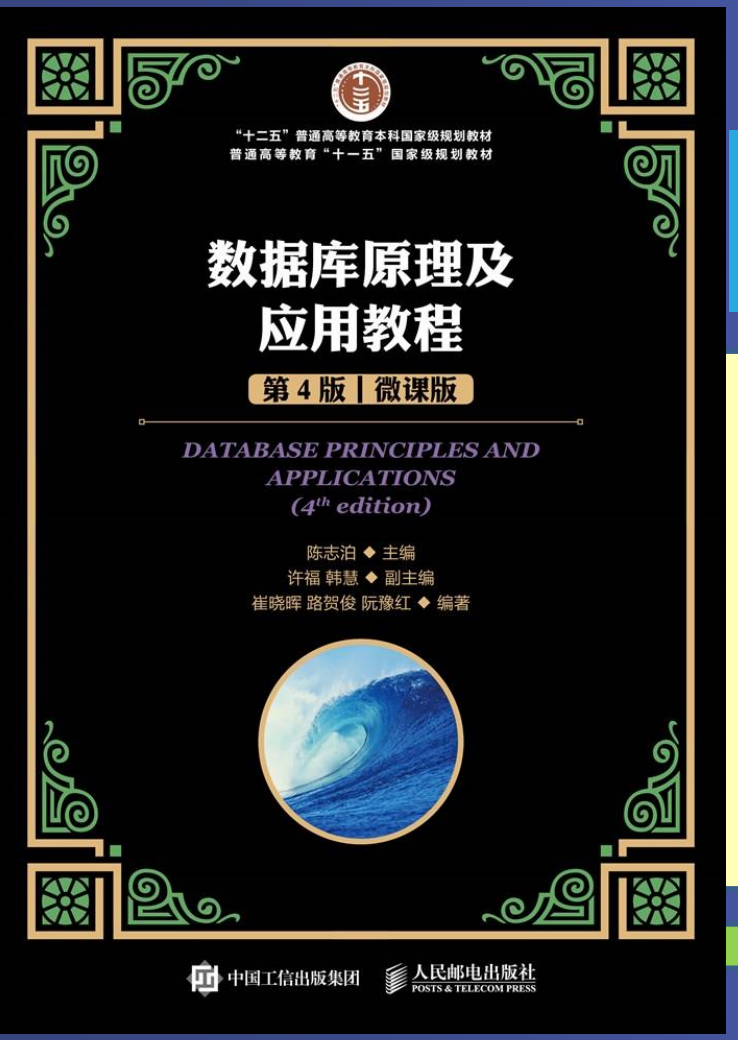


数据库原理及应用教程（第4版）

“十二五”普通高等教育本科国家级规划教材

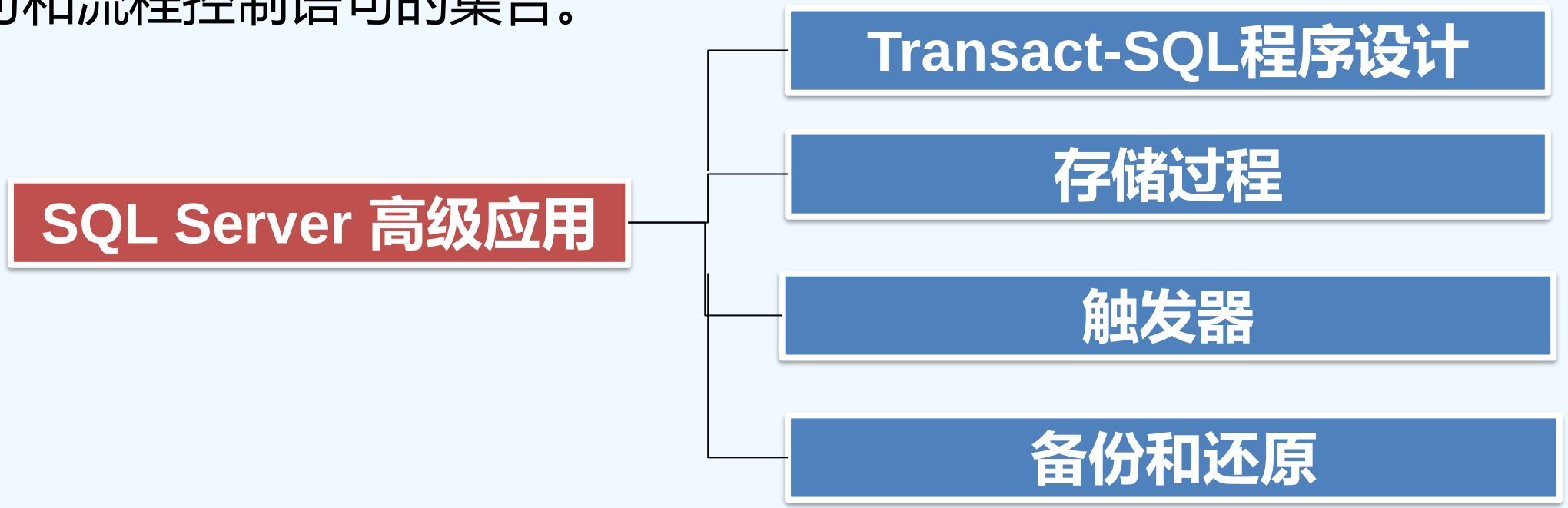
微课版，对重点和难点进行视频详解

中国工信出版集团　　人民邮电出版社

第7章 SQL Server 2012高级应用

本章重点

- Transact-SQL是T-SQL是使用SQL Server的核心。在SQL Server数据库管理系统中，存储过程和触发器具有重要的作用。存储过程和触发器都是SQL语句和流程控制语句的集合。



7.1 Transact-SQL程序设计

7.2 存储过程

7.3 触发器

7.4 备份和还原

7.5 小结

7.1.1 变量

全局变量

全局变量由系统定义和维护的，只能使用预先说明及定义的全局变量。

全局变量对用户而言是只读的，用户无法对它们进行修改或管理 。

局部变量

```
DECLARE  @变量名  变量类型
                [，@变量名  变量类型……]
SELECT @变量名=变量值
或
SET @变量名=变量值
```

[例7-1] 声明一个长度为8个字符的变量id，并赋值为'10010001'。

```
DECLARE @id char(8)
SELECT @id= ‘10010001 ‘
```

注释符

在Transact-SQL中可以使用两类注释符：

（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。

7.1.2 运算符

算术运算符

运 算 符	含 义
+	加
–	减
*	乘
/	除
%	求余数

7.1.2 运算符

赋值运算符

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

字符串连接运算符

加号 (+) 是字符串连接运算符，可以用它将字符串连接起来。其他所有字符串操作都使用字符串函数进行处理。

7.1.2 运算符

比较运算符

运 算 符	含 义
