第 5 章　SQL Server 2008 高级应用

Transact-SQL 是 SQL Server 2008 的结构化编程语言，在数据库系统管理、复杂的查询和应用系统的开发中被广泛使用。Transact-SQL 程序设计对于 SQL Server 2008 系统而言是至关重要的，是使用 SQL Server 2008 的主要形式。

本章主要有以下两方面内容：

（1）Transact-SQL 程序设计的基本语法要素，包括注释、变量、运算符、函数和流程控制语句、常用命令和常用函数。

（2）存储过程、触发器和自定义函数的有关基本概念，在 SQL Server 环境中存储过程、触发器和自定义函数的创建、查看、修改和删除等方法。

5.1　Transact-SQL 程序设计

Transact-SQL 是在标准 SQL 基础上进行扩充而推出的 SQL Server 专用的结构化 SQL，如引用了程序设计的思想、增强了程序的流程控制语句等。Transact-SQL 最主要的用途是设计服务器端的能够在后台执行的程序块，如存储过程、触发器等。大型的关系数据库系统都在标准 SQL 的基础上结合自身的特点推出了可以编程的结构化 SQL 编程语言，如 SQL Server 的 Transact-SQL、Oracle 的 PL/SQL 等。

5.1.1　Transact-SQL 程序结构

一般而言，一个 Transact-SQL 程序结构由如下要素组成：

- 注释
- 批处理
- 程序中使用的变量
- 流程控制语句
- 错误和消息的处理

下面通过一个实例来说明 Transact-SQL 程序结构。

【例 5.1】从 SC 中计算学号为 S2 学生所学课程的平均分，如该平均分大于或等于 60 分，则程序输出课程平均成绩超过 60，否则课程平均成绩不超过 60。

```
/*Transact-SQL 程序实例*/
DECLARE @AVGSCORE DECIMAL
SET @AVGSCORE=60
IF (SELECT AVG (SCORE) FROM SC WHERE SNUM=S2）>= @AVGSCORE
--输出结果
PRINT '课程平均成绩超过'+CONVERT(VARCHAR(10),@AVGSCORE)
ELSE
--输出结果
PRINT '课程平均成绩不超过'+CONVERT(VARCHAR(10),@AVGSCORE)
GO
```

5.1.2 Transact-SQL 程序的批处理

批处理是从客户机传递到服务器上的一组完整的数据和 SQL 指令。在一个批处理中，可以包含一条 SQL 指令，也可以包含多条 SQL 指令。

所有的批处理命令都使用 GO 作为结束的标志。当 SQL Server 编译器读到 GO 时，它就会把 GO 前面所有的语句当作一个批处理而打包成一个数据包发送给服务器。GO 本身并不是 Transact-SQL 语句的组成部分，它只是一个用于表示批处理结束的前端指令。SQL Server 会将一批 T-SQL 语句当成一个执行单元，将其编译后一次执行，而不是将一个个 T-SQL 语句编译后再一个个执行。

【例 5.2】批处理示例。

```
SELECT * FROM S
WHERE   SNAME='李刚'
UPDATE S
SET DNUM=2
WHERE SNAME='李刚'
GO
SELECT * FROM S
SNAME='李刚'
GO
```

5.1.3 系统数据类型

在 SQL Server 中，每个变量、参数和表达式都有数据类型。所谓数据类型就是以数据的表现方式和存储方式来划分的数据的种类。SQL Server 中提供多种基本数据类型，这些数据类型主要包括整型、浮点型、二进制型、逻辑型和字符型等。

1. 整数数据类型

整数型数据包括 bigint 型、int 型、smallint 型和 tinyint 型。

（1）bigint 型数据的存储大小为 8 个字节，共 64 位，其中 63 位用于表示数值的大小，1 位用于表示符号。bigint 型数据可以存储的数值范围是-2^{63}～$2^{63}-1$。

（2）int 型数据的存储大小为 4 个字节，共 32 位，其中 31 位用于表示数值的大小，1 位用于表示符号。int 型数据存储的数值范围是-2^{31}～$2^{31}-1$。

（3）smallint 型数据的存储大小为 2 个字节，共 16 位，其中 15 位用于表示数值的大小，1 位用于表示符号。smallint 型数据存储的数值范围是-2^{15}～$2^{15}-1$。

（4）tinyint 型数据的存储大小只有 1 个字节，共 8 位，全部用于表示数值的大小，由于没有符号位，所以 tinyint 型的数据只能表示正整数。tinyint 型数据存储的数值范围是 0～2^8-1。

2. 浮点数据类型

浮点数据类型用于存储十进制小数。在 SQL Server 中浮点数值的数据采用上舍入（Round up）的方式进行存储，也就是说，要舍入的小数部分不论其大小，只要是一个非零的数，就要在该数字的最低有效位上加 1，并进行必要的进位。由于浮点数据为近似值，所以并非数据类型范围内的所有数据都能精确地表示。

浮点数据类型包括 real 型、float 型、decimal 型和 numeric 型。

（1）real 型数据的存储大小为 4 个字节，可精确到小数点后第 7 位数字。这种数据类型的数

据存储范围为-3.40E+38～-1.18E-38。

（2）float 型的数据存储大小为 8 个字节，可精确到小数点后第 15 位数字。这种数据类型的数据存储范围为-1.79E+308～-2.23E-308。

float 型的数据可以写成 float[(n)]的形式。其中 n 是 1～15 之间的整数值，指定 float 型数据的精度。当 n 为 1～7 时，实际上用户定义了一个 real 型的数据，系统用 4 个字节存储；当 n 为 8～15 时，系统认为它是个 float 型的数据，用 8 个字节存储。这样既增强了数据定义的灵活性，又节省了空间。

（3）decimal 数据类型和 numeric 数据类型的功能完全一样，它们都可以提供小数所需要的实际存储空间，但也有一定的限制，用户可以用 2～17 个字节来存储数据，取值范围是-10^{38}+1～10^{38}-1。

decimal 型数据和 numeric 型数据的定义格式为 decimal[(p,[s])]和 numeric[(p,[s])]，其中 p 表示可供存储的值的总位数（不包括小数点），默认值为 18；s 表示小数点后的位数，默认值为 0；参数之间的关系是 0≤s≤p。例如 decimal(15,5)，表示共有 15 位数，其中整数 10 位，小数 5 位。

3. 二进制数据类型

二进制数据类型用于存储二进制数据，包括 binary 型、varbinary 型和 image 型。

（1）binary 型是固定长度的二进制数据类型，其定义形式为 binary(n)，其中 n 表示数据的长度，取值为 1～8000。在使用时应指定 binary 型数据的大小，默认值为 1 个字节。binary 类型的数据占用 n+4 个字节的存储空间。

在输入数据时必须在数据前加上字符“0X”作为二进制标识。例如，要输入“abc”，则应输入“0Xabc”。若输入的数据位数为奇数，则系统会自动在起始符号“0X”的后面添加一个 0。如上述输入“0Xabc”后，系统会自动变为“0X0abc”。

（2）varbinary 型是可变长度的二进制数据类型，其定义形式为 varbinary(n)，其中 n 表示数据的长度，取值为 1～8000。如果输入的数据长度超出 n 的范围，则系统会自动截掉超出部分。

varbinary 型具有变动长度的特性，因为 varbinary 型数据的存储长度为实际数值长度+4 个字节。当 binnary 型数据允许 null 值时，将被视为 varbinary 型的数据。

一般情况下，由于 binary 型的数据长度固定，因此它比 varbinary 型的数据处理速度快。

（3）image 型的数据也是可变长度的二进制数据，其最大长度为 2^{31}-1 个字节。

4. 逻辑数据类型

逻辑数据类型只有一种，即 bit 型。bit 数据类型只占用 1 个字节的存储空间，其值为 0 和 1。只要输入的值为非 0，系统都会当作 1 处理。另外 bit 型不能定义为 null 值。

5. 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、特殊符号等。一般情况下，使用字符类型数据时，必须在数据的前后加上单引号或双引号。字符数据类型包括 char 型、nchar 型、varchar 型和 nvarchar 型。

（1）char 型是固定长度的非 Unicode 字符数据类型，在存储时每个字符和符号占用一个字节的存储空间。其定义形式为 char[(n)]，其中 n 表示所有字符所占的存储空间，取值为 1～8000，即可容纳 8000 个 ANSI 字符，默认值为 1。若输入的数据字符数小于 n 定义的范围，则系统自动在其后添加空格来填满设定好的空间；若输入的数据字符数超过 n 定义的范围，则系统自动截掉超出部分。

（2）nchar 型是固定长度的 Unicode 字符数据类型，由于 Unicode 标准规定在存储时每个字符和符号占用 2 个字节的存储空间，因此 nchar 型的数据比 char 型的数据多占用一倍的存储空间。其定义形式为 nchar[(n)]，其中 n 表示所有字符所占的存储空间，取值为 1～4000，即可容纳 4000 个

Unicode 字符，默认值为 1。

使用 Unicode 标准字符集的好处是由于它使用两个字节作存储单位，使得一个存储单位的容量大大增加，这样就可以将全世界的语言文字都囊括在内。当用户在一个数据列中同时输入不同语言的文字符号时，系统不会出现编码冲突。

（3）varchar 型是可变长度的非 Unicode 字符数据类型。其定义形式为 varchar[(n)]。它与 char 型类似，n 的取值范围是 1～8000。由于 varchar 型具有可变长度的特性，所以 varchar 型数据的存储长度为实际数值的长度。如果输入数据的字符数小于 n 定义的长度，系统也不会像 char 型那样在数据后面用空格填充；但是如果输入的数据长度大于 n 定义的长度，系统会自动截掉超出部分。

一般情况下，由于 char 型的数据长度固定，因此它比 varchar 型数据的处理速度快。

（4）nvarchar 型是可变长度的 Unicode 字符数据类型，其定义形式为 nvarchar[(n)]。由于它采用了 Unicode 标准字符集，因此 n 的取值范围是 1～4000。nvarchar 型的其他特性与 varchar 类型相似。

6. 文本和图形数据类型

文本和图形数据类型是用于存储大量的非 Unicode 和 Unicode 字符以及二进制数据的固定长度和可变长度数据类型，包括 text 型和 ntext 型。

（1）text 型是用于存储大量非 Unicode 文本数据的可变长度数据类型，其容量理论上为 2^{31}-1 个字节。在实际应用时需要视硬盘的存储空间而定。

（2）ntext 型是用于存储大量 Unicode 文本数据的可变长度数据类型，其理论容量为 2^{30}-1 个字节。ntext 型的其他用法与 text 型基本一样。

7. 日期和时间数据类型

日期和时间数据类型代表日期和一天内的时间，包括 datetime 型和 smalldatetime 型。

（1）datetime 型是用于存储日期和时间的结合体的数据类型。它可以存储从公元 1753 年 1 月 1 日零时起到公元 9999 年 12 月 31 日 23 时 59 分 59 秒之间的所有日期和时间，其精确度可达 3%秒。

datetime 型数据所占用的存储空间为 8 个字节，其中前 4 个字节用于存储 1900 年 1 月 1 日以前或以后的天数，数值分正负，正数表示在此日期之后的日期，负数表示在此日期之前的日期；后 4 个字节用于存储从此日零时起所指定的时间经过的毫秒数。如果在输入时省略了时间部分，则系统将默认为 12:00:00:000AM；如果省略了日期部分，系统将默认为 1900 年 1 月 1 日。

（2）smalldatetime 型与 datetime 型相似，但其存储的日期时间范围较小，从 1900 年 1 月 1 日到 2079 年 6 月 6 日。它的精度也较低，只能精确到分钟级，其分钟个位上的值是根据秒数并以 30 秒为界四舍五入得到的。

smalldatetime 型数据所占用的存储空间为 4 个字节，其中前两个字节存储从基础日期 1900 年 1 月 1 日以来的天数，后两个字节存储此日零时起所指定的时间经过的分钟数。

