如果选择了后一种方式，还需要在“用户名”文本框中输入登录时使用的用户登录名，然后在“密码”文本框中输入登录密码。

（4）单击图 11.29 所示的“下一步”按钮，弹出如图 11.30 所示的“指定表复制或查询”对话框。

图 11.30　“指定表复制或查询”对话框

选择整个表或部分数据进行复制。若要把整个源表全部复制到目标数据库中，选择“从源数据库复制表和视图”选项；若只想使用一个查询将指定数据复制到目标数据库中，选择“用一条查询指定要传输的数据”选项。

（5）单击图 11.30 中的“下一步”按钮，弹出如图 11.31 所示的“选择源表和源视图”对话框。

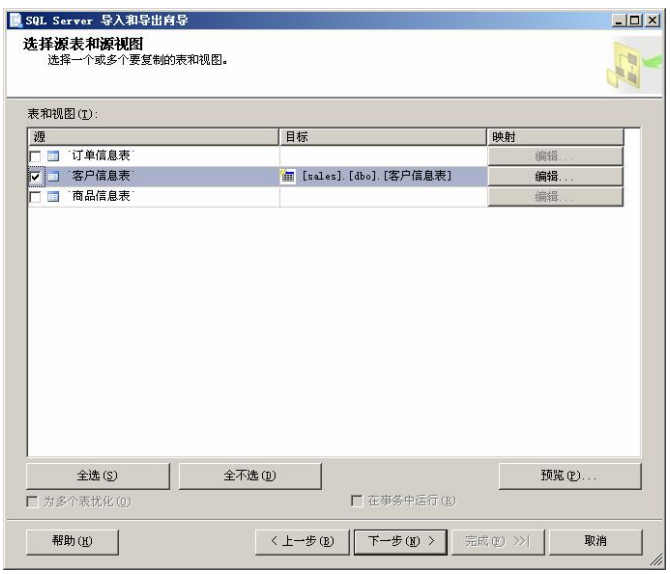


图 11.31 “选择源表和源视图”对话框

选择源表。在图 11.31 中列出了源数据库中所包含的表，可以从中选择一个或多个表作为源表，在“源”列中选定相应的复选框即可。选择一个源表以后，就会在“目标”列中显示出目标表的名称，默认与源表名称相同，但也可以更改为其他名称。

（6）单击图 11.31 中的“下一步”按钮，弹出如图 11.32 所示的“保存并执行包”对话框，选中“立即执行”复选框。

图 11.32　“保存并执行包”对话框

（7）单击图 11.32 所示的“下一步”按钮，弹出如图 11.33 所示的“完成该向导”对话框。

图 11.33　“完成该向导”对话框

（8）单击图 11.33 所示的“完成”按钮，开始执行数据导出操作。

通过以上操作，Access 数据库中的源表被导入 SQL Server 目标数据库中。

本 章 小 结

本章首先介绍了数据库备份基本概念和备份设备，然后讨论了数据库备份和还原，包括备份和还原的类型及方法，并通过实例详细说明，接着介绍了如何在向导的帮助下完成数据库的分离与附加，最后介绍了数据的导入和导出。

习　　题

1．简述 SQL Server 2005 中引起系统故障与数据损失的主要因素。

2．创建一个名为“课程信息”的磁盘备份设备，然后删除该备份设备。

3．SQL Server 2005 有哪些备份数据库的方法？

4．SQL Server 2005有哪些还原数据库备份的方法？

5．数据导入和导出的含义是什么？

本 章 实 训

一、实训目的

1．了解数据库备份的基本概念，理解数据库备份过程。

2．掌握创建、使用和删除备份设备的方法。

3．掌握数据库备份的步骤和方法。

4．掌握数据库还原的步骤和方法。

5．掌握数据库分离与附加的步骤和方法。

6．掌握数据导入和导出的步骤和方法。

二、实训要求

1．实训前做好上机实训的准备，针对实训内容，认真复习与本次实训有关的知识，完成实训内容的预习准备工作。

2．能认真独立完成实训内容。

3．实训后做好实训总结，根据实训情况完成总结报告。

三、实训学时

2学时。

四、实训内容

现有学生成绩数据库“XSCJDB”，包括学生表“XS”（学号、姓名、性别、年龄、系别）、课程表“KC”（课程号、课程名、学分数、学时数）和成绩表“CJ”（学号、课程号、成绩），以下实验均在此数据库上完成。

1．创建备份设备“STU1”、“STU2”和“STU3”。

2．创建学生成绩数据库“XSCJDB”的完全备份，备份到设备“STU1”上。

3．创建教师表“JS”（工号，姓名，职称，所属系别），然后对学生成绩数据库“XSCJDB”进行差异备份，备份到设备“STU2”上。

4．在教师表“JS”中插入若干记录，然后对学生成绩数据库“XSCJDB”进行事务日志备份，备份到设备“STU3”上。

5．删除学生成绩数据库“XSCJDB”，然后利用备份设备“STU1”、“STU2”和“STU3”还原学生成绩数据库“XSCJDB”上。

6．删除备份设备“STU3”。

7．对学生成绩数据库“XSCJDB”进行分离和附加操作。

8．把学生表“XS”中的数据导出到文本文件“d:\xs.txt”中。

9．删除学生表“XS”中的数据，然后把“d:\xs.txt”中的数据导入到学生表“XS”中。

五、实训思考题

1．备份分为哪几种类型？

2．确定备份计划应考虑哪些因素？

第 12 章

SQL Server 2005 数据库的安全性和完整性管理

对企业或组织来说，数据的泄漏或篡改将导致公司利益的损失。数据库的安全性问题因此而提出。数据库中的数据是从外界输入的，而数据的输入由于种种原因会发生输入无效或错误信息。因此数据库的安全性和完整性是数据库管理员首要考虑的问题。

12.1 数据库安全性概述

SQL Server 的安全性管理是建立在认证（authentication）和访问许可（permission）这两种机制上的。认证是指确定登录 SQL Server 的用户的登录账号和密码是否正确，以此来验证其是否具有连接 SQL Server 的权限。但是，通过认证并不代表能够访问 SQL Server 中的数据。用户只有在获取访问数据库的权限之后，才能够对服务器上的数据库进行权限许可下的各种操作。用户访问数据库权限的设置是通过用户账号来实现的。角色简化了安全性管理。所以在 SQL Server 的安全模型中包括以下几部分：

- SQL Server 身份验证；
- 登录账户；
- 数据库用户；
- 角色；
- 权限。

12.2 SQL Server 2005 身份验证

12.2.1 身份验证简介

SQL Server 支持两种模式的身份验证：Windows 验证模式、SQL Server 和 Windows 混合验证模式。

Windows 验证模式比起 SQL Server 验证模式来有许多优点。Windows 身份验证比 SQL Server 身份验证更加安全；使用 Windows 身份验证的登录账户更易于管理；用户只需登录 Windows 之后就可以使用 SQL Server，只需要登录一次。

在混合验证模式下，Windows 验证和 SQL Server 验证这两种验证模式都是可用的。对于 SQL Server 验证模式，用户在连接 SQL Server 时必须提供登录名和登录密码。

12.2.2 验证模式的修改

当安装 SQL Server 时，可以选择 SQL Server 的身份验证类型。安装完成之后也可以修改认证模式。修改步骤如下。

（1）打开 SQL Server Management Studio。

（2）在要更改的服务器上鼠标右键单击，在快捷菜单中选择属性，弹出服务器属性对话框。

（3）单击左侧列表中的“安全性”项，出现“安全性”页面，如图 12.1 所示。在图中修改身份验证。

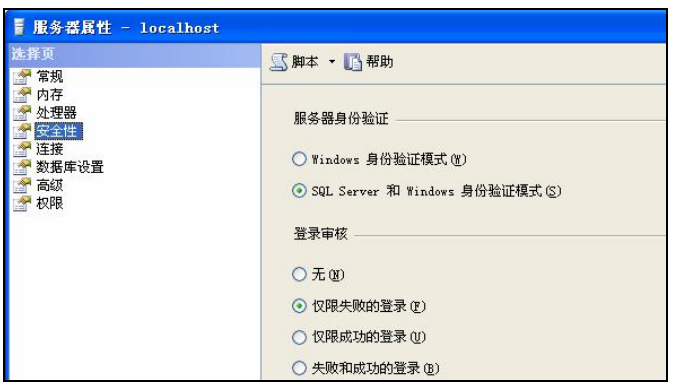


图 12.1　身份验证

12.3 SQL Server 2005 登录账户管理

12.3.1 使用 Management Studio 管理登录账户

1. 创建 Windows 登录账户

（1）在“对象资源管理器”中，单击树型目录中的“安全性”节点，如图 12.2 所示。

图 12.2　服务器安全性

（2）鼠标右键单击“安全性”的子节点“登录名”，在快捷菜单中选择“新建登录名…”，出现“登录名-新建”对话框，如图 12.3 所示。

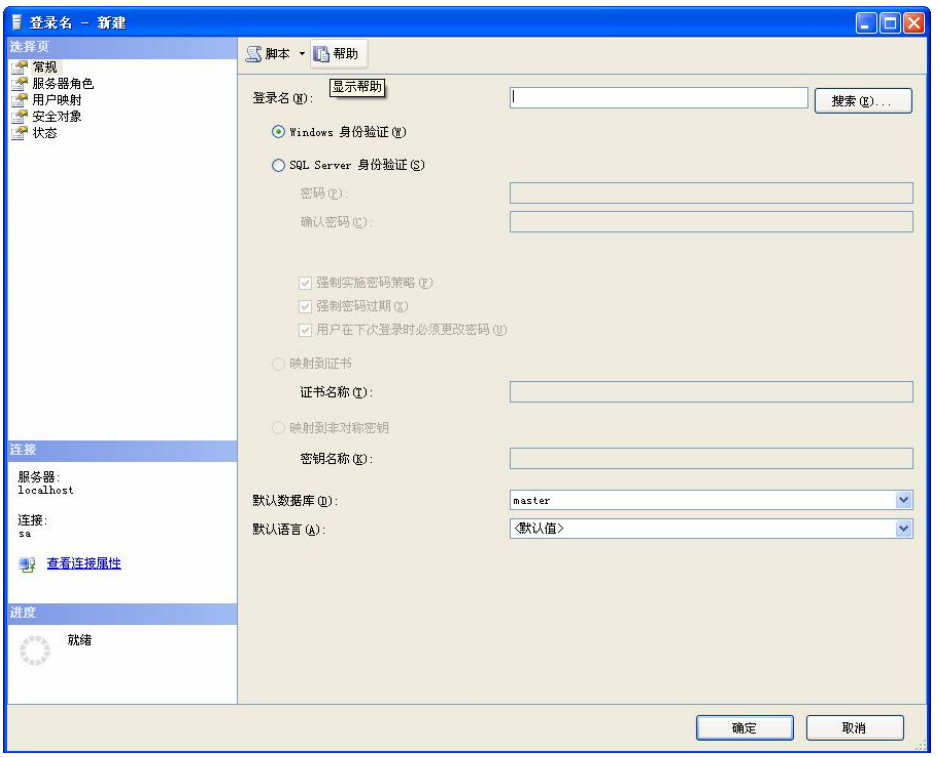


图 12.3　“登录名—新建”对话框

（3）在“登录名”编辑框中输入登录名称，输入的登录名必须是已存在的 Windows 登录用户。可以单击“搜索…”按钮，出现登录“选择用户和组”对话框，如图 12.4 所示。在对象名称编辑框中输入用户或组的名称，单击“检查名称”按钮检查对象是否存在。输入完成，单击“确定”按钮关闭选择用户或组对话框。

图 12.4　“选择用户和组”对话框

（4）确认选择的是“Windows 身份验证”。指定账户登录的默认数据库。

（5）单击窗口左侧列表中的“服务器角色”节点，指定账户所属服务器角色。

（6）单击窗口左侧列表中的“用户映射”节点，右侧出现用户映射页面。可以查看或修改 SQL 登录账户到数据库用户的映射。选择此登录账户可以访问的数据库，对具体的数据库，指定要映射到登录名的数据库用户（默认情况下，数据库用户名与登录名相同）。指定用户的默认架构，首次创建用户时，其默认架构是 dbo。

（7）设置完成单击“确定”按钮提交更改。

2. 创建 SQL Server 登录账户

一个 SQL Server 登录账户名是一个新的登录账户，该账户和 Windows 操作系统的登录账户没有关系。

（1）打开新建登录名对话框，选择“SQL Server 身份验证”，输入登录名，密码和确认密码，并选择缺省数据库，如图 12.5 所示。

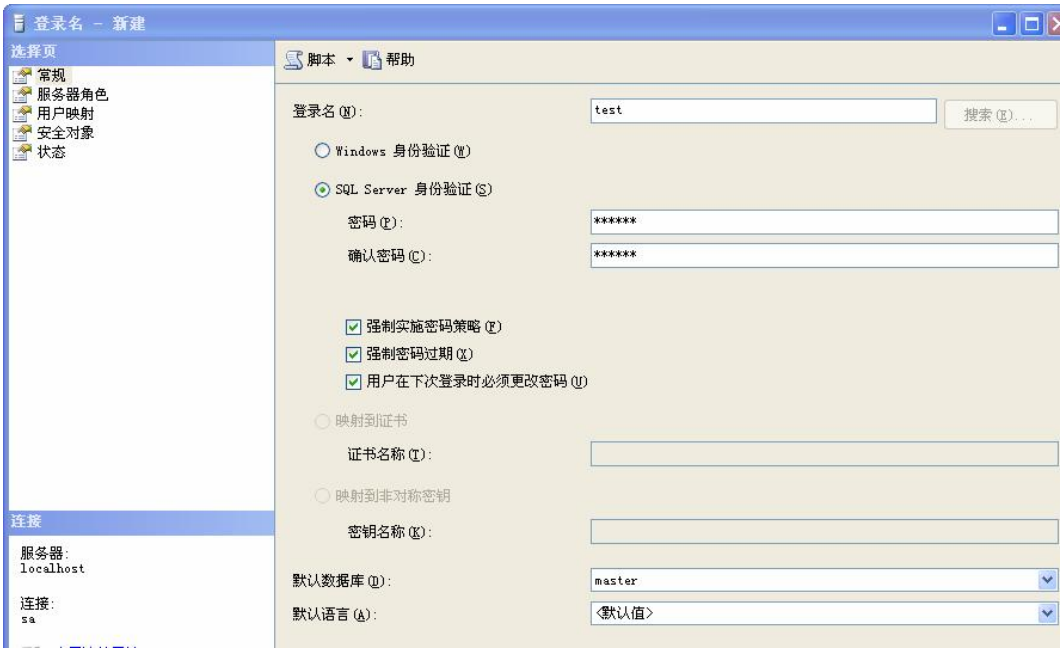


图 12.5　SQL Server 身份验证

（2）设置服务器角色和用户映射，请参考“创建 Windows 登录账户”的步骤 5 和步骤 6。

3. 登录账户管理

创建登录账户之后，在图 12.2 所示服务器安全性展开登录名节点上，鼠标右键单击相应的账户，出现快捷菜单，如图 12.6 所示，如果要修改该登录账户，选择属性菜单；如要删除该登录账户，则选择删除菜单。