=	等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于
<>	不等于
!=	不等于（非 SQL-92 标准）
!<	不小于（非 SQL-92 标准）
!>	不大于（非 SQL-92 标准）

7.1.2 运算符

逻辑运算符

运 算 符	含 义
ALL	如果一组比较中都为TRUE，运算结果就为TRUE
AND	如果两个表达式都为TRUE，运算结果就为TRUE
ANY	如果一组的比较中任何一个为TRUE，运算结果就为TRUE
BETWEEN	如果操作数在某个范围之内，运算结果就为TRUE
EXISTS	如果子查询包含一些行，运算结果就为TRUE
IN	如果操作数等于表达式列表中的一个，运算结果就为TRUE
LIKE	如果操作数与一种模式相匹配，运算结果就为TRUE
NOT	对逻辑值取反，即如果操作数的值为TRUE，运算结果为FALSE，否则为TRUE
OR	如果两个布尔表达式中的一个为TRUE，运算结果就为TRUE
SOME	如果一系列操作数中，有些值为TRUE，运算结果为TRUE

7.1.2 运算符

按位运算符

运算符	含义	运算规则
&	按位与	两个数对应的二进制位上都为1时，该位上的运算结果为1，否则为0
|	按位或	两个数对应的二进制位上有一个为1时，该位上的运算结果为1，否则为0
^	按位异或	两个数对应的二进制位上不同时，该位上的运算结果为1，否则为0

7.1.2 运算符

一元运算符

运算符	含　义
+	正号，数值为正
-	负号，数值为负
~	按位取反，对操作数进行按二进制位取反运算，即二进制位上原来为1，运算结果为0，否则为1

7.1.2 运算符

运算符优先级和结合性

优先级（从高到低）	运算符	说明
1	( )	小括号
2	+、-、~	正、负、按位取反
3	*、/、%	乘、除、求余数
4	+、-、+	加、减、字符串连接
5	=、>、<、>=、<=、<>、!=、!>、!<	各种比较运算符
6	^、&、|	位运算符
7	NOT	逻辑非
8	AND	逻辑与
9	ALL、ANY、BETWEEN、IN、LIKE、OR、SOME	逻辑运算符
10	=	赋值运算符