8. 货币数据类型

货币数据类型用于存储货币或现金值，包括 money 型和 smallmoney 型。在使用货币数据类型时，应在数据前加上货币符号，以便系统辨识其为哪国的货币，如果不加货币符号，则系统默认为“￥”。

（1）money 型是一个有 4 位小数的 decimal 值，取值范围为-2^{63} ～2^{63}-1，精确到货币单位的千分之十。存储大小为 8 个字节。

（2）smallmoney 型货币数据值介于-2147483648～+2147483647 之间，精确到货币单位的千分之十。存储大小为 4 个字节。

5.1.4　常量与变量

1. 变量

变量对于一种语言来说是必不可少的组成部分。Transact-SQL 语言允许使用两种变量：一种是用户自己定义的局部变量，另一种是系统提供的全局变量。

（1）局部变量。局部变量是用户自己定义的变量，它的作用范围在程序内部。通常只能在一个批处理中或存储过程中使用，用来存储从表中查询到的数据，或当作程序执行过程中暂存变量使用。局部变量使用 DECLARE 语句定义，并且指定变量的数据类型，然后可以使用 SET 或 SELECT 语句为变量初始化；局部变量必须以@开头，而且必须先声明后使用。声明格式如下：

```
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 支持的所有数据类型，也可以是用户自定义的数据类型。

局部变量不能使用“变量=变量值”的格式进行初始化，必须使用 SELECT 或 SET 语句来设置其初始值。初始化格式如下：

```
SELECT @局部变量=变量值
SET @局部变量=变量值
```

【例 5.3】在 stu 数据库中使用名为@find 的局部变量检索所有姓以“李”开头的学生信息，代码如下：

```
USE stu
DECLARE @find varchar(30)
SET @find = '李%'
SELECT stuname, stusex, stuage
FROM student
WHERE stuname LIKE @find
```

注意：如果声明字符型的局部变量，一定要在变量类型中指明其最大长度，否则系统认为其长度为 1。

（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 的配置设置值和统计数据。用户可在程序中用全局变量来测试系统的设定值或者为 Transact-SQL 命令执行后的状态值。引用全局变量时，全局变量的名字前面要有两个标记符“@@”。不能定义与全局变量同名的局部变量。全局变量的符号及其功能如表 5.1 所示。

表 5.1　全局变量及其功能

全局变量	功能
@@CONNECTIONS	自 SQL Server 最近一次启动以来登录或试图登录的次数
@@CPU_BUSY	自 SQL Server 最近一次启动以来 CPU 的工作时间
@@CURRSOR_ROWS	返回在本次连接最新打开的游标中的行数
@@DATEFIRST	返回 SET DATEFIRST 参数的当前值
@@DBTS	数据库的唯一时间标记值
@@ERROR	系统生成的最后一个错误，若为 0 则成功
@@FETCH_STATUS	最近一条 FETCH 语句的标志
@@IDENTITY	保存最近一次的插入身份值

续表

全局变量	功能
@@IDLE	自 CPU 服务器最近一次启动以来的累计空闲时间
@@IO_BUSY	服务器输入输出操作的累计时间
@@LANGID	当前使用的语言的 ID
@@LANGUAGE	当前使用的语言的名称
@@LOCK_TIMEOUT	返回当前锁的超时设置
@@MAX_CONNECTIONS	同时与 SQL Server 相连的最大连接数量
@@MAX_PRECISION	十进制与数据类型的精度级别
@@NESTLEVEL	当前调用存储过程的嵌套级，范围为 0～16
@@OPTIONS	返回当前 SET 选项的信息
@@PACK_RECEIVED	所读的输入包数量
@@PACKET_SENT	所写的输出包数量
@@PACKET_ERRORS	读与写数据包的错误数
@@RPOCID	当前存储过程的 ID
@@REMSERVER	返回远程数据库的名称
@@ROWCOUNT	最近一次查询涉及的行数
@@SERVERNAME	本地服务器名称
@@SERVICENAME	当前运行的服务器名称
@@SPID	当前进程的 ID
@@TEXTSIZE	当前最大的文本或图像数据大小

SQL Server 提供的全局变量共有 33 个，但并不是每一个都经常用到。下面举例说明全局变量的使用方法。

【例 5.4】@@ERROR 的使用举例。

@@ERROR 变量用于表示所执行的 T-SQL 语句是否成功，如果成功，则返回整数 0；否则返回一个表示错误类型的错误码，此错误码为整数。

程序如下：

```
USE stu
GO
UPDATE student SET stuage=26 WHERE stuname='赵刚'
IF @@ERROR=0
   PRINT '更新数据成功'
ELSE
   PRINT '更新数据失败'
```

此程序的功能是要将 student 表中姓名为“赵刚”的学生的年龄信息修改为 26，并判断是否成功。如果成功，则显示“更新数据成功”的提示；如果不成功，则显示“更新数据失败”的提示。

2. 常量

常量，也称为文字值或标量值，是表示一个特定数据值的符号。常量的格式取决于它所表示的值的数据类型。比如字符串常量的表示就是指在单引号内包含字母数字字符（a～z、A～Z 和 0～9）

和特殊字符（如感叹号（!）、at 符（@）和数字号（#））等的字符序列。其他类型的常量的表示与其他语言常量的表示方式大体相当，在这里不再说明，读者可以参照 Transact-SQL 语言参考手册加以学习。

5.1.5　注释符与运算符

1. 注释

注释是程序代码中不执行的文本字符串，也称为注解。在 SQL Server 中，可以使用两种类型的注释字符：

（1）ANSI 标准的注释符“--”，用于单行注释。

（2）与 C 语言相同的程序注释符，即“/*……*/”，/*用于注释文字的开头，*/用于注释文字的结尾，可在程序中标识多行文字为注释。

```
--下面声明变量 --单行注释
/* 多行注释
@Snum VARCHAR(4)：存放学号
@Sname VARCHAR(8)：存放姓名
*/
```

2. 运算符

运算符是一种符号，用来指定要在一个或多个表达式中指定的操作。SQL Server 中使用如下几种运算符：算术运算符、赋值运算符、位运算符、比较运算符、逻辑运算符、字符串串联运算符和一元运算符。

（1）算术运算符。算术运算符用来在两个表达式上执行数学运算，这两个表达式可以是任意两个数字数据类型的表达式。算术运算符包括+（加）、-（减）、*（乘）、/（除）、%（模）。

在 Transact-SQL 中，“+”包含三个方面的意义：表示正号，即在数值前添加“+”号表示该数值是一个正数；表示算术运算的加号，能将数值类型的两个数据相加；连接两个字符型或 binary 型的数据，这时的“+”号叫做字符串串联运算符。

（2）赋值运算符。Transact-SQL 有一个赋值运算符，即等号（=）。

【例 5.5】赋值运算符。

```
DECLARE @MyCounter INT
SET @MyCounter = 1
```

本例中的代码创建了@MyCounter 变量，然后赋值运算符将@MyCounter 设置成一个由表达式返回的值。

（3）位运算符。位运算符在两个表达式之间执行位操作，这两个表达式可以是任意两个整型数据类型的表达式。位运算符的符号及其含义如表 5.2 所示。

表 5.2　位运算符

运算符	含义
&	按位与（两个操作数）
|	按位或（两个操作数）
^	按位异或（两个操作数）
~	按位取反（一个操作数）

位运算符的操作数可以是整型或二进制字符串数据类型中的任何数据类型（但 image 数据类型除外），此外，两个操作数不能同时是二进制字符串数据类型中的某种数据类型。

（4）比较运算符。比较运算符用来测试两个表达式是否相同。除了 text、ntext 或 image 数据类型的表达式外，比较运算符可以用于所有的表达式。比较运算符的符号及其含义如表 5.3 所示。

表 5.3 比较运算符

运算符	含义
=	等于
>	大于
<	小于
>=	大于等于
<=	小于等于
<>	不等于

比较运算符的结果是布尔数据类型，它有三种值：TRUE、FALSE 和 NULL。那些返回布尔数据类型的表达式被称为布尔表达式。

（5）逻辑运算符。逻辑运算符用来对某个条件进行测试，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 值的布尔数据类型。逻辑运算符的符号及其含义如表 5.4 所示。

表 5.4 逻辑运算符

运算符	含义
ALL	如果一系列的比较都为 TRUE，那么就为 TRUE
AND	如果两个布尔表达式都为 TRUE，那么就为 TRUE
ANY	如果一系列的比较中任何一个为 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	如果两个布尔表达式中的一个为 TRUE，那么就为 TRUE
SOME	如果在一系列比较中，有些为 TRUE，那么就为 TRUE

5.1.6 标准（库）函数

在 Transact-SQL 语言中，函数被用来执行一些特殊的运算以支持 SQL Server 的标准命令。SQL Server 包含多种不同的函数用以完成各种工作，每一个函数都有一个名称，在名称之后有一对小括号，如 gettime( )。大部分的函数在小括号中需要一个或者多个参数。

Transact-SQL 编程语言提供了 4 种函数：行集函数、聚合函数、Ranking 函数、标量函数。

1. 行集函数

行集函数可以在 Transact-SQL 语句中当作表引用。

【例 5.6】通过行集函数 OPENQUERY()执行一个分布式查询，以便从服务器 local 中提取表 student 中的记录。

```
sp_addlinkedserver 'local',N'SQL Server'                    --创建服务器
SELECT * from openquery(local,'select * from student')   --执行 local 服务器上的查询
```

2. 聚合函数

聚合函数用于对一组值进行计算并返回一个单一的值。聚合函数经常与 select 语句的 group by 子句一同使用。常用的聚合函数主要有 sum（求和函数）、avg（求平均值函数）、max（求最大值函数）、min（求最小值函数）、count（求记录数函数），这些函数的参数是要求值的字段名称，除 count 函数之外，其他函数只能应用于数值型字段。

【例 5.7】利用聚合函数计算表中的记录总数。

```
SELECT count(*) FROM student
```

3. ranking 函数

ranking 函数为查询结果数据集分区中的每一行返回一个序列值。依据此函数，一些行可能取得和其他行一样的序列值。Transact-SQL 提供的 ranking 函数有：rank、dense_rank、ntile、row_number。对于 ranking 函数的含义请读者参阅 SQL 手册，在这里不再赘述，只举例说明此类函数的用法。

【例 5.8】rank 函数可以对指定字段进行分区和排序，并返回排序顺序的编号。下面语句实现对 student 表的数据按年龄排序。

```
select rank() over(order by stuage desc) as stuagenum,stuname from student
```

程序结果显示了对表中数据按年龄进行降序排序的情况，并显示出了序号。

4. 标量函数

标量函数用于对传递给它的一个或者多个参数值进行处理和计算，并返回一个单一的值。标量函数可以应用在任何一个有效的表达式中。标量函数可分为配置函数、游标函数、日期和时间函数、数学函数等。对于标量函数的具体应用方法请读者参阅 SQL 手册，在这里不再赘述，只举例说明此类函数的基本用法。

【例 5.9】返回当前日期的月、日和年可以使用日期函数来实现。

```
DECLARE @today datetime
SET @today=getdate()       --getdate 获取当前日期
SELECT month(@today),day(@today),year(@today)      --month、day、year 三个函数获取日期数据
                                                    --的月、日和年
```

5.1.7　流程控制语句

Transact-SQL 语言提供了一些可以用于改变语句执行顺序的命令，称为流程控制语句。在 SQL Server 中，流程控制语句主要用来控制 SQL 语句、语句块、存储过程的执行流程。Transact-SQL 语言使用的流程控制命令与常见的程序设计语言类似，主要有以下几种控制命令：

（1）BEGIN…END 语句。BEGIN…END 语句能够将多个 Transact-SQL 语句组合成一个语句块，并将在 BEGIN…END 内的所有程序视为一个单元处理。在条件语句（如 IF…ELSE）和循环等控制流程语句中，当符合特定条件便要执行两个或者多个语句时，就需要使用 BEGIN…END 语句，其语法形式为：

