[**SQL Server T-SQL高级查询**](http://www.cnblogs.com/hoojo/archive/2011/07/16/2108129.html)

高级查询在数据库中用得是最频繁的，也是应用最广泛的。

Ø 基本常用查询

--select

select \* from student;

--all 查询所有

select all sex from student;

--distinct 过滤重复

select distinct sex from student;

--count 统计

select count(\*) from student;

select count(sex) from student;

select count(distinct sex) from student;

--top 取前N条记录

select top 3 \* from student;

--alias column name 列重命名

select id as 编号, name '名称', sex 性别 from student;

--alias table name 表重命名

select id, name, s.id, s.name from student s;

--column 列运算

select (age + id) col from student;

select s.name + '-' + c.name from classes c, student s where s.cid = c.id;

--where 条件

select \* from student where id = 2;

select \* from student where id > 7;

select \* from student where id < 3;

select \* from student where id <> 3;

select \* from student where id >= 3;

select \* from student where id <= 5;

select \* from student where id !> 3;

select \* from student where id !< 5;

--and 并且

select \* from student where id > 2 and sex = 1;

--or 或者

select \* from student where id = 2 or sex = 1;

--between ... and ... 相当于并且

select \* from student where id between 2 and 5;

select \* from student where id not between 2 and 5;

--like 模糊查询

select \* from student where name like '%a%';

select \* from student where name like '%[a][o]%';

select \* from student where name not like '%a%';

select \* from student where name like 'ja%';

select \* from student where name not like '%[j,n]%';

select \* from student where name like '%[j,n,a]%';

select \* from student where name like '%[^ja,as,on]%';

select \* from student where name like '%[ja\_on]%';

--in 子查询

select \* from student where id in (1, 2);

--not in 不在其中

select \* from student where id not in (1, 2);

--is null 是空

select \* from student where age is null;

--is not null 不为空

select \* from student where age is not null;

--order by 排序

select \* from student order by name;

select \* from student order by name desc;

select \* from student order by name asc;

--group by 分组

按照年龄进行分组统计

select count(age), age from student group by age;

按照性别进行分组统计

select count(\*), sex from student group by sex;

按照年龄和性别组合分组统计，并排序

select count(\*), sex from student group by sex, age order by age;

按照性别分组，并且是id大于2的记录最后按照性别排序

select count(\*), sex from student where id > 2 group by sex order by sex;

查询id大于2的数据，并完成运算后的结果进行分组和排序

select count(\*), (sex \* id) new from student where id > 2 group by sex \* id order by sex \* id;

--group by all 所有分组

按照年龄分组，是所有的年龄

select count(\*), age from student group by all age;

--having 分组过滤条件

按照年龄分组，过滤年龄为空的数据，并且统计分组的条数和现实年龄信息

select count(\*), age from student group by age having age is not null;

按照年龄和cid组合分组，过滤条件是cid大于1的记录

select count(\*), cid, sex from student group by cid, sex having cid > 1;

按照年龄分组，过滤条件是分组后的记录条数大于等于2

select count(\*), age from student group by age having count(age) >= 2;

按照cid和性别组合分组，过滤条件是cid大于1，cid的最大值大于2

select count(\*), cid, sex from student group by cid, sex having cid > 1 and max(cid) > 2;

Ø 嵌套子查询

    子查询是一个嵌套在select、insert、update或delete语句或其他子查询中的查询。任何允许使用表达式的地方都可以使用子查询。子查询也称为内部查询或内部选择，而包含子查询的语句也成为外部查询或外部选择。

# from (select … table)示例

将一个table的查询结果当做一个新表进行查询

select \* from (

select id, name from student where sex = 1

) t where t.id > 2;

上面括号中的语句，就是子查询语句（内部查询）。在外面的是外部查询，其中外部查询可以包含以下语句：

     1、 包含常规选择列表组件的常规select查询

     2、 包含一个或多个表或视图名称的常规from语句

     3、 可选的where子句

     4、 可选的group by子句

     5、 可选的having子句

# 示例

查询班级信息，统计班级学生人生

select \*, (select count(\*) from student where cid = classes.id) as num

from classes order by num;

# in, not in子句查询示例

查询班级id大于小于的这些班级的学生信息

select \* from student where cid in (

select id from classes where id > 2 and id < 4

);

查询不是班的学生信息

select \* from student where cid not in (

select id from classes where name = '2班'

)

in、not in 后面的子句返回的结果必须是一列，这一列的结果将会作为查询条件对应前面的条件。如cid对应子句的id；

# exists和not exists子句查询示例

查询存在班级id为的学生信息

select \* from student where exists (

select \* from classes where id = student.cid and id = 3

);

查询没有分配班级的学生信息

select \* from student where not exists (

select \* from classes where id = student.cid

);

exists和not exists查询需要内部查询和外部查询进行一个关联的条件，如果没有这个条件将是查询到的所有信息。如：id等于student.id；

# some、any、all子句查询示例

查询班级的学生年龄大于班级的学生的年龄的信息

select \* from student where cid = 5 and age > all (

select age from student where cid = 3

);

select \* from student where cid = 5 and age > any (

select age from student where cid = 3

);

select \* from student where cid = 5 and age > some (

select age from student where cid = 3

);

Ø 聚合查询

1、 distinct去掉重复数据

select distinct sex from student;

select count(sex), count(distinct sex) from student;

2、 compute和compute by汇总查询

对年龄大于的进行汇总

select age from student

where age > 20 order by age compute sum(age) by age;

对年龄大于的按照性别进行分组汇总年龄信息

select id, sex, age from student

where age > 20 order by sex, age compute sum(age) by sex;

按照年龄分组汇总

select age from student

where age > 20 order by age, id compute sum(age);

