

经典 SQL. 2000 教程

实验三 数据库查询和 SQL Server 编程

3.1 数据库查询

数据库是为更方便有效地管理信息而存在的，人们希望数据库可以随时提供所需要的数据信息。因此，对用户来说，数据查询是数据库最重要的功能。本实验将学习数据查询的各种方法。

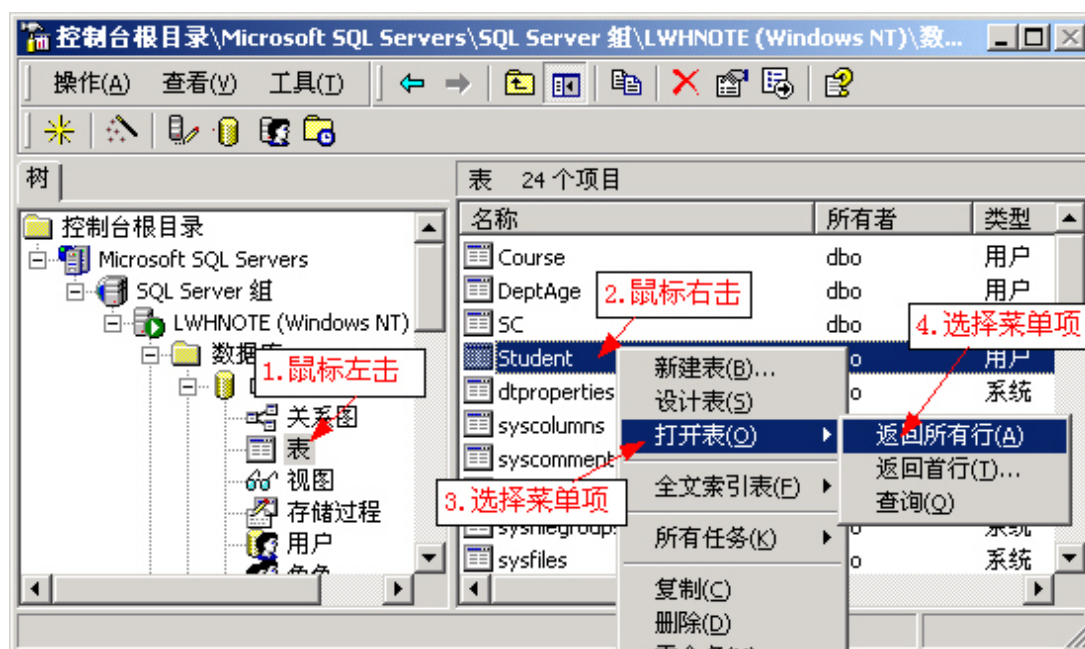
3.1.1 查询工具的使用

对数据表中的数据进行查询，既可以使用企业管理器，也可以使用查询分析器。对于简单查询，更适合使用企业管理器进行查询；对复杂查询，更适合使用查询分析器进行查询。

一、使用企业管理器进行查询

A. 查询数据表的全部值

如：显示学生数据表 Student 的全部数据。



显示的数据表如下

2:表"Student"中的数据，位置是"DB5"中、"LWHNOTE"上

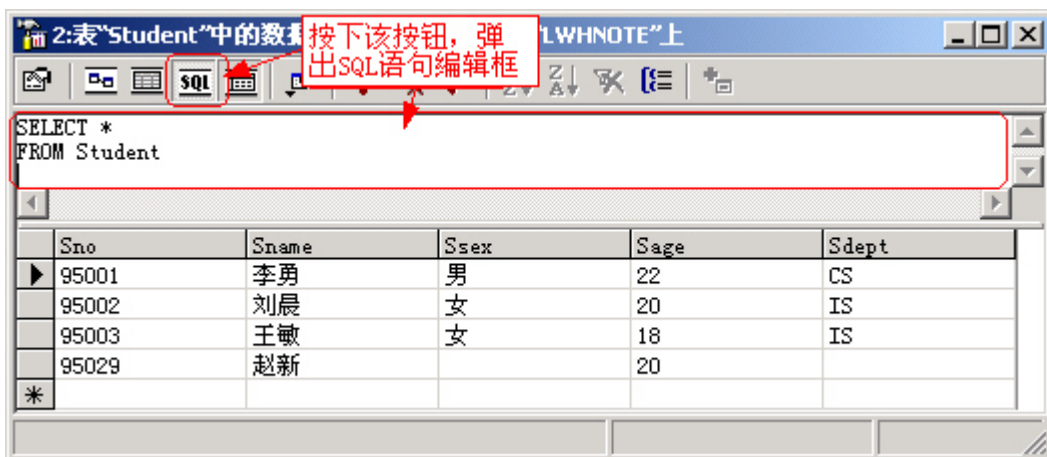
	Sno	Sname	Ssex	Sage	Sdept
▶	95001	李勇	男	22	CS
	95002	刘晨	女	20	IS
	95003	王敏	女	18	IS
	95029	赵新		20	
*					

B. 用 Select 语句查询数据表的值

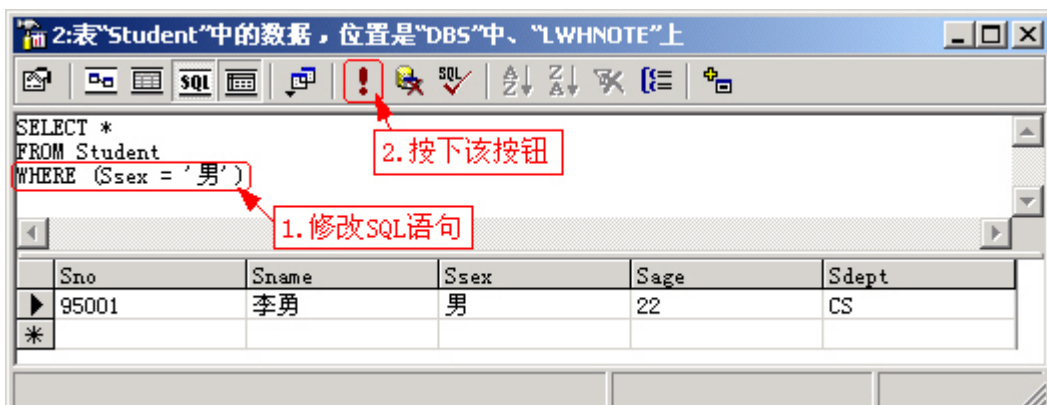
如：显示学生数据表 Student 中男生的数据。

同<A. 查询数据表的全部值>显示学生数据表 Student 的全部数据，见上表。

然后，如下图所示弹出 SQL 语句编辑框



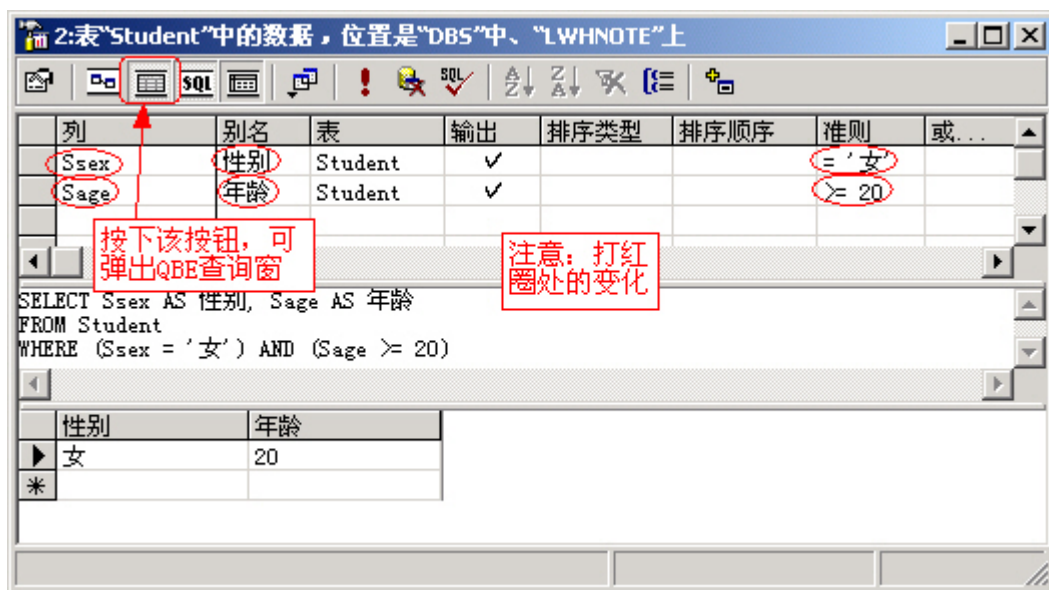
如下图所示修改 Select 语句，然后运行修改后的 SQL 语句，即可显示出满足条件的查询结果



C. 学习和使用 QBE 查询

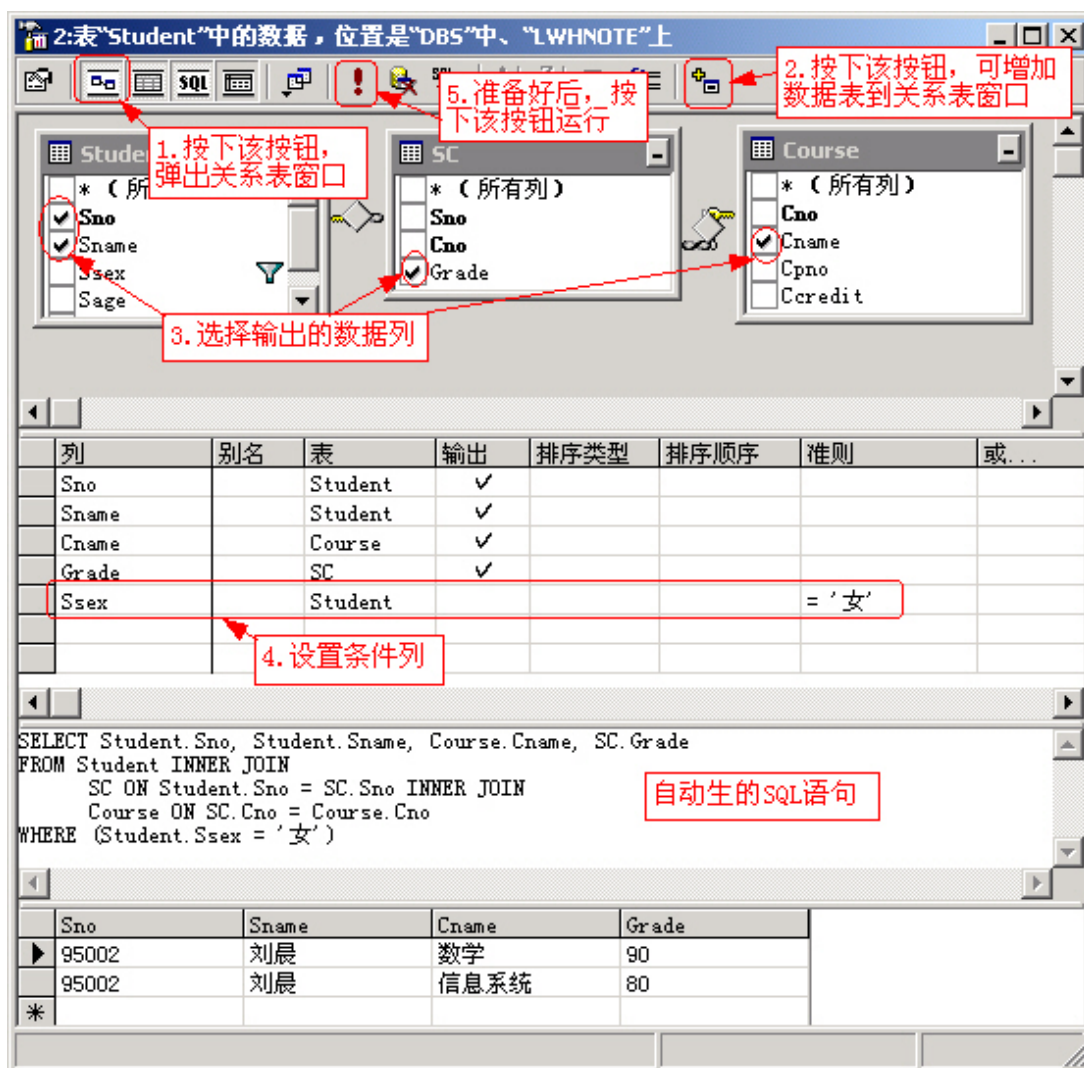
如：显示学生数据表 Student 中大于等于 20 岁的女生的数据。

在上图的结果中，按下图所示即可显示 QBE 查询窗口，然后修改查询条件

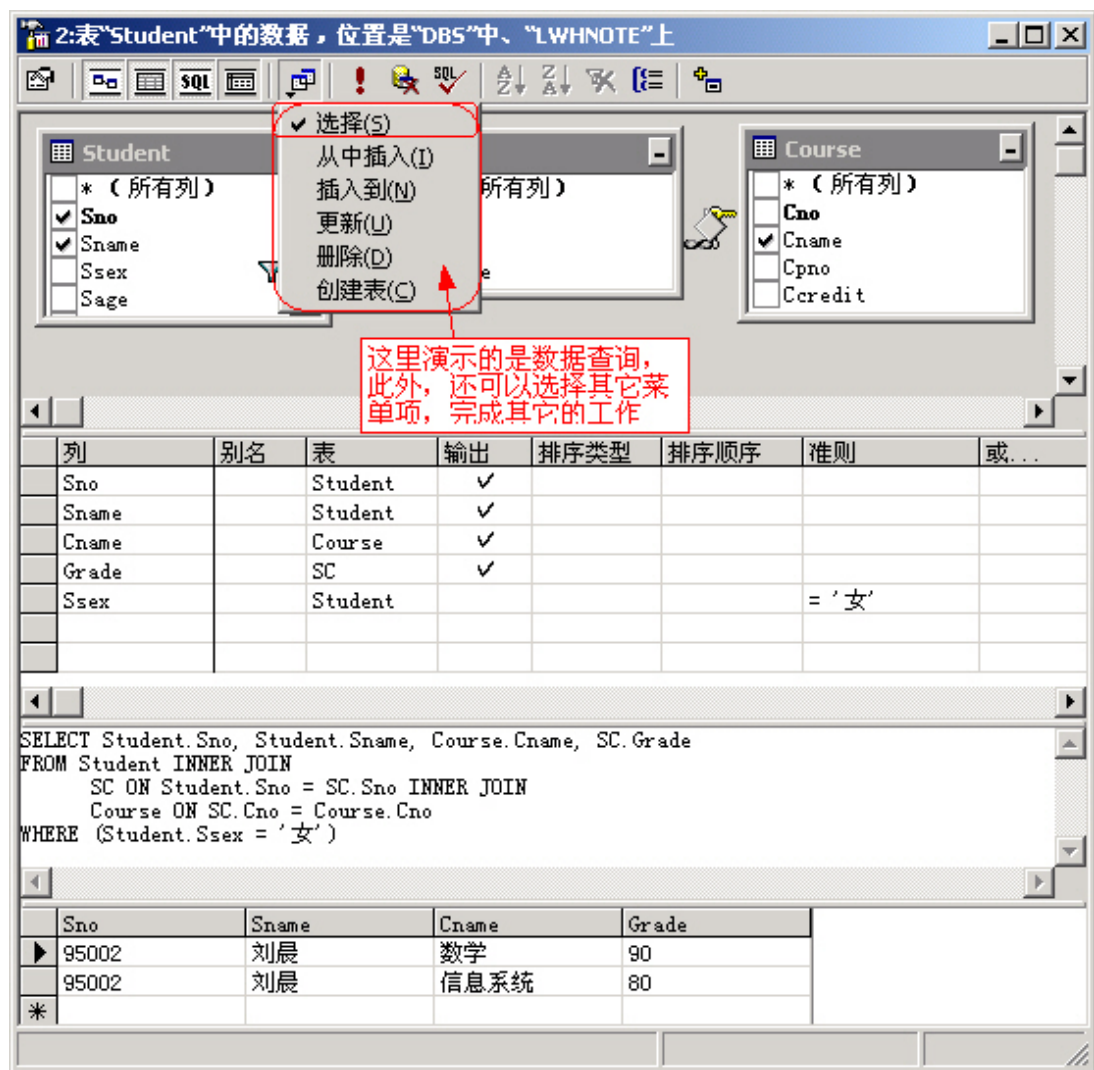


D. 实现多表查询

如：查询所有女生的学生成绩，结果中包括：学号、姓名、课程号和成绩。



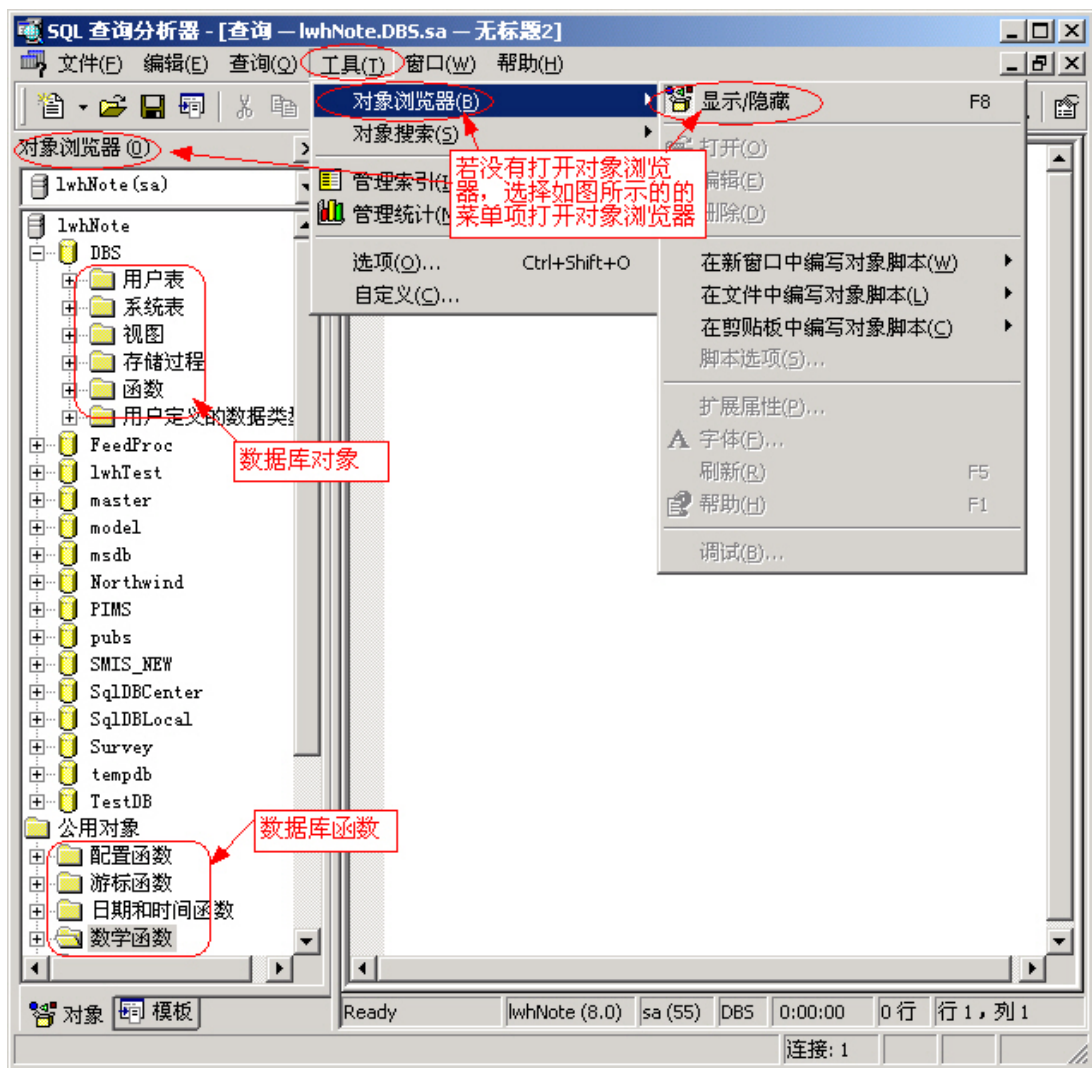
E. 在企业管理器的查询窗口中还可以做其它的工作



二、使用查询分析器进行查询

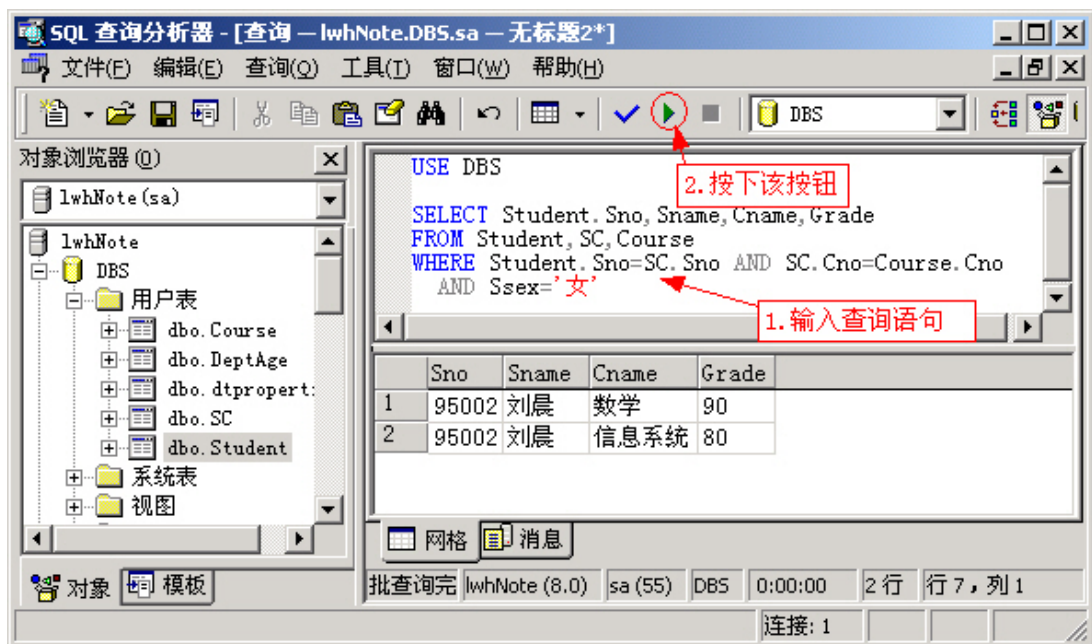
A. 设置查询分析器

利用查询分析器进行查询时，可能要用到数据库中的各种对象，打开对象浏览器，便于查询数据库的各种对象。



B. 利用查询分析器进行查询

如：如：查询所有女生的学生成绩，结果中包括：学号、姓名、课程号和成绩。



注：在以下讲解各种查询时，以查询分析为主。

3.1.2 简单查询

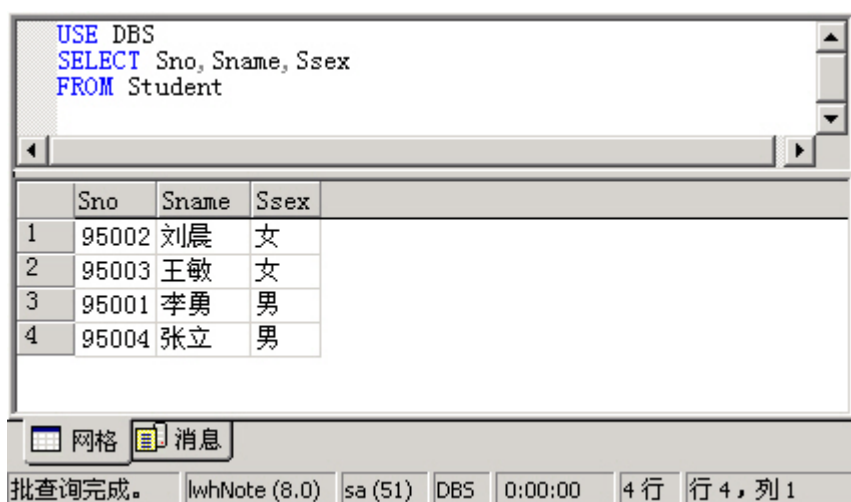
 帮助目录：<访问和更改关系数据|查询基础知识>

 帮助索引：<查询-SQL Server>

一、选择列

A. 用 **SELECT** 子句来指定查询所需的列，多个列之间用 “,” 分开

如：查询 **Student** 中所有学生的学号、姓名、性别。



The screenshot shows a SQL query window with the following text:

```
USE DBS
SELECT Sno, Sname, Ssex
FROM Student
```

