数据库系统概论

第8章　数据库编程

第8章 数据库编程

- 第一节 T-SQL编程基础
- 第二节 游标
- 第三节 存储过程
- 第四节 自定义函数
- 第五节 触发器

教学目标

- 掌握
  - 变量、运算符、控制语句
  - 游标、存储过程、自定义函数、触发器
- 了解
  - 系统函数
- 重点
  - 游标、存储过程、自定义函数、触发器
- 难点
  - 自定义函数

- 第一节 T-SQL编程基础
- 第二节 游标
- 第三节 存储过程
- 第四节 自定义函数
- 第五节 触发器

T-SQL

- T-SQL 即 Transact-SQL，是 SQL 在 Microsoft SQL Server 上的增强版
  - SQL SERVER专用标准结构化查询语言增强版
  - T-SQL 提供标准 SQL 的 DDL 和 DML 功能，加上延伸的函数、系统预存程序以及程序设计结构(例如 IF 和 WHILE)让程序设计更有弹性

T-SQL编程基础

- 标识符
- 注释语句
- 表达式
- 流程控制语句
- 系统函数

标识符

标识符分类

- 常规标识符（严格遵守标识符格式规则）
- 界定标识符（引号”或方括号[]）

标识符格式规则

- **字母或_、@、＃开头**的字母数字或_、@、$序列
- 不与保留字相同
- 长度小于128
- 不符合规则的标识符必须加以界定（双引号”“或方括号[]）

以下（）是T-SQL合法的标识符A. 1FD　B."my book"　C. My book　D. select

注释语句

- 注释语句是对程序代码的说明或暂时禁用，是程序代码中不编译执行的语句
  - 单行注释 --

    例：　--求‘3-105’课程的平均分
  - 多行注释 /**/

    例：　/*　作者：

    　　创建时间：　*/

表达式

- 数据类型
  - 整数数据、字符数据、货币数据、日期和时间数据、二进制字符串等
- 变量
  - 局部变量

    局部变量是用户定义，必须以@开头，在程序内声明，并只能在该程序内使用

    (1)局部变量的声明

    DECLARE @<局部变量名> <数据类型>[,…n]

    (2)局部变量的赋值

    SET|SELECT @<局部变量名>=<表达式>

SET、SELECT区别

❖SET 、 SELECT赋值的区别

- SET赋值语句一般用户赋给变量指定的数据变量
- SELECT赋值语句一般用于表中查询数据，然后查询的记录多于一条，将把最后一条记录的值赋给变量
- SET一次只能赋值一个变量
- SELECT可以一次赋值多个变量
  - SELECT @a=1,@b=2

SET、SELECT区别

例1：创建一个@myvar 变量，然后将一个字符串值放在变量中，最后输出 @myvar 变量的值

```
DECLARE @myvar  char(20)
select  @myvar = 'This is a test'
SELECT @myvar
```

例2： 用SET语句和SELECT语句为局部变量赋值

```
DECLARE @var1 datetime,@var2 char(10)
SET @var1 = getdate()
SELECT @var2 =
convert(char(10),@var1,102)
Select   @var2     --显示@var2
```

表达式

- 全局变量

  全局变量是SQL Server系统内部使用的变量，以@@开头

  - 全局变量不是由用户的程序定义的，它们是在服务器级定义的
  - 用户只能使用预先定义的全局变量

- 例3 ： 用全局变量查看SQL Server的版本、当前所使用的SQL Server服务器的名称以及所使用的服务名称等信息

```
print '目前所用SQL Server的版本信息如下:'
print @@VERSION
print '目前SQL Server服务器名称为：'+@@SERVERNAME
print '目前所用服务器为：'+@@SERVICENAME
```

❖运算符

- SQL Server 2008的运算符和其他高级语言类似，用于指定要在一个或多个表达式中执行的操作，将变量、常量和函数连接起来

优先级	运算符类别	所包含运算符
1	一元运算符	+（正）、-（负）、~（取反）
2	算术运算符	*（乘）、/（除）、%（取模）
3	算术字符串运算符	+（加）、-（减）、+（连接）
4	比较运算符	=（等于）、>（大于）、>=（大于等于）、<（小于）、<=（小于等于）、<>（ 或!=不等于）、!<（不小于）、!>（不大于）
5	按位运算符	&（位与）、|（位或）、^（位异或）
6	逻辑运算符	not（非）
7	逻辑运算符	and（与）
8	逻辑运算符	all（所有）、any（任意一个）、between（两者之间）、exists（存在）、in（在范围内）、like（匹配）、or（或）、some（任意一个）
9	赋值运算符	=（赋值）

流程控制语句

选择结构

循环结构

等待语句

返回语句

IF ELSE

- IF…ELSE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。其中，ELSE子句是可选的， SQL Server允许嵌套使用IF…ELSE语句，而且嵌套层数没有限制
- 语法格式

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

- 例：查找有没有学号为201215120的学生，有的话显示学生信息，没有显式没找到

```
IF   EXISTS( SELECT * FROM STUDENT WHERE SNO = '201215120')
BEGIN
      SELECT *
      FROM  STUDENT
      WHERE  SNO = '201215120'
END
ELSE
      PRINT '没找到！'
```

EXISTS 子查询　如果子查询结果非空返回真，否则返回假

CASE语句

- CASE语句可以计算多个条件式，并将其中一个符合条件的结果表达式返回。CASE语句按照使用形式的不同，可以分为简单CASE语句和搜索CASE语句

  - 简单CASE语句

```
CASE 表达式
    WHEN 表达式的值1  THEN 返回表达式1
    WHEN 表达式的值2  THEN 返回表达式2
    …
ELSE 返回表达式n
END
```

CASE语句

❖例：从学生表STUDENT中，选取SNO,SSEX，如果SSEX为‘男’则输出‘M’，如果为‘女’，则输出‘F’

```
SELECT  SNO,SSEX=
            CASE  SSEX
            WHEN ‘男’ THEN ‘M’
            WHEN ‘女’ THEN ‘F’
            END
FROM  STUDENT
```

