第12章数据库编程

北京理工大学 计算机学院
张文耀
zhwenyao@bit.edu.cn

主要内容

12.1 Transact-SQL

12.2 Transact-SQL游标

12.3 Transact-SQL存储过程

12.4 Transact-SQL用户定义函数

12.5 Transact-SQL触发器

12.1 Transact-SQL

- 标准SQL语言是非过程的，不能象其它高级语言那样编写可以控制流程（条件判断、循环等）和具有一定结构（函数、子程序等）的程序；
- 每种数据库产品都会对标准SQL语言进行一定的扩充以支持结构化程序的某些特征；
- **Oracle的PL/SQL；**
- **SQL Server的Transact-SQL。**

- **Transact-SQL**语言的主要特点如下：
  - 是一种交互式查询语言，功能强大，简单易学；
  - 既可以直接查询数据库，也可以嵌入到其它高级语言中执行；
- **Transact-SQL**不仅支持所有的**SQL**语句，而且还提供了丰富的编程功能，允许使用变量、运算符、函数、流程控制语句等。

Transact-SQL元素

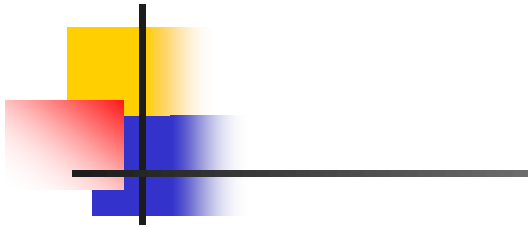
- **Transact-SQL元素**
  - 标识符
  - 数据类型
  - 常量
  - 函数
  - 表达式
  - 注释
  - 保留关键字

标识符

- 常规标识符
  ——符合一定命名规则的标识符
  - 第一个字符必须是字母，a～z 或 A～Z
  - 第一个字符后可以是数字、字母或各种符号
  - 当标识符的第一个字符是符号时，代表它有特殊用处
    - 以@开头的标识符代表局部变量或参数
    - 以@ @开头的标识符代表全局变量
    - 以#开头的标识符代表临时表或存储过程
    - 以##开头的标识符代表一个全局临时对象

- 分隔标识符
  - 所有不符合标识符规则的标识符必须进行分隔
    - 当对象名称包含空格时
    - 当保留关键字被用作对象名或对象部分的名字时
  - 分隔方法
    - 用中括号（[ ]）进行分隔
    - 用双引号（“ ”）进行分隔

```
SELECT * FROM [My Table] WHERE [order] = 10
--标识符包含一个空格并使用了关键字。
```



- 关于**SQL**的标识符说明
  - 数据库对象的名称就是其标识符
  - **SQL Server**内容都可以有标识符
  - 大多数对象要求有标识符
  - 有些对象的标识符是可选的，例如约束

数据类型

- 数据类型是指列、存储过程的参数、表达式和局部变量的数据特征，它决定了数据的存储格式，代表了不同的信息类型。
- **SQL Server**提供了多种系统数据类型，还可以自定义数据类型。以下对象可以具有数据类型
  - 表和视图中的列、存储过程中的参数
  - 变量
  - 返回一个或多个特定数据类型数据值的函数
  - 具有一个返回代码的存储过程

- 为对象分配数据类型时，可为对象定义如下属性：
  - 对象包含的数据种类
  - 所存储值的长度或大小
  - 数值的精度（仅适用于数字数据类型）
  - 数值的小数位数（仅适用于数字数据类型）
- 系统提供的数据类型

bigint	binary	bit	char	cursor
datetime	decimal	float	image	int
money	nchar	ntext	numeric	nvarchar
real	smalldatetime	smallint	smallmoney	sql_variant
table	text	timestamp	tinyint	varbinary
varchar	uniqueidentifier	xml		

bigint	占8个字节，值的范围为 -2^{63}～$2^{63}-1$
int	占4个字节，值的范围为 -2^{31}～$2^{31}-1$
smallint	占2个字节，值的范围为 -32768～32 767
tinyint	占1个字节，值的范围为 0～255

decimal [(p[,s])]	p为精度，最大38； s为小数位数，0≤s≤p
numeric [(p[,s])]	在SQL Server中，等价于decimal

float [(*n*)]	从 -1.79E+308 到 1.79E+308 之间的浮点数字数据；*n* 为用于存储科学记数法尾数的位数，同时指示其精度和存储大小，1≤*n*≤53
real	从 3.40E+38 到 3.40E+38 之间的浮点数字数据，存储大小为 4 字节；SQL Server 中，real 的同义词为 float(24)

money	占8个字节，值的范围为 -922 337 203 685 477.580 8 ～ +922 337 203 685 477.580 7
smallmoney	占4个字节，值的范围为 -214 748.3648 ～ 214 748.3647