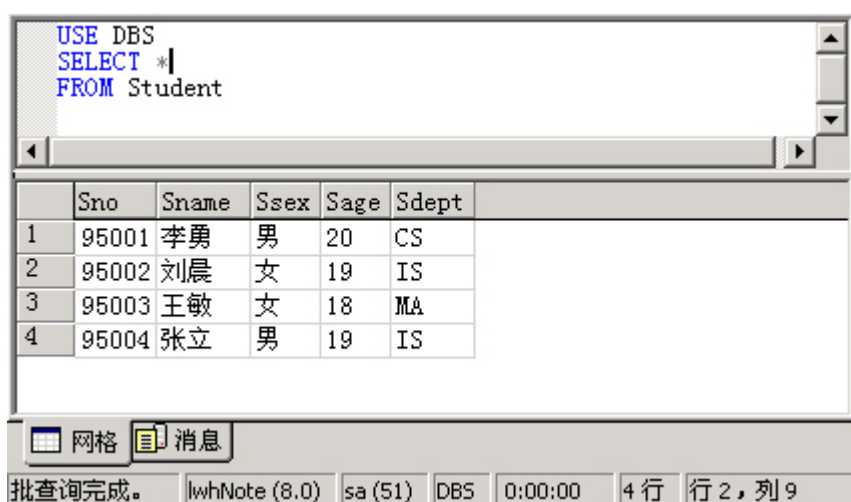
Below the query window, the results are displayed in a table with 4 rows and 3 columns (Sno, Sname, Ssex).

	Sno	Sname	Ssex
1	95002	刘晨	女
2	95003	王敏	女
3	95001	李勇	男
4	95004	张立	男

The status bar at the bottom indicates: 批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 4 行 | 行 4, 列 1

B. 可以用 “*” 来选取数据表的全部列

如：查询 **Student** 中的有学生的基本情况



The screenshot shows a SQL query window with the following text:

```
USE DBS
SELECT *
FROM Student
```

Below the query window, the results are displayed in a table with 4 rows and 6 columns (Sno, Sname, Ssex, Sage, Sdept).

	Sno	Sname	Ssex	Sage	Sdept
1	95001	李勇	男	20	CS
2	95002	刘晨	女	19	IS
3	95003	王敏	女	18	MA
4	95004	张立	男	19	IS

The status bar at the bottom indicates: 批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 4 行 | 行 2, 列 9

C. 在查询结果中增加计算列，还可修改数据列的显示名称

如：查询 **Student** 中所有学生的学号、姓名、性别和出生年号。

```
USE DBS
SELECT Sno 学号, Sname 姓名, Ssex 性别,
       Year(GetDate())-Sage 出生年号
FROM Student
```

	学号	姓名	性别	出生年号
1	95001	李勇	男	1983
2	95002	刘晨	女	1984
3	95003	王敏	女	1985
4	95004	张立	男	1984

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 4 行 | 行 3, 列 8

二、选择行

A. 使用 WHERE 子句，可以选择满足条件的部分记录

如：查询学生成绩在 85~90 分之间的学生情况

```
USE DBS
SELECT Sno, Cno, Grade
FROM SC
WHERE Grade>=85 AND Grade<=90
```

	Sno	Cno	Grade
1	95001	2	85
2	95001	3	88
3	95002	2	90

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 3 行 | 行 4, 列 30

B. 使用 DISTINCT 关键字，可以消除重复记录

如：查询有成绩的学生的学号，[参见](#)

```
USE DBS
SELECT DISTINCT Sno
FROM SC
```

	Sno
1	95001
2	95002

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 2 行 | 行 2, 列 17

注：若不使用 DISTINCT，是什么情况？

C. 使用 IN 关键字，适于选择不连续条件的记录

如：查询学生成绩为 80 或 85 的学生的学号

<pre>USE DBS SELECT * FROM SC WHERE Grade IN (80,85)</pre>				
	Sno	Cno	Grade	
1	95001	2	85	
2	95002	3	80	

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 2 行 | 行 5, 列 1

注：在 WHERE 子句中，若使用 “=” 如何实现？

D. 使用谓词 LIKE 和通配符 “%” 或 “_”，实现模糊查询

如：查询姓“张”的学生的基本情况

USE DBS
SELECT *
FROM Student
WHERE Sname LIKE '张%'

	Sno	Sname	Ssex	Sage	Sdept
1	95004	张立	男	19	IS

网格

消息

批查询完成。

lwhNote (8.0)

sa (51)

DBS

0:00:00

1 行

行 4, 列 21

注：“%”代表0个或多个字符，“_”代表一个字符。有的书上说，一个汉字占两个字符，但这里一个汉字只占一个字符位置，与系统设置有关，你的系统？

如：查询姓名只有二个汉字，且姓“张”的学生的基本情况

The screenshot displays a database query window with a SQL query and its results. The query is as follows:

```
USE DBS
SELECT *
FROM Student
WHERE Sname LIKE ' _勇'
```

The results are shown in a table with the following columns: Sno, Sname, Ssex, Sage, Sdept. The table contains one row of data:

	Sno	Sname	Ssex	Sage	Sdept
1	95001	李勇	男	20	CS

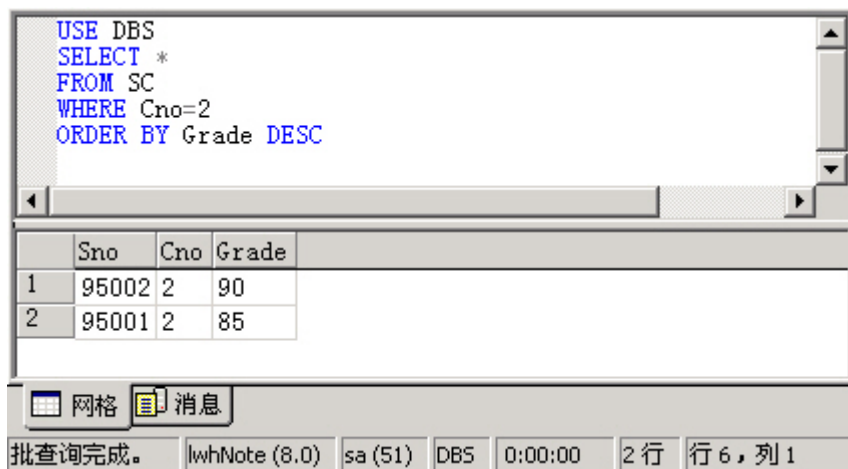
At the bottom of the window, there is a status bar with the following information:

- 批查询完成。 (Batch query completed.)
- lwhNote (8.0)
- sa (51)
- DBS
- 0:00:00
- 1 行 (1 row)
- 行 5, 列 1 (Row 5, Column 1)

三、对查询结果排序

A. 使用 ORDER 子句，对查询结果排序

如：查询所有学生中 2 号课的成绩，并按成绩由高向低排序



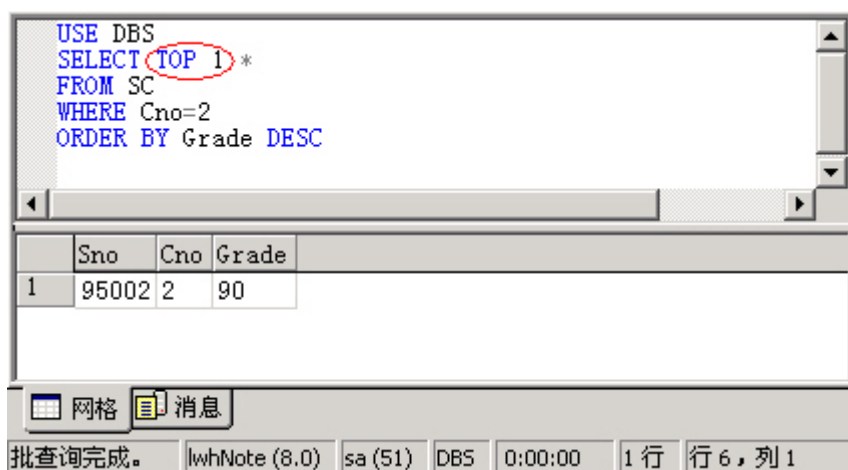
```
USE DBS
SELECT *
FROM SC
WHERE Cno=2
ORDER BY Grade DESC
```

	Sno	Cno	Grade
1	95002	2	90
2	95001	2	85

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 2 行 | 行 6，列 1

B. 使用 TOP 关键字，选择查询结果的前几个记录

如：查询所有学生中 2 号课成绩最高的学生记录



```
USE DBS
SELECT TOP 1 *
FROM SC
WHERE Cno=2
ORDER BY Grade DESC
```

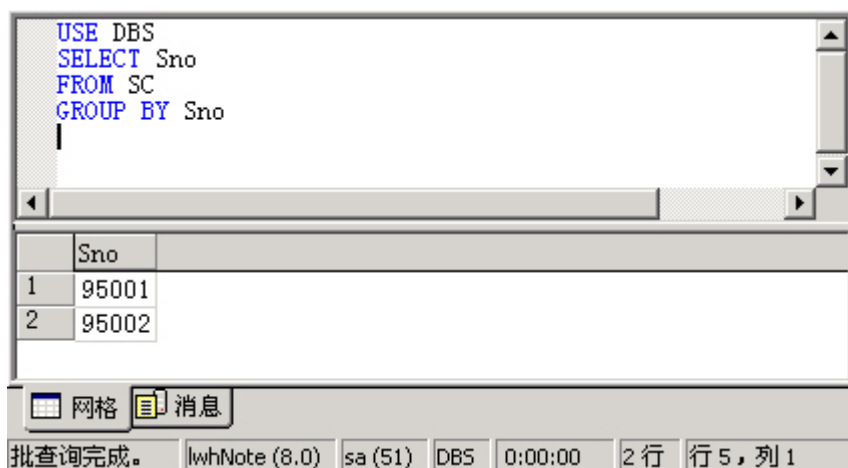
	Sno	Cno	Grade
1	95002	2	90

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 1 行 | 行 6，列 1

四、对查询结果分组

A. 使用 GROUP 子句，对查询结果分组

如：查询有成绩的学生的学号，[参见](#)



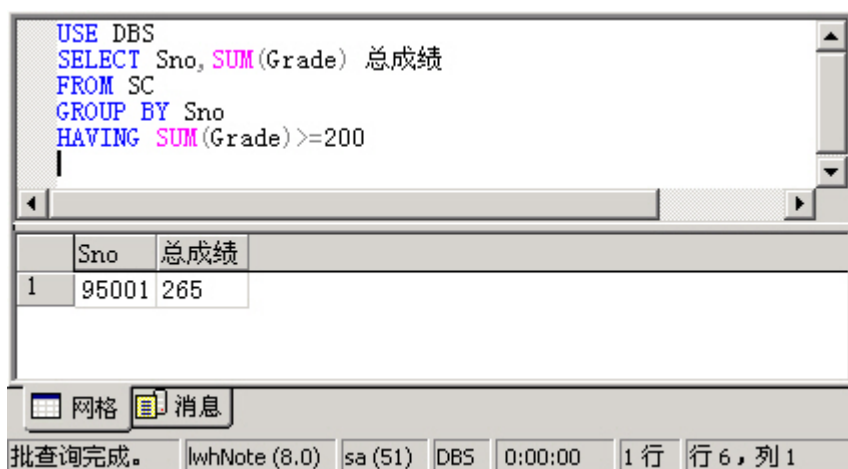
```
USE DBS
SELECT Sno
FROM SC
GROUP BY Sno
```

	Sno
1	95001
2	95002

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 2 行 | 行 5，列 1

B. 使用 HAVING 子句，对分组结果进行筛选

如：查询总成绩在 200 分以上的学生的学号和总成绩



```
USE DBS
SELECT Sno, SUM(Grade) 总成绩
FROM SC
GROUP BY Sno
HAVING SUM(Grade) >= 200
```

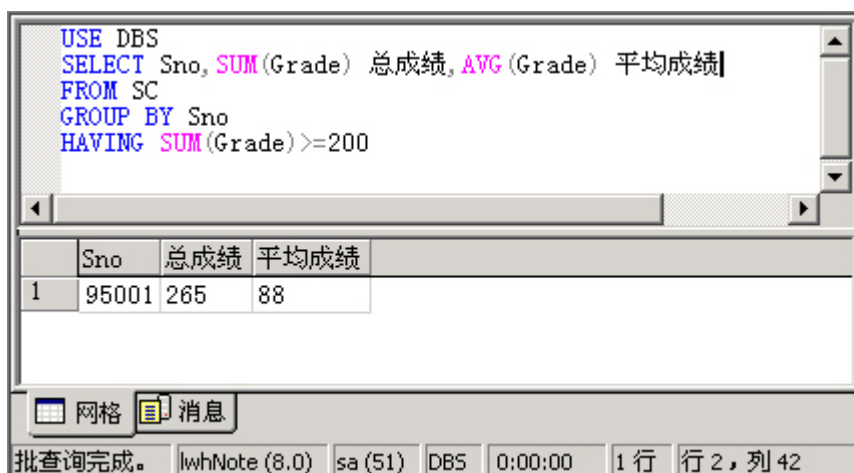
	Sno	总成绩
1	95001	265

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 1 行 | 行 6, 列 1

注：如果不使用 HAVING 子句，是什么情况？

C. 使用统计函数，SELECT 子句后面使用统计函数以外的数据列，都必须出现在 GROUP 子句中

如：查询总成绩在 200 分以上的学生的学号、总成绩和平均成绩



```
USE DBS
SELECT Sno, SUM(Grade) 总成绩, AVG(Grade) 平均成绩
FROM SC
GROUP BY Sno
HAVING SUM(Grade) >= 200
```

	Sno	总成绩	平均成绩
1	95001	265	88

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 1 行 | 行 2, 列 42

3.1.3 连接查询

 帮助目录：<访问和更改关系数据|查询基础知识>

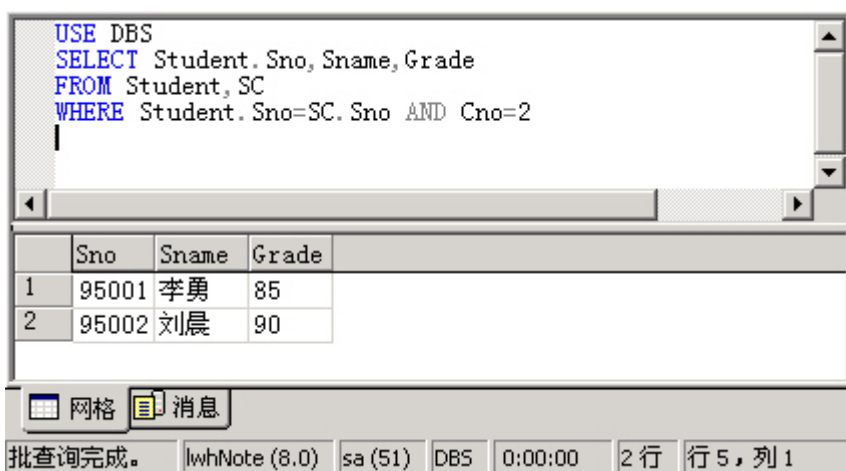
 帮助索引：<查询-SQL Server>

连接查询分为：等值连接查询、非等值连接查询、自连接查询、外部连接查询和复合条件连接查询。

一、等值连接查询

A. 用 WHERE 子句指定连接条件

如：查询所有有 2 号课程成绩的学生情况：学号、姓名、成绩



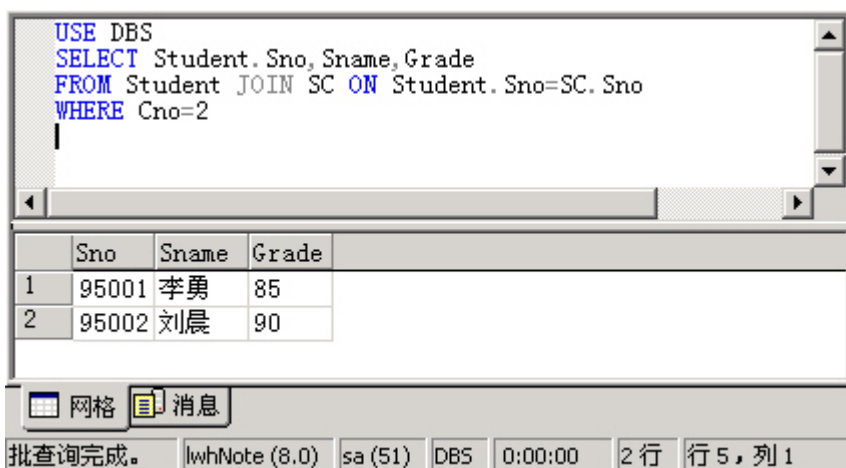
```
USE DBS
SELECT Student.Sno, Sname, Grade
FROM Student, SC
WHERE Student.Sno=SC.Sno AND Cno=2
```

	Sno	Sname	Grade
1	95001	李勇	85
2	95002	刘晨	90

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 2 行 | 行 5, 列 1

B. 在 FROM 子句中用 JOIN 连接符指定连接条件

如：查询所有有 2 号课程成绩的学生情况：学号、姓名、成绩。[参见](#)



```
USE DBS
SELECT Student.Sno, Sname, Grade
FROM Student JOIN SC ON Student.Sno=SC.Sno
WHERE Cno=2
```

	Sno	Sname	Grade
1	95001	李勇	85
2	95002	刘晨	90

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 2 行 | 行 5, 列 1

注：两种不同的方法，达到了同样的效果。

二、非等值连接查询

即在等值连接条件中不使用等号，而使用其它比较运算符就构成了非等值连接查询，可以使用的比较运算符有：>、>=、<、<=、!=，还可以使用 **BETWEEN...AND** 之类的的谓词。

注：在 DBS 数据库中，没有合适的例子。

三、自连接查询

如：查询学生中年龄相同的学生情况

USE DBS					
SELECT A. Sno, A. Sname, A. Ssex, A. Sage, A. Sdept					
FROM Student A JOIN Student B ON A. Sno!=B. Sno					
AND A. Sage=B. Sage					
	Sno	Sname	Ssex	Sage	Sdept
1	95004	张立	男	19	IS
2	95002	刘晨	女	19	IS

批查询完成。 |lwhNote (8.0) |sa (51) |DBS |0:00:00 |2行 |行 6, 列 1

注 1: 在自连接时, 对数据表必须使用别名。

注 2: 如何使用 WHERE 子句来实现上面的查询?

注 3: 观察下面查询语句的执行情况, 是否对自连接有更深刻的理解?