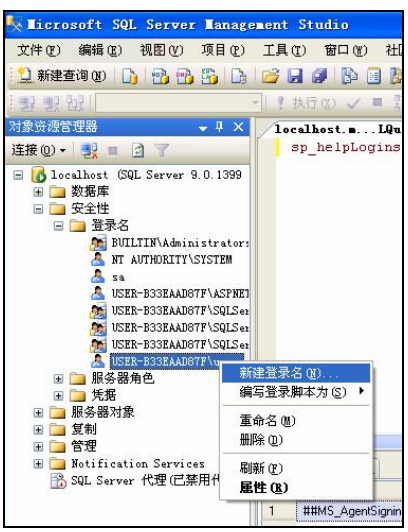


图 12.6　管理登录账户

12.3.2　使用 Transact-SQL 管理登录账户

在 Transact-SQL 中，管理登录账户的 SQL 语句有：CREATE LOGIN、DROP LOGIN、

ALTER LOGIN。下面简要说明如何使用 T-SQL 来创建和维护登录账户。

1. 新建登录账户 CREATE LOGIN

其语法格式为：

```
CREATE LOGIN login_name { WITH <option_list1> | FROM <sources> }
```

【例 12.1】创建带密码的登录名“test”。MUST_CHANGE 选项要求用户首次连接服务器时更改此密码。

```
CREATE LOGIN test WITH PASSWORD = '2<6aK'    MUST_CHANGE
```

【例 12.2】从 Windows 域账户创建 [Development\iewangjf] 登录名。

```
CREATE LOGIN [Development\iewangjf] FROM Windows
```

2. 删除登录账户 DROP LOGIN

其语法格式为：DROP LOGIN login_name

【例 12.3】删除登录账户“test”。

```
DROP LOGIN test
```

3. 更改登录账户 ALTER LOGIN

其语法格式为：

```
ALTER LOGIN login_name
    {
    <status_option>   | WITH <set_option> [ ,... ]
    } <status_option> ::= ENABLE | DISABLE
```

【例 12.4】启用禁用的登录。

```
ALTER LOGIN test ENABLE;
```

【例 12.5】将“test”登录密码更改为 <123www456>。

```
ALTER LOGIN test WITH PASSWORD = '<123www456>'
```

【例 12.6】将登录名“test”更改为“iewangjf”。

```
ALTER LOGIN test WITH NAME = iewangjf
```

12.4　SQL Server 2005 数据库用户

用户对数据的访问权限以及对数据库对象的所有关系都是通过用户账号来控制的，用户账号总是基于数据库的，即两个不同数据库中可以有两个相同的用户账号。

在数据库中，用户账号与登录账号是两个不同的概念，一个合法的登录账号只表明该账号通过了 Windows 认证或 SQL Server 认证，但不能表明其可以对数据库数据和数据对象进行某种操作。

通常而言，数据库用户账号总是与某一登录账号相关联。但有一个例外，那就是 guest 用户。用户通过 Windows 认证或 SQL Server 认证而成功登录到 SQL Server 之后的过程如下。

（1）检查该登录用户是否有合法的用户名，如果有合法用户名，则允许其以用户名访问数据库，否则执行步骤 2。

（2）SQL Server 检查是否有 guest 用户，如果有，则允许登录用户以 guest 用户来访问数

据库，如果没有，则该登录用户被拒绝。

由此可见，guest 用户主要是作为那些没有属于自己的用户账号的 SQL Server 登录者的缺省用户名，从而使该登录者能够访问具有 guest 用户的数据库。

12.4.1 使用 Management Studio 管理用户

（1）在 Management Studio 对象资源管理器中，扩展指定的数据库节点，直到看到用户节点，如图 12.7 所示。

（2）鼠标右键单击用户子节点，在弹出菜单中选择“新建用户…”，弹出“新建数据库用户”对话框，如图 12.8 所示。在用户名编辑框中输入用户名。

（3）在登录名编辑框中输入登录名或单击 “…”按钮，弹出“选择登录名”对话框，如图 12.9 所示。输入登录名或单击“浏览”按钮。如单击“浏览”按钮，则出现“查找对象”对话框，如图 12.10 所示。

（4）选中想添加的登录名，单击“确定”按钮关闭对话框。

（5）选择该用户登录的默认架构和所属角色，最后关闭“新建数据库用户”对话框。

图 12.7 数据库用户

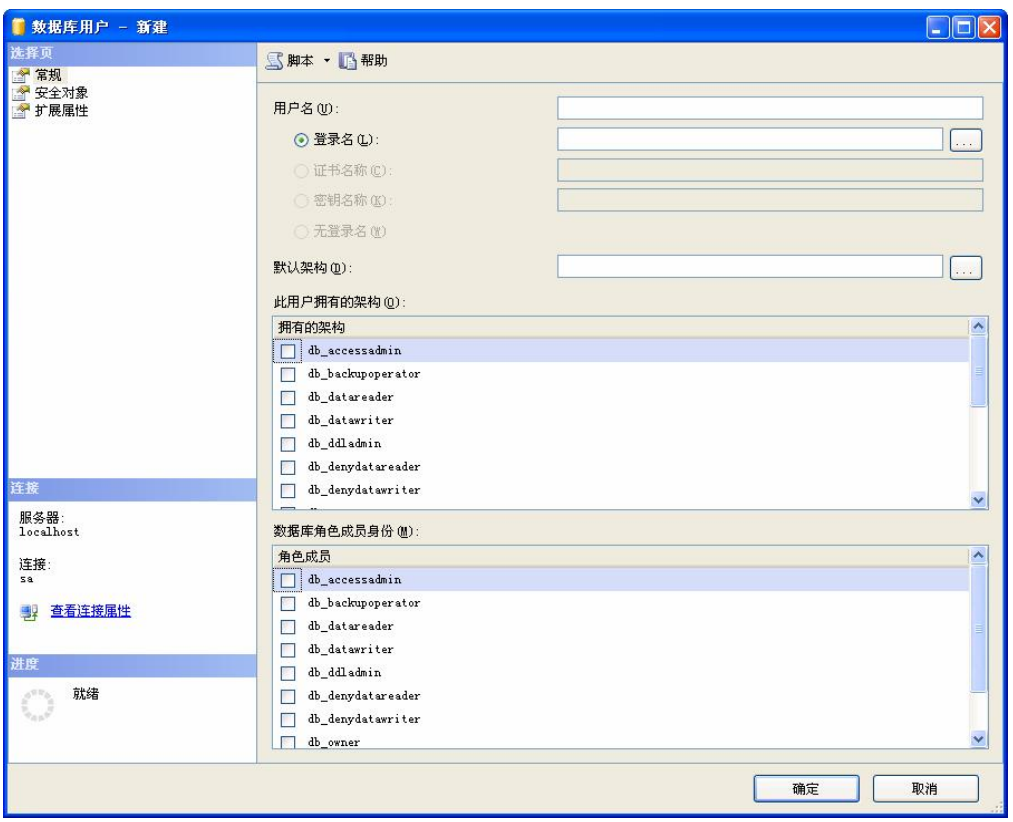


图 12.8 “新建数据库用户”对话框

图 12.9　“选择登录名”对话框

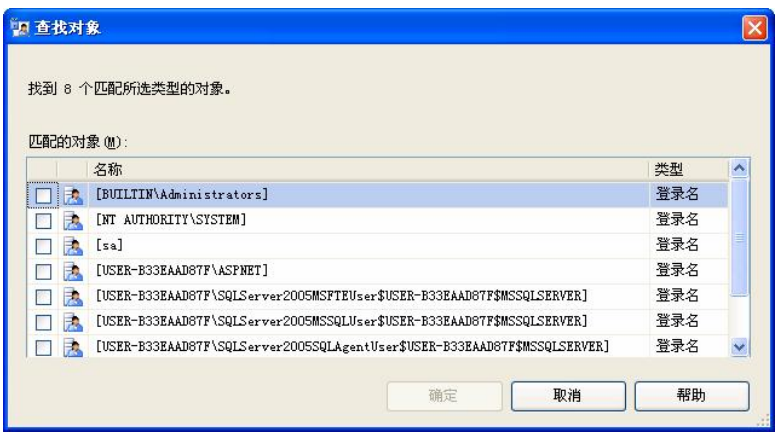


图 12.10　“查找对象”对话框

12.4.2　使用 Transact-SQL 管理用户

使用 Transact-SQL 管理用户的语句有 CREATE USER，DROP USER，ALTER USER。

1. 创建用户：CREATE USER

CREATE USER 语法格式为：

```
CREATE USER user_name        [ { { FOR | FROM }
{
      LOGIN login_name
      | CERTIFICATE cert_name
      | ASYMMETRIC KEY asym_key_name
} | WITHOUT LOGIN
] [ WITH DEFAULT_SCHEMA = schema_name ]
```

各参数简要说明如下。

- user_name：指定在此数据库中用于识别该用户的名称。
- LOGIN login_name：指定要创建数据库用户的 SQL Server 登录名。login_name 必须

是服务器中有效的登录名。如果忽略 FOR LOGIN，则新的数据库用户将映射到同名的 SQL Server 登录名。

- 如果未定义 DEFAULT_SCHEMA，则数据库用户将使用 dbo 作为默认架构。

【例 12.7】首先创建名为“Teacher”且具有密码的服务器登录名，然后在数据库“TEACH”中创建对应的数据库用户“WangWei”。

```
CREATE LOGIN Teacher WITH PASSWORD = '270<tea>39';
USE TEACH
CREATE USER WangWei
```

【例 12.8】创建具有默认架构“Teaching”的对应数据库用户“WangWei”

```
CREATE USER WangWei FOR LOGIN Teacher
    WITH DEFAULT_SCHEMA = Teaching
```

2. 更改用户名或更改其登录的默认架构 ALTER USER

其语法格式为：

```
ALTER USER user_name   WITH <set_item> [ ,...n ]
<set_item> ::=   NAME = new_user_name
          | DEFAULT_SCHEMA = schema_name
```

【例 12.9】更改数据库用户的名称。

```
ALTER USER WangWei WITH NAME = Wangjf
```

【例 12.10】更改用户的默认架构。

```
ALTER USER WangWei WITH DEFAULT_SCHEMA = Admining
```

3. 删除用户：DROP USER

其语法格式为：

```
DROP USER user_name
```

【例 12.11】删除用户“WangWei”。

```
DROP USER WangWei
```

12.5　SQL Server 2005 角色

角色等价于 Windows 的工作组，将登录名或用户赋予一个角色，角色具有权限，登录名或用户作为角色成员，继承了所属角色的权限。如图 12.11 所示。

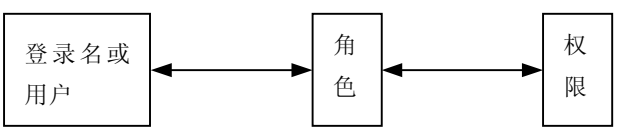


图 12.11　基于角色的访问控制

只需给角色指定权限，然后将登录名或用户指定为某个角色，而不必给每个登录名或用户指定权限，这样给实际工作带来了很大的便利。

在 SQL Server 中角色分为服务器角色和数据库角色。而数据库角色又分为固有数据库角

色、用户自定义数据库角色和应用程序角色。

12.5.1　角色管理简介

1. 服务器角色

服务器角色内建于 SQL Server，其权限无法更改，每一个角色拥有一定级别的数据库管理职能，如图 12.12 所示。

服务器角色包括以下几种。

- bulkadmin：可以运行 BULK INSERT 语句。
- dbcreator：可以创建、更改、删除和还原任何数据库。
- diskadmin：管理磁盘文件。
- processadmin：可以终止 SQL Server 实例中运行的进程。
- securityadmin：管理登录名及其属性。这类角色可以 GRANT、DENY 和 REVOKE 服务器级和数据库级权限，可以重置 SQL Server 登录名的密码。
- serveradmin：可以更改服务器范围的配置选项和关闭服务器。
- setupadmin：添加和删除链接服务器，并且也可以执行某些系统存储过程。
- sysadmin：可以在服务器中执行任何活动。

2. 固有数据库角色

固有数据库角色是指这些角色的数据库权限已被 SQL Server 预定义，不能对其权限进行任何修改，并且这些角色存在于每个数据库中，如图 12.13 所示。

图 12.12　服务器角色

图 12.13　固有数据库角色

固有数据库角色包括以下几种。

- db_accessadmin：可以为 Windows 登录账户、Windows 组和 SQL Server 登录账户添加或删除访问权限。

- db_backupoperator：可以备份该数据库。
- db_datareader：可以读取所有用户表中的所有数据。
- db_datawriter：可以在所有用户表中添加、删除或更改数据。
- db_ddladmin：可以在数据库中运行任何数据定义语言（DDL）命令。
- db_denydatareader：不能读取数据库内用户表中的任何数据。
- db_denydatawriter：不能添加、修改或删除数据库内用户表中的任何数据。
- db_owner：可以执行数据库的所有配置和维护活动。
- db_securityadmin：可以修改角色成员身份和管理权限。
- public：当添加一个数据库用户时，它自动成为该角色成员，该角色不能删除，指定给该角色的权限自动给予所有数据库用户。

db_owner 和 db_securityadmin 角色的成员可以管理固有数据库角色成员身份；但是，只有 db_owner 数据库的成员可以向 db_owner 固有数据库角色中添加成员。

3. 用户自定义数据库角色

当打算为某些数据库用户设置相同的权限，但是这些权限不等同于预定义的数据库角色所具有的权限时，就可以定义新的数据库角色来满足这一要求，从而使这些用户能够在数据库中实现某些特定功能。用户自定义数据库角色包含以下两种类型。

- 标准角色：为完成某项任务而指定的具有某些权限和数据库用户的角色。
- 应用角色：与标准角色不同的是，应用角色默认情况下不包含任何成员，而且是非活动的。将权限赋予应用角色，然后将逻辑加入到某一特定的应用程序中，从而激活应用角色而实现了对应用程序存取数据的可控性。