按照年龄分组，年龄汇总，id找最大值

select id, age from student

where age > 20 order by age compute sum(age), max(id);

compute进行汇总前面是查询的结果，后面一条结果集就是汇总的信息。compute子句中可以添加多个汇总表达式，可以添加的信息如下：

     a、 可选by关键字。它是每一列计算指定的行聚合

     b、 行聚合函数名称。包括sum、avg、min、max、count等

     c、 要对其执行聚合函数的列

     compute by适合做先分组后汇总的业务。compute by后面的列一定要是order by中出现的列。

3、 cube汇总

cube汇总和compute效果类似，但语法较简洁，而且返回的是一个结果集。

select count(\*), sex from student group by sex with cube;

select count(\*), age, sum(age) from student where age is not null group by age with cube;

cube要结合group by语句完成分组汇总

Ø 排序函数

   排序在很多地方需要用到，需要对查询结果进行排序并且给出序号。比如：

   1、 对某张表进行排序，序号需要递增不重复的

   2、 对学生的成绩进行排序，得出名次，名次可以并列，但名次的序号是连续递增的

   3、 在某些排序的情况下，需要跳空序号，虽然是并列

基本语法

排序函数 over([分组语句] 排序子句[desc][asc])

排序子句 order by 列名, 列名

分组子句 partition by 分组列, 分组列

# row\_number函数

根据排序子句给出递增连续序号

按照名称排序的顺序递增

select s.id, s.name, cid, c.name, row\_number() over(order by c.name) as number

from student s, classes c where cid = c.id;

# rank函数函数

根据排序子句给出递增的序号，但是存在并列并且跳空

顺序递增

select id, name, rank() over(order by cid) as rank from student;

跳过相同递增

select s.id, s.name, cid, c.name, rank() over(order by c.name) as rank

from student s, classes c where cid = c.id;

# dense\_rank函数

根据排序子句给出递增的序号，但是存在并列不跳空

不跳过，直接递增

select s.id, s.name, cid, c.name, dense\_rank() over(order by c.name) as dense

from student s, classes c where cid = c.id;

# partition by分组子句

可以完成对分组的数据进行增加排序，partition by可以与以上三个函数联合使用。

利用partition by按照班级名称分组，学生id排序

select s.id, s.name, cid, c.name, row\_number() over(partition by c.name order by s.id) as rank

from student s, classes c where cid = c.id;

select s.id, s.name, cid, c.name, rank() over(partition by c.name order by s.id) as rank

from student s, classes c where cid = c.id;

select s.id, s.name, cid, c.name, dense\_rank() over(partition by c.name order by s.id) as rank

from student s, classes c where cid = c.id;

# ntile平均排序函数

将要排序的数据进行平分，然后按照等分排序。ntile中的参数代表分成多少等分。

select s.id, s.name, cid, c.name,

ntile(5) over(order by c.name) as ntile

from student s, classes c where cid = c.id;

Ø 集合运算

操作两组查询结果，进行交集、并集、减集运算

1、 union和union all进行并集运算

--union 并集、不重复

select id, name from student where name like 'ja%'

union

select id, name from student where id = 4;

--并集、重复

select \* from student where name like 'ja%'

union all

select \* from student;

2、 intersect进行交集运算

--交集（相同部分）

select \* from student where name like 'ja%'

intersect

select \* from student;

3、 except进行减集运算

--减集（除相同部分）

select \* from student where name like 'ja%'

except

select \* from student where name like 'jas%';

Ø 公式表表达式

查询表的时候，有时候中间表需要重复使用，这些子查询被重复查询调用，不但效率低，而且可读性低，不利于理解。那么公式表表达式可以解决这个问题。

我们可以将公式表表达式（CET）视为临时结果集，在select、insert、update、delete或是create view语句的执行范围内进行定义。

--表达式

with statNum(id, num) as

(

select cid, count(\*)

from student

where id > 0

group by cid

)

select id, num from statNum order by id;

with statNum(id, num) as

(

select cid, count(\*)

from student

where id > 0

group by cid

)

select max(id), avg(num) from statNum;

Ø 连接查询

1、 简化连接查询

--简化联接查询

select s.id, s.name, c.id, c.name from student s, classes c where s.cid = c.id;

2、 left join左连接

--左连接

select s.id, s.name, c.id, c.name from student s left join classes c on s.cid = c.id;

3、 right join右连接

--右连接

select s.id, s.name, c.id, c.name from student s right join classes c on s.cid = c.id;

4、 inner join内连接

--内连接

select s.id, s.name, c.id, c.name from student s inner join classes c on s.cid = c.id;

--inner可以省略

select s.id, s.name, c.id, c.name from student s join classes c on s.cid = c.id;

5、 cross join交叉连接

--交叉联接查询，结果是一个笛卡儿乘积

select s.id, s.name, c.id, c.name from student s cross join classes c

--where s.cid = c.id;

6、 自连接（同一张表进行连接查询）

--自连接

select distinct s.\* from student s, student s1 where s.id <> s1.id and s.sex = s1.sex;

Ø 函数

1、 聚合函数

max最大值、min最小值、count统计、avg平均值、sum求和、var求方差

select

max(age) max\_age,

min(age) min\_age,

count(age) count\_age,

avg(age) avg\_age,

sum(age) sum\_age,

var(age) var\_age

from student;

2、 日期时间函数

select dateAdd(day, 3, getDate());--加天

select dateAdd(year, 3, getDate());--加年

select dateAdd(hour, 3, getDate());--加小时

--返回跨两个指定日期的日期边界数和时间边界数

select dateDiff(day, '2011-06-20', getDate());

--相差秒数

select dateDiff(second, '2011-06-22 11:00:00', getDate());