USE DBS					
SELECT *					
FROM Student A JOIN Student B ON A. Sno!=B. Sno					

四、外部连接查询

在前面所举的例子中, 连接的结果是从两个或两个以上的表的组合中挑选出符合连接条件的数据, 如果数据无法满足连接条件则将其丢弃, 通常称这种方法为内部连接 (Inner Join)。

在内部连接中, 参与连接的表的地位是平等的。与内部连接相对的方式称为外部连接 (Outer Join)。 在外部连接中, 参与连接的表有主从之分, 以主表的每行数据去匹配从表的数据列, 符合连接条件的数据将直接返回到结果集中, 对那些不符合连接条件的列, 将被填上 NULL 值后再返回到结果集中 (对 BIT 类型的列, 由于 BIT 数据类型不允许 NULL 值, 因此将会被填上 0 值再返回到结果集中)。

外部连接分为左外部连接 (Left Outer Join) 和右外部连接 (Right Outer Join) 两种。以主表所在的方向区分外部连接, 主表在左边则称为左外部连接, 主表在右边, 则称为右外部连接。如: 查询所有学生的总成绩 (包括没有成绩的学生): 学号、姓名、总成绩。

USE DBS					
SELECT A. Sno, Sname, SUM(Grade) 总成绩					
FROM Student A LEFT JOIN SC ON A. Sno=SC. Sno					
GROUP BY A. Sno, Sname					
	Sno	Sname	总成绩		
1	95001	李勇	265		
2	95002	刘晨	170		
3	95003	王敏	NULL		
4	95004	张立	NULL		

批查询完成。 |lwhNote (8.0) |sa (51) |DBS |0:00:00 |4行 |行 5, 列 1

试一试下面的语句，观察运行结果，是不是对左右连接有进一步的理解？

```
USE DBS
SELECT A. Sno, Sname, SUM(Grade) 总成绩
FROM SC Left JOIN Student A ON A. Sno=SC. Sno
GROUP BY A. Sno, Sname
```

```
USE DBS
SELECT A. Sno, Sname, SUM(Grade) 总成绩
FROM SC Right JOIN Student A ON A. Sno=SC. Sno
GROUP BY A. Sno, Sname
```

五、复合条件连接查询

在 WHERE 子句中使用多个连接条件的查询，称为条件连接查询

如：查询所有有 2 号课程成绩的学生情况：学号、姓名、成绩。[参见](#)

```
USE DBS
SELECT A. Sno, Sname, Grade
FROM Student A, SC
WHERE A. Sno=SC. Sno AND Cno=2
```

	Sno	Sname	Grade
1	95001	李勇	85
2	95002	刘晨	90

批查询完成。 | lwhNote (8.0) | sa (51) | DBS | 0:00:00 | 2 行 | 行 4, 列 29

3.1.4 嵌套查询

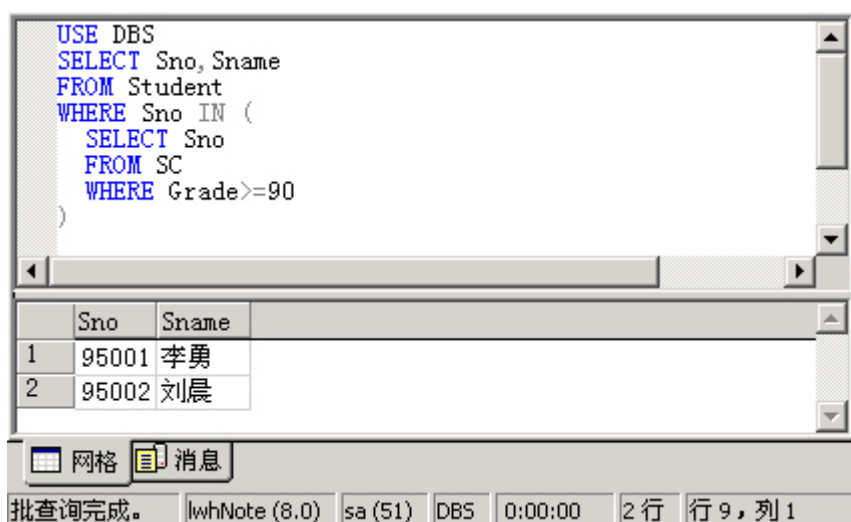
 帮助目录： <访问和更改关系数据|查询基础知识>

 帮助索引： <查询-SQL Server>

在一个 SELECT 语句的 WHERE 子句或 HAVING 子句中嵌套另一个 SELECT 语句的查询称为嵌套查询，又称子查询子查询。

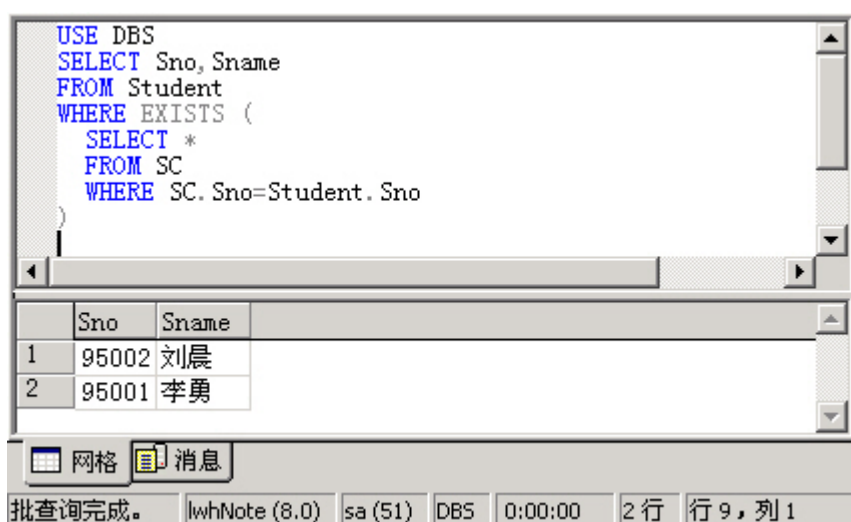
一、使用谓词 IN 连接子查询

如：查询某课程成绩在 90 分以上的学生情况学号、姓名



二、使用谓词 EXISTS 连接子查询

如：查询有课程成绩的学生情况学号、姓名



3.1.5 存储查询结果

帮助目录：<访问和更改关系数据|查询基础知识>

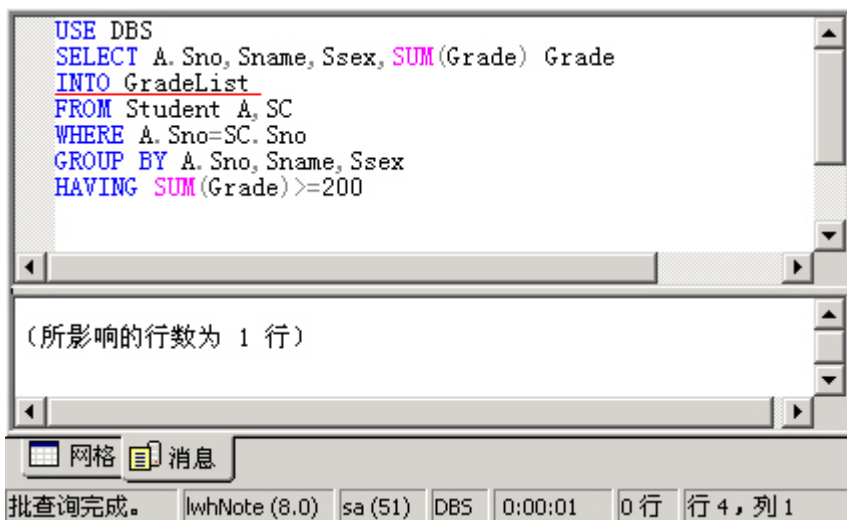
帮助索引：<查询-SQL Server>

查询的信息往往需要保存下来，以便使用。在用 SELECT 语句查询数据时，可以设定将数据存储到一个新建的表中或变量中。

一、将查询结果存储到表中

如：将总分在 200 分以上的学生的情况存（学号、姓名、性别、总成绩）到数据表

GradeListk 中。



到企业管理器中查看是否已经创建数据表 **GradeList**，并观察查询结果是否正确。

二、将查询结果保存到变量中

如：查询 95001 学生的 1 号课的成绩，将其保存到变量 **Grade** 中。



注：要将查询结果保存到变量时，只能将查询结果集中第一条记录的值赋给变量。

3.2 视图操作

视图是从一个或几个基本表（或视图）导出的表，它与基本表不同，是一个虚表。

数据库中只存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。所有基本表中的数据发生变化，从视图中查询出的数据也随之改变。

视图一经定义，就可以和基本表一样被查询、删除，也可以在一个视图上再定义新的视图，但对视图的更新（增加、修改、删除）操作则有一定的限制。

3.2.1 创建视图

 帮助目录：<访问和更改关系数据|查询基础知识>

 帮助索引: <视图>

格式:

```
CREATE VIEW <视图名>[(<列名>[,<列名>]...)]
```

```
AS <子查询>
```

```
[WITH CHECK OPTION];
```

其中:

① 子查询可以是任意复杂的 **SELECT** 语句, 但通常不允许含有 **ORDER BY** 子句和

DISTINCT 短语。

② **WITH CHECK OPTION** 表示对视图进行 **UPDATE**、**INSERT** 和 **DELETE** 操作时要保证更新、插入或删除的行满足视图定义中的谓词条件 (即子查询中的条件表达式)。

③ 组成视图的属性列或者全部省略或者全部指定, 没有第三种选择。如果省略了组成视图的各个属性列名, 则隐含该视图由子查询中 **SELECT** 子句目标列中的诸字段组成。但在下列三种情况下必须明确指定组成视图的所有列名:

- 其中某个目标列不是单纯的属性名, 而是集函数或列表表达式;
- 多表连接时选出了几个同名列作为视图的字段;
- 需要在视图中为某个列启用新的更合适的名字。

例 1 建立信息系学生的视图。

-- 建立视图			
CREATE VIEW IS_Student AS			
SELECT Sno, Sname, Sage			
FROM Student			
WHERE Sdept='IS'			
GO			
	Sno	Sname	Sage
1	95002	刘晨	19
2	95004	张立	19

注 1: 本例中省略了视图 **IS_Student** 的列名, 隐含了由查询中 **SELECT** 子句中的三个列名组成。

注 2: 执行 **CREATE VIEW** 语句的结果只是把对视图的定义存入数据字典, 并不执行其中的 **SELECT** 语句。只是在对视图查询时, 才按视图的定义从基本表中将数据查出。

例 2 建立信息系学生的视图, 并要求进行修改和插入操作时仍须保证该视图只有信息系的学生。

-- 建立视图			
CREATE VIEW IS_Student AS			
SELECT Sno, Sname, Sage			
FROM Student			
WHERE Sdept='IS'			
WITH CHECK OPTION			
GO			
	Sno	Sname	Sage
1	95002	刘晨	19
2	95004	张立	19

注 1: 由于在定义 IS_Student 视图时加上了 WITH CHECK OPTION 子句, 以后对该视图进行插入、修改和删除操作时, DBMS 会自动加上 Sdept='IS' 的条件。

注 2: 若一个视图是从单个基本表导出的, 并且只是去掉了基本表的某些行和某些列, 但保留了码, 我们称这类视图为行列子集视图。上面二个例子的视图 IS_Student 就是一个行列子集视图。

例 3 建立信息系选修了 2 号课程的学生的视图。

-- 建立视图			
CREATE VIEW IS_S1 (Sno, Sname, Grade) AS			
SELECT Student.Sno, Sname, Grade			
FROM Student, SC			
WHERE Sdept='IS' AND Student.Sno=SC.Sno AND SC.Cno='2'			
GO			
	Sno	Sname	Grade
1	95002	刘晨	90

注 1: 视图不仅可以建立在一个或多个基本表上。

注 2: 本例中定义了视图的属性列。

例 4 建立信息系选修了 2 号课程且成绩在 90 分以上的学生的视图。

-- 建立视图			
CREATE VIEW IS_S2 AS			
SELECT Sno, Sname, Grade			
FROM IS_S1			
WHERE Grade >= 90			
GO			
	Sno	Sname	Grade
1	95002	刘晨	90

注: 视图不仅可以建立在一个或多个基本表上, 也可以建立在一个或多个已定义好的视图上, 或同时建立在基本表与视图上。

例 5 定义一个反映学生出生年份的视图。

-- 建立视图				
CREATE VIEW BT_S (Sno, Sname, Sbirth) AS				
SELECT Sno, Sname, Year(GetDate())-Sage				
FROM Student				
GO				
	Sno	Sname	Sbirth	
1	95001	李勇	1983	
2	95002	刘晨	1984	
3	95003	王敏	1985	
4	95004	张立	1984	

注：定义基本表时，为了减少数据库中的冗余数据，表中只存放基本数据，由基本数据经过各种计算派生出的数据一般是不存储的。但由于视图中的数据并不实际存储，所以定义视图时可以根据应用的需要，设置一些派生属性列。这些派生属性由于在基本表中并不实际存在，所以有时也称他们为**虚拟列**。带虚拟列的视图我们称为**带表达式的视图**。

例 6 将学生的学号及他的平均成绩定义为一个视图。

-- 建立视图		
CREATE VIEW S_G (Sno, Gavg) AS		
SELECT Sno, AVG(Grade)		
FROM SC		
GROUP BY Sno		
GO		
	Sno	Gavg
1	95001	88
2	95002	85

注：可以用带有集函数和 GROUP BY 子句的查询来定义视图。这种视图称为**分组视图**。

例 7 将 Student 表中所有女生记录定义为一个视图。

-- 建立视图					
CREATE VIEW F_Student (Stdnum, name, sex, age, dept) AS					
SELECT *					
FROM Student					
WHERE Ssex='女'					
GO					
	Stdnum	name	sex	age	dept
1	95002	刘晨	女	19	IS
2	95003	王敏	女	18	MA

注：在建立视图时，明确指明属性列名，而不是简单地

3.2.2 删除视图

 帮助目录：<访问和更改关系数据|查询基础知识>

 帮助索引：<视图|删除>

格式：

DROP VIEW <视图名>;

说明：

① 视图是由基本表导出的，若基本表的结构改变了，视图与基本表的映射关系被破坏，视图就不能正常工作，最好的办法就是删除这些视图后，重新建立。

② 一个视图被删除后，由此视图导出的其他视图也将失效，用户应该使用 **DROP VIEW** 语句将他们一一删除。

也可以在企业管理器中删除视图。

3.2.3 查询视图

 帮助目录：<访问和更改关系数据|查询基础知识>

 帮助索引：<视图>

视图定义后，用户就可以象对基本表进行查询一样对视图进行查询了。

DBMS 执行对视图的查询时，首先进行有效性检查，检查查询涉及的表、视图等是否在数据库中存在，如果存在，则从数据字典中取出查询涉及的视图的定义，把定义中的子查询和用户对视图的查询结合起来，转换成对基本表的查询，然后再执行这个经过修正的查询。将对视图的查询转换为对基本表的查询的过程称为**视图消解(View Resolution)**。

例 1 在信息系学生的视图中找出年龄小于 20 岁的学生。

-- 例1 在信息系学生的视图中找出年龄小于20岁的学生。		
-- 对视图的查询		
SELECT	Sno, Sage	
FROM	IS_Student	
WHERE	Sage<20	
-- 对基本表的查询（即视图消解后的查询）		
SELECT	Sno, Sage	
FROM	Student	
WHERE	Sdept='IS' AND Sage<20	

	Sno	Sage
1	95002	19
2	95004	19

例 2 查询信息系选修了 2 号课程的学生。

-- 例2 查询信息系选修了2号课程的学生。		
-- 对视图和基本表的查询		
SELECT IS_Student.Sno, Sname		
FROM IS_Student, SC		
WHERE IS_Student.Sno=SC.Sno AND SC.Cno='2'		
-- 对基本表的查询（即视图消解后的查询）		
SELECT Student.Sno, Sname		
FROM Student, SC		
WHERE Student.Sno=SC.Sno AND Sdept='IS' AND SC.Cno='2'		

	Sno	Sname
1	95002	刘晨

注：本查询涉及对视图和基本表同时查询。

例 3 在 S_G 视图中查询平均成绩在 86 分以上的学生学号和平均成绩。

--例3 在S_G视图中查询平均成绩在86分以上的学生学号和平均成绩。		
SELECT *		
FROM S_G		
WHERE Gavg>=86		
-- 对基本表的查询（即视图消解后的查询）		
SELECT Sno, AVG(Grade)		
FROM SC		
GROUP BY Sno		
HAVING AVG(Grade)>=86		

	Sno	Gavg
1	95001	88

注：多数关系数据库系统对行列子集视图的查询均能进行正确转换。但对非行列子集的查询就不一定能做转换了。

3.2.4 更新视图

 帮助目录： <访问和更改关系数据|查询基础知识>

 帮助索引： <视图>

更新视图是指通过视图来插入 (INSERT)、删除 (DELETE) 和修改 (UPDATE) 数据。由于视图是不实际存储数据的虚表，因此对视图的更新，最终要转换为对基本表的更新。

为防止用户通过视图对数据进行增删改时，无意或故意操作不属于视图范围内的基本表数据，可在定义视图时加上 **WITH CHECK OPTION** 子句，这样在视图上增删改数据时，DBMS 会进一步检查视图定义中的条件，若不满足条件，则拒绝执行该操作。

例 1 将信息系学生视图 IS_Student 中学号为 95002 的学生姓名改为“刘辰”。


```

-- 例1 将信息系学生视图IS_Student中学号为95002的
-- 学生姓名改为“刘辰”。
-- 对视图更新
UPDATE IS_Student
SET Sname='刘辰'
WHERE Sno='95002'
-- 对基表更新
UPDATE Student
SET Sname='刘辰'
WHERE Sno='95002' AND Sdept='IS'

```

例 2 向信息系学生视图 IS_S 中插入一个新的学生记录，其中学号为 95029，姓名为赵新，年龄为 20 岁。

```

-- 例2 向信息系学生视图IS_S中插入一个新的学生记录，
-- 其中学号为95029，姓名为赵新，年龄为20岁。
-- 对视图插入
INSERT INTO IS_Student
VALUES ('95029', '赵新', 20)
-- 对基表插入
INSERT INTO Student (Sno, Sname, Sage)
VALUES ('95029', '赵新', 20)

```

Sno	Sname	Ssex	Sage	Sdept
95001	李勇	男	20	CS
95002	刘辰	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS
95029	赵新		20	

注：在 SQL Server 中向视图 IS_Student 中插入数据时，不会将'IS'自动插入到基表中。

例 3 删除计算机系学生视图 CS_S 中学号为 95029 的记录。

```

-- 例3 删除计算机系学生视图CS_S中学号为95029的记录。
-- 对视图删除
DELETE
FROM IS_Student
WHERE Sno='95002'
-- 对基表删除
DELETE
FROM Student
WHERE Sno='95029' AND Sdept='IS'

```

注：若 Student 有外键约束，是不能删除该记录的。

说明：在关系数据库中，并不是所有的视图都是可更新的，因为有些视图的更新不能唯一地有意义地转换成对相应基本表的更新。例如 DB2 规定（在 SQL Server 中也有类似的规定）：

- ① 若视图是由两个以上基本表导出的，则此视图不允许更新。
- ② 若视图的字段来自字段表达式或常数，则不允许对此视图执行 INSERT 和 UPDATE 操作，但允许执行 DELETE 操作。
- ③ 若视图的字段来自集函数，则此视图不允许更新。
- ④ 若视图定义中含有 GROUP BY 子句，则此视图不允许更新。

⑤ 若视图定义中含有 **DISTINCT** 短语，则此视图不允许更新。

⑥ 若视图定义中有嵌套查询，并且内层查询的 **FROM** 子句中涉及的表也是导出该视图的基本表，则此视图不允许更新。例如：将成绩在平均成绩之上的元组定义成一个视图

GOOD_SC:

```
CREATE VIEW GOOD_SC AS
SELECT Sno, Cno, Grade
FROM SC WHERE Grade > (SELECT AVG(Grade) FROM SC);
```

导出视图 GOOD_SC 的基本表是 SC，内层查询中涉及的表也是 SC，所以视图 GOOD_SC 是不允许更新的。

⑦ 一个不允许更新的视图上定义的视图也不允许更新。

3.3 SQL Server 编程

在前面几个实验中，已经学习了几乎所有编写 SQL Server 脚本必须掌握的知识。由一些基本逻辑组成的这些脚本可以转换为功能强大的应用程序。SQL Server 提供若干不同的逻辑语句，可以加到你的脚本中。它们包括 **IF...ELSE** 逻辑和用于循环的 **WHILE** 结构。SQL 也为生成错误信息提供了功能，你可以把这些出错信息返回给用户。

3.3.1 批处理



帮助目录: <Transact-SQL 参考>



帮助索引: <批处理>

批处理是一组 SQL 语句，它被应用程序同时发送给 SQL Server 执行。SQL Server 从批处理中读取所有语句，并把它们编译成可执行的单元。该可执行单元就是执行计划。SQL Server 然后就一次执行执行计划中的所有语句。

SQL Server 可有多种方法来处理批处理中发生的错误：

- 如果批处理中有任何语句出现了语法错误，SQL Server 都不会生成执行计划。因此，批处理中任何语句都无法运行。例如：错拼单词“**SELECT**”造成的语法错误。
- 大多数运行期间发生的错误，比如算术溢出，将使当前执行的语句终止并中断该语句之后的运行。

