# SQL常用函数总结

**一、字符转换函数  
1**、ASCII()  
返回字符表达式最左端字符的ASCII 码值。在ASCII（）函数中，纯数字的字符串可不用‘’括起来，但含其它字符的字符串必须用‘’括起来使用，否则会出错。  
  
**2**、CHAR()  
将ASCII 码转换为字符。如果没有输入0 ~ **255** 之间的ASCII 码值，CHAR（） 返回NULL 。  
  
**3**、LOWER()和UPPER()  
LOWER()将字符串全部转为小写；UPPER()将字符串全部转为大写。  
  
**4**、STR()  
把数值型数据转换为字符型数据。  
STR (<float\_expression>[，length[， <decimal>]])  
length 指定返回的字符串的长度，decimal 指定返回的小数位数。如果没有指定长度，缺省的length 值为10， decimal 缺省值为0。  
当length 或者decimal 为负值时，返回NULL；  
当length 小于小数点左边（包括符号位）的位数时，返回length 个\*；  
先服从length ，再取decimal ；  
当返回的字符串位数小于length ，左边补足空格。  
  
**二、去空格函数  
1**、LTRIM() 把字符串头部的空格去掉。  
  
**2**、RTRIM() 把字符串尾部的空格去掉。  
  
**三、取子串函数  
1**、left()   
LEFT (<character\_expression>， <integer\_expression>)  
返回character\_expression 左起 integer\_expression 个字符。  
  
**2**、RIGHT()   
RIGHT (<character\_expression>， <integer\_expression>)  
返回character\_expression 右起 integer\_expression 个字符。  
  
**3**、SUBSTRING()  
SUBSTRING (<expression>， <starting\_ position>， length)  
返回从字符串左边第starting\_ position 个字符起length个字符的部分。  
  
**四、字符串比较函数**  
**1**、CHARINDEX()  
返回字符串中某个指定的子串出现的开始位置。  
CHARINDEX (<’substring\_expression’>， <expression>)  
其中substring \_expression 是所要查找的字符表达式，expression 可为字符串也可为列名表达式。如果没有发现子串，则返回0 值。  
此函数不能用于TEXT 和IMAGE 数据类型。  
  
**2**、PATINDEX()  
返回字符串中某个指定的子串出现的开始位置。  
PATINDEX (<’%substring \_expression%’>， <column\_ name>)其中子串表达式前后必须有百分号“%”否则返回值为0。  
与CHARINDEX 函数不同的是，PATINDEX函数的子串中可以使用通配符，且此函数可用于CHAR、 VARCHAR 和TEXT 数据类型。  
  
**五、字符串操作函数**  
**1**、QUOTENAME()  
返回被特定字符括起来的字符串。  
QUOTENAME (<’character\_expression’>[， quote\_ character]) 其中quote\_ character 标明括字符串所用的字符，缺省值为“[]”。  
  
**2**、REPLICATE()  
返回一个重复character\_expression 指定次数的字符串。  
REPLICATE (character\_expression integer\_expression) 如果integer\_expression 值为负值，则返回NULL 。  
  
**3**、REVERSE()  
将指定的字符串的字符排列顺序颠倒。  
REVERSE (<character\_expression>) 其中character\_expression 可以是字符串、常数或一个列的值。  
  
**4**、REPLACE()  
返回被替换了指定子串的字符串。  
REPLACE (<string\_expression1>， <string\_expression2>， <string\_expression3>) 用string\_expression3 替换在string\_expression1 中的子串string\_expression2。  
  
**5**、SPACE()  
返回一个有指定长度的空白字符串。  
SPACE (<integer\_expression>) 如果integer\_expression 值为负值，则返回NULL 。  
  
**6**、STUFF()  
用另一子串替换字符串指定位置、长度的子串。  
STUFF (<character\_expression1>， <start\_ position>， <length>，<character\_expression2>)  
如果起始位置为负或长度值为负，或者起始位置大于character\_expression1 的长度，则返回NULL 值。  
如果length 长度大于character\_expression1 中 start\_ position 以右的长度，则character\_expression1 只保留首字符。  
  