7.1.3 批处理

批处理是包含一个或多个T-SQL语句的组，批处理的所有语句被整合成一个执行计划。

批处理是使用GO语句将多条SQL语句进行分隔，其中每两个GO之间的SQL语句就是一个批处理单元。

每个批处理被单独地处理，所以一个批处理中的错误不会阻止另一个批处理的运行。

7.1.4 流程控制语句

T-SQL使用的流程控制语句与常见的程序设计语言类似，主要有以下几种控制语句

BEGIN…END语句

BEGIN…END的语法格式如下：

```
BEGIN
    <命令行或程序块>
END
```

IF…ELSE语句

IF…ELSE语句的语法格式如下：

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

IF [NOT] EXISTS语句

IF [NOT] EXISTS语句的语法格式如下：

```
IF [NOT] EXISTS (SELECT 子查询)
    <命令行或程序块>
[ELSE
    <命令行或程序块>]
```

CASE语句

（1）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
```

（2）格式2：

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

WHILE…CONTINUE…BREAK语句

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

WAITFOR语句

```
WAITFOR {DELAY <'时间'> | TIME <'时间'>
                    | ERROREXIT |
PROCESSEXIT | MIRROREXIT}
```

GOTO语句

GOTO 标识符

RETURN语句

RETURN（[整数值]）

7.1.5 常用命令

BACKUP

用于将数据库内容或其事务处理日志备份到存储介质上（软盘、硬盘、磁带等）。

CHECKPOINT

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。

DECLARE命令用于声明一个或多个局部变量、游标变量或表变量。

例：

