数据库原理课程设计[实验五]

一、实验准备

本例所用数据库，我们命名为EDUA(教学活动数据库)，包括student、course和study三个基本表，表段结构及数据如下所示:

1.结构与数据

1.1 学生表(student)

(1)学生表的结构

列名	数据类型	长度	是否允许为空值	字段说明
sno	char	5	NO	学号
sname	char	8	NO	姓名
age	smallint			年龄
sex	nchar	1		性别

说明：sno为主键，age的范围为15~35之间，sex只能为“男”或“女”。

(2)学生表的记录

sno	sname	age	sex
98601	李强	20	男
98602	刘丽	21	女

sno	sname	age	sex
98603	张兵	20	男
98604	陈志坚	22	男
98605	王颖	21	女

1.2 课程表(course)

(1)课程表的结构

列名	数据类型	长度	是否允许为空值	说明
cno	char	4	NO	课程号
cname	char	20	NO	课程名
teacher	char	8		任课教师

说明：cno为主键。

(2)课程表的记录

cno	cname	teacher
C601	高等数学	周振兴
C602	数据结构	刘建平
C603	操作系统	刘建平
C604	编译原理	王志伟

1.3 选课表(study)

(1)选课表的结构

列名	数据类型	长度	是否允许为空值	说明
Sno	char	5	NO	学号
Cno	char	4	NO	课程号
Score	smallint			成绩

说明：sno和cno为主键，sno为外键（参照student表的sno），cno为外键（参照course表的cno），score的范围为0~100之间。

(2)选课表的记录

Sno	cno	score
98601	C601	90
98601	C602	90
98601	C603	85
98601	C604	87
98602	C601	90
98603	C601	75
98603	C602	70
98603	C604	56
98604	C601	90
98604	C604	85
98605	C601	95
98605	C603	80

2.建库建表语句

```
--create database EDUA; --建立数据库
use EDUA                --选择数据库
GO                      --批处理脚本开始
create table student(                --学生表
sno char(5) PRIMARY KEY NOT NULL,    --学号 主键
sname char(8) NOT NULL,              --姓名 非空
age smallint check(age between 15 and 35),  --年龄 具有check约束
sex nchar(1) check(sex='男' or sex='女')  --性别 具有check约束
);

create table course(                 --课程表
cno char(4)  PRIMARY KEY NOT NULL, --课程号 主键
cname char(20)  NOT NULL,           --课程名 非空
teacher char(8)                      --任课教师
);

create table study(                  --选课表
```

```
sno char(5) NOT NULL,                    --学号(外键)
cno char(4) NOT NULL,                    --课程号(外键)
score smallint check(score between 0 and 100),    --成绩 具有check约束
PRIMARY KEY(sno,cno),                    --sno与cno做主键
foreign key(sno) references student(sno),   -- 学号 参照学生表的学号
foreign key(cno) references course(cno)     --课程号 参照为课程表的课程号
);
GO                        --批处理脚本结束
```

3.数据录入与结果展示

本次数据量不大 我们采用sql语句批量生成的方式完成数据录入。

```
use EDUA                --选择数据库
GO                      --批处理脚本开始
INSERT INTO Student VALUES('98601','李强',20,'男');
INSERT INTO Student VALUES('98602','刘丽',21,'女');
INSERT INTO Student VALUES('98603','张兵',20,'男');
INSERT INTO Student VALUES('98604','陈志坚',22,'男');
INSERT INTO Student VALUES('98605','王颖',21,'女');

INSERT INTO Course VALUES('C601','高等数学','周振兴');
INSERT INTO Course VALUES('C602','数据结构','刘建平');
INSERT INTO Course VALUES('C603','操作系统','刘建平');
INSERT INTO Course VALUES('C604','编译原理','王志伟');

INSERT INTO Study VALUES('98601','C601',90);
INSERT INTO Study VALUES('98601','C602',90);
INSERT INTO Study VALUES('98601','C603',85);
INSERT INTO Study VALUES('98601','C604',87);
INSERT INTO Study VALUES('98602','C601',90);
INSERT INTO Study VALUES('98603','C601',75);
INSERT INTO Study VALUES('98603','C602',70);
INSERT INTO Study VALUES('98603','C604',56);
INSERT INTO Study VALUES('98604','C601',90);
INSERT INTO Study VALUES('98604','C604',85);
INSERT INTO Study VALUES('98605','C601',95);
```

```
INSERT INTO Study VALUES('98605','C603',80);
GO                          --批处理脚本结束
```

最终结果如下图所示：

	sno	sname	age	sex
1	98601	李强	20	男
2	98602	刘丽	21	女
3	98603	张兵	20	男
4	98604	陈志坚	22	男
5	98605	王颖	21	女

	cno	cname	teacher
1	C601	高等数学	周振兴
2	C602	数据结构	刘建平
3	C603	操作系统	刘建平
4	C604	编译原理	王志伟

	sno	cno	score
1	98601	C601	90
2	98601	C602	90
3	98601	C603	85
4	98601	C604	87
5	98602	C601	90
6	98603	C601	75
7	98603	C602	70
8	98603	C604	56
9	98604	C601	90
10	98604	C604	85
11	98605	C601	95
12	98605	C603	80

二、实验内容

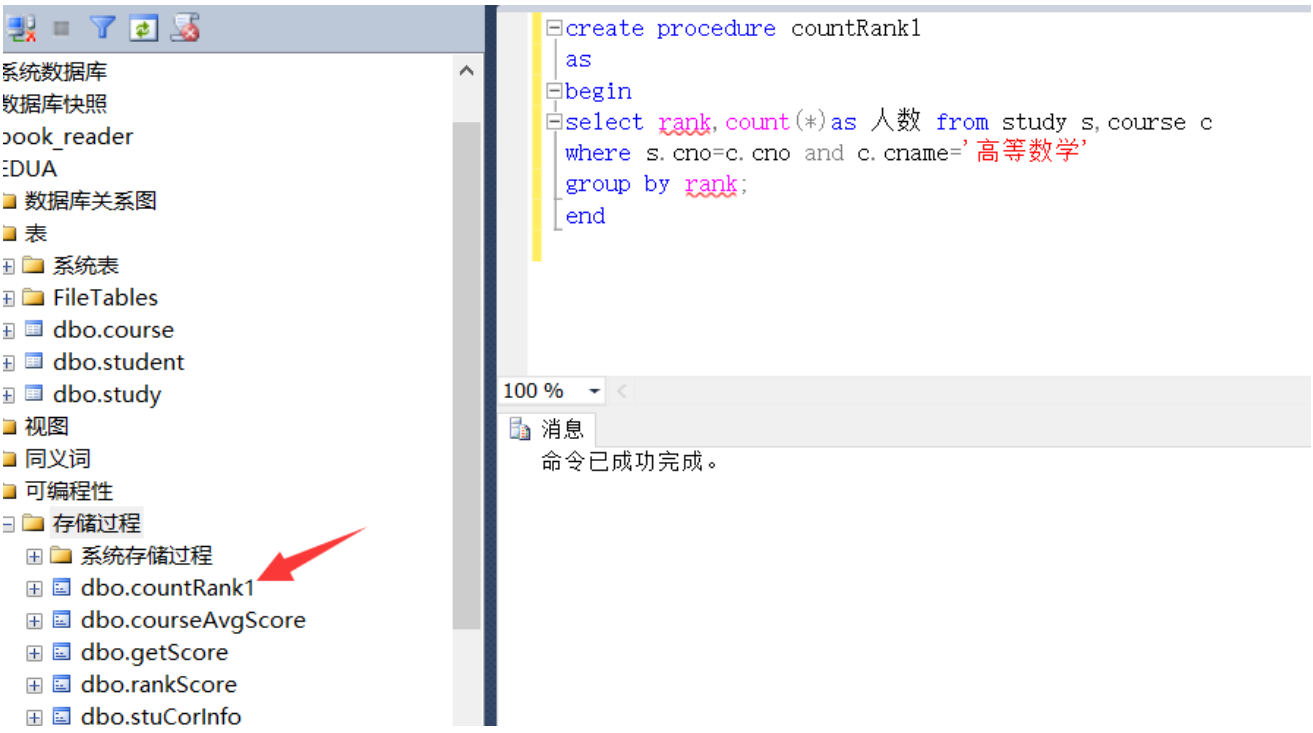
1.存储过程 [5项]

