

CASE 函数用法

```
1  --CASE函数用法：
2  --更改值时类型需要一致，可以用cast
3  /*
4  1.case用来做结果集字段值的替换的
5  2.它可以生成一个新列
6  3.相当于switch...case 和  if..else
7  第一种使用的语法：
8  case 表达式/字段  --如果case后面有接表达式或者字段，那么这种结构只能做等值判断.不能判断
   null值
9  when 值 then 自定义值
10 when 值 then 自定义值
11 else （如果上面的when都不满足就满足else）
12 end
13 */
14 select StudentNo,StudentName,Sex,
15 case ClassId
16     when 1 then '一期班'
17     when 2 then '2期班'
18     when 3 then '3期班'
19     when '4' then '4期班'
20     else '我不知道'
21 end as 班级名称
22 from Student
23
24 /*
25 第二种使用方式：
26 -case  --如果case后面没有接表达式或者值，那么这种结构就相当于if...else可以做范围判断.
   它可以做null值判断
27 when 条件表达式 then 自定义值
28 when 条件表达式 then 自定义值
29 else 值
30 end
31 */
32 select StudentNo,StudentName,
33 case
34     when BornDate>'2000-1-1' then '小屁孩'
35     when BornDate >'1990-1-1' then '小青年'
36     when BornDate >'1980-1-1' then '大叔'
37     when BornDate is null then '不知道'
38     when Sex='男' then '我是男的'
39     else '中年'
40 end
41 from Student
42
43 --百分制转换为素质教育
44 select StudentNo,StudentResult,
45 case
46     when StudentResult>=90 then 'A'
47     when StudentResult>=80 then 'B'
48     when StudentResult>=70 then 'C'
49     when StudentResult>=60 then 'D'
50     when StudentResult IS NULL then '没有参加考试'
```

```

51 else 'E'
52 end
53 from Result

```

子查询

```

1  --子查询--在一个查询中包含着另外一个查询。被包含的查询就叫了查询，包含子查询的查询就叫父查询
2  --查询比冯晨旭年龄大的学员信息
3  select borndate from Student where StudentName='冯晨旭'
4  select * from Student where BornDate<(select borndate from Student where
   StudentName='冯晨旭')
5  --查询七期班的学员信息
6  select classid from grade where classname='七期班'
7  select * from Student where ClassId=(select classid from grade where
   classname='七期班')
8  /*
9  子查询基本分类:
10  独立子查询
11      子查询可以独立运行
12  相关子查询
13      子查询中引用了父查询中的结果
14  */
15  select * from Student where StudentNo=(select distinct studentno from
   Result where StudentNo=Student.StudentNo)
16
17  --子查询的使用:
18  /*
19  1. 做为条件: 当查询需要一个外部值做为条件的时候, 可以使用一个独立的查询先得到这个值, 再进行
   条件的判断
20  注意1: 使用子查询做为条件的时候只能返回 一个字段的值
21  select * from Student where BornDate<(select * from Student where
   StudentName='冯晨旭')
22  注意2: 子查询返回的值不止一个。当子查询跟随在 =、!=、<、<=、>、>= 之后, 或子查询用作表达
   式时, 这种情况是不允许的. 可以使用in.
23  当子查询在=、!=、<、<=、>、>=, 必须返回一个值
24  注意3: 如果返回多列值, 那么只能使用exists(not exists)。多行一列可以用in(not in)
25  */
26  select * from Student where BornDate in(select BornDate from Student)
27  if exists (select * from sysdatabases)
28
29  --做为结果集: 如果要引用, 记得一定要为子查询获取的结果集添加 别名
30  --查询七期班的男生信息
31  select * from Student where ClassId=6 and Sex='男'
32  select * from Student where ClassId=6
33  select * from (select * from Student where ClassId=6) temp where Sex='男'
34
35  --做为列的值
36  select (select studentname from student where
   StudentNo=Result.StudentNo), StudentResult from Result
37
38  --使用子查询分页
39  --第一页
40  select top 5 * from Student
41  --第二页

```

```

42 select top 5 * from Student where StudentNo not in(select top 10 Studentno
43 from Student)
44
45 --05之后, 提供一个专门用于分页的函数
46 --ROW_NUMBER() over(排序字段): 可以根据指定的字段排序, 对排序之后的结果集的每一行添加一个不间断的行号
47 --注意: 如果使用生成行号的结果集做为子查询, 那么必须为结果集命别名, 同时为行号添加名称
48 select ROW_NUMBER() over(order by studentno), * from Student
49 select * from (select ROW_NUMBER() over(order by studentno) id, * from
50 Student) temp where id>=6 and id <=10
51
52 --学好语文, 与出查询的框架: 再填空
53
54 --查询年龄比“廖杨”大的学员, 显示这些学员的信息
55 select * from Student where BornDate<(select BornDate from Student where
56 StudentName='廖杨')
57
58 --查询二期班开设的课程
59 select * from Subject where ClassId=(select ClassId from grade where
60 classname='二期班')
61
62 --查询参加最近一次“office”考试成绩最高分和最低分
63 select subjectid from Subject where SubjectName='office'
64 select MAX(examdate) from Result where SubjectId=(select subjectid from
65 Subject where SubjectName='office')
66
67 --个人感想: 如果只是求出单个值(比如另一张表里的min, max, avg, sum)用子查询比较快, 如果是和另一张表一行行比对是否相等, join会比较快
68 select MAX(StudentResult) 最高分, MIN(StudentResult) 最低分 from Result where
69 SubjectId=(
70     select subjectid from Subject where SubjectName='office'
71 ) and ExamDate=(
72     select MAX(examdate) from Result where SubjectId=
73         (select subjectid from Subject where SubjectName='office'
74         )
75 )