搜索式CASE语句

- 搜索式CASE

```
CASE
    WHEN 逻辑表达式1 THEN 返回表达式1
    WHEN 逻辑表达式2 THEN 返回表达式2
    …
    ELSE 返回表达式n
END
```

搜索式CASE语句

- 例：从SC表中查询所有同学选课成绩情况，凡成绩为空者输出‘未考’、小于60输出‘不及格’、小于70输出‘及格’、小于90输出‘良好’、大于等于90输出‘优秀’

```
SELECT  SNO,CNO,GRADE=
        CASE
                WHEN GRADE IS NULL THEN ‘未考’
                WHEN GRADE<60  THEN ‘不及格’
                WHEN GRADE>=60 AND GRADE<70 THEN ‘及格’
                WHEN GRADE>=70 AND GRADE<90 THEN ‘良好’
                WHEN GRADE>90  THEN ‘优秀’
        END
FROM  SC
```

循环结构

- 设置重复执行 SQL 语句或语句块的条件。只要指定的条件为真，就重复执行语句。可以使用 BREAK 和 CONTINUE 关键字在循环内部控制 WHILE 循环中语句的执行

```
WHILE 逻辑表达式
Begin
   T-SQL语句组
   [break]/*终止整个语句的执行*/
   [continue]/*结束一次循环体的执行*/
END
```

❖例 求1 ~10的和

```
DECLARE @X int, @sum int
SET @X=0
SET @sum = 0
WHILE @x<10
   BEGIN
     SET @X=@X+1
     SET @sum = @sum + @X
     PRINT ‘sum='+convert(char(2),@sum) --类型转换函数convert
   END
```

等待语句

- 等待语句挂起一个程序中语句的执行，直到指定的某一时间点到来或在一定的时间间断之后才继续执行
- 语法格式

WAITFOR DELAY '<时间间隔>' | TIME '<时间>'

  - 其中，时间间隔以及时间均为datetime类型，格式为“hh:mm:ss”，分别说明等待的时间长度和时间点，在time内不能指定日期

等待语句

- 例1：设置等待一小时后执行查询

```
BEGIN
    WAITFOR DELAY '1:00:00’
    SELECT * FROM s
END
```

- 例2：设置到十点整执行查询

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

返回语句

RETURN语句

- RETURN语句用于无条件地终止一个查询、存储过程或者批处理，此时位于RETURN语句之后的程序将不会被执行
- 语法格式

```
RETURN [ integer_expression ]
```

系统函数

❖标量函数

函数分类	解释
配置函数	返回当前的配置信息
游标函数	返回有关游标的信息
日期和时间函数	对日期和时间输入值进行处理
数学函数	对作为函数参数提供的输入值执行计算
元数据函数	返回有关数据库和数据库对象的信息
安全函数	返回有关用户和角色的信息
字符串函数	对字符串（char 或 varchar）输入值执行操作
系统函数	执行操作并返回有关SQL Server中的值、对象和设置的信息
系统统计函数	返回系统的统计信息
文本和图像函数	对文本或图像输入值或列执行操作，返回有关这些值的信息

日期和时间函数

函数	参数	功能
DATEADD	（datepart,number,date）	以datepart指定的方式，返回date加上number之和
DATEDIFF	（datepart,date1,date2）	以datepart指定的方式，返回date2与date1之差
DATENAME	（datepart,date）	返回日期date中datepart指定部分所对应的字符串
DATEPART	（datepart,date）	返回日期date中datepart指定部分所对应的整数值
DAY	（date）	返回指定日期的天数
GETDATE	（）	返回当前的日期和时间
MONTH	（date）	返回指定日期的月份数
YEAR	（date）	返回指定日期的年份数

日期和时间函数

- 例：使用日期时间函数计算自己现在的年龄

```
SELECT '年龄'=
        DATEDIFF(YY,'1979-06-01',GETDATE())
```

- 例：返回指定日期中年/月/日的整数

```
SELECT  YEAR('2016-01-08')
```

字符串函数

字符串转换函数

- ASCII（字符串） 函数返回字符表达式最左端字符的ASCII 码值
- CHAR（整数表达式） 函数用于将ASCII 码转换为字符
- LOWER（字符串） 函数把字符串全部转换为小写
- UPPER（字符串） 函数把字符串全部转换为大写

去空格函数

- LTRIM（字符串） 函数把字符串头部的空格去掉
- RTRIM（字符串） 函数把字符串尾部的空格去掉

取子串函数

- LEFT(字符串,int num)函数返回的子串是从字符串最左边起到第num个字符的部分。若num为负值,则返回NULL值

字符串函数

- RIGHT(字符串,int num)函数返回的子串是从字符串最右边起到第num个字符的部分。若num为负值,则返回NULL值
- SUBSTRING (字符串,int pos,int length）函数返回的子串是从字符串左边第starting_position 个字符起length个字符的部分。SUBSTRING（）函数不能用于TEXT 和 IMAGE 数据类型

❖字符串比较函数

- CHARINDEX (<要找的子串>，字符串)函数返回字符串中某个指定的子串出现的开始位置，如果没有发现子串，则返回0 值
- PATINDEX (<’%substring _expression%’>， <字符串>) 函数返回字符串中某个指定的子串出现的开始位置，其中子串表达式前后必须有百分号 “%”否则返回值为0

字符串函数

❖字符串操作函数

- QUOTENAME（） 函数返回被特定字符括起来的字符串
- REPLICATE（） 函数返回一个重复character_expression 指定次数的字符串
- REVERSE（） 函数将指定的字符串的字符排列顺序颠倒
- REPLACE（） 函数返回被替换了指定子串的字符串
- SPACE（） 函数返回一个有指定长度的空白字符串
- STUF（）函数用另一子串替换字符串指定位置、长度的子串

数据类型转换函数

- 转换函数有两个：CONVERT和CAST
  - CAST函数允许把一个数据类型强制转换为另一种数据类型，其语法形式为：
    - CAST(expression AS data_type)
  - CONVERT函数允许用户把表达式从一种数据类型转换成另一种数据类型，还允许把日期转换成不同的样式，其语法形式为：
    - CONVERT (data_type[(length)],expression [,style])