> *存储过程(stored procedure)是一组为了完成特定功能的SQL语句集合，经编译后存储在服务器端的数据库中，利用存储过程可以加速SQL语句的执行。具有如下优点:*
>
> - *提高应用程序的通用性和可移植性：存储过程创建后，可以在程序中被多次调用，而不必重新编写该存储过程的SQL语句。并且数据库专业人员可以随时对存储过程进行*

> - *修改，且对程序源代码没有影响，这样就极大的提高了程序的可移植性。*
> - *可以提高SQL的速度，存储过程是编译过的，如果某一个操作包含大量的SQL代码或分别被执行多次，那么使用存储过程比直接使用单条SQL语句执行速度快的多。*
> - *减轻服务器的负担：当用户的操作是针对数据库对象的操作时，如果使用单条调用的方式，那么网络上还必须传输大量的SQL语句，如果使用存储过程，则直接发送过程的调用命令即可，降低了网络的负担。*
>
> *以上总结自Internet。*

（1）创建一个存储过程，该存储过程统计“高等数学”的成绩分布情况，即按照各分数段统计人数。

```
--方法一
--该项任务 同第三题的要求有关联
--因此，我们先完成第三题中的等级分级后，再基于第三题的成果来完成该题目
create procedure countRank1
as
begin
select rank,count(*)as 人数 from study s,course c
where s.cno=c.cno and c.cname='高等数学'
group by rank;
end
```




```
--执行查看结果：
execute countRank1;
```

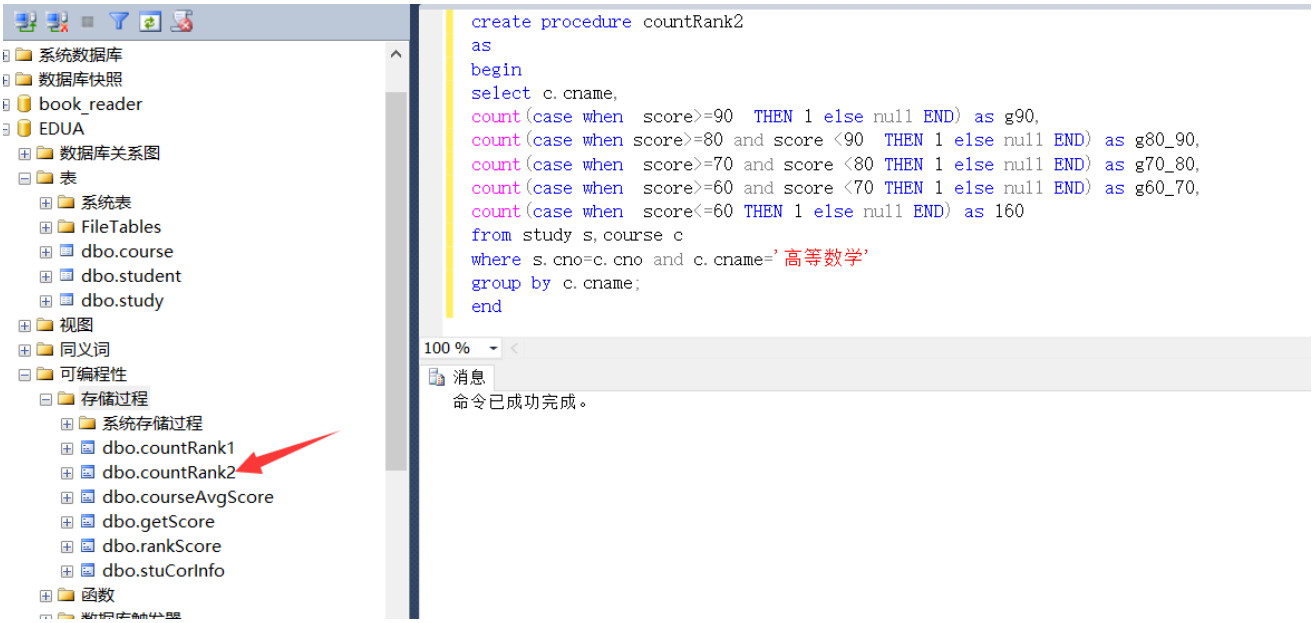
```
execute countRank1;
```

100 %

结果 消息

	rank	人数
1	A	4
2	C	1

```
--方法二
--抛开第三题 利用case when... then .. else .. end语句 配合count来完成统计
--将统计结果放在一条语句中呈现出来
create procedure countRank2
as
begin
select c.cname,
count(case when  score>=90  THEN 1 else null END) as g90,
count(case when score>=80 and score <90  THEN 1 else null END) as g80_90,
count(case when  score>=70 and score <80 THEN 1 else null END) as g70_80,
count(case when  score>=60 and score <70 THEN 1 else null END) as g60_70,
count(case when  score<=60 THEN 1 else null END) as l60
from study s,course c
where s.cno=c.cno and c.cname='高等数学'
group by c.cname;
end
```




```
execute countRank2;
```

```
execute countRank2;
```