```

表连接JOIN

```

1  --表连接Join
2  --查询所有学生的姓名、年龄及所在班级
3  select Student.StudentName, Student.BornDate, grade.classname
4  from Student, grade
5  where Student.ClassId=grade.ClassId
6  /*
7  连接: join 需要连接的表 ...on 这两个表如果建立关联, 意味着需要指定在那一个字段上建立关联
8  内连接: inner join : 如果没有添加inner关键字, 默认就是内连接: 它可以得到两个表中建立关联字段值相等的记录
9  左连接: left join
10 右连接: right join
11交叉连接: 相当于from多表但是没有写条件 cross join
12全连接: 相当于同时左右连接 full join
13*/
14 select Student.StudentName, Student.BornDate, grade.classname
15 from Student
16     inner join grade on Student.ClassId=grade.ClassId
17 --它可以得到两个表中建立关联字段值相等的记录

```

```

18 select * from PhoneNum
19 inner join PhoneType on PhoneNum.pTypeId=PhoneType.ptId
20 --左表是指写在连接关键字前面的表。左连接可以得到左表的所有记录，如果建立关联的字段值在右表中不存在,那么右表中的字段值就是null值替代
21 select * from PhoneNum
22 left join PhoneType on PhoneNum.pTypeId=PhoneType.ptId
23 --右表是指写在连接关键字后面的表。右连接可以得到右表的所有记录，如果建立关联的字段值在左表中不存在,那么左表中的字段值就是null值替代
24 select * from PhoneNum
25 right join PhoneType on PhoneNum.pTypeId=PhoneType.ptId
26 --连接
27 select * from PhoneNum
28 cross join PhoneType
29 --全连接
30 select * from PhoneNum
31 full join PhoneType on PhoneNum.pTypeId=PhoneType.ptId
32
33 --多表连接：先写出你需要的字段，使用表。方式。再考虑它来自于那一些表 再考虑这些表如何建立关联
34 --案例2：查询年龄超过20岁的学生的姓名、年龄及所在班级
35 select
36 Student.StudentName,DATEDIFF(yyyy,student.borndate,getdate()),grade.classname
37 from Student inner join grade on Student.ClassId=grade.ClassId
38 where DATEDIFF(yyyy,student.borndate,getdate())>=20
39 --案例3：查询学生姓名、年龄、班级及成绩
40 select
41 Student.StudentName,DATEDIFF(yyyy,student.borndate,getdate()),grade.classname,Result.StudentResult
42 from Student
43 inner join grade on Student.ClassId=grade.ClassId
44 inner join Result on Student.StudentNo=Result.StudentNo
45 --案例4：查询所有学生(参加及未参加考试的都算)及成绩
46 --案例5：请查询出所有没有参加考试（在成绩表中不存在的学生）的学生的姓名。
47 select Student.StudentName,Student.StudentNo
48 from Student left join Result on Student.StudentNo=Result.StudentNo
49 where Result.StudentNo is null
50 --练习3：查询所有学生(参加和未参加考试)的学生姓名、年龄、成绩，如果没有参加考试显示缺考,如果小于60分显示不及格
51 select Student.StudentName,DATEDIFF(YYYY,Student.BornDate,GETDATE()),
52 case
53     when Result.StudentResult<60 then '不及格'
54     when Result.StudentResult is null then '缺考'
55     else cast(Result.StudentResult as CHAR(3))
56 end as 考试成绩
57 from Student
58 left join Result on Student.StudentNo=Result.StudentNo

```

视图

- 视图是一张虚拟表，它表示一张表的部分数据或多张表的综合数据，其结构和数据是建立在对表的查询基础上
- 视图在操作上和数据库表没有什么区别，但两者的差异是其本质是不同:数据库表是实际存储记录的地方，然而视图并不保存任何记录。

- 相同的数据表，根据不同用户的不同需求，可以创建不同的视图（不同的查询语句）
- 视图的目的是方便查询，所以一般情况下不对视图进行增改，不能删
- 优点：
 - 筛选表中的行
 - 防止未经许可的用户访问敏感数据
 - 降低数据库的复杂程度

使用视图注意事项：

- 视图中的查询不能使用 `order by`，除非指定了 `top` 语句。
- 视图被认为是一个虚拟表，表是一个集合，是不能有顺序的。而 `order by` 则返回的是一个有顺序的，是一个游标。
- 在视图中使用 `select top percent + order by` 问题。

```
1 create view vw_tab
2 as
3 select top 100 percent * from tab order by col1 desc
```