12.5.2 角色的管理

1. 使用 Management Studio 管理角色

（1）为服务器角色添加登录账户。执行如下步骤。

① 在图 12.12 中展开服务器角色节点。在需要添加用户的角色上单击鼠标右键，选择“属性”菜单，弹出“属性”对话框。

② 单击“添加”按钮，则弹出“选择登录名”对话框，如图 12.9 所示。

③ 单击“浏览”按钮，则弹出“查找对象”对话框，如图 12.10 所示。

④ 选中需要添加的对象。

⑤ 单击每个对话框中的“确定”按钮关闭对话框。

（2）为固有数据库角色添加成员。鼠标右键单击想要添加成员的固有服务器角色节点，其余步骤与（1）类似。

（3）创建用户自定义角色。执行如下步骤。

① 展开要创建数据库节点，直到看到“数据库角色”节点，鼠标右键单击“数据库角色”，选择“新建数据库角色”，出现“新建数据库角色”对话框，如图 12.14 所示。

② 在角色名称编辑框中填入角色名称，在所有者编辑框中填入该角色的所有者。

③ 指定角色拥有的框架名称。单击“添加”按钮添加角色成员，则弹出“选择数据库用户或角色”对话框，如图 12.15 所示。

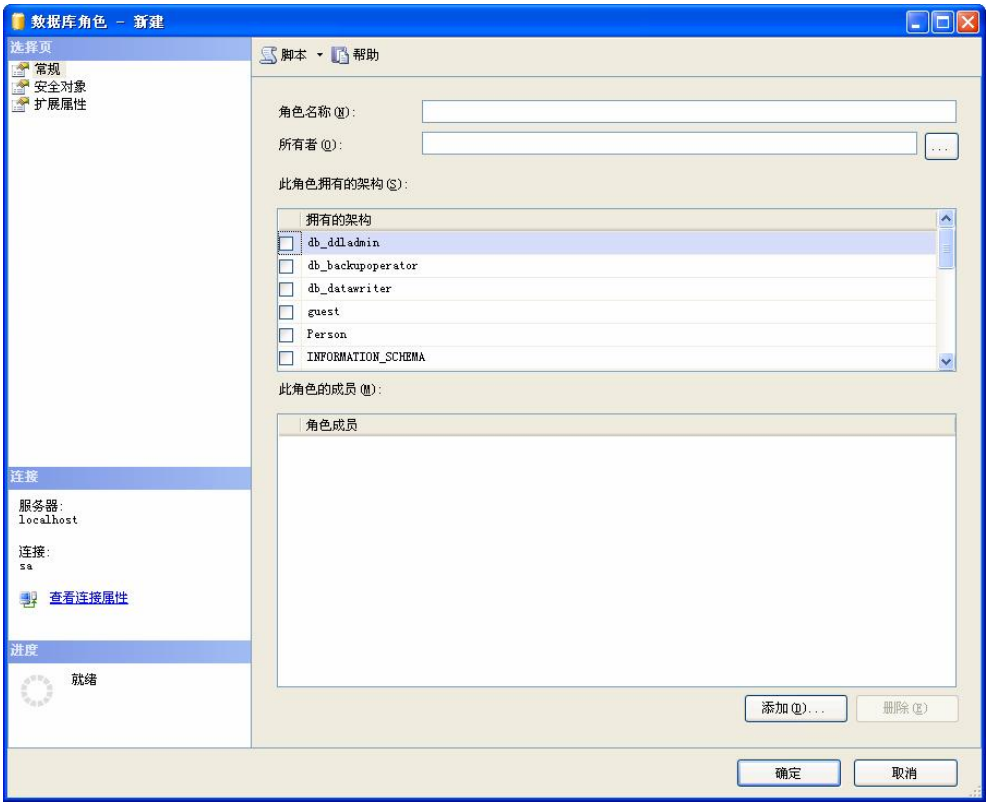


图 12.14　“新建数据库角色”对话框

图 12.15　“选择数据库用户或角色”对话框

④ 输入用户（如果需要，单击“浏览”按钮），单击“确定”按钮添加用户到角色。

⑤ 单击图 12.14 左侧选择页中的“安全对象”，则右侧“安全对象”页面，如图 12.16 所示。在此可以设置角色访问数据库的资源。

⑥ 单击安全对象中“添加”按钮，弹出“添加对象”对话框，如图 12.17 所示。

⑦ 选择对象类型，如选择“特定类型的所有对象”，则弹出“选择对象类型”对话框，如图 12.18 所示。

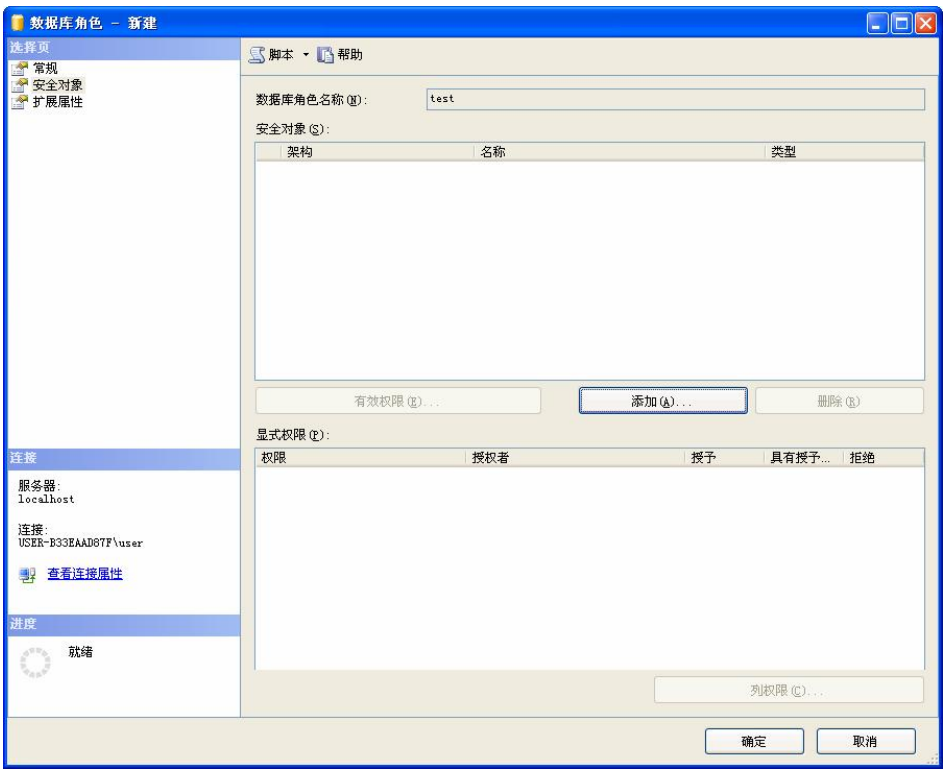


图 12.16 “安全对象”页面

图 12.17 “添加对象”对话框

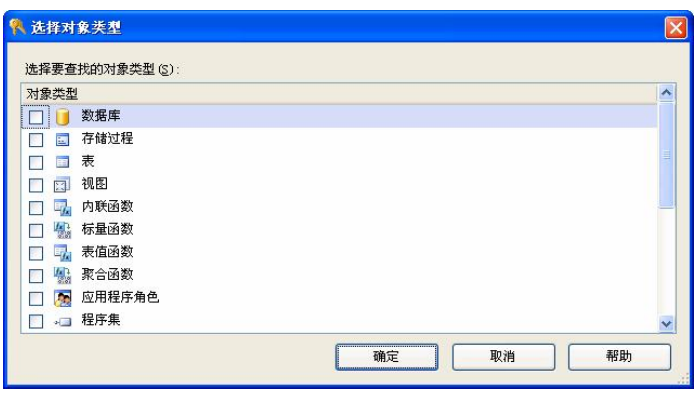


图 12.18 “选择对象类型”对话框

⑧ 在如图 12.18 所示的对话框中选择需要设置权限的对象类型，如选择表，单击“确定”按钮关闭，则显示所有表的权限设置，如图 12.19 所示。可以设置具体的表的权限。针对具体表，还可以设计对应的列权限。

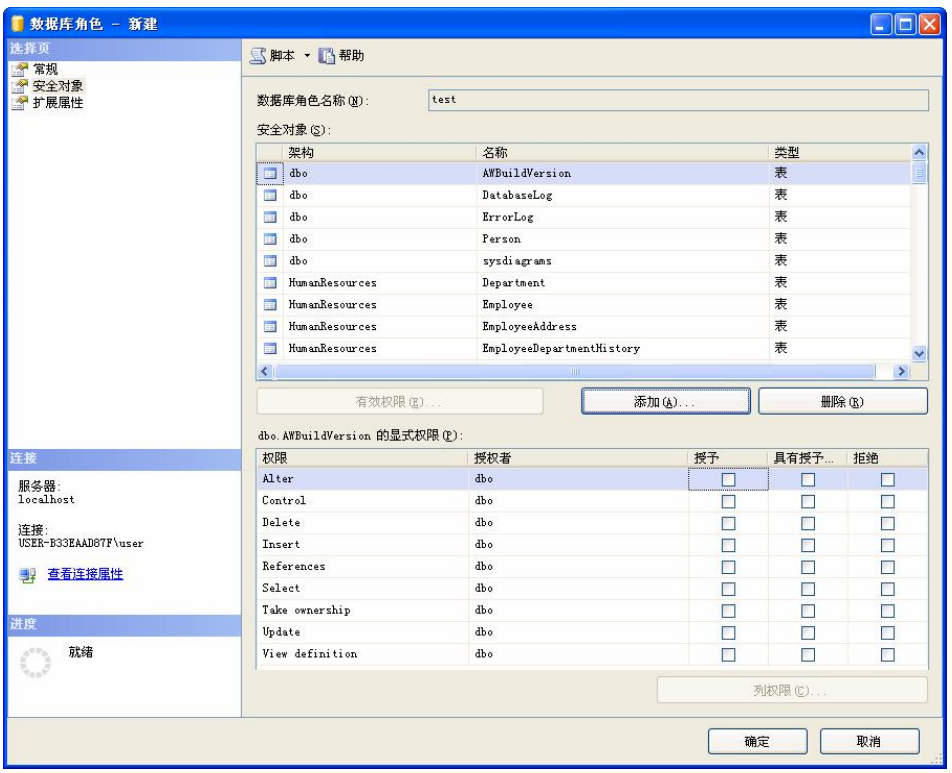


图 12.19　安全对象权限设置

⑨ 单击“确定”按钮完成角色的添加。

2. 使用 Transact-SQL 语句管理角色

对于服务器角色来说，其成员为登录账号，对于数据库角色来说，其成员为数据库用户、数据库角色、Windows 登录或 Windows 组。

（1）管理服务器角色。在 SQL Server 中管理服务器角色的存储过程主要有两个：sp_addsrvrolemember 和 sp_dropsrvrrolemember。

sp_addsrvrolemember 是添加登录账户到服务器角色内，使其成为该角色的成员。

其语法格式为：

```
sp_addsrvrolemember [@loginame =] 'login'  [@rolename =] 'role'
```

sp_dropsrvrrolemember 用来在某一服务器角色中删除登录账号，当该成员从服务器角色中被删除后，便不再具有该服务器角色所设置的权限。

其语法格式为：

```
sp_dropsrvrolemember [@loginame =] 'login'  [@rolename =] 'role'
```

【例 12.12】将登录账户“iewangjf”加入 sysadmin 角色中。

```
sp_addsrvrolemember  ‘iewangjf’  ‘sysadmin’
```

（2）管理数据库角色。管理数据库角色的语句有：CREATE ROLE，DROP ROLE，ALTER

ROLE。

CREATE ROLE 用来新建数据库角色，其语法格式为：

```
CREATE ROLE role_name [ AUTHORIZATION owner_name ]
```

其中 AUTHORIZATION owner_name 表示将拥有新角色的数据库用户或角色。如果未指定用户，则执行 CREATE ROLE 的用户将拥有该角色。

【例 12.13】创建用户“iewangjf”隶属的数据库角色“buyers”。

```
CREATE ROLE buyers AUTHORIZATION iewangjf
```

【例 12.14】创建 db_securityadmin 固有数据库角色隶属的数据库角色“auditors”。

```
CREATE ROLE auditors AUTHORIZATION db_securityadmin
```

管理角色成员的存储过程有：sp_addrolemember，sp_droprolemember，这两个存储过程和添加删除服务器角色的存储过程用法类似。

【例 12.15】将数据库用户“iewangjf”添加到当前数据库的“Sales”数据库角色中。

```
sp_addrolemember 'Sales', 'iewangjf'
```

（3）查看角色信息。查看角色信息的存储过程有 sp_helprolemember、sp_helprole。

sp_helprolemember 返回某个角色的成员的信息。其语法格式为：

```
sp_helprolemember [ [ @rolename = ] 'role' ]
```

sp_helprole 返回当前数据库中有关角色的信息。其语法格式为：

```
sp_helprole [ [ @rolename = ] 'role' ]
```

【例 12.16】显示 Sales 角色的成员。

```
sp_helprolemember 'Sales'
```

【例 12.17】返回当前数据库中的所有角色。

```
sp_helprole
```

12.6 SQL Server 2005 权限

12.6.1 概述

权限管理指将安全对象的权限授予主体，取消或禁止主体对安全对象的权限。SQL Server 通过验证主体是否已获得适当的权限来控制主体对安全对象执行的操作。

1. 主体

“主体”是可以请求 SQL Server 资源的个体、组和过程。主体分类如表 12.1 所示。

表 12.1　　主 体 分 类

主　体	内　容
Windows 级别的主体	Windows 域登录名、Windows 本地登录名
SQL Server 级别的主体	SQL Server 登录名
数据库级别的主体	数据库用户、数据库角色、应用程序角色

2. 安全对象

安全对象是 SQL Server Database Engine 授权系统控制对其进行访问的资源。每个 SQL Server 安全对象都有可能授予主体的关联权限，如表 12.2 所示。

表 12.2　　安全对象内容

安 全 对 象	内　　容
服务器	端点、登录账户、数据库
数据库	用户、角色、应用程序角色、程序集、消息类型、路由、服务、远程服务绑定、全文目录、证书、非对称密钥、对称密钥、约定、架构
架构	类型、XML 架构集合、对象
对象	聚合、约束、函数、过程、队列、统计信息、同义词、表、视图

3. 架构

架构是形成单个命名空间的数据库实体的集合。命名空间是一个集合，其中每个元素的名称都是唯一的。在 SQL Server 2005 中，架构独立于创建它们的数据库用户而存在。可以在不更改架构名称的情况下转让架构的所有权，这是与 SQL Server 2000 不同的地方。

完全限定的对象名称包含 4 部分：server.database.schema.object。

SQL Server 2005 还引入了“默认架构”的概念，用于解析未使用其完全限定名称引用的对象的名称。在 SQL Server 2005 中，每个用户都有一个默认架构，用于指定服务器在解析对象的名称时将要搜索的第一个架构。可以使用 CREATE USER 和 ALTER USER 的 DEFAULT_SCHEMA 选项设置和更改默认架构。如果未定义默认架构，则数据库用户将把 DBO 作为其默认架构。

【例 12.18】下面代码创建了用户“Jane”，默认架构为“Sales”，并设置其拥有数据库“db_ddladmin”角色。

```
USE AdventureWorks
CREATE USER Jane FOR LOGIN Jane
WITH DEFAULT_SCHEMA = Sales;
EXEC sp_addrolemember 'db_ddladmin', 'Jane';
```

这样 Jane 所做的任何操作默认发生在“Sales”架构上，她所创建的对象默认属于“Sales”架构，所引用的对象默认在“Sales”架构上。当她执行以下语句时：

```
CREATE PROCEDURE usp_GetCustomers
AS
SELECT * FROM Customer
```

该存储过程创建在“Sales”架构上，其他用户引用它时需要写为“Sales.usp_GetCustomers”。

4. 权限

在 SQL Server 2005 中，能够授予的安全对象和权限的组合有 181 种！具体的 GRANT、DENY、REVOKE 语句格式和具体的安全对象有关。使用时请参阅联机丛书。

主要安全对象权限如表 12.3 所示。

表 12.3　　主要安全对象权限

安 全 对 象	权　　限
数据库	BACKUP DATABASE、BACKUP LOG、CREATE DATABASE、CREATE DEFAULT、CREATE FUNCTION、CREATE PROCEDURE、CREATE RULE、CREATE TABLE 和 CREATE VIEW
标量函数	EXECUTE 和 REFERENCES
表值函数、表、视图	DELETE、INSERT、REFERENCES、SELECT 和 UPDATE
存储过程	DELETE、EXECUTE、INSERT、SELECT 和 UPDATE