datetime	占8个字节，表示从1753年1月1日到9999年12月31日的日期
smalldatetime	占4个字节，表示从1900年1月1日至2079年6月6日的日期

char [(*n*)]	字符个数 0～8 000
varchar [(*n*)]	字符个数0～8 000
text	字符个数0～2GB
nchar [(*n*)]	字符个数0～4 000
nvarchar[(*n*)]	字符个数0～4 000
ntext	字符个数0～1GB

binary[(*n*)]	存储字节个数 0～8 000
varbinary[(*n*)]	存储字节个数 0～8 000
image	存储字节个数 0～2 G

bit	存储位数据
cursor	存储对游标的引用
rowversion (timestamp)	时间戳
sql_variant	可存储除 **text**、**ntext**、**image**、**rowversion** 之外的其他类型
table	存储函数返回结果
uniqueidentifier	存储 **GUID** 以及 **UUID**

- 选择数据类型的指导原则
  - 若列值的长度相差很大，那么使用变长数据类型；
  - 谨慎使用 **tinyint** 数据类型，虽然节省空间，但扩展性很小；
  - 对于小数数据来说，一般使用 **decimal** 数据类型，可以精确地控制精度；
  - 如果行的存储量超过**8000**字节，使用 **text** 或**image**；若不大于**8 000**字节，可使用**char**、**varchar**或者**binary**数据类型；
  - 对于货币数据，使用 **money** 数据类型；
  - 不要使用类型为 **float** 或者 **real** 的列作为主键，因为它们不精确，不适合用于比较。

- **SQL Server**提供了用户自定义数据类型的功能
  - 用户定义的数据类型与系统数据类型一样，都是用来限制操作者输入数据的种类和长度。可以在使用系统数据类型的任何地方使用自定义数据类型。
  - 用户定义的数据类型是与表、视图等并列的数据库对象，可以对它执行创建、修改、删除等操作。
- 用户定义的数据类型可以实现以下功能：
  - 可以让不同表中重复出现的各列具有相同的特性，使相似的数据种类标准化。
  - 可以将规则和默认值捆绑到用户定义的数据类型上，以约束使用此数据类型的每个列。

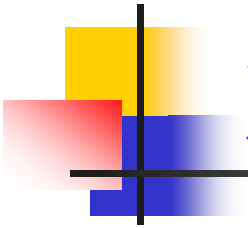
常量

- 常量是表示特定数据值的符号。
- 常量的格式取决于它所表示的值的数据类型
- 常量示例

```
SELECT * FROM emp WHERE name = 'O''Brien';
SELECT * FROM emp WHERE name = ''O’Brien'';
```

常量类型	示例
字符串	**'O''Brien'**
Unicode 字符串	**N'Michl'**
二进制字符串常量	**0x12Ef**
bit 常量	**0** 或 **1**
datetime 常量	**'April 15, 1998'**、**'04/15/98'**、**'04:24 PM'**
integer 常量	**1894**
decimal 常量	**1894.1204** 、**2.0**
float 和 **real** 常量	**0.5E-2**
money 常量	**$542023.14**
uniqueidentifier 常量	**0xff19966f868b11d0b42d00c04fc964ff**

Unicode 字符串的格式与普通字符串相似，但它前面有一个 N 标识符（N 代表 SQL-92 标准中的国际语言 (National Language)）。N 前缀必须是大写字母。



函数

- **Transact-SQL**中提供了大量函数
- 使用函数的例子
  - **SELECT AVG(salary) FROM emp;**
  - **SELECT DB_NAME();**

聚合函数	执行的操作是将多个值合并为一个值。
配置函数	是一种标量函数，可返回有关配置设置的信息。
加密函数	支持加密、解密、数字签名和数字签名验证。
游标函数	返回有关游标状态的信息。
日期和时间函数	可以更改日期和时间的值。
数学函数	执行三角、几何和其他数字运算。
元数据函数	返回数据库和数据库对象的属性信息。
排名函数	是一种非确定性函数，可以返回分区中每一行的排名值。
行集函数	返回可在 **Transact-SQL** 语句中表引用所在位置使用的行集。
安全函数	返回有关用户和角色的信息。
字符串函数	可更改 **char**、**varchar**、**nchar**、**nvarchar**、**binary** 和 **varbinary** 的值。
系统函数	对系统级的各种选项和对象进行操作或报告。
系统统计函数	返回有关 **SQL Server** 性能的信息。
文本和图像函数	可更改 **text** 和 **image** 的值。

表达式

- 表达式是标识符、值和运算符的组合，SQL Server可以对其求值以获取结果。
- 运算符
  - Transact-SQL提供了如下几种类型的运算符：算术运算符、比较运算符、字符连接运算符和逻辑运算符
  - 字符连接运算符
    - 连接运算符：+，其用于将两个字符数据连接起来。
  - 逻辑运算符
    - 逻辑运算符一般用于条件判断