```
BEGIN
  <命令行或程序块>
END
```

在 BEGIN…END 中可以嵌套另外的 BEGIN…END 来定义另一程序块。

【例 5.10】说明 BEGIN…END 语句的用法。

```
DECLARE @message varchar(50)
BEGIN
 SET @message='欢迎使用 SQL Server 2008'
 PRINT @message
END
```

（2）IF…ELSE 语句。IF…ELSE 语句是条件判断语句，用来判断当某一条件成立时执行某段程序，条件不成立时执行另一段程序。其语法如下：

```
IF <条件表达式>
  <命令行或程序块>
[ELSE [条件表达式]
  <命令行或程序块>]
```

其中，<条件表达式>可以是各种表达式的组合，但表达式的值必须是逻辑值“真”或“假”；ELSE 子句是可选的，最简单的 IF 语句没有 ELSE 子句部分。

如果不使用程序块，则 IF 或 ELSE 只能执行一条命令。

IF…ELSE 可以进行嵌套，在 Transact-SQL 中最多可嵌套 32 级。

【例 5.11】从 SC（选修表）中求出学号为 S003 的学生的平均成绩，如果此平均成绩大于或等于 90 分，则输出“优秀”字样。

```
IF (SELECT avg(Score) FROM SC WHERE Snum=' S003' group by Snum) >= 90
  BEGIN
      PRINT '优秀'
  END
```

（3）CASE 语句。CASE 命令有两种语句格式：

1）CASE <表达式>

```
WHEN <表达式> THEN <表达式>
…
WHEN <表达式> THEN <表达式>
[ELSE <表达式>]
END
```

该语句的执行过程是：将 CASE 后面表达式的值与各 WHEN 子句中的表达式的值进行比较，如果二者相等，则返回 THEN 后的表达式的值，然后跳出 CASE 语句；否则返回 ELSE 子句中的表达式的值。

ELSE 子句是可选项。当 CASE 语句中不包含 ELSE 子句时，如果所有比较失败时，CASE 语句将返回 NULL。

【例 5.12】从学生表 S 中选取 Snum、Ssex，如果 Ssex 为“男”显示“M”，如果为“女”显示“F”。

```
SELECT Snum,
      CASE Ssex
      WHEN '男' THEN 'M'
      WHEN '女' THEN 'F'
      END AS "性别"
FROM S
```

2）CASE

```
WHEN <条件表达式> THEN <表达式>
```

```
…
WHEN <条件表达式> THEN <表达式>
[ELSE <表达式>]
END
```

该语句的执行过程是：首先测试 WHEN 后的表达式的值。如果其值为真，则返回 THEN 后面的表达式的值，否则测试下一个 WHEN 子句中的表达式的值。如果所有 WHEN 子句后的表达式的值都为假，则返回 ELSE 后的表达式的值。如果在 CASE 语句中没有 ELSE 子句，则 CASE 表达式返回 NULL。

CASE 命令可以嵌套到 SQL 命令中。

【例 5.13】从 SC 表中查询所有学生选课成绩情况，将百分制转换为五分制：凡成绩为空者显示“未考”；小于 60 分显示“不及格”；60～70 分显示“及格”；70～80 分显示“中等”；80～90 分显示“良好”；大于或等于 90 分时显示“优秀”。

```
SELECT Snum, Cnum,
    CASE
      WHEN Score IS NULL THEN '未考'
      WHEN Score <60 THEN '不及格'
      WHEN Score >=60 AND SCORE<70 THEN '及格'
      WHEN Score >=70 AND SCORE<80 THEN '中等'
      WHEN Score >=80 AND SCORE<90 THEN '良好'
      WHEN Score >=90 THEN '优秀'
    END AS "等级"
FROM SC
```

（4）WHILE…CONTINUE…BREAK 语句。WHILE…CONTINUE…BREAK 语句用于设置重复执行 SQL 语句或语句块的条件。WHILE 命令在设定的条件成立时，会重复执行命令行或程序块。CONTINUE 命令可以让程序跳过 CONTINUE 命令之后的语句，回到 WHILE 循环的第一行，继续进行下一次循环。BREAK 语句则使程序完全跳出循环，结束 WHILE 语句的执行。WHILE 语句也可以嵌套。其语法如下：

```
WHILE <条件表达式>
BEGIN
  <命令行或程序块>
  [BREAK]
  [CONTINUE]
  [命令行或程序块]
END
```

【例 5.14】打印学生表 S 中姓“岳”的学生的学号、姓名。

```
DECLARE
@Snum VARCHAR(4)            --存放学号
@Sname VARCHAR(8)           --存放姓名
--声明游标
DECLARE CurStudent CURSOR
FOR SELECT Snum, Sname FROM S WHERE Sname like '岳%' ORDER BY Snum
--打开游标
OPEN CurStudent
--返回第一条记录并将各个字段存入变量
FETCH CurStudent
```

```
INTO @Snum,@Sname
WHILE   @@FETCH_STATUS = 0          --有记录时循环处理
BEGIN
PRINT @Snum
PRINT @Sname
--返回下一条记录
FETCH CurStudent
      INTO @Snum,
           @Sname
END
--关闭游标
CLOSE CurStudent
DEALLOCATE CurStudent
```

（5）GOTO 语句。GOTO 命令用来改变程序执行的流程，使程序跳到标有标识符的指定的程序行再继续往下执行，而位于 GOTO 语句和标识符之间的程序将不会被执行。GOTO 语句和标识符可以用在语句块、批处理和存储过程中。作为跳转目标的标识符可为数字与字符的组合，但必须以“:”结尾。在 GOTO 命令行，标识符后不必跟“:”，语法如下：

```
GOTO 标识符
```

（6）RETURN 语句。RETURN 命令用于无条件地结束当前程序的执行，此时位于该语句后的程序将不会被执行，返回到上一个调用它的程序或其他程序。语法如下：

```
RETURN([整数值])
```

在括号内可以指定一个返回值。如果没有指定返回值，SQL Server 系统会根据程序执行的结果返回一个内定值，如表 5.5 所示。如果运行过程中产生了多个错误，SQL Server 系统将返回绝对值最大的数值；如果此时用户定义了返回值，则可以返回用户定义的值。RETURN 语句不能返回 NULL 值。

表 5.5 Transact-SQL 返回值

返回值	含义	返回值	含义
0	程序执行成功	-8	非致命的内部错误
-1	找不到对象	-9	已达到系统的极限
-2	数据类型错误	-10	致命的内部不一致性错误
-3	死锁	-11	致命的内部不一致性错误
-4	违反权限原则	-12	表或指针破坏
-5	语法错误	-13	数据库破坏
-6	用户造成的一般错误	-14	硬件错误
-7	资源错误，如磁盘空间不足		

（7）WAITFOR 语句。WAITFOR 语句，称为延迟语句，设定在达到指定时间或时间间隔之前或者指定语句至少修改或返回一行之前，阻止执行批处理、存储过程或事务。其语法格式为：

```
WAITFOR
{
    DELAY 'time_to_pass'          /* 设定等待时间 */
  | TIME 'time_to_execute'        /* 设定等待某一时刻 */
}
```

备注：执行 WAITFOR 语句时，事务正在运行，并且其他请求不能在同一事务下运行。WAITFOR 不更改查询的语义。如果查询不能返回任何行，WAITFOR 将一直等待或等到满足 TIMEOUT 条件（如果已指定）。

【例 5.15】延迟 30 秒执行查询。

```
USE Library
GO
WAITFOR DELAY '00:00:30'
SELECT   * FROM   Reader
```

【例 5.16】在时刻 21:20:00 执行查询。

```
USE Library
GO
WAITFOR TIME '21:20:00'
SELECT   * FROM   Reader
```

【例 5.17】延迟 20 秒执行。

```
DECLARE @COUNT INT
SET @COUNT=0
WHILE @COUNT<10
BEGIN
UPDATE 订单
SET 客户编号
WHERE 订单编号=21
IF @@ERROR=0 BREAK
SET @COUNT=@COUNT+1
WAITFOR DELAY '00:00:20'
END
```

（8）TRY…CATCH 语句。TRY…CATCH 语句类似于 C#或 C++语句中的异常处理，当执行 TRY 语法块中的代码出现错误时，系统将会把控制传递到 CATCH 语法块中去处理。其语法代码为：

```
BEGIN TRY
{ SQL_STATEMENT|STATEMENT_BLOCK}
END TRY
BEGIN CATCH
{ SQL_STATEMENT|STATEMENT_BLOCK}
END CATCH
```

【例 5.18】删除学生表中学生号等于 5 的学生信息。

```
BEGIN TRY
DELETE   S WHERE SNUM=5
END TRY
BEGIN CATCH
PRINT '出错信息为'+ ERROR_MESSAGE()
DELETE SC WHERE SNUM=5
DELETE S WHERE SNUM=5
END CATCH
```

5.1.8　常用命令

（1）BACKUP。BACKUP 命令用于将数据库内容或其事务处理日志备份到存储介质（软盘、

硬盘、磁带等）上。

（2）CHECKPOINT。CHECKPOINT 命令用于将当前工作的数据库中被更改过的数据页或日志页从数据缓冲器中强制写入硬盘。

（3）DBCC。DBCC（DataBase Consistency Checker，数据库一致性检查程序）命令用于验证数据库完整性、查找错误、分析系统使用情况等。DBCC 命令后必须加上子命令，系统才知道要做什么。例如，DBCC CHECKALLOC 命令检查目前数据库内所有数据页的分配和使用情况。

（4）USE。USE 命令用于改变当前使用的数据库为指定的数据库，语法如下：

```
USE {databasename}
```

用户必须是目标数据库的用户成员或目标数据库建有 GUEST 用户账号时，使用 USE 命令才能成功切换到目标数据库。

（5）EXECUTE。EXECUTE 命令用来执行存储过程。

（6）KILL。KILL 命令用于终止某一过程的执行。

（7）PRINT。PRINT 命令向客户端返回一个用户自定义的信息，即显示一个字符串、局部变量或全局变量。如果变量值不是字符串，则必须先用数据类型转换函数 CONVERT()将其转换为字符串。

（8）RAISERROR。RAISERROR 命令用于在 SQL Server 系统返回错误信息时同时返回用户指定的信息。

（9）READTEXT。READTEXT 命令用于从数据类型为 text、ntext 或 image 的列中读取数据，语法如下：

```
READTEXT {table.column text_pointer offset size} [HOLDLOCK]
```

命令从偏移位置 offset+1 个字符起读取 size 个字符，如果 size 为 0，则会读取 4KB 的数据。其中，text_pointer 是指向存储文本的第一个数据库页的指针，它可以用 TEXTPTR(9)函数来获取；HOLDLOCK 选项用于锁定所读取的数据直到传输结束，这段时间内其他用户只能读取数据不能更改数据。如果数据列为汉字，则 offset 值应取 0 或其他偶数，如果用奇数则会出现乱码。

（10）RESTORE。RESTORE 命令用来将数据库或其事务处理日志备份文件由存储介质回存到 SQL Server 系统中。

（11）SHUTDOWN。SHUTDOWN 命令用于停止 SQL Server 的执行，语法如下：

