第 7 章　SQL Server 2005 程序设计

SQL 标准的确定使得大多数数据库厂家纷纷采用 SQL 语言作为其数据库操作语言，而各公司又在 SQL 标准的基础上进行不同程度的扩充，形成各自的数据库语言。Transact-SQL 就是其中的一种，它是 SQL Server 数据库应用的中心，所有 SQL Server 应用程序，无论使用哪种编程接口开发，在操作 SQL Server 数据库时均采用 Transact-SQL 语句。

Transact-SQL 增强了 SQL 的功能，扩展了过程化程序设计功能，同时又保持与 SQL 标准的兼容性。这一章将介绍使用 Transact-SQL 语言实现数据库操作及程序设计。

7.1　Transact-SQL 程序设计基础

7.1.1　Transact-SQL 语法格式约定

Transact-SQL 语句由以下语法元素组成：

- 标识符。
- 数据类型。
- 函数。
- 表达式。
- 运算符。
- 注释。
- 关键字。

在编写 Transact-SQL 脚本程序时，常采用不同的书写格式来区分这些语法元素。在 SQL Server 2005 中，对于语法格式的约定包括：

（1）大写字母：代表 Transact-SQL 保留的关键字。例如 SELECT * FROM titles 中的 SELECT 和 FROM。

（2）小字字母：表示对象标识符和表达式等。例如上面语句中的 titles 标识符。

（3）大括号{}或尖括号<>：大括号或尖括号中的内容为必选参数，其中可包含多个选项，各选项之间用竖线分隔，用户必须从这些选项中选择一项。

（4）方括号[]：它所列出的项目列表为可选项，用户可根据需要选择使用。

（5）竖线｜：表示参数之间是“或”的关系，可以从中选择任意一个使用。

（6）[,…n]：表示重复前面的语法单元，各项之间用逗号分隔。

（7）[…n]：表示重复前面的语法单元，各项之间用空格分隔。

（8）注释：注释为 Transact-SQL 脚本程序中的说明信息，SQL Server 不执行这部分内容。SQL Server 支持以下两种注释格式：

- 单行注释：使用两个连字符（--）作为注释的开始标志。从它到本行行尾的所有内容均为注释信息。

- 块注释：块注释的格式为/*…*/，其间的所有内容均为注释信息。块注释与单行注释的不同之处是它可以跨越多行，并且可以插入在程序代码中的任何地方。

7.1.2　标识符

标识符是指用户在 SQL Server 中定义的服务器、数据库、数据库对象、变量和列等对象名称。SQL Server 标识符分为常规标识符和定界标识符两类。

1．常规标识符

在 Transact-SQL 语句中，常规标识符不需要定界符进行分隔。例如，下面语句中的 jobs 和 MyDB 两个标识符即为常规标识符。

```
SELECT * FROM jobs
GO
CREATE DATABASE MyDB
GO
```

常规标识符遵守以下的命名规则：

- 标识符长度可以为 1～128 个字符。
- 标识符的首字符必须为 Unicode 2.0 标准所定义的字母或_、@、＃符号。
- 标识符第一个字符后面的字符可以为 Unicode 2.0 所定义的字符、数字或@、#、$、_符号。
- 标识符内不能嵌入空格和特殊字符。
- 标识符不能与 SQL Server 中的保留关键字同名。

2．定界标识符

定界标识符允许在标识符中使用 SQL Server 保留关键字或常规标识符中不允许使用的一些特殊字符，但必须由双引号或方括号定界符进行分隔。

例 7-1　下面语句所创建的数据库名称中包含空格，所创建的表名与 SQL Server 保留字相同，所以在 Transact-SQL 语句中需要使用定界符来分隔这些标识符。

```
--所创建的数据库名称中包含空格：
CREATE DATABASE [My DB]
GO
USE [My DB]
--所创建的表名与 Transact-SQL 保留字相同：
CREATE TABLE [table]
(
    column1 CHAR(8) NOT NULL PRIMARY KEY,
    column2 SMALLINT NOT NULL
)
GO
```

7.1.3　运算符

运算符用来执行列、常量或变量间的数学运算和比较操作。SQL Server 支持的运算符分算术运算符、位运算符、比较运算符、逻辑运算符、字符串连接运算符、赋值运算符和单目运算符 7 种。

1. 算术运算符

用于执行数字型表达式的算术运算，SQL Server 2005 支持的算术运算符包括：

- +：加。
- −：减。
- *：乘。
- /：除。
- %：取模。

2. 位运算符

对整数或二进制数据进行按位与（&）、或（|）、异或（^）、求反（～）等逻辑运算。在 Transact-SQL 语句中对整数数据进行位运算时，首先把它们转换为二进制数，然后再进行计算。其中与（&）、或（|）、异或（^）运算需要两个操作数。

3. 比较运算符

用来比较两个表达式的值是否相同。SQL Server 支持的比较运算符包括：

- >：大于。
- =：等于。
- <：小于。
- >=：大于等于。
- <=：小于等于。
- <>：不等于。
- !=：不等于。
- !>：不大于。
- !<：不小于。

4. 逻辑运算符

用于测试条件是否为真，它与比较运算符一样，根据测试结果返回布尔值 TRUE、FALSE 或 UNKNOW。逻辑运算符有以下几种：

- AND。
- OR。
- NOT。
- [NOT] BETWEEN….AND。
- [NOT] LIKE。
- [NOT] IN。
- IS [NOT] NULL。
- ALL、SOME、ANY。
- [NOT] EXISTS。

5. 字符串连接符

“+”可以实现字符串之间的连接操作。SQL Server 中，字符串之间的其他操作通过字符串函数实现。

例 7-2 下列表达式用字符串连接符实现两字符串间的连接。

```
SELECT 'abc' + '123'
```

其计算结果为 abc123。

6. 赋值运算符

SQL Server 中的赋值运算符为等号（=），它将表达式的值赋给一个变量，比如：

```
DECLARE @var INT
SET @var = 100 + 50
```

与其他高级语言不同，SQL Server 中，变量的赋值必须在 SET 语句中完成，不能作为一个独立语句。如下面的语句是错误的。

```
@var = 100 + 50
```

也可以在 SELECT 语句中为变量赋值，如：

```
SELECT @var = 100 + 50
```

7.1.4　变量

变量和参数是 Transact-SQL 语句之间传递数据的两种途径：变量常用在批处理脚本程序内的 Transact-SQL 语句之间传递数据，而参数则用在存储过程和执行该存储过程的批处理脚本程序之间传递数据。

1. 变量声明

变量是由用户声明并可赋值的实体。Transact-SQL 中用 DECLARE 语句声明变量，并在声明后将变量的值初始化为 NULL。DECLARE 语句的语法格式为：

```
DECLARE @variable_name date_type
[,@variable_name data_type…]
```

例如，下面的语句声明一个 datetime 类型变量：

```
DECLARE @date_var datetime
```

在一个 DECLARE 语句中可以同时声明多个局部变量，它们相互之间用逗号分隔。例如，下面的语句声明两个变量@var1 和@var2，它们的数据类型分别为 INT 和 MONEY：

```
DECLARE @var1 INT,@var2 MONEY
```

注意：Transact-SQL 与其他高级语言不同，局部变量名必须以“@”开头。变量的定义可以在程序的任何位置，但必须保证先定义后使用。

2. 变量赋值。

变量声明后，DECLARE 语句将变量初始化为 NULL，这时，我们可以调用 SET 语句或 SELECT 语句为变量赋值，但建议使用 SET 语句。SET 语句的语法格式为：

```
SET @variable_name = expression
```

SELECT 语句为变量赋值的语法格式为：

```
SELECT @variable_name = expression [FROM <表名> WHERE <条件>]
```

Expression 为有效的 SQL Server 表达式，它可以是一个常量、变量、函数、列名和子查询等。

例 7-3　下面的代码用 SET 语句为声明的@date_var 变量赋值：

```
DECLARE @date_var DATETIME          --声明
SET @date_var = '2004-4-1'          --赋值
SELECT '@date_var' = @date_var      --显示变量值
GO
```

例 7-4　下面的程序用 SELECT 语句将查询结果赋值给变量：

```
DECLARE @date_var DATETIME              --声明
SELECT @date_var = MIN(pubdate)     --赋值
FROM titles
SELECT '@date_var' = @date_var      --显示变量值
GO
```