**六、数据类型转换函数**  
**1**、CAST()  
CAST (<expression> AS <data\_ type>[ length ])  
  
**2**、CONVERT()  
CONVERT (<data\_ type>[ length ]， <expression> [， style])  
  
**1**）data\_type为SQL Server系统定义的数据类型，用户自定义的数据类型不能在此使用。  
**2**）length用于指定数据的长度，缺省值为30。  
**3**）把CHAR或VARCHAR类型转换为诸如INT或SAMLLINT这样的INTEGER类型、结果必须是带正号或负号的数值。  
**4**）TEXT类型到CHAR或VARCHAR类型转换最多为8000个字符，即CHAR或VARCHAR数据类型是最大长度。  
**5**）IMAGE类型存储的数据转换到BINARY或VARBINARY类型，最多为8000个字符。  
**6**）把整数值转换为MONEY或SMALLMONEY类型，按定义的国家的货币单位来处理，如人民币、美元、英镑等。  
**7**）BIT类型的转换把非零值转换为1，并仍以BIT类型存储。  
**8**）试图转换到不同长度的数据类型，会截短转换值并在转换值后显示“+”，以标识发生了这种截断。  
**9**）用CONVERT（）函数的style 选项能以不同的格式显示日期和时间。style 是将DATATIME 和SMALLDATETIME 数据转换为字符串时所选用的由SQL Server 系统提供的转换样式编号，不同的样式编号有不同的输出格式。  
  
**七、日期函数  
1**、day(date\_expression)   
返回date\_expression中的日期值  
  
**2**、month(date\_expression)  
返回date\_expression中的月份值  
  
**3**、year(date\_expression)  
返回date\_expression中的年份值  
  
**4**、DATEADD()  
DATEADD (<datepart>， <number>， <date>)  
返回指定日期date 加上指定的额外日期间隔number 产生的新日期。  
  
**5**、DATEDIFF()  
DATEDIFF (<datepart>， <date1>， <date2>)  
返回两个指定日期在datepart 方面的不同之处，即date2 超过date1的差距值，其结果值是一个带有正负号的整数值。  
  
**6**、DATENAME()  
DATENAME (<datepart>， <date>)  
以字符串的形式返回日期的指定部分此部分。由datepart 来指定。  
  
**7**、DATEPART()  
DATEPART (<datepart>， <date>)  
以整数值的形式返回日期的指定部分。此部分由datepart 来指定。  
DATEPART (dd， date) 等同于DAY (date)  
DATEPART (mm， date) 等同于MONTH (date)  
DATEPART (yy， date) 等同于YEAR (date)  
  
**8**、GETDATE()  
以DATETIME 的缺省格式返回系统当前的日期和时间。  
 **9、Sql Server 中一个非常强大的日期格式化函数**Select CONVERT(varchar(100), GETDATE(), 0): 05 16 2006 10:57AM  
Select CONVERT(varchar(100), GETDATE(), 1): 05/16/06  
Select CONVERT(varchar(100), GETDATE(), 2): 06.05.16  
Select CONVERT(varchar(100), GETDATE(), 3): 16/05/06  
Select CONVERT(varchar(100), GETDATE(), 4): 16.05.06  
Select CONVERT(varchar(100), GETDATE(), 5): 16-05-06  
Select CONVERT(varchar(100), GETDATE(), 6): 16 05 06  
Select CONVERT(varchar(100), GETDATE(), 7): 05 16, 06  
Select CONVERT(varchar(100), GETDATE(), 8): 10:57:46  
Select CONVERT(varchar(100), GETDATE(), 9): 05 16 2006 10:57:46:827AM  
Select CONVERT(varchar(100), GETDATE(), 10): 05-16-06  
Select CONVERT(varchar(100), GETDATE(), 11): 06/05/16  
Select CONVERT(varchar(100), GETDATE(), 12): 060516  
Select CONVERT(varchar(100), GETDATE(), 13): 16 05 2006 10:57:46:937  
Select CONVERT(varchar(100), GETDATE(), 14): 10:57:46:967  
Select CONVERT(varchar(100), GETDATE(), 20): 2006-05-16 10:57:47  
Select CONVERT(varchar(100), GETDATE(), 21): 2006-05-16 10:57:47.157  
Select CONVERT(varchar(100), GETDATE(), 22): 05/16/06 10:57:47 AM  
Select CONVERT(varchar(100), GETDATE(), 23): 2006-05-16  
Select CONVERT(varchar(100), GETDATE(), 24): 10:57:47  
Select CONVERT(varchar(100), GETDATE(), 25): 2006-05-16 10:57:47.250  
Select CONVERT(varchar(100), GETDATE(), 100): 05 16 2006 10:57AM  
Select CONVERT(varchar(100), GETDATE(), 101): 05/16/2006  
Select CONVERT(varchar(100), GETDATE(), 102): 2006.05.16  
Select CONVERT(varchar(100), GETDATE(), 103): 16/05/2006  
Select CONVERT(varchar(100), GETDATE(), 104): 16.05.2006  
Select CONVERT(varchar(100), GETDATE(), 105): 16-05-2006  
Select CONVERT(varchar(100), GETDATE(), 106): 16 05 2006  
Select CONVERT(varchar(100), GETDATE(), 107): 05 16, 2006  
Select CONVERT(varchar(100), GETDATE(), 108): 10:57:49  
Select CONVERT(varchar(100), GETDATE(), 109): 05 16 2006 10:57:49:437AM  
Select CONVERT(varchar(100), GETDATE(), 110): 05-16-2006  
Select CONVERT(varchar(100), GETDATE(), 111): 2006/05/16  
Select CONVERT(varchar(100), GETDATE(), 112): 20060516  
Select CONVERT(varchar(100), GETDATE(), 113): 16 05 2006 10:57:49:513  
Select CONVERT(varchar(100), GETDATE(), 114): 10:57:49:547  
Select CONVERT(varchar(100), GETDATE(), 120): 2006-05-16 10:57:49  
Select CONVERT(varchar(100), GETDATE(), 121): 2006-05-16 10:57:49.700  
Select CONVERT(varchar(100), GETDATE(), 126): 2006-05-16T10:57:49.827  
Select CONVERT(varchar(100), GETDATE(), 130): 18 ???? ?????? 1427 10:57:49:907AM  
Select CONVERT(varchar(100), GETDATE(), 131): 18/04/1427 10:57:49:920AM