--相差小时数

select dateDiff(hour, '2011-06-22 10:00:00', getDate());

select dateName(month, getDate());--当前月份

select dateName(minute, getDate());--当前分钟

select dateName(weekday, getDate());--当前星期

select datePart(month, getDate());--当前月份

select datePart(weekday, getDate());--当前星期

select datePart(second, getDate());--当前秒数

select day(getDate());--返回当前日期天数

select day('2011-06-30');--返回当前日期天数

select month(getDate());--返回当前日期月份

select month('2011-11-10');

select year(getDate());--返回当前日期年份

select year('2010-11-10');

select getDate();--当前系统日期

select getUTCDate();--utc日期

3、 数学函数

select pi();--PI函数

select rand(100), rand(50), rand(), rand();--随机数

select round(rand(), 3), round(rand(100), 5);--精确小数位

--精确位数，负数表示小数点前

select round(123.456, 2), round(254.124, -2);

select round(123.4567, 1, 2);

4、 元数据

select col\_name(object\_id('student'), 1);--返回列名

select col\_name(object\_id('student'), 2);

--该列数据类型长度

select col\_length('student', col\_name(object\_id('student'), 2));

--该列数据类型长度

select col\_length('student', col\_name(object\_id('student'), 1));

--返回类型名称、类型id

select type\_name(type\_id('varchar')), type\_id('varchar');

--返回列类型长度

select columnProperty(object\_id('student'), 'name', 'PRECISION');

--返回列所在索引位置

select columnProperty(object\_id('student'), 'sex', 'ColumnId');

5、 字符串函数

select ascii('a');--字符转换ascii值

select ascii('A');

select char(97);--ascii值转换字符

select char(65);

select nchar(65);

select nchar(45231);

select nchar(32993);--unicode转换字符

select unicode('A'), unicode('中');--返回unicode编码值

select soundex('hello'), soundex('world'), soundex('word');

select patindex('%a', 'ta'), patindex('%ac%', 'jack'), patindex('dex%', 'dexjack');--匹配字符索引

select 'a' + space(2) + 'b', 'c' + space(5) + 'd';--输出空格

select charIndex('o', 'hello world');--查找索引

select charIndex('o', 'hello world', 6);--查找索引

select quoteName('abc[]def'), quoteName('123]45');

--精确数字

select str(123.456, 2), str(123.456, 3), str(123.456, 4);

select str(123.456, 9, 2), str(123.456, 9, 3), str(123.456, 6, 1), str(123.456, 9, 6);

select difference('hello', 'helloWorld');--比较字符串相同

select difference('hello', 'world');

select difference('hello', 'llo');

select difference('hello', 'hel');

select difference('hello', 'hello');

select replace('abcedef', 'e', 'E');--替换字符串

select stuff('hello world', 3, 4, 'ABC');--指定位置替换字符串

select replicate('abc#', 3);--重复字符串

select subString('abc', 1, 1), subString('abc', 1, 2), subString('hello Wrold', 7, 5);--截取字符串

select len('abc');--返回长度

select reverse('sqlServer');--反转字符串

select left('leftString', 4);--取左边字符串

select left('leftString', 7);

select right('leftString', 6);--取右边字符串

select right('leftString', 3);

select lower('aBc'), lower('ABC');--小写

select upper('aBc'), upper('abc');--大写

--去掉左边空格

select ltrim(' abc'), ltrim('# abc#'), ltrim(' abc');

--去掉右边空格

select rtrim(' abc '), rtrim('# abc# '), rtrim('abc');

6、 安全函数

select current\_user;

select user;

select user\_id(), user\_id('dbo'), user\_id('public'), user\_id('guest');

select user\_name(), user\_name(1), user\_name(0), user\_name(2);

select session\_user;

select suser\_id('sa');

select suser\_sid(), suser\_sid('sa'), suser\_sid('sysadmin'), suser\_sid('serveradmin');

select is\_member('dbo'), is\_member('public');

select suser\_name(), suser\_name(1), suser\_name(2), suser\_name(3);

select suser\_sname(), suser\_sname(0x01), suser\_sname(0x02), suser\_sname(0x03);

select is\_srvRoleMember('sysadmin'), is\_srvRoleMember('serveradmin');

select permissions(object\_id('student'));

select system\_user;

select schema\_id(), schema\_id('dbo'), schema\_id('guest');

select schema\_name(), schema\_name(1), schema\_name(2), schema\_name(3);

7、 系统函数

select app\_name();--当前会话的应用程序名称

select cast(2011 as datetime), cast('10' as money), cast('0' as varbinary);--类型转换

select convert(datetime, '2011');--类型转换

select coalesce(null, 'a'), coalesce('123', 'a');--返回其参数中第一个非空表达式

select collationProperty('Traditional\_Spanish\_CS\_AS\_KS\_WS', 'CodePage');

select current\_timestamp;--当前时间戳

select current\_user;

select isDate(getDate()), isDate('abc'), isNumeric(1), isNumeric('a');

select dataLength('abc');

select host\_id();

select host\_name();

select db\_name();

select ident\_current('student'), ident\_current('classes');--返回主键id的最大值

select ident\_incr('student'), ident\_incr('classes');--id的增量值

select ident\_seed('student'), ident\_seed('classes');

select @@identity;--最后一次自增的值

select identity(int, 1, 1) as id into tab from student;--将studeng表的烈属，以/1自增形式创建一个tab

select \* from tab;

select @@rowcount;--影响行数

select @@cursor\_rows;--返回连接上打开的游标的当前限定行的数目

select @@error;--T-SQL的错误号

select @@procid;