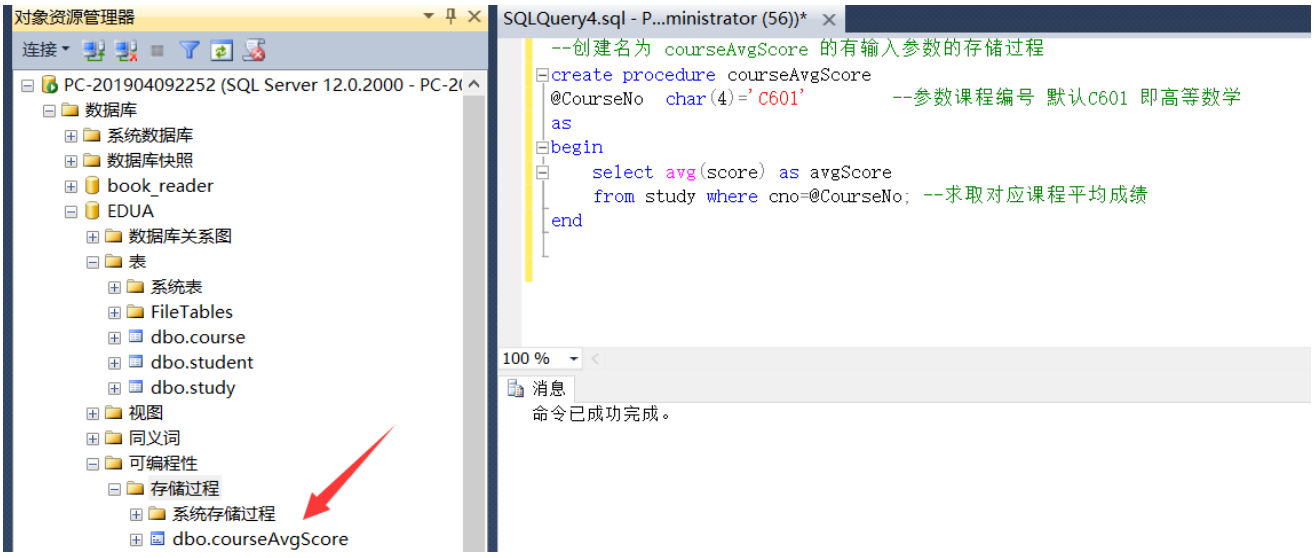
100 %

结果 消息

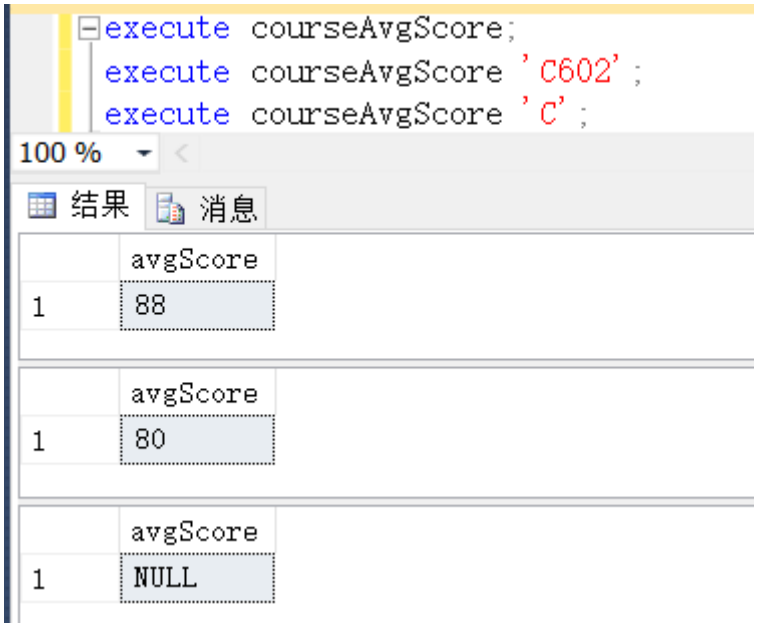
	cname	g90	g80_90	g70_80	g60_70	l60
1	高等数学	4	0	1	0	0

（2）创建一个存储过程，该存储过程有一个参数用来接收课程号，该存储过程统计给定课程的平均成绩。

```
--创建名为 courseAvgScore 的有输入参数的存储过程
create procedure courseAvgScore
@CourseNo  char(4)='C601'          --参数课程编号 默认C601 即高等数学
as
begin
    select avg(score) as avgScore
    from study where cno=@CourseNo; --求取对应课程平均成绩
end
```

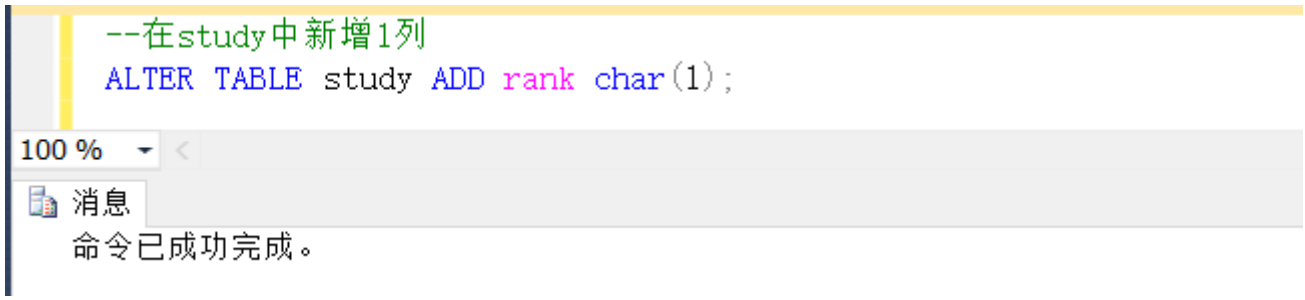



```
execute courseAvgScore;          --采用参数默认值 统计高等数学平均成绩
execute courseAvgScore 'C602';   --给定参数 C602 统计数据结构平均成绩
execute courseAvgScore 'C';      --给定一个非法参数 预期应出现NULL
```



（3）创建一个存储过程，该存储过程将学生选课成绩从百分制改为等级制（即 A、B、C、D、E）。

```
--思路：我们采取在study表中增设一列来表示等级制的成绩
--       其中，A：[90,100]  B：[80,90) C：[70,80)
--             D：[60,70)   E：[0,60)
--在study中新增1列
ALTER TABLE study ADD rank char(1);
--创建名为rankScore的无参存储过程
create procedure rankScore
as
begin
update study set rank='A' where score>=90;
update study set rank='B' where score>=80 and score<90;
update study set rank='C' where score>=70 and score<80;
update study set rank='D' where score>=60 and score<70;
update study set rank='E' where score<=60;
end
```



```
--创建名为rankScore的无参存储过程
create procedure rankScore
as
begin
update study set rank='A' where score>=90;
update study set rank='B' where score>=80 and score<90;
update study set rank='C' where score>=70 and score<80;
update study set rank='D' where score>=60 and score<70;
update study set rank='E' where score<=60;
end
```

100 %

消息

命令已成功完成。

```
--执行查看结果
execute rankScore;
select * from study;
```

```
execute rankScore;
select * from study;
```