- 算术运算符

运算符	含义	运算符	含义
+	加	/	除
-	减	%	取模
*	乘		

- 比较运算符

运算符	含义	运算符	含义
=	等于	!>	不大于
>	大于	!<	不小于
<	小于	<>, !=	不等于
>=	大于或等于	()	控制实行优先级
<=	小于或等于		

注释

- 插入到 **Transact-SQL** 语句或脚本中、用于解释语句作用的文本段。
- **SQL Server 2005** 支持两种类型的注释字符：
  - 行注释--
  - 块注释/* ... */

保留关键字

- 保留下来供 **SQL Server** 使用的词
- 不应用做数据库中的对象名

Transact-SQL的过程

- 若要执行单个**Transact-SQL** 语句无法完成的任务，可以将 **Transact-SQL** 语句以多种方式组合在一起
  - 批处理
  - 脚本
  - 存储过程
  - 触发器

批处理

- 批处理是作为一个单元从应用程序发送到服务器的一组 **Transact-SQL** 语句。
- **SQL Server**将每个批处理作为一个可执行单元来执行。
- **SQL**将批处理语句作为一个整体编译为一个执行计划，一起提交给服务器，可以节省系统开销。
- 在查询分析器中，可以用**GO**命令标志一个批处理的结束。
  - **GO**不是一个执行语句，是通知查询分析器有多少语句要包含在当前的批处理中。
  - 查询分析器将两个**GO**之间的语句组成一个字符串交给服务器去执行

- 批处理有如下限制：
  - 一个批处理本身应该是完整的，不能在一个批处理中引用其它批处理定义的变量，也不能将注释从一个批处理开始，在另一个批处理中结束。
  - 如果批处理中的语句出现编译错误，将不能生成执行计划，批处理中的任何一个语句都不会执行
  - **CREATE DEFAULT、CREATE PROCEDURE、CREATE RULE、CREATE TRIGGER、CREATE VIEW**语句不能与其它语句位于同一个批处理中。
  - 不能在一个批处理中修改一个表的结构，然后在同一个批处理中引用刚修改的新列。

脚本

- 脚本是存储在文件中的一系列 **Transact-SQL** 语句。
- **Transact-SQL** 脚本包含一个或多个批处理。
  - **GO** 命令表示批处理的结束。
  - 如果 **Transact-SQL** 脚本中没有 **GO** 命令，那么它将被作为单个批处理来执行。
- 如果需要经常执行一段**Transact-SQL**语句或是要进行调试（可能需要反复修改），将执行的**T-SQL**语句保存到一个脚本文件中可以节省时间。

存储过程和触发器

- 存储过程
  - 在服务器上预定义并预编译的一组 **Transact-SQL** 语句
  - 可以接受参数，并可以将结果集、返回代码和输出参数返回给调用的应用程序。
  - 一次编译，多次执行；可以提高性能。
- 触发器
  - 一种特殊的存储过程；
  - 不由用户直接调用，而是在对特定的表或列进行特定的数据修改时触发。

Transact-SQL的变量和参数

- **Transact-SQL**中的变量分为局部变量与全局变量
- 局部变量
  - 局部变量是用户自定义的变量。
  - 使用范围是定义它的批处理、存储过程或触发器。
  - 局部变量前面通常加上@标记。
  - 用**DECLARE** 定义局部变量，并指明此变量的数据类型
  - 用**SET**或**SELECT**命令对其赋值。
  - 局部变量的数据类型可以是用户自定义的数据类型，也可以是系统数据类型，但不能将其定义为**TEXT**或**IMAGE**数据类型。

- 定义局部变量的语法如下：

  DECLARE @local_variable data_type

  [, @local_variable data_type]…

  - **DECLARE**命令可以定义多个局部变量；多个局部变量之间用逗号分隔
- 用SET为局部变量赋值常用语法格式为：

  SET @local_variable= expression

- 用**SELECT**为局部变量赋值的语法如下：

```
SELECT @variable_name=expression select statement
        [, @variable_name=expression select statement]
        [FROM list of tables]
        [WHERE expression]
        [GROUP BY... ]   [HAVING... ]   [ORDER BY]
```

  - **SELECT**命令可以将一个表达式的值赋给一个局部变量，也可以将一个**SELECT**查询的结果赋给一个局部变量

【例】使用局部变量。

```
USE MyDb;
GO
DECLARE @no varchar(10);
SET @no='2004060003';
SELECT 编号,姓名
    FROM readers
    WHERE 编号= @no;
```