也可以用 SET 语句将查询结果赋值给变量，上例改写为：

```
DECLARE @date_var DATETIME                    --声明
SET @date_var = (SELECT MIN(pubdate)          --赋值
                 FROM    titles )
SELECT '@date_var' = @date_var                --显示变量值
GO
```

7.1.5 流程控制语句

流程控制语句用于控制 Transact-SQL 语句、语句块或存储过程的执行流程。

1．BEGIN…END 语句

该语句用于将多条 Transact-SQL 语句封装起来，构成一个语句块，它用在 IF…ELSE、WHILE 等语句中，使语句块内的所有语句作为一个整体被执行。

BEGIN…END 语句的语法格式为：

```
BEGIN
    {SQL 语句 | 语句块}
END
```

比如：

```
IF EXISTS (SELECT title_id FROM titles WHERE title_id = 'TC5555')
BEGIN
    DELETE FROM titles WHERE title_id = 'TC5555'
    PRINT 'TC5555 is deleted.'
END
ELSE
    PRINT 'TC5555 not found.'
```

2．条件语句

条件语句的语法格式为：

```
IF   <布尔表达式>
    {SQL 语句 | 语句块}
[ELSE
    {SQL 语句 | 语句块}]
```

条件语句的执行流程是：当条件满足时，也就是布尔表达式的值为真时，执行 IF 语句后的语句或语句块。ELSE 语句为可选项，它引入另一个语句或语句块，当布尔表达式的值为假时，执行该语句或语句块。

布尔表达式可以包含列名、常量和运算符所连接的表达式，也可以包含 SELECT 语句。包含 SELECT 语句时，该语句必须括在括号内。比如：

```
IF EXISTS (SELECT pub_id FROM publishers WHERE pub_id='9999'
   PRINT 'Luceerne Publishing'
ELSE
```

```
PRINT 'NOT Found Lucerne Publishing'
```

在这个例子中，如果 publishers 表中存在标识为 9999 的出版社，则打印该出版社的名称：Lucerne Publishing；否则打印提示信息：NOT Found Lucerne Publishing。

在条件语句中，IF 子句和 ELSE 子句都允许嵌套，SQL Server 对它们的嵌套级数没有限制。比如：

```
DECLARE @var INT
SET @var=0
IF @var>50
    IF @var>100
    PRINT '@var>100'
    ELSE
        PRINT '50<@var<=100'
ELSE
    IF @var<20
        PRINT '@var<20'
    ELSE
PRINT '20<@var<=50'
```

3．*转移语句*

转移语句的语法格式为：

```
GOTO　<标号>
```

它将 SQL 语句的执行流程无条件转移到用户所指定的标号处。GOTO 语句和标号可用在存储过程、批处理或语句块中。标号名称必须遵守 Transact-SQL 标识符命名规则。定义标号时，在标号名后加上冒号。GOTO 语句常用在循环语句和条件语句内，它使程序跳出循环或进行分支处理。

例 7-5　下面的代码利用转移语句和条件语句求 10 的阶乘。

```
DECLARE @s INT, @times　INT
SELECT @s=1,@times=1
label1:          --定义语句标号
    SET @s=@s*@times
    SET @times=@times+1
    IF @times<=10
        GOTO label1
    PRINT '10 的阶乘=' + str(@s)
```

注意：*在程序设计中，一般不要使用 GOTO 语句，因为它会使程序不易阅读和理解。所有使用 GOTO 语句能完成的逻辑都可以用条件语句或循环语句完成。*

4．*循环语句*

循环语句根据所指定的条件重复执行一个 Transact-SQL 语句或语句块，只要条件成立，循环体就会被重复执行下去。循环语句还可以与 BREAK 语句和 CONTINUE 语句一起使用，BREAK 语句导致程序从循环中跳出，而 CONTINUE 语句则使程序跳过循环体内 CONTINUE 语句后面的 Transact-SQL 语句，并立即进行下次条件测试。

循环语句的语法格式为：

```
WHILE <布尔表达式>
     {SQL 语句丨语句块}
```

```
[BREAK]
{SQL 语句｜语句块}
[CONTINUE]
[SQL 语句｜语句块]
```

下面以一个例子说明 WHILE 结构的用法。

例 7-6 该例子的功能是求 1 到 10 之间的奇数和。

```
DECLARE @i SMALLINT,@sum SMALLINT
SET @i=0
SET @sum=0
WHILE @i>=0
BEGIN
    SET @i=@i+1
    IF @i<=10
        IF (@i % 2)=0
            CONTINUE
    ELSE
          SET @sum=@sum+@i
    ELSE
    BEGIN
        PRINT '1 到 10 之间的奇数和为'+STR(@sum)
        BREAK
    END
END
```

5. *等待语句*

等待语句挂起一个连接中各语句的执行，直到指定的某一时间点到来或在一定的时间间隔之后继续执行。等待语句的语法格式为：

```
WAITFOR {DELAY 'interval' ｜TIME 'time'}
```

其中，DELAY 子句指定 SQL Server 等待的时间间隔，TIME 子句指定一时间点。Interval 和 time 参数为 DATETIME 数据类型，其格式为“hh:mm:ss”，它们分别说明等待的时间长度和时间点，在 time 内不能指定日期。

比如指定在 10 点钟执行一个查询语句。

```
BEGIN
    WAITFOR TIME '10:00:00'
    SELECT * FROM Borrow
END
```

再比如，下面的语句设置在 5 秒后执行一次查询操作：

```
BEGIN
    WAITFOR DELAY '00:00:05'
    SELECT * FROM Borrow
END
```

6. *返回语句*

返回语句结束查询、存储过程或批的执行，使程序无条件返回，其后面的语句不再执行。返回语句的语法格式为：

```
RETURN  [整数表达式]
```

7. CASE 语句

CASE 语句用于计算条件列表并返回多个可能的结果表达式之一。

CASE 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。

简单 CASE 函数语法：

```
CASE input_expression
    WHEN when_expression THEN result_expression
   [ ...n ]
    [
   ELSE else_result_expression
    ]
END
```

CASE 搜索函数语法：

```
CASE
    WHEN Boolean_expression THEN result_expression
   [ ...n ]
    [
   ELSE else_result_expression
    ]
END
```

其中：input_expression：使用简单 CASE 格式时所计算的表达式。input_expression 是任意有效的表达式。

WHEN when_expression：使用简单 CASE 格式时要与 input_expression 进行比较的简单表达式。when_expression 是任意有效的表达式。

THEN result_expression：当 input_expression = when_expression 计算结果为 TRUE，或者 Boolean_expression 计算结果为 TRUE 时返回的表达式。result expression 是任意有效的表达式。

ELSE else_result_expression：比较运算计算结果不为 TRUE 时返回的表达式。如果忽略此参数且比较运算计算结果不为 TRUE，则 CASE 返回 NULL。else_result_expression 是任意有效的表达式。else_result_expression 及任何 result_expression 的数据类型必须相同或必须是隐式转换的数据类型

WHEN Boolean_expression：使用 CASE 搜索格式时所计算的布尔表达式。Boolean_expression 是任意有效的布尔表达式。

例 7-7　在 READER 表中，我们是用 1、2、3 分别代表学生、教师和临时读者。如果我们希望在查询结果中直接显示具体的读者类型，可以用如下的查询语句：

```
SELECT cardid,name,sex,dept,'class'=
CASE class
   WHEN 1 THEN '学生'
   WHEN 2 THEN '教师'
   WHEN 3 THEN '临时读者'
  END
FROM reader
```

7.1.6 异常处理

1. TRY...CATCH 语句

TRY...CATCH 语句是在 SQL Server 2005 数据库中新加入的，用于实现异常的处理，Transact-SQL 语句组可以包含在 TRY 块中，如果 TRY 块内部发生错误，则会将控制转入到 CATCH 块中。

TRY...CATCH 语法结构如下：

```
BEGIN TRY
   {语句丨语句块}
END TRY
BEGIN CATCH
   {语句丨语句块}
END CATCH
```

正常情况下，执行 BEGIN TRY 与 END TRY 之间的代码，BEGIN CATCH 与 END CATCH 之间的代码不会执行，如果 BEGIN TRY 与 END TRY 之间的代码在执行过程中出现了错误，程序将终止错误语句之后代码的运行，将控制转入到 CATCH 块中。