8、 配置函数

set datefirst 7;--设置每周的第一天，表示周日

select @@datefirst as '星期的第一天', datepart(dw, getDate()) AS '今天是星期';

select @@dbts;--返回当前数据库唯一时间戳

set language 'Italian';

select @@langId as 'Language ID';--返回语言id

select @@language as 'Language Name';--返回当前语言名称

select @@lock\_timeout;--返回当前会话的当前锁定超时设置（毫秒）

select @@max\_connections;--返回SQL Server 实例允许同时进行的最大用户连接数

select @@MAX\_PRECISION AS 'Max Precision';--返回decimal 和numeric 数据类型所用的精度级别

select @@SERVERNAME;--SQL Server 的本地服务器的名称

select @@SERVICENAME;--服务名

select @@SPID;--当前会话进程id

select @@textSize;

select @@version;--当前数据库版本信息

9、 系统统计函数

select @@CONNECTIONS;--连接数

select @@PACK\_RECEIVED;

select @@CPU\_BUSY;

select @@PACK\_SENT;

select @@TIMETICKS;

select @@IDLE;

select @@TOTAL\_ERRORS;

select @@IO\_BUSY;

select @@TOTAL\_READ;--读取磁盘次数

select @@PACKET\_ERRORS;--发生的网络数据包错误数

select @@TOTAL\_WRITE;--sqlserver执行的磁盘写入次数

select patIndex('%soft%', 'microsoft SqlServer');

select patIndex('soft%', 'software SqlServer');

select patIndex('%soft', 'SqlServer microsoft');

select patIndex('%so\_gr%', 'Jsonisprogram');

10、 用户自定义函数

# 查看当前数据库所有函数

--查询所有已创建函数

select definition,\* from sys.sql\_modules m join sys.objects o on m.object\_id = o.object\_id

and type in('fn', 'if', 'tf');

# 创建函数

if (object\_id('fun\_add', 'fn') is not null)

drop function fun\_add

go

create function fun\_add(@num1 int, @num2 int)

returns int

with execute as caller

as

begin

declare @result int;

if (@num1 is null)

set @num1 = 0;

if (@num2 is null)

set @num2 = 0;

set @result = @num1 + @num2;

return @result;

end

go

调用函数

select dbo.fun\_add(id, age) from student;

--自定义函数，字符串连接

if (object\_id('fun\_append', 'fn') is not null)

drop function fun\_append

go

create function fun\_append(@args nvarchar(1024), @args2 nvarchar(1024))

returns nvarchar(2048)

as

begin

return @args + @args2;

end

go

select dbo.fun\_append(name, 'abc') from student;

# 修改函数

alter function fun\_append(@args nvarchar(1024), @args2 nvarchar(1024))

returns nvarchar(1024)

as

begin

declare @result varchar(1024);

--coalesce返回第一个不为null的值

set @args = coalesce(@args, '');

set @args2 = coalesce(@args2, '');;

set @result = @args + @args2;

return @result;

end

go

select dbo.fun\_append(name, '#abc') from student;

# 返回table类型函数

--返回table对象函数

select name, object\_id, type from sys.objects where type in ('fn', 'if', 'tf') or type like '%f%';

if (exists (select \* from sys.objects where type in ('fn', 'if', 'tf') and name = 'fun\_find\_stuRecord'))

drop function fun\_find\_stuRecord

go

create function fun\_find\_stuRecord(@id int)

returns table

as

return (select \* from student where id = @id);

go

select \* from dbo.fun\_find\_stuRecord(2);

# [SQL Server Transact-SQL 编程](http://www.cnblogs.com/hoojo/archive/2011/07/15/2107740.html)

T-SQL语句用于管理SQL Server数据库引擎实例，创建和管理数据库对象，以及查询、插入、修改和删除数据。

Ø 变量

     1、 局部变量（Local Variable）

          局部变量是用户可以自定义的变量，它的作用范围是仅在程序内部，在程序中通常用来储存从表中查询到的数据或当做程序执行过程中的暂存变量。使用局部变量必须以@开头，而且必须用declare命令后才能使用。

          基本语法：

声明变量

declare @变量名 变量类型 [@变量名 变量类型]

为变量赋值

set @变量名 = 变量值;

select @变量名 = 变量值;

          示例：

--局部变量

declare @id char(10)--声明一个长度的变量id

declare @age int --声明一个int类型变量age

select @id = 22 --赋值操作

set @age = 55 --赋值操作

print convert(char(10), @age) + '#' + @id

select @age, @id

go

简单hello world示例

declare @name varchar(20);

declare @result varchar(200);

set @name = 'jack';

set @result = @name + ' say: hello world!';

select @result;

查询数据示例

declare @id int, @name varchar(20);

set @id = 1;

select @name = name from student where id = @id;

select @name;

select赋值

declare @name varchar(20);

select @name = 'jack';

select \* from student where name = @name;

          从上面的示例可以看出，局部变量可用于程序中保存临时数据、传递数据。Set赋值一般用于赋值指定的常量个变量。而select多用于查询的结果进行赋值，当然select也可以将常量赋值给变量。

          注意：在使用select进行赋值的时候，如果查询的结果是多条的情况下，会利用最后一条数据进行赋值，前面的赋值结果将会被覆盖。

     2、 全局变量（Global Variable）

          全局变量是系统内部使用的变量，其作用范围并不局限于某一程序而是任何程序均可随时调用的。全局变量一般存储一些系统的配置设定值、统计数据。

全局变量

select @@identity;--最后一次自增的值

select identity(int, 1, 1) as id into tab from student;--将studeng表的烈属，以/1自增形式创建一个tab

select \* from tab;

select @@rowcount;--影响行数

select @@cursor\_rows;--返回连接上打开的游标的当前限定行的数目

select @@error;--T-SQL的错误号

select @@procid;