```
DECLARE @x CHAR, @y CHAR(10)
SELECT @x ='123', @y ='data_type'
PRINT @x
PRINT @y
```

EXECUTE

EXECUTE或EXEC命令用来执行存储过程 。

KILL

KILL命令用于终止某一过程的执行。

PRINT

PRINT的语法格式如下:

PRINT 'any ASCII text' | @local_variable | @@FUNCTION | string_expression

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。

RAISERROR

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。

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。

SELECT

SELECT 命令可用于给变量赋值，其语法格式如下：

SELECT {@local_variable = expression } [，...n]

SELECT 命令可以一次给多个变量赋值。

SET 命令有两种用法。

- ✓用于给局部变量赋值。
- ✓用于用户执行SQL 命令时，SQL Server 处理选项的设定 。
- ✓SET：选项ON；
- ✓SET：选项OFF；
- ✓SET：选项值。

SHUTDOWN

- ✓SHUTDOWN [WITH NOWAIT]
- ✓SHUTDOWN 命令用于停止SQL Server 的执行。

USE

- ✓USE {database}
- ✓USE命令用于改变当前使用的数据库为指定的数据库。

7.1.5 常用函数

统计函数

STDEV函数

✓STDEV函数返回表达式中所有数据的标准差。

STDEVP函数

✓STDEVP 函数返回表达式中所有数据的总体标准差。

VAR函数

✓VAR函数返回表达式中所有数据的统计变异数。

VARP函数

✓VARP函数返回表达式中所有数据的总体变异数。

算数函数

函　数		功　能
三角函数	SIN	返回以弧度表示的角的正弦
	COS	返回以弧度表示的角的余弦
	TAN	返回以弧度表示的角的正切
	COT	返回以弧度表示的角的余切
反三角函数	ASIN	返回正弦是FLOAT 值的以弧度表示的角
	ACOS	返回余弦是FLOAT 值的以弧度表示的角
	ATAN	返回正切是FLOAT 值的以弧度表示的角
角度弧度转换	DEGREES	把弧度转换为角度
	RADIANS	把角度转换为弧度

指数函数	EXP(表达式)	返回以e为底、以表达式为指数的幂值
对数函数	LOG(表达式) LOG10(表达式)	返回表达式的以e为底的自然对数值 返回表达式的以10 为底的对数值
平方根函数	SQRT(表达式)	返回表达式的平方根
取近似值函数	CEILING(表达式) FLOOR(表达式) ROUND(表达式,n)	返回大于等于表达式的最小整数 返回小于等于表达式的最大整数 将表达式四舍五入为指定的精度n
符号函数	ABS(表达式) SIGN(表达式)	返回表达式的绝对值 测试表达式的正负号，返0、1或 - 1
其他函数	PI( ) RAND( )	返回值为π，即3.1415926535897936 返回0～1之间的随机浮点数

字符串函数

字符串转换函数

ASCII（character_expression）

✓返回字符表达式最左端字符的ASCII 码值

CHAR （integer_expression）

✓CHAR函数用于将ASCII 码转换为字符

LOWER （character _expression）

✓LOWER函数用于把字符串全部转换为小写

UPPER （character _expression）

✓UPPER函数用于把字符串全部转换为大写

STR（float _expression [，length[，<decimal>]]）

✓STR函数用于把数值型数据转换为字符型数据

字符串函数

去空格函数

- ✓LTRIM （character _expression）
- ✓LTRIM函数用于把字符串头部的空格去掉。
- ✓RTRIM （character _expression）
- ✓RTRIM函数用于把字符串尾部的空格去掉 。

取子串函数

- ✓LEFT（character_expression, integer_expression）
- ✓LEFT函数返回的子串是从字符串最左边起到第integer_expression 个字符的部分。
- ✓RIGHT（character_expression, integer_expression）
- ✓RIGHT函数返回的子串是从字符串右边第integer_expression 个字符起到最后一个字符的部分。

字符串函数

字符串比较函数

- ✓CHARINDEX （substring_expression，expression）
- ✓CHARINDEX函数返回字符串中某个指定的子串出现的开始位置。

- ✓PATINDEX（'%substring_expression%'，expression）
- ✓PATINDEX函数返回字符串中某个指定的子串出现的开始位置。

字符串函数

字符串操作函数

- ✓QUOTENAME（character_expression [，quote_ character])
- ✓QUOTENAME函数返回被特定字符括起来的字符串。

- ✓REPLICATE（character_expression，integer_expression）
- ✓REPLICATE函数返回一个重复指定次数的字符串。

- ✓REVERSE（character_expression）
- ✓REVERSE函数将指定的字符串的字符排列顺序颠倒。

字符串函数

字符串操作函数

- ✓REPLACE（string_expression1，string_expression2，string_expression3）
- ✓REPLACE函数返回被替换了指定子串的字符串。
- ✓SPACE（integer_expression）
- ✓SPACE函数返回一个有指定长度的空格字符串。
- ✓STUFF（character_expression1，start_ position，length，character_expression2）
- ✓STUFF函数用另一子串替换字符串中指定位置长度的子串。
- ✓数据类型转换函数

字符串函数

数据类型转换函数

- ✓CAST（<expression> AS <data_type>[length]）
- ✓CONVERT（<data_type>[（length）]，<expression> [，style]）

日期函数

- ✓ DAY（<date_expression>）
- ✓ DAY函数返回date_expression 中的日期值。

- ✓ MONTH（<date_expression>）
- ✓ MONTH函数返回date_expression中的月份值。

- ✓ YEAR（<date_expression>）
- ✓ YEAR函数返回date_expression中的年份值。

- ✓ DATEADD（<datepart> <number> <date>）
- ✓ DATEADD函数返回指定日期date加上指定的额外日期间隔number产生的新日期。

日期函数

DATEDIFF（<datepart>，<date1>，<date2>）

DATEDIFF函数返回两个指定日期在datepart方面的不同之处，即date2超过date1的差距值，其结果值是一个带有正负号的整数值。

DATENAME（<datepart>，<date>）

DATENAME函数以字符串的形式返回日期的指定部分，此部分由datepart 来指定。

DATEPART（<datepart>，<date>）

DATEPART函数以整数值的形式返回日期的指定部分，此部分由datepart 来指定。

GETDATE（）

GETDATE函数以DATETIME 的缺省格式返回系统当前的日期和时间，它常作为其他函数或命令的参数使用。

【例1】

```
DECLARE @cCName char(10),
        @iDiscount int

SELECT @cCName = cname, @iDiscount = discnt FROM Customers
WHERE cid = 'C003'

IF @iDiscount >= 10
BEGIN
PRINT @cCName
PRINT 'is a VIP customer!'
END
ELSE
BEGIN
 PRINT @cCName
 PRINT 'is a General customer!'
END;
```

【例2】

```
USE AdventureWorks;
GO
SELECT ProductNumber, Name, 'Price Range' =
            CASE
              WHEN ListPrice = 0 THEN 'Mfg item - not for
resale'
              WHEN ListPrice < 50 THEN 'Under $50'
              WHEN ListPrice >= 50 and ListPrice < 250 THEN
'Under $250'
              WHEN ListPrice >= 250 and ListPrice < 1000
THEN 'Under $1000'
              ELSE 'Over $1000'
            END
FROM Product
ORDER BY ProductNumber ;
GO
```

【例3】