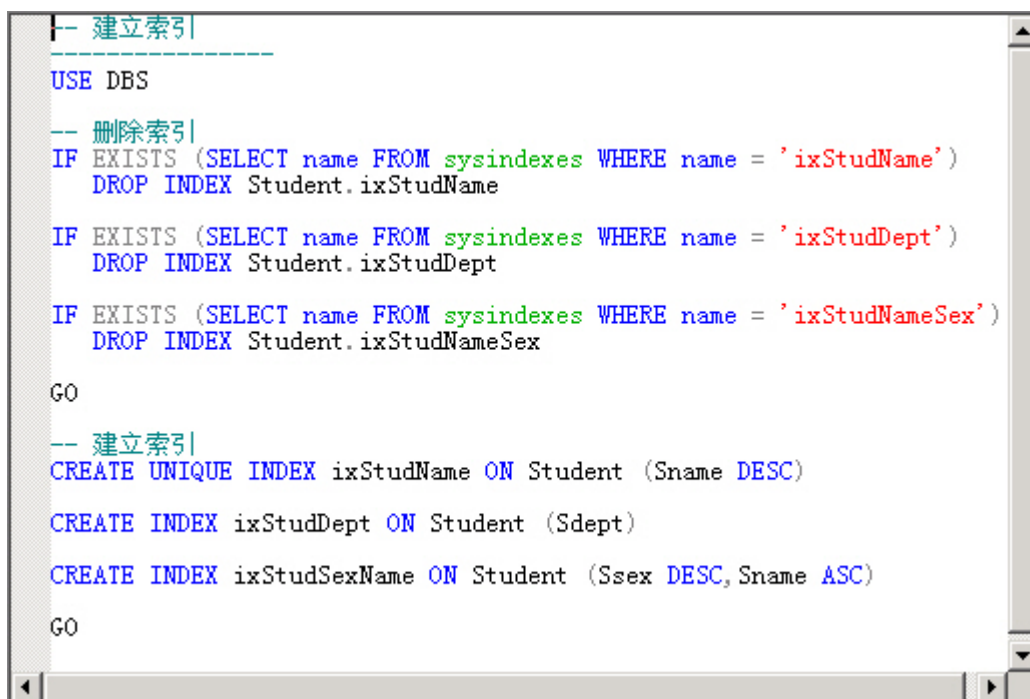
- 大多数 **CREATE** 语句不能在同一个批处理中混合使用。换句话说，在同一个批处理里，你不可以既运行 **CREATE TABLE** 语句，也运行其他 **CREATE** 语句。

- 在同一个批处理中你不能运行一个 **ALTER TABLE** 命令后接着引用增加到该表中的新列，这是因为 **SQL Server** 提前编译批处理并在该表中查找那些列。因为那些列还未生成，所以导致编译失败。

- 如果你在同一个批处理中运行多个存储过程，**SQL Server** 要求它们运行时使用 **EXECUTE** 语句。

为了在 **SQL** 脚本中给批定界，**SQL Server** 用到了关键字 **GO**。用这个关键字时，我们要做的是把 **GO** 放在一系列语句之后，使它们能作为一个单独的批处理运行。

如：在下面的程序中，就有 2 个批处理。



```
-- 建立索引
USE DBS

-- 删除索引
IF EXISTS (SELECT name FROM sysindexes WHERE name = 'ixStudName')
    DROP INDEX Student.ixStudName

IF EXISTS (SELECT name FROM sysindexes WHERE name = 'ixStudDept')
    DROP INDEX Student.ixStudDept

IF EXISTS (SELECT name FROM sysindexes WHERE name = 'ixStudNameSex')
    DROP INDEX Student.ixStudNameSex

GO

-- 建立索引
CREATE UNIQUE INDEX ixStudName ON Student (Sname DESC)

CREATE INDEX ixStudDept ON Student (Sdept)

CREATE INDEX ixStudSexName ON Student (Ssex DESC, Sname ASC)

GO
```

批处理的另一个好处是它们可以把你的代码分成小段，即使前一个小段运行失败，其他的仍然可以继续。这使得我们能使用事务的概念。

3.3.2 事务

 帮助目录： <Transact-SQL 参考>

 帮助索引： <事务>

事务是 **SQL Server** 防止你的数据出现不一致状态的基础结构。事务是用户定义的一个

操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位，是数据库环境中的逻辑工作单位。SQL Server 中事务有两种，它们是隐式事务和显式事物。

一、隐式事务

隐式事务是 SQL Server 为你而做的事务。隐式事务又称自动提交事务。如果运行一条 INSERT 语句,SQL Server 将把它包装到事务中,如果此 INSERT 语句失败,SQL Server 将回滚或取消这个事务。每条 SQL 语句均被视为一个自身的事务。

如：在一个 sql 批文件中，向数据表 SC 中插入数据，每条 SQL 语句都是一个隐式事务，如下图

```
-- 向数据表SC中插入数据
USE DBS
GO

-- 删除SC表中的数据
DELETE SC

-- 向SC表中插入数据
INSERT SC VALUES('95001','1',92);
INSERT SC VALUES('95001','2',85);
INSERT SC VALUES('95001','2',88);
INSERT SC VALUES('95002','2',90);
GO
```

例中第 3 条插入语句有错误（关键字与第 2 条插入语句相同），系统会正确插入 1、2、4 条语句，执行第 3 条语句后发现错误，自动回滚。

二、显式事务

显式事务是一种由你自己指定的事务。这种事务允许你自己决定哪批工作必须成功完成，否则所有部分都不完成。为了给自己的事务定界，可以使用关键字 BEGIN TRANSACTION 和 ROLLBACK TRANSACTION 或 COMMIT TRANSACTION。

- BEGIN TRANSACTION——用来通知 SQL Server 一个事务就要开始了。BEGIN TRANSACTION 后面发生的每一条 SQL 语句都是同一个事务中的一部分。
- ROLLBACK TRANSACTION——用来通知 SQL Server 自 BEGIN TRANSACTION 后的所有工作都应取消，对数据库中任何数据的改变都被还原，任何已经创建或删除的对象被清除或恢复。
- COMMIT TRANSACTION——用来通知 SQL Server 自 BEGIN TRANSACTION 后的全部工作都要完成并成为数据库的一个永久性部分。

在同一个事务中，你不能同时使用 ROLLBACK TRANSACTION 和 COMMIT

TRANSACTION。你必须意识到，即使你的脚本中有错误，而你又让 SQL Server 提交事务，该事务也将执行。如果你打算依赖于现实事务保证数据完整性，必须在脚本中建立错误检查机制。

例 1：显式事务（回滚），在一个 sql 批文件中，若在一个显式事务中删除数据表 SC，然后回滚事务，其删除无效。

```
-- 向数据表SC中插入数据，用事务的概念
USE DBS
GO
BEGIN TRANSACTION
-- 显示删除前SC中的数据（有n条记录）
SELECT * FROM SC
GO
-- 删除SC表中的数据
DELETE SC
GO
-- 显示删除后SC中的数据（无记录）
SELECT * FROM SC
GO
-- 回滚事务，并显示回滚后SC中的数据（有n条记录）
ROLLBACK
SELECT * FROM SC
GO
```

例 2：显式事务（提交），在一个 sql 批文件中，若在一个显式事务中删除数据表 SC，然后提交事务，其删除有效。

```
-- 向数据表SC中插入数据，用事务的概念
USE DBS
GO
BEGIN TRANSACTION
-- 显示删除前SC中的数据（有n条记录）
SELECT * FROM SC
GO
-- 删除SC表中的数据
DELETE SC
GO
-- 显示删除后SC中的数据（无记录）
SELECT * FROM SC
GO
-- 回滚事务，并显示回滚后SC中的数据（无记录）
COMMIT
SELECT * FROM SC
GO
```

3.3.3 流控制

 帮助目录： <Transact-SQL 参考>

SQL Server 提供了几种流控制关键字，可用于控制 SQL Server 如何去执行批处理或过程中的语句。如果不使用流控制关键词，SQL Server 将按顺序方式执行脚本中的语句。相反，如果使用了流控制语言，就可以将语句分组并根据批处理或过程中的判断标准来

控制它们的执行。

必须注意：流控制语句功能虽然强大，但它不能跨越多个批处理或过程。

一、IF...ELSE 语句

 帮助索引：<IF...ELSE>

```
IF @FLAG=1
    SELECT * FROM Student
ELSE
    SELECT * FROM Course
```

二、BEGIN...END 语句

 帮助索引：<BEGIN...END>

多条 SQL 语句可以使用 **BEGIN...END** 复合成一条语句。

```
IF @FLAG=1
BEGIN
    <多条 SQL 语句>
END
```

三、WHILE 语句

 帮助索引：<WHILE...BREAK和WHILE...CONTINUE>

```
WHILE <逻辑表达式>
    <SQL 语句>
[BREAK]
[CONTINUE]
```

说明：执行到 **BREAK** 语句时，会跳出当前循环；执行到 **CONTINUE** 语句时，会跳过当前循环剩余的语句。

3.3.4 注释

 帮助目录：<Transact-SQL 参考>

 帮助索引：<注释>

注释代码是一件非常重要的事情，这样可以保证你和别人在阅读这些代码时弄清它的意思。你应该在代码中放进注释，说明它用来做什么、使用什么参数、输出了什么。你还应该

在代码发生变化时加以注释并说明原因。

你也可以用注释符使目前不想运行的部分代码临时失效。这一功能在调试时非常有用。

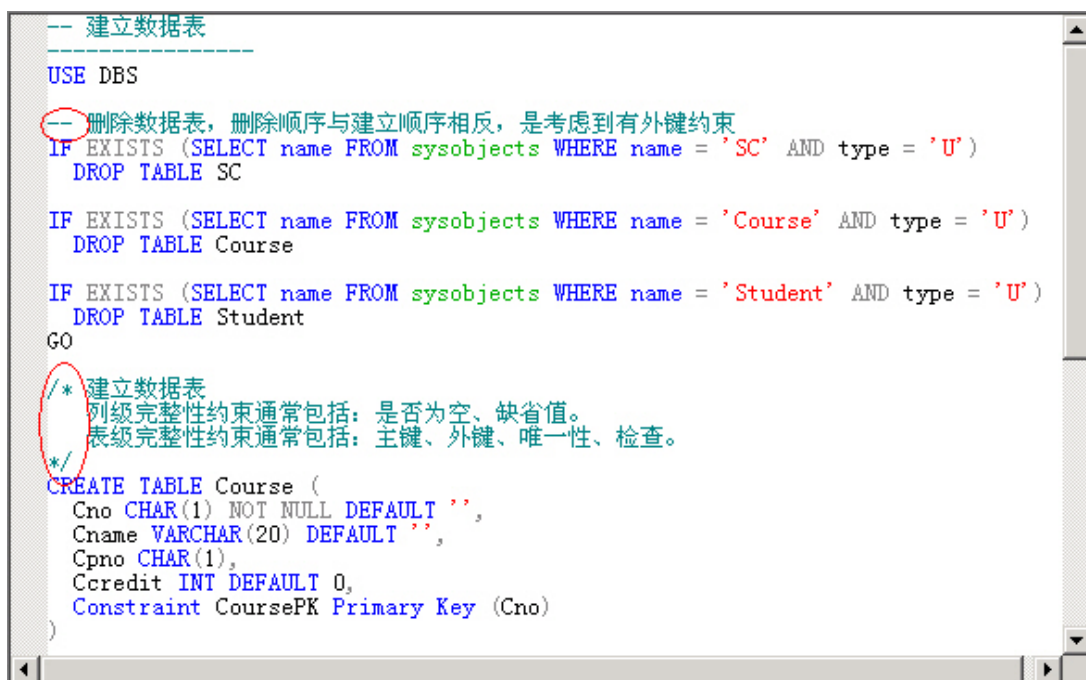
SQL Server 支持两种注释符：

- `/*...*/` ——这种注释符可用来注释一整块的程序，包括多个行。`/*`用来表示注释开始，`*/`用来表示注释结束。必须注意：在注释语句中不能使用单词 **GO**。否则注释区就会破裂，SQL Server 将试图执行注释区的语句，导致系统出错。

- `--` ——这种注释符是用来注释单个行的，这可以用在每行的开头和结尾。当要在多行中使用时，你必须在每行的开头都放上 `--` 注释符。

注释的重要性怎么强调都不为过，注释中最重要的部分是解释使用这些代码的目的以及它们怎样工作。如果你有一段时间没有理会一段代码，当代码需要变动时，注释变得更重要。

如：下面的程序中使用了两种注释。



```
-- 建立数据表
USE DBS

-- 删除数据表，删除顺序与建立顺序相反，是考虑到有外键约束
IF EXISTS (SELECT name FROM sysobjects WHERE name = 'SC' AND type = 'U')
    DROP TABLE SC

IF EXISTS (SELECT name FROM sysobjects WHERE name = 'Course' AND type = 'U')
    DROP TABLE Course

IF EXISTS (SELECT name FROM sysobjects WHERE name = 'Student' AND type = 'U')
    DROP TABLE Student
GO

/* 建立数据表
   列级完整性约束通常包括：是否为空、缺省值。
   表级完整性约束通常包括：主键、外键、唯一性、检查。
*/
CREATE TABLE Course (
    Cno CHAR(1) NOT NULL DEFAULT '',
    Cname VARCHAR(20) DEFAULT '',
    Cpno CHAR(1),
    Ccredit INT DEFAULT 0,
    Constraint CoursePK Primary Key (Cno)
)
```

3.3.5 变量

 帮助目录：<Transact-SQL 参考>

 帮助索引：<变量>

变量在编程中占有极其重要的地位。一个变量完全由你来创建，并可为它赋值。本地变量是由用户自己创建并赋值的，也只有创建它的用户可以看到它。全局变量是由一个用户创建并赋值，但可被系统中所有人看到的变量。当定义变量时，通过定义变量的数据类型来告诉 SQL Server 该变量处理什么样的数据。

SQL Server 中的变量有两种：全局变量和局部变量。

全局变量由 SQL server 提供，用@@两个符号表明，如：@@cursor_rows、

@@fetch_status、@@error 等，任何过程都可以引用这些变量，不必在使用前对它们声明，SQL Server 通过名字识别它们。

局部变量是由过程开发人员声明并使用的。下面对局部变量进行说明。

一、声明变量（DECLARE 语句）

格式：

```
DECLARE @<变量名> <变量类型>[, @<变量名> <变量类型>]...
```

说明：可以声明除 image 以外的所有数据类型。

举例：

```
DECLARE @ASTR VARCHAR(10)
```

```
DECLARE @NUM INT, @DATE DATETIME, @FLAG BIT
```

二、设置变量的值

格式一：使用 SELECT 语句

```
SELECT @ASTR = 'TAXI'
```

```
SELECT @NUM = COUNT(*) FROM Student
```

注：第二种情况下查询的结果只能是一行。

格式二：使用 SET 语句

```
SET @ASTR = 'TAXI'
```

三、读取变量的值

格式：使用 SELECT 语句

```
SELECT @ASTR
```

格式：使用 PRINT 语句

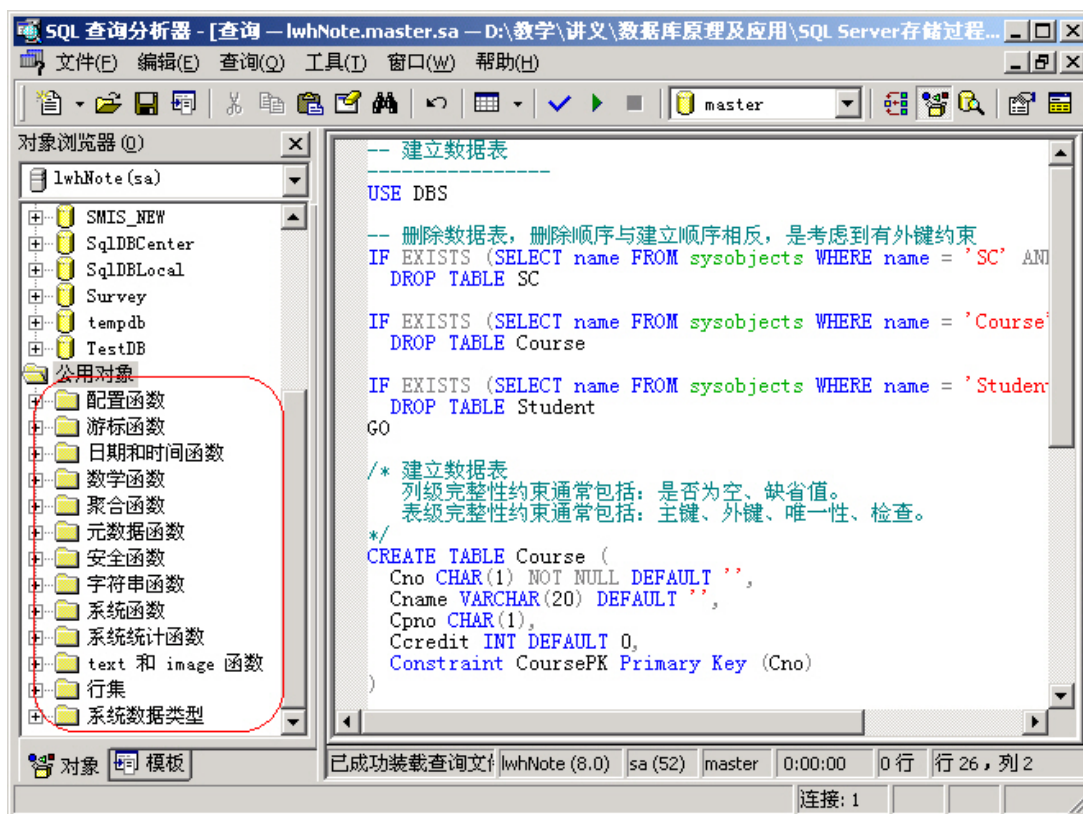
```
PRINT @ASTR
```

3.3.6 函数

 帮助目录：<Transact-SQL 参考>

 帮助索引：<函数>

函数是 Transact-SQL 编程的一个重要组成部分，SQL 中的函数的分类可以参见查询分析器中对象浏览器。如下图



至于某个函数的使用，参见系统帮助。

3.4 实验内容

阅读本实验 3.1~3.2 的内容，完成下面的实验内容。

一、恢复数据库

在查询分析器中打开上个实验中保存的 Sql 文件 CreateTabs.sql 重新建立 DBS，并用 InsStudent.sql、InsCourse.sql、IncSC.sql 追加 DBS 中三个数据表的数据。

二、查询数据表中的数据

按照<3.1 数据库查询>中指导的内容熟悉各种查询语句。

三、视图操作

按照<3.2 视图操作>中指导的内容熟悉各种视图操作。

四、SQL Server 编程

理解 CreateTabs.sql、InsStudent.sql、InsCourse.sql、IncSC.sql 中的全部语句。

3.5 思考题

- ① 在进行数据库查询时，你能找出企业管理器与查询分析器在使用的差异吗？各有什么好处？
- ② 你能体会到 **IN** 谓词在查询时的便利吗？
- ③ 在使用 **SELECT** 语句查询时，如何限制返回的行数？
- ④ 什么数据类型可以与 **LIKE** 关键字一起使用？
- ⑤ 什么子句可以改变查询结果的排序？
- ⑥ 视图与数据库查询有什么关系？
- ⑦ 视图与数据表有什么联系与区别？
- ⑧ 是否可以将数据表建立全部写成批处理文件？如何实现？
- ⑨ 是否理解事务的概念和执行特点？
- ⑩ 如何定义一个显式事务？如何提交或取消一个事务？

实验四 存储过程、触发器和游标

在大型数据库系统中，存储过程和触发器具有很重要的作用。无论是存储过程还是触发器，都是 **SQL** 语句和流程控制语句的集合。就本质而言，触发器也是一种存储过程。存储过程在运算时生成执行方式，所以，以后对其再运行时其执行速度很快。**SQL Server 2000** 不仅提供了用户自定义存储过程的功能，而且也提供了许多可作为工具使用的系统存储过程。

4.1 存储过程

在大型数据库系统中存储过程和触发器具有很重要的作用无论是存储过程还是触发器都是 **SQL** 语句和流程控制语句的集合就本质而言触发器也是一种存储过程存储过程在运算时生成执行方式所以以后对其再运行时其执行速度很快 **SQL Server 2000** 不仅提供了用户自定义存储过程的功能而且也提供了许多可作为工具使用的系统存储过程。

4.1.1 存储过程概述



帮助索引: <存储过程>

一、存储过程的概念

存储过程(Stored Procedure)是一组为了完成特定功能的 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)来执行它。

在 SQL Server 的系列版本中存储过程分为两类:系统提供的存储过程和用户自定义存储过程。

系统过程主要存储在 master 数据库中并以 sp_ 为前缀,并且系统存储过程主要是从系统表中获取信息,从而为系统管理员管理 SQL Server 提供支持。通过系统存储过程 MS SQL Server 中的许多管理性或信息性的活动(如了解数据库对象、数据库信息)都可以被顺利地有效地完成。尽管这些系统存储过程被放在 master 数据库中,但是仍可以在其它数据库中对其进行调用,在调用时不必在存储过程名前加上数据库名。而且当创建一个新数据库时,一些系统存储过程会在新数据库中被自动创建。

用户自定义存储过程是由用户创建并能完成某一特定功能(如查询用户所需数据信息)的存储过程。本实验所涉及到的存储过程主要是指用户自定义存储过程。

二、存储过程的优点

当利用 SQL Server 创建一个应用程序时,Transact-SQL 是一种主要的编程语言。

若运用 Transact-SQL 来进行编程,有两种方法。其一,在本地存储 Transact-SQL 程序,并创建应用程序向 SQL Server 发送命令来对结果进行处理。其二,可以把部分用