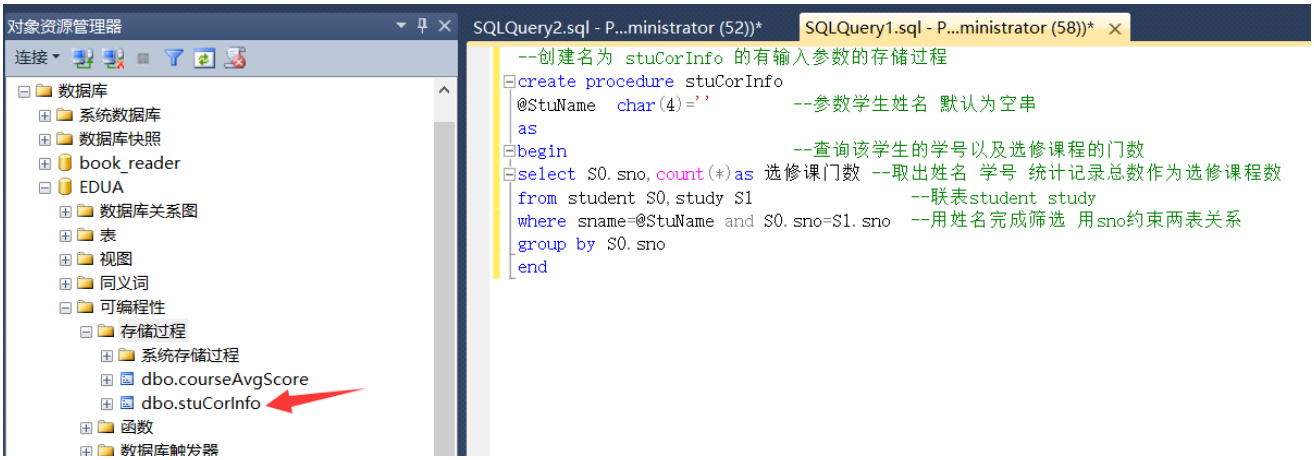
100 %

结果 消息

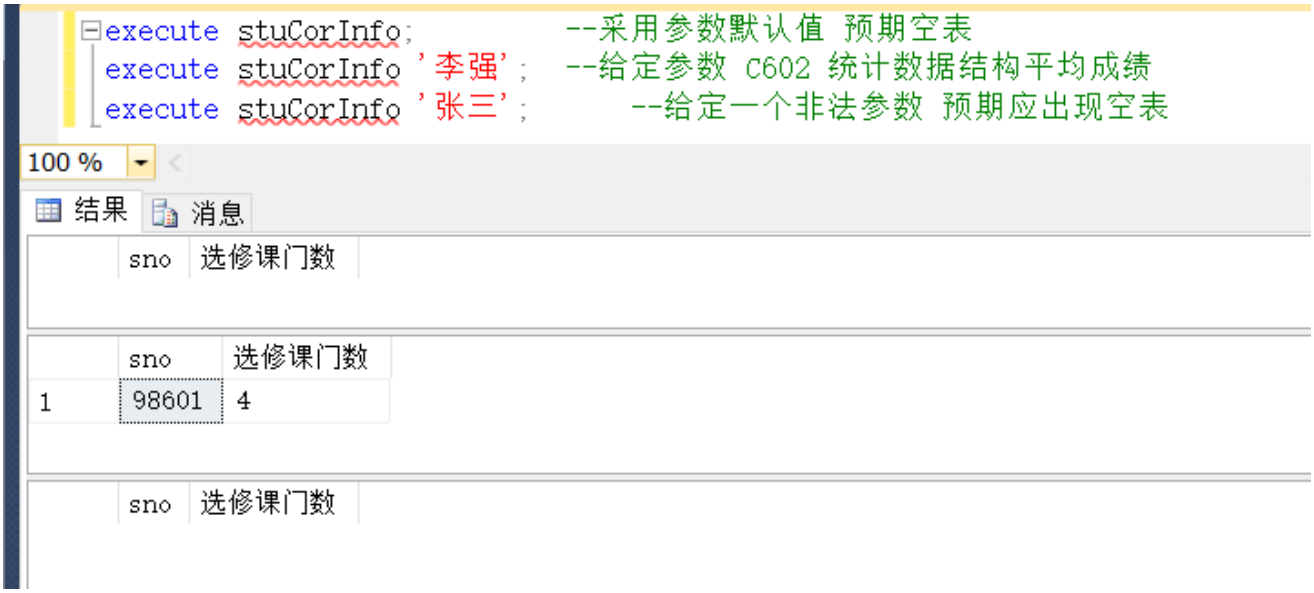
	sno	cno	score	rank
1	98601	C601	90	A
2	98601	C602	90	A
3	98601	C603	85	B
4	98601	C604	87	B
5	98602	C601	90	A
6	98603	C601	75	C
7	98603	C602	70	C
8	98603	C604	56	E
9	98604	C601	90	A
10	98604	C604	85	B
11	98605	C601	95	A
12	98605	C603	80	B

（4）创建一个存储过程，该存储过程有一个参数用来接收学生姓名，该存储过程查询该学生的学号以及选修课程的门数。

```
--创建名为 stuCorInfo 的有输入参数的存储过程
create procedure stuCorInfo
@StuName  char(4)=''          --参数学生姓名 默认为空串
as
begin                         --查询该学生的学号以及选修课程的门数
select S0.sno,count(*)as 选修课门数   --取出姓名 学号 统计记录总数作为选修课程数
from student S0,study S1                --联表student study
where sname=@StuName and S0.sno=S1.sno  --用姓名完成筛选 用sno约束两表关系
group by S0.sno
end
```

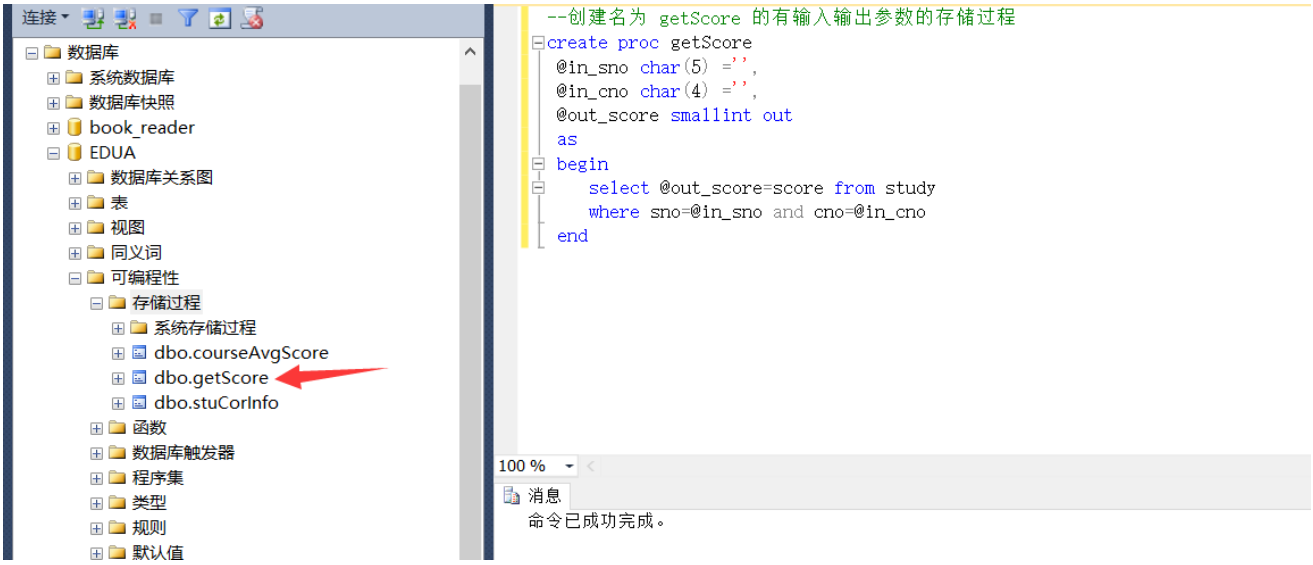



```
execute stuCorInfo;          --采用参数默认值 预期空表
execute stuCorInfo '李强';    --给定参数 C602 统计数据结构平均成绩
execute stuCorInfo '张三';       --给定一个非法参数 预期应出现空表
```