--配置函数

set datefirst 7;--设置每周的第一天，表示周日

select @@datefirst as '星期的第一天', datepart(dw, getDate()) AS '今天是星期';

select @@dbts;--返回当前数据库唯一时间戳

set language 'Italian';

select @@langId as 'Language ID';--返回语言id

select @@language as 'Language Name';--返回当前语言名称

select @@lock\_timeout;--返回当前会话的当前锁定超时设置（毫秒）

select @@max\_connections;--返回SQL Server 实例允许同时进行的最大用户连接数

select @@MAX\_PRECISION AS 'Max Precision';--返回decimal 和numeric 数据类型所用的精度级别

select @@SERVERNAME;--SQL Server 的本地服务器的名称

select @@SERVICENAME;--服务名

select @@SPID;--当前会话进程id

select @@textSize;

select @@version;--当前数据库版本信息

--系统统计函数

select @@CONNECTIONS;--连接数

select @@PACK\_RECEIVED;

select @@CPU\_BUSY;

select @@PACK\_SENT;

select @@TIMETICKS;

select @@IDLE;

select @@TOTAL\_ERRORS;

select @@IO\_BUSY;

select @@TOTAL\_READ;--读取磁盘次数

select @@PACKET\_ERRORS;--发生的网络数据包错误数

select @@TOTAL\_WRITE;--sqlserver执行的磁盘写入次数

Ø 输出语句

     T-SQL支持输出语句，用于显示结果。常用输出语句有两种：

     基本语法

print 变量或表达式

select 变量或表达式

      示例

select 1 + 2;

select @@language;

select user\_name();

print 1 + 2;

print @@language;

print user\_name();

     print在输出值不少字符串的情况下，需要用convert转换成字符串才能正常输出，而且字符串的长度在超过8000的字符以后，后面的将不会显示。

Ø 逻辑控制语句

     1、 if-else判断语句

          语法

if <表达式>

<命令行或程序块>

else if <表达式>

<命令行或程序块>

else

<命令行或程序块>

          示例

if简单示例

if 2 > 3

print '2 > 3';

else

print '2 < 3';

if (2 > 3)

print '2 > 3';

else if (3 > 2)

print '3 > 2';

else

print 'other';

简单查询判断

declare @id char(10),

@pid char(20),

@name varchar(20);

set @name = '广州';

select @id = id from ab\_area where areaName = @name;

select @pid = pid from ab\_area where id = @id;

print @id + '#' + @pid;

if @pid > @id

begin

print @id + '%';

select \* from ab\_area where pid like @id + '%';

end

else

begin

print @id + '%';

print @id + '#' + @pid;

select \* from ab\_area where pid = @pid;

end

go

       2、 while…continue…break循环语句

          基本语法

while <表达式>

begin

<命令行或程序块>

[break]

[continue]

<命令行或程序块>

end

          示例

--while循环输出到

declare @i int;

set @i = 1;

while (@i < 11)

begin

print @i;

set @i = @i + 1;

end

go

--while continue 输出到

declare @i int;

set @i = 1;

while (@i < 11)

begin

if (@i < 5)

begin

set @i = @i + 1;

continue;

end

print @i;

set @i = @i + 1;

end

go

--while break 输出到

declare @i int;

set @i = 1;

while (1 = 1)

begin

print @i;

if (@i >= 5)

begin

set @i = @i + 1;

break;

end

set @i = @i + 1;

end

go

     3、 case

          基本语法

case

when <条件表达式> then <运算式>

when <条件表达式> then <运算式>

when <条件表达式> then <运算式>

[else <运算式>]

end

          示例

select \*,

case sex

when 1 then '男'

when 0 then '女'

else '火星人'

end as '性别'

from student;

select areaName, '区域类型' = case

when areaType = '省' then areaName + areaType

when areaType = '市' then 'city'

when areaType = '区' then 'area'

else 'other'

end

from ab\_area;

       4、 其他语句

批处理语句go

Use master

Go

延时执行，类似于定时器、休眠等

waitfor delay '00:00:03';--定时三秒后执行

print '定时三秒后执行';

[**SQL Server 触发器**](http://www.cnblogs.com/hoojo/archive/2011/07/20/2111316.html)

触发器是一种特殊类型的存储过程，它不同于之前的我们介绍的存储过程。触发器主要是通过事件进行触发被自动调用执行的。而存储过程可以通过存储过程的名称被调用。

Ø 什么是触发器

    触发器对表进行插入、更新、删除的时候会自动执行的特殊存储过程。触发器一般用在check约束更加复杂的约束上面。触发器和普通的存储过程的区别是：触发器是当对某一个表进行操作。诸如：update、insert、delete这些操作的时候，系统会自动调用执行该表上对应的触发器。SQL Server 2005中触发器可以分为两类：DML触发器和DDL触发器，其中DDL触发器它们会影响多种数据定义语言语句而激发，这些语句有create、alter、drop语句。

    DML触发器分为：

    1、 after触发器（之后触发）

        a、 insert触发器

        b、 update触发器

        c、 delete触发器

    2、 instead of 触发器 （之前触发）

    其中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，且只能定义在表上。而instead of触发器表示并不执行其定义的操作（insert、update、delete）而仅是执行触发器本身。既可以在表上定义instead of触发器，也可以在视图上定义。

    触发器有两个特殊的表：插入表（instered表）和删除表（deleted表）。这两张是逻辑表也是虚表。有系统在内存中创建者两张表，不会存储在数据库中。而且两张表的都是只读的，只能读取数据而不能修改数据。这两张表的结果总是与被改触发器应用的表的结构相同。当触发器完成工作后，这两张表就会被删除。Inserted表的数据是插入或是修改后的数据，而deleted表的数据是更新前的或是删除的数据。

|  |  |  |
| --- | --- | --- |
| 对表的操作 | Inserted逻辑表 | Deleted逻辑表 |
| 增加记录（insert） | 存放增加的记录 | 无 |
| 删除记录（delete） | 无 | 存放被删除的记录 |
| 修改记录（update） | 存放更新后的记录 | 存放更新前的记录 |