```
USE AdventureWorks;
GO
SELECT ProductNumber, Category =
                    CASE ProductLine
                       WHEN 'R' THEN 'Road'
                       WHEN 'M' THEN 'Mountain'
                       WHEN 'T' THEN 'Touring'
                       WHEN 'S' THEN 'Other sale items'
                       ELSE 'Not for sale'
                    END,
               Name
 FROM Product
 ORDER BY ProductNumber;
GO
```

7.2.1 存储过程的概念、优点及分类

不使用存储过程
- 执行语句未知
- 按顺序执行

效率很低

存储过程是一组为了完成特定功能的语句集 。

存储过程的优点：

模块化的程序设计，独立修改。

高效率的执行 ，一次编译。

减少网络流量 。

可以作为安全机制使用。

存储过程的分类：

系统存储过程，master数据库，SP前缀。

用户自定义存储过程。

扩展存储过程，XP前缀。

7.2.2 创建存储过程

当创建存储过程时，需要确定存储过程的三个组成部分：

(1) 所有的输入参数以及传给调用者的输出参数。

(2) 被执行的针对数据库的操作语句，包括调用其他存储过程的语句。

(3) 返回给调用者的状态值以指明调用是成功还是失败。

↓

用CREATE PROCEDURE命令创建存储过程

```
CREATE PROCEDURE procedure_name [ ; number ]
[ { @parameter data_type }
[ VARYING ] [ = default ] [ OUTPUT ]] [ ,...n ]
[ WITH
{ RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[ FOR REPLICATION ]
AS sql_statement [ ...n ]
```

【例7-15】 在Teach数据库中，创建一个名称为MyProc的不带参数的存储过程，该存储过程的功能是从数据表S中查询所有男同学的信息。

```
USE Teach
GO
CREATE PROCEDURE MyProc AS
SELECT * FROM S WHERE Sex='男'
```

【例7-16】 定义具有参数的存储过程。在Teach数据库中，创建一个名称为InsertRecord的存储过程，该存储过程的功能是向S数据表中插入一条记录，新记录的值由参数提供。

```
USE Teach
GO
CREATE PROCEDURE InsertRecord
(   @sno VARCHAR(6),
    @sn NVARCHAR(10),
    @sex NCHAR(1),
    @age INT,
    @dept NVARCHAR(20)
)
AS
INSERT INTO S VALUES(@sno,@sn,@sex,@age, @dept)
```

【例7-17】 定义具有参数默认值的存储过程。在Teach数据库中，创建一个名称为Insert RecordDefa的存储过程，该存储过程的功能是向S数据表中插入一条记录，新记录的值由参数提供，如果未提供系别Dept的值时，由参数的默认值代替。

```
USE Teach
GO
CREATE PROCEDURE InsertRecordDefa
(   @sno VARCHAR(6),
    @sn NVARCHAR(10),
    @sex NCHAR(1),
    @age INT,
    @dept NVARCHAR(20)= '无')
AS
INSERT INTO S VALUES(@sno, @sn, @sex, @age, @dept)
```

【例7-18】 定义能够返回值的存储过程。在Teach数据库中，创建一个名称为QueryTeach的存储过程。该存储过程的功能是从数据表S中根据学号查询某一同学的姓名和系别，查询的结果由参数@sn和@dept返回。

```
USE Teach
GO
CREATE PROCEDURE QueryTeach
(   @sno VARCHAR(6),
    @sn NVARCHAR(10) OUTPUT,
    @dept NVARCHAR(20) OUTPUT)
AS
SELECT @sn=SN,@dept=Dept
FROM S
WHERE SNo=@sno
```

【例4】

```
CREATE PROCEDURE prcListAgentsByCity @cCity char(15)
  AS
  BEGIN
  IF EXISTS(SELECT * FROM Agent WHERE City = @cCity)
  BEGIN
    PRINT 'List of Agents:‘
    SELECT aid,aname,city,percent
    FROM Agent
    WHERE cCity = @city
    Return 0
  END
  ELSE BEGIN
    PRINT 'No Records Found for given city'
    RETURN 1
       END
  END
```

【例4-续】

```
DECLARE @ReturnValue int
    EXEC @ReturnValue = prcListAgentsByCity 'New
York'
    SELECT @ReturnValue
```

【例5】

```
CREATE PROCEDURE prcGetProductDetail
            @cPid char(10),
             @cPname char(30) OUTPUT,
             @iQuantity int OUTPUT,
            @fPrice float OUTPUT
          AS
          BEGIN
          IF EXISTS(SELECT * FROM Product WHERE pid = @cPid)
           BEGIN
              SELECT @cPname =pname,@iQuantity=quantity,@fPrice=price
              FROM Product
              WHERE pid = @cPid
              RETURN 0
           END
           ELSE
              RETURN 1
           END
```

【例6】