- 以上语句执行完毕以后查询 `select * from vw_tab` 也不会排序
- 与 `select * from vw_tab order by col1 desc` 不一样
- `top` 已经取得了上次 `order by` 的结果前几个结果
- 并且 `top` 输出的结果没有再次排序
- 所以无法保证输出的结果是 `desc` 排序后的结果
- 据说换成99.999%就可以排序
- 如果指定列名，则列名必须唯一（使用*不考虑）
- `create view vw_name as` 后不能跟 `begin end`.

```
1  --视图
2  --视图就是一张虚拟表，根据用户的查询创建的命令。使用视图就像使用表一样
3  select * from vw_getinfo
4  /*
5  使用代码创建视图
6  create view vw_视图名称
7  as
8      你需要查询的命令
9  go
10 */
11 if exists(select * from sysobjects where name='vw_getStudentBySex')
12     drop view vw_getStudentBySex
13 go
14 create view vw_getStudentBySex --这个创建语句必须是批处理的第一句
15 as
16 --视图里面不能使用order by ,除非你同时使用了top
17     select top 99.99999 percent * from Student order by StudentName
18     --select * from grade 视图中只能创建一个select查询
19     --update grade --不能在视图中创建update delete 和insert
20 go
21 --使用视图
22 --对视图一般不去执行增加删除和修改操作，但是一定需要注意的是，对视图的这些操作会直接影响原始的物理表.但是这些操作仅仅局限于单个表的操作，如果操作涉及到多个表就不会成功，比如主键外键
23 select * from vw_getStudentBySex
```

```
24 update vw_getStudentBySex set phone='123456' where Studentno=1
25 update vw_getinfo set address='广州传智',studentresult=95 where Studentno=3
26 delete from vw_getStudentBySex where Studentno=3
```

变量

变量分为：

- 局部变量：
 - 局部变量必须以标记@作为前缀，如 @Age int
 - 局部变量：先声明，再赋值
- 全局变量（系统变量）：
 - 全局变量必须以标记@@作为前缀，如 @@version
 - 全局变量由系统定义和维护，我们只能读取，不能修改全局变量的值

局部变量

声明局部变量

- DECLARE @变量名 数据类型
- DECLARE @name varchar(20)
- DECLARE @id int

赋值

- SET @变量名 = 值 --set用于普通的赋值
- SELECT @变量名 = 值 --用于从表中查询数据并赋值，， 可以一次给多个变量赋值

例如：

- SET @name='张三'
- SET @id = 1
- SELECT @name = sName FROM student WHERE sid=@id

输出变量的值

- SELECT 以表格的方式输出,可以同时输出多个变量
- PRINT 以文本的方式输出，一次只能输出一个变量的值
 - SELECT @name,@id
 - PRINT @name
 - PRINT @id
 - print @name,@id --错误！！

全局变量

@@ERROR：最后一个T-SQL错误的错误号

@@IDENTITY：最后一次插入的标识值

@@LANGUAGE：当前使用的语言的名称

@@MAX_CONNECTIONS：可以创建的同时连接的最大数目

@@ROWCOUNT：受上一个SQL语句影响的行数

@@SERVERNAME：本地服务器的名称

@@TRANSCOUNT: 当前连接打开的事务数

@@VERSION: SQL Server的版本信息

@@PROCID

- ERROR_MESSAGE()- 返回错误的实际信息。
- ERROR_LINE()- 返回错误所在行的行号。
- ERROR_NUMBER() - 返回错误号。
- ERROR_SEVERITY() - 返回错误严重级别。
- ERROR_STATE() - 返回错误状态。
- ERROR_PROCEDURE() - 返回错误所在的存储过程或触发器的名称。

```
1  --局部变量
2  语法:
3  --declare @变量的名称 变量的类型[=默认值]
4  declare @name nvarchar(50)='asdada' --varchar默认值为1
5  print @name+'123' --print不能做自动转换
6  select @name,1,'asdasd',32423
7  go
8  declare @name nvarchar(50)
9  --赋值 set select:如果后面是完整的子查询那么两个没有任何的区别
10 set @name='fadsfas'
11 select @name='ereterte'
12 --查询比刘健大的学员信息
13 go
14 declare @time datetime
15 select @time=(select borndate from Student where StudentName='刘健')
16 set @time=(select borndate from Student where StudentName='刘健')
17 select * from Student where BornDate<@time
18 --两种赋值方式的区别:
19 --1.set一次只能为一个变量赋值,而select支持一次为多个变量赋值
20 go
21 declare @name nvarchar(10),@age int=20
22 --set @name='aa',@age=30 --报错
23 select @name='aa',@age=30
24 --2.如果=号右边是不完整sql语句,那么当语句返回多行一列值的时候,select会得到最后一个值
25 go
26 declare @name nvarchar(10),@age int=20
27 --当没有用 EXISTS 引入子查询时,在选择列表中只能指定一个表达式
28 --子查询返回的值不止一个。当子查询跟随在 =、!=、<、<=、>、>= 之后,或子查询用作表达式
   时,这种情况是不允许
29 --set @name=(select StudentName from Student) --不允许
30 --select @name=(select StudentName from Student) --不允许
31 --set @name=StudentName from Student --报错,set后面只能使用独立子查询
32 select @name=StudentName from Student
33 print @name --显示最后一个值
34 declare @id int
35 set @id = 5
36 select @name = StudentName from Student where StudentNo=@id
37 print @name --报正常结果
38 --如果=号右边的查询没有返回任何值,那么select会保留原始拥有的默认值
39 go
40 go
41 declare @name nvarchar(10)='aaaaa',@age int=20
42 --set @name=(select Studentname from Student where StudentNo=100)
```

```

43 --print @name
44 --print 'asfasd'
45 select @name=Studentname from Student where StudentNo=100
46 print @name
47 print 'asfasd'
48
49 --查询参加最近一次“office”考试成绩最高分和最低分
50 go
51 declare @subjectName nvarchar(50)='office' --科目名称
52 declare @subjectId int --科目ID
53 set @subjectId=(select SubjectId from Subject where
SubjectName=@subjectName) --获取科目ID
54 declare @time datetime --最近一次考试时间
55 select @time=MAX(ExamDate) from Result where SubjectId=@subjectId --获取这一
科目最近一次考试日期
56 select MAX(StudentResult),MIN(StudentResult) from Result where
SubjectId=@subjectId and ExamDate=@time
57
58 select * from Student
59 select @@ROWCOUN --语法错误
60 print @@error
61 update Result set StudentResult-=1
62 print @@error
63
64 select @@SERVERNAME
65 select @@servicename
66