- 全局变量
  - 由**SQL Server**系统提供并赋值的变量。
  - 全局变量是在服务器级定义的，不是由用户例程定义的。用户不能建立全局变量，也不能修改全局变量的值。
  - 用户只能使用系统预定义的全局变量。
  - 引用全局变量时，前面一定加上@@标记。
  - 用户不能定义与系统全局变量同名的局部变量，否则将产生不可预测的结果。

【例】@@ROWCOUNT，返回最近一次数据库操作所涉及到的行数。

```
UPDATE Readers SET  已借数量=2;
SELECT @@ROWCOUNT AS 行数;
GO
```

执行结果为

```
行数
 5
```

- 参数
  - 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。
  - 可以是输入参数，也可以是输出参数。

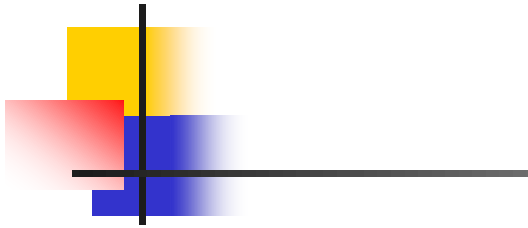
【例】 存储过程的参数

```
USE AdventureWorks;
GO
/* 存储过程ParmSample的参数@EmpIDParm */
CREATE PROCEDURE ParmSample @EmpIDParm int AS
    SELECT EmployeeID, Title
        FROM HumanResources.Employee
        WHERE EmployeeID = @EmpIDParm
GO
/* 批处理中通过参数传递值给存储过程 */
EXEC ParmSample @EmpIDParm = 109
GO
```

Transact-SQL的控制流程

- **Transact-SQL的流程控制关键字**

BEGIN...END	**BEGIN 和 END 语句用于将多个 Transact-SQL 语句组合为一个逻辑块。**
GOTO	**GOTO 语句使 Transact-SQL 批处理的执行跳至标签。**
IF...ELSE	**IF 语句用于条件的测试。得到的控制流取决于是否指定了可选的 ELSE 语句。**
RETURN	**RETURN 语句无条件终止查询、存储过程或批处理。**
WAITFOR	**WAITFOR 语句挂起批处理、存储过程或事务的执行**
WHILE	**只要指定的条件为 True 时，WHILE 语句就会重复语句或语句块。**
CONTINUE	**重新开始 WHILE 循环。**
BREAK	**终止WHILE循环。**
CASE	**CASE 函数用于计算多个条件并为每个条件返回单个值。**



- 控制流语句不能跨多个批处理、用户定义函数或存储过程。

Transact-SQL的错误处理

- **Transact-SQL使用RAISERROR语句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；
  - 可以动态建立消息；
  - 可以使用局部变量为 **RAISERROR** 语句提供消息文本。

【例】使用RAISERROR建立错误消息

```
RAISERROR (N'<<%*.*s>>', -- 信息.
        10, -- 错误级别,
        1, -- 状态,
        7, -- 第4个参数用于宽度.
        3, -- 用于精度的参数.
        N'abcde'); -- 第6个参数用于字符串.
-- 返回的信息为: <<    abc>>.
GO
```

【例】使用局部变量提供错误消息

```
DECLARE @StringVariable NVARCHAR(50);
SET @StringVariable = N'<<%7.3s>>';
RAISERROR (@StringVariable, -- 信息.
        10, -- 错误级别,
        1, -- 状态,
        N'abcde'); -- 提供字符串的参数.
-- 返回的信息为: <<    abc>>.
GO
```

12.2 Transact-SQL的游标

- 游标是标准**SQL**的内容，但是不同数据库的游标有一定的区别
- 游标的作用：处理多行数据
- 游标包括
  - 游标结果集：由定义该游标的**SELECT**语句返回的行的集合
  - 游标位置：指向这个集合中某一行的指针
- **Transact-SQL** 游标主要用在存储过程、触发器和脚本中。

- 使用游标的典型步骤

1. 用**DECLARE CURSOR** 定义 **Transact-SQL** 服务器游标，以及相应的特性，例如游标的滚动行为；
2. 用**OPEN**语句打开游标，执行**SELECT**语句，填充结果集；
3. 用**FETCH** 语句提取单个行，并把每列中的数据转移到指定的变量中；
4. 用**CLOSE**关闭游标，释放某些资源，比如游标结果集和对当前行的锁定；
5. 用**DEALLOCATE** 语句完全释放分配给游标的资源。