```
SHUTDOWN [WITH NOWAIT]
```

当使用 WITH NOWAIT 选项时，SHUTDOWN 命令立即停止 SQL Server，在终止所有的用户进程并对每一现行的事务发生一个回滚后，退出 SQL Server。

当没有用 NOWAIT 参数时，SHUTDOWN 命令将按以下步骤执行：

1）终止任何用户登录 SQL Server。

2）等待尚未完成的 Transact-SQL 命令或存储过程执行完毕。

3）在每个数据库中执行 CHECKPOINT 命令。

4）停止 SQL Server 的执行。

5.2 存储过程

在 SQL Server 2008 数据库系统中，存储过程具有很重要的作用，它是 SQL 语句和流程控制语句的集合。存储过程在运算时生成执行方式，所以以后对其再运行时其执行速度很快。SQL Server

2008 不仅提供了用户自定义存储过程的功能，而且也提供了许多可作为工具使用的系统存储过程。

5.2.1　存储过程的概念

存储过程（Stored Procedure）是一组编译好存储在服务器上的完成特定功能的 T-SQL 代码，是某数据库的对象。客户端应用程序可以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）来执行存储过程。

5.2.2　存储过程的优点

当利用 SQL Server 创建一个应用程序时，Transact-SQL 是一种主要的编程语言。若运用 Transact-SQL 来进行编程，有两种方法：一种是在本地存储 Transact-SQL 程序，并创建应用程序向 SQL Server 发送命令来对结果进行处理；另一种是可以把部分用 Transact-SQL 编写的程序作为存储过程存储在 SQL Server 中，并创建应用程序来调用存储过程，对数据结果进行处理，存储过程能够通过接收参数向调用者返回结果集，结果集的格式由调用者确定，返回状态值给调用者，指明调用是成功还是失败，包括针对数据库的操作语句，并且可以在一个存储过程中调用另一存储过程。

我们通常更偏爱于使用第二种方法。使用存储过程而不使用存储在客户端计算机本地的 T-SQL 程序的优点包括：

（1）存储过程允许标准组件式编程。存储过程在被创建以后可以在程序中被多次调用，而不必重新编写该存储过程的 SQL 语句。而且数据库专业人员可随时对存储过程进行修改，但对应用程序源代码毫无影响（因为应用程序源代码只包含存储过程的调用语句），从而极大地提高了程序的可移植性。

（2）存储过程能够实现较快的执行速度。如果某一操作包含大量的 Transact-SQL 代码或分别被多次执行，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 T-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。

（3）存储过程能够减少网络流量。对于同一个针对数据库对象的操作（如查询、修改），如果这一操作所涉及的 Transact-SQL 语句被组织成一个存储过程，那么当在客户计算机上调用该存储过程时，网络中传送的只是该调用语句，否则将是多条 SQL 语句，从而大大增加了网络流量，降低网络负载。

（4）存储过程可被作为一种安全机制来充分利用。系统管理员通过对执行某一存储过程的权限进行限制，从而能够实现对相应的数据访问权限的限制，避免非授权用户对数据的访问，保证数据的安全。

5.2.3　存储过程的分类

在 SQL Server 2008 中存储过程分为 5 类：

（1）系统存储过程：系统提供的存储过程，sp_*，例如 sp_rename。

（2）扩展存储过程：SQL Server 环境之外的动态链接库 DLL，xp_。

（3）用户存储过程：创建在用户数据库中的存储过程。

（4）CLR 存储过程：对 Microsoft .NET Framework 公共语言运行时 CLR 方法的调用，可以接受和返回用户提供的参数。

（5）临时存储过程：属于用户存储过程，#开头（局部：一个用户会话）和##开头（全局：所有用户会话）。

5.2.4 存储过程的建立

SQL Server 2008 创建存储过程有两种方法：一种是在 SSMS 中创建，另一种是使用 T-SQL 语句创建。

1. 在 SSMS 中创建存储过程

教务管理系统，创建一个查看班级名称、班级编号、班主任的存储过程，在 SSMS 中创建存储过程的步骤如下：

（1）启动 SSMS，连接到数据库实例，在“对象资源管理器”对话框中选择“数据库实例”→“数据库”→STUDENT→“可编程性”→“存储过程”并右击，在弹出的快捷菜单中选择“新建存储过程”选项，打开创建存储过程的查询编辑器模板，其中已经加入了一些创建存储过程的代码，如图 5.1 所示。

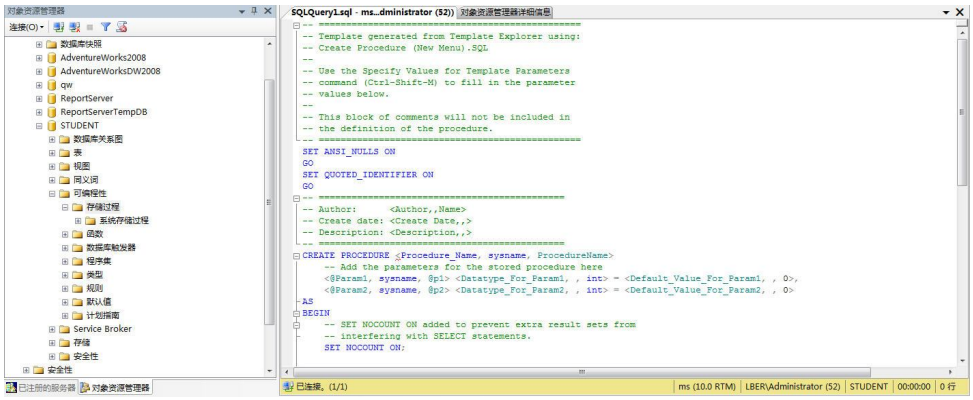

图 5.1 “对象资源管理器”对话框和创建存储过程的查询编辑模板

（2）选择“查询”→“指定模板参数的值”选项，弹出“指定模板参数的值”对话框，如图 5.2 所示。

图 5.2 “指定模板参数的值”对话框

（3）设置好相应的参数值，单击“确定”按钮，返回到创建存储过程的查询编辑窗口，此时内容已经改变。

2. 使用 T-SQL 创建存储过程

T-SQL 是通过 CREATE PROCEDURE 命令创建存储过程，其语法格式如下：

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
[ { @parameter [ type_schema_name. ] data_type }
[ VARYING ] [ = default ] [ [ OUTPUT ]
] [,...n ]
[ WITH <procedure_option> [,...n ]
[ FOR REPLICATION ]
AS { <sql_statement> [;][ ...n ] | <method_specifier> }
[;]
<procedure_option> ::=
[ ENCRYPTION ]
[ RECOMPILE ]
[ EXECUTE_AS_Clause ]
<sql_statement> ::=
{ [ BEGIN ] statements [ END ] }
<method_specifier> ::=
EXTERNAL NAME assembly_name.class_name.method_name
```

参数说明：

- schema_name：过程所属架构的名称。
- procedure_name：新存储过程的名称。
- number：用于对同名过程进行分组的可选整数。
- @ parameter：过程中的参数。
- [type_schema_name.] data_type：参数以及所属架构的数据类型。
- VARYING：指定作为输出参数支持的结果集。
- default：参数的默认值。如果定义了 default 值，则无须指定此参数的值即可执行过程。默认值必须是常量或 NULL。
- OUTPUT：指示参数是输出参数。此选项的值可以返回给调用 EXECUTE 的语句。
- RECOMPILE：指示数据库引擎不缓存该过程的计划，该过程在运行时编译。
- ENCRYPTION：指示 SQL Server 将 CREATE PROCEDURE 语句的原始文本转换为模糊格式。
- EXECUTE AS：指定在其中执行存储过程的安全上下文。
- 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。
- <sql_statement>：要包含在过程中的一个或多个 Transact-SQL 语句。
- EXTERNAL NAME,assembly_name.class_name.method_name：指定.NET Framework 程序集的方法，以便 CLR 存储过程引用。

【例 5.19】学生管理系统，创建一个查看班级名称、班级编号、班主任的存储过程。

```
CREATE PROCEDURE pro_t_class
AS
SELECT class_id,c_name,c_master from t_class
```

【例 5.20】学生管理系统，创建一个查看班级名称、班级编号、班主任的存储过程，其中班级编号作为参数。

```
CREATE PROCEDURE pro_t_class_2
```

```
@c_name varchar(10)
AS
SELECT class_id,c_name,c_master from t_class
WHERE c_name=@c_name
```

【例 5.21】学生管理系统，创建一个查看班级编号及其人数的存储过程，其中班级编号作为参数，人数作为输出参数。

```
CREATE PROCEDURE pro_stu_count
@class_id varchar(10),
@stu_sum numeric(4)output
AS
SELECT count(*) into@stu_sum from t_student where class_id=@class_id
```

5.2.5 执行存储过程

存储过程创建完后，要产生效果，必须执行，在 SQL Server 2008 中执行存储过程的方法有两种：一种是通过 EXEC 语句，另一种是在 SSMS 中执行。

1. 使用 EXEC 语句执行存储过程

T-SQL 语言中执行存储过程的语句是 EXEC，其语法格式如下：

```
EXECUTE a stored procedure or function
[ { EXEC | EXECUTE } ]
{
[ @return_status = ]
{ module_name [ ;number ] | @module_name_var }
[ [ @parameter = ] { value
| @variable [ OUTPUT ]
| [ DEFAULT ]
}
]
[,...n ]
[ WITH RECOMPILE ]
}
[;]
```

参数说明：

- @return_status：可选的整型变量，存储模块的返回状态。
- module_name：是要调用的存储过程或标量值用户定义函数的完全限定或者不完全限定名称。
- Number：是可选整数，用于对同名的过程分组。
- @module_name_var：是局部定义的变量名，代表模块名称。
- @parameter：module_name 的参数，与在模块中定义的相同。
- Value：传递给模块或传递命令的参数值。
- @variable：是用来存储参数或返回参数的变量。
- OUTPUT：指定模块或命令字符串返回一个参数。
- DEFAULT：根据模块的定义提供参数的默认值。
- WITH RECOMPILE：执行模块后，强制编译、使用和放弃新计划。

【例 5.22】执行例 5.19 创建无参数的存储过程 pro_t_class。

```
EXEC pro_t_class
```

【例 5.23】运行例 5.20 创建的带输入参数的存储过程 pro_t_class_2。

```
EXEC pro_t_class_2 'N0610'
```

【例 5.24】运行例 5.21 创建的带输出参数的存储过程 pro_stu_count。

```
DECLARE @stu_sum numeric(4)
EXEC pro_stu_count '4',@stu_sum output
```

2. 在 SSMS 中执行存储过程

除了使用 EXEC 语句执行存储过程外，使用 SSMS 也可以执行存储过程。在 SSMS 中执行存储过程 pro_t_class 的步骤如下：

（1）启动 SSMS，连接到数据库实例，在“对象资源管理器”对话框中选择“数据库实例”→“数据库”→STUDENT→“可编程性”→“存储过程”→pro_t_class 并右击，在弹出的快捷菜单中选择“执行存储过程”选项，弹出“执行过程”对话框，如图 5.3 所示。

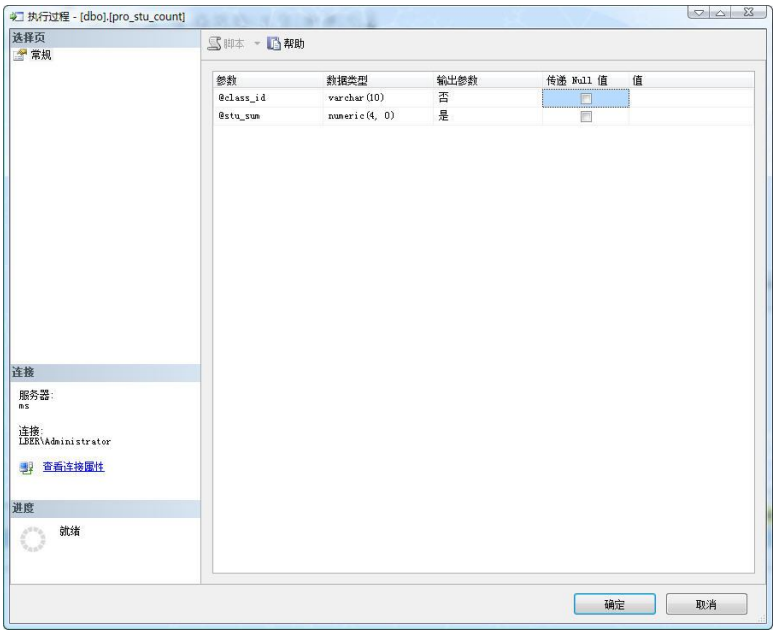

图 5.3　“执行过程”对话框

（2）单击“确定”按钮，执行存储过程。

5.2.6　修改存储过程

存储过程是一段 T-SQL 代码，在使用过程中，如果业务不满足需要，则要对存储过程进行修改，修改存储过程有两种方法，即 SSMS 和 T-SQL 方式。

1. 在 SSMS 中修改存储过程

学生管理系统 STUDENT 数据库，以存储过程 pro_stu_count 为例，修改该存储过程的步骤如下：启动 SSMS，连接到数据库实例，在“对象资源管理器”对话框中选择“数据库实例”→“数据库”→STUDENT→“可编程性”→“存储过程”→pro_stu_count 并右击，在弹出的快捷菜单中选择“修改”选项，打开“修改存储过程”模板，在代码区修改好存储过程，单击工具栏中的“执行”按钮，完成存储过程的修改，如图 5.4 所示。