例 7-8 下面的代码中，将会执行 BEGIN CATCH 与 END CATCH 之间的代码。

```
DECLARE @x int,@y int,@z int
  SET @x=10
  SET @y=0
BEGIN TRY
  SET @z=@x/@y         ---因为@y 为零，该语句在执行时会触发错误
  PRINT @z
END TRY
BEGIN CATCH
  PRINT '被零除错误'
END CATCH
```

2. 与异常有关的函数

在 CATCH 块的作用域内，可以使用下列的系统函数获取错误信息。

ERROR_NUMBER()：返回错误号。

ERROR_SEVERITY()：返回错误的严重级别。

ERROR_STATE()：返回错误状态号。

ERROR_PROCEDURE()：返回发生错误的存储过程或触发器的名称。

ERROR_LINE()：返回发生错误的行号。

ERROR_MESSAGE()：返回错误的消息文本。

3. 抛出错误语句

RAISERROR 生成错误消息并启动会话的错误处理。RAISERROR 可以引用 sys.messages 目录视图中存储的用户定义消息，也可以动态建立消息。该消息作为服务器错误消息返回到调用应用程序，或返回到 TRY…CATCH 语句中的 CATCH 块。RAISERROR 语法如下：

```
RAISERROR ( message|msg_id,,severity, state    [ ,argument [ ,...n ] ] )
[ WITH option [ ,...n ] ]
```

说明：

message|msg_id：返回到调用应用程序的错误消息，可以是 sys.messages 系统表中存储的用户定义消息（msg_id），也可以是字符串常量或字符变量（message）。如果使用字符串常量或字符变量，系统默认的消息号为 50000。

severity：用户定义的与该消息关联的严重级别，任何用户都可以指定 0～18 之间的严重级别。只有 sysadmin 固定服务器角色成员或具有 ALTER TRACE 权限的用户才能指定 19～25 之间的严重级别。若要使用 19～25 之间的严重级别，必须选择 WITH LOG 选项。

State：介于 1～127 之间的任意整数，如果在多个位置引发相同的用户定义错误，则针对每个位置使用唯一的状态号有助于找到引发错误的代码段。

例：RAISERROR ('不能进行更新操作',16 ,1)

7.1.7　游标

SQL 语言可以认为是一种面向集合的语言，它对数据库中数据的操作是面向集合的操作。所谓面向集合的操作是指对结果集执行一个特定的动作。但实际上，某些业务规则却要求对结果集逐行执行操纵，而不是对整个集合执行操纵。ANSI-92 定义的游标正是基于逐行操纵结果集的，当然 Transact-SQL 也遵循这一标准。也就是说，Transact-SQL 游标可以使用户逐行访问 SQL Server 返回的结果集。

游标的优点：

- 允许程序对由查询语句 SELECT 返回的行集合中的每一行执行相同或不同的操作，而不是对整个行集合执行同一个操作。
- 游标实际上作为面向集合的数据库管理系统（RDBMS）和面向行的程序设计之间的桥梁，使这两种处理方式通过游标沟通起来。

（1）游标的定义。游标定义的语法格式如下：

```
DECLARE <游标名> [INSENSITIVE] [SCROLL] CURSOR
FOR
<查询语句>
[FOR <READ ONLY|UPDATE [OF column_list]>
```

各选项的含义：INSENSITIVE 选项说明：游标结果集合填充后，所有应用程序对游标基表中数据的修改不能反映到当前游标结果集合中。此外，这种游标也禁止应用程序通过该游标对其基表中的数据进行修改。在 SQL Server 中，如果使用该选项，游标结果集合会被拷贝到一个临时表中（保存在 tempdb 数据库中），所有对游标的操作都是基于该临时拷贝数据。

SCROLL 选项指出所定义的游标可以向前也可以向后提取记录行数据，如果没有该选项，则只能向后提取数据行。

READ ONLY 说明所定义的游标为只读。

UPDATE [OF column_list]说明可以通过游标修改其基表中的数据。其中可修改的列由 column_list 参数列出。如果省略 OF column_list，则所有列都可以修改。

（2）打开游标。

```
OPEN <游标名>
```

说明：

当游标打开成功时，游标位置指向结果集的第一行之前。

只能打开已经声明但尚未打开的游标。

（3）从一个打开的游标中提取数据行。游标声明被打开后，游标位置位于结果集的第一行之前，由此可以从结果集中提取数据行。提取数据行的语法格式如下：

FETCH [[NEXT|PRIOR|FIRST|LAST|ABSOLUTE { n | @nvar}| RELATIVE { n | @nvar}]　FROM] <游标名> [INTO <变量名列表>]

说明：

- NEXT：表示提取下一条记录。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。
- PRIOR：表示提取前一条记录。
- FIRST：表示提取第一条记录。
- LAST：表示提取最后一条记录。
- ABSOLUTE { n | @nvar}：如果 n 或 @nvar 为正数，则返回从游标头开始的第 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，并且 @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。
- 如果以上选项都省略，默认为 NEXT。

在 SQL Server 2005 中有两个全局变量可以提供关于游标活动的信息：

@@FETCH_STATUS 保存着 FETCH 语句执行后的状态信息，其值和含义如表 7-1 所示。

表 7-1　FETCH 语句执行后的状态信息

值	含义
0	表示成功完成 FETCH 语句
−1	表示 FETCH 有错误，或者当前游标位置已在结果集中的最后一行，结果集中不再有数据
−2	提取的行不存在

@@ROWCOUNT 保存着自游标打开后的第一个 FETCH 语句，直到最近一次的 FETCH 语句为止，已从游标结果集中提取的行数。一旦结果集中所有行都被提取，那么 @@ROWCOUNT 的值就是该结果集的总行数。关闭游标时，该变量也被删除。在 FETCH 语句执行后查看这个变量，可提供从该 FETCH 指定的游标结果集中已提取的行数。

（4）关闭游标。关闭游标即删除游标当前结果集合，并释放游标对数据库的所有锁定，关闭游标并不改变它的定义，但不能再从游标中提取数据，要使用已关闭游标中的数据，可以再次用 OPEN 语句打开。关闭游标的语法格式如下：

CLOSE <游标名>

（5）释放游标。释放游标将释放所有分配给此游标的资源，包括该游标的名称。释放游标的语法格式如下：

```
DEALLOCATE <游标名>
```

如果释放一个已经打开但尚未关闭的游标，系统会自动先关闭这个游标，然后再释放。

（6）游标应用举例。在图书管理信息系统（BookSys）中，有一个名为 Book 的图书信息表，其中有一个名为 Price 的图书单价字段。考虑到图书维护成本的不断增长，要按如下规则对图书单价进行提价：30 元以下的，提价 10%；60 元以下的，提价 20%；60 元以上的，提价 30%。因为对结果集执行的不是一个统一的操作，而是需要对每一行记录的单价进行判断，故需要用游标实现。用游标实现上述功能如下：

```
DECLARE cursorBook CURSOR            --声明一个名为 cursorBook 的游标
FOR
  SELECT BookID,Price FROM BOOK
DECLARE @BookID CHAR(20)             --声明两个局部变量，用于存储两个字段的值
DECLARE @Price DECIMAL(5,2)
OPEN cursorBook                      --打开游标
/*从游标中提取字段值，分别放到两个变量中*/
FETCH NEXT FROM cursorBook INTO @BookID,@Price
WHILE @@FETCH_STATUS=0               --如果提取成功
BEGIN
    IF @Price<30                     --对价格进行判断
        UPDATE Book SET Price=(1+0.1)*Price WHERE BookID=@BookID
    ELSE IF @Price<60
        UPDATE Book SET Price=(1+0.2)*Price WHERE BookID=@BookID
    ELSE IF @Price>=60
        UPDATE Book SET Price=(1+0.3)*Price WHERE BookID=@BookID
    FETCH NEXT FROM cursorBook INTO @BookID,@Price
END
CLOSE cursorBook                 --关闭游标
DEALLOCATE cursorBook            --释放游标
```