游标示例

```
USE AdventureWorks;
GO
-- （1）声明变量以存储由FETCH提取的值.
DECLARE @LastName varchar(50), @FirstName varchar(50);
-- （2）声明游标contact_cursor
DECLARE contact_cursor CURSOR FOR
   SELECT LastName, FirstName FROM Contact
       WHERE LastName LIKE 'B%'
       ORDER BY LastName, FirstName;
-- （3）打开游标
OPEN contact_cursor;
-- （4）执行第一次提取并将值存储在变量中.
-- 变量的顺序必须与游标SELECT语句中列的个数和顺序相同
FETCH NEXT FROM contact_cursor
     INTO @LastName, @FirstName;
```

```
-- （5）检查 @@FETCH_STATUS 判断提取操作是否成功
WHILE @@FETCH_STATUS = 0
  BEGIN
    -- （6）连接并显示变量中当前值.
    PRINT 'Contact Name: ' + @FirstName + ' ' +
          @LastName;
    -- （7）当前一个提取成功时执行下一个提取.
    FETCH NEXT FROM contact_cursor
          INTO @LastName, @FirstName;
  END
-- （8）关闭并删除游标
CLOSE contact_cursor;
DEALLOCATE contact_cursor;
```

12.3 Transact-SQL存储过程

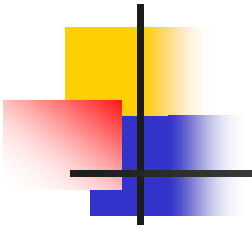
- 什么是存储过程
  - 在服务器上预定义并预编译的一组 **Transact-SQL** 语句
  - 存储在数据库中的一段程序代码
- **SQL Server**提供了一种方法，可以将一些固定的操作集中起来由**SQL Server**数据库服务器来完成，以实现某个任务，这种方法就是存储过程。
- 存储过程存储在数据库内，可由应用程序通过一个调用执行，而且允许用户声明变量、有条件执行，以及其它编程功能。

- 存储过程的作用
  - 接受输入参数并以输出参数的格式向调用过程或批处理返回多个值。
  - 包含用于在数据库中执行操作（包括调用其他过程）的编程语句。
  - 向调用过程或批处理返回状态值，以指明执行成功或失败（以及失败的原因）。
  - 可以使用 **Transact-SQL EXECUTE** 语句执行存储过程。存储过程与函数不同，因为存储过程不返回取代其名称的值，也不能直接在表达式中使用。

- 存储过程的优点
  - 由于在执行前已经存储在数据库中，使用存储过程会比非存储过程获得更好的性能。
  - 与其他应用程序共享应用逻辑，确保一致的数据访问和修改。
    - 存储过程封装了业务逻辑。若规则或策略有变化，则只需要修改服务器上的存储过程，所有的客户端就可以直接使用。
  - 屏蔽数据库模式的详细资料。
    - 用户不需要访问底层的数据库和数据库内的对象。

- 提供了安全性机制。
  - 用户可以被赋予执行存储过程的权限，而不必在存储过程引用的所有对象上都有权限
- 改善性能。
  - 预编译的 **Transact-SQL** 语句，可以根据条件决定执行哪一部分
- 减少网络通信量。
  - 客户端用一条语句调用存储过程，就可以完成可能需要大量语句才能完成的任务，这样减少了客户端和服务器之间的请求/回答包。

简而言之，
存储过程提高了效率，提高了安全性，有利于SQL语句重用。



- 存储过程的分类
  - 用户定义的存储过程
  - 系统存储过程
    一般以**sp_**做前缀（还有一部分以**xp**开头的）
    实现**SQL Server**的一些管理活动

- 创建存储过程
  - 用 **Transact-SQL** 语句 **CREATE PROCEDURE** 来创建存储过程，其基本语法格式如下：

    CREATE PROCEDURE 过程名[参数1，参数2，…参数n]
    AS
    Transact-SQL语句块;

【创建无参存储过程】

```
CREATE PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment;
```

【创建带参数的存储过程】

```
CREATE PROCEDURE Production.uspGetList
    @Product varchar(40) , @MaxPrice money
AS
   --查询产品的名称和价格
  SELECT p.[Name] AS Product, p.ListPrice AS 'List Price'
     FROM Production.Product AS p
     JOIN  Production.ProductSubcategory AS s
        ON p.ProductSubcategoryID = s.ProductSubcategoryID
 WHERE s.[Name] LIKE @Product AND p.ListPrice< @MaxPrice;
```

- 创建存储过程的注意事项
  - 只能在当前数据库内创建存储过程，除了临时存储过程。
  - 临时存储过程创建在 **tempdb** 数据库中。
  - 存储过程可以引用表、视图、用户定义函数、其他存储过程以及临时表。
  - 若存储过程创建了局部临时表，则当存储过程执行结束后临时表消失。
  - 执行 **CREATE PROCEDURE** 语句的用户必须是 **sysadmin**、**db_owner** 或 **db_ddladmin**角色的成员，或必须拥有 **CREATE PROCEDURE** 权限。