```

IF...ELSE...

```

1  ----IF ELSE
2  --没有bool值，只有条件表达式
3  --没有{}, 只能begin..end
4  --可以多重，可以嵌套
5  --如果包含的语句只有一句，那么也可不使用begin..and
6  --if或者else里面必须有处理语句，如没有就报错
7  if(1=1)
8      begin
9          print 'asdfas'
10         print '21312423'
11     end
12
13 --计算office科目平均分数并输出，如果平均分数超过60分输出成绩最高的三个学生的成绩，否则输出后三名的学生
14 go
15 declare @subjectName nvarchar(50)='office' --科目名称
16 declare @subjectId int =(select subjectId from Subject where
SubjectName=@subjectName) --科目ID
17 declare @avg int --平均分
18 set @avg=(select AVG(StudentResult) from Result where SubjectId=@subjectId
and StudentResult is not null)
19 if(@avg>=60)
20     begin
21         print '考得不错，输出前三名'

```



```

22     select top 3 * from Result where SubjectId=@subjectId order by
StudentResult desc
23     end
24 else
25     begin
26         print '考得不好, 输出后三名'
27         select top 3 * from Result where SubjectId=@subjectId order by
StudentResult
28     end
29

```

WHILE循环

```

1  --WHILE循环
2  --可以嵌套
3  --没有{}, 使用begin..end
4  --不能使用true/false, 只能使用条件表达式
5  --如果里面只有一句就可以不使用begin..and包含
6  --可以使用continue/break
7  --如果office不及格的人超过半数(考试题出难了), 则给每个人增加2分, 循环加, 直到不及格的人数
少于一半
8  go
9  declare @subjectName nvarchar(50)='office'--科目名称
10 declare @subjectId int =(select SubjectId from Subject where
SubjectName=@subjectName) --科目ID
11 declare @classId int =(select classid from Subject where
SubjectName=@subjectName) --这一科目所属的班级ID
12 declare @totalNum int --这一科目考试总人数
13 set @totalNum=(select COUNT(*) from Student where ClassId=@classId) --需要考
试这一科目的总人数
14 declare @unpassNum int =(select COUNT(*) from Result where
SubjectId=@subjectId and StudentResult<60) --这一科目考试通过的学员人数
15 while(@unpassNum>@totalNum/2) --如果不及格人数超过总人数一半
16     begin
17         update Result set StudentResult+=2 where SubjectId=@subjectId and
StudentResult<=98
18         --重新计算不及格人数
19         set @unpassNum=(select COUNT(*) from Result where
SubjectId=@subjectId and StudentResult<60)
20     end
21
22     --也可以这样:
23 go
24 declare @subjectName nvarchar(50)='office'--科目名称
25 declare @subjectId int =(select SubjectId from Subject where
SubjectName=@subjectName) --科目ID
26 declare @classId int =(select classid from Subject where
SubjectName=@subjectName) --这一科目所属的班级ID
27 declare @totalNum int --这一科目考试总人数
28 set @totalNum=(select COUNT(*) from Student where ClassId=@classId) --需要考
试这一科目的总人数
29 --declare @unpassNum int =(select COUNT(*) from Result where
SubjectId=@subjectId and StudentResult<60) --这一科目考试通过的学员人数
30 while(1=1)
31     begin

```

```

32         if((select COUNT(*) from Result where SubjectId=@subjectId and
StudentResult<60) >@totalNum/2)
33             update Result set StudentResult+=2 where SubjectId=@subjectId
and StudentResult<=98
34         else
35             break --跳出循环
36     end

