1. T-SQL编程基础
2. 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

1. 标识符
2. 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[]）

1. 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[]）

1. 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 --

多行注释 /\*\*/

1. 表达式
2. 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

1. 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

·局部变量的声明

DECLARE @<局部变量名> <数据类型>[,…n]

·局部变量的**赋值**

SET|SELECT @<局部变量名>=<表达式>

1. 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

示例：SELECT @a=1,@b=2

1. 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

1. 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



1. 流程控制语句
2. 选择结构
   1. 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>]

示例：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（EXISTS 子查询 如果子查询结果非空返回真，否则返回假）

IF EXISTS( SELECT \* FROM STUDENT WHERE SNO = ‘201215120’)

BEGIN

SELECT \*

FROM STUDENT

WHERE SNO = ‘201215120’

END

ELSE

PRINT ‘没找到！’

* 1. 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

示例：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

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

示例：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

1. 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

示例：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)

END

1. 等待结构

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

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

示例1：设置等待一小时后执行查询

BEGIN

WAITFOR DELAY '1:00:00’

SELECT \* FROM s

END

示例2：设置到十点整执行查询

BEGIN

WAITFOR TIME '10:00:00'

SELECT \* FROM s

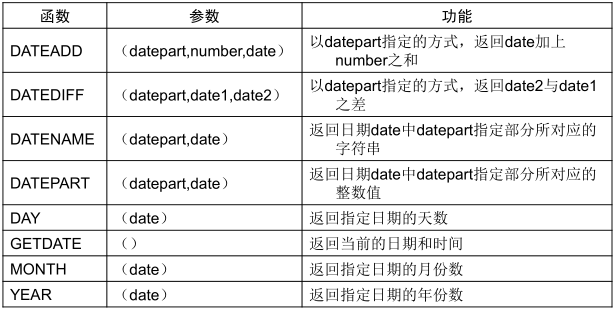
END

1. 返回语句return

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

RETURN [ integer\_expression ]

1. 系统函数

示例1：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())

数据类型转换函数

·CAST函数允许把一个数据类型强制转换为另一种数据类型

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

CONVERT (data\_type[(length)],expression [,style])

1. 游标
2. 概述

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

1. 声明游标

**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

示例：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**

1. 打开游标

**OPEN < 游标名>**

游标声明后，如果要从游标中读取数据，必须打开游标

示例：**open** num\_cursor;

1. 读取游标中的数据

**FETCH**

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

**|RELATIVE {n|@nvar}]**

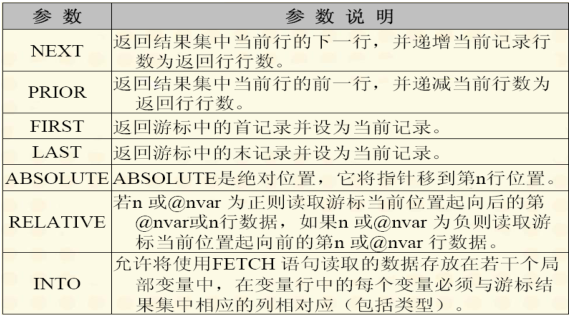
**FROM ]**

**{{[GLOBAL] < 游标名>}|<@ 游标变量>}**

**[INTO @< 变量名>[,...n]]**

当游标被打开后，就可以从游标中逐行地读取数据

默认情况下，指针指向第一条记录之前



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

示例：

**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

1. 关闭游标

**CLOSE [GLOBAL] < 游标名>|@< 游标变量>**

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

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

1. 释放游标

**DEALLOCATE [GLOBAL] <游标名>|@<游标变量>**

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

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

1. 存储过程
2. 概述

**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：

·运行效率高，提前进行语法分析及优化工作

·降低了客户机和服务器之间的通信量，用户只需要发送存储过程名字和参数

·方便实施企业规则，集中控制管理，方便维护

1. 创建存储过程

**CREATE { PROC | PROCEDURE }** [schema\_name.] procedure\_name

[ { @parameter data\_type } [ = default ] [ OUTPUT ] ] [ ,...n ]

AS

<sql\_statement>

过程名：数据库服务器合法的对象标识

参数列表：用名字来标识调用时给出的参数值，必须指定值的数据类型。参数也可以定义输入参数、输出参数或输入/输出参数，默认为输入参数

过程体：是一个<过程化SQL块>，包括声明部分和可执行语句部分

示例：从student表中查询学生信息

**create proc** myproc

**as**

select \* from student

1. 执行存储过程

**EXEC**|EXECUTE [ @return\_status = ] [schema\_name.]procedure\_name

[[@parameter =] {value | @variable [OUTPUT] | [ DEFAULT ]}][ ,...n ]

示例：从student表中查询学生信息

**exec** myproc

1. 更改存储过程

**ALTER { PROC | PROCEDURE }** [schema\_name.] procedure\_name

[ { @parameter data\_type } [ = default ] [ OUTPUT ] ] [ ,...n ]

AS

<sql\_statement>

示例1：修改存储过程myproc的功能为查找年龄小于19岁的学生信息

alter proc myproc

as

select \* from student where sage < 19;

示例2：修改存储过程myproc的功能为查找年龄小于某岁的学生信息**（带输入参数）**

**alter proc** myproc

**@sage int**

As

select **\*** from student where sage < @sage;

--执行存储过程

**exec myproc 19**

示例3：查找CS系中学生的平均年龄，并返回**（带输出参数）**

alter proc myproc

**@savg float out**

as

select **AVG(sage)** from student where sdept = 'CS'

--执行存储过程

declare @myvar float

**exec** myproc **@myvar out**

select @myvar

示例4：将指定记录插入student表

create proc pis

@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

--执行存储过程

exec pis '200901031','张三','男',18,'软件学院'