- 执行存储过程
  - 使用**Transact-SQL EXECUTE** 语句
  - 如果存储过程是批处理中的第一条语句，那么不使用**EXECUTE** 关键字也可以执行存储过程。
  - 示例：
    教材**P288－289**的【例**12－12**】、【**12－13**】

```
CREATE PROCEDURE Production.uspGetList @Product varchar(40)
    , @MaxPrice money
    , @ComparePrice money OUTPUT
    , @ListPrice money OUTPUT
AS
BEGIN
   --查询产品的名称和价格
     SELECT p.[Name] AS Product, p.ListPrice AS 'List Price'
        FROM Production.Product AS p
        JOIN Production.ProductSubcategory AS s
           ON p.ProductSubcategoryID = s.ProductSubcategoryID
        WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice;
  -- 计算@ListPprice.
  SET @ListPrice = (SELECT MAX(p.ListPrice)
        FROM Production.Product AS p
        JOIN  Production.ProductSubcategory AS s
           ON p.ProductSubcategoryID = s.ProductSubcategoryID
        WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice);
  -- 计算@compareprice.
  SET @ComparePrice = @MaxPrice;
END
GO
```

【12—12】

【例 12-13】 执行存储过程

【12—13】

```
DECLARE @ComparePrice money, @Cost money
--执行存储过程 Production.uspGetList，输入产品名称'%Bikes'和价格 700
EXECUTE Production.uspGetList '%Bikes%', 700,
    @ComparePrice  OUTPUT,
    @Cost OUTPUT
IF @Cost <= @ComparePrice
BEGIN
        PRINT N'该产品可以低于
        $'+RTRIM(CAST(@ComparePrice AS varchar(20)))+N'的价格购买.'
END
ELSE
    PRINT N'该类所有的产品价格均超过
    $'+ RTRIM(CAST(@ComparePrice AS varchar(20)))+'.'
```

执行的结果如下。它表明数据库中有满足条件 14 种产品。

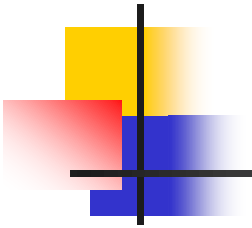
```
Product                                                  List Price
-------------------------------------------------------------------
Road-750 Black, 58                                          539.99
Mountain-500 Silver, 40                                      564.99
Mountain-500 Silver, 42                                      564.99
Mountain-500 Silver, 44                                      564.99
   .
   .
   .
(14 行受影响)
该产品可以低于
          $700.00 的价格购买.
```

- 存储过程的编译
  - 第一次执行存储过程时，将编译该过程，确定存储过程的执行计划。
  - 如果执行计划保存在数据库缓存中，随后的执行操作可以重新使用该执行计划，不需要重新编译存储过程。
  - 可以显示地重新编译存储过程
    - **CREATE PROCEDURE [WITH RECOMPILE]**，存储过程将在每次执行时都重新编译
    - **EXECUTE [WITH RECOMPILE]**，在执行存储过程时指定 **WITH RECOMPILE** 选项，可强制对存储过程进行重新编译
    - **sp_recompile**，系统存储过程强制在下次运行存储过程或触发器时进行重新编译。

- 查看存储过程的信息
  - 查看所有类型存储过程的额外信息
    ——调用系统存储过程 **sp_help**、**sp_helptext**、**sp_depends**
  - 显示数据库中的存储过程以及拥有者名字的列表
    ——调用系统存储过程 **sp_stored_procedures**
  - 查询存储过程的信息
    ——查询系统表 **sysobjects**、**syscomments**、**sysdepends**

12.4 用户定义函数

- 函数也是一种存储过程，只不过它能返回值，返回值可以是单个标量值或结果集。
- 返回标量值的函数称为标量函数。
- 返回结果集的函数称为表值函数。表值函数又分为内联（**inline**）表值函数和多语句表值函数。
- 可以在**SQL**的表达式中直接使用函数
  例如：**SELECT GETDATE();**
- **SQL Server**函数分为系统函数和用户定义函数**UDF(User Defined Function)**。



- 用户定义函数
  - 系统内置函数的扩展和补充的手段
  - 某些情况下，用户定义函数可以替代视图和存储过程
  - **SQL Server** 支持的三种用户定义函数
    - 标量函数
    - 多语句表值函数
    - 内联表值函数

标量函数

- 标量函数返回一个标量（单值）结果。
- 可在与标量函数返回的数据类型相同的值所能使用的任何位置使用该标量函数，包括 **SELECT** 语句中列的列表和 **WHERE** 子句、表达式、表定义中的约束表达式。
- 返回类型可以是除 **text**、**ntext**、**image**、**cursor** 和 **timestamp** 外的任何数据类型。