```

Transaction

- 指访问并可能更新数据库中各种数据项的一个程序执行单元(unit)--也就是由多个sql语句组成，必须作为一个整体执行 ACID
 - 这些sql语句作为一个整体一起向系统提交，要么都执行、要么都不执行
 - 语法步骤：
 - 开始事务：BEGIN TRANSACTION --开启事务
 - 事务提交：COMMIT TRANSACTION --提交操作
 - 事务回滚：ROLLBACK TRANSACTION --取消操作
 - 判断某条语句执行是否出错：
 - 全局变量@@ERROR;
 - @@ERROR只能判断当前一条T-SQL语句执行是否有错，为了判断事务中所有T-SQL语句是否有错，我们需要对错误进行累计；
- 例： `SET @errorSum=@errorSum+@@error`

```

1  --事务
2  --aa转1000给bb
3  update bank set cmoney=cmoney-1000 where name='aa'
4  update bank set cmoney=cmoney+1000 where name='bb'
5  --上面这种情况要求我们需要 使用某一种方式进行处理 ，这种方式就是事务
6  --事务：就是指这些语句要么都能成功执行，要么都不执行---事务只是一种处理机制。
7  --事务是对 有可能对表数据进行更改的操作而言--增加删除和修改。对查询没用。
8  ---事务的特点：ACID
9  --1.原子性atomic:事务不可以再分，事务必须是原子工作单元；对于其数据修改，要么全都执行，要么全都不执行
10 --2.一致性consistent:事务处理前后。，数据需要保持某种程度的一致
11 --3.隔离性isolate: 每一个事务是独立的，不受其它事务影响
12 --4.持久性durable:事务一旦提交，对数据的修改永久保留
13
14 --如何使用事务：将你执行的增加删除和修改命令包含在 事务的 开启和提交 或者 开启和回滚 之间。
15 --事务开启: begin transaction
16 --事务提交: commit transaction
17 --事务回滚: rollback transaction
18
19 --使用事务进行转帐 一种处理机制
20 declare @cuowuhao int =0 --记录执行语句过程中有可能出现的错误号
21 begin transaction
22     update bank set cmoney=cmoney-1000 where name='aa'
23     set @cuowuhao=@cuowuhao+@@ERROR --只是做一个错误号的累加
24     --if(@@ERROR<>0) --说明这一句的执行出现的错误
25     -- rollback transaction --不可能某一句出现了错误就立刻进行提交或者回滚
26     update bank set cmoney=cmoney+1000 where name='bb'
27     set @cuowuhao+=@@ERROR

```

```

28      --if(@@ERROR<>0) --说明这一句的执行出现的错误
29      -- rollback transaction
30      select * from bank --显示错误操作的结果
31      if(@cuowuhao<>0) --有错误 只有全部语句执行完之后，再进行整体的判断
32          rollback transaction
33      else
34          commit transaction
35      select * from bank --回滚了

```

Stored Procedure

- 前面学的 `if else/while/变量/insert/select` 等，都可以在存储过程中使用
- 优点：
 - 执行速度更快 – 在数据库中保存的存储过程SQL语句都是编译过的
 - 允许模块化程序设计 – 类似方法的复用
 - 提高系统安全性 – 防止SQL注入
 - 减少网络流通量 – 只要传输 存储过程的名称
- 系统存储过程
 - 由系统定义，存放在master数据库中
 - 名称以 `sp` 开头或 `xp` 开头,自定义的存储过程可以以 `usp_` 开头。
- 自定义存储过程
 - 由用户在自己的数据库中创建的存储过程

系统存储过程

`sp_databases`: 列出服务器上的所有数据库。

`sp_helpdb`: 报告有关指定数据库或所有数据库的信息

`sp_renamedb`: 更改数据库的名称

`sp_tables`: 返回当前环境下可查询的对象的列表

`sp_columns`: 回某个表列的信息

`sp_help`: 查看某个表的所有信息

`sp_helpconstraint`: 查看某个表的约束

`sp_helpindex`: 查看某个表的索引

`sp_stored_procedures`: 列出当前环境中的所有存储过程。

`sp_password`: 添加或修改登录帐户的密码。

`sp_helptext`: 显示默认值、未加密的存储过程、用户定义的存储过程、触发器或视图的实际文本。

```

1  --存储过程
2  /*
3  在c#中
4  方法说明:
5  方法名称 参数 调用 返回值
6  参数主要是 一 一对应的原则
7  1.类型对应--子类可以替换父类  int--double (任意类型都可以转化成字符串)
8  2.数量对应--默认值、可变参数
9  3.顺序对应: 可以使用 参数: 值的方法调用