数据类型转换函数

- 例：

查询学生基本信息表STUDENT中的学号、姓名、年龄，并且将这三个字段通过“+”运算符连接显示在查询结果中

```
SELECT  SNO+SNAME+CAST(SAGE AS CHAR(2))
FROM STUDENT
```

- 第一节 T-SQL编程基础
- 第二节 游标
- 第三节 存储过程
- 第四节 自定义函数
- 第五节 触发器

游标

- 游标是一种能从包括多条数据记录的结果集中每次提取一条记录的机制
- 游标的使用
  - 声明游标
  - 打开游标
  - 读取游标中的数据
  - 关闭游标
  - 释放游标

1. 声明游标

❖语法格式：

DECLARE <游标名> [INSENSITIVE] [SCROLL]

CURSOR

FOR <SELECT语句>

[FOR {READ ONLY | UPDATE [OF <列名> [,...n]]}]

- Insensitive 指定游标只对基本表的副本操作，游标的任何操作不对基本表产生影响
- Scroll 指定游标推进方向（FIRST、LAST、PRIOR、NEXT、RELATIVE、ABSOLUTE）均可用，否则只有next可用

声明游标

- 例：声明一个游标，统计没有选修课程的学生的人数

```
declare  num_cursor  cursor
for
   select sno
   from student
for READ ONLY
```

2. 打开游标

- 游标声明后，如果要从游标中读取数据，必须打开游标

OPEN <游标名>

open num_cursor;

3. 读取游标中的数据

- 当游标被打开后，就可以从游标中逐行地读取数据

FETCH

[[NEXT|PRIOR|FIRST|LAST|ABSOLUTE {n|@nvar}

|RELATIVE {n|@nvar}]

FROM]

{{[GLOBAL] <游标名>}|<@游标变量>}

[INTO @<变量名>[,...n]]

- 默认情况下，指针指向第一条记录之前

读取游标中的数据

参 数	参 数 说 明
NEXT	返回结果集中当前行的下一行，并递增当前记录行数为返回行行数。
PRIOR	返回结果集中当前行的前一行，并递减当前行数为返回行行数。
FIRST	返回游标中的首记录并设为当前记录。
LAST	返回游标中的末记录并设为当前记录。
ABSOLUTE	ABSOLUTE是绝对位置，它将指针移到第n行位置。
RELATIVE	若n 或@nvar 为正则读取游标当前位置起向后的第@nvar或n行数据，如果n 或@nvar 为负则读取游标当前位置起向前的第n 或@nvar 行数据。
INTO	允许将使用FETCH 语句读取的数据存放在若干个局部变量中，在变量行中的每个变量必须与游标结果集中相应的列相对应（包括类型）。

读取游标中的数据

- 执行FETCH语句后，可通过@@FETCH_STATUS全局变量返回游标当前的状态。@@FETCH_STATUS 变量有三个不同的返回值
  - 0：FETCH 语句执行成功
  - -1：FETCH 语句执行失败或者行数据超出游标数据结果集的范围
  - -2：表示提取的数据不存在

```
declare num_cursor cursor    --声明
for    select sno   from student
for READ ONLY
Open num_cursor;   --打开
declare @sno  varchar(10),@num int  --声明变量
set @num = 0

fetch next from num_cursor  --取信息
into @sno
while @@fetch_status = 0  --检测状态
begin
    if not exists(select *
       from  sc
       where sno = @sno)
     set @num = @num + 1
    fetch next from num_cursor
    into @sno
end
select @num 未选课人数
CLOSE num_cursor
DEALLOCATE num_cursor
```

4. 关闭游标

- 使用CLOSE命令关闭游标
  - 处理完游标中数据后，必须关闭游标来释放数据结果集和定位于数据记录上的锁
  - 语法格式:

    CLOSE [GLOBAL] <游标名>|@<游标变量>

  - CLOSE语句可以关闭游标，但不释放游标的数据结构
    - 如果要再次使用游标，可用OPEN命令重新打开
- 自动关闭游标

5. 释放游标

- 用CLOSE命令关闭游标并没有释放游标占用的数据结构。使用DEALLOCATE命令将释放游标占用的数据结构，游标使用的任何资源也随之释放
  - 语法格式

    DEALLOCATE [GLOBAL] <游标名>|@<游标变量>

  - 游标的关闭指释放游标的结果集所占用的资源，游标的释放指释放游标占用的所有资源，当然也包括结果集占用的资源

❖例：根据学生成绩计算统计各个等级的人数：

[90-100]为A, [80-89]为B

[70-79]为C, [60-69]为D

[0-59]为E

```
-- 定义局部变量
DECLARE @mygrade int,@mylevel char(1)
DECLARE @E int,@D  int,@C   int,@B   int,@A   int
select @E = 0,@D  = 0,@C   = 0,@B   = 0,@A   = 0

-- 下面定义游标.
DECLARE level_cursor CURSOR FOR
SELECT grade
FROM students.sc

-- 下面打开游标.
OPEN level_cursor
```

```
-- 下面从游标中取出第一行，放到
--对应的变量中.
FETCH NEXT FROM level_cursor
INTO @mygrade

--循环处理
WHILE @@FETCH_STATUS = 0
BEGIN
            -- 计算级别.
 if @mygrade is null
  set @E = @E + 1
 else
  if @mygrade<60
    set @E = @E+1
  else
    if @mygrade<70
       set @D  = @D  +1
    else
      if @mygrade<80
            set @C   = @C   + 1
          else
            if @mygrade < 90
              set @B   = @B   + 1
            else
              set @A   = @A   + 1
          -- 从游标中取下一行.
          FETCH NEXT FROM level_cursor
          INTO @mygrade
END

-- 关闭游标.
CLOSE level_cursor

--释放资源
DEALLOCATE level_cursor

select  @E,@D,@C,@B,@A
```