- 标量函数示例

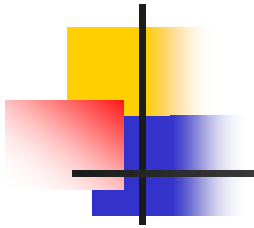
```
CREATE FUNCTION
   dbo.ufnGetInventoryStock(@ProductID int)
   RETURNS int
AS
BEGIN
   DECLARE @ret  int;
   SELECT @ret = SUM(p.Quantity)
      FROM Production.ProductInventory p
      WHERE p.ProductID = @ProductID
                  AND p.LocationID = '6';
  IF (@ret IS NULL)
       SET @ret = 0
  RETURN @ret
END;
```

```
CREATE FUNCTION fn_DateFormat
        (@indate datetime, @separator char(1))
    RETURNS Nchar(20)
AS
BEGIN
   RETURN
     CONVERT(Nvarchar(2),
   datepart(mm,@indate))
        + @separator
        + CONVERT(Nvarchar(2), datepart(dd,
   @indate))
        + @separator
        + CONVERT(Nvarchar(2), datepart(yy,
   @indate))
END
```

```
SELECT dbo.fn_DateFormat(GETDATE(), ':')
```

多语句表值函数

- 多语句表值函数返回一个由一条或多条 **Transact-SQL** 语句建立的表
- 返回数据类型是**table**；
- 可以替代视图，但是功能比视图强大；
  视图受限于单个**SELECT**语句，而用户自定义函数可以包含更多语句。
- 类似于存储过程，但与存储过程不同的是，多语句表值函数可以在 **SELECT** 语句的 **FROM** 子句中被引用



- 多语句表值函数示例
  P292【例12-15】

```
USE Northwind
GO
CREATE FUNCTION fn_Employees (@length nvarchar(9))
   RETURNS @fn_Employees table
           ( EmployeeID int PRIMARY KEY NOT NULL,
             [Employee Name] nvarchar(61) NOT NULL)
AS
BEGIN
   IF @length = 'ShortName'
      INSERT @fn_Employees
         SELECT EmployeeID, LastName
         FROM Employees
  ELSE IF @length = 'LongName'
      INSERT @fn_Employees
         SELECT EmployeeID, (FirstName + ' ' + LastName)
         FROM Employees
  RETURN
END
```

```
SELECT * FROM dbo.fn_Employees('LongName')
```

内联表值函数

- 内联（内嵌）表值函数返回单条 **SELECT** 语句产生的结果表，类似于视图。
- 相对于视图，内联表值函数可以使用参数，提供了更强的适应性。
- 实际使用内联函数时，可以将它看作一个参数化的视图，可以从中删除和修改数据。最终删除和修改的是基表的数据。
- 不能在内联表值函数体内使用**BEGIN…END**语句块，只能指定一个**RETURN**子句和一个查询，而且不能对返回的表的结构进行定义。

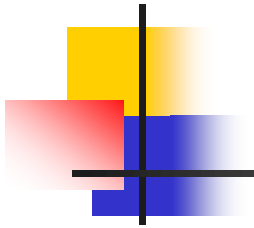
- 内联表值函数示例

```
CREATE FUNCTION fn_CustomerNamesInRegion
    ( @RegionParameter nvarchar(30) )
   RETURNS table
AS
  RETURN (
      SELECT CustomerID, CompanyName
         FROM Northwind.dbo.Customers
        WHERE Region = @RegionParameter
  );
```

```
SELECT * FROM fn_CustomerNamesInRegion(N'WA')
```

12.5 Transact-SQL触发器

- 触发器是数据库服务器中发生事件时自动执行的特殊存储过程。
- 触发器的作用
  - 主要用于实现数据完整性
  - 还可以实现审核更改以及更多的功能
- 触发器没有接口（输入参数和输出参数），不能被显示调用，只有当某一事件（触发器定义的触发事件）发生时由数据库服务自动执行。
- 触发器是引发它的事务的一部分。



- **SQL Server支持的触发器**
  - **DML**（数据操纵语言）触发器
    ——由**DML**事件触发，即针对表或视图的
    - **INSERT**
    - **DELETE**
    - **UPDATE**
  - **DDL**（数据定义语言）触发器
  - 登录触发器

- **SQL Server中定义DML触发器的基本格式**

  CREATE TRIGGER <触发器名>

  ON {<表名>|<视图名>}

  {FOR | AFTER | INSTEAD OF} <触发事件>

  AS <SQL语句>

  其中，触发事件为：**INSERT**、**UPDATE** 或 **DELETE**。

  - 创建一个触发器时必须指定：
    - 名称；
    - 在其上定义触发器的表；
    - 触发器将何时激发；
    - 触发器要执行的操作。

- **DML触发器的类型**
  - **AFTER触发器**
  - **INSTEAD OF 触发器**

——**FOR** 和 **AFTER** 是完全相等的，如果仅指定 **FOR** 关键字，则 **AFTER** 是默认设置。

——**SQL Server没有BEFORE触发器**

deleted和inserted表

- **DML** 触发器语句使用两种特殊的表：删除的表**deleted**和插入的表**inserted**。**SQL Server**会自动创建和管理这两种表。
- 可以使用它们测试特定数据修改的影响以及设置 **DML** 触发器操作条件。
- 不能直接修改临时表中的数据或对表执行数据定义语言 **(DDL)** 操作。