上例也可以通过可更新游标来实现，代码修改如下：

```
DECLARE cursorBook CURSOR        --声明一个名为 cursorBook 的游标
FOR
  SELECT BookID,Price FROM BOOK
  FOR UPDATE OF Price
DECLARE @BookID CHAR(20)             --声明两个局部变量，用于存储两个字段的值
DECLARE @Price DECIMAL(5,2)
OPEN cursorBook                      --打开游标
/*从游标中提取字段值，分别放到两个变量中*/
FETCH NEXT FROM cursorBook INTO @BookID,@Price
WHILE @@FETCH_STATUS=0               --如果提取成功
BEGIN
    IF @Price<30                     --对价格进行判断
        UPDATE Book SET Price=(1+0.1)*Price WHERE current of cursorBook
    ELSE IF @Price<60
        UPDATE Book SET Price=(1+0.2)*Price WHERE current of cursorBook
```

```
        ELSE IF @Price>=60
            UPDATE Book SET Price=(1+0.3)*Price WHERE current of cursorBook
        FETCH NEXT FROM cursorBook INTO @BookID,@Price
    END
    CLOSE cursorBook                --关闭游标
    DEALLOCATE cursorBook           --释放游标
```

说明：current of cursorBook 表示用游标中的当前记录来定位基表要修改的记录。

7.2 存储过程

存储过程是数据库中重要的数据对象，一个设计良好的数据库应用系统通常都会用到存储过程。SQL Server 2005 数据库提供了多种建立存储过程的机制，使用户可以使用 Transact-SQL 或 CLR 方式建立存储过程。SQL Server 2005 数据库还提供了用户可以直接使用的系统存储过程，通过这些存储过程，用户可以方便的管理数据库。

7.2.1 存储过程概述

一个存储过程是被 SQL Server 编译成一个单一执行计划的一组 Transact-SQL 语句。当存储过程第一次被执行时，这个计划被存储在内存的高速缓冲存储区中，以便于这个计划可以多次重复使用。每次存储过程运行时 SQL Server 并不需要重新对应用程序进行编译。Transact-SQL 中的存储过程可以接收输入参数，以参数形式返回输出值，或者返回成功、失败的状态信息。当存储过程被调用时，程序中所有的语句都被处理。

存储过程分为三类：系统提供的存储过程、用户定义的存储过程和扩展存储过程。

（1）系统提供的存储过程：在安装 SQL Server 时，系统创建了很多系统存储过程。系统存储过程主要用于从系统表中获取信息，也为系统管理员和合适用户（即有权限的用户）提供更新系统表的途径。它们中的大部分可以在用户数据库中使用。系统存储过程的名字都以“sp_”为前缀。

（2）自定义的存储过程：是由用户为完成某一特定功能而编写的存储过程。在 SQL Server 2005 中，按编写的语言，又分为两种类型：Transact-SQL 和 CLR。

- Transact-SQL 存储过程，是指保存的 Transact-SQL 语句集合，可以接收和返回用户提供参数的存储过程。Transact-SQL 存储过程多用于数据库业务逻辑的处理。
- CLR 存储过程，是指对 Microsoft .NET Framework 公共语言运行时（CLR）方法的调用，它们在.NET Framework 程序集中是作为类的公共静态方法实现的，可以通过 SQL Server 2005 数据库引擎直接运行。

（3）扩展存储过程：是对动态链接库（DLL）函数的调用。

7.2.2 存储过程的优点

存储过程具有如下的优点：

（1）减少网络流量。因为存储过程存储在服务器上，并在服务器上运行。只有调用存储过程的命令和返回的结果才在网络上传输。所以，可以减少网络流量。

（2）增强代码的重用性和共享性。一个存储过程是为了完成某一个特定功能而编写的一

个模块，该模块可以被很多用户重用，也可以被很多用户共享。所以，存储过程可以增强代码的重用性和共享性，加快应用的开发速度，提高开发的质量和效率。

（3）加快系统运行速度。第一次执行后的存储过程会在缓冲区中创建查询树，使得第二次执行时不用进行预编译，从而加快速度。

（4）加强安全性。因为可以不授予用户访问存储过程所涉及的表的权限，而只授予访问存储过程的权限，这样，既可以保证用户通过存储过程操纵数据库中的数据，又可以保证用户不能直接访问与存储过程相关的表，从而保证表中数据的安全性。

7.2.3　用 Transact-SQL 语句创建存储过程

创建存储过程的语法为：

```
CREATE PROCEDURE <procedure_name>
    [WITH ENCRYPTION]
    [< @parameter data_type> [ = default ] [ OUTPUT ]
    ] [ ,...n ]
AS sql_statement [ ...n ]
```

其中：

- WITH ENCRYPTION：加密存储过程代码，保护作者知识产权。
- procedure_name：存储过程的名称。
- @parameter：参数名称，注意名称前必须有“@”符号。
- data_type：参数的数据类型。
- default：输入参数的缺省值。
- OUTPUT：表明该参数是输出参数。
- sql_statement：SQL 语句，这是存储过程的重点构造部分。

下面是推荐的创建存储过程的 3 个步骤：

（1）写 SQL 语句。如查看书的数据量：SELECT COUNT(*) FROM titles。

（2）测试 SQL 语句。执行这些 SQL 语句，确认符合要求。

（3）若得到所需结果，则创建过程。

下面用实例来说明。

例 7-9　创建一存储过程，检索读者（READER）表中的所有记录。该存储过程不带任何输入输出参数。

```
USE BookSys                          --使用 BookSys 数据库
/*判断是否存在存储过程 procReader，存在则删除*/
IF EXISTS(SELECT NAME FROM SYSOBJECTS
          WHERE NAME='procReader')
DROP PROCEDURE procReader
GO
CREATE PROCEDURE procReader       --创建存储过程
AS
SELECT * FROM    READER
```

例 7-10　创建一存储过程，根据传入的读者卡号，检索该读者的借书信息，包括卡号、姓名、书号、书名、借书时间和还书时间。该存储过程带一输入参数：@CardID，即传入一个

读者卡号。

```
USE BookSys                                  --使用 BookSys 数据库
/*判断是否存在存储过程 procReader1，存在则删除*/
IF EXISTS(SELECT NAME FROM SYSOBJECTS
        WHERE NAME='procReader1')
DROP PROCEDURE procReader1
GO
CREATE PROCEDURE procReader1            --创建存储过程
@CardID CHAR(10)
AS
SELECT BORROW.CARDID, READER.[NAME], BORROW.BOOKID,
BOOK.BOOKNAME,BDATE,SDATE
FROM BORROW,BOOK,READER
WHERE BORROW.BOOKID=BOOK.BOOKID
AND BORROW.CARDID=READER.CARDID
AND READER.CARDID=@CardID
```

例 7-11 创建一存储过程，根据转入的卡号（CARDID），判断该卡号是否可以借书：如果该卡号已借图书数量大于或等于所允许的最多借阅量，则返回“不能借”，否则，返回“可以借”（假设学生最多可以借 3 本，教师最多可以借 10 本，临时人员最多可以借 2 本，下同）。

```
USE BookSys                                          --使用 BookSys 数据库
/*判断是否存在存储过程 procIfAllowBorrow，存在则删除*/
IF EXISTS(SELECT NAME FROM SYSOBJECTS
        WHERE NAME='procIfAllowBorrow')
DROP PROCEDURE procIfAllowBorrow
GO
CREATE PROCEDURE procIfAllowBorrow          --创建存储过程
@CARDID CHAR(10),@ReturnInfo VARCHAR(10) OUTPUT
AS
DECLARE @ClassID INT
SELECT @ClassID=CLASS FROM READER
WHERE READER.CARDID=@CARDID
IF @ClassID=1   /*学生*/
BEGIN
    IF EXISTS(SELECT CARDID FROM BORROW
    GROUP BY CARDID HAVING COUNT(CARDID)>=3)
        SET @ReturnInfo='不能借'
    ELSE
        SET @ReturnInfo='可以借'
END
ELSE IF @ClassID=2  /*教师*/
BEGIN
    IF EXISTS(SELECT CARDID FROM BORROW
    GROUP BY CARDID HAVING COUNT(CARDID)>=10)
        SET @ReturnInfo='不能借'
    ELSE
        SET @ReturnInfo='可以借'
```

```
END
ELSE IF @ClassID=3  /*临时人员*/
BEGIN
    IF EXISTS(SELECT CARDID FROM BORROW
    GROUP BY CARDID HAVING COUNT(CARDID)>=2)
        SET @ReturnInfo='不能借'
    ELSE
        SET @ReturnInfo='可以借'
END
RETURN
```