常用：  
Select CONVERT(varchar(100), GETDATE(), 8): 10:57:46  
Select CONVERT(varchar(100), GETDATE(), 24): 10:57:47  
Select CONVERT(varchar(100), GETDATE(), 108): 10:57:49  
Select CONVERT(varchar(100), GETDATE(), 12): 060516  
Select CONVERT(varchar(100), GETDATE(), 23): 2006-05-16

**八、统计函数**

avg() -返回的平均价值   
count() -返回的行数   
first() -返回第一个值   
last() -返回最后一个值   
max()-返回的最大价值   
min() -返回最小的价值   
total() -返回的总和  
  
**九、数学函数**  
abs(numeric\_expr)                   求绝对值  
ceiling(numeric\_expr)                取大于等于指定值的最小整数  
exp(float\_expr)                      取指数  
floor(numeric\_expr)                   小于等于指定值得最大整数  
pi()                                  3.1415926.........  
power(numeric\_expr,power)            返回power次方         
rand([int\_expr])                     随机数产生器  
round(numeric\_expr,int\_expr)         安int\_expr规定的精度四舍五入  
sign(int\_expr)                      根据正数,0,负数,,返回+1,0,-1  
sqrt(float\_expr)                     平方根  
  
**十、系统函数**suser\_name()       用户登录名  
user\_name()        用户在数据库中的名字  
user               用户在数据库中的名字  
show\_role()        对当前用户起作用的规则  
db\_name()          数据库名                       
object\_name(obj\_id) 数据库对象名        
col\_name(obj\_id,col\_id) 列名          
col\_length(objname,colname)  列长度   
valid\_name(char\_expr)     是否是有效标识符   
  
**十一、以上函数的部分实例**1:replace 函数  
第一个参数你的字符串，第二个参数你想替换的部分，第三个参数你要替换成什么  
select replace('lihan','a','b')  
                                                                    
-----------------------------  
lihbn

（所影响的行数为 1 行）  
=========================================================  
2:substring函数  
第一个参数你的字符串，第二个是开始替换位置，第三个结束替换位置  
select substring('lihan',0,3);  
-----   
li

（所影响的行数为 1 行）  
=========================================================  
3:charindex函数  
第一个参数你要查找的char，第二个参数你被查找的字符串 返回参数一在参数二的位置  
select  charindex('a','lihan')  
-----------   
4

（所影响的行数为 1 行）

===========================================================  
4：ASCII函数  
返回字符表达式中最左侧的字符的 ASCII 代码值。  
select ASCII('lihan')  
-----------   
108

（所影响的行数为 1 行）

================================================================  
5：nchar函数  
根据 Unicode 标准的定义，返回具有指定的整数代码的 Unicode 字符。  
参数是介于 0 与 65535 之间的正整数。如果指定了超出此范围的值，将返回 NULL。  
select nchar(3213)  
----   
unicode字符

（所影响的行数为 1 行）

=========================================================  
6：soundex  
返回一个由四个字符组成的代码 (SOUNDEX)，用于评估两个字符串的相似性。  
SELECT SOUNDEX ('lihan'), SOUNDEX ('lihon');  
----- -----   
L546  L542

（所影响的行数为 1 行）  
=========================================================  
7：char  
参数为介于 0 和 255 之间的整数。如果该整数表达式不在此范围内，将返回 NULL 值。  
SELECT char(125)  
----   
}