（5）创建一个存储过程，该存储过程有两个输入参数用来接收学号和课程号，一个输出参数用于获取相应学号和课程号对应的成绩。

```
--创建名为 getScore 的有输入输出参数的存储过程
create proc getScore
 @in_sno char(5) ='',                    --输入参数学号
 @in_cno char(4) ='',                    --输入参数课程号
 @out_score smallint out                 --输出参数成绩
 as
 begin
    select @out_score=score from study
    where sno=@in_sno and cno=@in_cno
 end
```




```
--声明三个变量 分别用来指定输入参数 以及保存输出参数
declare @mysno char(5),
        @mycno char(4),
        @myscore smallint;
--为输入参数赋值
set @mysno='98601';
set @mycno='C602';
--执行存储过程 并传入参数
exec getScore @mysno,@mycno,@myscore out;
--将结果打印出来 (由于smallint无法实现自动转换 故用强转换convert)
print '学号 ['+@mysno+'] 学生的 ['+@mycno+'] 课程'
+'成绩为:'+convert(varchar(10),@myscore);
```

```
declare @mysno char(5),
        @mycno char(4),
        @myscore smallint;
set @mysno='98601';
set @mycno='C602';
exec getScore @mysno,@mycno,@myscore out;
print '学号 ['+@mysno+'] 学生的 ['+@mycno+'] 课程'
+'成绩为:'+convert(varchar(10),@myscore);
```

100 %

消息

```
学号 [98601] 学生的 [C602] 课程成绩为:90
```

2.触发器[3项]

> *触发器（trigger）是SQL server 提供给程序员和数据分析员来保证数据完整性的一种方法，它是与表事件相关的特殊的存储过程，它的执行不是由程序调用，也不是手工启动，而是由事件来触发，当对一个表进行操作（insert，delete， update）时就会激活它执行。触发器经常用于加强数据的完整性约束和匹配业务规则等。 触发器可以从 DBA_TRIGGERS，USER_TRIGGERS 数据字典中查到。*
>
> ***触发器和存储过程最显然的区别:***
>
> *触发器与存储过程的区别是运行方式的不同，触发器不能执行EXECUTE语句调用，而是在用户执行Transact-SQL语句时自动触发执行而存储过程需要用户、应用程序或者触发器来显示地调用并执行。*
>
> ***触发器的优点***
>
> *1.触发器是自动的。当对表中的数据做了任何修改之后立即被激活。*
>
> *2.触发器可以通过数据库中的相关表进行层叠修改。*
>
> *3.触发器可以强制限制。这些限制比用CHECK约束所定义的更复杂。与CHECK约束不同的是，触发器可以引用其他表中的列。*
>
> *4.实现由主键和外键所不能保证的复杂参照完整性和数据的一致性，它能够对数据库中的相关表进行级联修改，提高比CHECK约束更复杂的的数据完整性，并支持自定义错误消息。*
>
> ***触发器的工作原理***
>
> *触发器触发时:*

1. *系统自动在内存中创建deleted表或inserted表；*
2. *只读，不允许修改，触发器执行完成后，自动删除。*

inserted表：

1. *临时保存了插入或更新后的记录行；*
2. *可以从inserted表中检查插入的数据是否满足业务需求；*
3. *如果不满足，则向用户发送报告错误消息，并回滚插入操作。*

deleted表：

1. *临时保存了删除或更新前的记录行；*
2. *可以从deleted表中检查被删除的数据是否满足业务需求；*
3. *如果不满足，则向用户报告错误消息，并回滚插入操作。*

inserted表和deleted表对照：

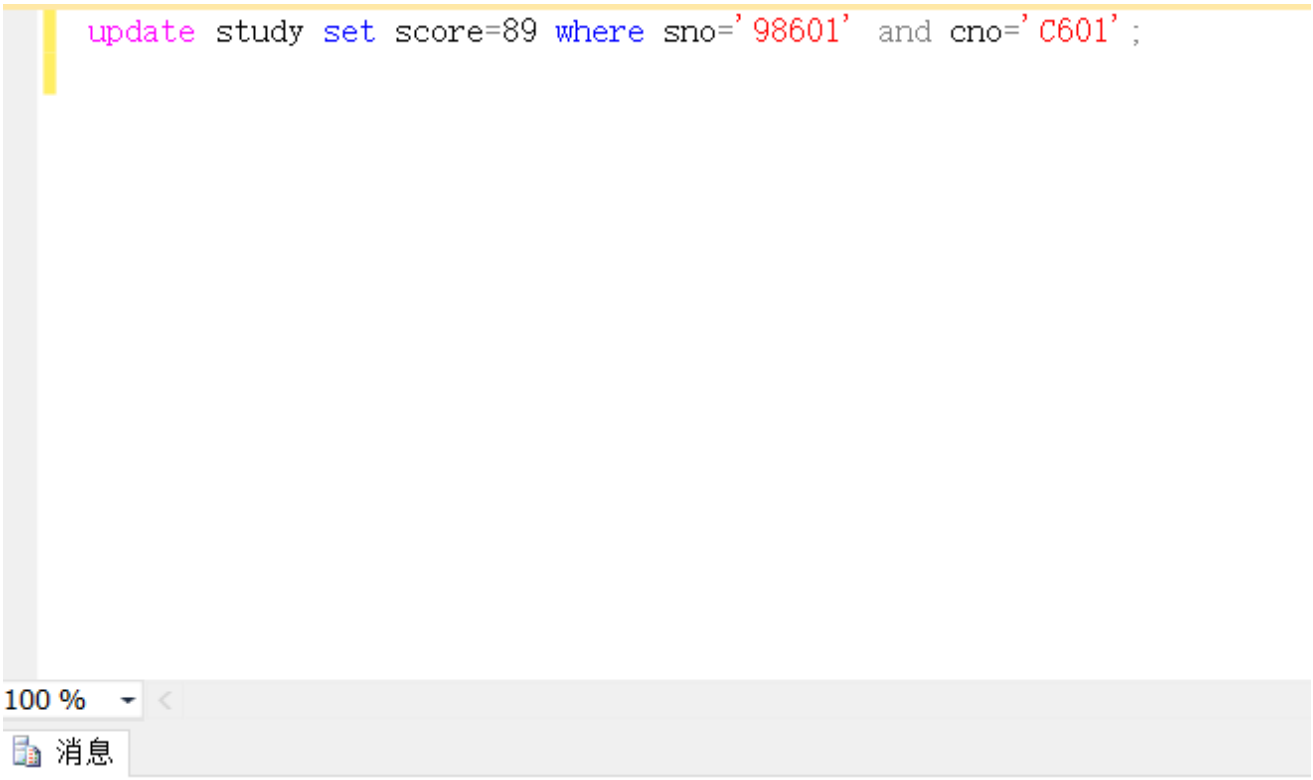
修改操作记录	***inserted表***	***deleted表***
增加(insert)记录	*存放新增的记录*	.....
删除(deleted)记录	.....	*存放被删除的记录*
修改(update)记录	*存放更新后的记录*	*存放更新前的记录*