```
CREATE PROCEDURE prcDisplayProductDetails @cPid
char(10)
AS
BEGIN
DECLARE @cPname char(30)
DECLARE @iQuantity int
DECLARE @fPrice float
DECLARE @ReturnValue int
EXEC @ReturnValue = prcGetProductDetail @cPid,
            @cPname output,
            @iQuantity output,
            @fPrice output
```

【例6-续】

```
IF (@ReturnValue = 0)
    BEGIN
    PRINT 'The name of the product: '+ @cPname
    PRINT `Quantity : '  + CONVERT( char(30), @iQuantity)
    PRINT `price : ' + CONVERT (char(30), @fPrice)
  END
  ELSE
    PRINT 'No records for the given city code`
    END
```

利用对象资源管理器创建存储过程

（1）在选定的数据库下打开“可编程性”节点。

（2）找到“存储过程”节点，单击鼠标右键，在弹出的快捷菜单中选择“新建存储过程”。

（3）在新建的查询窗口中可以看到关于创建存储过程的语句模板，在其中添上相应的内容，单击工具栏上的“执行”按钮即可。


查看存储过程
sp_helptext 存储过程名称
重新命名存储过程
通过对象资源管理器
删除存储过程
DROP PROCEDURE {procedure}} [,…n]
执行存储过程
EXEC MyProc
修改存储过程
ALTER PROCEDURE procedure_name [;number]

7.3.1 触发器概述

触发器是一种特殊类型的存储过程。

触发器主要有以下优点：

- ✓触发器是在某个事件发生时自动激活而执行的。
- ✓触发器可以实现比约束更为复杂的完整性要求。
- ✓触发器可以根据表数据修改前后的状态，根据其差异采取相应的措施。
- ✓触发器可以防止恶意的或错误的INSERT、UPDATE和DELETE操作。

触发器的种类

DML触发器

✓DML触发器是在执行数据操纵语言（DML）事件时被激活而自动执行的触发器。

DDL触发器

✓DDL触发器是在响应各种数据定义语言（DDL）事件而激活执行的存储过程。

登录触发器

✓登录触发器是由登录（LOGON）事件而激活的触发器。

7.3.2 触发器的工作原理

SQL Server在工作时为每个触发器在服务器的内存上建立两个特殊的表：插入表和删除表。

对表的操作	Inserted表	Deleted表
增加记录(INSERT)	存放增加的记录	无
删除记录(DELETE)	无	存放被删除的记录
修改记录(UPDATE)	存放更新后的记录	存放更新前的记录

(1) INSERT触发器的工作原理

口当对表进行INSERT操作时，INSERT触发器被激发时，新的数据行被添加到创建触发器的表和Inserted表。

(2) DELETE触发器的工作原理

口当试图删除触发器保护的表中的一行或多行记录时，即对表进行DELETE操作时，DELETE触发器被激发，系统从被影响的表中将删除的行放入到一个特殊的Deleted表中。

(3) UPDATE触发器的工作原理

口当试图更新定义有UPDATE触发器的表中的数据时，即当执行UPDATE操作时触发器被激活。

7.3.3 创建触发器

创建DML触发器

（1）使用CREATE TRIGGER创建DML触发器。

（2）使用对象资源管理器创建DML触发器。

创建DDL触发器

创建触发器的语法：

```
CREATE TRIGGER trigger_name
    ON table_name
    [WITH ENCRYPTION]
    FOR [INSERT | DELETE | UPDATE]
    AS sql_statements
```

【例7】Insert触发器

```
CREATE TRIGGER trgInsertOrder
ON ORDERS FOR insert AS
 DECLARE @ProductQuantity int
 DECLARE @OrderQuantity int
 DECLARE @cPid char(10)

 SELECT @cPid=Inserted.pid ,@OrderQuantity = Inserted.qty
 FROM Inserted
 SELECT @ProductQuantity = quantity
 FROM Products
 WHERE pid = @cpid
```

【例7-续】

```
IF (@OrderQuantity>@ProductQuantity)
  BEGIN
  PRINT 'The quantity for the product in this order is more
than the product quantity in stock.'
  ROLLBACK TRANSACTION
  END
  RETURN
```

【例8】Delete触发器

```
CREATE TRIGGER trgDeleteCustomer
ON Customers FOR delete
AS
DECLARE @cCid  char(10)
SELECT @cCid = Deleted.cid FROM Deleted
IF EXISTS(SELECT * FROM ORDERS WHERE cid=@cCid)
BEGIN
      PRINT 'Deletion of this customer is restricted.'
      ROLLBACK TRANSACTION
END
RETURN
```

【例9】Update触发器