```

```

10 返回值：通过return 返回值。但是只能返回单个值
11 可以通过Ref/out扩展方法的"返回值"
12 void Show(形参)
13 {
14     自定义局部变量
15     逻辑语句
16 }
17 */
18
19 /*
20 创建存储语法：
21 go
22 create procedure usp_存储过程名称
23 (形参) 可以以这个里面定义参数
24 as --相当于方法体
25 {
26     自定义局部变量
27     逻辑语句
28 }
29 go
30
31 exec proc usp_存储过程名称
32 */
33 --获取指定性别和班级名称的学员信息
34 if exists(select * from sysobjects where
name='usp_getAllStuInfoBySexAndClassName')
35     drop proc usp_getAllStuInfoBySexAndClassName
36 go --create procedure必须是批处理中仅有的语句
37 create proc usp_getAllStuInfoBySexAndClassName
38     @className nvarchar(50)
39     @sex char(2)'男', --形参只是声明，不是定义，所有不需要declare
40 as
41     declare @classId int --科目 ID
42     set @classId=(select classid from grade where classname=@className)
43     select * from student where Sex=@sex and classId=@classId
44 go --这个go要写，不然exec也会在procedure里，等于在procedure里call exec，结果会循环
45 --执行存储过程，返回指定班级名称和性别的学员信息
46
47 --创建有默认值的存储过程
48 --往往会有默认值的参数写在所有参数列表的最后，或者赋值的时候写default
49 --参数传递顺序一致：第一个实参默认就是传递给第一个形参。。依次累推
50 --如果有默认值，那么可以使用default
51 --也可以使用 参数 = 值 的方式调用存储过程，这样就与顺序没有关系了。
52 --一旦使用了 '@name = value' 形式之后，所有后续的参数就必须以 '@name = value' 的形式
    传递。前面没有限制
53 --但是后面开始@name=，后面所有赋值都要写@name=的形式
54 exec usp_getAllStuInfoBySexAndClassName @className='七期班',@sex='女'
55
56 -- output值
57 --根据班级和性别查询学员，同时返回总人数和指定性别的人数
58 if exists(select * from sysobjects where name='usp_getSInfoAndCount')
59     drop proc usp_getSInfoAndCount
60 go
61 create proc usp_getSInfoAndCount
62     @totalNum int output, --如果一个参数添加了output修饰，那么说明：它是一个输出参数。标
    明了output说明了你向服务器请求返回这个参数的值。而服务器也知道标识了output的参数在以后
    需要返回
63     @claNum int output, --指定班级和性别的总人数

```

```

64 @ClassName nvarchar(50), --输入参数: 需要用户传入值
65 @sex char(2)='男'
66 as
67     declare @cid int=(select classid from grade where classname=@ClassName)
--根据班级名称获取班级ID
68     select * from Student where ClassId=@cid and Sex=@sex
69     set @totalNum=(select COUNT(*) from Student )--总人数
70     set @claNum =(select COUNT(*) from Student where ClassId=@cid and
Sex=@sex)
71     return 10000
72 go
73
74 --调用在输出参数的存储过程
75 --服务器向你返回值, 用户就需要创建对应的变量做接收
76 --标明了output说明了你会向服务器请求返回这个参数的值。而服务器也知道标识了output的参数在
以后需要返回
77 declare @tnum int,@cnum int
78 exec usp_getSInfoAndCount @ClassName='七期班' ,@totalNum=@tnum
output,@claNum=@cnum output --要加output, 要对应
79 print @tnum
80 print @cnum
81 print 'ok'
82
83 ---返回指定人数
84 if exists(select * from sysobjects where name='usp_getNameByNo')
85     drop proc usp_getNameByNo
86 go
87 create proc usp_getNameByNo
88     @cid int
89 as
90     declare @cnt int
91     set @cnt=(select COUNT(*) from Student where ClassId=@cid)
92 --return只能返回 整数值, 非整数只能用output
93     return @cnt
94 go
95 --调用存储过程, 返回指定学号的学员名字
96 declare @count int
97 exec @count= usp_getNameByNo 6 --记住这个写法
98 print @count

```

Trigger

- 触发器的作用:
 - 自动化操作, 减少了手动操作以及出错的几率。
 - 触发器是一种特殊类型的存储过程, 它不同于前面介绍过的一般的存储过程。【在SQL内部把触发器看做是存储过程但是不能传递参数】
- 一般的存储过程通过存储过程名称被直接调用, 而触发器主要是通过事件进行触发而被执行。
- 触发器是一个功能强大的工具, 在表中数据发生变化时自动强制执行。触发器可以用于SQL Server约束、默认值和规则的完整性检查, 还可以完成难以用普通约束实现的复杂功能。
- 那究竟何为触发器? 在SQL Server里面也就是对某一个表的一定的操作, 触发某种条件, 从而执行的一段程序。触发器是一个特殊的存储过程。

常见的触发器

- DML触发器
 - Insert、delete、update（不支持select）
 - after触发器(for)、instead of触发器（不支持before触发器）
- (*)DDL触发器
 - create table、create database、alter、drop....