注明:下述三道题目均为先验触发器(即instead of)在语句生效前判断。

（1）为study表创建一个UPDATE触发器，当更新成绩时，要求更新后的成绩不能低于原来的成绩。

```
--创建一个名为 UPDATE_ScoreMustLarger的触发器 思路如下:
create trigger UPDATE_ScoreMustLarger on study  --在study表上创建触发器
instead of update                               --指定是插入数据触发
as
begin
--定义临时变量 分别保存更新数据时提供的三个字段(sno cno score)以及更新前的score
declare @newSno char(5),@newCno char(4),@newScore smallint,@oldScore smallint
--从iserted表中取出临时保存的插入数据 取出其各个字段内容
select @newSno=sno,@newCno=cno,@newScore=score from inserted;
--从deleted表中定位将要被删除的旧记录 并取出其score值
select @oldScore=score from deleted;
if(@newScore>=@oldScore)                    --若更新的成绩不低于原成绩
update study set score=@newScore            --执行更新
where sno=@newSno and cno=@newCno;
else                                        --提示错误信息
print '成绩不能低于此前录入的值,您的更新语句并未被执行!';
end
```

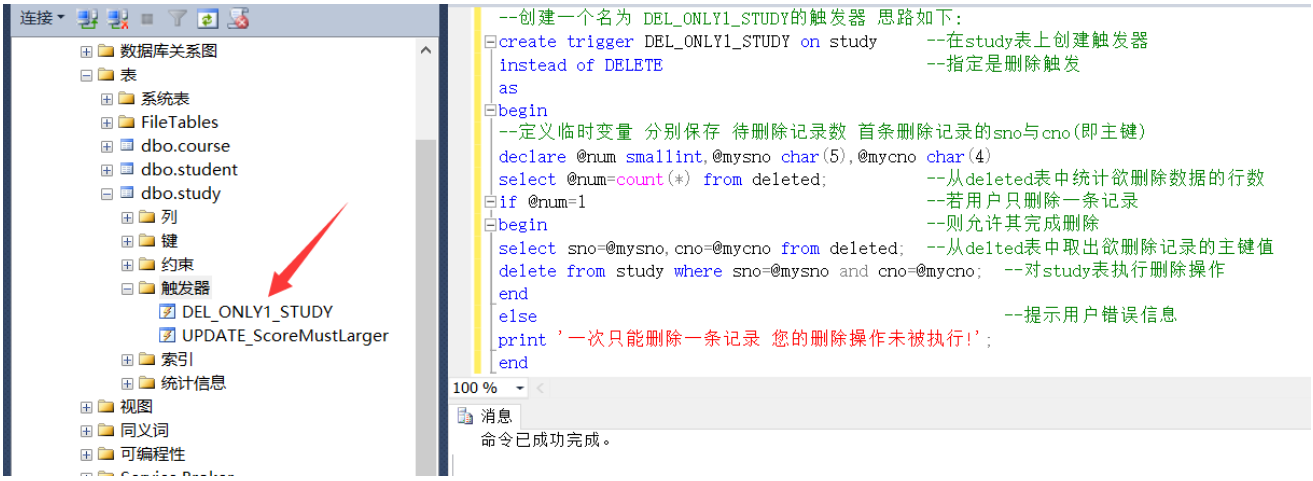

```
--触发触发器
---尝试更新一条记录使其score低于原记录的score
update study set score=89 where sno='98601' and cno='C601';
--可以看到 由于违反触发器规则 因此该语句无法顺利执行
```



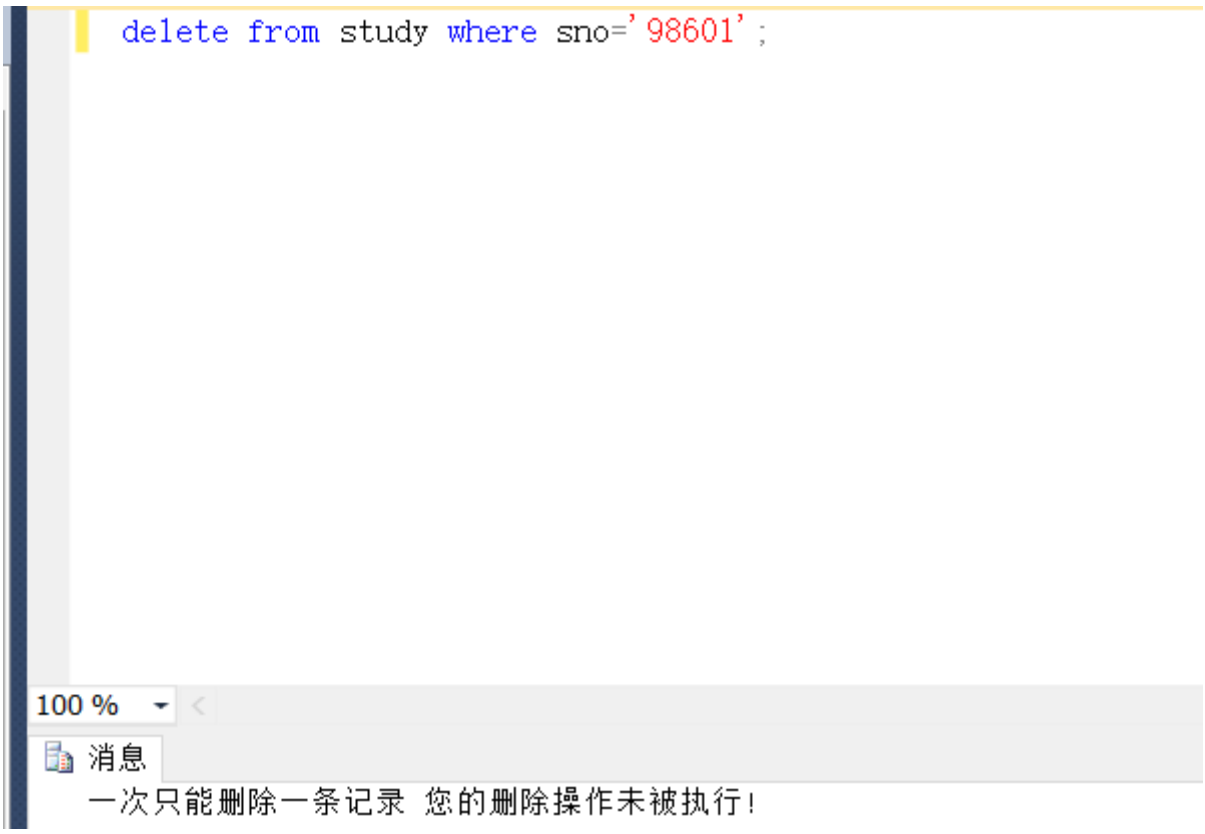
（2）为study表创建一个DELETE触发器，要求一次只能从study表中删除一条记录。

```
--创建一个名为 DEL_ONLY1_STUDY的触发器 思路如下:
create trigger DEL_ONLY1_STUDY on study      --在study表上创建触发器
instead of DELETE                            --指定是删除触发
as
begin
--定义临时变量 分别保存 待删除记录数 首条删除记录的sno与cno(即主键)
declare @num smallint,@mysno char(5),@mycno char(4)
select @num=count(*) from deleted;           --从deleted表中统计欲删除数据的行数
if @num=1                                    --若用户只删除一条记录
```

```
begin                                                    --则允许其完成删除
select sno=@mysno,cno=@mycno from deleted;   --从delted表中取出欲删除记录的主键值
delete from study where sno=@mysno and cno=@mycno;   --对study表执行删除操作
end
else                                                     --提示用户错误信息
print '一次只能删除一条记录 您的删除操作未被执行!';
end
```