7.2.4　执行存储过程

执行存储过程的完整语法如下：

```
[ EXEC]   [@return_value=]procedure_name [Value_List]
```

其中：

- [@return_value=]用于接收存储过程的返回值。
- procedure_name：要执行的存储过程的名称。
- Value_List：输入参数值。参数之间用逗号分隔，输出参数一定要传变量。

例 7-12　执行例 7-10 的存储过程。假设我们要检索的卡号是 T0001。

```
EXECUTE procReader1 'T0001'
```

执行结果如下：

```
CARDID     NAME    BOOKID      BOOKNAME        BDATE                 SDATE
---------- ------  --------    --------------  ------------          --------
T0001      刘勇    TP2003--002  数据结构        2003-11-18 00:00:00   2003-12-09 00:00:00
（所影响的行数为 1 行）
```

例 7-13　执行例 7-11 的存储过程，查看卡号（CARDID）为 T0001 的读者是否可以借书。

```
DECLARE @V VARCHAR(10)
EXECUTE procIfAllowBorrow 'T0001',@V OUTPUT
SELECT @V
执行结果如下：
----------
可以借
（所影响的行数为 1 行）
```

7.2.5　删除存储过程

删除存储过程是指删除由用户创建的存储过程。

格式：DROP PROCEDURE 存储过程名。

比如删除例 7-9 创建的存储过程：

```
DROP PROCEDURE procReader
```

7.2.6　在 SQL Server Management Studio 中修改存储过程

（1）启动 SQL Server Management Studio 工具。

（2）在“对象资源管理器”中。连接到 SQL Server 2005 数据库引擎实例，展开该实例。再依次展开“数据库”节点→用户数据库（本例为 BookSys）→“可编程性”→“存储过程”，在“存储过程”节点下可以看到用户创建的所有存储过程，选择要修改的存储过程（本例为“procIfAllowBorrow”）。右击该节点，在弹出的菜单中选择“修改”命令（如图 7-1 所示）。

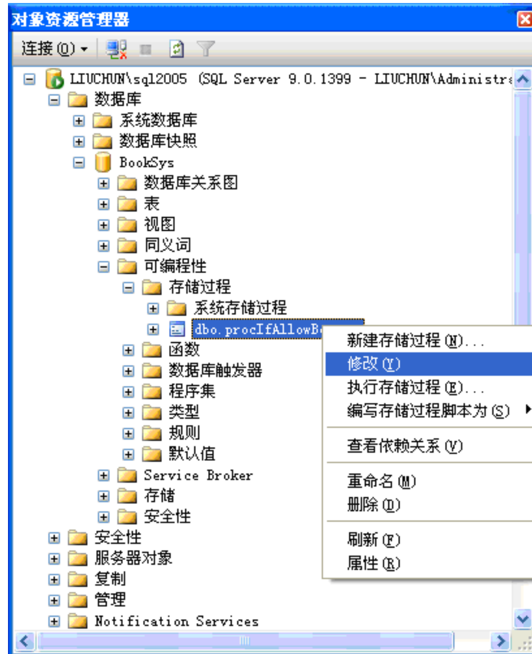

图 7-1 修改存储过程

（3）选择“修改”命令后，系统会打开代码编辑器，显示该存储过程的代码（如图 7-2 所示），用户可以修改并保存。

图 7-2 存储过程代码编辑器

7.2.7　使用 SQL Server Management Studio 中模板新建存储过程

参考图 7-1，直接在“存储过程”节点上右击，在弹出菜单中选择“新建存储过程”命令，系统会打开代码编辑器，并在代码编辑器中显示创建存储过程的模板。用户可以修改模板中的参数并添加相关 Transact-SQL 代码。

7.3　函数

函数使用零个或多个输入值，返回一个数据值或表格形式的一组值。SQL Server 2005 数据库允许用户编写自定义函数，方便业务逻辑实现和代码的可重用性，同时 SQL Server 2005 数据库也提供了许多内置函数供用户使用。

7.3.1　函数类型

SQL Server 2005 数据库中可以有多种函数，根据返回值的类型和是否由系统提供，分为标量函数、表值函数和内置函数。其中标量函数又分为内连标量函数、多语句标量函数，表值函数又分为内连表值函数和多语句表值函数。

（1）内连标量函数：是指返回类型为 RETURN 子句中定义的数据类型的单个值。内连标量函数的函数体是单个 Transact-SQL 语句。

（2）多语句标量函数：是指返回类型为 RETURN 子句中定义的数据类型的单个值。函数体是包含在 BEGIN...END 之间的一组 Transact-SQL 语句。

（3）内连表值函数：返回 TABLE 数据类型，它没有函数体，返回的表是单个 SELECT 语句执行后的返回结果集。

（4）多语句表值函数：返回 TABLE 数据类型，函数体中包含一组 Transact-SQL 语句，这些语句可以生成行，并插入到返回表中。

（5）内置函数：也称系统函数，是 SQL Server 2005 提供的返回标量数据类型或 TABLE 数据类型的函数。内置函数不能修改。

7.3.2　函数的优点

函数实现了模块化程序设计，函数创建后保存在数据库中，用户可以在需要的时候调用，用户定义的函数可以独立于应用程序进行修改。函数和存储过程类似，具有执行速度快、减少网络流量、增强代码的重用性和共享性等优点。

7.3.3　函数与存储过程

实现相同的功能，可以使用函数也可以使用存储这程。而且函数在编写和执行时有着更多的优势。一般来说，如果存储过程返回单个标量值，则使用标量函数更有优势；如果存储过程返回单个结果集，则可以使用表值函数来替代。

7.3.4　用 Transact-SQL 语句创建函数

1. 创建标量函数

```
CREATE  FUNCTION  <函数名>
( [ { 参数名  参数数据类型  [ = 默认值 ] }  [ ,...n ] ]  )
RETURNS  返回的数据类型
  [ WITH  ENCRYPTION  ]
  [ AS ]
  BEGIN
       函数体
    RETURN <表达式>
  END
```

例 7-14　创建内连标量函数，返回给定日期的月信息。

```
CREATE  FUNCTION getMonth        ----函数名
(@date datetime)            ----参数
RETURNS int             --返回类型
 AS
  BEGIN
      RETURN  DATEPART(MM,@date)          ----返回值
  END
```

例 7-15　创建多语句标量函数。计算 1 到给定自然数之间的偶数和。

```
CREATE  FUNCTION  SUMN(@N SMALLINT)
RETURNS SMALLINT
AS
BEGIN
DECLARE @i SMALLINT,  @sum  SMALLINT
SET @i=1
SET @sum=0
WHILE @i<=@N
BEGIN
  IF (@i % 2)=0         ----如果是偶数
    SET @sum=@sum+@i      ----累加
  SET @i=@i+1            -----修改循环变量
END
RETURN @sum             ----返回结果
END
```

2. 创建内连表值函数

```
CREATE  FUNCTION  <函数名>
( [ { 参数名  参数数据类型  [ = 默认值 ] } [ ,...n ] ] )
RETURNS TABLE
  [ WITH  ENCRYPTION ]
  [ AS ]
  RETURN  SELECT 语句
```

例 7-16　用函数返回指定读者的所有借书（未还）记录。

```
 CREATE  FUNCTION getBorrow(@cardid char(14) )
```

```
RETURNS TABLE
 AS
RETURN SELECT * FROM borrow WHERE cardid=@cardid     ----返回结果
```

3. 创建多语句表值函数

```
CREATE   FUNCTION   <函数名>
( [ { 参数名   参数数据类型  [ = 默认值 ] } [ ,...n ] ] )
RETURNS @return_variable TABLE < table_type_definition >
  [ WITH  ENCRYPTION  ]
  [ AS ]
  BEGIN
    函数体
    RETURN
  END
```