inserted表与deleted表是干什么的？

- inserted表包含新数据
 - insert、update触发器会用到
- deleted表包含旧数据
 - delete、update触发器会用到
- INSERT触发器
 - 当有人向表中插入新的一行时，被标记为FOR INSERT的触发器的代码就会执行。对于插入的每一行来说，SQL Server会创建一个新行的副本并把该副本插入到一个特殊的表中，该表只在触发器的作用域内存在，该表被称为Inserted表。特别需要注意的是，Inserted表只在触发器激活时存在。在触发器开启之前或完成之后，都要认为该表示不存在的。
- DELETE触发器
 - 它和INSERT触发器的工作方式相同，只是Inserted表示空的(毕竟是进行删除而非插入，所以对于Inserted表示没有记录)。相反，每个被删除的记录的副本将会插入到另一个表中，该表称为Deleted表，和Inserted表类似，该表只存在于触发器激活的时间内。
- UPDATE触发器
 - 除了有一点改变以外，UPDATE触发器和前面的触发器是很类似的。对表中现有的记录进行修改时，都会激活被声明FOR UPDATE的触发器的代码。唯一的改变是没有UPDATE表。SQL Server认为每一行好像删除了现有记录，并插入了全新的记录。声明为FOR UPDATE的触发器并不是只包含一个表，而是两个特殊的表，称为Inserted表和Deleted表。当然，这两个表的行数是完全相同。

After/For触发器和instead of触发器

After触发器：

- 在语句执行完毕之后触发
- 按语句触发，而不是所影响的行数，无论所影响为多少行，只触发一次。
- 只能建立在常规表上，不能建立在视图和临时表上。（*）
- 可以递归触发，最高可达32级。
- update(列)，在update语句触发时，判断某列是否被更新，返回布尔值。
- 介绍如何编写after触发器。

instead of触发器

- 用来替换原本的操作
- 不会递归触发
- 可以在约束被检查之前触发
- 可以建在临时表和视图上（*）
- 介绍如何编写instead of 触发器

触发器使用建议

- 尽量避免在触发器中执行耗时操作，因为触发器会与SQL语句认为在同一个事务中。（事务不结束，就无法释放锁。）

- 避免在触发器中做复杂操作，影响触发器性能的因素比较多（如：产品版本、所使用架构等等），要想编写高效的触发器考虑因素比较多（编写触发器容易，编写复杂的高性能触发器难！）。
- 尽量不要对同一个表做多个触发器，可能都会执行（不知道发生什么）

```

1  --触发器
2  /*
3  语法：
4  create trigger tr_触发器名称
5  on 表 after/instead of 增加删除(delete)修改
6  as
7      任意的逻辑代码--存储过程
8  go
9  */
10 if exists(select * from sysobjects where name='tr_grade_insert')
11 drop trigger tr_grade_insert
12 go
13 create trigger tr_grade_insert
14 on grade for insert --为grade表创建触发器，在你对grade表进行插入操作后触发
15 as
16     select * from student
17 go
18 insert into grade values('21312321')
19
20 ---两个临时表
21 if exists(select * from sysobjects where name='tr_grade_insert')
22 drop trigger tr_grade_insert
23 go
24 create trigger tr_grade_insert
25 on grade for insert --为grade表创建触发器，在你对grade表进行插入操作后触发
26 as
27 begin
28     print 'inserted表存储操作之后的 与当前操作相关的数据，而与之前表的数据无关'
29     select * from inserted
30     print 'deleted表存储操作之前的数据'
31     select * from deleted
32 end
33 go
34 insert into grade values('亲爱sdas的')
35
36 --注意begin...end
37 CREATE TRIGGER tr_updateStudent ON score --相当于外键检查约束
38 after INSERT -- 后置的新增触发器
39 AS
40 Begin
41     declare @sid int,@scoreid int--定义两个变量
42     select @sid = sid,@ scoreid=id from inserted--获得新增行的数据
43     if exists(select * from student where sid=@sid)--判断分数学员是否存在
44         print '插入成功'
45     else --如果不存在，则把更新成功的分数记录给删除掉
46         delete from score where sid = @scoreId
47 End
48 Insert into score (studentId,english) values(100,100)

```

- 全表扫描：对数据进行检索（select）效率最差的是全表扫描，就是一条条的找。
- 如果没有目录，查汉语字典就要一页页的翻，而有了目录只要查询目录即可。为了提高检索的速度，可以为经常进行查询的列添加索引，相当于创建目录。
- 创建索引的方式，在表设计器中点击右键，选择“索引/键”→添加→在列中选择索引包含的列。
- 使用索引能提高查询效率，但是索引也是占据空间的，而且添加、更新、删除数据的时候也需要同步更新索引，因此会降低Insert、Update、Delete的速度。**只在经常检索的字段上(Where)创建索引。**
- (*) 即使创建了索引，仍然有可能全表扫描，比如like、函数、类型转换等。
- 删除索引

```
drop index T8.IX_T8_tage
```

没有聚集索引，默认按照主键排序，更改的是物理表格。存储的位置

```
1  if exists (select * from sysindexes where name = 'IX_student_Studentname')
2  drop index student.IX_Student_studentname --记得加上表名
3  go
4
5  create clustered index IX_Student_studentname --or unclustered
6  on student(studentname)
7
8  CREATE UNIQUE INDEX index_name ON table_name (column_name) --在表上创建一个唯一的索引。唯一的索引意味着两个行不能拥有相同的索引值。
9  CREATE INDEX PersonIndex ON Person (LastName DESC)
```

临时表

局部临时表

- `create table #tbName(列信息);`
- 表名前缀 #
- 只在当前会话中有效，不能跨连接访问
- 作用域范围类似C#：
 - 如果直接在连接会话中创建的，则当前连接断开后删除，如果是在存储过程中创建的则当前存储过程执行完毕后删除