课堂练习

- 根据员工工资计算其个人所得税，3000元为起征点，超出3000元的部分按照10%的比例征收个人所得税

例如：工资表如下

员工编号	工资	个人所得税
1	3100	
2	3500	
3	3800	
……	……	

1号员工个人所得税为10元，2号员工个人所得税为50元，3号员工个人所得税为80元…，请使用游标编写一段Transact-SQL程序段，计算每个员工的个人所得税并更新员工工资表中的个人所得税

```
-- 定义局部变量
declare @salary  int ,@cno int
--声明游标
declare  salary_cursor cursor
for
   select 工资，员工编号
   from 员工工资表
for read only
open salary_cursor ;  -- 打开游标.
fetch next from salary_cursor
into @salary,@cno
--循环处理
while @@FETCH_STATUS = 0
begin
            if @salary>3000
            begin
            update 员工工资表 set 个人所得税=@salary*0.1 where  员工编号=@cno;
            end
end
-- 关闭游标.
close salary_cursor
--释放资源
deallocate salary_cursor
```

第8章 数据库编程

- 第一节 T-SQL编程基础
- 第二节 游标
- 第三节 存储过程
- 第四节 自定义函数
- 第五节 触发器

存储过程

- 存储过程（Stored Procedure）是一组完成特定功能的SQL语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行存储过程
- 存储过程的优点
  - 存储过程已在服务器注册
  - 存储过程具有安全特性
  - 存储过程可以强制应用程序的安全性
  - 存储过程允许模块化程序设计
  - 存储过程可以减少网络通信流量

存储过程

创建存储过程

执行存储过程

删除存储过程

1. 创建存储过程

❖语法：

CREATE { PROC | PROCEDURE } [schema_name.] procedure_name

[{ @parameter data_type }[VARYING] [= default] [OUTPUT]] [,...n]

[WITH RECOMPILE | ENCRYPTION]

AS

{ <BEGIN>

  <sql_statement> [;][...n]

 <END>

}

示例

- 例1：从SC表中查询不及格课程超过3门的学生信息

```
create procedure  myproc
as
select *
from student
where sno in
        (
           select sno
           from sc
           where grade<60
           group by sno
           having count(*)>3
        );
```

示例

❖例2：将指定记录插入student表

```
create proc  proc_insert_student
   @sno  varchar(10),
   @sname varchar(20),
   @ssex varchar(2) = "男",
   @sage smallint,
   @sdept varchar(50)
as
begin
      insert into  student(sno,sname,ssex,sage,sdept)
      values(@sno,@sname,@ssex,@sage,@sdept)
end
```

给输入参数指定默认值

带输出参数的存储过程

❖例3 查询指定学号学生的平均成绩，并将平均成绩返回

```
create proc proc_avergrade
   @sno  varchar(10),
   @savg int out   --输出参数
as
begin
    select @savg = avg(grade)
    from sc
    where sno = @sno
end
```

课堂练习一

1.编写一个存储过程，在sc表统计每个学生的平均分

2.对练习1的存储过程进行改进，添加一个输入参数——学号，使存储过程能根据输入的学号计算该学生的平均分

3.在练习2的基础上添加一个输出参数——平均分，计算指定学号的平均分，然后将平均分输出

课堂练习二

- 请编写一个存储过程proc_sum，输入参数为学院，输出参数为人数，功能为根据输入的学院，统计该学院的学生人数，并返回学生人数。学生表的结构为（sno,sname,sex,department）各个字段含义为学号、姓名、性别、学院

```
create proc proc_sum
@department char(9),
@p_num int out
as
begin
  select @p_num=count(*)
  from student
  where department=@department
end
```

2. 执行存储过程

❖语法格式：

```
EXEC|EXECUTE  [ @return_status = ] [schema_name.]procedure_name
    [[@parameter =] {value | @variable [OUTPUT] | [ DEFAULT ]}][ ,...n ]
```

❖例 执行例2 的存储过程

```
exec  proc_insert_student '200901031','张三','男',18,'软件学院'
```

❖例 执行例3的存储过程

```
declare @avg int
set @avg = 0
exec  proc_avergrade '200215121',@avg out
select @avg
```

3. 删除存储过程

- 删除存储过程可以使用DROP命令，DROP命令可以将一个或者多个存储过程或者存储过程组从当前数据库中删除，其语法形式如下：

```
drop procedure {procedure_name}[,…n]
```

第8章 数据库编程

- 第一节 T-SQL编程基础
- 第二节 游标
- 第三节 存储过程
- 第四节 自定义函数
- 第五节 触发器

自定义函数

- SQL Server 2008支持3种类型的Transact-SQL用户自定义函数：标量函数、内嵌表值函数和多语句表值函数
  - 标量函数返回一个标量(单值)结果
  - 内嵌表值函数返回一个table数据类型
  - 多语句表值函数返回的数据必须存放于临时表中（性能不好）
- 在SQL Server中使用用户自定义函数有以下优点:
  - 允许模块化程序设计
  - 执行速度更快
  - 减少网络流量