12.6.2 权限的管理

1. 使用 Management Studio 管理权限

可通过对象或主体管理对象权限，下面讲述通过对象来设置权限。

（1）鼠标右键单击对象，在快捷菜单中选择属性，弹出属性对话框。

（2）单击左侧选择页中“权限”，则显示权限页，如图 12.20 所示，在此可以指定该对象的角色或用户的权限。

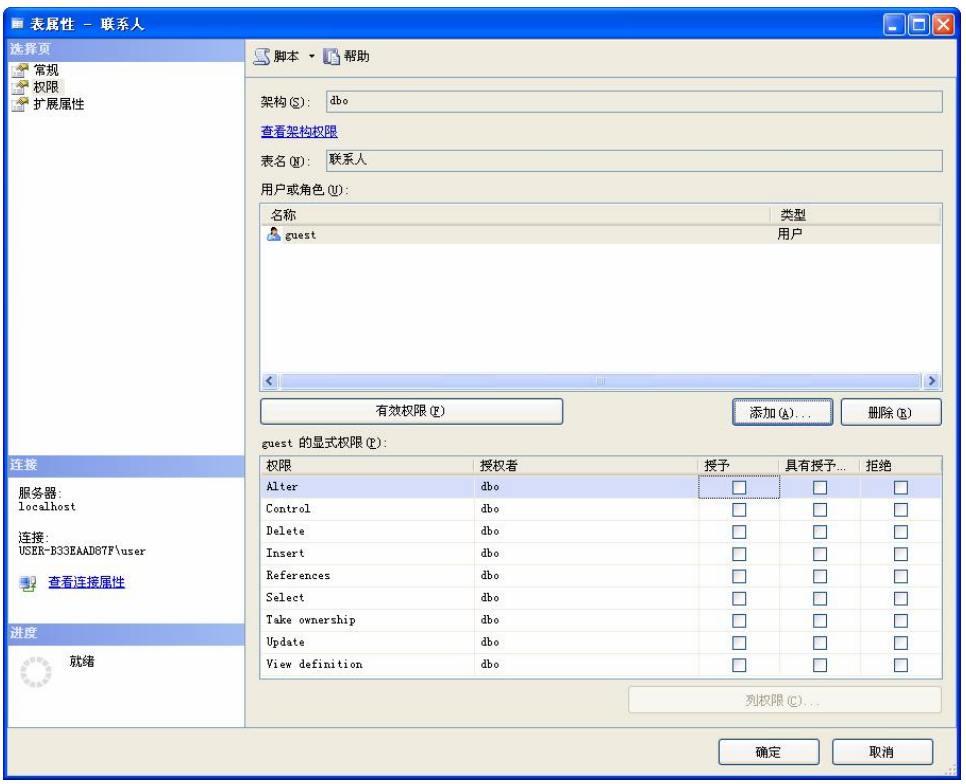


图 12.20　权限页

（3）单击“添加”按钮，弹出“选择用户或角色”对话框，如图 12.21 所示。

图 12.21　“选择用户或角色”对话框

（4）输入用户或角色名，或单击“浏览”按钮，选择需要添加授权的用户或角色，如图 12.22 所示。

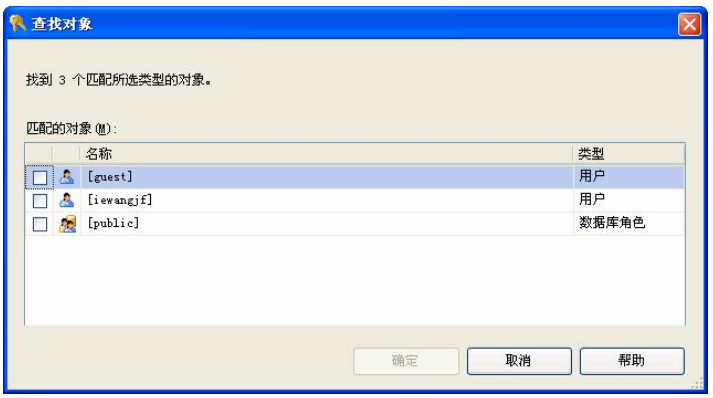


图 12.22　查找对象

（5）单击“确定”按钮关闭查找对象对话框，再关闭选择用户或角色对话框，回到表属性权限页中，如图 12.23 所示。

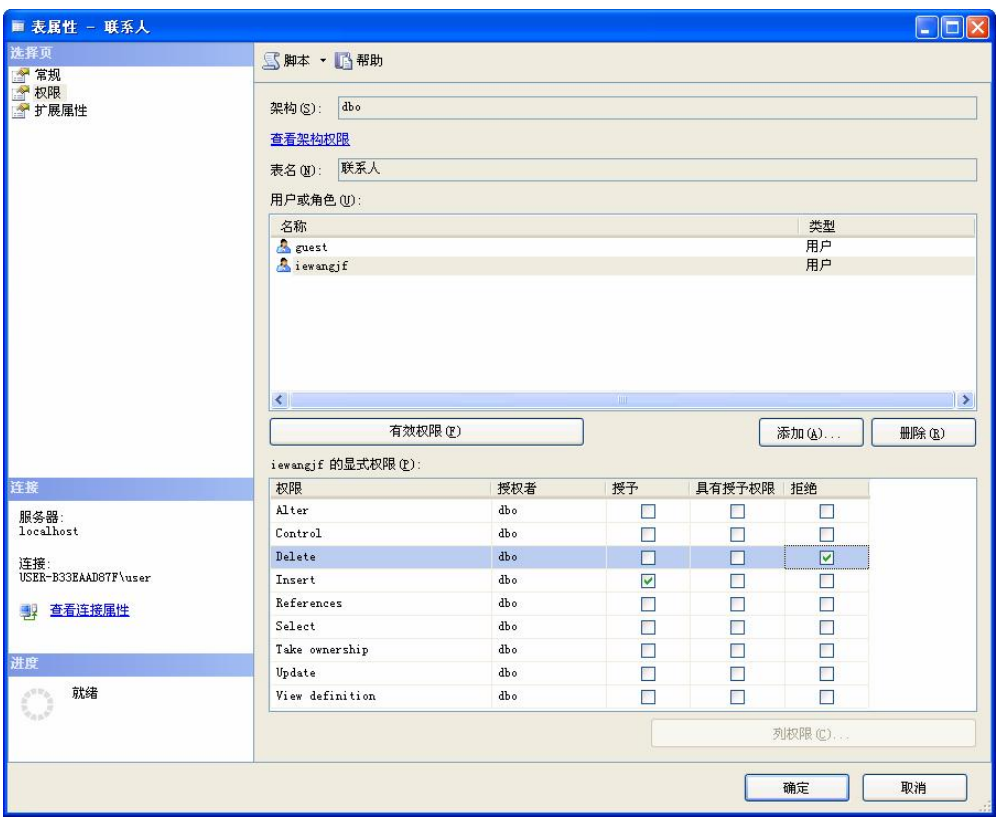


图 12.23　选择用户或角色之后的权限页

（6）选择需要设置权限的用户或角色。设置该用户对每个具体权限的“授予”、“具有授予权限”、“拒绝”3 种权限。如图 12.23 所示，设置用户“iewangjf”对该表的插入（Insert）权限，而拒绝了对表数据的删除（Delete）权限。

（7）如果允许用户具有查询（Select）权限，则列权限可用，单击“列权限”按钮，“列权限”对话框出现，如图 12.24 所示。

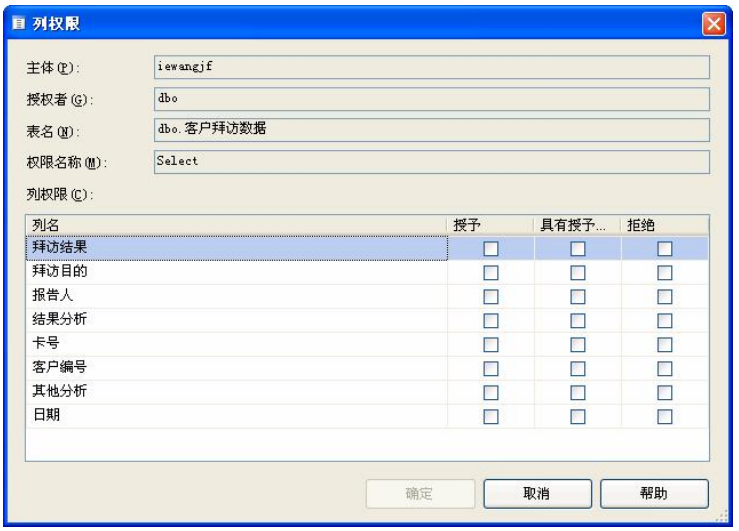


图 12.24 “列权限”对话框

如果不进行设置，则用户从其所属角色中继承权限。设置完列权限之后，单击“确定”按钮关闭列权限对话框。

（8）设置完权限之后，单击“确定”按钮关闭属性页。

另一种方法是通过设置用户或角色的权限来设置权限，请参考 12.5.2 节“创建用户自定义角色”中的相关内容。

2. 使用 Tractans-SQL 管理权限

在 SQL Server 中使用 GRANT、REVOKE 和 DENY3 种命令来管理权限。

（1）GRANT 用来把权限授予某一用户，以允许该用户执行针对该对象的操作，如 UPDATE、SELECT、DELETE、EXECUTE。或允许其运行某些语句，如 CREATE TABLE、CREATE DATABASE。

其简化语法格式为：

```
GRANT { ALL [ PRIVILEGES ] }
        | permission [ ( column [ ,...n ] ) ] [ ,...n ]
            [ ON [ class :: ] securable ]
    TO principal [ ,...n ]
    [ WITH GRANT OPTION ] [ AS principal ]
```

各参数简要说明如下。

- Permission：权限的名称。
- column：指定表中将授予权限的列的名称，需要使用括号“()”。
- securable：指定将授予权限的安全对象。
- TO principal：主体的名称。可为其授予安全对象权限的主体随安全对象而异。
- AS principal：指定一个主体，可将该权限授予其他用户。

【例 12.19】授予用户“WangWei”对数据库的 CREATE TABLE 权限。

```
GRANT CREATE TABLE TO WangWei
```

【例 12.20】授予用户“WangWei”对数据库的 CREATE VIEW 权限并使该用户具有为其他主体授予 CREATE VIEW 的权限。

```
GRANT CREATE VIEW TO WangWei WITH GRANT OPTION
```

【例 12.21】授予用户“WangWei”对表“Person.Address”的 SELECT 权限。

```
GRANT SELECT ON OBJECT::Person.Address TO WangWei
```

（2）REVOKE 用于取消用户对某一对象或语句的权限，这些权限是经过 GRANT 语句授予的。其语法格式和 GRANT 一致。

【例 12.22】从用户“WangWei”以及“WangWei”已授予 VIEW DEFINITION 权限的所有主体中撤消对数据库的 VIEW DEFINITION 权限。

```
REVOKE VIEW DEFINITION FROM CarmineEs CASCADE
```

【例 12.23】撤销用户“WangWei”对表“Person.Address”的 SELECT 权限。

```
REVOKE SELECT ON OBJECT::Person.Address FROM WangWei
```

（3）DENY 用来禁止用户对某一对象或语句的权限，明确禁止其对某一用户对象，执行某些操作。其语法格式和 GRANT 一致。

【例 12.24】拒绝用户“WangWei”对数据库中表“Person.Address”的“SELECT”权限。

```
DENY SELECT ON OBJECT::Person.Address TO WangWei
```

12.7　数据库完整性概述

强制数据完整性可保证数据库中数据的质量。数据完整性是指数据的精确性和可靠性，例如，输入“employee_id”（employee_id 为主键）值为 123 的雇员，则该数据库不应允许其他雇员使用具有相同值的 employee_id。如果想将“employee_rating”列的值范围设定为 1～5，则数据库不应接受值 6。数据完整性分为下列类别。

1. 实体完整性

实体完整性将行定义为特定表的唯一实体。实体完整性通过索引、UNIQUE 约束、PRIMARY KEY 约束或 IDENTITY 属性，强制表的标识符列或主键的完整性。

2. 域完整性

域完整性是指数据库表中的列必须满足某种特定的数据类型或约束。可以强制域完整性限制类型（通过使用数据类型）、限制格式（通过使用 CHECK 约束和规则）或限制值的范围（通过使用 FOREIGN KEY 约束、CHECK 约束、DEFAULT 定义、NOT NULL 定义和规则）。

3. 引用完整性

引用完整性以外键与主键之间或外键与唯一键之间的关系为基础通过 FOREIGN KEY 和 CHECK 进行约束。引用完整性确保键值在所有表中一致。这类一致性要求不能引用不存在的值，如果一个键值发生更改，则整个数据库中，对该键值的所有引用要统一进行更改。

4. 用户定义完整性

用户定义完整性使用户可以定义不属于其他任何完整性类别的特定业务规则。SQL

Server 提供了一些工具来帮助用户实现数据完整性，其中最主要的是约束、规则、触发器，其中触发器在前面章节中已经介绍。

12.8 约　　束

约束是 SQL Server 提供的自动保持数据库完整性的一种方法，SQL Server 2005 提供了下列机制来强制列中数据的完整性：

- PRIMARY KEY 约束；
- FOREIGN KEY 约束；
- UNIQUE 约束；
- CHECK 约束；
- DEFAULT 定义；
- 允许空值。

12.8.1 PRIMARY KEY 约束

表中包含唯一标识表中每一行的一列或一组列作为 PRIMARY KEY 约束。一个表只能有一个 PRIMARY KEY 约束，并且 PRIMARY KEY 约束中的列不能接收空值。由于 PRIMARY KEY 约束可保证数据的唯一性，因此经常对标识列定义这种约束，如学生的学号可作为学生表的主键，而课程编号可作为课程表的主键。有时为了数据的唯一性，还需要另外创建主键列，如银行业务表中的流水号。

其定义语法格式为：

```
[CONSTRAINT constraint_name]
    { PRIMARY KEY | UNIQUE } [CLUSTERED | NONCLUSTERED]
     (column_name1[, column_name2,…,column_name16])
```

各参数说明如下。

- [CONSTRAINT constraint_name]：指定约束的名称，在数据库中应是唯一的。如果不指定，则系统会自动生成。
- CLUSTERED | NONCLUSTERED：为 PRIMARY KEY 或 UNIQUE 约束创建聚集索引或非聚集索引。PRIMARY KEY 约束默认为 CLUSTERED，UNIQUE 约束默认为 NONCLUSTERED。

【例 12.25】创建一个简单的学生表。

```
CREATE TABLE student(
   StuID int PRIMARY KEY,
   StuName varchar(20)   )
```

或者将 Primary key 写到后面。

```
CREATE TABLE student(
   StuID int,
   StuName varchar(20),
   CONSTRAINT pk_stu_ID PRIMARY KEY(stuID)   )
```

12.8.2　FOREIGN KEY 约束

外键（FOREIGN KEY）是用于建立和加强两个表数据之间连接的一列或多列。在外键引用中，当一个表的列被引用作为另一个表的主键列时，就在两表之间创建了连接，这个列就成为第二个表的外键。

FOREIGN KEY 约束要求列中的每个值，都存在于引用的表的对应被引用列中。FOREIGN KEY 约束只能引用在引用的表中是 PRIMARY KEY 或 UNIQUE 约束的列。

其定义的语法格式为：

```
[ CONSTRAINT constraint_name ]
    [ FOREIGN KEY ]
      REFERENCES [ schema_name . ] referenced_table_name [ ( ref_column [ ,... n ] )
```

各参数说明如下。

- FOREIGN KEY REFERENCES：为列中的数据提供引用完整性约束。对于单列，引用 FOREIGN KEY 可以省略。
- [schema_name .] referenced_table_name]：FOREIGN KEY 约束引用的表的名称，以及该表所属架构的名称。
- (ref_column [,... n])：FOREIGN KEY 约束引用的表中的一列或多列。

【例 12.26】创建选课表。

```
CREATE TABLE schedule(
scheduleid int PRIMARY KEY,
subjectid int REFERENCES subject(subjectid),
stuID int CONSTRAINT fk_stu_id   FOREIGN KEY(StuID) REFERENCES student(StuID)   )
```