Transact-SQL 编写的程序作为存储过程存储在 SQL Server 中,并创建应用程序来调用存储过程,对数据结果进行处理。存储过程能够通过接收参数向调用者返回结果集,结果集的格式由调用者确定;返回状态值给调用者,指明调用是成功或是失败;包括针对数据库的操作语句,并且可以在一个存储过程中调用另一存储过程。

我们通常更偏爱于使用第二种方法,即在 SQL Server 中使用存储过程而不是在客户计算机上调用 Transact-SQL 编写的一段程序,原因在于存储过程具有以下优点:

A. 存储过程允许标准组件式编程

存储过程在被创建以后,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。而且数据库专业人员可随时对存储过程进行修改,但对应用程序源代码毫无影响(因为应用程序源代码只包含存储过程的调用语句),从而极大地提高了程序的可移植性。

B. 存储过程能够实现较快的执行速度

如果某一操作包含大量的 **Transact-SQL** 代码或分别被多次执行，那么存储过程要比批处理的执行速度快很多。因为存储过程是预编译的，在首次运行一个存储过程时，查询优化器对其进行分析、优化，并给出最终被存在系统表中的执行计划。而批处理的

Transact-SQL 语句在每次运行时都要进行编译和优化，因此速度相对要慢一些。

C. 存储过程能够减少网络流量

对于同一个针对数据库对象的操作（如查询、修改），如果这一操作所涉及到的 **Transact-SQL** 语句被组织成一存储过程，那么当在客户计算机上调用该存储过程时，网络中传送的只是该调用语句，否则将是多条 **SQL** 语句，从而大大增加了网络流量，降低网络负载。

D. 存储过程可被作为一种安全机制来充分利用

系统管理员通过对执行某一存储过程的权限进行限制，从而能够实现对相应的数据访问权限的限制，避免非授权用户对数据的访问保证数据的安全。

4.1.2 创建存储过程

 帮助索引：<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：

- 所有的输入参数以及传给调用者的输出参数；
- 被执行的针对数据库的操作语句，包括调用其它存储过程的语句；
- 返回给调用者的状态值，以指明调用是成功还是失败。

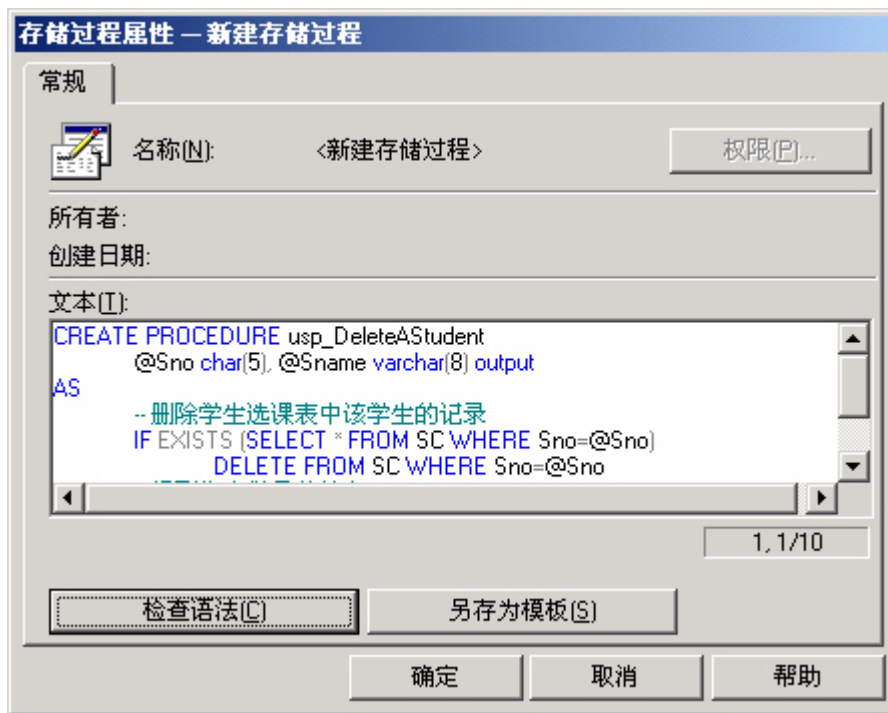
一、使用企业管理器创建存储过程

如：编写一个存储过程 **usp_DeleteAStudent**，实现删除 DBS 数据库中一个学生（指定学号）的记录，并返回删除学生的姓名。

说明：由于一个学生可能已经选课，所以在删除一个学生的数据前，必须首先删除该学生选课数据，然后再删除学生数据。

A. 检查是否存在与你希望创建的存储过程（如：**usp_DeleteAStudent**）同名的存储过程，若有，则首先删除该存储过程。

B. 给 DBS 数据库创建一个新存储过程 **usp_DeleteAStudent**



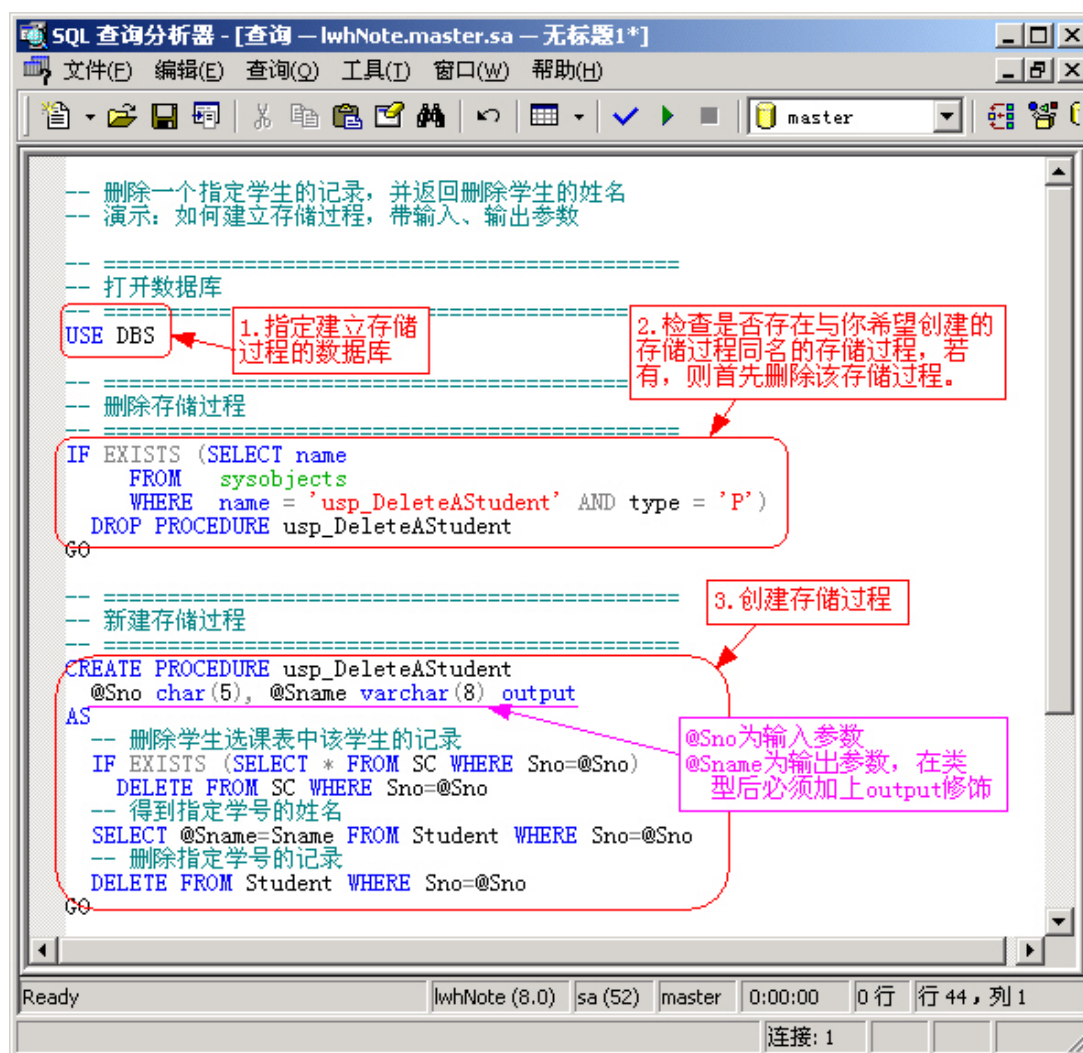
新建存储过程的详细内容如下：

```
CREATE PROCEDURE usp_DeleteAStudent
    @Sno char(5), @Sname varchar(8) output
AS
-- 删除学生选课表中该学生的记录
IF EXISTS (SELECT * FROM SC WHERE Sno=@Sno)
    DELETE FROM SC WHERE Sno=@Sno
-- 得到指定学号的姓名
SELECT @Sname=Sname FROM Student WHERE Sno=@Sno
-- 删除指定学号的记录
DELETE FROM Student WHERE Sno=@Sno
```

E. 按“确定”按钮，保存其存储过程

二、使用查询分析器创建存储过程

例 1：编写一个存储过程 `usp_DeleteAStudent`，实现删除 DBS 数据库中一个学生（指定学号）的记录，并返回删除学生的姓名。

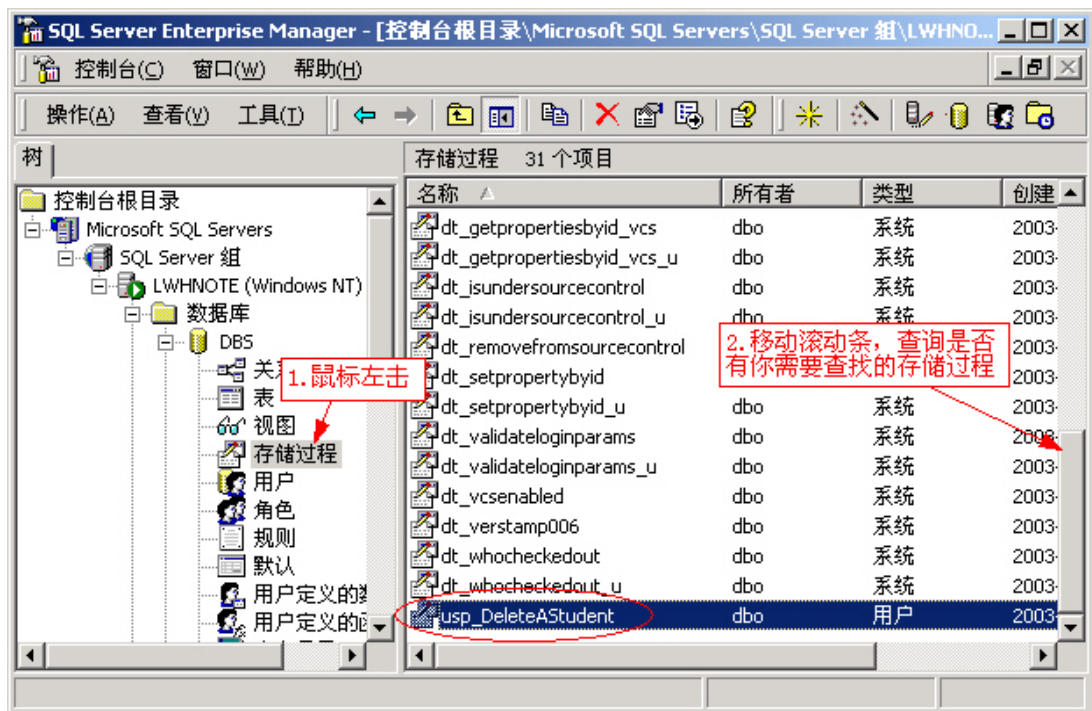


4.1.3 管理存储过程

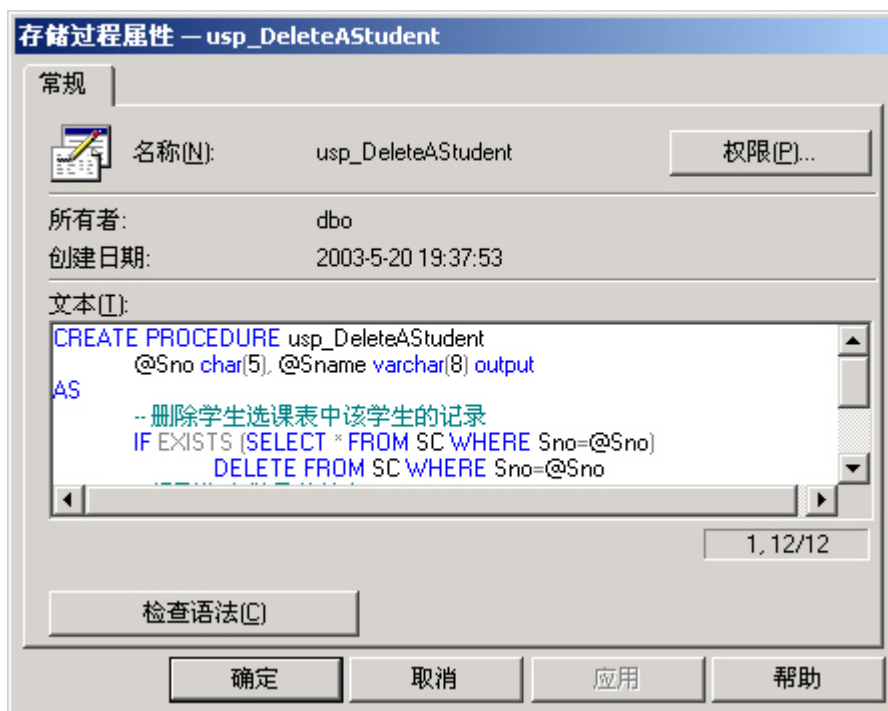
一、查看存储过程

存储过程被创建以后，它的名字存储在系统表 `sysobjects` 中；它的源代码存放在系统表 `syscomments` 中。可以通过 SQL Server 提供的系统存储过程来查看关于用户创建的存储过程信息。