1.标量函数

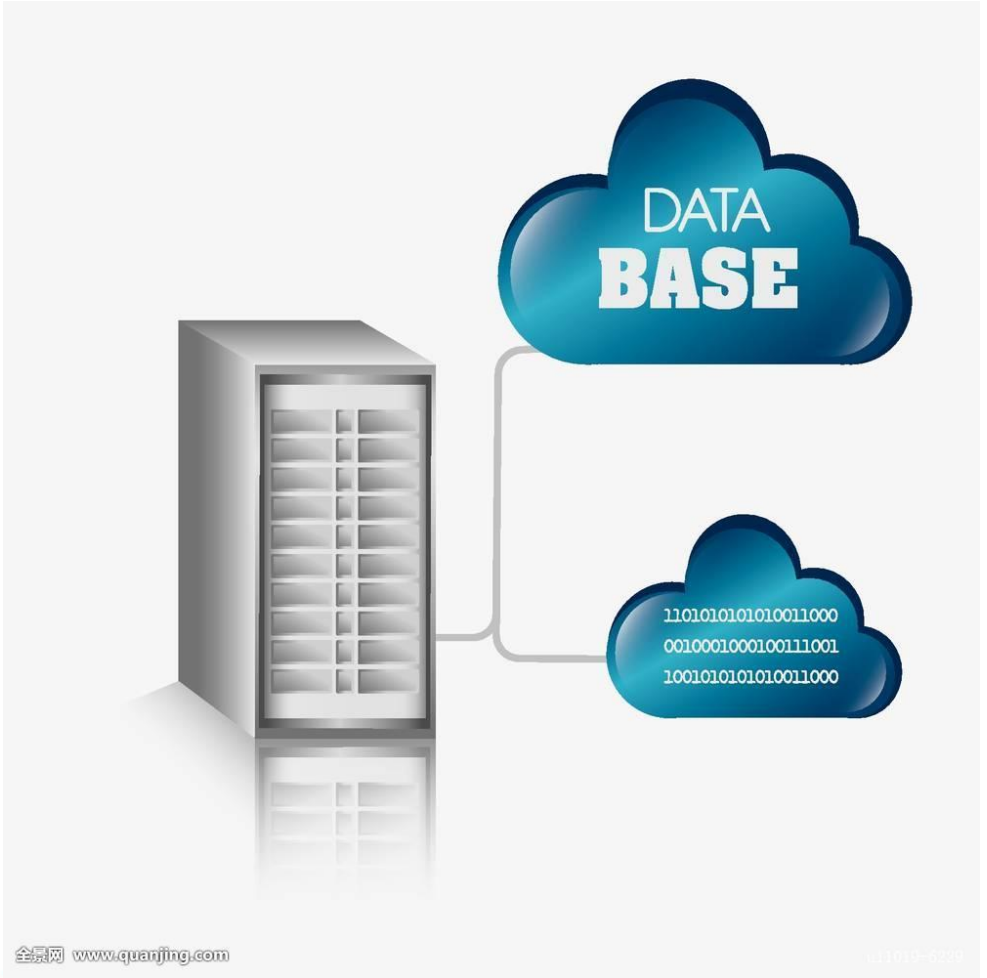
❖创建语法：

```
CREATE FUNCTION [ schema_name.] function_name
([{ @parameter_name [ AS ] data_type [ = default ] } [ ,...n ]])
RETURNS return_data_type
[ WITH <ENCRYPTION>|<SCHEMABINDING> [ ,...n ] ]
[ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
```

自定义日期函数

- 定义一个函数返回不带时间的日期

```
CREATE FUNCTION dbo.DateOnly(@date datetime)
RETURNS VARCHAR(12)
AS
BEGIN
    RETURN CONVERT(VARCHAR(12),@DATE,101)
END
```



课堂练习

- 请写一个函数whichgeneration，根据输入的参数生日 (birthday datetime) 来判断学生是“80后”还是“90后”，如果是“80后”返回字符串“80s”，如果是“90后”返回“90s”，其他返回“too old”

课堂练习

```
CREATE FUNCTION whichgeneration(@birthday datetime)
RETURNS VARCHAR(12)
AS
BEGIN
        if year(@birthday)<1980
        return “too old”;
        else if year(@birthday)<1990
        return “80s”;
        else
         return “90s”;
END
```

2.内嵌表值函数

❖创建语法：

```
CREATE FUNCTION [ schema_name. ] function_name
([{ @parameter_name  data_type [ = default ] } [ ,...n ]])
RETURNS TABLE
[ WITH <function_option> [ ,...n ] ]
[ AS ]
    RETURN ( select_stmt )
```

内嵌表值函数示例

- 定义一个函数返回学生的学号和姓名

```
CREATE FUNCTION dbo.Fun1()
RETURNS table
AS
RETURN
    SELECT SNO,SNAME
    FROM  STUDENT
```

内联表值函数示例

- 查看计算机系学生的成绩

```
CREATE FUNCTION [dbo].[attendance]
(
   @sdept  varchar(20)  --系
)
RETURNS TABLE
AS
RETURN
(
    SELECT student.sno,cno,grade
    from sc,student
    where student.sno = sc.sno  and
          sdept = @sdept
)
```

3.多语句表值函数

❖创建语法：

```
CREATE FUNCTION [ schema_name. ] function_name
([{@parameter_name  data_type [ = default ] } [ ,...n ]])
RETURNS @return_variable TABLE < table_type_definition >
[ WITH <function_option> [ ,...n ] ]
[ AS ]
   BEGIN
      function_body
      RETURN
   END
```

多语句表值函数示例

- 现有一个员工表test （字段省），一个部门表bm （字段省）。查询某部门的员工信息
  - 创建内嵌表值函数

```
CREATE FUNCTION  fn_bumen
(@bm char(2))
returns table
as
return (select * from test where 部门=@bm)
```

多语句表值函数示例

- 创建多语句表值函数

```
CREATE FUNCTION fn_salary ( @bm char(2) )
RETURNS @salary table
(姓名 varchar(10),部门名称 varchar(10),工资 numeric(8,2))
as
begin
insert @salary
select a.姓名,b.部门名称,a.工资
from test a left join bm b on a.部门=b.部门 and a.部门=@bm
return
end
```

视图、存储过程和自定义函数

	视图	存储过程	自定义函数
语句	只能是SELECT语句	可以包含程序流、逻辑以及SELECT语句	可以包含程序流、逻辑以及SELECT语句
输入	不能接受参数	可以有输入输出参数	有输入参数
返回值	只能返回结果集	返回值只能是整数	可以返回标量值、表
典型应用	多个表格的连接查询	完成某个特定的较复杂的任务	可以完成比较复杂的任务，可以出现在select语句中

- 第一节 T-SQL编程基础
- 第二节 游标
- 第三节 存储过程
- 第四节 自定义函数
- 第五节 触发器

触发器

- 触发器（Trigger）是用户定义在关系表上的一类由事件驱动的特殊过程
  - 由服务器自动激活
  - 可以进行更为复杂的检查和操作，具有更精细和更强大的数据控制能力
- SQL SERVER2008触发器
  - DML触发器
  - DDL触发器

SQL SERVER2008触发器

- 触发器是一种特殊的存储过程，它在执行事件时自动生效。SQL Server2008 包括两大类触发器：DML 触发器和 DDL 触发器
  - DML 触发器在数据库中发生数据操作语言 (DML) 事件时将启用。DML 事件包括在指定表或视图中修改数据的 INSERT 语句、UPDATE 语句或 DELETE 语句。DML 触发器可以查询其他表，还可以包含复杂的 Transact-SQL 语句。将触发器和触发它的语句作为可在触发器内回滚的单个事务对待。如果检测到错误（例如，磁盘空间不足），则整个事务即自动回滚
  - DDL 触发器是 SQL Server 2008 的新增功能。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