其中表“schedule”具有两个外键，字段“subjectid”引用的是“subject”表的主键“subjectid”，而 StuID 引用的是“student”的主键 StuID。Constraint fk_stu_id 是约束名称，如果省略，则系统会指定约束名。

外键的目的是控制可以存储在外键表中的数据，插入到“schedule”中的每一行数据的“Stuid”值必须已在“student”中存在。它还可以控制对主键表中数据的更改，如要删除“student”中的 StuID 为 20 的学生时，必须先删除“schedule”中的学生 StuID 为 20 的行。

12.8.3　UNIQUE 约束

使用 UNIQUE 约束确保在非主键列中不输入重复的值。可以对一个表定义多个 UNIQUE 约束，UNIQUE 约束允许 NULL 值，其语法格式请参照 12.8.1 小节。

12.8.4　CHECK 约束

CHECK 约束通过不基于其他列中的数据的逻辑表达式确定有效值。例如，可以通过任何基于逻辑运算符返回 TRUE 或 FALSE 的逻辑（布尔）表达式创建 CHECK 约束。

【例 12.27】将学生表身份证增加 Check 约束，身份证是 15 位或 18 位。

```
CREATE TABLE student(
      StuID int PRIMARY KEY,
```

```
    StuName varchar(20),
    StuIDCard varchar(18) UNIQUE CHECK(len(StuIDCard)=15 or len(StuIDCard)=15)   )
```

12.8.5 DEFAULT 定义

定义列的缺省值，插入数据时，如果列不允许空值且没有 DEFAULT 定义，就必须为该列指定值。否则数据库会返回错误，指出该列不允许空值。

12.8.6 允许空值

NULL 的意思是没有输入，出现 NULL 通常表示值未知或未定义。SQL Server 建议避免允许空值，因为空值会使查询和更新变得更复杂。如果不允许空值，用户向表中输入数据时必须在列中输入一个值，否则数据库将不接收该表行。

【例 12.28】显示在“AdventureWorks”数据库中创建的“PurchaseOrderDetail”表的完整表定义，其中包含所有约束定义。

```
CREATE TABLE [dbo].[PurchaseOrderDetail] (
    [PurchaseOrderID] [int] NOT NULL
        REFERENCES Purchasing.PurchaseOrderHeader(PurchaseOrderID),
    [LineNumber] [smallint] NOT NULL,
    [ProductID] [int] NULL REFERENCES Production.Product(ProductID),
    [UnitPrice] [money] NULL,
    [OrderQty] [smallint] NULL,
    [ReceivedQty] [float] NULL,
    [DueDate] [datetime] NULL,
    [rowguid] [uniqueidentifier] ROWGUIDCOL NOT NULL
        CONSTRAINT [DF_PurchaseOrderDetail_rowguid] DEFAULT (newid()),
    [ModifiedDate] [datetime] NOT NULL
        CONSTRAINT[DF_PurchaseOrderDetail_ModifiedDate] DEFAULT (getdate()),
    [LineTotal]    AS (([UnitPrice]*[OrderQty])),
    [StockedQty]    AS (([ReceivedQty]-[RejectedQty])),
CONSTRAINT [PK_PurchaseOrderDetail_PurchaseOrderID_LineNumber]
    PRIMARY KEY CLUSTERED ([PurchaseOrderID], [LineNumber])
    WITH (IGNORE_DUP_KEY = OFF)
)   ON [PRIMARY]
```

12.8.7 使用 Management Studio 管理约束

在所要修改的表上用鼠标右键单击，在快捷菜单中选择“修改”，出现表的属性页，可设置允许空值约束，在所要修改的列名上用鼠标右键单击，出现快捷菜单，如图 12.25 所示。快捷菜单上“设置主键”用来设置 PRIMARY KEY 约束，“关系”用来设置 FOREIGN KEY 约束，“索引/键”用来设置 UNIQUE 约束，“CHECK 约束”可直接设置 CHECK 约束。下方“默认值或绑定”则可用来设置 DEFAULT 定义。

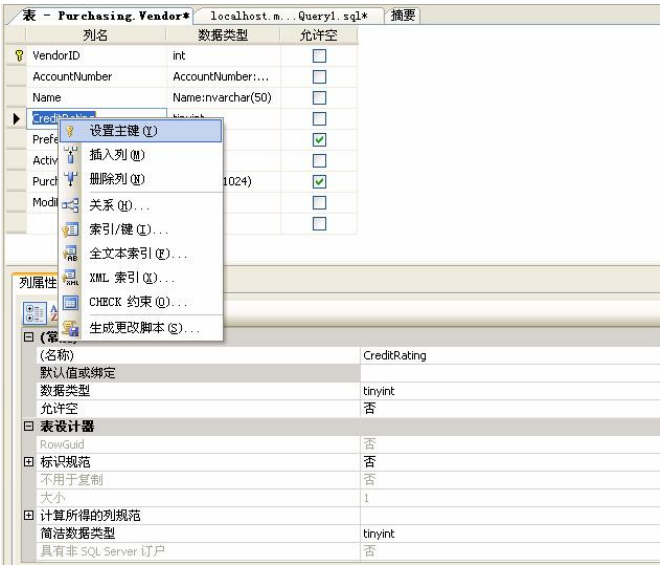


图 12.25　修改表

12.9　规　　则

12.9.1　概述

规则就是数据库中对存储在表中的列或用户自定义数据类型中的值的规定和限制。规则是单独存储的独立的数据库对象。规则与其作用的表或用户自定义数据类型是相互独立的，即表或用户自定义对象的删除、修改不会对与之相连的规则产生影响。

规则和约束可以同时使用。表的列可以有一个规则或多个 CHECK 约束。规则与 CHECK 约束很相似。

注意　后续版本的 Microsoft SQL Server 将删除规则。请避免在新的开发工作中使用该功能，并应着手修改当前还在使用该功能的应用程序。推荐使用 CHECK 约束。

12.9.2　规则的管理

1. 规则的创建

其语法格式为：

```
CREATE RULE [ schema_name . ] rule_name    AS condition_expression
```

【例 12.29】创建具有范围的规则，创建具有列表的规则。

```
CREATE RULE grade_Rule    AS    @range>= 0 AND    @range <100
CREATE RULE sex_Rule    AS    @list IN ('男', '女')
```

2. 规则的绑定

其语法格式为：

```
sp_bindrule [ @rulename = ] 'rule' , [ @objname = ] 'object_name'
```

【例 12.30】将刚才创建的成绩规则绑定到学生成绩表中。

```
Sp_bindRule grade_rule, 'dbo.grade.grade'
```

3．规则的取消绑定

其语法格式：

```
sp_unbindrule [ @objname = ] 'object_name'
```

【例 12.31】取消刚才设置的成绩列的绑定。

```
Sp_unbindRule   'dbo.grade.grade'
```

4．规则的删除

如果规则正绑定到列或别名数据类型，则需先解除绑定才能删除该规则。

其语法格式为：

```
DROP RULE { [ schema_name . ] rule_name } [ ,...n ] [ ; ]
```

【例 12.32】删除规则“grade”和“sex”

```
Drop rule grade_Rule,sex_Rule
```

本 章 小 结

本章介绍了数据库安全性和完整性的基本知识，并详细阐述了身份验证、登录账户、用户、角色、权限的图形界面和 T-SQL 操作，然后介绍了数据库完整性的知识，数据库完整性保证了数据的正确和有效，最后介绍了约束和规则，要掌握约束的内容。

习　　题

一、选择题

1．下列不是数据库角色的是（　　）。

A．固有数据库角色　　B．用户自定义数据库角色

C．应用程序角色　　D．进程管理角色

2．授予用户权限的 Transact-SQL 语句是（　　）。

A．GRANT　　B．REVOKE　　C．DENY　　D．LOGIN

3．可以在服务器中执行任何活动的服务器角色是（　　）。

A．数据库创建角色（dbcreator）　　B．安全管理角色（securityadmin）

C．服务器管理角色（serveradmin）　　D．系统管理角色（sysadmin）

二、填空题

1．SQL Server 支持两种模式的身份验证：_____、_____。

2．完全限定的对象名称包含 4 部分：_____._____._____._____。

3．约束分为_____、_____、_____、_____、_____、_____。

三、简答题

1．简述如何使用 Management Studio 添加角色。

2．什么是架构？什么是默认架构？举例说明。

3．举例说明外键约束。

本 章 实 训

一、实训目的

1．理解 SQL Server 的安全机制。

2．掌握用 Management Studio 进行安全管理的基本操作。

3．掌握用 T-SQL 语句实现建立登录账户，设置数据访问权限等功能的方法与语句。

4．掌握实现各种数据完整性的基本操作，并通过将数据修改为非有效数据体会数据完整性的功能。

二、实训要求

1．实训前做好上机实训的准备，针对实训内容，认真复习与本次实训有关的知识，完成实训内容的预习准备工作。

2．认真独立完成实训内容。

3．实训后做好实训总结，根据实训情况完成总结报告。

三、实训学时

4 学时。

四、实训内容

1．使用 Management Studio 进行安全管理。

（1）创建登录账户“testLogin”，并设置密码。

（2） 在“AdventureWorks”数据库中创建数据库用户“testUser”，登录账号为“testLogin”，数据库角色为“public”。

（3）为数据库用户“testUser”设置访问权限，设置“dbo.person”表的 SELECT 权限，“person.address”表的 UPDATE 权限。

（4）关闭 Management Studio，然后以“testLogin”登录。进入“AdventureWorks”数据库中，修改“dbo.person”的数据，检查是否会报错，使用 SELECT 语句选择“person.address”中的数据，检查是否会报错。

2．使用 T-SQL 执行进行安全管理。

在 Management Studio 中执行以下 SQL 语句：

```
use AdventureWorks
exec sp_addlogin ' testLogin1', '<123>', ' AdventureWorks '
exec sp_grantdbaccess ' testLogin1', 'testUser1'
grant select on [dbo].[Person] to [testUser1]
grant update on [person].[Address] to [testUser1]
go
```

验证步骤同 1。

3．使用 Management Studio 创建表

（1）在数据库中新建数据库“db_stu”。

（2）选择“db_stu”数据库下的表，单击鼠标右键选择“新建表”，输入如图 12.26 所示的同学表结构。

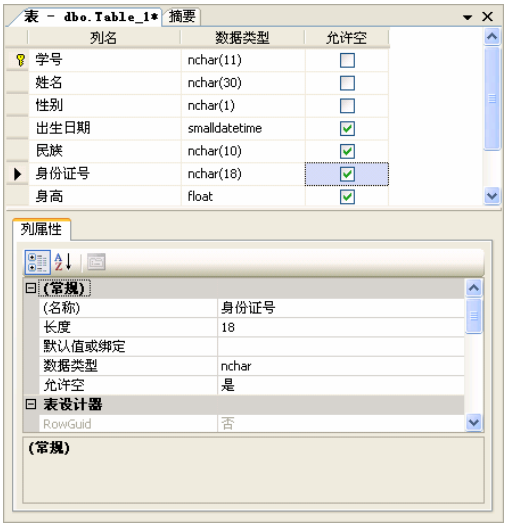


图 12.26　表约束

（3）设置允许为空的字段。

（4）在学号字段上设立主键。

（5）设置民族字段的默认值为“汉族”。

（6）选择身份证号字段，单击鼠标右键选择“索引/键”，设置唯一（UNIQUE）约束。

（7）选择性别字段，单击鼠标右键选择“CHECK 约束”，添加约束表达式（[性别] in ('男', '女')）。

（8）选择身高字段，单击鼠标右键选择“CHECK 约束”，添加约束表达式（[身高] between 0.5 and 2.5）。

4．使用 T-SQL 来创建约束。

在 Management Studio 中执行如下 SQL 语句：

```
create table student（
        学号  nchar(11) primary key ,                          /*主键*/
        姓名  nchar(30) not null ,
        性别  nchar(1) check([性别] in ('男', '女')),          /*检查约束*/
        出生日期  smalldatetime not null ,
        民族  nchar(5) default N'汉族' ,                       /*默认*/
        身份证号  char(18) not null unique,                    /*唯一约束*/
        身高  decimal(5, 2) check([身高] between 0.5 and 2.5) /*检查约束*/
)
GO
```

5．检验约束

在 Management Studio 中打开相应的表修改指定的列，查看其执行或修改结果，体会数据完整性的作用。

（1）非空约束：执行 INSERT student（姓名）VALUES（NULL）。

（2）主键约束：修改“student.学号”，使两同学的学号相同

（3）默认约束：执行 INSERT 同学表（学号，姓名，出生日期，身份证号，身高）。

VALUES ('20070101010', '王伟', '1988-01-01', '123456789012345678',1.78)

观察“性别”、“民族”字段的取值。

（4）唯一约束：修改“student.身份证号”使两同学的身份证号相同。

（5）检查约束：修改“student.性别”为女、“身高”为 0。

五、实训思考题

1．SQL Server 的安全模型主要包括哪几部分？各部分之间有何联系？

2．数据完整性类型及其实现技术有哪些？

第 13 章

VB.NET 与 SQL Server 2005 联合开发

高校教务管理工作是高等教育工作中一个极为重要的环节，是整个院校管理的基础和核心。随着计算机及通信技术的飞速发展，高等教育对教务管理工作提出了更高的要求。面对种类繁多的数据和报表，手工处理方式已经很难跟上现代化管理的步伐，尽快改变传统的管理模式，运用现代化手段进行科学管理，已经成为整个教育系统亟待解决的课题之一。

本章将全面剖析教务管理的内容，由此得出教务管理系统的需求分析和数据建模，并最终演示如何利用 Visual Basic.NET 和 SQL Server 2005 完成系统的开发。

13.1 ADO.NET 数据库访问对象模型

ADO.NET 是重要的应用程序级的接口，用于在 Microsoft.NET 平台上提供数据访问服务。可使用 ADO.NET 访问那些使用新的.NET 数据提供程序的数据源，也可访问那些使用 OLEDB.NET 数据提供程序的现有 OLE DB 数据源。ADO.NET 是专为基于消息的 Web 应用程序而设计的，同时还能为其他应用程序结构提供较好的功能。通过支持对数据的松耦合访问，ADO.NET 减少了与数据库的活动连接数目（即减少了多个用户争用数据库服务器上的有限资源的可能性），从而实现了最大程度的数据共享。

ADO.NET 提供了几种数据访问方法。如果 Web 应用程序或 XML、Web Services 需要访问多个源中的数据，需要与其他应用程序（包括本地和远程应用程序）相互操作或者可受益于保持和传输缓存结果，则数据集是一个极好的选择。作为一种替换方法，ADO.NET 提供了数据命令和数据读取器，以便与数据源直接通信。使用数据命令和数据读取器直接进行的数据库操作包括：运行查询和存储过程，创建数据集对象，使用 DDL 命令直接更新和删除。

分布式 ADO.NET 应用程序的基本对象“数据集”（DataSet）支持基于 XML 的持久性和传输格式，使得 ADO.NET 可以实现最大限度的数据共享。数据集是一种关系数据结构，可使用 XML 进行读取、写入或序列化。越来越多的应用程序要求应用程序层与多个 Web 站点之间进行松耦合的数据交换而 ADO.NET 数据集使得生成这样的应用程序变得很方便。

