T-SQL函数的类别和描述：

|  |  |
| --- | --- |
| 函数类别 | 作用 |
| 聚合函数 | 返回一个标量值，表示在某个值域上的聚合，应用于特定的聚合选择或者汇总 |
| 配置变量 | 返回SQL Server执行环境的信息。这些信息可用于给对象编程 |
| 转换 | 将值从一种数据类型转换为另一种类型。也用于将字符格式化为日期、时间与数值 |
| 游标 | 在迭代游标时，用过程方式遍历结果集中的行 |
| 日期和时间 | 从日期值中解析出日期与时间部分，也用于对日期/时间值进行比较和操作 |
| 数学 | 执行各种通用或专用的数学运算，可用于代数、三角、统计、估算与财政运算 |
| 元数据 | 返回SQL Server配置细节信息，以及服务器和数据库设置细节信息 |
| 等级 | 在有序的结果集中枚举前几个值 |
| 安全性 | 返回SQL Server用户的角色成员与权限信息，也包含一组管理事件与跟踪的函数 |
| 字符串处理 | 用于解析、替换、操纵字符值 |
| 系统 | 执行许多不同任务的工具函数。包括值比较、值类型侧试。这个类别也包罗了其他功能 |
| 系统统计 | 揭示数据库系统消耗与环境信息的管理工具 |

函数的目标是返回一个值。大多数函数都返回一个标量值(scalar value)，标量值代表一个数据单元或一个简.单值。实际上，函数可以返回任何数据类型，包括表、游标等可返回完整的多行结果集的类型。

在T-SQL中，一般用SELECT语句来返回值。如果需要从查询中返回一个值，就可以把SELECT当成输出运算符，而不用使用等号:

SELECT Function()

对于SQL函数而言，参数表示输入变量或者值的占位符。函数可以有任意个参数，有些参数是必须的，而有些参数是可选的。可选参数通常被置于以逗号隔开的参数表的末尾，以便于在函数调用中去除不需要的参数。

由于数据库引擎的内部工作机制，SQL Server必须根据所谓的确定性，将函数分成两个不同的组。**如果函数的输出只与输入参数的值相关，而与其他外部因素无关，这个函数就是确定性函数**。**如果函数的输出基于环境条件，或者产生随机或者依赖结果的算法，这个函数就是非确定性的**。为什么要把看起来简单的事弄得如此复杂呢?主要原因是非确定性函数与全局变量不能在一些数据库编程对象中使用(如用户自定义函数)。部分原因是SQL Server缓存与预编译可执行对象的方式。例如，即席查询可以使用任何函数，不过如果打算构建先进的、可重用的编程对象，理解这种区别很重要。

在函数中使用用户变量：

变量既可以用于输入，也可用于输出。 在T-SQL中，用户变量以@符号开头，用于声明为特定的数据类型。**可以使用SET或者SELECT语句给变量赋值。**

SET @Resut = SORT(@Num);SELECT @Result1 = SORT(@Num1), @Result2 = SORT(@Num2) --SELECT 可以在一个操作内同时给多个变量赋值

在服务器的资源耗费上，**用一个SELECT语句给多个变量赋值一般比用多个SET命令的效率要高**。将一个甚至多个值选进参数的限制是，对变量的赋值不能和数据检索操作同时进行。

**聚合函数：**

AVG()函数用于返回一组数值中所有非空数值的平均值。

COUNT()函数用于返回一个列内所有非空值的个数，这是一个整型值。

**配置变量：**

配置变量不是函数，不过它们的用法和系统函数相同。每个全局变量都能够返回SQL Server执行环境的标量信息。

**@@ERROR**

这个变量包含当前连接发生的**最后一次**错误的代码。在执行的语句没有错误时@@ERROR变量的值是0。出现标准错误时，错误是由数据库引擎引发的。**所有的标*准错误代码*与*消息*都保存在sys.messages系统视图中**，可以使用如下脚本查询:

SELECT \* FROM sys.messages

定制错误可以通过调用**RAISERROR**语句来手动引发，并调用**sp\_addmessage**系统存储过程将其添加到sys.messages表中。

即席查询成功检索@@ERROR的值后，@@ERROR的值将返回0， 因为@@ERROR只保了上次执行的语句的错误代码。如果希望检索更多的错误信息，可以使用如下脚本从**sysmessages**视图中得到:

SELECT \* FROM master.dbo.sysmessages WHERE error = **@@ERROR**

**@@TOTAL\_ERROR**

这个变量用于记录从打开当前连接开始发生的总错误次数。和@@ERROR一样。它对每个用户会话是唯一的，并将在连接关闭时被重置。

**@@TOTAL\_READ**

这个变量记录从打开当前连接时开始计算的磁盘读取总数。

**错误函数**

返回所有错误数据的更好办法是使用错误函数。这些函数返回的信息可以存储在错误跟踪表中，以供错误审查。错误函数嵌套在错误处理例程中。*通过使用嵌套在BEGIN TRY 和 END TRY语句中的代码快， 后跟一个放在 BEGIN CATCH 和 END CATCH 语句中的代码快就可以实现错误处理。*所谓的错误捕捉其实就是这个意思。在编写错误处理代码时，必须要把这些代码放在会引发系统错误的catch代码快中。

下列几个错误函数用于返回错误的特定信息：  
ERROR\_MESSAGE() ——返回错误的描述。  
ERROR\_NUMBER() ——返回错误号。  
ERROR\_SEVERITY() ——返回错误的严重级别。错误的严重级别是0~25的整数。

[](http://images2015.cnblogs.com/blog/455983/201509/455983-20150912001727575-1408878290.png)

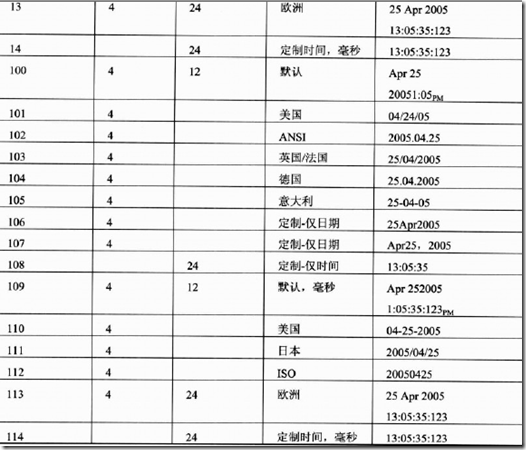
ERROR\_STATE() ——返回错误的状态号。错误状态是一个整数，可以唯一地表时系统错误的原因。  
ERROR\_LINE() ——返回例程中导致出错的行号。  
ERROR\_PROCEDURE() ——返回发生错误的存储过程名或触发器名。

**转换函数**

数据类型转换可以通过CAST()和CONVERT()函数来实现。对于简单类型的转换，两者功能相同，只是语法不同。CASR()函数一般更容易使用，功能也更简单。CONVERT()函数的优点是可以格式化日期和数值。 CAST()和CONVERT()都不能实现四舍五入和或截断操作。*将小数转换为decimal类型交特殊。decimal的精度和小数默认为18和0，转换数据时如果在decimal类型中不提供这两个值， SQL Server会截断数字的小数部分而不报错。*

**CONVERT()函数**

与CAST()相比，CONVERT()具有一些改进的功能，它可以返回经过格式化的字符串值，且可以把日期格式化成很多形式。有28种预定义的符合各种国际和特殊要求的日期与时间输出格式。

[](http://images2015.cnblogs.com/blog/455983/201509/455983-20150912001734715-278330208.png)[](http://images2015.cnblogs.com/blog/455983/201509/455983-20150912001740200-2104859290.png)

*格式代码0，1和2也可用于数字类型，它们对小数与千位分隔符格式产生影响*。而不同的数据类型所受的影响是不一样的。一般来说，使用格式代码0(或者不指定这个参数的值)，将返回该数据类型最惯用的格式。使用1或者2通常显示更为详细或者史精确的值。以下例子使用格式代码1：

[复制代码](javascript:void(0);)

DECLARE @Num Money

SET @Num=**1234**,**56**

SELECT CONVERT(varchar(**50**), @Num, **1**)

--1,234.56

[复制代码](javascript:void(0);)

**STR()函数**

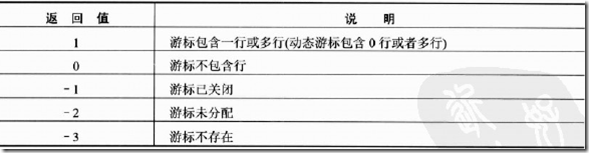
这是一个将数字转换为字符串的快捷函数。这个函数有3个参数：数值、总长度和小数位数。如果数字的整数位数和小数位数(要加上小数点占用的一个字符)的总和小于总长度，对结果中左边的字符将用空格填充。