    Update数据的时候就是先删除表记录，然后增加一条记录。这样在inserted和deleted表就都有update后的数据记录了。注意的是：触发器本身就是一个事务，所以在触发器里面可以对修改数据进行一些特殊的检查。如果不满足可以利用事务回滚，撤销操作。

Ø 创建触发器

    语法

create trigger tgr\_name  
on table\_name  
with encrypion –加密触发器  
 for update...  
as  
 Transact-SQL

    # 创建insert类型触发器

--创建insert插入类型触发器  
if (object\_id('tgr\_classes\_insert', 'tr') is not null)  
 drop trigger tgr\_classes\_insert  
go  
create trigger tgr\_classes\_insert  
on classes  
 for insert --插入触发  
as  
 --定义变量  
 declare @id int, @name varchar(20), @temp int;  
 --在inserted表中查询已经插入记录信息  
 select @id = id, @name = name from inserted;  
 set @name = @name + convert(varchar, @id);  
 set @temp = @id / 2;   
 insert into student values(@name, 18 + @id, @temp, @id);  
 print '添加学生成功！';  
go  
--插入数据  
insert into classes values('5班', getDate());  
--查询数据  
select \* from classes;  
select \* from student order by id;

     insert触发器，会在inserted表中添加一条刚插入的记录。

    # 创建delete类型触发器

--delete删除类型触发器  
if (object\_id('tgr\_classes\_delete', 'TR') is not null)  
 drop trigger tgr\_classes\_delete  
go  
create trigger tgr\_classes\_delete  
on classes  
 for delete --删除触发  
as  
 print '备份数据中……';   
 if (object\_id('classesBackup', 'U') is not null)  
 --存在classesBackup，直接插入数据  
 insert into classesBackup select name, createDate from deleted;  
 else  
 --不存在classesBackup创建再插入  
 select \* into classesBackup from deleted;  
 print '备份数据成功！';  
go  
--  
--不显示影响行数  
--set nocount on;  
delete classes where name = '5班';  
--查询数据  
select \* from classes;  
select \* from classesBackup;

   delete触发器会在删除数据的时候，将刚才删除的数据保存在deleted表中。

    # 创建update类型触发器

--update更新类型触发器  
if (object\_id('tgr\_classes\_update', 'TR') is not null)  
 drop trigger tgr\_classes\_update  
go  
create trigger tgr\_classes\_update  
on classes  
 for update  
as  
 declare @oldName varchar(20), @newName varchar(20);  
 --更新前的数据  
 select @oldName = name from deleted;  
 if (exists (select \* from student where name like '%'+ @oldName + '%'))  
 begin  
 --更新后的数据  
 select @newName = name from inserted;  
 update student set name = replace(name, @oldName, @newName) where name like '%'+ @oldName + '%';  
 print '级联修改数据成功！';  
 end  
 else  
 print '无需修改student表！';  
go  
--查询数据  
select \* from student order by id;  
select \* from classes;  
update classes set name = '五班' where name = '5班';

     update触发器会在更新数据后，将更新前的数据保存在deleted表中，更新后的数据保存在inserted表中。

    # update更新列级触发器

if (object\_id('tgr\_classes\_update\_column', 'TR') is not null)  
 drop trigger tgr\_classes\_update\_column  
go  
create trigger tgr\_classes\_update\_column  
on classes  
 for update  
as  
 --列级触发器：是否更新了班级创建时间  
 if (update(createDate))  
 begin  
 raisError('系统提示：班级创建时间不能修改！', 16, 11);  
 rollback tran;  
 end  
go  
--测试  
select \* from student order by id;  
select \* from classes;  
update classes set createDate = getDate() where id = 3;  
update classes set name = '四班' where id = 7;

     更新列级触发器可以用update是否判断更新列记录；

    # instead of类型触发器

       instead of触发器表示并不执行其定义的操作（insert、update、delete）而仅是执行触发器本身的内容。

       创建语法

create trigger tgr\_name  
on table\_name  
with encryption  
 instead of update...  
as  
 T-SQL

      # 创建instead of触发器

if (object\_id('tgr\_classes\_inteadOf', 'TR') is not null)  
 drop trigger tgr\_classes\_inteadOf  
go  
create trigger tgr\_classes\_inteadOf  
on classes  
 instead of delete/\*, update, insert\*/  
as  
 declare @id int, @name varchar(20);  
 --查询被删除的信息，病赋值  
 select @id = id, @name = name from deleted;  
 print 'id: ' + convert(varchar, @id) + ', name: ' + @name;  
 --先删除student的信息  
 delete student where cid = @id;  
 --再删除classes的信息  
 delete classes where id = @id;  
 print '删除[ id: ' + convert(varchar, @id) + ', name: ' + @name + ' ] 的信息成功！';  
go  
--test  
select \* from student order by id;  
select \* from classes;  
delete classes where id = 7;

      # 显示自定义消息raiserror

if (object\_id('tgr\_message', 'TR') is not null)  
 drop trigger tgr\_message  
go  
create trigger tgr\_message  
on student  
 after insert, update  
as raisError('tgr\_message触发器被触发', 16, 10);  
go  
--test  
insert into student values('lily', 22, 1, 7);  
update student set sex = 0 where name = 'lucy';  
select \* from student order by id;

    # 修改触发器

alter trigger tgr\_message  
on student  
after delete  
as raisError('tgr\_message触发器被触发', 16, 10);  
go  
--test  
delete from student where name = 'lucy';

    # 启用、禁用触发器

--禁用触发器  
disable trigger tgr\_message on student;  
--启用触发器  
enable trigger tgr\_message on student;

    # 查询创建的触发器信息

--查询已存在的触发器  
select \* from sys.triggers;  
select \* from sys.objects where type = 'TR';  
  