触发器的作用

- 触发器可以对数据库进行级联修改
- 实现比CHECK约束更为复杂的限制
- 比较数据修改前后的差别
- 强制表的修改要合乎业务规则

DML触发器

- DML触发器是在对表进行插入、更新或删除操作时自动执行的存储过程
  - 触发器定义在特定的表上，与表相关
  - 自动触发执行
  - 不能直接调用
  - 是一个事务（可回滚）
- 分类
  - DELETE 触发器
  - INSERT 触发器
  - UPDATE 触发器

Inserted表和Deleted表

- SQL Server 2008为每个触发器都创建了两个专用临时表：Inserted表和Deleted表。这两个表的结构总是与被该触发器作用的表的结构相同，触发器执行完成后，与该触发器相关的这两个表也会被删除

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		存放要除的旧记录

Inserted和Deleted表

- 触发器触发时
  - 系统自动在内存中创建deleted表或inserted表
  - 只读，不允许修改；触发器执行完成后，自动删除
- Inserted 表
  - 临时保存了插入或更新后的记录行
  - 可以从inserted表中检查插入的数据是否满足业务需求
  - 如果不满足，则向用户报告错误消息，并回滚插入操作
- Deleted 表
  - 临时保存了删除或更新前的记录行
  - 可以从deleted表中检查被删除的数据是否满足业务需求
  - 如果不满足，则向用户报告错误消息，并回滚插入操作

创建DML触发器

- 创建DML触发器的语法格式为：

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ NOT FOR REPLICATION ]
AS
begin  sql_statement  [ ; ] end
```

使用INSERT触发器

- INSERT触发器通常被用来更新时间标记字段，或者验证被触发器监控的字段中数据满足要求的标准，以确保数据的完整性
  - 例：建立一个触发器，当向sc表中添加数据时，如果添加的数据与student表中的数据不匹配（没有对应的学号），则将此数据删除

```
CREATE TRIGGER tr_sc_insert    ON sc
FOR INSERT
AS
BEGIN
    DECLARE @bh char(10)
    Select @bh=Inserted.sno from Inserted
    If not exists(select sno from student where student.sno=@bh)
        Delete from sc where sno=@bh
END
```

插入记录行

表 - students.SC | WIN2K3.stud...uery15.sql* | WI

Sno	Cno	Grade
200215121	1	90
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80
200711011	4	90
1	1	1
NULL	NULL	NULL

向inserted表中插入新行的副本，触发insert触发器。

inserted

Sno	Cno	Grade
1	1	1

触发器检查inserted表中插入的新行数据，确定是否需要回滚或执行其他操作

INSERT触发器

- 例：创建一个触发器，当插入或更新成绩列时，该触发器检查插入的数据是否处于设定的范围内（0，100）

```
CREATE TRIGGER tr_sc_grade
  ON  sc
  AFTER INSERT,UPDATE
AS
BEGIN
            -- SET NOCOUNT ON added to prevent extra result sets from
            -- interfering with SELECT statements.
            SET NOCOUNT ON;

           DECLARE @score int;
            SELECT @score=inserted.grade from inserted
            IF (@score<0 or @score > 100)
            BEGIN
              RAISERROR ('成绩的取值必须在0到100之间', 16, 1)
              ROLLBACK TRANSACTION
            END
END
```

Raiserror函数

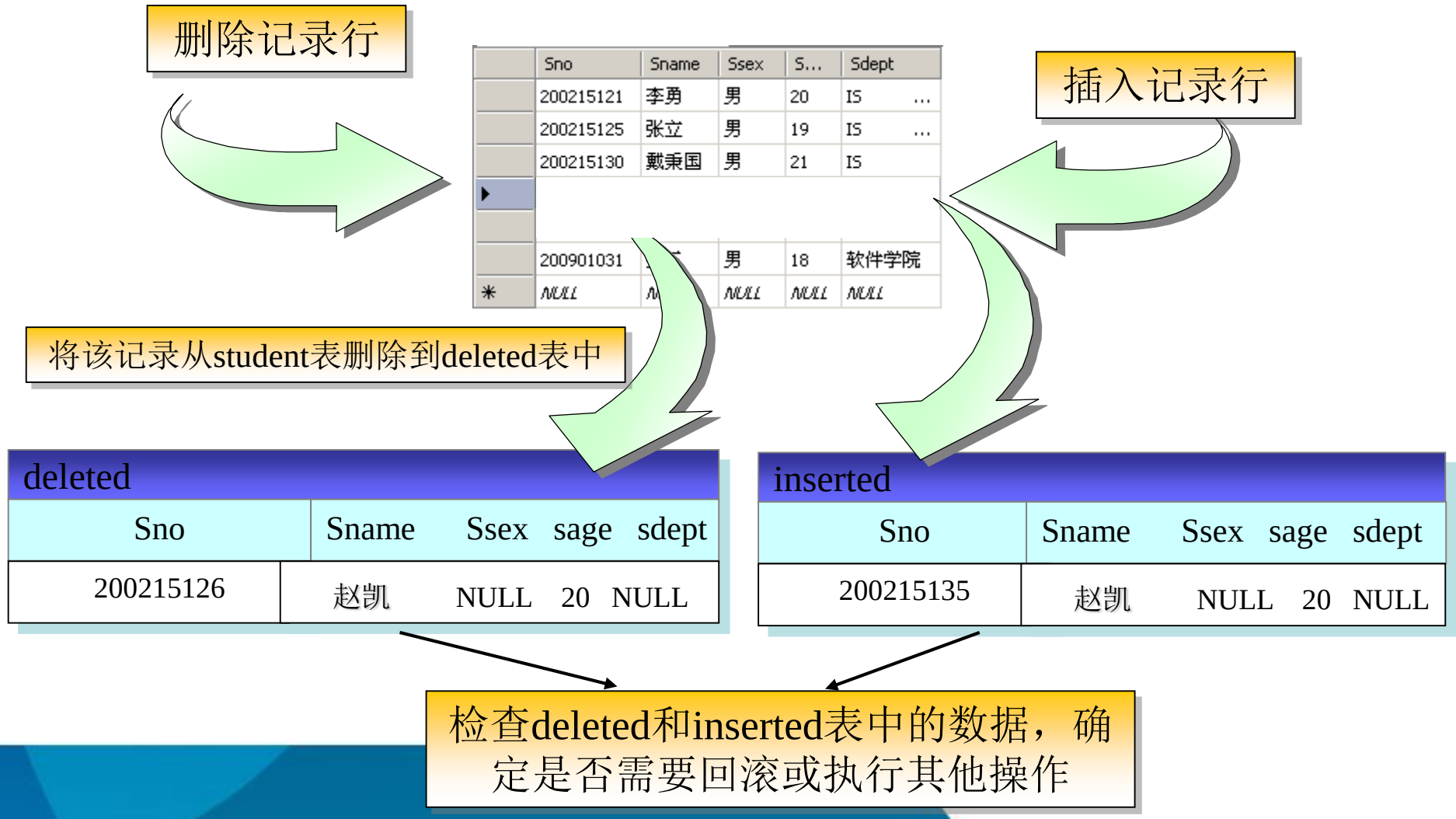
- Raiserror 函数的作用：抛出一个错误
- 参数
  - 第一个参数：错误的提示消息
  - 第二个参数：错误的消息级别（0~18之间）
  - 第三个参数：错误的状态号（1~127之间）
    - 如果在多个位置引发相同的用户定义错误，则针对每个位置使用唯一的状态号有助于找到引发错误的代码段

使用UPDATE触发器

- 当在一个有UPDATE触发器的表中修改记录时，表中原来的记录被移动到删除表中，修改过的记录插入到了插入表中，触发器可以参考删除表和插入表以及被修改的表，以确定如何完成数据库操作

创建一个修改触发器，该触发器防止用户修改表student的学号