**游标函数和变量**

游标可以处理多行数据，在过程循环中一次访问一行。和基于集合的高效操作相比，这个功能*对系统资源的消耗更大*。可以用一个函数和两个全局变量来管理游标操作。

**CURSOR\_STATUS()函数**

返回一个整数值，表示传递给这个函数的游标类型变量的状态。

[](http://images2015.cnblogs.com/blog/455983/201509/455983-20150912001744028-613941567.png)  
**@@CURSOR\_ROWS全局变量**

这个变量是一个整型值，表示在当前连接中打开的游标中的行数。根据游标类型，这个值也能不代表结果集中的实际行数。

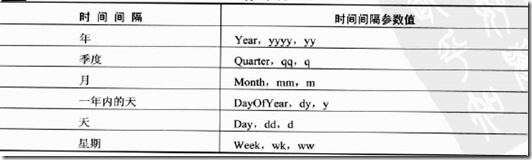
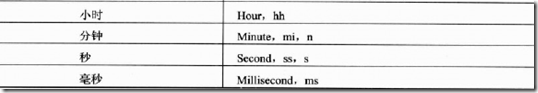
**@@FETCH\_STATUS全局变量**这个变量是一个标记，用于表时当前游标指针的状态。这个变量主要用来判断某行是否存在，以及在执行了FETCH\_NEXT语句后，是否已执行到结果集的尾部。打开游标时，@@FETCH\_STATUS变量值为-1。一旦把第一个值放在游标中，@@FETCH\_STATUS变虽值就变成0。当不再把更多的行放在游标中时，该变量的值将变回-1。

**日期函数**

|  |  |  |
| --- | --- | --- |
| 数据类型 | 存储 | 范围 |
| datetime | 8字节 | 1/1/1753——12/31/9999 |
| smalldatetime | 4字节 | 1/1/1900——6/6/2079 |

**DATEADD()和DATEDIFF()**

SELECT DATEADD(Day, **90**, '9-9-2015')SELECT DATEDIFF(month, '9-9-2014', '2-12-2014')

[](http://images2015.cnblogs.com/blog/455983/201509/455983-20150912001745856-1086426817.png)  
[](http://images2015.cnblogs.com/blog/455983/201509/455983-20150912001747403-1648951220.png)

**DATEPART()和DATENAME()**

SELECT DATEPART(month, '2015-09-09')--5SELECT DATENAME(month, '2015-09-09')—September

**GETDATE()和GETUTCDATE()**

GETUTCDATE()函数使用服务器上的时区设置来求出UTC时间（格林威治时间）。这两个函数都能精确到3.33毫秒。

**SYSDATETIME()和SYSUTCDATETIME()**

SQL Server 2008以上版本函数。等价于GETDATE()和GETUTCDATE()，但返回的是新的datetime2数据类型，精确到100纳秒。

**DAY()、MONTH()和YEAR()**

这三个函数分别返回以整数表示的datetime或者smalldatetime类型值的日、月、年。它们的用途很广泛，如可以创建独特的个性化日期格式。假设需要创建一个自定义的日期值作为字符串，通过将这三个函数的输出结果转换成字符类型，然后进行连接操作，就可以对输出结果以任何形式进行组合了。

**字符串操作函数**

字符串函数可以解析、替换、操纵字符型值。

**ASCII()、CHAR()、UNICODE()和NCHAR()**

这四个函l数是相似的，它们都可以在*字符*和*字符的标准数字表示*之间转换。美国标准信息交换码(ASCII)标准字符集包含128个字母、数字和标点符号。Unicode支持多种语言特定的字符集。每个Unicode字符需要2个字节的存储空间，是ASCII字符的两倍。使用2个字就可以表示超过65 000个不同的字符。SQL Server同时支持ASCII与Unicode两种标准。

ASCII()和CHAR()是两个基于ASCII的函数，这两个函数可将计算机上应用的每个字符表时为数字。

要确定代表一个字符的数字是什么，就应给ASCII()函数传送只包含一个字符的字符串，如下:

SELECT ASCII(‘A’)

如要将一个己知数字转换为字符，使用CHAR()函数即可:

SELECT ASCII(**65**)

UNICODE()函数是ASCII()的Unicod等价函数，NCHAR()函数和CHAR()数的功能相同，只不过NCHAR()是用于unicod字符的。SQL Server的 nchar 和 nvarchar 类型能存储任何unicod字符，可以和这两个函数一起使用。对于特别大的值，ntext 类型和 nvarchar(max) 类型也支持Unicod字符。