--查看触发器触发事件  
select te.\* from sys.trigger\_events te join sys.triggers t  
on t.object\_id = te.object\_id  
where t.parent\_class = 0 and t.name = 'tgr\_valid\_data';  
  
--查看创建触发器语句  
exec sp\_helptext 'tgr\_message';

    # 示例，验证插入数据

if ((object\_id('tgr\_valid\_data', 'TR') is not null))  
 drop trigger tgr\_valid\_data  
go  
create trigger tgr\_valid\_data  
on student  
after insert  
as  
 declare @age int,  
 @name varchar(20);  
 select @name = s.name, @age = s.age from inserted s;  
 if (@age < 18)  
 begin  
 raisError('插入新数据的age有问题', 16, 1);  
 rollback tran;  
 end  
go  
--test  
insert into student values('forest', 2, 0, 7);  
insert into student values('forest', 22, 0, 7);  
select \* from student order by id;

    # 示例，操作日志

if (object\_id('log', 'U') is not null)  
 drop table log  
go  
create table log(  
 id int identity(1, 1) primary key,  
 action varchar(20),  
 createDate datetime default getDate()  
)  
go  
if (exists (select \* from sys.objects where name = 'tgr\_student\_log'))  
 drop trigger tgr\_student\_log  
go  
create trigger tgr\_student\_log  
on student  
after insert, update, delete  
as  
 if ((exists (select 1 from inserted)) and (exists (select 1 from deleted)))  
 begin  
 insert into log(action) values('updated');  
 end  
 else if (exists (select 1 from inserted) and not exists (select 1 from deleted))  
 begin  
 insert into log(action) values('inserted');  
 end  
 else if (not exists (select 1 from inserted) and exists (select 1 from deleted))  
 begin  
 insert into log(action) values('deleted');  
 end  
go  
--test  
insert into student values('king', 22, 1, 7);  
update student set sex = 0 where name = 'king';  
delete student where name = 'king';  
select \* from log;  
select \* from student order by id;

[**SQL Server 存储过程**](http://www.cnblogs.com/hoojo/archive/2011/07/19/2110862.html)

Transact-SQL中的存储过程，非常类似于Java语言中的方法，它可以重复调用。当存储过程执行一次后，可以将语句缓存中，这样下次执行的时候直接使用缓存中的语句。这样就可以提高存储过程的性能。

Ø 存储过程的概念

    存储过程Procedure是一组为了完成特定功能的SQL语句集合，经编译后存储在数据库中，用户通过指定存储过程的名称并给出参数来执行。

    存储过程中可以包含逻辑控制语句和数据操纵语句，它可以接受参数、输出参数、返回单个或多个结果集以及返回值。

    由于存储过程在创建时即在数据库服务器上进行了编译并存储在数据库中，所以存储过程运行要比单个的SQL语句块要快。同时由于在调用时只需用提供存储过程名和必要的参数信息，所以在一定程度上也可以减少网络流量、简单网络负担。

    1、 存储过程的优点

        A、 存储过程允许标准组件式编程

        存储过程创建后可以在程序中被多次调用执行，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以随时对存储过程进行修改，但对应用程序源代码却毫无影响，从而极大的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T-SQL语句代码，分别被多次执行，那么存储过程要比批处理的执行速度快得多。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存储计划。而批处理的T-SQL语句每次运行都需要预编译和优化，所以速度就要慢一些。

        C、 存储过程减轻网络流量

        对于同一个针对数据库对象的操作，如果这一操作所涉及到的T-SQL语句被组织成一存储过程，那么当在客户机上调用该存储过程时，网络中传递的只是该调用语句，否则将会是多条SQL语句。从而减轻了网络流量，降低了网络负载。

        D、 存储过程可被作为一种安全机制来充分利用

        系统管理员可以对执行的某一个存储过程进行权限限制，从而能够实现对某些数据访问的限制，避免非授权用户对数据的访问，保证数据的安全。

Ø 系统存储过程

    系统存储过程是系统创建的存储过程，目的在于能够方便的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他的系统管理任务。系统存储过程主要存储在master数据库中，以“sp”下划线开头的存储过程。尽管这些系统存储过程在master数据库中，但我们在其他数据库还是可以调用系统存储过程。有一些系统存储过程会在创建新的数据库的时候被自动创建在当前数据库中。

    常用系统存储过程有：

exec sp\_databases; --查看数据库  
exec sp\_tables; --查看表  
exec sp\_columns student;--查看列  
exec sp\_helpIndex student;--查看索引  
exec sp\_helpConstraint student;--约束  
exec sp\_stored\_procedures;  
exec sp\_helptext 'sp\_stored\_procedures';--查看存储过程创建、定义语句  
exec sp\_rename student, stuInfo;--修改表、索引、列的名称  
exec sp\_renamedb myTempDB, myDB;--更改数据库名称  
exec sp\_defaultdb 'master', 'myDB';--更改登录名的默认数据库  
exec sp\_helpdb;--数据库帮助，查询数据库信息  
exec sp\_helpdb master;

    系统存储过程示例：

--表重命名  
exec sp\_rename 'stu', 'stud';  
select \* from stud;  
--列重命名  
exec sp\_rename 'stud.name', 'sName', 'column';  
exec sp\_help 'stud';  
--重命名索引  
exec sp\_rename N'student.idx\_cid', N'idx\_cidd', N'index';  
exec sp\_help 'student';  
  
--查询所有存储过程  
select \* from sys.objects where type = 'P';  
select \* from sys.objects where type\_desc like '%pro%' and name like 'sp%';

Ø 用户自定义存储过程

   1、 创建语法

create proc | procedure pro\_name  
 [{@参数数据类型} [=默认值] [output],  
 {@参数数据类型} [=默认值] [output],  
 ....  
 ]  
as  
 SQL\_statements