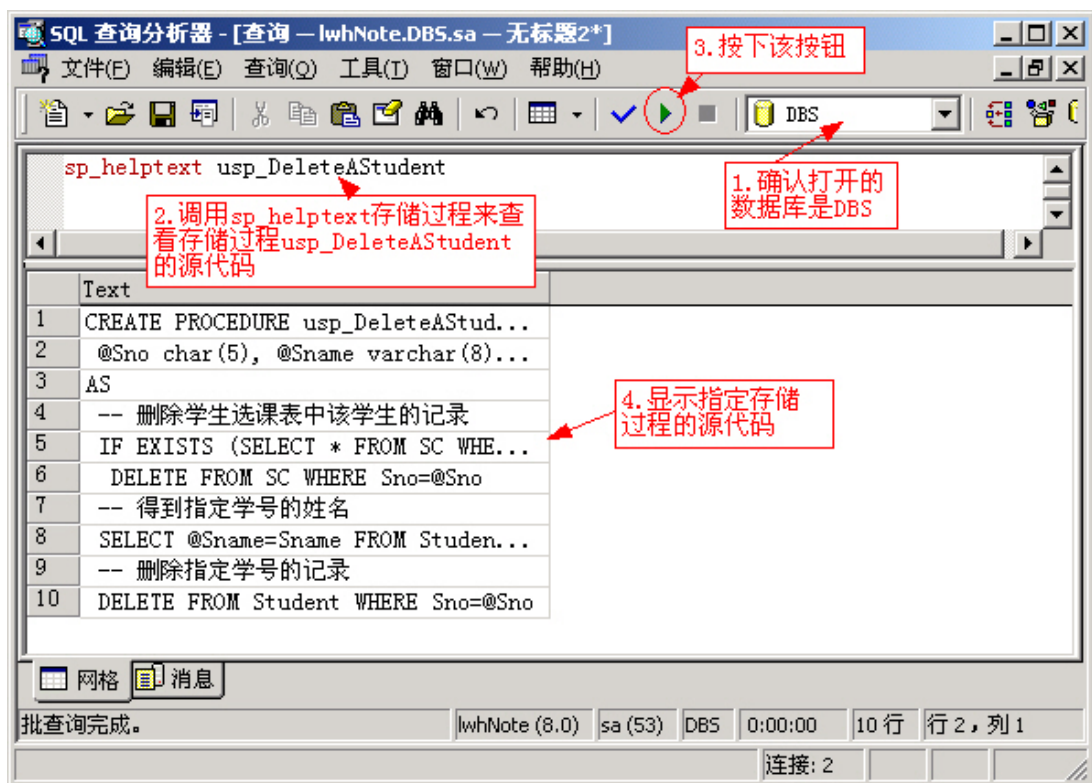
A. 通过企业管理器查询存储过程



若要查看源代码，双击存储过程文件即可，如下图所示 `usp_DeleteAStudent` 存储过程的源代码



B. 通过查询分析器查询存储过程

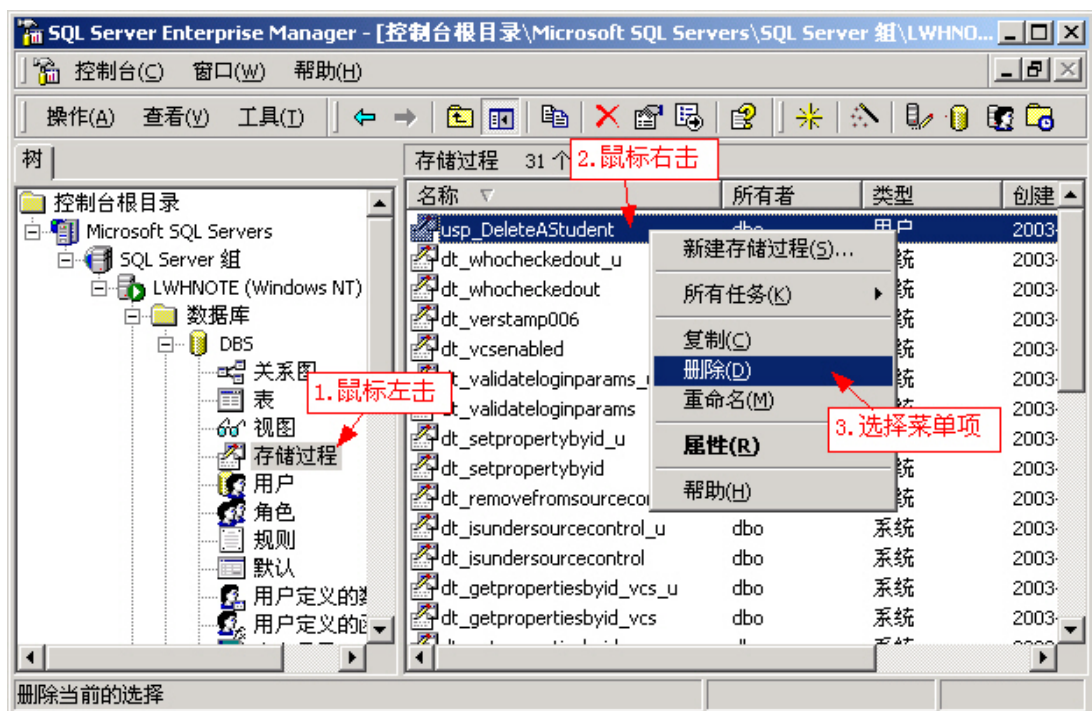


注：上面演示了系统存储过程（sp_helptext）的调用。

二、删除存储过程

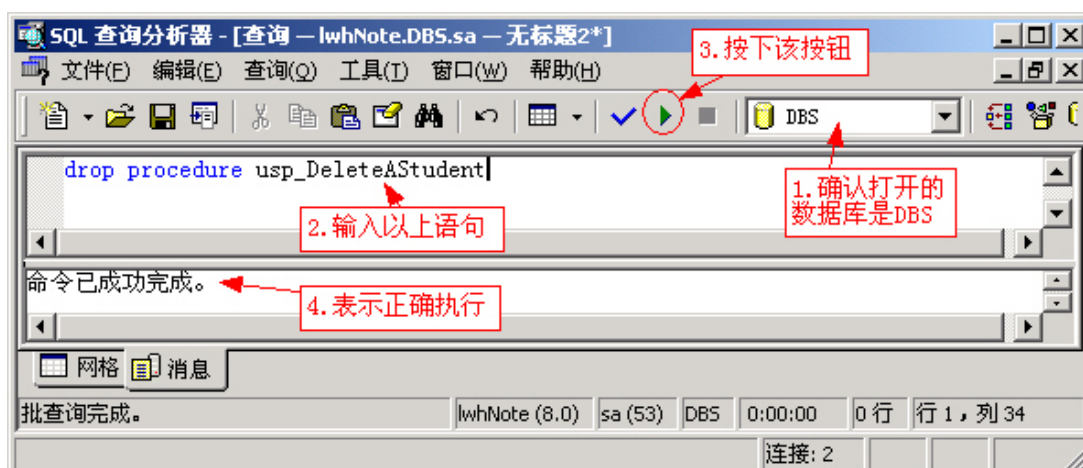
A. 通过企业管理器删除存储过程

如：删除 DBS 数据库中存储过程 usp_DeleteAStudent



B. 通过查询分析器删除存储过程

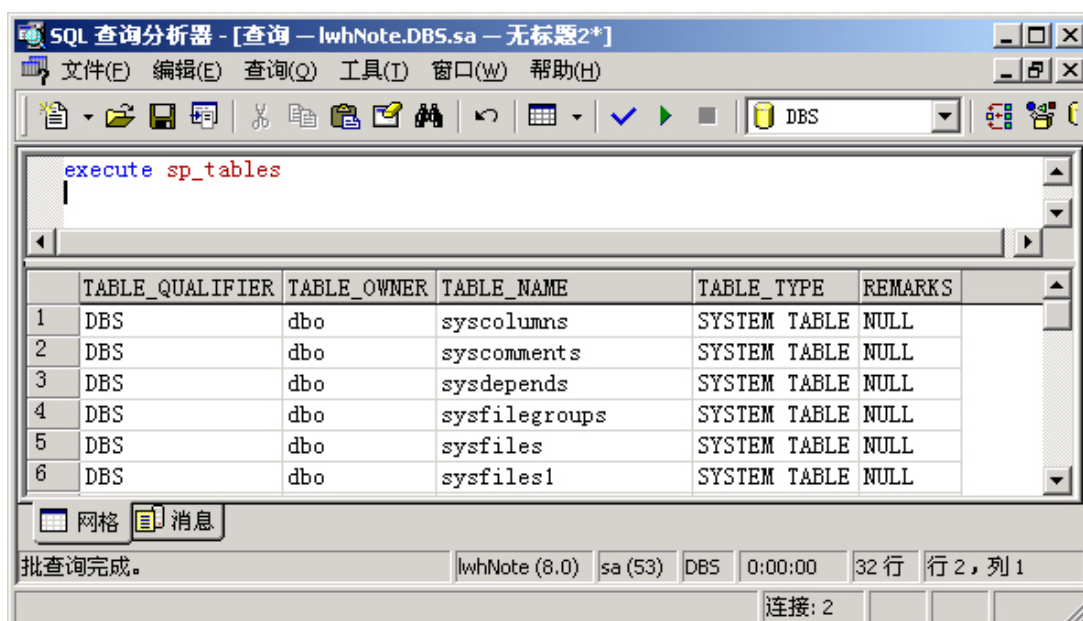
如：删除 DBS 数据库中存储过程 usp_DeleteAStudent



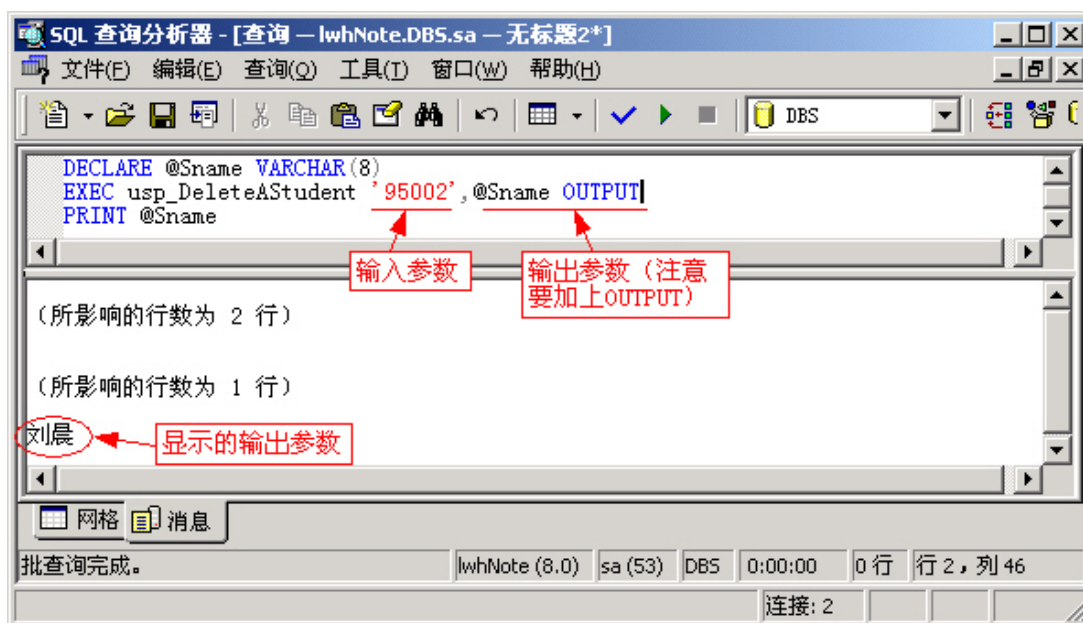
三、执行存储过程

无论是系统存储过程，还是用户自定义存储过程，均可被用户调用执行。在本实验中，我们试验在查询分析器中如何调用存储过程。此外，这些存储过程还可在编写客户端程序时，由其它语言调用执行。

例 1: 执行不带任何参数的存储过程。sp_tables 返回当前环境下可查询的对象的列表，即任何可出现在 FROM 子句中的对象。



例 2: 执行带输入参数和输出参数的存储过程。usp_DeleteAStudent 删除指定学号的学生记录，并返回该学生的姓名。



4.1.4 系统存储过程介绍

系统存储过程就是系统创建的存储过程，目的在于能够方便地从系统表中查询信息或完成与更新数据库表相关的管理任务或其它的系统管理任务。系统过程以“sp_”为开头，在 Master 数据库中创建并保存在该数据库中，为数据库管理者所有。一些系统过程只能由系统管理员使用，而有些系统过程通过授权可以被其它用户所使用。

系统存储过程主要包括以下几类：（这里主要给出每类系统过程中经常使用的系统过程）

一、目录存储过程

sp_Server_info	返回 SQL Server、数据库网关或基础数据源的特性名和匹配值的列表
sp_databases	列出驻留在 SQL Server 实例中的数据库或可以通过数据库网关访问的数据库
sp_tables	返回当前环境下可查询的对象的列表
sp_columns	返回当前环境中可查询的指定表或视图的列信息
sp_fkeys	返回当前环境的单个表的外键信息
sp_pkeys	返回当前环境中单个表的主键信息
sp_stored_procedures	返回当前环境中的存储过程列表

二、安全管理类存储过程

sp_addlogin	创建新的 SQL Server 登录，使用户可以连接使用 SQL Server 身份验证的 SQL Server 实例。
sp_droplogin	删除 SQL Server 登录，以阻止使用该登录名访问 SQL Server
sp_helplogins	提供有关每个数据库中的登录及相关用户的信息
sp_grantlogin	使 Windows NT 用户或组帐户可以使用 Windows 身份验证连接到 SQL Server
sp_denylogin	阻止 Windows NT 用户或组连接到 SQL Server
sp_grantdbaccess	为 SQL Server 登录或 Windows NT 用户或组在当前数据库中添加一个安全帐户，并使其能够被授予在数据库中执行活动的权限
sp_revokedbaccess	从当前数据库中删除安全帐户

sp_defaultdb	更改登录的默认数据库
sp_password	添加或更改 SQL Server 登录的密码

4.2 触发器

在大型数据库系统中存储过程和触发器具有很重要的作用无论是存储过程还是触发器都是 SQL 语句和流程控制语句的集合就本质而言触发器也是一种存储过程存储过程在运算时生成执行方式所以以后对其再运行时其执行速度很快 SQL Server 2000 不仅提供了用户自定义存储过程的功能而且也提供了许多可作为工具使用的系统存储过程。

4.2.1 触发器概述

 帮助索引: <触发器>

在上面一节介绍了一般意义的存储过程，即用户自定义的存储过程和系统存储过程。本节将介绍一种特殊的存储过程，即触发器。

一、触发器的概念

触发器是一种特殊类型的存储过程，它不同于我们前面介绍过的存储过程。触发器主要是通过事件进行触发而被执行的，而存储过程可以通过存储过程名字而被直接调用。当对某一表进行诸如 UPDATE、INSERT、DELETE 这些操作时，SQL Server 就会自动执行触发器所定义的 SQL 语句，从而确保对数据的处理必须符合由这些 SQL 语句所定义的规则。

二、触发器的作用

触发器的主要作用就是其能够实现由主键和外键所不能保证的复杂的参照完整性和数据的一致性。除此之外，触发器还有其它许多不同的功能：

A. 强化约束

触发器能够实现比 CHECK 语句更为复杂的约束。

B. 跟踪变化

触发器可以侦测数据库内的操作从而不允许数据库中未经许可的指定更新和变化。

C. 级联运行

触发器可以侦测数据库内的操作，并自动地级联影响整个数据库的各项内容。例如：某个表上的触发器中包含有对另外一个表的数据操作（如删除，更新，插入），而该操作又导致该表上触发器被触发。

D. 存储过程的调用

为了响应数据库更新，触发器可以调用一个或多个存储过程，甚至可以通过外部过程的

调用而在 DBMS 本身之外进行操作。

由此可见，触发器可以解决高级形式的业务规则或复杂行为限制以及实现定制记录等一些方面的问题。例如，触发器能够找出某一表在数据修改前后状态发生的差异，并根据这种差异执行一定的处理。此外一个表的同一类型(INSERT、UPDATE、DELETE)的多个触发器能够对同一种数据操作采取多种不同的处理。

总体而言，触发器性能通常比较低。

三、触发器的种类

SQL Server 2000 支持两种类型的触发器：AFTER 触发器和 INSTEAD OF 触发器。

AFTER 触发器要求只有执行某一操作(INSERT、UPDATE、DELETE)之后，触发器才被触发，且只能在表上定义。可以为针对表的同一操作定义多个触发器。

INSTEAD OF 触发器表示并不执行其所定义的操作(INSERT、UPDATE、DELETE)，而仅是执行触发器本身。既可在表上定义 INSTEAD OF 触发器，也可以在视图上定义 INSTEAD OF 触发器，但对同一操作只能定义一个 INSTEAD OF 触发器。

注：在本实验中不讲INSTEAD OF触发器

4.2.2 触发器的原理

 帮助索引：<触发器>

SQL Server 是如何管理触发器来完成这些任务呢？下面我们将对其工作原理及实现做较为详细的介绍。

每个触发器有两个特殊的表：插入表(INSERTED)和删除表(DELETED)。这两个表是逻辑表，并且这两个表是由系统管理的，存储在内存中，不是存储在数据库中，因此不允许用户直接对其修改。这两个表的结构总是与被该触发器作用的表有相同的表结构。这两个表是动态驻留在内存中的，当触发器工作完成，这两个表也被删除。这两个表主要保存因用户操作而被影响到的原数据值或新数据值。另外这两个表是只读的，即用户不能向这两个表写入内容，但可以引用表中的数据。例如可用如下语句查看 DELETED 表中的信息：

```
select * from deleted
```

一、插入表的功能

对一个定义了插入类型触发器的表来讲，一旦对该表执行了插入操作，那么对向该表插入的所有行来说，都有一个相应的副本存放到插入表中。即插入表就是用来存储向原表插入的内容。

二、删除表的功能

对一个定义了删除类型触发器的表来讲，一旦对该表执行了删除操作，则所有的删除行存放至删除表中。这样做的目的是，一旦触发器遇到了强迫它中止的语句被执行时，删除

的那些行可以从删除表中得以恢复。

需要强调的是，更新操作包括两个部分，即先将更新的内容去掉，然后将新值插入。因此对一个定义了更新类型触发器的表来讲，当报告会更新操作时，在删除表中存放了旧值，然后在插入表中存放新值。

由于触发器仅当被定义的操作被执行时才被激活，即仅当在执行插入、删除和更新操作时，触发器将执行。每条 SQL 语句仅能激活触发器一次，可能存在一条语句影响多条记录的情况。在这种情况下就需要变量@@rowcount 的值，该变量存储了一条 SQL 语句执行后所影响的记录数，可以使用该值对触发器的 SQL 语句执行后所影响的记录求合计值。一般来说，首先要用 IF 语句测试@@rowcount 的值以确定后面的语句是否执行。

4.2.3 创建触发器

 帮助索引: <创建触发器>

为了演示触发器的功能，下面再引入一个简易仓库管理的例子。在采购配件前，必须首先制定采购计划（如采购计划表 PlanA），然后交由采购员采购，采购回来后，要将配件入库，入库时，除了要修改入库表（配件入库表 InStock）外，还要修改配件库存表（仓库库存表 Stock），还要修改采购计划表（采购计划表 PlanA），因为要修改实际完成的采购量（FinishQty）。三个数据表如下：

采购计划表 PlanA

列名	数据类型	长度	允许空
PlanSN	char	10	
SpareCode	varchar	10	
PlanQty	int	4	✓
FinishQty	int	4	✓
PlanA 采购计划			

备件入库表 InStock

列名	数据类型	长度	允许空
InStockSN	char	10	
PlanSN	char	10	
DateInStock	datetime	8	✓
InStockQty	int	4	✓
InStock 备件入库			

备件库存表 Stock

列名	数据类型	长度	允许空
InStockSN	char	10	
SpareCode	varchar	10	
DateInStock	datetime	8	✓
StockQty	int	4	✓
►	Stock 备件库存		

给计划表追加如下数据

PlanSN	SpareCode	PlanQty	FinishQty
2003050101	1011	10	0
2003050102	1012	50	0
2003051001	2011	40	0
2003051002	2012	100	0
►			

一、创建触发器时应注意的问题

在创建触发器以前必须考虑到以下几个方面：

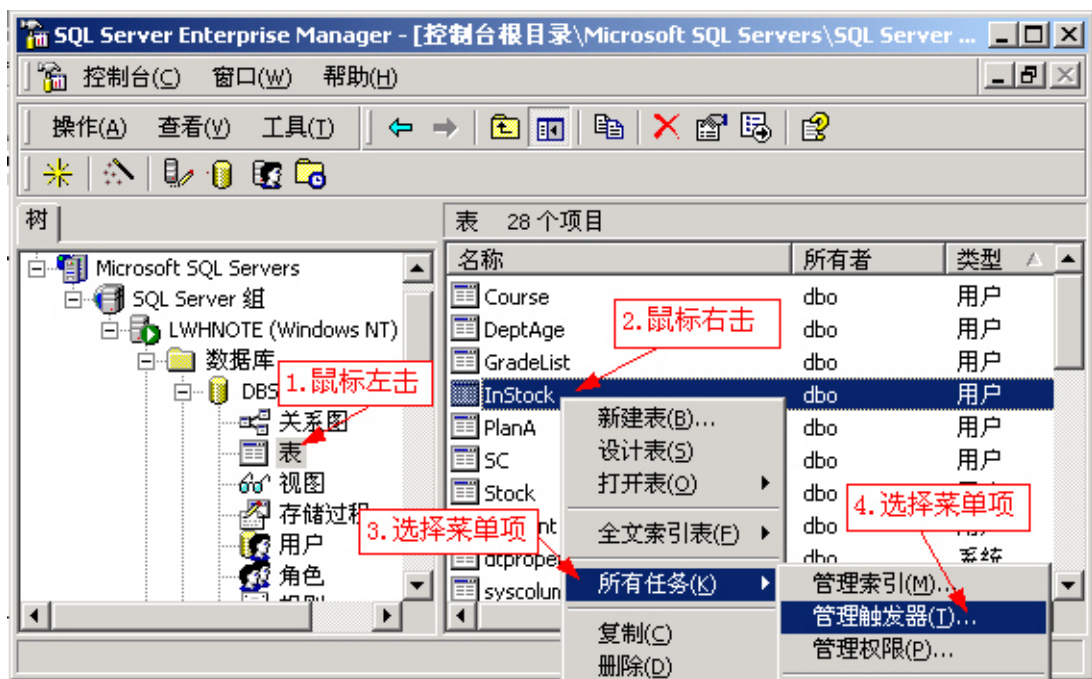
- CREATE TRIGGER 语句必须是批处理的第一个语句；
- 表的所有者具有创建触发器的缺省权限，表的所有者不能把该权限传给其它用户；
- 触发器是数据库对象，所以其命名必须符合命名规则；
- 尽管在触发器的 SQL 语句中可以参照其它数据库中的对象，但是触发器只能创建在当前数据库中；
- 虽然触发器可以参照视图或临时表，但不能在视图或临时表上创建触发器，而只能在基表或在创建视图的表上创建触发器；
- 一个触发器只能对应一个表，这是由触发器的机制决定的；

当创建一个触发器时，必须指定触发器的名字，在哪一个表上定义触发器，激活触发器的修改语句，如 INSERT、DELETE、UPDATE。当然两个或三个不同的修改语句也可以都触发同一个触发器，如 INSERT 和 UPDATE 语句都能激活同一个触发器。

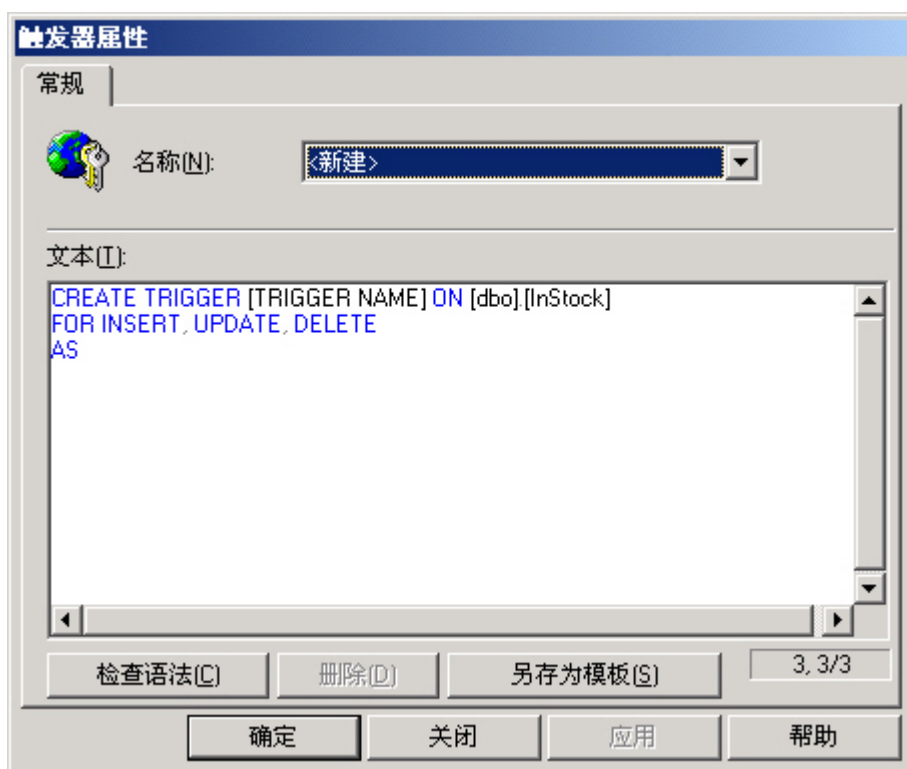
二、用企业管理器创建触发器

如：对备件入库表建立一个插入触发器 utr_InStockIns，其功能为：备件入库时，除了要将入库数据追加到备件入库表(InStock)外，还要修改计划表(PlanA)中对应计划号的完成数量(增加 FinishQty)和备件库存表(Stock)中对应备件代码的库存数量(StockQty)。