13.1.1 ADO.NET 结构

以前，数据处理主要依赖于基于连接的双层模型。但当数据处理越来越多地使用多层结构时，程序员开始向断开方式转换，以便为应用程序提供更好的可缩放性。

1. ADO.NET

设计 ADO.NET 组件的目的是为了从数据操作中分解出数据访问。ADO.NET 的两个核心组件会完成此任务：DataSet 和.NET Framework 数据提供程序，后者是一组包括 Connection、Command、DataReader 和 DataAdapter 对象在内的组件。

ADO.NET DataSet 是 ADO.NET 断开式结构的核心组件。DataSet 的设计目的很明确：为了实现独立于任何数据源的数据访问。因此，它可以用于多种不同的数据源，XML 数据，或管理应用程序本地的数据。DataSet 包含一个或多个 DataTable 对象的集合，这些对象由数据行和数据列以及主键、外键、约束和有关 DataTable 对象中数据的关系信息组成。

ADO.NET 结构的另一个核心元素是.NET Framework 数据提供程序，其组件的设计目的相当明确：为了实现数据操作和对数据的快速、只进、只读访问。Connection 对象提供与数据源的连接。Command 对象能够访问用于返回数据、修改数据、运行存储过程以及发送或检索参数信息的数据库命令。DataReader 从数据源中提供高性能的数据流。DataAdapter 提供连接 DataSet 对象和数据源的桥梁。DataAdapter 使用 Command 对象在数据源中执行 SQL 命令，以便将数据加载到 DataSet 中，并使对 DataSet 中数据的更改与数据源保持一致。

可以为任何数据源编写.NET Framework 数据提供程序。.NET Framework 提供了 4 个.NET Framework 数据提供程序：SQL Server.NET Framework 数据提供程序、OLE DB.NET Framework 数据提供程序、ODBC.NET Framework 数据提供程序和 Oracle.NET Framework 数据提供程序。

图 13.1 阐释了 ADO.NET 结构的组件。

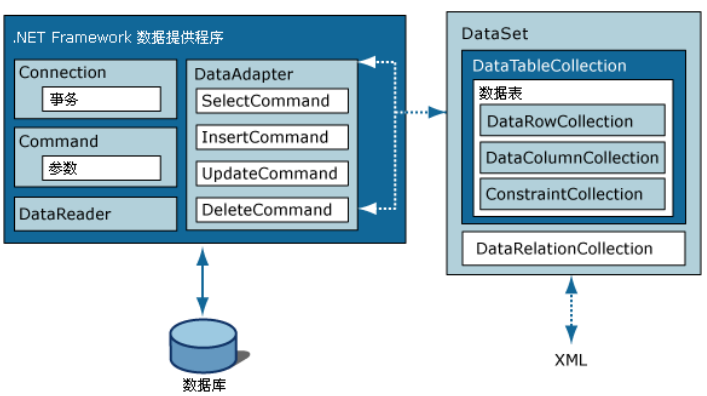


图 13.1 ADO.NET 结构

以下代码示例显示如何将 System.Data 命名空间包含在应用程序中以使用 ADO.NET。

```
Imports System.Data
```

ADO.NET 类在 System.Data.dll 中，并且与 System.Xml.dll 中的 XML 类集成。当编译使用 System.Data 命名空间的代码时，需要引用 System.Data.dll 和 System.Xml.dll。

2. 选择 DataReader 或 DataSet

在决定应用程序应使用 DataReader 还是使用 DataSet 时，应考虑应用程序所需的功能类

型。DataSet 用于执行以下功能。

- 在应用程序中将数据缓存在本地，以便对数据进行处理。如果只需要读取查询结果，DataReader 是更好的选择。
- 在层间或从 XML Web 服务中对数据进行远程处理。
- 与数据进行动态交互，例如绑定到 Windows 窗体控件或组合并关联来自多个源的数据。
- 对数据执行大量的处理，而不需要与数据源保持打开的连接，从而将该连接释放给其他客户端使用。

如果不需要 DataSet 提供的功能，则可以使用 DataReader 以只进、只读方式返回数据，从而提高应用程序的性能。虽然 DataAdapter 使用 DataReader 来填充 DataSet 的内容，但可以使用 DataReader 来提高性能，因为这样可以节省 DataSet 所使用的内存，并将省去创建 DataSet 并填充其内容所需的处理。

13.1.2　数据集介绍

DataSet 对象是支持 ADO.NET 的断开式、分布式数据方案的核心对象。DataSet 是数据的内存驻留表示形式，无论数据源是什么，它都会提供一致的关系编程模型。它可以用于多个不同的数据源，XML 数据，或管理应用程序本地的数据。DataSet 表示包括相关表、约束和表间关系在内的整个数据集。图 13.2 显示 DataSet 对象模型。

DataSet 中的方法和对象与关系数据库模型中的方法和对象一致。

DataSet 也可以按 XML 的形式来保持和重新加载其内容，并按 XML 架构定义语言 (XSD) 架构的形式来保持和重新加载其架构。

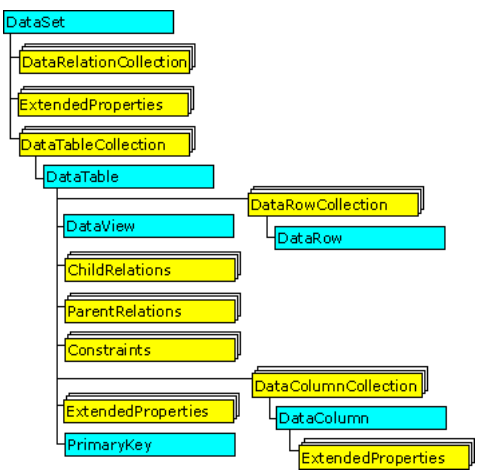


图 13.2　DataSet 对象模型

1. DataTableCollection

一个 ADO.NET DataSet 包含 DataTable 对象所表示的 0 个或多个表的集合。DataTableCollection 包含 DataSet 中的所有 DataTable 对象。

DataTable 在 System.Data 命名空间中定义，表示内存驻留数据表。它包含 DataColumnCollection 所表示的列和 ConstraintCollection 所表示的约束的集合，这些列和约束一起定义了表的架构。DataTable 还包含 DataRowCollection 所表示的行的集合，而 DataRowCollection 则包含表中的数据。除了当前状态，DataRow 还会保留其当前版本和初始版本，以标识对行中存储的值的更改。

2. DataRelationCollection

DataSet 在其 DataRelationCollection 对象中包含关系。关系由 DataRelation 对象来表示，它使一个 DataTable 中的行与另一个 DataTable 中的行相关联。关系类似于存在于关系数据库中的主键列和外键列之间的连接路径。DataRelation 标识 DataSet 中两个表的匹配列。

关系能够在 DataSet 中从一个表导航至另一个表。DataRelation 的基本元素为关系的名称、相关表的名称以及每个表中的相关列。关系可以通过一个表的多个列来生成，方法是将一组 DataColumn

对象指定为键列。当关系被添加到 DataRelationCollection 中时，如果已对相关列值作出更改，它可能会选择添加一个 UniqueKeyConstraint 和一个 ForeignKeyConstraint 来强制完整性约束。

13.2 系统功能设计

要全面理解高校教务管理系统的需求，首先需要了解高校教务管理的基本流程，如图 13.3 所示。

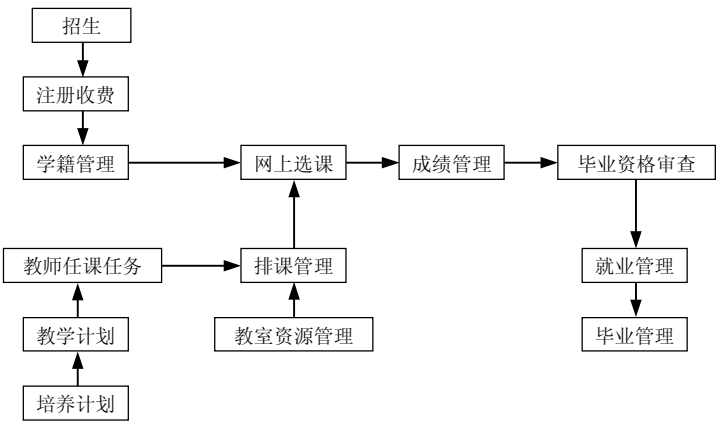


图 13.3 教务管理基本流程

通过对系统进行分析，可以将系统分为以下几个模块。

1. “基础数据管理”功能模块

用于维护整个教务系统正常运行所需的基础数据集，以保证教务系统有一个统一的标准的基础数据集，便于数据的共享，内容包括入学年份、学年学期、院系数据、专业设置、教研室情况等。

2. “教学计划管理”功能模块

用于维护学校中各系各专业的课程、课程计划安排信息，作为选课和毕业审查的标准，包括的功能有课程计划登记、课程计划审批、选课情况查询、选课信息审批等。

3. “学籍管理”功能模块

主要包括高校学籍管理的常用信息，提供对学生学籍基本信息的录入、查询、修改、打印输出、维护等常用功能，并提供学号编排、学生照片输入与显示、学籍变动(留级、休学、跳级、转班、转学、退学等)、奖惩登记、毕业情况等功能。

4. “教师管理”功能模块

用于管理教师相关的信息，提高教学质量，保证教学工作的高效运行。主要包括教师基本信息、教师任课档案、教师考评管理等。

5. “注册收费管理”功能模块

包括“注册管理”与“收费管理”。“注册管理”功能模块用于记录学生新学期的注册情况，如果未注册，将记录学生的未注册原因及去向。“收费管理”功能模块用于记录学生开学初的收费情况，每个学生的收费标准来自学生学籍信息中的收费类别。

6. “排课选课管理”功能模块

用于根据教学计划、教室资源、教师资源等，制定每学期的课程表。主要包括课程信息管理、排课条件设定、教室信息设定，实现人工排课和自动排课。

7. “考务成绩管理”功能模块

用于根据课程自动生成本学期的考试地点、考试时间、监考老师等数据，并对考试的过程和结果进行监控。主要包括考试日程安排、考试情况记录、成绩录入、补考安排等。

8. “毕业管理”功能模块

对学生毕业进行处理，同时对毕业信息、学位授予、证书授予及校友信息等进行管理。主要包括实习管理、论文管理、毕业审核等。

9. “教材管理”功能模块

用于对教材库存、教材计划、教材预订、班级收款、教材采购及教材销售等工作进行有效管期。

由于篇幅有限，本实例详细介绍如图 13.4 所示功能的开发过程，并简化其中各功能，其他功能读者可以参照这些功能的开发方法实现。

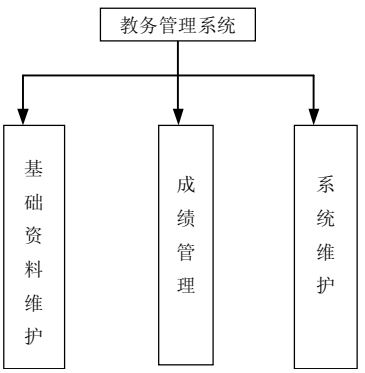


图 13.4　教务管理系统功能模块图

13.3　数据库和表设计

教务管理系统所涉及到的学生资料以及其他有关数据都要存储于数据库中。使用如下 SQL 语句创建数据和表，以及表与表之间的约束关系。

```
CREATE DATABASE JIAOWU;
GO
USE JIAOWU;
GO
--部门
CREATE TABLE [dbo].[Department](
    [DepartmentID] [char](3) NOT NULL,          --部门编号
    [DepartmentName] [varchar](30) NOT NULL,    --部门名称
    [DepartmentHead] [char](8) NOT NULL,        --负责人
 CONSTRAINT [PK_Department] PRIMARY KEY CLUSTERED
([DepartmentID] ASC)
) ON [PRIMARY]
GO
--课程类型
CREATE TABLE [dbo].[Coursetype](
    [coursetypeID] [varchar](3) NOT NULL,       --类型编号
    [typename] [varchar](18) NOT NULL,          --类型名称
```

```
    CONSTRAINT [PK_Coursetype] PRIMARY KEY CLUSTERED
    ([coursetypeID] ASC)
) ON [PRIMARY]
GO
--课程
CREATE TABLE [dbo].[Course](
    [courseID] [char](8) NOT NULL,                  --课程编号
    [coursename] [varchar](20) NOT NULL,            --课程名称
    [coursetypeID] [varchar](3) NOT NULL,           --课程类型
    [totalperiod] [tinyint] NULL,                   --总学时
    [weekperiod] [tinyint] NULL,                    --周学时
    [credithour] [tinyint] NULL,                    --学分
    [remark] [varchar](50),                         --备注
    CONSTRAINT [PK_Course] PRIMARY KEY CLUSTERED
    ([courseID] ASC)
) ON [PRIMARY]
GO
--班级
CREATE TABLE [dbo].[Class](
    [classID] [char](7) NOT NULL,                   --班级编号
    [className] [varchar](12) NOT NULL,             --班级名称
    [specialityID] [char](5),                       --专业
    [specialityName] [varchar](30),                 --专业名称
    [EntranceYear] [char](4),                       --入学年份
    [MonitorID] [char](10),                         --班长
    CONSTRAINT [PK_Class] PRIMARY KEY CLUSTERED
    ([classID] ASC)
) ON [PRIMARY]
GO
--成绩
CREATE TABLE [dbo].[Grade](
    [studentID] [char](10) NOT NULL,                --学号
    [courseID] [char](8) NOT NULL,                  --课程号
    [grade] [tinyint] NULL,                         --成绩
    CONSTRAINT [PK_Grade] PRIMARY KEY CLUSTERED
    ([studentID] ASC,[courseID] ASC)
) ON [PRIMARY]
GO
--专业
```

```
CREATE TABLE [dbo].[Speciality](
    [specialityID] [char](5) NOT NULL,              --专业编号
    [specialityName] [varchar](30) NOT NULL,        --专业名称
    [departmentID] [char](3) NULL,                  --部门编号
    CONSTRAINT [PK_Speciality] PRIMARY KEY CLUSTERED
    ([specialityID] ASC)
) ON [PRIMARY]
GO
--学生
CREATE TABLE [dbo].[student](
    [studentID] [char](10) NOT NULL,                --学号
    [studentName] [varchar](10) NOT NULL,           --姓名
    [nation] [char](10) NULL,                       --民族
    [sex] [char](2) NULL,                           --性别
    [birthday] [datetime] NULL,                     --出生日期
    [classID] [char](7) NULL,                       --班级编号
    [telephone] [varchar](16) NULL,                 --电话
    [credithour] [tinyint] NOT NULL,                --已修学分
    [ru_date] [char](4) NULL,                       --入学年份
    [address] [varchar](50) NULL,                   --籍贯
    [pwd] [varchar](16) NULL,                       --口令
    [remark] [varchar](200) NULL,                   --备注
    CONSTRAINT [PK_student] PRIMARY KEY CLUSTERED
    ([studentID] ASC)
) ON [PRIMARY]
GO
--用户
CREATE TABLE [dbo].[userLogin](
     [loginid] [char](20) NOT NULL,
    [username] [char](20) NOT NULL,                 --用户名
    [password] [varchar](20) NULL,                  --口令
    [allowLogin] [bit] NULL,                        --用户类型
    CONSTRAINT [PK_user] PRIMARY KEY CLUSTERED
    ([loginid] ASC)
) ON [PRIMARY]
GO
--教师
CREATE TABLE [dbo].[Teacher](
    [teacherID] [char](8) NOT NULL,                 --教师编号
```

```
    [teacherName] [varchar](10) NOT NULL,          --姓名
    [departmentID] [char](3) NULL,                 --部门编号
    [sex] [char](2) NULL,                          --性别
    [technicalPost] [char](16) NULL,               --职称
    [telephone] [char](16) NULL,                   --电话
    [homeAddr] [varchar](50) NULL,                 --籍贯
    [pwd] [varchar](16) NULL,                      --口令
    [remark] [varchar](200) NULL,                  --备注
    CONSTRAINT [PK_Teacher] PRIMARY KEY CLUSTERED
    ([teacherID] ASC)
) ON [PRIMARY]
GO
--建立表间约束
ALTER TABLE [dbo].[Course]    WITH CHECK ADD
CONSTRAINT [FK_Course_Coursetype] FOREIGN KEY([coursetypeID])
REFERENCES [dbo].[Coursetype] ([coursetypeID])
GO
ALTER TABLE [dbo].[Class]    WITH CHECK ADD
CONSTRAINT [FK_Class_Speciality] FOREIGN KEY([specialityID])
REFERENCES [dbo].[Speciality] ([specialityID])
GO
ALTER TABLE [dbo].[Grade]    WITH CHECK ADD
CONSTRAINT [FK_Grade_Course] FOREIGN KEY([courseID])
REFERENCES [dbo].[Course] ([courseID])
GO
ALTER TABLE [dbo].[Grade]    WITH CHECK ADD
CONSTRAINT [FK_Grade_student] FOREIGN KEY([studentID])
REFERENCES [dbo].[student] ([studentID])
GO
ALTER TABLE [dbo].[Speciality]    WITH CHECK
ADD    CONSTRAINT [FK_Speciality_Department] FOREIGN KEY([departmentID])
REFERENCES [dbo].[Department] ([DepartmentID])
GO
ALTER TABLE [dbo].[student]    WITH CHECK
ADD    CONSTRAINT [FK_student_Class] FOREIGN KEY([classID])
REFERENCES [dbo].[Class] ([classID])
GO
ALTER TABLE [dbo].[Teacher]    WITH CHECK
ADD CONSTRAINT [FK_Teacher_Department] FOREIGN KEY([departmentID])
```

```
REFERENCES [dbo].[Department] ([DepartmentID])
GO
```