例 7-17　以参数方式给定不同类型读者的借书时限（最大可借天数），返回所有借书超期的读者姓名和超期天数。

```
CREATE   FUNCTION getouttime(@s int,@t int,@temp int)
RETURNS @reader TABLE (              ----定义返回表结构
Cardid char(14),
Outdate int
)
AS
BEGIN
INSERT INTO @reader                  ----插入有超期借书的学生
  SELECT NAME, DATEDIFF(DD,BDATE,GETDATE())-@s AS outdate
FROM reader,borrow WHERE reader.cardid=borrow.cardid
AND class=1 AND sdate IS NULL
AND DATEDIFF(DD,bdate,GETDATE())>@s
INSERT INTO @reader                  -----插入有超期借书的教师
  SELECT NAME, DATEDIFF(DD,BDATE,GETDATE())-@t AS outdate
FROM reader,borrow WHERE reader.cardid=borrow.cardid
AND class=2   AND sdate IS NULL
 AND DATEDIFF(DD,bdate,GETDATE())>@t
INSERT INTO @reader                  ----插入有超期借书的临时读者
  SELECT NAME, DATEDIFF(DD,bdate,GETDATE())-@temp AS outdate
FROM reader,borrow WHERE reader.cardid=borrow.cardid
AND class=3   AND sdate IS NULL
AND DATEDIFF(DD,bdate,GETDATE())>@temp
RETURN
END
```

7.3.5　删除函数

删除函数是指删除由用户创建的函数。

格式：DROP　　FUNCTION　<函数名>

比如删除例 7-17 所创建的函数：

DROP FUNCTION getouttime

7.3.6 在 SQL Server Management Studio 中修改函数

（1）启动 SQL Server Management Studio 工具。

（2）在“对象资源管理器”中。连接到 SQL Server 2005 数据库引擎实例，展开该实例。再依次展开“数据库”节点→用户数据库（本例为 BOOKSYS）→“可编程性”→“函数”，根据函数类型选择“表值函数”或“标量值函数”节点，在该节点下可以看到用户创建的所有相关函数，选择要修改的函数（本例为“getouttime”）。右击该函数名，在弹出的菜单中选择“修改”命令（如图 7-3 所示）。

图 7-3 修改函数

（3）选择“修改”命令后，系统会打开代码编辑器，显示该函数的代码，用户可以修改并保存。

7.3.7 函数的调用

1. 标量函数调用

SQL Server 2005 中，标量函数可以直接调用，凡是可以出现表达式的地方都可以调用标量函数。标量函数的调用语法为：

<架构名>.函数名

例如：select dbo.getmonth(getdate())

又如：declare @x int

SET @x= dbo.getmonth(getdate())

2. 表值函数调用

SQL Server 2005 中，表值函数返回的是结果集，在 Transact-SQL 中可以像访问表或视图

一样调用表值函数。

例如：SELECT cardid,bookid FROM getborrow('S0101') WHERE sdate IS NULL

7.3.8　常用内置函数

1．日期与时间函数

日期与时间函数对日期和时间输入值执行操作，并返回一个字符串、数字值或日期和时间值。

（1）GETDATE 函数：GETDATE()没有输入参数，返回当前系统日期和时间。例如：SELECT GETDATE()。

（2）DATEPART 函数：DATEPART(datepart,date)，返回 date 参数指定的日期中的 datepart 参数指定的日期部分的整数。参数 date 为输入的日期，类型为 datetime。参数 datepart 为指定要返回的日期部分的参数。表 7-2 列出了 Microsoft SQL Server 2005 可识别的日期部分及其缩写。

表 7-2　Datepart 的取值和缩写

日期部分	缩写
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw
hour	hh
minute	mi, n
second	ss, s
millisecond	ms

例：SELECT DATEPART(yy,getdate())--返回当前日期中的“年”

（3）DATEDIFF 函数：DATEDIFF (datepart, startdate, enddate)，返回 enddate 和 startdate 表示的两个日期之差，结果由参数 datepart 决定，datepart 的取值参考表 7-2。例如，datepart 取 dd，则计算两个日期相差的天数。

参数 startdate 表示开始日期，enddate 表示结束日期，如果 enddate>startdate，结果为正整数，否则结果为负整数。

例：SELECT datediff(dd,'2007-09-10','2008-05-10')　　--计算 2007 年 9 月 10 日到 2008 年 5 月 10 日之间相差的天数。结果为 243 天。

（4）DATEADD 函数：DATEADD(datepart, number, date)，返回参数 date 指定的日期上增加 number 后得到的新的日期。参数 number 的含义由 datepart 决定，如 datepart 为 dd，则表示在 date 对应的日期上加 number 天后对应的日期。Datepart 的取值参考表 7-3。

（5）YEAR、MONTH、DAY 函数：它们均接收一个日期参数，返回给定日期中的年、月、日。

（6）DATENAME 函数：DATENAME (datepart,date)，返回指定日期的指定日期部分的字符串。

例：SELECT datename(weekday,'2009-2-1')　　---返回结果为“星期日”

2. 字符串函数

SQL Server 2005 提供了大量的操作字符串的函数，如表 7-3 所示。下面详细介绍 Transact-SQL 程序设计中常用的几个字符串函数。

表 7-3　字符串函数

函数	函数	函数
ASCII	NCHAR	SOUNDEX
CHAR	PATINDEX	SPACE
CHARINDEX	QUOTENAME	STR
DIFFERENCE	REPLACE	STUFF
LEFT	REPLICATE	SUBSTRING
LEN	REVERSE	UNICODE
LOWER	RIGHT	UPPER
LTRIM	RTRIM	

（1）CHAR 函数：CHAR(integer_expression)，将整数参数 integer_expression 表示的 ASCII 代码转换为字符。integer_expression 是介于 0～255 之间的整数。如果该整数表达式不在此范围内，将返回 NULL 值。

CHAR 可用于将控制字符插入字符串中。例如在字符串中插入回车符：

```
declare @x char(15)
set @x='abcd'+char(13)+'edf'      ---字符串变量@x 包含“回车”。
print @x
```

（2）LEFT 函数：LEFT (character_expression, integer_expression)，返回字符串中从左边开始指定个数的字符。

（3）RIGHT 函数：RIGHT (character_expression, integer_expression)，返回字符串中从右边开始指定个数的字符。

（4）LEN 函数：LEN (string_expression)，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。

（5）LTRIM 函数：LTRIM (character_expression)，返回删除了前导空格之后的字符串。

（6）RTRIM 函数：RTRIM (character_expression)，截断所有尾随空格后返回一个字符串。

例：删除查询结果中的左右空格。

```
SELECT LTRIM(RTRIM(BOOKNAME)) FROM BOOK
```

（7）STR 函数：STR (float_expression [, length [, 小数位数]])，返回由数字数据转换来的字符数据。

例：SELECT STR(123.45, 6, 1)　----结果为：123.4

（8）SUBSTRING 函数：SUBSTRING (expression,start, length)，取子串，从参数 expression 表示的字符串中的 start 开始，取长度为 length 的子字符串。

例：　select substring('abcde',3,2)　　----结果为“cd”

3．数学函数

SQL Server 2005 提供了大量的数学函数，如表 7-4 所示。

表 7-4　数学函数

函数名	功能	函数名	功能
ABS	返回指定数值表达式的绝对值（正值）的数学函数	DEGREES	返回以弧度指定的角的相应角度
ACOS	返回其余弦是所指定的 float 表达式的角（弧度）；也称为反余弦	EXP	返回指定的 float 表达式的指数值
ASIN	返回以弧度表示的角，其正弦为指定 float 表达式。也称为反正弦	FLOOR	返回小于或等于指定数值表达式的最大整数
ATAN	返回以弧度表示的角，其正切为指定的 float 表达式。它也称为反正切函数	LOG	返回指定 float 表达式的自然对数
ATN2	返回以弧度表示的角，其正切为两个指定的 float 表达式的商。它也称为反正切函数	LOG10	返回指定 float 表达式的常用对数（即以 10 为底的对数）
CEILING	返回大于或等于指定数值表达式的最小整数	PI	返回 PI 的常量值
COS	返回指定表达式中以弧度表示的指定角的三角余弦	POWER	返回指定表达式的指定幂的值
COT	返回指定的 float 表达式中所指定角度（以弧度为单位）的三角余切值	RADIANS	对于在数值表达式中输入的度数值返回弧度值
RAND	返回从 0～1 之间的随机 float 值	SQRT	返回指定表达式的平方根
ROUND	返回一个数值表达式，舍入到指定的长度或精度	SQUARE	返回指定表达式的平方
SIGN	返回指定表达式的正号（+1）、零（0）或负号（-1）。	TAN	返回输入表达式的正切值
SIN	以近似数字（float）表达式返回指定角度（以弧度为单位）的三角正弦值		