```
CREATE TRIGGER trgUpdateOrderQuantity
    ON ORDERS FOR UPDATE
    AS
    DECLARE @cCid char(10)
    DECLARE @cPid char(10)
    DECLARE @cAid char(10)
    DECLARE @fPrice float
    DECLARE @iQty int
    DECLARE @fdiscnt float
    DECLARE @iPercent int
    DECLARE @iOrdno int
    DECLARE @fDollars float
```

【例9-续】Update触发器

```
 IF UPDATE(qty)
    BEGIN
    SELECT
@cCid=cid,@cPid=pid,@cAid=aid,@iOrdno=ordno,@iQty=qty
    FROM Inserted
    SELECT @fdiscnt = discnt FROM CUSTOMERS
    where cid = @cCid
    SELECT @iPercent = percent FROM AGENTS
    where aid = @cAid
    SELECT @fPrice = price FROM PRODUCTS
    where pid = @cPid
    SELECT @fDollars = 0.40*(@iQty*@fPrice) - 0.01*(@fdiscnt +
@iPercent)*(@iQty*@fPrice)
    UPDATE ORDERS SET dollars = @fDollars
    WHERE ordno = @iOrdno
    END
```

【例10】 INSERT, UPDATE, DELETE触发器

```
USE CAP

IF EXISTS (SELECT name FROM sysobjects WHERE name = 'reminder'
AND type = 'TR')
DROP TRIGGER reminder
GO

CREATE TRIGGER reminder
ON Orders FOR INSERT, UPDATE, DELETE
AS
EXEC master..xp_sendmail 'MaryM', 'The order table was modified
by somebody.'
GO
```

【例11】Instead of 触发器

```
CREATE TABLE FirstHalf
 (
     EmployeeID int NOT NULL PRIMARY KEY,
     PhoneNo char (10) NOT NULL
  )
 CREATE TABLE SecondHalf
 (
     EmployeeID int NOT NULL PRIMARY KEY REFERENCES
     FirstHalf (EmployeeID) ,
     Address varchar (50) NOT NULL,
     City varchar (25) NOT NULL
  )
```

【例11-续】Instead of 触发器

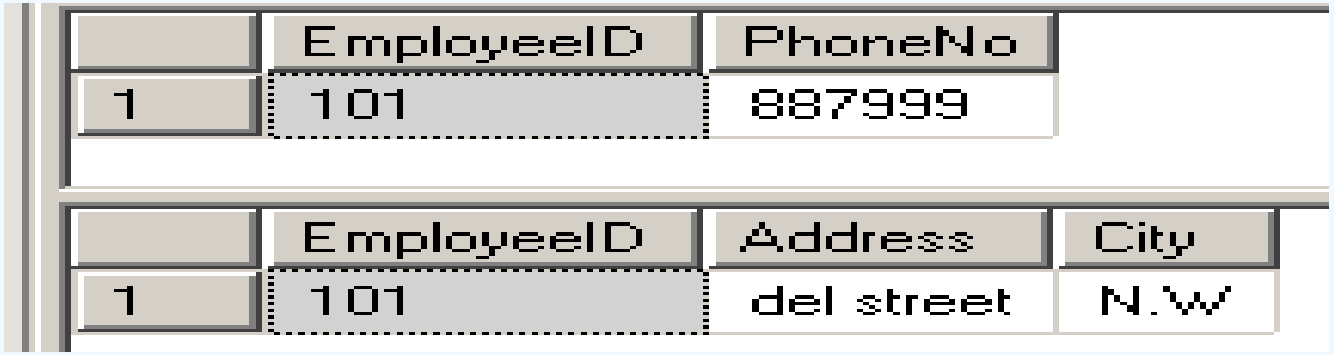
```
CREATE VIEW WholeThing
AS
SELECT   f.EmployeeID, f.PhoneNo, s.Address, s.City
FROM FirstHalf  f  JOIN SecondHalf s ON f.EmployeeID = s.EmployeeID
```

【例11-续】Instead of 触发器

```
CREATE TRIGGER tri_WholeThing ON WholeThing
INSTEAD OF INSERT
AS
IF @@ROWCOUNT = 0   RETURN
INSERT FirstHalf
   SELECT   EmployeeID, PhoneNo FROM   inserted
INSERT SecondHalf
  SELECT   EmployeeID, Address, City
  FROM   inserted
```

【例11-续】Instead of 触发器