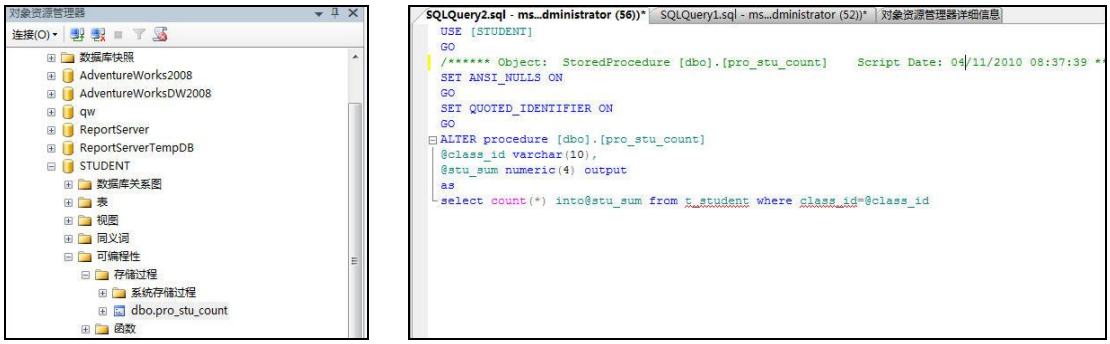

图 5.4 “对象资源管理器”和“修改存储过程”模板

2. 使用 T-SQL 修改存储过程

T-SQL 语言修改存储过程是使用 ALTER PROCEDURE 语句，其语法格式如下：

```
ALTER { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
[ { @parameter [ type_schema_name. ] data_type }
[ VARYING ] [ = default ] [OUTPUT ]
] [,...n ]
[ WITH <procedure_option> [,...n ] ]
[ FOR REPLICATION ]
AS
{ <sql_statement> [ ...n ] | <method_specifier> }
<procedure_option> ::=
[ ENCRYPTION ]
[ RECOMPILE ]
[ EXECUTE_AS_Clause ]
<sql_statement> ::=
{ [ BEGIN ] statements [ END ] }
<method_specifier> ::=
EXTERNAL NAME
assembly_name.class_name.method_name
```

其参数说明跟 CREATE PROCEDURE 的类似，这里不详细讲解了。

【例 5.25】修改例 5.19 创建的不带参数的存储过程 pro_t_class，增加按班级 ID 升级排序。

```
ALTER PROCEDURE pro_t_class
AS
SELECT class_id,c_name,c_master from t_class
ORDER BY class_id
```

5.2.7 删除存储过程

如果存储过程不再使用，则需要删除它，SQL Server 2008 提供了两种删除存储过程的方法：SSMS 和 T-SQL 方式。

1. 通过 SSMS 删除存储过程

在 SSMS 中删除存储过程的步骤如下：

（1）启动 SSMS，连接到数据库实例，在“对象资源管理器”对话框中选择“数据库实例”→“数据库”→STUDENT→“可编程性”→“存储过程”→pro_t_class 并右击，在弹出的快捷菜单中选择“删除”选项。

（2）弹出“删除对象”对话框，单击“确定”按钮，完成存储过程的删除，如图 5.5 所示。

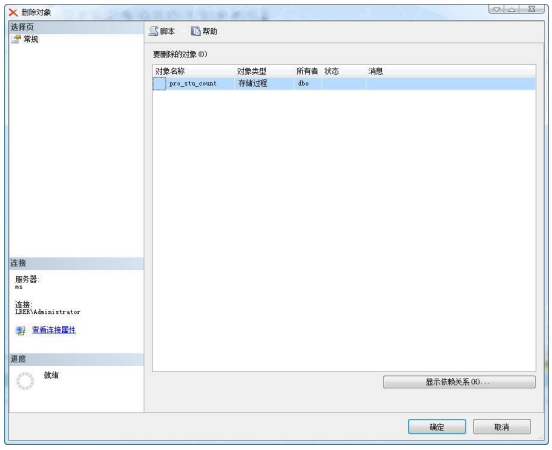

图 5.5　“删除存储过程”对话框

2. 使用 T-SQL 删除存储过程

T-SQL 语言删除存储过程的语法如下：

```
DROP { PROC | PROCEDURE } { [ schema_name. ] procedure } [,...n ]
```

参数说明：

- schema_name：过程所属架构的名称，不能指定服务器名称或数据库名称。
- procedure：要删除的存储过程或存储过程组的名称。

【例 5.26】删除存储过程 pro_t_class。

```
DROP PROCEDURE pro_t_class
```

5.3　触发器

5.3.1　触发器的概念

触发器是一类特殊的存储过程，它与表紧密相连，在对特定表或视图发出 UPDATE、INSERT 或 DELETE 语句时自动执行。

Microsoft SQL Server 2008 提供了两种主要机制来强制执行业务规则和数据完整性：约束和触发器。

注意：触发器不需要调用，也不传递参数或接受参数，当对一个表的特殊事件出现时，它就会被激活，主要作用就是实现由主键和外键所不能保证的复杂的参照完整性和数据一致性。

5.3.2　触发器的作用

（1）完成比约束更复杂的数据约束。触发器可以实现比约束更为复杂的数据约束。

（2）检查所做的 SQL 是否允许。触发器可以检查 SQL 所做的操作是否被允许，例如，在产品库存表里，如果要删除一条产品记录，在删除记录时，触发器可以检查该产品库存数量是否为零，如果不为零则取消该删除操作。

（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时，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，一个订单取消的时候，触发器可以自动修改产品库存表，在订购量的字段上减去被取消订单的订购数量。

（4）调用更多的存储过程。约束的本身是不能调用存储过程的，但是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）发送 SQL Mail。在 SQL 语句执行完成后，触发器可以判断更改过的记录是否达到一定条件，如果达到这个条件，触发器可以自动调用 SQL Mail 来发送邮件。例如，当一个订单交费之后，可以向物流人员发送 Email，通知他尽快发货。

（6）返回自定义的错误信息。约束是不能返回信息的，而触发器可以。例如插入一条重复记录时，可以返回一个具体的友好的错误信息给前台应用程序。

（7）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，例如原本的 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，那么触发器可以不执行该语句。

（8）防止数据表结构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 DROP 和 ALTER 开头的 SQL 语句里不进行对数据表的操作。

5.3.3 触发器的种类

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（Data Manipulation Language）事件时执行的存储过程。DML 事件包括在指定表或视图中修改数据的 INSERT 语句、UPDATE 语句、DELETE 语句。DML 触发器可以查询其他表，还可以包含复杂的 T-SQL 语句。

2. DDL 触发器

当服务器或数据库中发生数据定义语言（DDL）事件时将调用这些触发器。但与 DML 触发器不同的是，它们不会为响应针对表或视图的 UPDATE、INSERT、DELETE 语句而激发，相反，它们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，如审核和规范数据库操作、防止数据库表结构被修改等。

5.3.4 DML 触发器的分类

（1）After 触发器：这类触发器是在记录已经改变完之后（after）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的操作，是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），而去执行触发器本身所定义的操作。

5.3.5 DML 触发器的工作原理

在 SQL Server 2008 里，为每个 DML 触发器都定义了两个特殊的表：一个是插入表（Inserted），一个是删除表（Deleted）。

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，这两个表也将会从内存中删除。

插入表（Inserted）里存放的是更新前的记录：对于插入记录操作来说，插入表里存放的是要

插入的数据；对于更新记录操作来说，插入表里存放的是要更新的记录。

删除表（Deldeted）里存放的是更新后的记录：对于更新记录操作来说，删除表里存放的是更新前的记录（更新完后即被删除）；对于删除记录操作来说，删除表里存放的是被删除的旧记录。

1. After 触发器的工作原理

After 触发器是在记录变更完成之后（如果是存储过程，而且有事务，则要在事务提交之后）才会被激活并执行的。以删除记录为例，分为以下步骤：

（1）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。

（2）把数据表里的记录删除。

（3）激活 After 触发器，执行 After 触发器里的 SQL 语句。

（4）触发器执行完毕之后，删除内存中的删除表，退出整个操作。

例如，在产品库存表里，如果要删除一条产品记录，在删除记录时，触发器可以检查该产品库存数量是否为零，如果不为零则取消删除操作。看一下数据库是怎么操作的：

（1）接收 SQL 语句，将要从产品库存表里删除的产品记录取出来，放在删除表里。

（2）从产品库存表里删除该产品记录。

（3）从删除表里读出该产品的库存数量字段，判断是否为零，如果为零，则完成操作，从内存里清除删除表；如果不为零，则用 Rollback Transaction 语句来回滚操作。

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。

5.3.6　设计 DML 触发器的注意事项及技巧

在了解触发器的种类和工作原理之后，现在可以开始动手来设计触发器了，不过在动手之前，还有一些注意事项必须先了解一下。

1. 设计触发器的限制

在触发器中，有一些 SQL 语句是不能使用的，这些语句如表 5.6 所示。

表 5.6　在 DML 触发器中不能使用的语句

不能使用的语句	语句功能
ALTER DATABASE	修改数据库
CREATE DATABASE	新建数据库
DROP DATABASE	删除数据库
LOAD DATABASE	导入数据库
LOAD LOG	导入日志
RECONFIGURE	更新配置选项
RESTORE DATABASE	还原数据库
RESTORE LOG	还原数据库日志

另外，在对作为触发操作的目标的表或视图使用了如表 5.7 所示的 SQL 语句时，不允许在 DML

触发器里再使用这些语句。

表 5.7　在目标表中使用过的 DML 触发器不能再使用的语句

不能使用的语句	语句功能
CREATE INDEX	建立索引
ALTER INDEX	修改索引
DROP INDEX	删除索引
DBCC DBREINDEX	重新生成索引
ALTER PARTITION FUNCTION	通过拆分或合并边界值更改分区
DROP TABLE	删除数据表
ALTER TABLE	修改数据表结构

2. 如何在触发器中取得字段修改前和修改后的数据

上面介绍过，SQL Server 2008 为每个触发器都定义了两个虚拟表：一个是插入表（Inserted），一个是删除表（Deleted），现在把这两个表存放的数据列表说明一下，如表 5.8 所示。

表 5.8　插入/删除表的功能

激活触发器的动作	Inserted 表	Deleted 表
INSERT	存放要插入的记录	
UPDATE	存放要更新的记录	存放更新前的旧记录
DELETE	存放要删除的旧记录	

【例 5.27】删除库存产品记录时，在删除时触发器要判断库存数量是否为零。

```
IF(SELECT 库存数量 FROM deleted)>0
BEGIN
PRINT  '库存数量大于 0 时不能删除此记录'
Rollback transaction
END
```