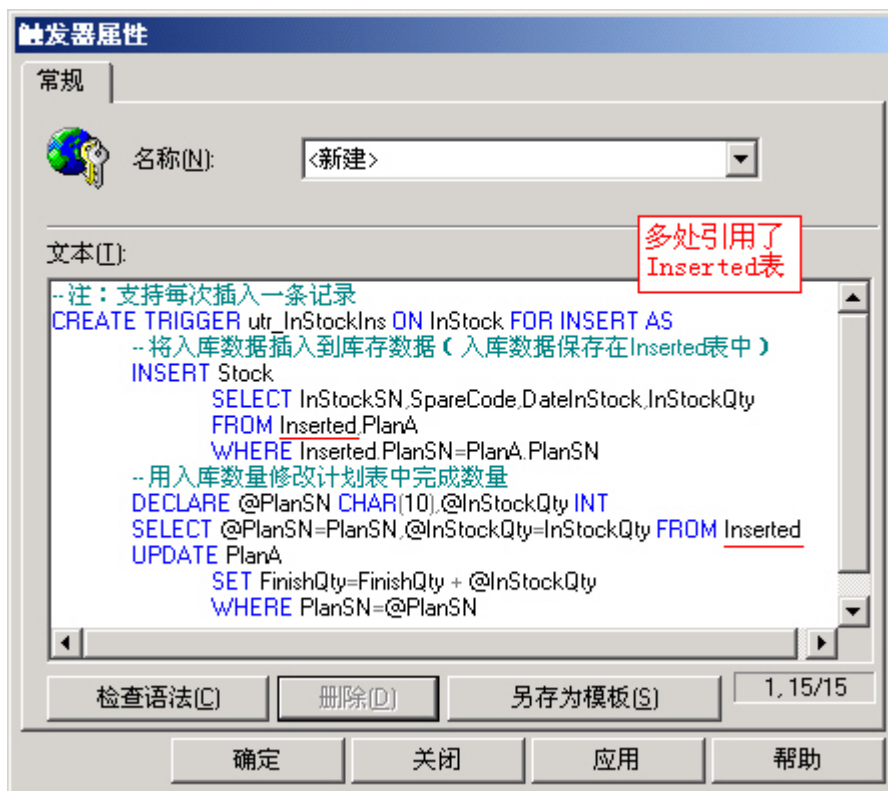
A. 如下图，针对数据表 InStock 弹出管理触发器对话框



B. 下图弹出了缺省的“触发器属性”对话框



C. 修改触发器属性对话框中的文本，为数据表 InStock 的插入动作 Insert 定义一个触发器。



上面的代码说明如下：

-- 注：支持每次插入一条记录

CREATE TRIGGER utr_InStockIns ON InStock FOR INSERT AS

-- 将入库数据插入到库存数据（入库数据保存在 Inserted 表中）

INSERT Stock

SELECT InStockSN, SpareCode, DateInStock, InStockQty

FROM Inserted, PlanA

WHERE Inserted.PlanSN=PlanA.PlanSN

-- 用入库数量修改计划表中完成数量

DECLARE @PlanSN CHAR(10), @InStockQty INT

SELECT @PlanSN=PlanSN, @InStockQty=InStockQty FROM Inserted

UPDATE PlanA

SET FinishQty=FinishQty + @InStockQty

WHERE PlanSN=@PlanSN

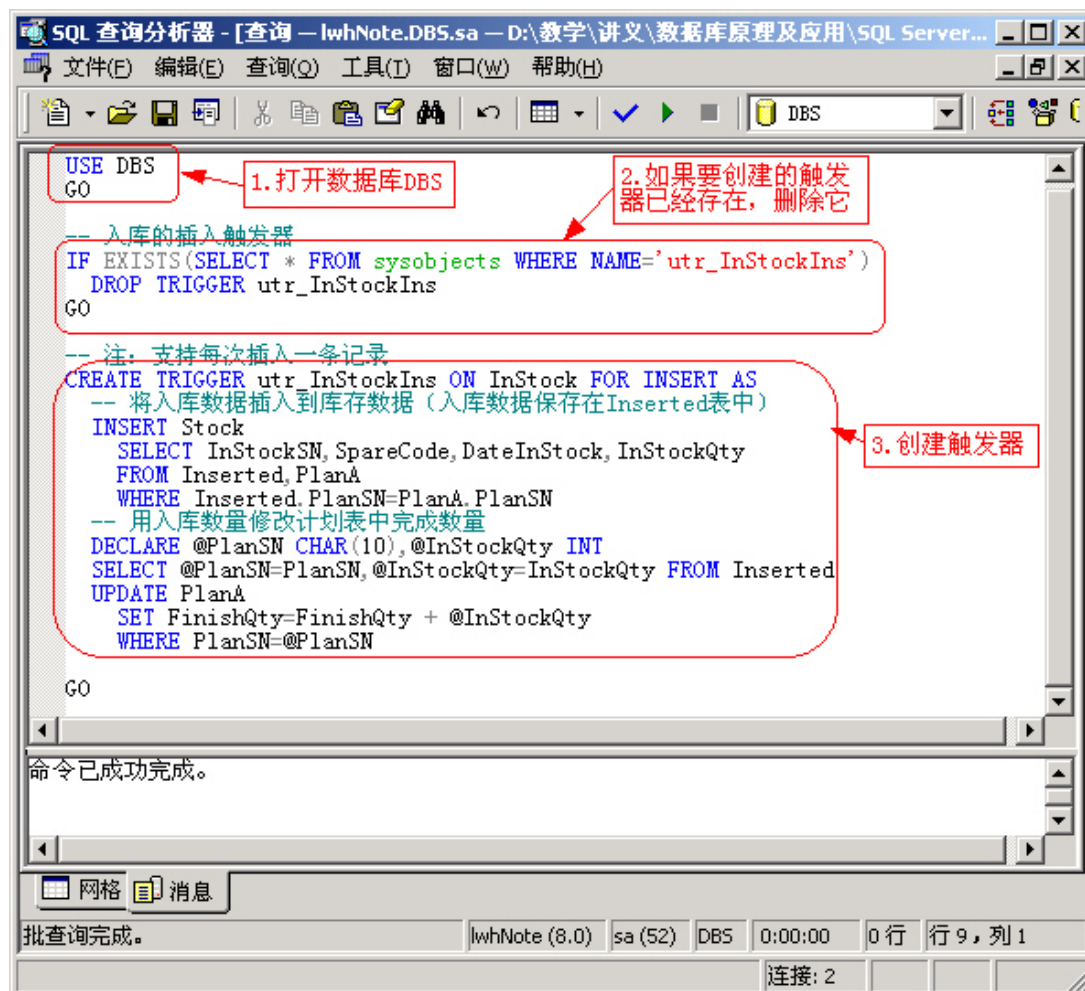
D. 语法检查正确后，按“确定”按钮即可。

三、用查询分析器创建触发器

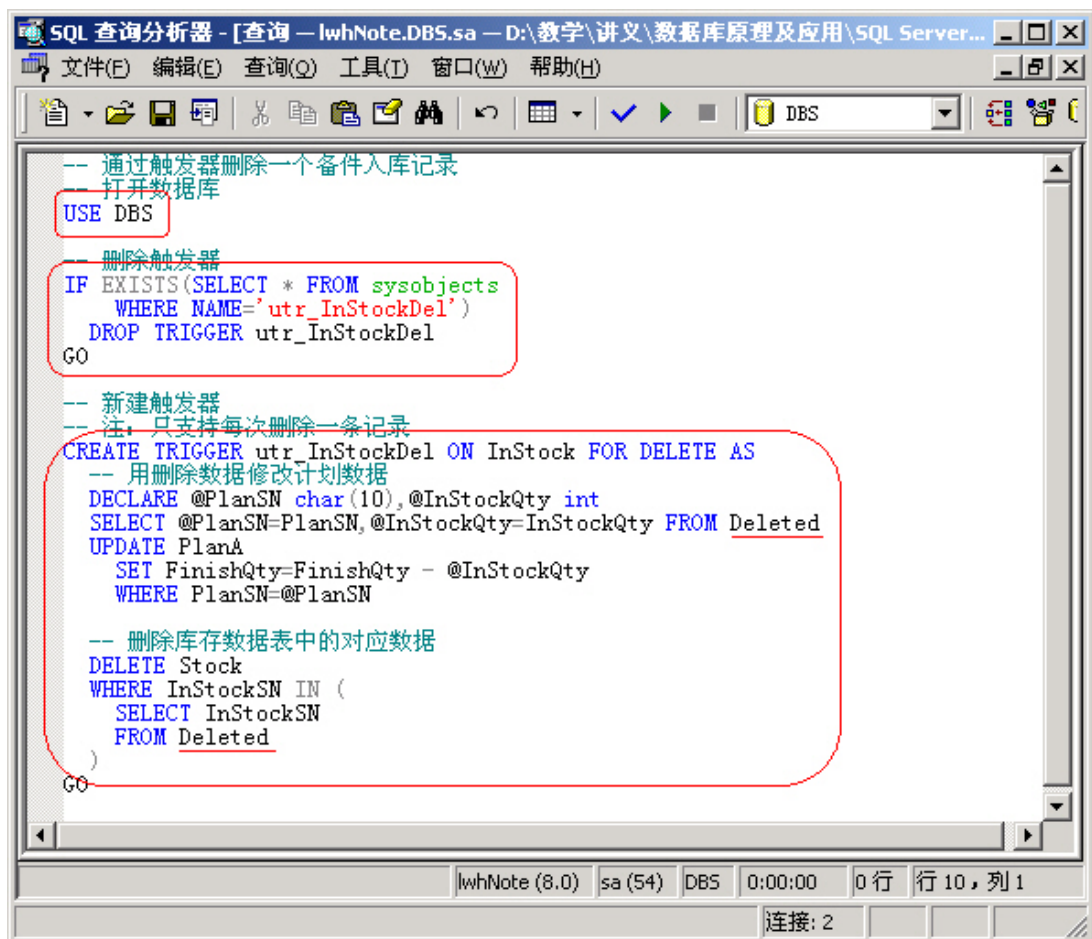
例 1：对备件入库表建立一个插入触发器 utr_InStockIns，其功能为：备件入库时，除了要将入库数据追加到备件入库表(InStock)外，还要修改计划表(PlanA)中对应计划号的完成数量(增加 FinishQty)和备件库存表(Stock)中对应备件代码的库存数量

(StockQty)。

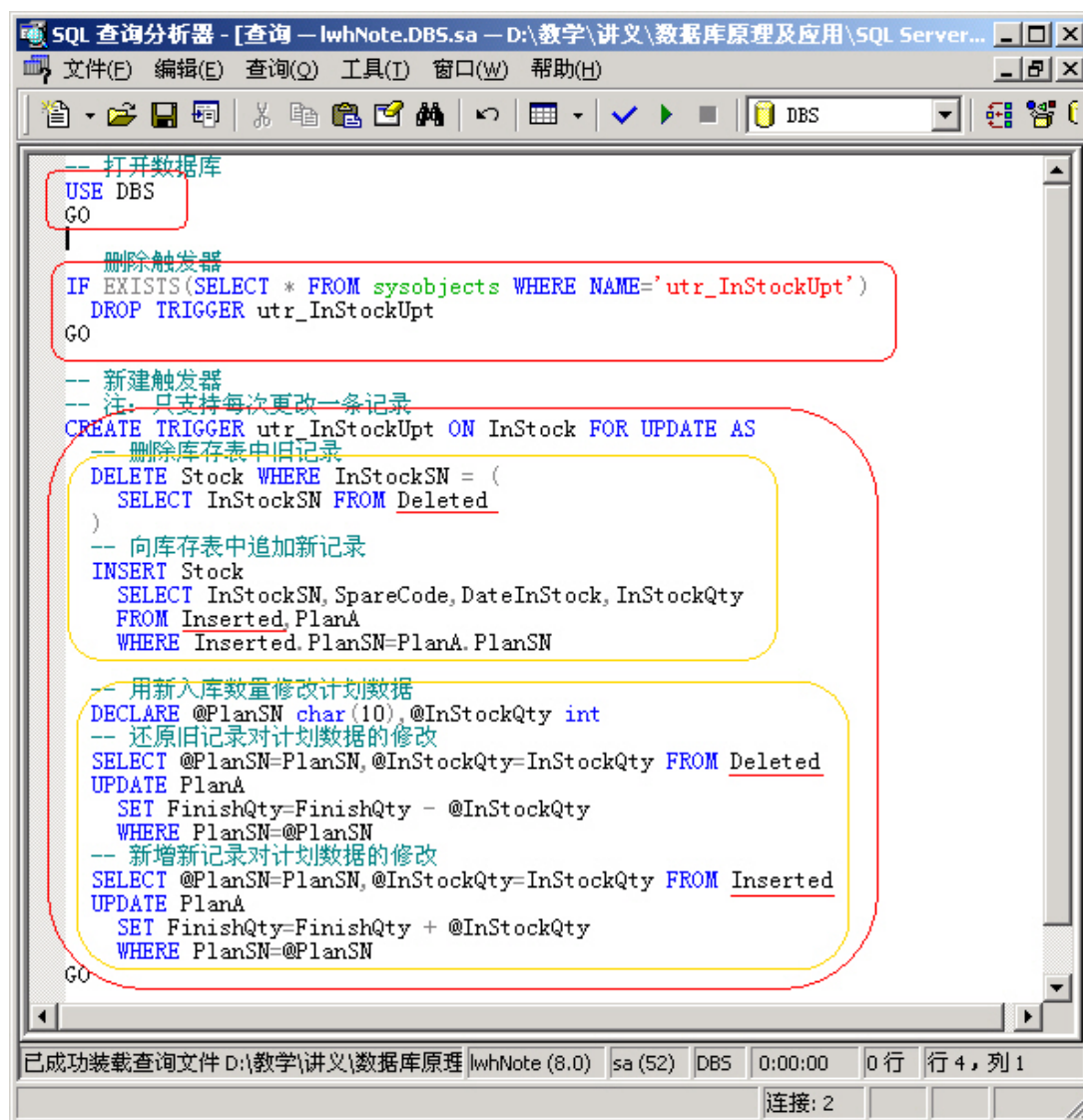
注：同企业管理器创建的触发器



例 2：对备件入库表建立一个删除触发器 utr_InStockDel，其功能为：删除备件入库中的记录时，除了要删除入库表(InStock)的记录外，还要修改计划表(PlanA)中对应计划号的完成数量(减少 FinishQty)，还要删除备件库存表(Stock)中对应备件代码的记录。



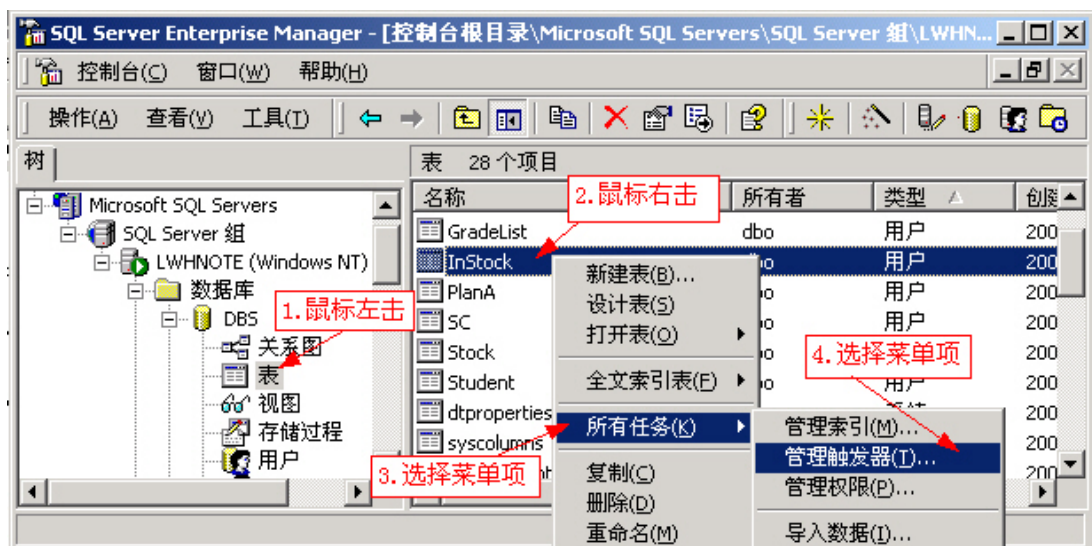
例 3：对备件入库表建立一个更改除触发器 utr_InStockUpt，其功能为：更改备件入库中的记录时，除了要更改入库表(InStock)的记录外，还要更改备件库存表(Stock)中对应备件代码的数据，同时，还要更改计划表(PlanA)中对应计划号的完成数量(FinishQty)。