```
CREATE TRIGGER tr_student_sno    ON  student
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    if update(sno)
    begin
                raiserror('不能修改学号',16,10)
                rollback transaction
    end
END
```



Sno	Sname	Ssex	S...	Sdept
200215121	李勇	男	20	IS ...
200215125	张立	男	19	IS ...
200215130	戴秉国	男	21	IS
200901031		男	18	软件学院
NULL		NULL	NULL	NULL

deleted

Sno	Sname	Ssex	sage	sdept
200215126	赵凯	NULL	20	NULL

inserted

Sno	Sname	Ssex	sage	sdept
200215135	赵凯	NULL	20	NULL

使用DELETE触发器

- DELETE触发器通常用于两种情况，第一种情况是为了防止那些确实需要删除但会引起数据一致性问题的记录的删除，第二种情况是执行可删除主记录的子记录的级联删除操作

例 建立一个与sc表结构一样的表s1，当删除表sc中的记录时，自动将删除掉的记录存放到s1表中

```
CREATE TRIGGER tr_student_delete
  ON  sc
  AFTER DELETE
AS
BEGIN
   SET NOCOUNT ON;
   insert into s1 select * from deleted
END
```

删除记录行

表 - students.SC | WIN2K3.stud...LQuery4.sql*

Sno	Cno	Grade
200215121	1	90
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80
200711011	4	90
NULL	NULL	NULL

将该记录从sc表删除到deleted表中插入要删除行的副本，删除然后触发delete触发器。

deleted

Sno	Cno	Grade
200215121	3	88

触发器检查deleted表中被删除的数据，确定是否需要回滚或执行其他操作

使用DELETE触发器

❖例：当删除表student中的记录时，自动删除表sc中对应学号的记录

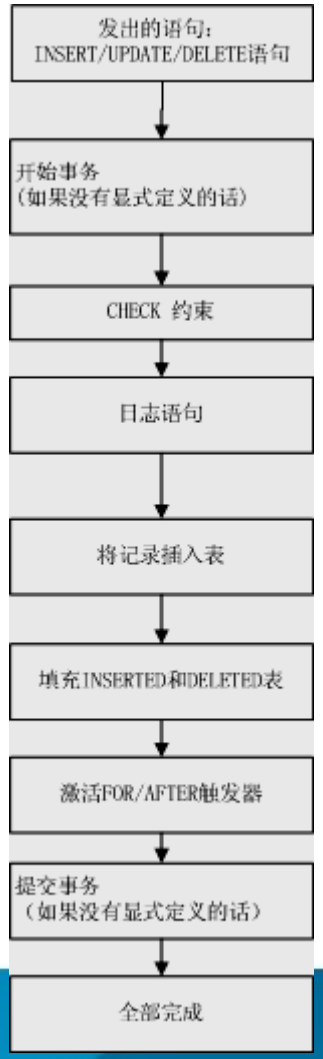
```
CREATE TRIGGER tr_student_sc_delete
   ON  student
   AFTER DELETE
AS
BEGIN
            SET NOCOUNT ON;
            DECLARE @sno char(10)
            Select @sno=deleted.sno from deleted
            Delete  from sc where sno=@sno
END
```

INSTEAD OF触发器实例

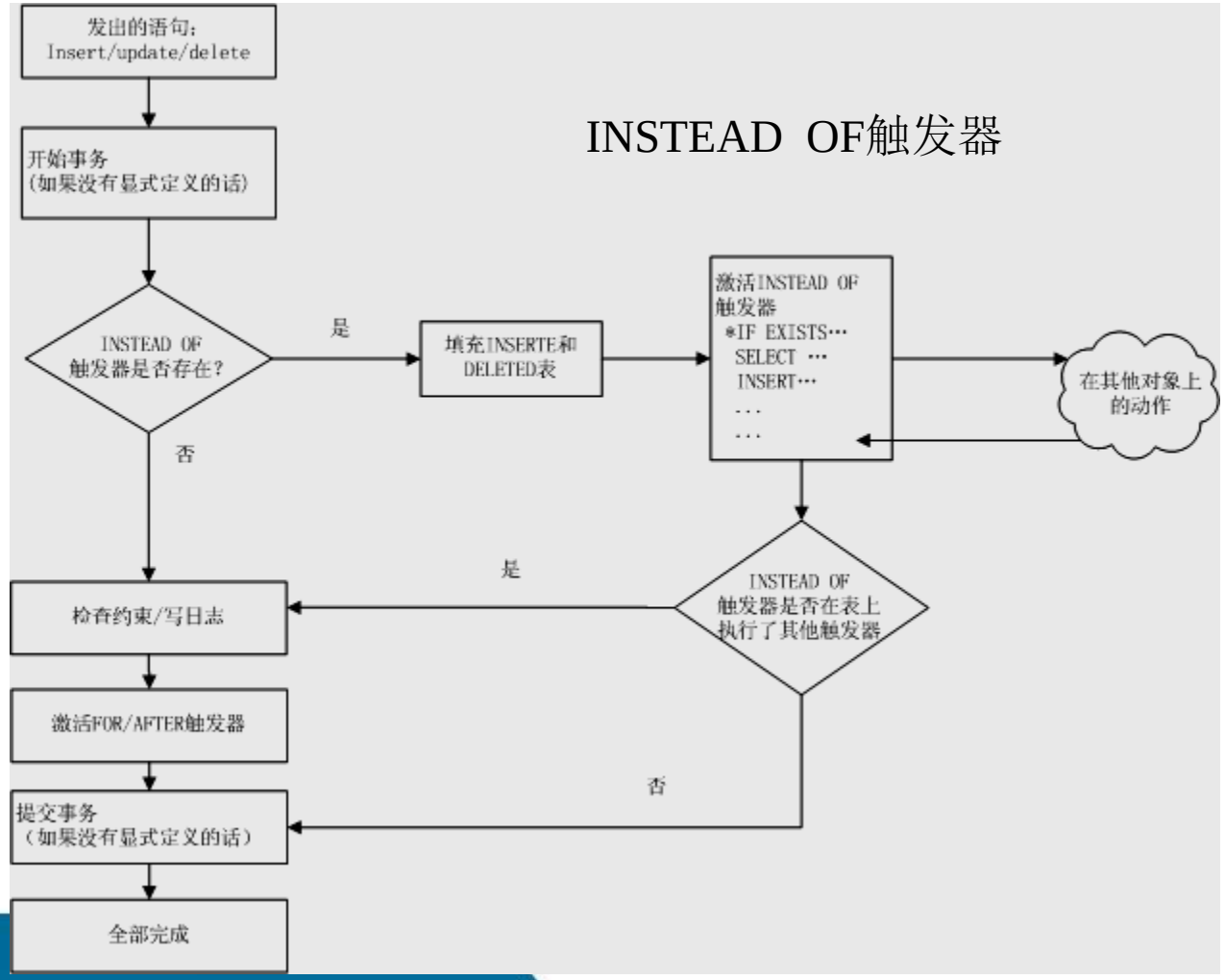
- 在student表删除学生记录的同时删除学生的选课记录