3. 使用 DML 触发器的注意事项

（1）After 触发器只能用于数据表中，Instead Of 触发器可以用于数据表和视图上，但两种触发器都不可以建立在临时表上。

（2）一个数据表可以有多个触发器，但是一个触发器只能对应一个表。

（3）在同一个数据表中，对每个操作（如 INSERT、UPDATE、DELETE）而言可以建立许多个 After 触发器，但 Instead Of 触发器针对每个操作只能建立一个。

（4）如果针对某个操作既设置了 After 触发器又设置了 Instead Of 触发器，那么 Instead of 触发器一定会激活，而 After 触发器就不一定会激活。

（5）Truncate Table 语句虽然类似于 Delete 语句可以删除记录，但是它不能激活 Delete 类型的触发器，因为 Truncate Table 语句是不记入日志的。

（6）WRITETEXT 语句不能触发 INSERT 和 UPDATE 型的触发器。

（7）不同的 SQL 语句可以触发同一个触发器，如 INSERT 和 UPDATE 语句都可以激活同一个触发器。

5.3.7　创建 DML 触发器

创建 DML 触发器有两种方法：一种是用 SSMS 创建触发器，一种是用 T-SQL 语言创建触发器。

1. 在 SSMS 中创建触发器

通过 SSMS 创建触发器的步骤如下：

（1）启动 SSMS，连接到数据库实例，在“对象资源管理器”对话框中选择“数据库实例”→“数据库”→NORTHWIND→“表”→Products→“触发器”并右击，在弹出的快捷菜单中选择“新建触发器”选项，如图 5.6 所示。

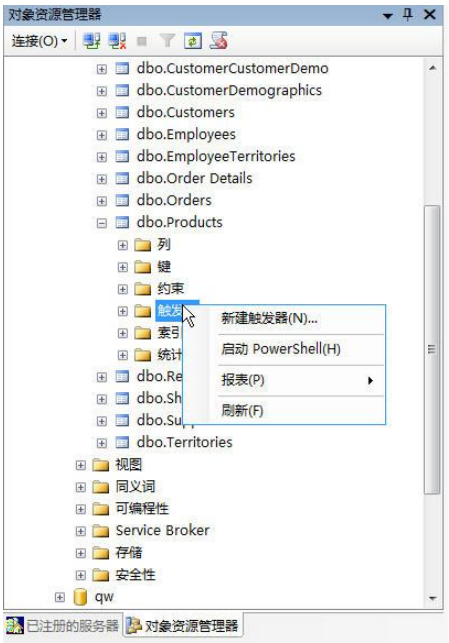

图 5.6　定位到触发器对话框

（2）打开“创建触发器”模板，如图 5.7 所示。

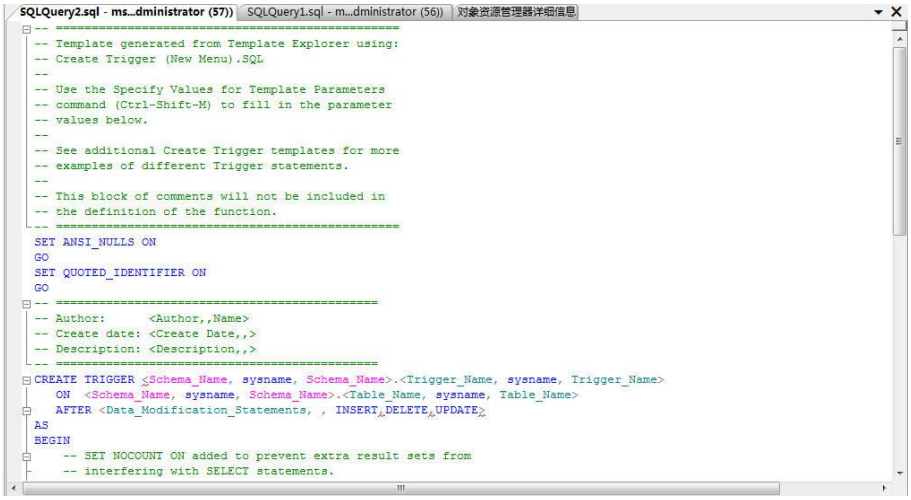

图 5.7　“创建触发器”模板

（3）在“创建触发器”模板中，修改其代码，或者是选择“查询”→“指定参数的模板”选项，弹出“指定模板参数的值”对话框，如图 5.8 所示。

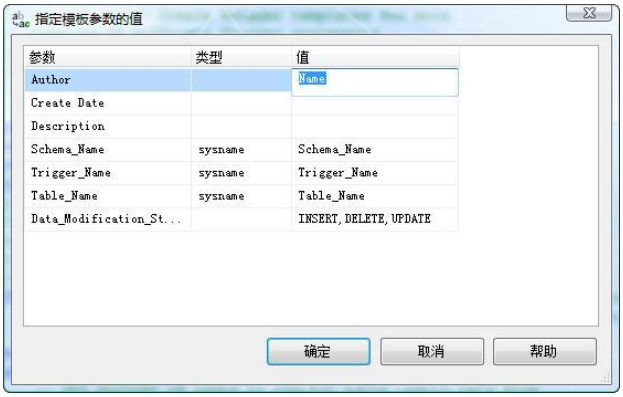

图 5.8 “指定模板参数的值”对话框

（4）指定模板参数后，在模板里修改其他代码，然后单击“运行”按钮，完成触发器的创建。

（5）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 产品_Insert
   ON products
   AFTER INSERT
AS
BEGIN
    PRINT '又添加了一种产品'
END
GO
```

（6）关掉查询编辑器模板，刷新一下触发器对话框，可以看到刚才建立的 procudt_Insert 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，不同的地方是把上面的 SQL 语句中的 AFTER INSERT 分别改为 AFTER UPDATE 和 AFTER DELETE 即可，如下所示，有兴趣的读者可以自行测试：

```
CREATE TRIGGER 产品_Update
   ON 产品
AFTER UPDATE
AS
BEGIN
   PRINT '有一种产品更改了'
END
GO
CREATE TRIGGER 产品_Delete
   ON 产品
AFTER DELETE
AS
BEGIN
   PRINT '又删除了一种产品'
END
GO
```

2. 使用 T-SQL 语言创建 DML 触发器

T-SQL 语言创建触发器是使用 CREATE TRIGGER 语句，其语法如下：

```
Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [,] [ UPDATE ] [,] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ...n ] | EXTERNAL NAME <method specifier [ ; ] > }
<dml_trigger_option> ::=
[ ENCRYPTION ]
[ EXECUTE AS Clause ]
<method_specifier> ::=
assembly_name.class_name.method_name
```

参数说明：

- schema_name：DML 触发器所属架构的名称。
- trigger_name：触发器的名称。
- table | view：对其执行 DML 触发器的表或视图，有时称为触发器表或触发器视图。
- WITH ENCRYPTION：对 CREATE TRIGGER 语句的文本进行加密。
- EXECUTE AS：指定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。
- INSTEAD OF：指定 DML 触发器是“代替”SQL 语句执行的，因此其优先级高于触发语句的操作。
- { [DELETE] [,] [INSERT] [,] [UPDATE] }：指定数据修改语句，这些语句可以在 DML 触发器对此表或视图进行尝试时激活该触发器。
- sql_statement：触发条件和操作。触发器条件指定其他标准，用于确定尝试的 DML 语句是否导致执行触发器操作。

用中文改了一下，以上代码就一目了然了：

```
CREATE TRIGGER  触发器名
ON  数据表名或视图名
AFTER INSERT 或 DELETE 或 UPDATE
AS
BEGIN
--这里是要运行的 SQL 语句
END
GO
```

【例 5.28】这是一个包含提醒电子邮件的触发器例子，如果订单表里记录有改动的话（无论增加订单还是修改、删除订单），则给物流人员张三发送电子邮件。

```
CREATE TRIGGER  订单_Insert
ON orders
AFTER INSERT, UPDATE, DELETE
AS
```

```
 EXEC master..xp_sendmail '张三',
 '订单有更改，请查询确定'
GO
```

【例 5.29】在订单明细表里，折扣字段不能大于 0.6，如果插入记录时折扣大于 0.6，则回滚操作。

```
CREATE TRIGGER 订单明细_Insert
   ON orderdetails
   AFTER INSERT
AS
BEGIN
  if (Select 折扣 from inserted)>0.6
 begin
      print '折扣不能大于 0.6'
    Rollback Transaction
 end
END
GO
```

5.3.8 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 服务器接到执行 SQL 语句请求之后，先建立临时的 Inserted 表和 Deleted 表，然后实际更改数据，最后才激活触发器的。而 Instead Of 触发器看起来就简单多了，在 SQL Server 服务器接到执行 SQL 语句请求后，先建立临时的 Inserted 表和 Deleted 表，然后就触发了 Instead Of 触发器，至于那个 SQL 语句是插入数据、更新数据还是删除数据，就一概不管了，把执行权全权交给了 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）数据库里的数据禁止修改。例如电信部门的通话记录是不能修改的，一旦修改，则通话费用的计数将不正确。在这个时候，就可以用 Instead Of 触发器来跳过 Update 修改记录的 SQL 语句。

（2）有可能要回滚修改的 SQL 语句。用 After 触发器并不是一个最好的方法，如果用 Instead Of 触发器，在判断折扣大于 0.6 时就终止了更新操作，避免在修改数据之后再回滚操作，减少服务器负担。

（3）在视图中使用触发器。因为 After 触发器不能在视图中使用，如果想在视图中使用触发器，则只能用 Instead Of 触发器。

（4）用自己的方式去修改数据。如果不满意 SQL 直接修改数据的方式，可以用 Instead Of 触发器来控制数据的修改方式和流程。

2. 建立 Instead Of 触发器

创建 Instead Of 触发器的语法如下：

```
CREATE TRIGGER 触发器名
ON 数据表名或视图名
Instead Of INSERT 或 DELETE 或 UPDATE
AS
```

```
BEGIN
--这里是要运行的 SQL 语句
END
GO
```

从上面可以看出，Instead Of 触发器与 After 触发器的语法几乎一致，只是简单地把 After 改为 Instead Of。用 After 触发器并不是一个最好的方法，如果用 Instead Of 触发器，在判断折扣大于 0.6 时就终止了更新操作，避免在修改数据之后再回滚操作，减少服务器负担。现将原来的触发器改为 Instead Of 触发器：