全局临时表

- `create table ##tbName(列信息);`
- 表名前缀 ##
- 多个会话可共享全局临时表
- 当创建全局临时表的会话断开，并且没有用户正在访问全局临时表时删除

表变量

- `declare @varT1 table(col1 int,col2 char(2));` //存储更小量的数据，比临时表有更多的限制。
- 临时数据都存储在tempdb,当服务重新启动的时候，会重建tempdb.

```
1  --临时表
2  --局部临时表，存储在系统数据库-tempdb-临时表中，并且有很长的地址表示这是局部临时表
3  --局部临时表只能当前创建临时表的会话中使用。如果关闭了这个会话，那么临时表就自动消失
```



```

4  --当前会话类似当前的sql窗口
5  create table #temp
6  (
7      --字段名称 类型 特征
8      cid int,
9      cname nvarchar(50)
10
11 )
12
13 select * into #newtemp from grade
14 select * from #newtemp
15 truncate table grade
16 insert into grade select classname from #newtemp
17
18 --全局临时表只要不关闭当前会话，那么在其他会话中还是可以使用，但是如果关闭当前会话，那么临时也会消失
19 create table ##temp
20 (
21     --字段名称 类型 特征
22     cid int,
23     cname nvarchar(50)
24
25 )
26 insert into ##temp select * from grade
27 select * from ##temp
28 truncate table grade
29 insert into grade select cname from ##temp

```

Try/Catch

因此正常情况下，catch中记录完异常之后，要“抛出”异常，当异常发生的时候，明确地告诉调用方发生了什么问题。

throw是比raiserror更加方便和直观的异常抛出方式，也是推荐的异常处理方式，具体差异网上一大把就不多说了

throw有两种使用方式，抛出自定义异常和直接在catch块中抛出异常。

抛出自定义异常的时候有三个必须参数，下面会细说，catch块中可以直接用throw不需要任何参数的方式抛出捕获到的异常

throwi语句的前一句需要一分号结尾，前一句又不能保证一定有分号，

所以可以直接把分号写在throw的前面，比如文中的;throw 50000,'Price can not be less than 0',1 写法

当抛出自定义错误的时候，throw语句有三个参数，参考如下

throwi语句安全级别默认为16并且不会被修改，换句话说就是throwi语句执行之后将抛出错误，打断当前Session的批处理语句，throw后面的语句将不会执行

第一个参数是错误号，用户自定义错误号要大于50000(50000 to 2147483647)

第二个参数是自定义错误信息内容，可以根据需要自定义

第三个参数是Error State，他的作用是可以标记异常发生的位置，

比如同样是“参数不能小于0”的错误提示，整个存储过程中有可能有两个地方进行同样的校验

就可以在两个地方使用Error State不同来抛出异常;

```
1 throw 50000, 'Price can not be less than 0', 1
2 throw 50000, 'Price can not be less than 0', 2,
```

因为Error State不同, 就可以根据具体的Error State更加方便地知道是哪个地方发生了异常, 参考
throw语句存储自定义异常到messages系统表

可以将自定义的异常信息加入到sys.messages 系统表中, 先加英文的, 再加中文的。

然后使用的时候可以使用这个预定义的消息, 而不是每次写一个字符串

```
1 begin try
2 ...
3 end try
4
5 begin catch
6 ...
7 end catch
8
9 throw
```

QA

T-SQL中的方括号是什么

如果在列名称或标识符中使用关键字或特殊字符, 则必须使用括号。您可以命名一个列[First Name] (带空格) - 但是每次引用该列时都需要使用括号。新的工具将它们添加到任何地方, 以防万一或一致。

加了[]是为了防止歧义, 使计算机能识别。有些字段可能是关键字, 这时候你直接用字段名就会报错, 如果加了[]就可以正常执行了

uniqueidentifier是什么

uniqueidentifier类型可以配合T-SQL中的newid和newsequentialid来生成唯一标识符, 具体区别如下(摘抄自微软官方文档)。

Nonsequential GUIDs: You can generate nonsequential global unique identifiers to be stored in an attribute of a UNIQUEIDENTIFIER type. You can use the T-SQL function NEWID to generate a new GUID, possibly invoking it with a default expression attached to the column. You can also generate one from anywhere. for example, the client by using an application programming interface (API) that generates a new GUID. The GUIDs are guaranteed to be unique across space and time.

Sequential GUIDs: You can generate sequential GUIDs within the machine by using the T-SQL function NEWSEQUENTIALID.

```
1 declare @id uniqueidentifier --声明sql变量
2 SET @id= NEWID() --赋值
3 SET @ID= NEWSEQUENTIALID()
```

object_id是什么

[SQL SERVER 中的 object_id\(\)函数](#)

`object_id('name','type')` 返回的是obj_id, `object_name(@@PROCID)` 返回的是name

OBJECT_ID:此方法返回数据库对象标识号。类型为int, 表示该对象在系统中的编号。

OBJECT_NAME:根据对象ID得到对象名

OBJECT_DEFINITION:返回对象的源文件.