```
CREATE  TRIGGER tr_student_instead
   ON  student
   instead of   DELETE
AS
BEGIN
    SET NOCOUNT ON;
    delete from  sc
    where sno in (    select deleted.sno      from deleted    )

    delete from  student
    where sno in (      select deleted.sno      from deleted     )

END
```

DML触发器执行过程



AFTER/FOR触发器



INSTEAD OF触发器

综合练习

- 某书店后台数据库的部分关系模式如下
  - **图书类别**（类别代号，类别名）
  - **图书**（书号，书名，ISBN，作者，单价，类别代号）
  - **订单**（订单号，顾客编号，订购日期，出货日期）
  - **订单明细**（订单号，书号，数量，总价）

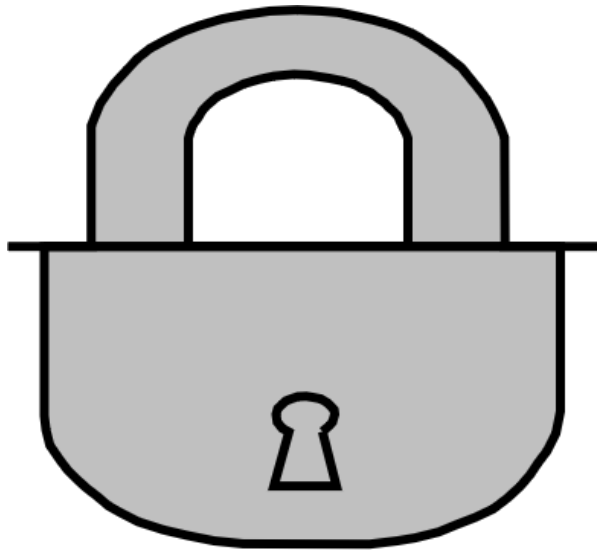
综合练习

- 创建一个函数FunBook，根据用户提供的图书类别名查看相应类别图书的详细信息
- 创建一个触发器TrInsUpd，当向图书表中插入或更新一条记录的类别代号时，新记录的类别代号必须在图书类别表中存在，否则提示类别代号不正确
- 创建一个存储过程PrcSelect，根据用户提供的图书类别名查看相应类别图书的详细信息
- 创建一个函数FunBookSale，该函数根据给定的书号返回该图书销售的数量

```
Create function FunBook(@className varchar )
Return table
as
begin
 return select 图书.* from 图书,图书类别 where 图书.类别代号=图书类别.
图书代号 and 类别名=@className
end

CREATE TRIGGER TrInsUpd
  ON  图书
  BEFORE INSERT,UPDATE
AS
BEGIN
        DECLARE @dh int;
        SELECT @dh=inserted.类别代号 from inserted
        IF (not exists(select 类别代号 from 图书类别 where 类别代号=@dh))
        BEGIN
          RAISERROR (‘类别编号不存在', 16, 1)
          ROLLBACK TRANSACTION
        END
END
```

```
create procedure Prcselect
@classname varchar
as
begin
 select 图书.* from 图书,图书类别 where 图书.类别代号=图书类别.图书代号 and 类别名=@className
 end
```

```
create function FunBookSale (@bookno varchar)
returns  int
as
begin
 return  select  sum（数量） from 订单 where 书号=@bookno
end
```



知者乐水，仁者乐山。知者动，仁者静。知者乐，仁者寿。