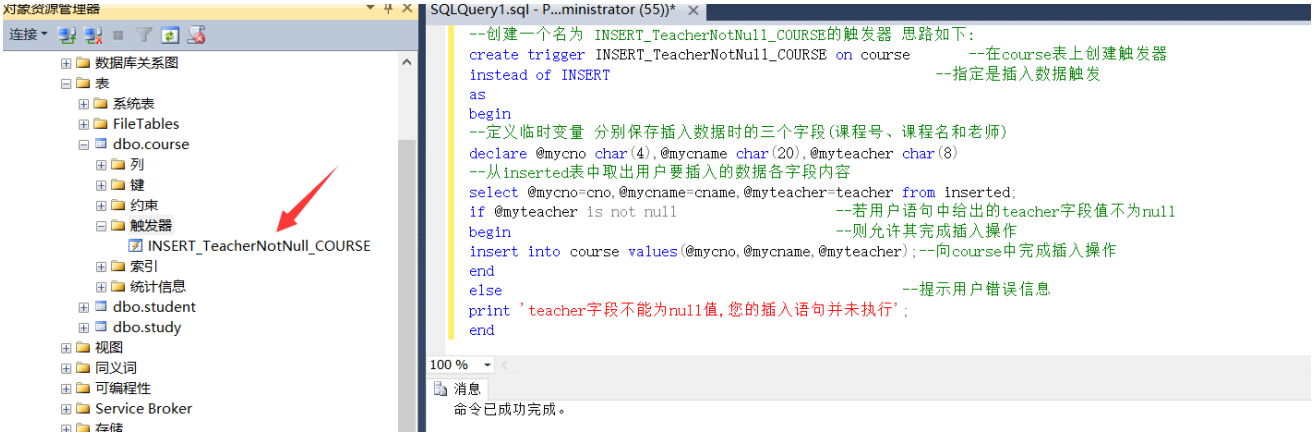

```
--触发触发器
---尝试删除多条记录
delete from study where sno='98601';
--可以看到 由于违反触发器规则 因此该语句无法顺利执行
```



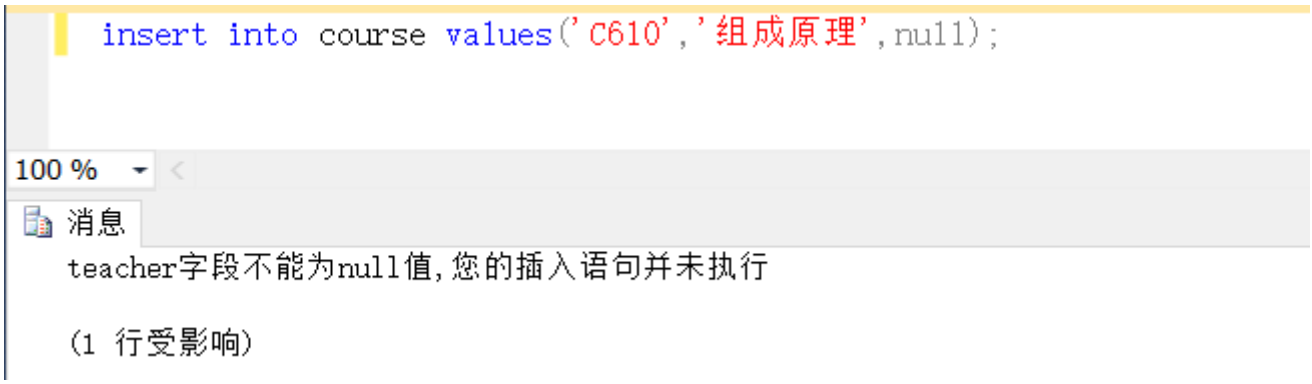
（3）为course表创建一个INSERT触发器，要求插入的课程记录中任课教师不能为空。

```
--创建一个名为 INSERT_TeacherNotNull_COURSE的触发器 思路如下:
create trigger INSERT_TeacherNotNull_COURSE on course        --在course表上创建
触发器
```

```
instead of INSERT                                          --指定是插入数据触发
as
begin
--定义临时变量 分别保存插入数据时的三个字段(课程号、课程名和老师)
declare @mycno char(4),@mycname char(20),@myteacher char(8)
--从inserted表中取出用户要插入的数据各字段内容
select @mycno=cno,@mycname=cname,@myteacher=teacher from inserted;
if @myteacher is not null                          --若用户语句中给出的teacher字段值不为
null
begin                                              --则允许其完成插入操作
insert into course values(@mycno,@mycname,@myteacher);--向course中完成插入操作
end
else                                                       --提示用户错误信息
print 'teacher字段不能为null值,您的插入语句并未执行';
end
```




```
--触发触发器
---尝试插入一条teacher为null的记录
insert into course values('C610','组成原理',null);
```



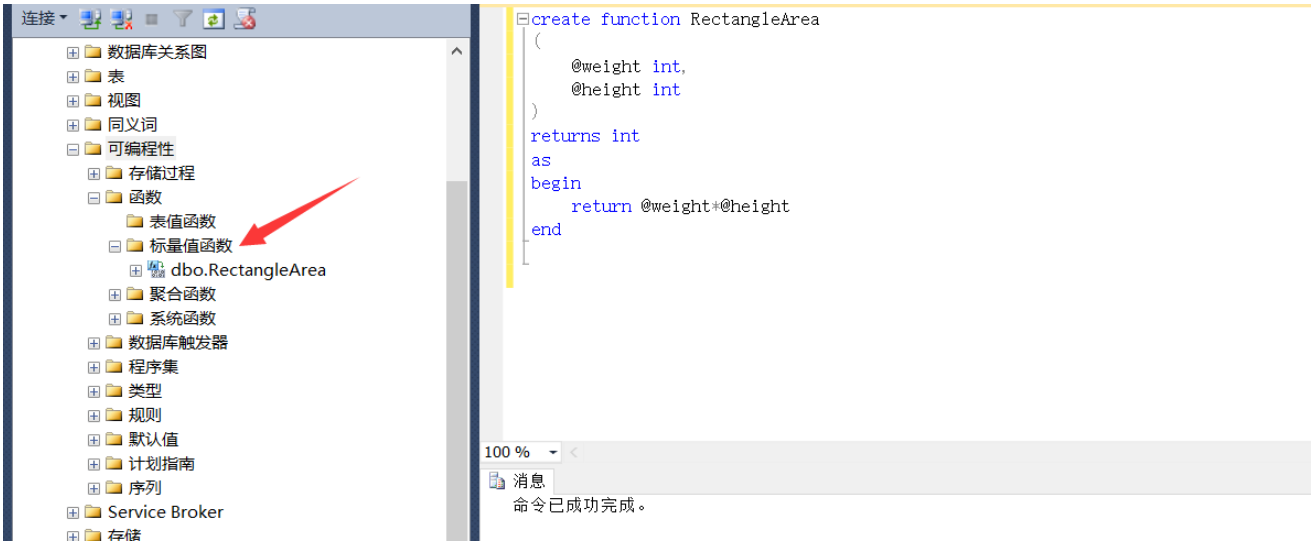
3.用户自定义函数[2项]