```
CREATE TRIGGER order_details_Insert
 ON order_details
 Instead Of INSERT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE
     @订单 ID int,
     @产品 ID int,
     @单价  money,
     @数量  smallint,
     @折扣  real
    SET @订单 ID = (select  订单 ID from inserted)
    SET @产品 ID = (select  产品 ID from inserted)
    SET @单价  = (select  单价  from inserted)
    SET @数量  = (select  数量  from inserted)
    SET @折扣  = (select  折扣  from inserted)
    IF (@折扣)>0.6
                PRINT '折扣不能大于 0.6'
    ELSE
INSERT INTO order_details(订单 ID,产品 ID,单价,数量,折扣)
VALUES(@订单 ID,@产品 ID,@单价,@数量,@折扣)
END
```

5.3.9　查看 DML 触发器

查看已经设计好的 DML 触发器有两种方式：一种是用 SSMS 来查看，一种是利用系统存储过程来查看。

1. 在 SSMSS 中查看触发器

在 Management Studio 中查看触发器的步骤如下：

（1）在对象资源管理器下选择数据库，定位到要查看触发器的数据表上，并找到触发器项，如图 5.9 所示。

（2）双击要查看的触发器名，Management Studio 自动弹出一个“查询编辑器”模板，其中显示的是该触发器的内容，如图 5.10 所示。

2. 用系统存储过程查看触发器

SQL Server 2008 里已经建好了两个系统存储过程，可以用这两个系统存储过程来查看触发器的情况。

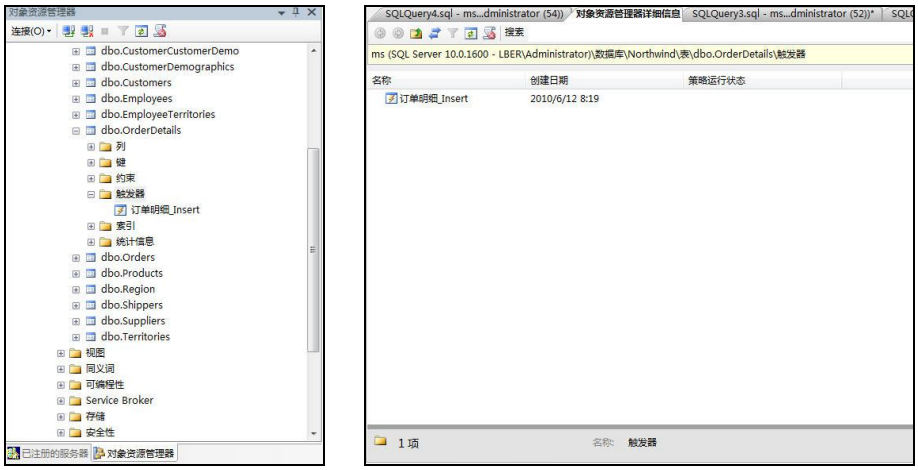

图 5.9　查看触发器列表

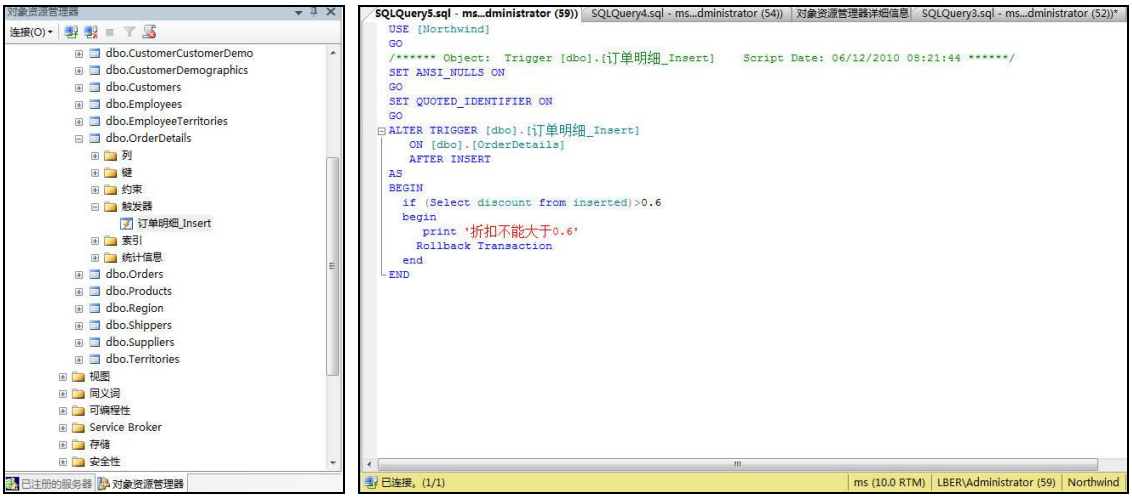

图 5.10　查看触发器内容

（1）sp_help：系统存储过程 sp_help 可以了解如触发器名称、类型、创建时间等基本信息，其语法格式为：

sp_help '触发器名'

【例 5.30】用系统存储过程 sp_help 查看触发器的情况。

sp_help '产品_Insert'

（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'

【例 5.31】用系统存储过程 sp_helptext 查看触发器的情况。

sp_helptext '产品_Insert'

5.3.10　修改 DML 触发器

在 Management Studio 中修改触发器之前，必须要先查看触发器的内容，通常在查询编辑器模板里显示的就是用来修改触发器的代码。编辑完代码之后，单击“执行”按钮运行即可。修改触发器的语法如下：

```
ALTER TRIGGER  触发器名
 ON  数据表名或视图名
  AFTER INSERT 或 DELETE 或 UPDATE
AS
BEGIN
--这里是要运行的 SQL 语句
END
GO
```

如果只是要修改触发器的名称，则可以使用存储过程 sp_rename，其语法如下：

```
sp_rename '旧触发器名','新触发器名'
```

值得一提的是，修改触发器名称有可能会使某些脚本或存储过程运行出错。

5.3.11　删除 DML 触发器

在 Management Studio 中删除触发器，必须要先查到触发器列表，右击其中的一个触发器，在弹出的快捷菜单中选择“删除”选项，此时将会弹出“删除对象”对话框，在其中单击“确定”按钮，删除操作完成。用以下 SQL 语句也可以删除触发器：

```
DROP TRIGGER  触发器名
```

注意：如果一个数据表被删除了，那么 SQL Server 会自动将与该表相关的触发器删除。

5.3.12　DDL 触发器

DDL 触发器是一种特殊的触发器，它在响应数据定义语言（DDL）语句时触发。一般用于在数据库中执行管理任务。与 DML 触发器一样，DDL 触发器也是通过事件来激活，并执行其中的 SQL 语句的。但与 DML 触发器不同，DML 触发器是响应 INSERT、UPDATE 或 DELETE 语句而激活的，DDL 触发器是响应 CREATE、ALTER 或 DROP 开头的语句而激活的。

一般来说，在以下几种情况下可以使用 DDL 触发器：

（1）数据库里的库架构或数据表架构很重要，不允许被修改。

（2）防止数据库或数据表被误操作删除。

（3）在修改某个数据表结构的同时修改另一个数据表的相应的结构。

（4）要记录对数据库结构操作的事件。

1. 创建 DDL 触发器

只要注意到 DDL 触发器和 DML 触发器的区别，设计 DDL 触发器与设计 DML 触发器也很类似。

建立 DDL 触发器的语法代码如下：

```
CREATE TRIGGER trigger_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 > [ ; ] }
```

用中文取代一下英文可以看得更明白：

```
CREATE TRIGGER  触发器名
ON ALL SERVER 或 DATABASE
FOR  或 AFTER
激活 DDL 触发器的事件
AS
```

要执行的 SQL 语句

其中：

- ON 后面的 All Server 是将 DDL 触发器作用到整个当前的服务器上。如果指定了这个参数，在当前服务器上的任何一个数据库都能激活该触发器。
- ON 后面的 DataBase 是将 DDL 触发器作用到当前数据库，只能在这个数据库上激活该触发器。
- For 或 After 是同一个意思，指定的是 After 触发器，DDL 触发器不能指定 Stead Of 触发器。
- 激活 DDL 触发器的事件包括两种，在 DDL 触发器作用在当前数据库情况下可以使用如表 5.9 所示的事件。

表 5.9 DDL 触发器作用在当前数据库情况下的可用事件

可用事件	可用事件
CREATE_APPLICATION_ROLE	DROP_PARTITION_SCHEME
ALTER_APPLICATION_ROLE	CREATE_PROCEDURE
DROP_APPLICATION_ROLE	ALTER_PROCEDURE
CREATE_ASSEMBLY	DROP_PROCEDURE
ALTER_ASSEMBLY	CREATE_QUEUE
DROP_ASSEMBLY	ALTER_QUEUE
ALTER_AUTHORIZATION_DATABASE	DROP_QUEUE
CREATE_CERTIFICATE	CREATE_REMOTE_SERVICE_BINDING
ALTER_CERTIFICATE	ALTER_REMOTE_SERVICE_BINDING
DROP_CERTIFICATE	DROP_REMOTE_SERVICE_BINDING
CREATE_CONTRACT	CREATE_ROLE
DROP_CONTRACT	ALTER_ROLE
GRANT_DATABASE	DROP_ROLE
DENY_DATABASE	CREATE_ROUTE
REVOKE_DATABASE	ALTER_ROUTE
CREATE_EVENT_NOTIFICATION	DROP_ROUTE
DROP_EVENT_NOTIFICATION	CREATE_SCHEMA
CREATE_FUNCTION	ALTER_SCHEMA
ALTER_FUNCTION	DROP_SCHEMA
DROP_FUNCTION	CREATE_SERVICE
CREATE_INDEX	ALTER_SERVICE
ALTER_INDEX	DROP_SERVICE
DROP_INDEX	CREATE_STATISTICS
CREATE_MESSAGE_TYPE	DROP_STATISTICS
ALTER_MESSAGE_TYPE	UPDATE_STATISTICS
DROP_MESSAGE_TYPE	CREATE_SYNONYM
CREATE_PARTITION_FUNCTION	DROP_SYNONYM
ALTER_PARTITION_FUNCTION	CREATE_TABLE
DROP_PARTITION_FUNCTION	ALTER_TABLE
CREATE_PARTITION_SCHEME	DROP_TABLE
ALTER_PARTITION_SCHEME	CREATE_TRIGGER

续表

可用事件	可用事件
ALTER_TRIGGER	CREATE_VIEW
DROP_TRIGGER	ALTER_VIEW
CREATE_TYPE	DROP_VIEW
DROP_TYPE	CREATE_XML_SCHEMA_COLLECTION
CREATE_USER	ALTER_XML_SCHEMA_COLLECTION
ALTER_USER	DROP_XML_SCHEMA_COLLECTION
DROP_USER	

在 DDL 触发器作用在当前服务器情况下，可以使用如表 5.10 所示的事件。

表 5.10　DDL 触发器作用在当前服务器情况下的可用事件

可用事件	可用事件
ALTER_AUTHORIZATION_SERVER	DROP_ENDPOINT
CREATE_DATABASE	ALTER_LOGIN
CREATE_ENDPOINT	DENY_SERVER
CREATE_LOGIN	DROP_DATABASE
GRANT_SERVER	DROP_LOGIN
ALTER_DATABASE	REVOKE_SERVER

【例 5.32】建立一个 DDL 触发器，用于保护数据库中的数据表不被修改、不被删除。

具体操作步骤如下：

（1）启动 Management Studio，登录到指定的服务器上。

（2）在对象资源管理器下选择数据库，定位到 Northwind 数据库上。

（3）单击“新建查询”按钮，在弹出的查询编辑器的编辑区中输入以下代码：

```
CREATE TRIGGER 禁止对数据表操作
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
  PRINT '对不起，您不能对数据表进行操作'
  ROLLBACK ;
```

（4）单击“执行”按钮，生成触发器。

【例 5.33】建立一个 DDL 触发器，用于保护当前 SQL Server 服务器中所有的数据库不能被删除。

具体代码如下：

```
CREATE TRIGGER 不允许删除数据库
ON all server
FOR DROP_DATABASE
AS
  PRINT '对不起，您不能删除数据库'
  ROLLBACK ;
GO
```

【例 5.34】建立一个 DDL 触发器，用来记录数据库修改状态。

具体操作步骤如下：

（1）建立一个用于记录数据库修改状态的表。

```
CREATE TABLE 日志记录表(
    编号 int IDENTITY(1,1) NOT NULL,
    事件 varchar(5000) NULL,
    所用语句 varchar(5000) NULL,
    操作者 varchar(50) NULL,
    发生时间 datetime NULL,
    CONSTRAINT PK_日志记录表 PRIMARY KEY CLUSTERED
)
GO
```

（2）建立 DDL 触发器。