7.4　触发器

触发器是一种特殊的存储过程，当在指定的数据表进行插入、修改或删除行操作时被自动调用。触发器为数据提供了有效的监控和处理机制，确保数据和业务的完整性。SQL Server 2005 数据库在传统的触发器基础上进行了扩展，实现了对数据库结构操作时的触发机制（DDL 触发器）。

7.4.1 触发器概述

1．触发器的概念

触发器（Trigger）是用户对某一表中的数据做插入、更新和删除操作时被触发执行的一段程序，通常我们使用触发器来检查用户对表的操作是否符合整个应用系统的需求以及是否符合商业规则，以维持表内数据的完整性和正确性。

触发器和存储过程一样是由 Transact-SQL 语句写成的程序。存储过程是由用户利用 EXECUTE 命令执行它，但触发器是在用户对触发对象进行操作时被触发执行的。

触发器的功能和表内设置的一些列限制（Constraints）有些重叠，事实上如果列限制的功能能够达到应用程序要求的话，应该无须用额外的触发器来做相关的工作。不过触发器可在需要参考到其他数据库内的数据来做检查时使用，另外，如果要做比较复杂的安全措施，例如：将操作某一表的用户的名字和时间记录到另外一表的话，使用列限制的方式就无法做到，而触发器能做到。

2．与触发器有关的两个特殊表

SQL Server 为每个触发器创建了两个专用临时表：INSERTED 表和 DELETED 表。这是两个逻辑表，由系统来维护，不允许用户直接对这两个表进行修改。它们存放于内存中，而不是存放在数据库中。这两个表的结构总是与被该触发器作用的表的结构相同。触发器工作完成后，与该触发器相关的这两个表也会被删除。

（1）INSERTED 表：存放由于 INSERT 或 UPDATE 语句的执行而导致要加到该触发器作用的表中去的所有新行。即把插入或更新表的新行值，在插入或更新表的同时，也将其副本存入 INSERTED 表中。

（2）DELETED 表：存放由于 DELETE 或 UPDATE 语句的执行而导致要从被该触发器作用的表中删除的所有行。即把被作用表中要删除或要更新的旧值移到 DELETED 表中。

对于 INSERT 操作，只在 INSERTED 表中保存插入行的新值，而 DELETED 表中无数据。对于 DELETE 操作，只在 DELETED 表中保存被删除行的旧值，而 INSERTED 表中无数据。对于 UPDATE 操作，可以将它考虑为 DELETE 操作和 INSERT 操作的结果，所以在 INSERTED 表中存放着更新后的新值，DELETED 表中存放着更新前的旧值。

3．触发器的分类

SQL Server 2005 提供两大类触发器：DML 触发器和 DDL 触发器。

（1）DML 触发器：DML 触发器是当数据库中发生数据操纵语言（DML）事件时要执行的操作。DML 事件包括对表或视图发出的 INSERT、UPDATE、DELETE 语句。DML 触发器包括两种类型：AFTER 触发器和 INSTEAD OF 触发器。下面将会对这两种触发器作详细介绍。

（2）DDL 触发器：DDL 触发器是 SQL Server 2005 新增的功能。它是一种特殊的触发器，在响应数据定义语言（DDL）时触发。

4．创建触发器的 Transact-SQL 语句

```
CREATE TRIGGER <触发器名>
ON <表名 | 视图名 >
[WITH ENCRYPTION]
<FOR | AFTER | INSTEAD OF> < [ INSERT ] [ , ] [ UPDATE ] [ , ] [DELETE]>
```

```
AS
        [IF UPDATE (列名)]
        <SQL 语句 [ ...n ]>
```

其中：

WITH ENCRYPTION 选项加密触发器代码，保护作者的知识产权。

UPDATE (列名)是一个函数，用于测试是否对表或视图的指定列进行了 INSERT 或 UPDATE 操作。

触发器可以响应更新、插入与删除。可以对两个或三个这类操作建立相同的触发器。例如，可以对删除和更新生成触发器。

在 SQL Server 2005 中，FOR 和 AFTER 的作用相同，FOR 是为了保持与以前版本兼容，建议使用 AFTER。SQL Server 7.0 不支持 AFTER，只能使用 FOR。

触发器与表相关联。如果删除表，则与这个表相关联的所有触发器都将被删除。

7.4.2 AFTER 触发器

AFTER 触发器是在执行 INSERT、UPDATE、DELETE 任一操作之后被触发。AFTER 触发器只能在表上定义。可以针对表的同一操作定义多个触发器，也可以针对多个操作定义同一触发器。

1．INSERT 触发器

INSERT 触发器由 INSERT 语句触发，即用户在表中插入一条记录且插入成功时，触发 INSERT 触发器。

例 7-18 创建一触发器以实现如下功能：当往 BORROW 表中插入一条记录时，如果书号或卡号不存在，则撤消插入。

```
USE BookSys                              --使用 BookSys 数据库
/*判断是否存在触发器 t_BORROW1，存在则删除*/
IF EXISTS(SELECT NAME FROM SYSOBJECTS
        WHERE NAME='t_BORROW1')
DROP TRIGGER t_BORROW1
GO
CREATE TRIGGER t_BORROW1                 --创建触发器
ON BORROW
AFTER INSERT                             --INSERT（插入）触发器
AS
    DECLARE @BOOKID CHAR(20)             --声明一个变量以存储书号
    DECLARE @CARDID CHAR(10)             --声明一个变量以存储卡号
    /*从临时表中检索出新记录的书号和卡号*/
    SELECT @BOOKID=BOOKID,@CARDID=CARDID FROM INSERTED
    /*如果书号在其父表中不存在或者卡号在其父表中不存在*/
    IF (NOT EXISTS(SELECT * FROM BOOK WHERE BOOKID=@BOOKID))
OR (NOT EXISTS(SELECT * FROM READER WHERE CARDID=@CARDID))
    BEGIN
     ROLLBACK TRANSACTION                --撤消插入到表中的记录
     RAISERROR('书号或卡号不存在！',16,1)
END
```

INSERT 触发器的工作过程如下：

（1）用户或系统运行 INSERT 语句。

（2）如果记录不违反限制，则将记录插入到临时表 INSERTED 中。

（3）触发触发器。

（4）如果触发器执行完毕而无错误，则 INSERTED 表被删除，插入操作完成。

注意：如果在触发器中检测到插入的数据不符合要求，可以用 ROLLBACK TRANSACTION 语句取消插入操作。如果触发器出现异常，但没有执行 ROLLBACK TRANSACTION 语句，已插入的数据不会回滚。

2．DELETE 触发器

DELETE 触发器的过程与 INSERT 触发器相似，只是使用的是 DELETED 表，其中包含刚刚删除的记录，而不是 INSERTED 表。

例 7-19 创建一触发器以实现如下功能：当试图删除 BORROW 表中的一条记录时，若还书日期为空或还书日期距今还不到半年，则撤消事务。我们基于这样的假设：如果某书尚未归还，显然不能删除；另外，我们希望将读者的还书记录保留半年之久。

```
USE BookSys                          --使用 BookSys 数据库
/*判断是否存在触发器 t_BORROW2，存在则删除*/
IF EXISTS(SELECT NAME FROM SYSOBJECTS
        WHERE NAME='t_BORROW2')
DROP TRIGGER t_BORROW2
GO
CREATE TRIGGER t_BORROW2             --创建触发器
ON BORROW
AFTER DELETE                         --DELETE（删除）触发器
AS
DECLARE @SDATE SMALLDATETIME         --声明一个变量以存储还书日期
/*从临时表中检索出被删除记录的还书日期至变量中*/
SELECT @SDATE=SDATE FROM DELETED
/*如果还书日期为空或者还不到半年*/
IF (@SDATE IS NULL) OR DATEDIFF(MM,@SDATE,GETDATE())>6
BEGIN
    RAISERROR('不允许删除这条记录，因为读者尚未还书，或还书还不到半年',16,1)
    ROLLBACK TRANSACTION             --撤消从表中删除的记录
END
```