1. 删除存储过程

**drop procedure** {procedure\_name}[,…n]

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

1. 自定义函数
2. 概述

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

**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**

示例：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

执行：

select \*

from **DateOnly**(2020-1-1)

1. 内嵌表值函数

CREATE FUNCTION [ schema\_name. ] function\_name

([{ @parameter\_name data\_type [ = default ] } [ ,...n ]])

RETURNS TABLE

[ WITH <function\_option> [ ,...n ] ]

[ AS ]

RETURN ( select\_stmt )

示例1：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

示例2：查看计算机系学生的成绩

CREATE FUNCTION [dbo].[attendance] ( @sdept varchar(20) )

RETURNS TABLE

AS

RETURN

(

SELECT student.sno,cno,grade

from sc,student

where student.sno = sc.sno and sdept = @sdept

)

1. 多语句表值函数

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

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

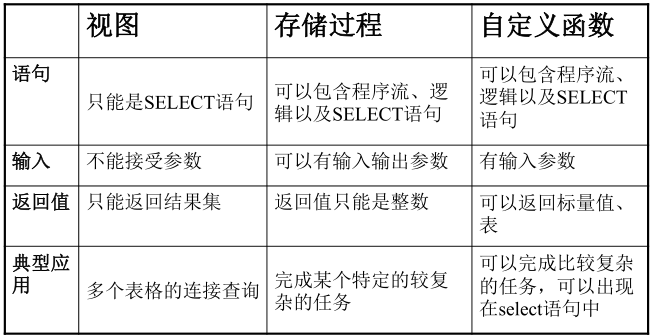
**CREATE FUNCTION** fn\_bumen

(@bm char(2))

**returns table**

as

return (select \* from test where 部门=@bm)



1. 触发器
2. 概述

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

1. 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。

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

1. 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

1. 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

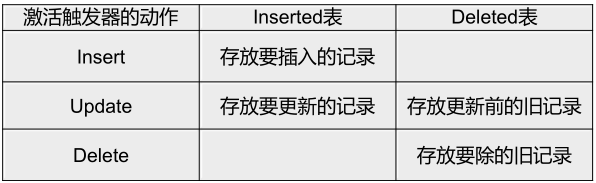
**不能直接调用**

是一个事务（可回滚）

分类：DELETE 触发器、INSERT 触发器、UPDATE 触发器

1. 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**



·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

1. 创建DML触发器

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

1. 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

示例1：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](mailto:student.sno=@bh))

Delete from sc where sno=@bh

END

示例2：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)

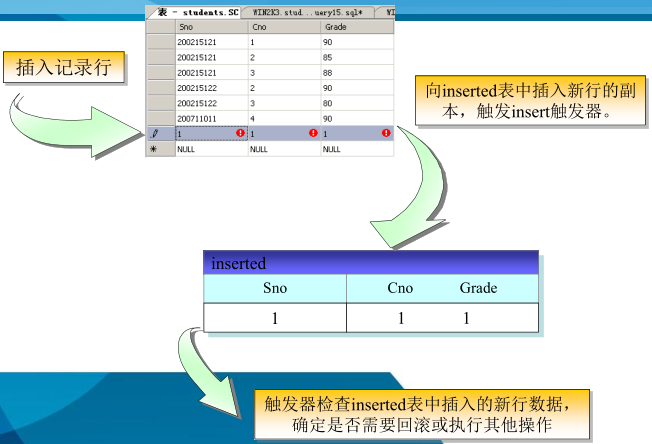
BEGIN

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

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

1. 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

示例：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)

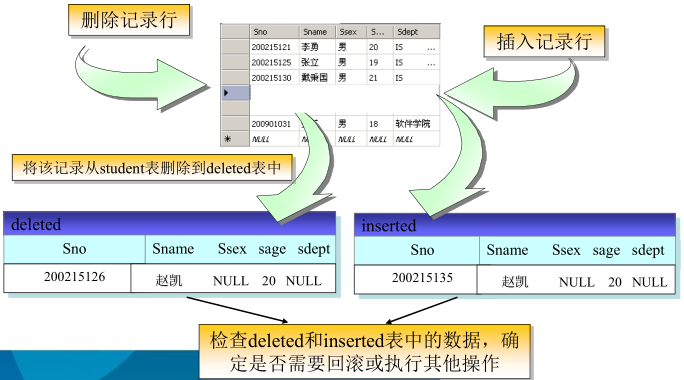
begin

raiserror('不能修改学号',16,10)

rollback transaction

end

END



1. 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

示例：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

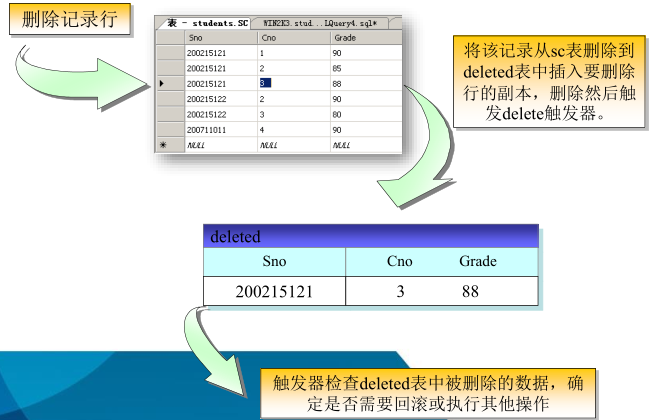
AS

BEGIN

SET NOCOUNT ON;

Insert into s1 select \* from deleted

END



1. INSTEAD OF触发器

示例：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