```
CREATE TRIGGER 记录日志
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
DECLARE @log XML
SET @log = EVENTDATA()
INSERT 日志记录表
  (事件, 所用语句,操作者, 发生时间)
  VALUES
  (
    @log.value('(/EVENT_INSTANCE/EventType)[1]', 'nvarchar(100)'),
    @log.value('(/EVENT_INSTANCE/TSQLCommand)[1]', 'nvarchar(2000)'),
    CONVERT(nvarchar(100), CURRENT_USER),
    GETDATE()
  ) ;
GO
```

其中，Eventdata 是一个数据库函数，作用是以 XML 格式返回有关服务器或数据库事件的信息；@log.value 是返回 log 这个 XML 节点的值，节点的位置是括号里的第一个参数。

5.3.13 存储过程和触发器的区别

存储过程和触发器存储过程是一组 Transact-SQL 语句，它们只需编译一次，以后即可多次执行。因为 Transact-SQL 语句不需要重新编译，所以执行存储过程可以提高性能。触发器是一种特殊的存储过程，不由用户直接调用。创建触发器时，将其定义为在对特定表或列进行特定类型的数据修改时激发。

5.4 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。用户自定义函数不能用于执行一系列改变数据库状态的操作，但它可以像系统函数一样在查询或存储过程等的程序段中使用，也可以像存储过程一样通过 EXECUTE 命令来执行。用户自定义函数中存储了一个 Transact-SQL 例程，可以返回一定的值。

在 SQL Server 中根据函数返回值形式的不同将用户自定义函数分为 3 种类型：标量型函数、内联表值型函数和多声明表值型函数。

（1）标量型函数返回一个确定类型的标量值。其返回值类型为除 text、ntext、image、cursor、timestamp 和 table 类型外的其他数据类型。函数体语句定义在 BEGIN…END 语句内，其中包含了可以返回值的 Transact-SQL 命令。

（2）内联表值型函数以表的形式返回一个返回值，即它返回的是一个表。内联表值型函数没有由 BEGIN…END 语句括起来的函数体。其返回的表由一个位于 RETURN 子句中的 SELECT 命令段从数据库中筛选出来。内联表值型函数的功能相当于一个参数化的视图。

（3）多声明表值型函数可以看做标量型和内联表值型函数的结合体。它的返回值是一个表，但它和标量型函数一样有一个用 BEGIN…END 语句括起来的函数体，返回值的表中的数据是由函数体中的语句插入的。由此可见，它可以进行多次查询，对数据进行多次筛选。

5.4.1　创建用户自定义函数

SQL Server 为 3 种类型的用户自定义函数提供了不同的命令创建格式。

1. 创建标量型用户自定义函数

创建标量型用户自定义函数的语法格式如下：

```
CREATE FUNCTION[owner_name.] function_name
 (
    {@参数名 参数类型=[默认值]}[,...n]
  )
 RETURNS 返回类型
 [WITH <encryption|schemabinding>[,...n]]
 [AS]
 BEGIN
    function_body
    RETURN 返回表达式
 END
```

参数说明：

- owner_name：指定用户自定义函数的所有者。
- function_name：指定用户自定义函数的名称。
- @参数名：定义一个或多个参数的名称。一个函数最多可以定义 1024 个参数，每个参数前用@符号标明。参数的作用范围是整个函数。参数只能替代常量，不能替代表名、列名或其他数据库对象的名称。用户自定义函数不支持输出参数。
- 参数类型：指定标量型参数的数据类型，可以是除 text、ntext、image、cursor、timestamp 和 table 类型之外的其他数据类型。
- 返回类型：指定标量型返回值的数据类型，可以是除 text、ntext、image、cursor、timestamp 和 table 类型之外的其他数据类型。
- 返回表达式：指定标量型用户自定义函数返回的标量值表达式。
- function_body：指定一系列的 Transact-SQL 语句，它们决定了函数的返回值。
- encryption：加密选项。让 SQL Server 对系统表中有关 CREATE FUNCTION 的声明加密，以防止用户自定义函数作为 SQL Server 复制的一部分被发布。

- schemabinding：计划绑定选项，将用户自定义函数绑定到它所引用的数据库对象。如果指定了此选项，则函数所涉及的数据库对象从此将不能被删除或修改，除非函数被删除或去掉了此选项。应该注意的是，要绑定的数据库对象必须与函数在同一数据库中。

【例 5.35】标量型自定义函数。

```
CREATE FUNCTION testfun(@city varchar(50))
RETURNS int
AS
BEGIN
    DECLARE @num int
    SELECT @num=count(*)   FROM person.address WHERE city=@city
    RETURN @num
END
```

以上函数的功能为统计 AdventureWorks 数据库中的 person.address 表中 city 字段值为某值的记录的数量。

此函数执行时只需要在查询中输入函数名即可。

【例 5.36】输出函数结果。

```
PRINT testfun('Bothell')
```

2. 创建内联表值型用户自定义函数

创建内联表值型用户自定义函数的语法格式如下：

```
CREATE FUNCTION [owner_name.] function_name
 (
    {@参数名 参数类型=[默认值]}[,...n]
  )
 RETURNS table
 [WITH <encryption|schemabinding>[,...n]]
 [AS]
   RETURN [select-stmt]
```

参数说明：

- table：指定返回值为一个表。
- select-stmt：单个 SELECT 语句，确定返回的表的数据。
- 其余参数与标量型用户自定义函数的相同。

【例 5.37】内联表值型用户自定义函数。

```
CREATE FUNCTION testfuntable(@city varchar(50))
RETURNS table
AS
RETURN SELECT * FROM person.address WHERE city=@city
```

以上函数的功能是显示出 AdventureWorks 数据库中的 person.address 表中 city 字段值为某值的记录。

内联表值型用户自定义函数执行时，要把该函数当成一个数据表，而使用 SELECT 语句显示函数结果。

【例 5.38】使用 SELECT 语句显示函数结果。

```
SELECT * FROM testfuntable('Bothell')
```

3. 创建多声明表值型用户自定义函数

创建多声明表值型用户自定义函数的语法格式如下：

```
CREATE FUNCTION [owner_name.] function_name
 (
   {@参数名 参数类型=[默认值]}[,...n]
  )
 RETURNS @return_variable table <(column_definition|table_constraint)[,...n]>
 [WITH <encryption|schemabinding>[,...n]]
 [AS]
 BEGIN
   function_body
   RETURN
 END
```

参数说明：@return_variable 是一个 table 类型的变量，用于存储和累积返回的表中的数据行。其余参数与标量型用户自定义函数的相同。

在多声明表值型用户自定义函数的函数体中允许使用下列 Transact-SQL 语句：赋值语句、流程控制语句、定义作用范围在函数内的变量和游标的 DECLARE 语句、SELECT 语句、编辑函数中定义的表变量的 INSERT/UPDATE/DELETE 语句，在函数中允许涉及诸如声明游标、打开游标、关闭游标、释放游标这样的游标操作，对于读取游标而言，除非在 FETCH 语句中使用 INTO 从句来对某一变量赋值，否则不允许在函数中使用 FETCH 语句来向客户端返回数据。此外，不确定性函数不能在用户自定义函数中使用。所谓不确定性函数是指那些使用相同的调用参数在不同时刻调用得到的返回值不同的函数。

【例 5.39】多声明表值型用户自定义函数。

```
CREATE FUNCTION testfunmultable()
RETURNS @cityinfo table
(
  CITY varchar(50),
  NUM int
)
AS
  BEGIN
     DECLARE @citynum int
     DECLARE @citynum1 int
     DECLARE @i int
     SET @i=1
     DECLARE @city varchar(50)
     DECLARE @cursor cursor
     SET @cursor=cursor for select distinct city from person.address
     SELECT @citynum=count(distinct city) from person.address
     OPEN @cursor
     WHILE @i<=@citynum
     BEGIN
        FETCH next FROM @cursor into @city
        SELECT @citynum1=count(*) FROM person.address where city=@city

        INSERT into @cityinfo values(@city,@citynum1）
        SET @i=@i+1
     END
```

```
    RETURN
END
```

以上函数的功能是显示出 AdventureWorks 数据库中的 person.address 表中每个城市的记录数。程序中首先统计出 person.address 表中 city 字段中存储的城市的总数，然后定义游标依次读取每个城市的名称，并计算出对应的城市记录数，将结果存入到临时表中，并随函数返回。

多声明表值型用户自定义函数执行时，要把该函数当成一个数据表，而使用 SELECT 语句显示函数结果。

【例 5.40】用 SELECT 语句显示函数结果。

```
SELECT * FROM testfunmultable ()
```

5.4.2 修改和删除用户自定义函数

用 ALTER FUNCTION 命令也可以修改用户自定义函数。此命令的语法与 CREATE FUNCTION 相同，因此使用 ALTER FUNCTION 命令其实相当于重建了一个同名的函数。

另外，可以用 DROP FUNCTION 命令删除用户自定义函数，其语法如下：

```
DROP FUNCTION { [ owner_name. ] function_name } [,...n ]
```

【例 5.41】删除用户自定义函数 testfunmultable。

```
DROP FUNCTION testfunmultable
```

本章小结

本章主要讲述了在 SQL Server 2008 中运用 T-SQL 进行一系列的程序设计，其中包括局部变量、全局变量、流程控制语句和常用函数等。Transact-SQL 是 SQL Server 对原有标准 SQL 的扩充，可以帮助我们完成更为强大的数据库操作功能。在 SQL Server 下，利用 Transact-SQL、SSMS 可以完成对各种数据库对象，如数据库、数据表、视图、存储过程、触发器、用户自定义函数的管理（包括创建、修改、查看、删除等），尤其是其在存储过程的设计、触发器的设计方面应用更为广泛。

习题五

1．使用 T-SQL 语句建立一个宾馆数据库。具体文件属性如表 5.11 所示。

表 5.11 文件属性

参数	参数值
数据库名	bg
逻辑数据文件名	Bg_dat
操作系统数据文件名	bg_dat.mdf
数据文件的初始大小	2MB
数据文件的最大大小	20MB
数据文件增长幅度	2MB
日志逻辑文件名	Bg_log

续表

参数	参数值
操作系统日志文件名	bg_log.ldf
日志文件初始大小	1MB
日志文件增长幅度	15%

2．使用 T-SQL 语句建立客房标准信息表、订房信息表、客房信息表等，具体表结构如表 5.12 至表 5.14 所示。

表 5.12　roomtype（客房标准信息表）

列名	中文说明	数据类型	允许空值	说明
typeid	客房类型编号	varchar	×	主键
typename	客房类型名称	varchar	×	
area	面积	Numeric	√	
bednum	床位数量	tiny	×	8>Badnum>0
price	单价	numeric	√	默认为 100
htelephone	是否有电话	varchar	√	默认为有

表 5.13　rooms（客房信息表）

列名	中文说明	数据类型	允许空值	说明
roomno	客房编号	varchar	×	主键
typeid	客房类型	varchar	×	
roomposition	客房位置	varchar	√	
roomprice	单价	numeric	√	
putup	是否被预订	varchar	×	默认为否
roommemo	备注	text	√	

表 5.14　bookin（订房信息表）

列名	中文说明	数据类型	允许空值	说明
bookno	订房编号	varchar	×	主键
guest_id	身份证号	varchar	×	唯一
roomno	客房编号	varchar	×	外键参照 rooms 表的 roomno
staydate	入住日期	Datetime	√	默认为系统日期
returndate	结算日期	Datetime	√	
discount	折扣	numeric	√	
ammount	金额	numeric	√	

3．用 T-SQL 语句插入如表 5.15 所示的客房信息。

表 5.15　要插入的客户信息

客房编号	客房类型	客房位置	单价	是否被预订	备注
1003	3	10 层朝南	100	否	

4．查询所有客户的身份证号、客房编号、入住日期、结算日期等信息。

5．查询被预订的“标准房”（客房类型）的客房的客房编号，按客房编号的升序排序。

6．按客房类型统计各类客房的平均价格。

7．查询身份证号为“230102198711040064”的客户最近入住酒店的日期、客房编号、住了几天等信息。

8．创建存储过程 proc_kroom，实现功能：根据客房类型号查询是否有该类型的未被预订的房间，如果有，则显示提示信息“此类型的房间有未被预订的客房”；如果没有，则显示提示信息“此类型的房间已预订满!”。

9．创建触发器，实现如果删除客房标准信息表中的某种客房类型记录，则相应的客房信息表中所有此类型的客房记录都删除。

10．存储过程与触发器有什么不同？

11．存储过程与自定义函数有什么不同？