- **deleted表**用于存储 **DELETE** 和 **UPDATE** 语句所影响的行的副本。在执行 **DELETE** 或 **UPDATE** 语句的过程中，行从触发器表中删除，并传输到删除的表中。 **deleted表**和触发器表通常没有相同的行。
- **inserted表**用于存储 **INSERT** 和 **UPDATE** 语句所影响的行的副本。在执行插入或更新事务过程中，新行会同时添加到 **inserted** 表和触发器表中。 **inserted表**中的行是触发器表中的新行的副本。

AFTER 触发器

- 在执行了 **INSERT**、**UPDATE** 或 **DELETE** 语句操作之后（因此你为**AFTER**触发器）执行 **AFTER** 触发器。
- 只有在触发 **SQL** 语句中指定的所有操作都已成功执行后才激发。所有的引用级联操作和约束检查也必须成功完成后，才能执行此触发器。
- **AFTER** 触发器只能在表上创建，不能在临时表或视图上创建。
- **AFTER**触发器按语句触发而不是按行触发。
- 可以为每一种修改操作语句定义多个**AFTER**触发器，它们将按顺序执行。

INSTEAD OF 触发器

- **INSTEAD OF** 触发器将代替原触发语句的操作。
- 原触发语句不对表进行修改，但仍完成一些操作。**Inserted**和**deleted**表的内容是由原语句的操作结果决定的。
- **INSTEAD OF** 触发器可以在表上创建，也可以在视图上创建。
- 每个表或视图的同一种操作只能定义一个 **INSTEAD OF** 触发器。

- **INSTEAD OF** 触发器在约束之前被触发，因而可以识别出因违反约束而失败的操作，并用正确的操作替换它。
- 可为带有一个或多个基表的视图定义 **INSTEAD OF** 触发器，从而能够扩展视图可支持的更新类型。

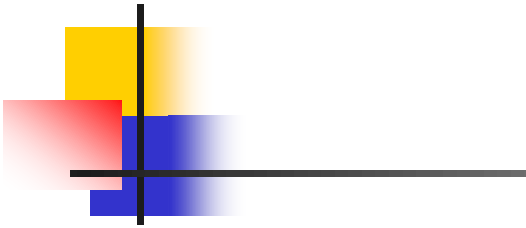
INSTEAD OF和AFTER触发器的比较，P296。

触发器示例

```
USE pubs
IF EXISTS (SELECT name FROM sysobjects
     WHERE name = 'reminder' AND type = 'TR')
    DROP TRIGGER reminder
GO
CREATE TRIGGER reminder
ON titles
FOR INSERT, UPDATE
AS sql语句
GO
```

```
Use Northwind
GO
CREATE TRIGGER Empl_Delete ON Employees
FOR DELETE
AS
IF (SELECT COUNT(*) FROM Deleted) > 1
BEGIN
   RAISERROR('You cannot delete more than one
   employee at a time.', 16, 1)
   ROLLBACK TRANSACTION
END
```

```
Create Table Student(                          --学生表
    StudentID int primary key,                 --学号
    ....
)
Create Table BorrowRecord(                     --学生借书记录表
    BorrowRecordID   int identity(1,1),        --流水号
    StudentID        int ,                     --学号
    BorrowDate       datetime,                 --借出时间
    ReturnDate       datetime,                 --归还时间
    ...
)
```



- 如果更改了学生的学号，希望他的借书记录仍然与这个学生相关(也就是同时更改借书记录表的学号)。

- 如果该学生已经毕业，希望删除他的学号的同时，也删除它的借书记录。

```
Create Trigger truStudent
    On Student                  --在Student表中创建触发器
    for Update                  --为什么事件触发
As                              --事件触发后所要做的事情
  if Update(StudentID)
    begin
      Update BorrowRecord
        Set br.StudentID=i.StudentID
        From BorrowRecord br ,
             deleted   d,   --deleted临时表
             inserted i     --inserted临时表
        Where br.StudentID=d.StudentID
    end;
```

```
Create trigger trdStudent
        On Student
        for Delete
As
    Delete BorrowRecord
    From BorrowRecord br , deleted d
    Where br.StudentID=d.StudentID
```

- 更多的触发器示例
  P298【例12－18】
  P300【例12－19】
  P310(e328)习题10和11
  （参照教材的示例做）

本章小结

- **Transact-SQL**
- **Transact-SQL游标**
- **Transact-SQL存储过程**
- **Transact-SQL用户定义函数**
- **Transact-SQL触发器**