（所影响的行数为 1 行）

==========================================================  
8：str函数  
第一个参数必须为数字，第二个参数表示转化成char型占的位置，小于参数一位置返回\*，大于右对齐  
SELECT str(12345,3)  
----   
\*\*\*

（所影响的行数为 1 行）

SELECT str(12345,12)  
------------   
       12345

（所影响的行数为 1 行）  
===========================================================  
9：difference函数  
返回一个整数值，指示两个字符表达式的 SOUNDEX 值之间的差异。  
返回的整数是 SOUNDEX 值中相同字符的个数。返回的值从 0 到 4 不等：0 表示几乎不同或完全不同，4 表示几乎相同或完全相同。  
SELECT difference('lihan','liha')  
-----------   
3

（所影响的行数为 1 行）

==================================================================  
10：stuff函数（四个参数）  
函数将字符串插入另一字符串。它在第一个字符串中从开始位置删除指定长度的字符；然后将第二个字符串插入第一个字符串的开始位置。  
SELECT stuff('lihan',2,3,'lihan')  
--------   
llihann

（所影响的行数为 1 行）  
===============================================================

11：left函数  
返回最左边N个字符，由参数决定  
select left('lihan',4)  
-----   
liha

（所影响的行数为 1 行）  
================================================================

12 right函数  
返回最右边N个字符，由参数决定  
select right('lihan',4)  
-----   
ihan

（所影响的行数为 1 行）  
================================================================

13：replicate函数  
我的认为是把参数一复制参数二次  
select replicate('lihan',4)  
--------------------   
lihanlihanlihanlihan

（所影响的行数为 1 行）

================================================================

14：len函数  
返回参数长度  
select len('lihan')  
-----------   
5

（所影响的行数为 1 行）

================================================================  
15：reverse函数  
反转字符串  
select reverse('lihan')  
-----   
nahil

（所影响的行数为 1 行）

=================================================================

16：lower和upper函数  
参数大小写转化  
select lower(upper('lihan'))  
--------------------   
lihan

（所影响的行数为 1 行）

====================================================================

17：ltrim和rtrim函数  
删除左边空格和右面空格  
select ltrim('    lihan                 ')  
--------------------------   
lihan

（所影响的行数为 1 行）  
select rtrim('    lihan')  
---------   
    lihan

（所影响的行数为 1 行）

**追加：**排名函数是SQL Server2005新加的功能。在SQL Server2005中有如下四个排名函数：  
1. row\_number  
2. rank  
3. dense\_rank  
4. ntile      
    下面分别介绍一下这四个排名函数的功能及用法。在介绍之前假设有一个t\_table表，表结构与表中的数据如图1所示：

  
图1

其中field1字段的类型是int，field2字段的类型是varchar

一、row\_number

    row\_number函数的用途是非常广泛，这个函数的功能是为查询出来的每一行记录生成一个序号。row\_number函数的用法如下面的SQL语句所示：

select row\_number() over(order by field1) as row\_number,\* from t\_table

    上面的SQL语句的查询结果如图2所示。

  
图2

    其中row\_number列是由row\_number函数生成的序号列。在使用row\_number函数是要使用over子句选择对某一列进行排序，然后才能生成序号。

    实际上，row\_number函数生成序号的基本原理是先使用over子句中的排序语句对记录进行排序，然后按着这个顺序生成序号。over子句中的order by子句与SQL语句中的order by子句没有任何关系，这两处的order by 可以完全不同，如下面的SQL语句所示：

select row\_number() over(order by field2 desc) as row\_number,\* from t\_table order by field1 desc

    上面的SQL语句的查询结果如图3所示。

  
图3

    我们可以使用row\_number函数来实现查询表中指定范围的记录，一般将其应用到Web应用程序的分页功能上。下面的SQL语句可以查询t\_table表中第2条和第3条记录：

with t\_rowtable  
as  
(  
    select row\_number() over(order by field1) as row\_number,\* from t\_table  
)  
select \* from t\_rowtable where row\_number>**1** and row\_number < **4** order by field1

    上面的SQL语句的查询结果如图4所示。

  
图4

    上面的SQL语句使用了CTE，关于CTE的介绍将读者参阅[《SQL Server2005杂谈（1）：使用公用表表达式（CTE）简化嵌套SQL》](http://www.blogjava.net/nokiaguy/archive/2009/nokiaguy/archive/2009/01/31/252787.html)。  
    另外要注意的是，如果将row\_number函数用于分页处理，over子句中的order by 与排序记录的order by 应相同，否则生成的序号可能不是有续的。  
    当然，不使用row\_number函数也可以实现查询指定范围的记录，就是比较麻烦。一般的方法是使用颠倒Top来实现，例如，查询t\_table表中第2条和第3条记录，可以先查出前3条记录，然后将查询出来的这三条记录按倒序排序，再取前2条记录，最后再将查出来的这2条记录再按倒序排序，就是最终结果。SQL语句如下：

select \* from (select top **2** \* from( select top **3** \* from t\_table order by field1) a order by field1 desc) b order by field1