建立成功后，JIAOWU 数据库的数据库关系图如图 13.5 所示。

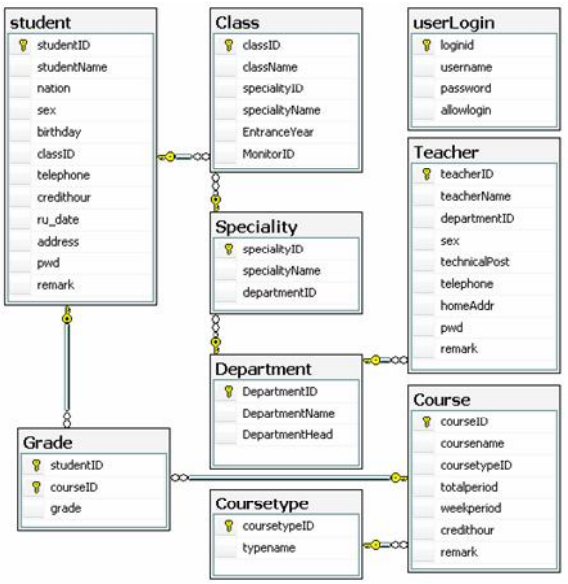


图 13.5　数据库关系图

13.4　程序开发

13.4.1　创建项目

启动 Microsoft Visual Studio.NET 2005，在主菜单中选择“文件”|“新建”|“项目”命令，弹出“新建项目”对话框，在“项目类型”列表框中选择“Visual Basic 项目”，然后在“模版”列表框中选择“Windows 应用程序”。在“名称”文本框中输入项目名称“JiaoWuMis”，并设置项目保存的位置。单击“确定”按钮完成新项目的创建。

13.4.2　初始界面

在主菜单中选择“项目”|“添加 Windows 窗体”命令，弹出“添加新项”对话框。在“模板”列表框中选择“初始屏幕”，并把名称改为“SplashJW.vb”，单击“确定”按钮完成创建。创建的初始界面如图 13.6 所示。

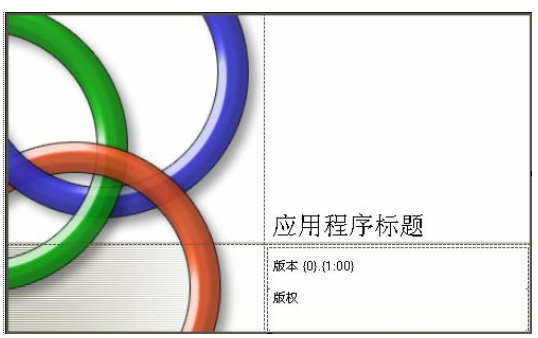


图 13.6　初始界面

在初始界面上可以设置应用程序标题、版本、版权等信息，可以通过属性窗口进行修改，也可以在项目属性中进行修改。在主菜单中选择“项目”|“JiaoWuMis 属性”命令，打开项

目属性窗口，如图 13.7 所示。

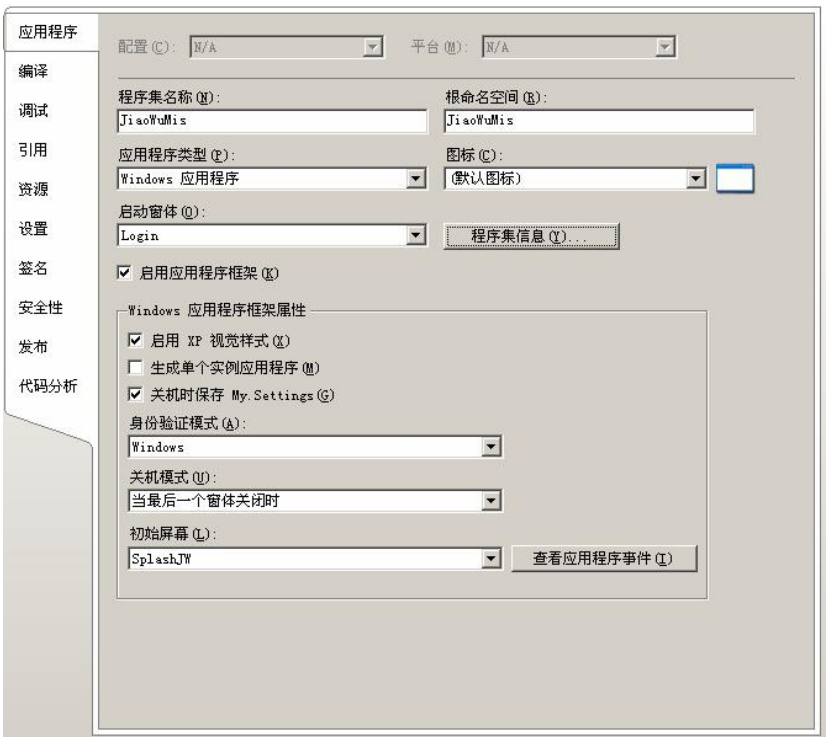


图 13.7　项目属性

在项目属性窗口中，设置“初始屏幕”为“SplashJW”，单击“程序集信息”按钮，打开“程序集信息”对话框，在对话框中设置标题、版本、版权等信息，如图 13.8 所示。

图 13.8　程序集信息

按“F5”运行程序，就可以看到初始界面中的应用程序标题、版本、版权等信息已替换为新设置的值。

13.4.3　登录窗口

在主菜单中选择“项目”|“添加 Windows 窗体”命令，弹出“添加新项”对话框。在“模板”列表框中选择“登录窗体”，并把名称改为“Login.vb”，单击“确定”按钮完成创建。创建的登录窗口如图 13.9 所示。

图 13.9　登录窗口

在登录窗口中需要判断用户名和密码是否正确，需要从数据库中获得用户信息。访问数据库首先要和数据库建立连接，需要指定连接字符串。连接字符串可以在项目属性中进行设置，供整个项目使用，在项目属性窗口中选择“设置”选项，如图 13.10 所示。

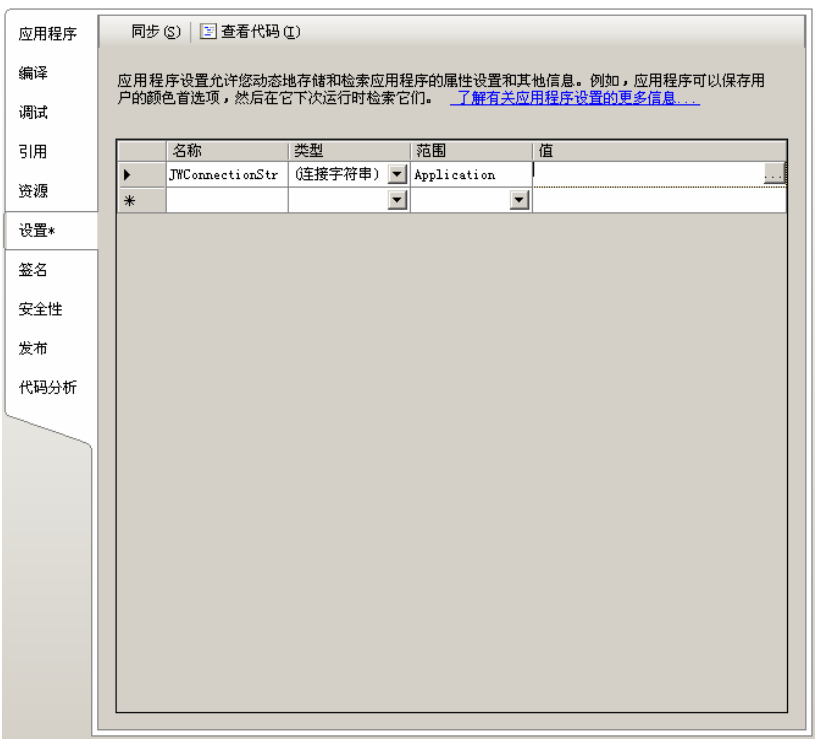


图 13.10　设置连接字符串

设置连接的名称为“JWConnectionStr”，类型为“(连接字符串)”，范围为“Application”，连接字符串的值可以通过单击右侧的按钮，弹出“连接属性”对话框设置，如图 13.11 所示。

在“连接属性”对话框中选择服务器名，设置登陆服务器使用的登录方式。在这里选择“使用 Windows 身份验证”，然后选择连接的数据库。单击“测试连接”按钮可以测试连接是否成功，如连接成功，单击“确定”按钮可以设置连接字符串。

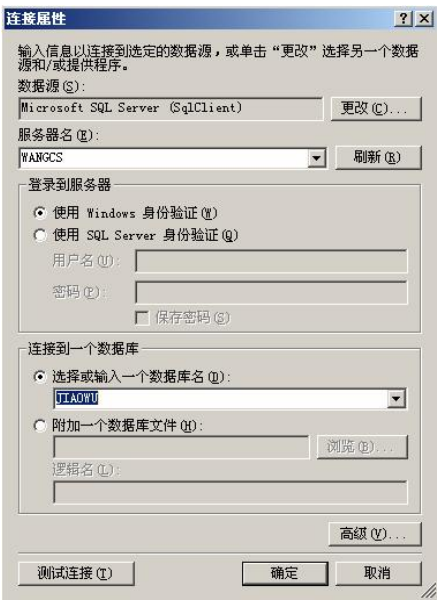


图 13.11　连接属性

登录窗体的代码如下：

```
Imports System.Data
Imports System.Data.SqlClient
Public Class Login
'确定
Private Sub OK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles OK.Click
        '建立连接
        Dim cn As New SqlConnection(My.Settings.JWConnectionStr)
        '建立命令，调用存储过程
        Dim cmd As New SqlCommand("usp_userLogin", cn)
        cmd.CommandType = CommandType.StoredProcedure
        '增加参数
        cmd.Parameters.Add("@loginid", SqlDbType.VarChar, 20)
        cmd.Parameters.Add("@password", SqlDbType.VarChar, 20)
        cmd.Parameters.Add("@reason", SqlDbType.VarChar, 20)
        '给参数赋值
        cmd.Parameters(0).Value = UsernameTextBox.Text
        cmd.Parameters(1).Value = PasswordTextBox.Text
```

```
            '参数默认为输入参数，输出参数需设置为 ParameterDirection.Output
            cmd.Parameters(2).Direction = ParameterDirection.Output
            Try
                '打开连接
                cn.Open()
                '执行命令
                cmd.ExecuteNonQuery()
                Dim reason As String
                 '获得输出参数的值
                reason = cmd.Parameters(2).Value.ToString
                If reason = "成功" Then
                    Form1.Show()
                    Me.Close()
                Else
                    MsgBox(reason)
                    UsernameTextBox.Text = ""
                    PasswordTextBox.Text = ""
                    UsernameTextBox.Focus()
                End If
            Catch ex As Exception
                MsgBox(ex.ToString)
            Finally
                cn.Close()
            End Try
        End Sub
         '取消
        Private Sub Cancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Cancel.Click
            Me.Close()
        End Sub
    End Class
```

代码说明如下。

在“确定”按钮的“Click”事件中，建立一个命令 cmd，cmd 调用了存储过程 usp_userLogin，usp_userLogin 的定义如例 9.7 所示。

调用存储过程时，须注意参数的添加、赋值，以及参数值的获得。

13.4.4　主窗口

利用系统默认生成的窗口作为主窗口，并为其添加主菜单控件和其他控件，布局如图 13.12 所示。

主窗口属性设置如表 13.1 所示。

图 13.12　主窗口

表 13.1　　主窗口属性设置

控 件 类 型	对　象　名	属　　性	值
Form	FrmMain	Name	FrmMain
		Text	教务管理系统
		IsMdiContainer	True

主窗口菜单属性如表 13.2 所示。

表 13.2　　主窗口菜单属性设置

菜　单　栏	菜　单　项	属　　性	值
ToolStripMenuItem1		Text	基础资料
	MenuItemDepartment	Text	院系资料
	MenuItemSpec	Text	专业资料
	MenuItemTeacher	Text	教师资料
	MenuItemCourse	Text	课程资料
	MenuItemClass	Text	班级资料
	MenuItemStudent	Text	学生资料
ToolStripMenuItem2		Text	教学管理
	MenuItemQuery	Text	成绩查询
	MenuItemInput	Text	成绩录入
ToolStripMenuItem3		Text	用户管理
	MenuItemAddUser	Text	添加用户
	MenuItemModify	Text	修改密码
ToolStripMenuItem4		Text	关于
ToolStripMenuItem5		Text	退出系统