DELETE 触发器的工作过程如下：

（1）用户或系统运行 DELETE 语句。

（2）如果记录不违反外部关键字限制，则删除表中的记录并将其插入到临时表 DELETED 中。

（3）触发触发器。

（4）如果触发器执行完毕而无错误，则 DELETED 表被删除，删除操作完成。

3．UPDATE 触发器。

更新操作可以看成一个删除加一个插入：删除旧值和插入新值。

例 7-20 创建一触发器以实现如下功能：如果要更改 READER 表中的 CARDID，则先检

查 BORROW 表中是否有记录引用了该 CARDID。如果有引用，则不能更改；如果没有引用，则可以更改。

```
USE BookSys                    --使用 BookSys 数据库
/*判断是否存在触发器 t_READER1，存在则删除*/
IF EXISTS(SELECT NAME FROM SYSOBJECTS
        WHERE NAME='t_READER1')
DROP TRIGGER t_READER1
GO
CREATE TRIGGER t_READER1
ON READER
AFTER UPDATE
AS
IF UPDATE(CARDID)      ----如果 CARDID 列被修改
BEGIN
    /*如果 BORROW 表中存在该 CARDID*/
    IF EXISTS(SELECT BORROW.CARDID FROM BORROW, DELETED
    WHERE BORROW.CARDID=DELETED.CARDID)
      BEGIN
        ROLLBACK TRANSACTION
        RAISERROR('该卡号正在使用，不能更改',16,1)
      END
END
```

UPDATE 触发器的工作过程如下：

（1）用户或系统运行 UPDATE 语句。

（2）如果记录不违反限制，则更新表并把旧记录插入到 DELETED 表中，把新记录插入到 INSERTED 表中。

（3）触发触发器。

（4）如果触发器执行完毕而无错误，则 DELETED 表和 INSERTED 表被删除，更新操作完成。

7.4.3　INSTEAD OF 触发器

SQL Server 2005 开始引入了 INSTEAD OF 触发器。AFTER 触发器的一个主要缺点是发生在触发的语句执行之后。从前面的例子中可以看出，如果触发器出现异常或更新操作（插入、更新、删除）不满足触发器中定义的规则，则要撤消事务。

INSTEAD OF 触发器可以对表或视图生成，但表或视图中的每个操作只能有一个 INSTEAD OF 触发器。

对于视图，INSTEAD OF 触发器改进视图的可更新性。我们看到，视图只能一次更新、插入或删除一个基表数据。使用 INSTEAD OF 触发器则可以克服这个限制。但 INSTEAD OF 触发器不可以用于使用 WITH CHECK OPTION 的可更新视图。

1．INSTEAD OF INSERT 触发器

与 AFTER 触发器一样，INSTEAD OF INSERT 触发器也使用 INSERTED 表，但逻辑稍有不同。

（1）用户或系统运行 INSERT 语句。

（2）记录只插入到 INSERTED 表中。

（3）如果记录不违反任何限制，则触发器负责将记录实际地插入到数据表中，否则不插入到表中。

注意：第一，记录只插入 INSERTED 表中而不插入基表中。因此，如果触发器中的任何测试失败，什么也不用撤消。第二，表中并没有真正发生插入。触发器可以在测试数值之后决定插入记录，但如果触发器代码进行其他操作，则并不进行插入。触发器执行，而基础 INSERT 语句并不执行。

例 7-21 下面的这个触发器用 INSTEAD OF 触发器替代了例 7-17 的 AFTER 触发器。

```
USE BookSys                                   --使用 BookSys 数据库
/*判断是否存在触发器 t_BORROW3，存在则删除*/
CREATE TRIGGER t_BORROW3                      --创建触发器
ON BORROW
INSTEAD OF INSERT
AS
DECLARE @BOOKID CHAR(20)
DECLARE @CARDID CHAR(10)
SELECT @BOOKID=BOOKID,@CARDID=CARDID FROM INSERTED
/*如果 CARDID 和 BOOKID 在各自的父表中都存在*/
IF (EXISTS(SELECT * FROM BOOK WHERE BOOKID=@BOOKID)) AND
(EXISTS(SELECT * FROM READER WHERE CARDID=@CARDID))
     /*由触发器负责插入新记录*/
     INSERT INTO BORROW SELECT   * FROM INSERTED
ELSE
     RAISERROR('书号或卡号不存在！',16,1)
```

从中可以看出它的一大优点：如果不存在卡号或书号，不需要撤消事务。

2. INSTEAD OF DELETE 触发器

INSTEAD OF DELETE 触发器和 INSTEAD OF INSERT 触发器相似，但使用 DELETED 表。

（1）用户或系统运行 DELETE 语句。

（2）要删除记录的一个副本拷贝到 DELETED 表中。

（3）如果记录不违反任何限制，则触发器负责将记录从表中删除，否则不删除。

和 INSTEAD OF INSERT 触发器相似，除了触发器的操作以外，表格不进行任何操作。

注意：FOREIGN KEY 限制定义了 ON DELETE CASCASE 选项的表不能定义 INSTEAD OF DELETE 触发器。

例 7-22 下面的这个触发器用 INSTEAD OF DELETE 触发器替代了例 7-18 的 AFTER 触发器。

```
USE BookSys                                   --使用 BookSys 数据库
/*判断是否存在触发器 t_BORROW4，存在则删除*/
IF EXISTS(SELECT NAME FROM SYSOBJECTS
          WHERE NAME='t_BORROW4')
DROP TRIGGER t_BORROW4
GO
```

```
CREATE TRIGGER t_BORROW4              --创建触发器
ON BORROW
INSTEAD OF DELETE
AS
DECLARE @BOOKID CHAR(20)
DECLARE @CARDID CHAR(10)
DECLARE @SDATE SMALLDATETIME
SELECT @BOOKID=BOOKID,@CARDID=CARDID,

    @SDATE=SDATE FROM DELETED
/*如果还书日期已超过半年*/
IF DATEDIFF(MM,@SDATE,GETDATE())>6
    /*由触发器负责删除该条记录*/
    DELETE FROM BORROW
    WHERE BOOKID=@BOOKID AND CARDID=@CARDID
ELSE
    RAISERROR('不允许删除这条记录，因为读者尚未还书，或还书尚不到半年',16,1)
```

3. INSTEEAD OF UPDATE 触发器

INSTEAD OF UPDATE 触发器用 DELETED 表和 INSERTED 表存储更新前后的记录。基表不进行任何数据修改。其逻辑如下：

（1）用户或系统运行 UPDATE 语句。

（2）旧记录插入到 DELETED 表中，新记录插入到 INSERTED 表中。

（3）如果记录不违反任何限制，则由触发器负责更新基表中的记录。

注意：FOREIGN KEY 限制定义了 ON UPDATE CASCASE 选项的表不能定义 INSTEAD OF UPDATE 触发器。

例 7-23　下面的这个触发器用 INSTEAD OF UPDATE 触发器替代了例 7-19 的 AFTER 触发器。

```
USE BookSys                           --使用 BookSys 数据库
/*判断是否存在触发器 t_READER2，存在则删除*/
IF EXISTS(SELECT NAME FROM SYSOBJECTS
        WHERE NAME='t_READER2')
DROP TRIGGER t_READER2
GO
CREATE TRIGGER t_READER2
ON READER
INSTEAD OF UPDATE
AS
IF UPDATE(CARDID)
BEGIN
    DECLARE @newCARDID CHAR(10)
    DECLARE @oldCARDID CHAR(10)
    IF NOT EXISTS(SELECT BORROW.CARDID FROM BORROW,DELETED
    WHERE BORROW.CARDID=DELETED.CARDID)
    BEGIN
```

```
            SELECT @newCARDID=CARDID FROM INSERTED

            SELECT @oldCARDID=CARDID FROM DELETED
            UPDATE READER SET CARDID=@newCARDID
            WHERE CARDID=@oldCARDID
        END
        ELSE
            RAISERROR('该卡号正在使用，不能更改',16,1)
    END
```

7.4.4 在 SQL Server Management Studio 中修改触发器

（1）启动 SQL Server Management Studio 工具。

（2）在“对象资源管理器”中。连接到 SQL Server 2005 数据库引擎实例，展开该实例。再依次展开“数据库”→用户数据库（本例为 BOOKSYS）→“表”节点。选择要修改触发器的表并展开，再展开“触发器”节点，在该节点下可以看到用户创建的与该表相关的触发器，选择要修改的触发器（本例为“t_BORROW2”）。右击该触发器，在弹出的菜单中选择“修改”命令（如图 7-4 所示）。

（3）选择“修改”命令后，系统会打开代码编辑器，显示该函数的代码，用户可以修改并保存。

图 7-4　修改触发器