   2、 创建不带参数存储过程

--创建存储过程  
if (exists (select \* from sys.objects where name = 'proc\_get\_student'))  
 drop proc proc\_get\_student  
go  
create proc proc\_get\_student  
as  
 select \* from student;  
  
--调用、执行存储过程  
exec proc\_get\_student;

   3、 修改存储过程

--修改存储过程  
alter proc proc\_get\_student  
as  
select \* from student;

   4、 带参存储过程

--带参存储过程  
if (object\_id('proc\_find\_stu', 'P') is not null)  
 drop proc proc\_find\_stu  
go  
create proc proc\_find\_stu(@startId int, @endId int)  
as  
 select \* from student where id between @startId and @endId  
go  
  
exec proc\_find\_stu 2, 4;

   5、 带通配符参数存储过程

--带通配符参数存储过程  
if (object\_id('proc\_findStudentByName', 'P') is not null)  
 drop proc proc\_findStudentByName  
go  
create proc proc\_findStudentByName(@name varchar(20) = '%j%', @nextName varchar(20) = '%')  
as  
 select \* from student where name like @name and name like @nextName;  
go  
  
exec proc\_findStudentByName;  
exec proc\_findStudentByName '%o%', 't%';

   6、 带输出参数存储过程

if (object\_id('proc\_getStudentRecord', 'P') is not null)  
 drop proc proc\_getStudentRecord  
go  
create proc proc\_getStudentRecord(  
 @id int, --默认输入参数  
 @name varchar(20) out, --输出参数  
 @age varchar(20) output--输入输出参数  
)  
as  
 select @name = name, @age = age from student where id = @id and sex = @age;  
go  
  
--   
declare @id int,  
 @name varchar(20),  
 @temp varchar(20);  
set @id = 7;   
set @temp = 1;  
exec proc\_getStudentRecord @id, @name out, @temp output;  
select @name, @temp;  
print @name + '#' + @temp;

   7、 不缓存存储过程

--WITH RECOMPILE 不缓存  
if (object\_id('proc\_temp', 'P') is not null)  
 drop proc proc\_temp  
go  
create proc proc\_temp  
with recompile  
as  
 select \* from student;  
go  
  
exec proc\_temp;

   8、 加密存储过程

--加密WITH ENCRYPTION   
if (object\_id('proc\_temp\_encryption', 'P') is not null)  
 drop proc proc\_temp\_encryption  
go  
create proc proc\_temp\_encryption  
with encryption  
as  
 select \* from student;  
go  
  
exec proc\_temp\_encryption;  
exec sp\_helptext 'proc\_temp';  
exec sp\_helptext 'proc\_temp\_encryption';

   9、 带游标参数存储过程

if (object\_id('proc\_cursor', 'P') is not null)  
 drop proc proc\_cursor  
go  
create proc proc\_cursor  
 @cur cursor varying output  
as  
 set @cur = cursor forward\_only static for  
 select id, name, age from student;  
 open @cur;  
go  
--调用  
declare @exec\_cur cursor;  
declare @id int,  
 @name varchar(20),  
 @age int;  
exec proc\_cursor @cur = @exec\_cur output;--调用存储过程  
fetch next from @exec\_cur into @id, @name, @age;  
while (@@fetch\_status = 0)  
begin  
 fetch next from @exec\_cur into @id, @name, @age;  
 print 'id: ' + convert(varchar, @id) + ', name: ' + @name + ', age: ' + convert(char, @age);  
end  
close @exec\_cur;  
deallocate @exec\_cur;--删除游标

   10、 分页存储过程

---存储过程、row\_number完成分页  
if (object\_id('pro\_page', 'P') is not null)  
 drop proc proc\_cursor  
go  
create proc pro\_page  
 @startIndex int,  
 @endIndex int  
as  
 select count(\*) from product  
;   
 select \* from (  
 select row\_number() over(order by pid) as rowId, \* from product   
 ) temp  
 where temp.rowId between @startIndex and @endIndex  
go  
--drop proc pro\_page  
exec pro\_page 1, 4  
--  
--分页存储过程  
if (object\_id('pro\_page', 'P') is not null)  
 drop proc pro\_stu  
go  
create procedure pro\_stu(  
 @pageIndex int,  
 @pageSize int  
)  
as  
 declare @startRow int, @endRow int  
 set @startRow = (@pageIndex - 1) \* @pageSize +1  
 set @endRow = @startRow + @pageSize -1  
 select \* from (  
 select \*, row\_number() over (order by id asc) as number from student   
 ) t  
 where t.number between @startRow and @endRow;  
  
exec pro\_stu 2, 2;

Ø Raiserror

Raiserror返回用户定义的错误信息，可以指定严重级别，设置系统变量记录所发生的错误。

   语法如下：

Raiserror({msg\_id | msg\_str | @local\_variable}  
 {, severity, state}  
 [,argument[,…n]]  
 [with option[,…n]]  
)

   # msg\_id:在sysmessages系统表中指定的用户定义错误信息

   # msg\_str:用户定义的信息，信息最大长度在2047个字符。

   # severity：用户定义与该消息关联的严重级别。当使用msg\_id引发使用sp\_addmessage创建的用户定义消息时，raiserror上指定严重性将覆盖sp\_addmessage中定义的严重性。

    任何用户可以指定0-18直接的严重级别。只有sysadmin固定服务器角色常用或具有alter trace权限的用户才能指定19-25直接的严重级别。19-25之间的安全级别需要使用with log选项。

   # state：介于1至127直接的任何整数。State默认值是1。

raiserror('is error', 16, 1);  
select \* from sys.messages;  
--使用sysmessages中定义的消息  
raiserror(33003, 16, 1);  
raiserror(33006, 16, 1);