13.4.5　基础资料

（1）在项目“JiaoWuMis”中，新增一个 Windows 窗口，命名为“Frm Department”。在

设计模式下打开“Frm Department”窗口。

（2）转到数据源窗口，如这个窗口不可见，那么在主菜单中选择“数据”|“显示数据源”命令。点击链接“添加新的数据源”，打开数据源配置向导，如图 13.13 所示。

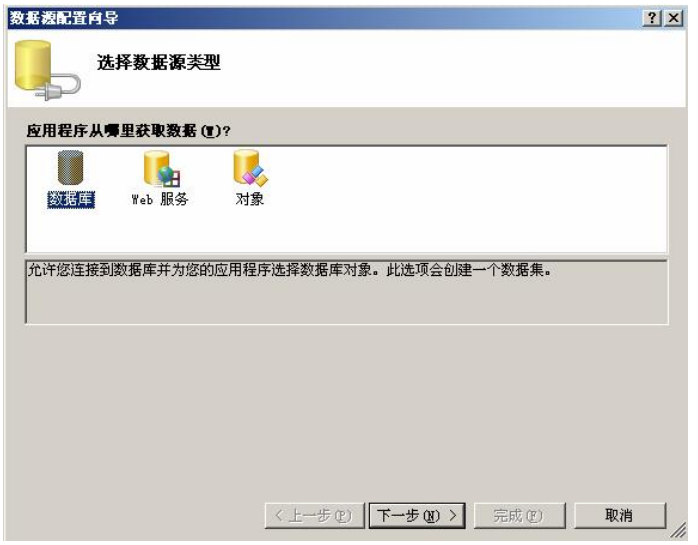


图 13.13　数据源配置向导

（3）选择数据源类型，单击“下一步”按钮。选择数据连接，如图 13.14 所示。

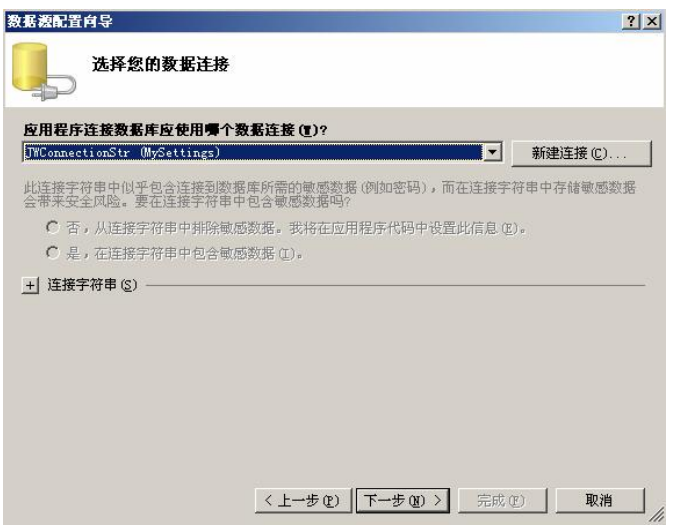


图 13.14　选择数据连接

（4）选择前面设置的连接“JWConnectionStr”，单击“下一步”按钮。选择数据库对象，如图 13.15 所示。

（5）选择数据库对象，并设置数据集（DataSet）名称。单击“完成”按钮完成数据源的配置。

（6）在数据源下选择“Department”数据表，单击表名后面的下拉箭头。选择“DataGridView”，如图 13.16 所示。

图 13.15　选择数据对象

（7）把“Department”数据表拖放到窗口的表面。这个操作会在窗口的表面添加一系列的控件，如图 13.17 所示。

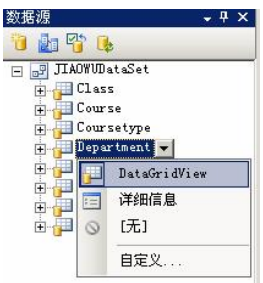


图 13.16　配置数据源

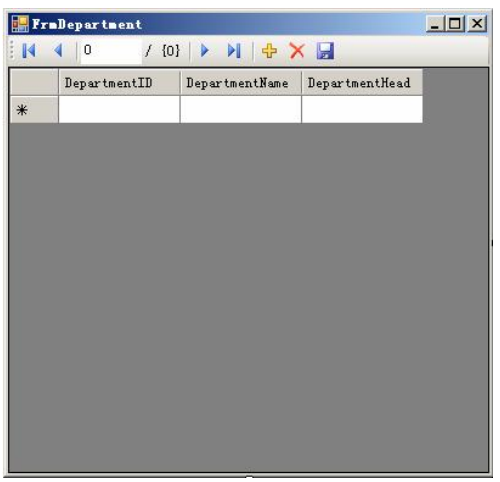


图 13.17　重新调整后的窗口

（8）选择“DepartmentDataGridView”控件，单击右上角的三角形，在弹出菜单中选择“编辑列”命令，打开“编辑列”对话框，如图 13.18 所示。分别将“DepartmentID”、“DepartmentName”、“DepartmentHeader”的“HeaderText”属性修改为“部门编号”、“部门名称”、“部门负责人”。

（9）在主菜单中，双击“院系资料”菜单，编写如下程序：

```
FrmDepartment.MdiParent = Me
FrmDepartment.Show()
```

运行程序，该窗体已可以实现部门的增加、删除、修改等基本功能，运行效果如图 13.19 所示。

采用相同的方法即可完成其他窗体的创建。

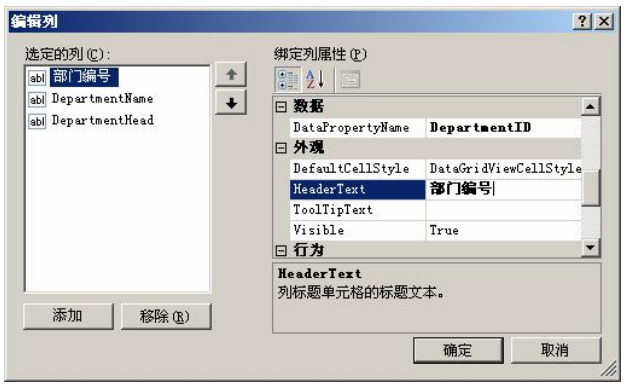


图 13.18　编辑列

图 13.19　院系资料

13.4.6　成绩管理

1. 成绩查询

在项目中添加一个新的窗口，命名为“FrmQuery”，并添加控件，布局如图 13.20 所示。

图 13.20　成绩查询

窗口及其控件的属性设置如表 13.3 所示。

表 13.3　　控件属性设置

控 件 类 型	对　象　名	属　　性	值
Form	FrmQuery	Name	FrmQuery
		Text	成绩查询
Label	Label1	Text	班级
	Label2	Text	课程
	Label3	Text	学号
ComboBox	ComboBox1	Name	ComboBox1
	ComboBox2	Name	ComboBox2
TextBox	TextBox1	Text	

续表

控 件 类 型	对 象 名	属 性	值
Button	Button1	Text	按班级课程查询
	Button2	Text	按学号查询
DatagridView	DatagridView1	Name	DatagridView1

成绩查询提供了两种查询方式：一种是按班级、课程进行查询，另一种是按照学号进行查询。班级和课程分别使用了两个组合框（ComboBox）控件，用户可以直接进行选择。在窗口的加载（Load）事件中，需要给这两个组合框填充数据。

窗口“Load”事件代码如下：

```
Private Sub FrmQuery_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        '建立连接
        Dim cn As New SqlConnection(My.Settings.JWConnectionStr)
        '建立两个数据适配器，分别用于班级、课程
        Dim da1 As New SqlDataAdapter("select classid,classname from class", cn)
        Dim da2 As New SqlDataAdapter("select courseId,coursename from course", cn)
        '定义两个数据表
        Dim tblClass, tblCourse As New DataTable
        '打开连接
        cn.Open()
        '填充数据表
        da1.Fill(tblClass)
        da2.Fill(tblCourse)
        '关闭连接
        cn.Close()
        '将数据与控件进行绑定。DataSource：指定数据源，为数据集或数据表对象
        'DisplayMember：显示的数据  ValueMember：数据实际值
        ComboBox1.DataSource = tblClass
        ComboBox1.DisplayMember = "className"
        ComboBox1.ValueMember = "classID"
        ComboBox2.DataSource = tblCourse
        ComboBox2.DisplayMember = "courseName"
        ComboBox2.ValueMember = "courseID"
End Sub
```

“按班级课程查询”按钮单击事件代码如下：

```
'按班级课程查询
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim cn As New SqlConnection(My.Settings.JWConnectionStr)
        Dim tbl As New DataTable
        '建立数据适配器
```

```
        Dim da As New SqlDataAdapter("select grade.studentId as 学号,studentname as 姓名,grade as 成绩
from grade,student where grade.studentid=student.studentid and student.classid='" & ComboBox1.SelectedValue & "' and
grade.courseid='" & ComboBox2.SelectedValue & "'", cn)
        '打开连接，填充数据表，然后关闭连接
        cn.Open()
        da.Fill(tbl)
        cn.Close()
        '设置 DataGridView1 的数据源属性，让数据显示出来
        DataGridView1.DataSource = tbl
    End Sub
```

“按学号查询”按钮的单击事件代码如下：

```
    '按学号查询
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        Dim cn As New SqlConnection(My.Settings.JWConnectionStr)
        Dim tbl As New DataTable
        Dim da As New SqlDataAdapter("select studentname as 姓名,coursename as 课程,grade as 成绩 from
student,grade,course where grade.studentid=student.studentid and grade.courseid=course.courseid and grade.studentid='" &
TextBox1.Text & "'", cn)
        cn.Open()
        da.Fill(tbl)
        cn.Close()
        DataGridView1.DataSource = tbl
    End Sub
```

2. 成绩录入

成绩录入有多种方式，单个录入比较简单，就不介绍了。主要介绍一下按照班级、课程录入成绩的方法。首先，在项目中新建一个窗口“FrmInput”，添加一些控件并修改属性，窗口的布局如图 13.21 所示。

图 13.21　成绩录入

在窗口的“Load”事件中，需要给班级、课程两个组合框填充数据。选择好班级和课程后，单击“加载”按钮，在 DataGridView1 控件中打开成绩单。录入完成后，单击“保存”按钮保存新录入的成绩。

窗口的“Load”事件代码如下：

```
Private Sub FrmInput_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim cn As New SqlConnection(My.Settings.JWConnectionStr)
    Dim da1 As New SqlDataAdapter("select classid,classname from class", cn)
    Dim da2 As New SqlDataAdapter("select courseId,coursename from course", cn)
    Dim tblClass, tblCourse As New DataTable
    cn.Open()
    da1.Fill(tblClass)
    da2.Fill(tblCourse)
    cn.Close()
    ComboBox1.DataSource = tblClass
    ComboBox1.DisplayMember = "className"
    ComboBox1.ValueMember = "classID"
    ComboBox2.DataSource = tblCourse
    ComboBox2.DisplayMember = "courseName"
    ComboBox2.ValueMember = "courseID"
End Sub
```

“保存”按钮的单击事件代码如下：

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim cn As New SqlConnection(My.Settings.JWConnectionStr)
    Dim rw As DataGridViewRow
    Dim cmdInsert As New SqlCommand("insert into grade values(@sid,@cid,@cj)", cn)
    cmdInsert.Parameters.Add("@sid", SqlDbType.Char, 10)
    cmdInsert.Parameters.Add("@cid", SqlDbType.Char, 8)
    cmdInsert.Parameters.Add("@cj", SqlDbType.TinyInt)
    cmdInsert.Parameters(1).Value = ComboBox2.SelectedValue
    cn.Open()
    For Each rw In DataGridView1.Rows
        If rw.Cells(0).Value <> "" Then
            cmdInsert.Parameters(0).Value = rw.Cells(0).Value
            cmdInsert.Parameters(2).Value = rw.Cells(2).Value
            cmdInsert.ExecuteNonQuery()
        End If
    Next
    cn.Close()
End Sub
```

“加载”按钮的单击事件代码如下：

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    Dim cn As New SqlConnection(My.Settings.JWConnectionStr)
    Dim da As New SqlDataAdapter("SELECT studentID, studentName, " AS grade FROM student WHERE classID = '" & ComboBox1.SelectedValue & "'", cn)
    Dim tbl As New DataTable
    da.Fill(tbl)
    DataGridView1.DataSource = tbl
    End Sub
```

13.4.7　用户管理

1. 添加用户

在项目中新建一个窗口“FrmAddUser”，添加 4 个 Label 控件、4 个 TextBox 控件、2 个 Button 控件，并修改控件的属性，窗口的布局如图 13.22 所示。

图 13.22　添加用户

在“添加”按钮的单击事件中，首先要验证 4 个 TextBox 的“Text”属性是否为空，然后验证两次输入的密码是否相同。如验证通过，执行一个命令来完成添加用户操作。

“添加”按钮的代码如下：

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    If TextBox1.Text = "" Or TextBox2.Text = "" Or TextBox3.Text = "" Then
        MsgBox("登录 ID、用户名、密码均不能为空！")
        Exit Sub
    End If
    If TextBox3.Text <> TextBox4.Text Then
        MsgBox("两次输入的密码不同，请重新输入！")
        Exit Sub
    End If
    Dim cn As New SqlConnection(My.Settings.JWConnectionStr)
    Dim cmd As New SqlCommand("insert into userlogin values('" & TextBox1.Text & "','" & TextBox2.Text & "','" & TextBox3.Text & "',0)", cn)
    cn.Open()
    cmd.ExecuteNonQuery()
    cn.Close()
End Sub
```

2. 修改密码

往项目中新增一个窗口“FrmPassword”，添加控件并设置相应的属性，窗体布局如图 13.23

所示。

图 13.23　修改密码

为了保护数据库的安全，修改用户密码需输入用户名和原密码，进行验证。新密码要求用户输入两次，避免密码设置错误。在“修改”按钮的单击事件中，首先要验证两次输入的用户口令是否正确，然后验证用户原密码是否正确，最后完成修改操作。

“修改”按钮的单击事件代码如下：

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        If TextBox3.Text <> TextBox4.Text Then
            MsgBox("两次输入的密码不同，请重新输入！")
            Exit Sub
        End If
        Dim cn As New SqlConnection(My.Settings.JWConnectionStr)
        Dim cmd As New SqlCommand("select password from userlogin where loginid='" & TextBox1.Text & "'", cn)
        Dim pwd As String
        cn.Open()
        pwd = cmd.ExecuteScalar
        If pwd <> TextBox2.Text Then
            MsgBox("原密码错误，请重新输入！")
        Else
            cmd.CommandText = "update userlogin set password='" & TextBox3.Text & "' where loginid='" & TextBox1.Text & "'"
            cmd.ExecuteNonQuery()
        End If
        cn.Close()
        End Sub
```

13.4.8　“关于”窗口

在主菜单中选择“项目”|“添加 Windows 窗体”命令，弹出“添加新项”对话框。在“模板”列表框中选择“关于窗体”，并把名称改为“FrmAbout.vb”，单击“确定”按钮完成创建。

创建的“关于”窗口如图 13.24 所示。

图 13.24　“关于”窗口

在“关于”窗口中，产品名称、版本、版权等信息可以在图 13.8 所示的程序集信息对话框中进行设置，而不需要在窗口中修改。

本 章 小 结

本章采用 ADO.NET 数据访问对象模型，开发了一个数据库应用程序——教务管理系统。

首先简单介绍了 ADO.NET 的基本知识，以及使用 ADO.NET 访问数据的方法。然后在实例中分别使用拖动和编写代码两种方法实现对数据库的操作，以及调用 SQL Server 2005 中的存储过程。

该系统功能简单，读者可以进一步将它完善。可以考虑加入网络方面的功能，如学生可以通过该系统网上选课，在线查询考试成绩、学籍信息等。

参 考 文 献

[1] 李维杰，孙乾君. SQL Server 2005 数据库原理与应用简明教程. 北京：清华大学出版社，2007

[2] 喻梅，汪洋，于健.SQL Server 2005 基础教程.北京：清华大学出版社，2007

[3] Solid Quality Learning.SQL Server 2005 从入门到精通.北京：清华大学出版社，2007

[4] 王珊，萨师煊. 数据库系统概论（第四版）.北京：高等教育出版社，2006

[5] 周立柱.SQL Server 数据库原理——设计与实现.北京：清华大学出版社，2004

[6] 陈雁，周如意，滕刚，王文，李武韬.数据库系统原理与设计.北京：中国电力出版社，2006