> *用户自定义函数不能用于执行一系列改变数据库状态的操作，但是它可以像系统函数一样在查询或存储过程等的程序段中使用，也可以像存储过程一样通过EXECUTE 命令来执行。*
>
> *自定义函数分为 标量函数、表值函数、多语句表值函数 三种。*
>
> ***标量函数：**对单一值操作，返回单一值。在begin...end块中定义函数主体。只要在能够使用表达式的地方，就可以使用标量函数。*

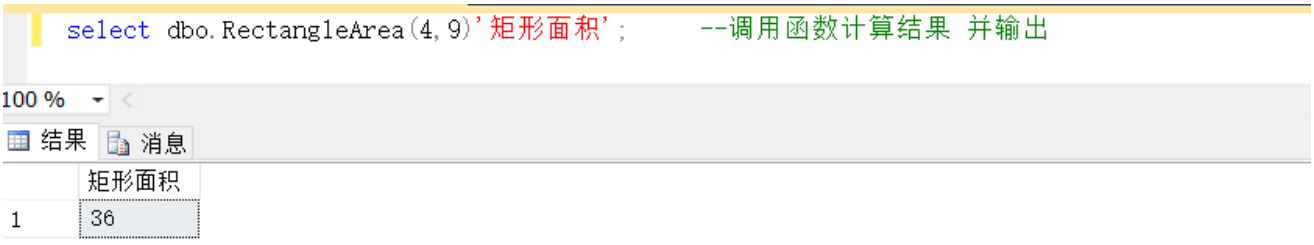
> *表值函数：返回值是一个记录集合——表。在此函数中，无begin...end块中定义函数主体，只有return 语句包含一条单独的select语句。*
>
> *多语句表值函数：返回值是一个记录集合——表。返回值是由选择的结果构成的记录集。*

（1）创建一个返回标量值的用户定义函数 RectangleArea：输入矩形的长和宽就能计算矩形的面积。

```
create function RectangleArea        --定义函数
(
    @weight int,                     --矩形的宽
    @height int                      --矩形的长
)
returns int                          --返回的标量值类型
as
begin
    return @weight*@height           --函数内部计算过程
end
```

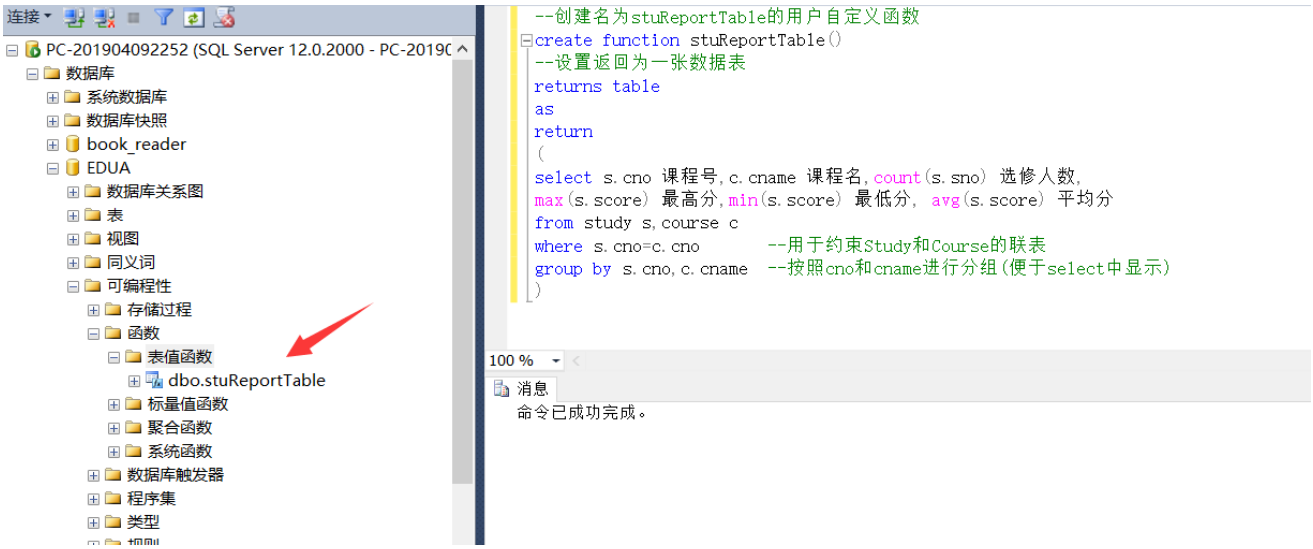



```
select dbo.RectangleArea(4,9)'矩形面积';      --调用函数计算结果 并输出
```

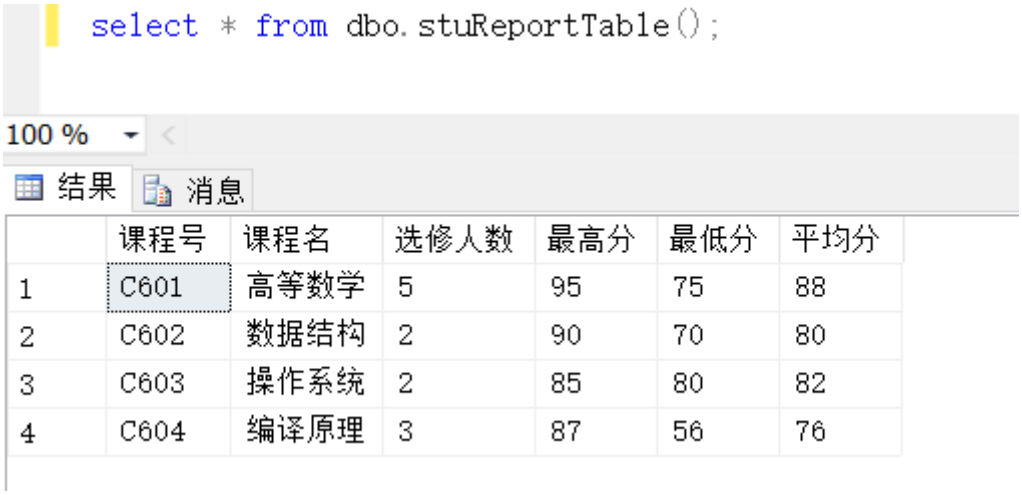


（2）创建一个用户自定义函数，功能为产生一张有关学生成绩统计的报表。该报表显示每一门课程的课程号、课程名、选修人数、本门最高分、最低分和平均分。调用这个函数，生成相应的报表并给用户浏览。

```
--创建名为stuReportTable的用户自定义函数
create function stuReportTable()
--设置返回为一张数据表
returns table
as
return
(
select s.cno 课程号,c.cname 课程名,count(s.sno) 选修人数,
max(s.score) 最高分,min(s.score) 最低分, avg(s.score) 平均分
from study s,course c
where s.cno=c.cno        --用于约束Study和Course的联表
group by s.cno,c.cname   --按照cno和cname进行分组(便于select中显示)
)
```




```
--执行函数
select * from dbo.stuReportTable();
```



	课程号	课程名	选修人数	最高分	最低分	平均分
1	C601	高等数学	5	95	75	88
2	C602	数据结构	2	90	70	80
3	C603	操作系统	2	85	80	82
4	C604	编译原理	3	87	56	76