  
                                                                图5   
    上面的SQL语句查询出来的结果如图5所示。

    这个查询结果除了没有序号列row\_number，其他的与图4所示的查询结果完全一样。

二、rank

    rank函数考虑到了over子句中排序字段值相同的情况，为了更容易说明问题，在t\_table表中再加一条记录，如图6所示。  
                          
                                                          图6   
    在图6所示的记录中后三条记录的field1字段值是相同的。如果使用rank函数来生成序号，这3条记录的序号是相同的，而第4条记录会根据当前的记录 数生成序号，后面的记录依此类推，也就是说，在这个例子中，第4条记录的序号是4，而不是2。rank函数的使用方法与row\_number函数完全相 同，SQL语句如下：

select rank() over(order by field1),\* from t\_table order by field1

  
                                                                      图7

    上面的SQL语句的查询结果如图7所示。

三、dense\_rank

    dense\_rank函数的功能与rank函数类似，只是在生成序号时是连续的，而rank函数生成的序号有可能不连续。如上面的例子中如果使用dense\_rank函数，第4条记录的序号应该是2，而不是4。如下面的SQL语句所示：

select dense\_rank() over(order by field1),\* from t\_table order by field1

    上面的SQL语句的查询结果如图8所示。

  
图8

    读者可以比较图7和图8所示的查询结果有什么不同

四、ntile  
    ntile函数可以对序号进行分组处理。这就相当于将查询出来的记录集放到指定长度的数组中，每一个数组元素存放一定数量的记录。ntile函数为每条记 录生成的序号就是这条记录所有的数组元素的索引（从1开始）。也可以将每一个分配记录的数组元素称为“桶”。ntile函数有一个参数，用来指定桶数。下 面的SQL语句使用ntile函数对t\_table表进行了装桶处理：

select ntile(**4**) over(order by field1) as bucket,\* from t\_table

  
                                                               图9

    上面的SQL语句的查询结果如图9所示。

    由于t\_table表的记录总数是6，而上面的SQL语句中的ntile函数指定了桶数为4。

    也许有的读者会问这么一个问题，SQL Server2005怎么来决定某一桶应该放多少记录呢？可能t\_table表中的记录数有些少，那么我们假设t\_table表中有59条记录，而桶数是5，那么每一桶应放多少记录呢？

    实际上通过两个约定就可以产生一个算法来决定哪一个桶应放多少记录，这两个约定如下：

1. 编号小的桶放的记录不能小于编号大的桶。也就是说，第1捅中的记录数只能大于等于第2桶及以后的各桶中的记录。

2. 所有桶中的记录要么都相同，要么从某一个记录较少的桶开始后面所有捅的记录数都与该桶的记录数相同。也就是说，如果有个桶，前三桶的记录数都是10，而第4捅的记录数是6，那么第5桶和第6桶的记录数也必须是6。

    根据上面的两个约定，可以得出如下的算法：

    // mod表示取余，div表示取整   
    if(记录总数 mod 桶数 == 0)  
    {  
        recordCount = 记录总数 div 桶数;  
        将每桶的记录数都设为recordCount  
    }   
    else  
    {  
        recordCount1 = 记录总数 div 桶数 + 1;  
        int n = 1;  //  n表示桶中记录数为recordCount1的最大桶数  
        m = recordCount1 \* n;  
        while(((记录总数 - m)  mod  (桶数 -  n))  != 0 )  
        {  
            n++;  
            m = recordCount1 \* n;  
        }   
        recordCount2 = (记录总数 - m) div  (桶数 - n);  
        将前n个桶的记录数设为recordCount1  
        将n + 1个至后面所有桶的记录数设为recordCount2  
    }

    根据上面的算法，如果记录总数为59，桶数为5，则前4个桶的记录数都是12，最后一个桶的记录数是11。

    如果记录总数为53，桶数为5，则前3个桶的记录数为11，后2个桶的记录数为10。

    就拿本例来说，记录总数为6，桶数为4，则会算出recordCount1的值为2，在结束while循环后，会算出recordCount2的值是1，因此，前2个桶的记录是2，后2个桶的记录是1。