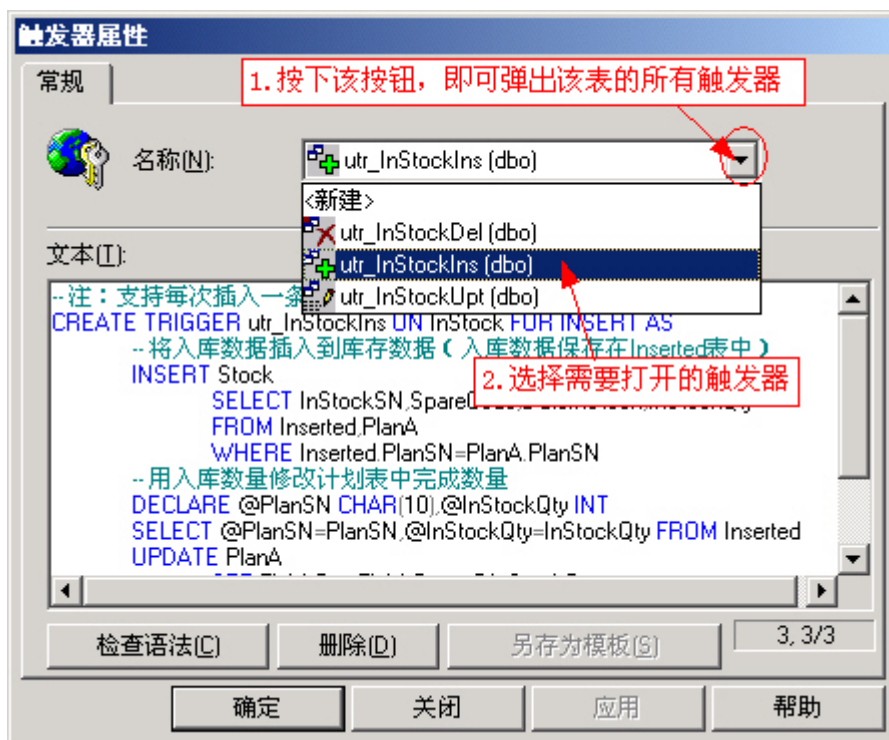
4.2.4 管理触发器

一、使用企业管理器管理触发器

如：查看数据表 InStock 有哪些触发器



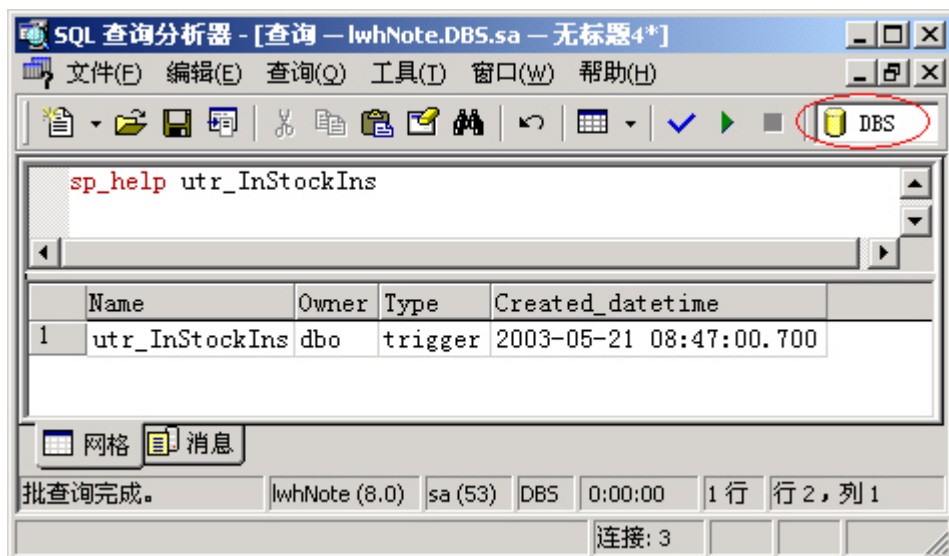
在下面的触发器属性对话框中，可以查看和修改该表的所有触发器



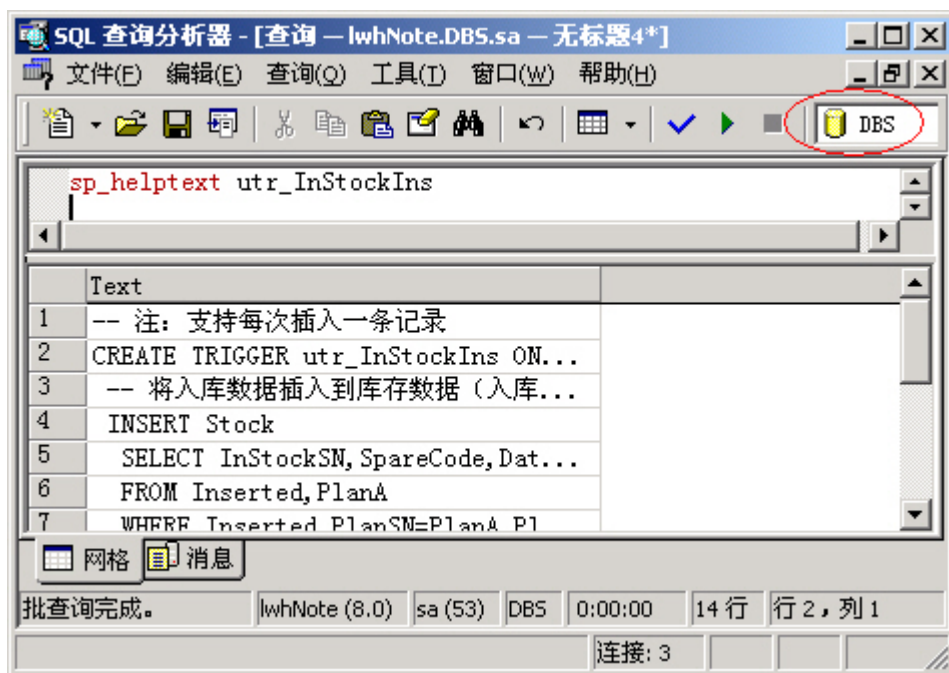
若要删除当前选定的触发器，按上图中“删除”按钮即可。

二、使用查询分析器管理触发器

A. 使用 sp_help 可以了解触发器的一般信息

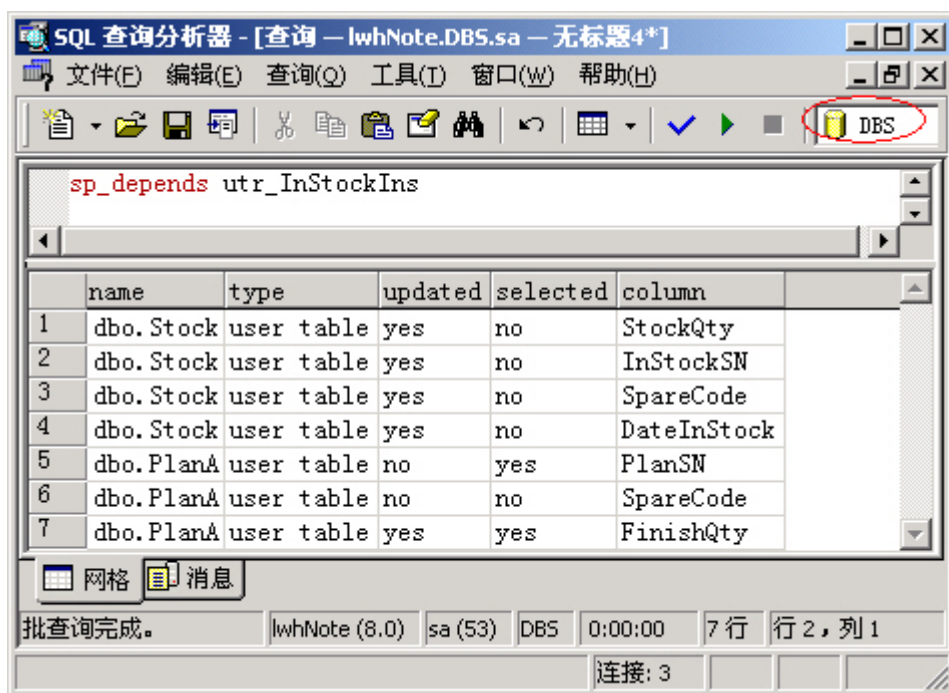


B. 使用 sp_helptext 可以了解触发器的正文信息

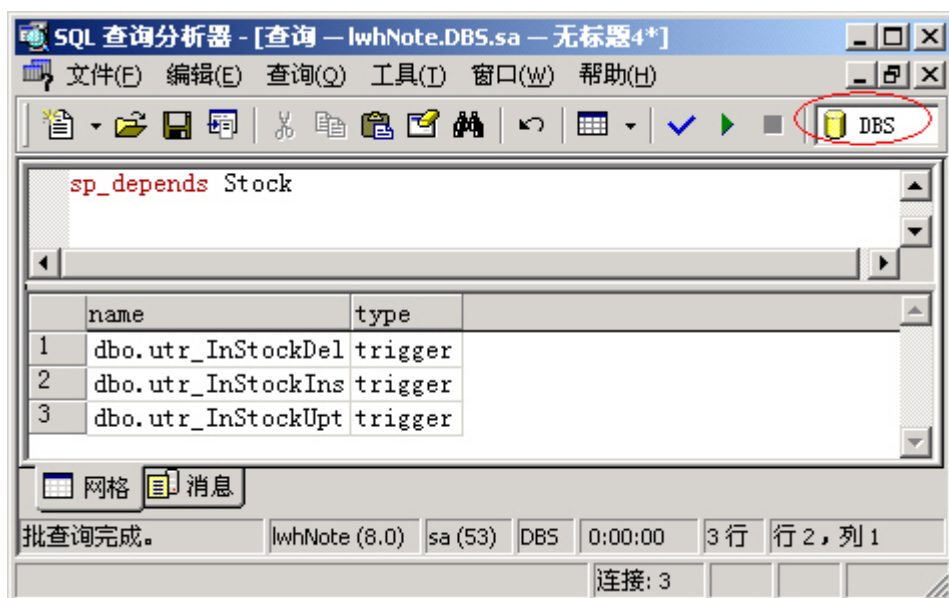


C. 使用 sp_depends 可以查看指定触发器所引用的所有表或指定表所涉及的所有触发器

下图演示了触发器 utr_InStockIns 所引用的所有表



下图演示了数据表 Stock 所涉及到的所有触发器



D. 删除触发器

格式: DROP TRIGGER 触发器名字

如: DROP TRIGGER utr_InStockIns

注: 实际工作中, 通常将建立触发器的文本保存到一个SQL文件中, 便于以后修改, 所以, 无论是在企业管理器中还是查询分析器中, 修改触发器显得没有多大意义, 往往是在SQL文件中, 首先判断是否有存在的触发器, 若存在, 则删除, 然后再创建。这样, 修改触发器实际上只需要修改创建触发器的SQL文件即可。参见[4.2.3 创建触发器](#)中“三、用查询分析器创

建触发器”。

4.2.5 触发器的应用

假设你已经按照[4.2.3 创建触发器](#)中要求创建了插入触发器utr_InStockIns、删除触发器utr_InStockDel和更改触发器utr_InStockUpt，才能做下面的实验。

可以参见[4.2.4 管理触发器](#)来检查是否已经创建。

一、插入型触发器的应用

假设采购计划表(PlanA)中的数据如下图，而备件入库表(InStock)和备件库存表(Stock)中没有数据。

	PlanSN	SpareCode	PlanQty	FinishQty
	2003050101	1011	10	0
	2003050102	1012	50	0
	2003051001	2011	40	0
	2003051002	2012	100	0
▶				

现在打开备件入库表(InStock)，向该表中追加 1 条记录，如下

	InStockSN	PlanSN	DateInStock	InStockQty
	2003051501	2003050101	2003-5-15	5
▶				

然后打开计划表和备件库存表，查看是否发生了变化？

	PlanSN	SpareCode	PlanQty	FinishQty
	2003050101	1011	10	5
	2003050102	1012	50	0
	2003051001	2011	40	0
	2003051002	2012	100	0
▶				

采购计划表中第1个计划的完成数量由0变成了5

	InStockSN	SpareCode	DateInStock	StockQty
	2003051501	1011	2003-5-15	5
▶				

备件库存表中增加了1条记录

按照上面的方法，再向备件入库表(InStock)中追加几条记录，如下图

	InStockSN	PlanSN	DateInStock	InStockQty
	2003051501	2003050101	2003-5-15	5
	2003051502	2003050101	2003-5-15	3
	2003051503	2003050102	2003-5-15	10
▶				

采购计划表中(PlanA)的数据变化如下

	PlanSN	SpareCode	PlanQty	FinishQty
	2003050101	1011	10	8
	2003050102	1012	50	10
	2003051001	2011	40	0
	2003051002	2012	100	0
▶				

备件库存表(Stock)中的数据变化如下

	InStockSN	SpareCode	DateInStock	StockQty
	2003051501	1011	2003-5-15	5
	2003051502	1011	2003-5-15	3
	2003051503	1012	2003-5-15	10
▶				

注：结合上节建立的插入触发器，理解插入触发器的工作原理。

二、删除触发器的应用

假设你已经完成了“插入触发器的应用”中的例子。

打开备件入库表，删除下图中指示的记录

	InStockSN	PlanSN	DateInStock	InStockQty
	2003051501	2003050101	2003-5-15	5
	2003051502	2003050101	2003-5-15	3
	2003051503	2003050102	2003-5-15	10
▶				

鼠标右击，弹出菜单后，选择“删除”

备件入库表(InStock)中的数据变化如下

	InStockSN	PlanSN	DateInStock	InStockQty
	2003051501	2003050101	2003-5-15	5
	2003051503	2003050102	2003-5-15	10
▶				

入库编号为2003051502的记录已删除

采购计划表(PlanA)中的数据变化如下

	PlanSN	SpareCode	PlanQty	FinishQty
	2003050101	1011	10	5
	2003050102	1012	50	10
	2003051001	2011	40	0
	2003051002	2012	100	0
▶				

8变成了5

备件库存表(Stock)中的数据变化如下

	InStockSN	SpareCode	DateInStock	StockQty
	2003051501	1011	2003-5-15	5
	2003051503	1012	2003-5-15	10
▶				

入库编号为2003051502的记录已删除

注：结合上节建立的删除触发器，理解删除触发器的工作原理。

三、更改触发器的应用

假设你已经完成了“插入触发器的应用”和“删除触发器的应用”中的例子。
打开备件入库表，更改下图中指示的记录

	InStockSN	PlanSN	DateInStock	InStockQty
	2003051501	2003050101	2003-5-15	5
	2003051503	2003050102	2003-5-15	10
*				

将入库编号为2003051503的记录的入库数量由10变成5

备件入库表(InStock)中的数据变化如下

	InStockSN	PlanSN	DateInStock	InStockQty
	2003051501	2003050101	2003-5-15	5
	2003051503	2003050102	2003-5-15	5

入库编号为2003051503的记录的入库数量由10变成了5

采购计划表(PlanA)中的数据变化如下

	PlanSN	SpareCode	PlanQty	FinishQty
	2003050101	1011	10	5
	2003050102	1012	50	5
	2003051001	2011	40	0
	2003051002	2012	100	0

计划号为2003050102的记录的完成数量由10变成了5

备件库存表(Stock)中的数据变化如下

	InStockSN	SpareCode	DateInStock	StockQty
	2003051501	1011	2003-5-15	5
	2003051503	1012	2003-5-15	5

入库编号为2003051503的记录的库存数量由10变成了5

注：结合上节建立的更改触发器，理解更改触发器的工作原理。

注：由于我们直接在企业管理器中修改（插入、删除、更改）数据，每次修改一条记录后，

当光标移到其它记录上或按“”执行后，系统就自动地调用了各自的触发器，且每次只有1条记录修改的记录。所以，在企业管理器中，无法测试一次修改多条记录的情况。

想一想：如何一次修改多条记录？

4.3 游标

帮助目录：<访问和更改关系数据|游标>

本节可以了解游标的优点、种类、作用，学会如何定义、打开、存取、关闭、释放游标以及游标的应用。

4.3.1 游标概述



一、游标的概念及其特点

在数据库开发过程中, 当我们需要从某一结果集中逐一地读取每条记录时, 如何解决这种问题呢? 游标为我们提供了一种极为优秀的解决方案。

游标实际上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, 游标总是与一条 **Transact-SQL** 选择语句相关联, 因为游标由结果集 (可以是零条、一条或由相关的选择语句检索出的多条记录) 和结果集中指向特定记录的游标位置组成。

当决定对结果集进行处理时, 必须声明一个指向该结果集的游标。如果曾经用 **C** 语言写过对文件进行处理的程序, 那么游标就像您打开文件所得到的文件句柄一样, 只要文件打开成功, 该文件句柄就可代表该文件。对于游标而言, 其道理是相同的。可见游标能够实现按与传统程序读取平面文件类似的方式处理来自基础表的结果集, 从而把表中数据以平面文件的形式呈现给程序。

我们知道关系数据库管理系统实质是面向集合的, 在 **SQL SERVER** 中并没有一种描述表中单一记录的表达形式, 除非使用 **where** 子句来限制只有一条记录被选中。因此我们必须借助于游标来进行面向单条记录的数据处理。

由此可见, 游标允许应用程序对查询语句 **select** 返回的行结果集中每一行进行相同或不同的操作, 而不是一次对整个结果集进行同一种操作; 它还提供对基于游标位置而对表中数据进行删除或更新的能力; 而且, 正是游标把作为面向集合的数据库管理系统和面向行的程序设计两者联系起来, 使两个数据处理方式能够进行沟通。

二、游标的种类

SQL SERVER 支持三种类型的游标: **Transact-SQL** 游标, **API** 服务器游标和客户游标。

A. Transact-SQL 游标

Transact-SQL 游标是由 **DECLARE CURSOR** 语法定义, 主要用在 **Transact-SQL** 脚本、存储过程和触发器中。**Transact-SQL** 游标主要用在服务器上, 由从客户端发送给服务器的 **Transact-SQL** 语句或是批处理、存储过程、触发器中的 **Transact-SQL** 进行管理。**Transact-SQL** 游标不支持提取数据块或多行数据。

B. API 游标

API 游标支持在 **OLE DB**、**ODBC** 以及 **DB_library** 中使用游标函数, 主要用在服务器上。每一次客户端应用程序调用 **API** 游标函数, **SQL SEVER** 的 **OLE DB** 提供者、**ODBC** 驱

动器或 DB_library 的动态链接库(DLL)都会将这些客户请求传送给服务器以对 API 游标进行处理。

C. 客户游标

客户游标主要是当在客户机上缓存结果集时才使用。在客户游标中，有一个缺省的结果集被用来在客户机上缓存整个结果集。客户游标仅支持静态游标而非动态游标。由于客户游标并不支持所有的 Transact-SQL 语句或批处理，所以客户游标常常仅被用作服务器游标的辅助。因为在一般情况下，服务器游标能支持绝大多数的游标操作。

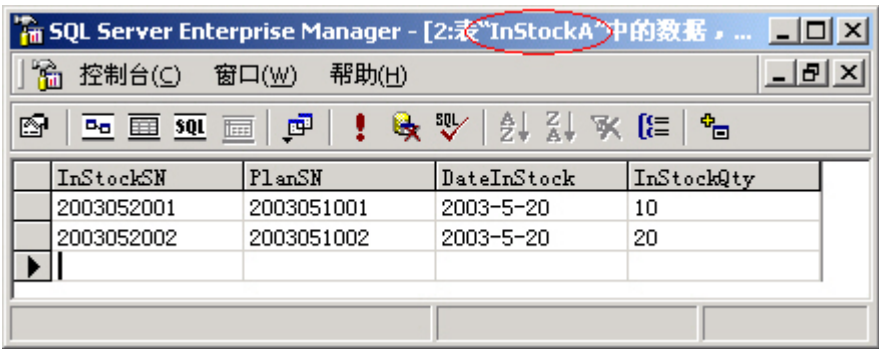
由于 Transact-SQL 游标和 API 游标使用在服务器，端所以被称为服务器游标，也被称为后台游标，而客户端游标被称为前台游标。本实验主要讲述服务器（后台）游标。

4.3.2 游标的使用

 帮助索引: <游标>

一、插入触发器 utr_InStockIns 存在的问题

如：在[4.2.3 创建触发器](#)中创建的插入触发器utr_InStockIns，每次只支持插入一条记录。假如有一个结构与InStock完全相同的数据表InStockA，该表中有二条入库记录，如下



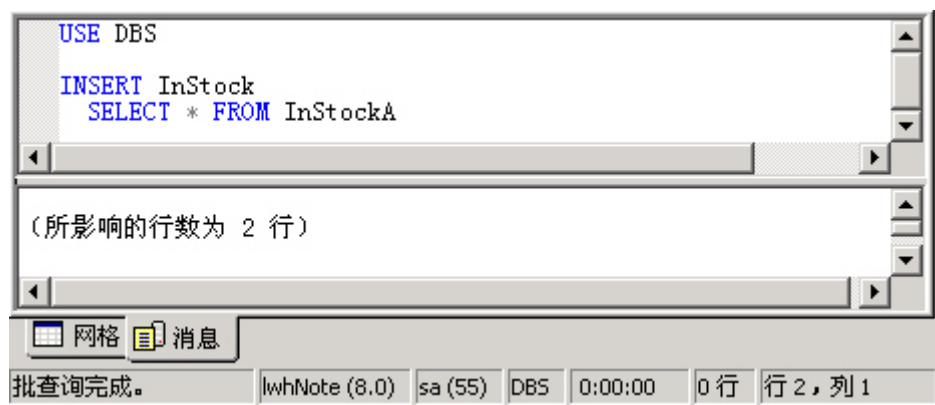
InStockSN	PlanSN	DateInStock	InStockQty
2003052001	2003051001	2003-5-20	10
2003052002	2003051002	2003-5-20	20

假设采购计划表(PlanA)中的数据如下图，而备件入库表(InStock)和备件库存表(Stock)中没有数据。

PlanSN	SpareCode	PlanQty	FinishQty
2003050101	1011	10	0
2003050102	1012	50	0
2003051001	2011	40	0
2003051002	2012	100	0

下面，在查询分析器中，一次向备件入库表(InStock)追加多条记录，其记录来自于刚

才建立的 InStockA 中，如下



查看备件入库表(InStock)，表中追加了 2 条记录，正确

InStockSN	PlanSN	DateInStock	InStockQty
2003052001	2003051001	2003-5-20	10
2003052002	2003051002	2003-5-20	20

再查看采购计划表(PlanA)，表中只修改了最后一个入库编号的完成数量，有错误

PlanSN	SpareCode	PlanQty	FinishQty
2003050101	1011	10	0
2003050102	1012	50	0
2003051001	2011	40	0
2003051002	2012	100	20

Annotations: A red box labeled "没有修改，错误" (Not modified, error) points to the '0' in the 'FinishQty' column for PlanSN 2003051001. Another red box labeled "正确" (Correct) points to the '20' in the 'FinishQty' column for PlanSN 2003051002.

查看备件库存表(Stock)，表中追加了 2 条记录，正确

InStockSN	SpareCode	DateInStock	StockQty
2003052001	2011	2003-5-20	10
2003052002	2012	2003-5-20	20

是什么原因造成上面的错误？下图是 utr_InStockIns 触发器源代码


```

-- 建立触发器
USE DBS
GO

-- 入库的插入触发器
IF EXISTS(SELECT * FROM sysobjects WHERE NAME='utr_InStockIns')
    DROP TRIGGER utr_InStockIns
GO

-- 注：支持每次插入一条记录
CREATE TRIGGER utr_InStockIns ON InStock FOR INSERT AS
-- 将入库数据插入到库存数据（入库数据保存在Inserted表中）
INSERT Stock
    SELECT InStockSN, SpareCode, DateInStock, InStockQty
    FROM Inserted, PlanA
    WHERE Inserted.PlanSN=PlanA.PlanSN
-- 用入库数量修改计划表中完成数量
DECLARE @PlanSN CHAR(10), @InStockQty INT
SELECT @PlanSN=PlanSN, @InStockQty=InStockQty FROM Inserted
UPDATE PlanA
    SET FinishQty=FinishQty + @InStockQty
    WHERE PlanSN=@PlanSN
GO

```

Inserted的记录指针移到了最后

只取了Inserted中最后一个记录的计划号和入库数量

二、游标的使用

使用每一个游标必须有四个组成部分，且这四个关键部分必须符合下面的顺序：

- DECLARE 游标
- OPEN 游标
- 从一个游标中 FETCH 信息
- CLOSE 或 DEALLOCATE 游标

如何改进上面的程序，使之可以支持一次插入多个记录？

在对采购计划表修改时，需要对备件入库的每一个记录数据（即插入表 **Inserted**）进行扫描，然后逐一修改采购计划表中的完成数量。即我们需要对上图中，紫色区域的内容进行修改，修改后的程序如下

```

-- 建立触发器
USE DBS
GO

-- 入库的插入触发器
IF EXISTS(SELECT * FROM sysobjects WHERE NAME='utr_InStockIns')
    DROP TRIGGER utr_InStockIns
GO

-- 注：支持每次插入多条记录
CREATE TRIGGER utr_InStockIns ON InStock FOR INSERT AS
-- 将入库数据插入到库存数据（入库数据保存在Inserted表中）
INSERT Stock
    SELECT InStockSN, SpareCode, DateInStock, InStockQty
    FROM Inserted, PlanA
    WHERE Inserted.PlanSN=PlanA.PlanSN
-- 用入库数量修改计划表中完成数量
DECLARE @PlanSN CHAR(10), @InStockQty INT
-- 声明游标，与插入表相关联
DECLARE curTemp CURSOR FOR
    SELECT PlanSN, InStockQty FROM Inserted
-- 打开游标
OPEN curTemp
-- 从游标中读取下一个记录，其值赋值两个变量
FETCH NEXT FROM curTemp INTO @PlanSN, @InStockQty
-- 根据全局变量@@FETCH_STATUS,
-- 判断是否从游标中取得了正确的数据值到变量@PlanSN和@InStockQty中
WHILE @@FETCH_STATUS=0
BEGIN
    UPDATE PlanA
        SET FinishQty=FinishQty + @InStockQty
        WHERE PlanSN=@PlanSN
    -- 从游标中读取下一个记录，其值赋值两个变量
    FETCH NEXT FROM curTemp INTO @PlanSN, @InStockQty
END
-- 关闭游标
CLOSE curTemp
-- 释放游标所占资源
DEALLOCATE curTemp
GO

```

对“一、插入触发器 utr_InStockIns 存在的问题”中所做的试验再做一遍，你会发现采购计划表中的数据得到了正确的修改。

注：注意观察修改后的程序中是使用使用游标的，以及使用游标的四个关键部分。

注：在上例中，我们针对 utr_InStockIns 存在的问题，使用游标得到了解决，想一想，在另外两个触发器中，是否也存在着同样的问题？如何改进？

4.4 实验内容

阅读本实验 4.1~4.3 的内容，完成下面的实验内容。

一、存储过程

在DBS数据库中，创建二个存储过程usp_CreateSpareTabs（用于创建备件管理系统中的在个数据表PlanA、InStock和Stock）和usp_InsPlanA（用于向采购计划表PlanA

追加 4 条记录），数据表的结构及计划表的数据参见[实验 4.2.3](#)。

在 DBS 数据库中，建立一个带输入参数的存储过程 `usp_InsAStudent`，其能够向数据表 `Student` 中追加一个记录，其记录内容可以由输入参数提供。

在 DBS 数据库中，建立一个带输入参数和输出参数的存储过程 `usp_GetAvgGrade`，其能够得到指定学号的学生所有课程的平均成绩。

注：上面的存储过程均在查询分析器中创建，创建后到企业管理中查询和浏览。

二、触发器

按照<4.2.3 创建触发器>中指导的内容创建备件入库表 `InStock` 的三类触发器。

按照<4.2.5 触发器的应用>中指导的内容检验备件入库表 `InStock` 的三类触发器是否能正确应用。

三、游标

按照<4.3.2 游标的使用>中指导的内容测试备件入库表一次输入多个记录时存在的问题，以及使用游标来解决这个问题。

理解游标的工作原理。

注：存储过程和触发器是以后实际数据库应用程序编写过程中非常有用的知识，望同学们认真学习和领会。

4.5 思考题

① 在进行数据库查询时，你能找出企业管理器与查询分析器在使用的差异吗？各有什么好处？

② 试比较存储过程与 C 语言中的函数有什么区别与联系？

③ 在执行存储过程时，对于无参数、有输入参数、有输入输出参数的三种情况试验过吗？

④ 你能理解触发器为什么是一类特殊的存储过程？特殊在什么地方？

⑤ 你知道如何测试各类触发器？如何测试触发器是否支持一次修改多条记录？

⑥ 是否能够总结出各类触发器的用途？

⑦ 在什么情况使用游标？

⑧ 在指导中我们只针对 `utr_InStockIns` 存在的问题，使用游标得到了解决，想一想，在另外两个触发器中，是否也存在着同样的问题？如何改进？