```
Insert into wholeThing
 values(101,'887999','del street','N.W');
```



	EmployeeID	PhoneNo
1	101	887999

	EmployeeID	Address	City
1	101	del street	N.W

7.3.4 查看触发器

查看表中触发器

执行系统存储过程sp_helptrigger查看表中触发器的语法格式如下：

```
EXEC sp_helptrigger 'table'[,'type']
```

查看触发器的定义文本

利用系统存储过程sp_helptext可查看某个触发器的内容，语法格式为：

```
EXEC sp_helptext 'trigger_name'
```

查看触发器的所有者和创建时间

系统存储过程sp_help可用于查看触发器的所有者和创建日期，

语法格式如下：

```
EXEC sp_help 'trigger_name'
```

7.3.5 修改触发器

利用对象资源管理器修改触发器

利用对象资源管理器修改触发器，可以在已有的触发器的基础上进行修改，不需要重新编写。

利用ALTER TRIGGER语句修改触发器

（1）修改DML触发器的ALTER TRIGGER语句的语法格式如下：

```
ALTER TRIGGER schema_name.trigger_name
ON ( table | view )
[ WITH ENCRYPTION ]
{ FOR | AFTER | INSTEAD OF }
{ [ DELETE ] [ , ] [ INSERT ] [ , ] [ UPDATE] }
AS  sql_statement [ ; ]
```

利用ALTER TRIGGER语句修改触发器

（2）修改DDL触发器的ALTER TRIGGER语句的语法格式如下：

```
ALTER TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH ENCRYPTION ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS  sql_statement  [ ; ]
```

使触发器无效

在有些情况下，用户希望暂停触发器的作用，但并不删除它，这时就可以通过DISABLE TRIGGER语句使触发器无效，语法格式如下：

```
DISABLE TRIGGER { [ schema.] trigger_name [ ,...n ] | ALL }
ON object_name
```

使触发器重新有效

要使DML触发器重新有效，可使用ENABLE TRIGGER语句，语法格式如下：

```
ENABLE TRIGGER {[ schema_name.] trigger_name [ ,...n ] | ALL }
ON object name
```

7.4.1 备份和还原概述

备份是对SQL Server数据库或事务日志进行复制，数据库备份记录了在进行备份这一操作时数据库中所有数据的状态，如果数据库因意外而损坏，这些备份文件将在数据库还原时用来还原数据库。

还原就是把遭受破坏、丢失的数据或出现错误的数据库还原到原来的正常状态。

数据库备份的类型

数据库完整备份	数据库备份是指对数据库内的所有对象都进行备份
事务日志备份	事务日志备份只备份数据库的事务日志内容。
差异备份	是完整备份的补充，只备份自从上次数据库完整备份后数据库变动的部分。
文件和文件组备份	是针对单一数据库文件或者是文件组做备份

备份和还原的策略

◆SQL Server 提供了几种方法来减少备份或还原操作的执行时间。

口使用多个备份设备来同时进行备份处理。

口综合使用完整数据库备份、差异备份或事务日志备份来减少每次需要备份的数据量。

口使用文件或文件组备份以及事务日志备份，这样可以只备份或还原那些包含相关数据的文件，而不是整个数据库。

◆ 在SQL Server 2000 中有三种数据库还原模式

简单还原　完全还原　批日志还原

7.4.2 创建备份设备

使用对象资源管理器创建备份设备

使用系统存储过程sp_addumpdevice创建备份设备

```
sp_addumpdevice  [ @devtype = ] 'device_type' , [ @logicalname
= ] 'logical_name' , [ @physicalname = ] 'physical_name'
```

使用sp_dropdevice来删除备份设备

7.4.3 备份数据库

使用对象资源管理器备份数据库

使用T-SQL语句备份数据库

(1) 完整备份和差异备份
(2) 事务日志备份
(3) 文件和文件组备份

7.4.4 还原数据库

使用对象资源管理器可以很方便地实现对数据库的还原操作。

7.5 小结

- 本章主要讲述了在SQL Server 2012中运用Transact-SQL语句和命令进行程序设计，其中包括局部变量、全局变量、注释符、流程控制命令、一些常用命令和常用函数；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，尤其是在存储过程的设计、触发器的设计方面应用更为广泛。
- 备份和还原是维护数据库安全性和完整性的主要方法，在SQL Server 2012中有四种备份类型，分别为：完整数据库备份、事务日志备份、差异备份、文件和文件组备份，它们的联合使用可以获取较好的备份和效用。还原就是把遭受破坏、丢失的数据或出现错误的数据库，还原到原来的正常状态，在SQL Server 2012中有三种数据库还原模式，它们分别是简单还原、完全还原和批日志还原。