**CHARINDEX()和PATINDEX()**  
CHARINDEX()是原始的SQL函数，*用于寻找在一个字符串中某子字符串第一次出现的位置*。如函数名所示，这个函数返回一个整型值，表示某子字符串的第一个字符在整个字符串串的位置索引。

PATINDEX()函数和CHARINDEX()函数类似，它执行和同的操作，但方法稍许不同，该函数增加了对通配符的支持。顾名思义，*它将返回一个字符模式的索引*。这个函数也可以和ntext、nchar(max)和nvarchar(max)等大字符类型一起使用。注意，如果和这些大字符类型一起使用，PATINDEX()函数将返回bigint类型的值，而不是int类型的值。

SELECT CHARINDEX('0','ndong')SELECT PATINDEX('%s\_n%','My name is ndong')

**LEN()**

LEN()函数用于返回一个代表字符串长度的整型值。

**LEFT()和RIGHT()**

LEFT()和RIGHT()函数是相似的，它们都返回一定长度的子字符串。这两个函数的区别是它们返回的分别是字符串的不同部分。LEFT()函数返回字符串最左边的字符，顺序从左数到右。RIGHT()函数正好相反，它从最右边的字符开始，以从右到左的顺序返回特定数量的字符串。

**SUBSTRING()**

SUBSTRING()函数能够从字符串的一个位置开始，往右数若干字符，返回一个特定长度的子字符串。和LEFT()函数不同之处是，该函数可以指定从哪个位置开始计数，这样就可以在字符串的任何位置摘取子字符串了。*这个函数需要三个参数：要解析的字符串、起始位置索引、要返回的子字符串长度。*如果要返回到所输入字符串尾部的所有字符，可以使用比所需长度史大的长度值。SUBSTRING()函数将返回最大可能长度的字符数，而不会将多出的长度以空格填充。只要指定字符串最左边的字符(1)为起始索引，就可以用SUBSTRING()函数替代LEFT()函数。

**LOWER()和UPPER()**

用于将字符串中所有字符分别都转换为小写和大写，这在比较用户输入或者存储用于比较的字符串时是非常有用的。字符串比较通常是区分大小写的，这取决于SQL Server安装时的设置。

**LTRIM()和RTRIM()**  
这两个函数分别返回字符串的左边和右边的空白修剪之后的字符串。

REPLACE()

REPLACE()函数可以把字符串中的某个字符或某个子字符串替换为另一个字符或者子字符串，该函数可以用于全局查找和替换工具中。*这个函数需要三个参数：待搜索的字符串、待查找的字符串、替换用的字符串。*

**REPLICATE()和SPACE()**

在需要将一些字符重复填充进一个字符串时，这两个函数是非常有用的。例如为每个名字填满20个字符，不足20字符的用\*填满：

SELECT name + REPLICATE('\*', **20**-LEN(name))FROM #temptable

SPACE()和REPLICATE()类似，区别在于该函数使用空格进行填充。  
**REVERSE()**

这个函数用于将字符串中的字符颠倒过来。

STUFF()

*这个函数可将字符串中的一部分替换为另一个字符串*。它本质上是*将一个字符串以特定的长度插入另一个字符串中的特定位置上*。这对于源值与目的值的长度不一样的字符串替换是很有用的。

SELECT STUFF( 'Please input you name' , **14**, **3**, 'your' )

**元数据函数**

这是一些工具函数，它们返回SQL Server配置细节、服务器与数据库设置细节的信息，包括一组用于返回不同对象的属性状态的通用以及专用函数，这些函数把对Master数据库中系统表以及用户数据库的查询封装在函数中。建议使用这些函数以及其他的系统函数，而不是自己创建对系统表的查询，以防今后SQL Server版本对模式进行更改。

**排序函数**

**ROW\_NUMBER()**

*ROW\_NUMBER()函数根据作为参数传递给这个函数的ORDER BY子句（包含在OVER()中的ORDER BY子句）的值，返回一个不断递增的整数值。*ROW\_NUMBER函数生成序号的基本原理是先使用OVER子句中的ORDER BY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。如果ROW\_NUMBER的ORDER BY的值和结果集中的顺序相匹配，返回值将是递增的，以升序排列。如果ROW\_NUMBER的ORDER BY子句的值和结果集中的顺序不同，这些值将不会按顺序列出，但它们表示函数的ORDER BY子句的顺序。

ROW\_NUMBER ( ) OVER ( [ PARTITION BY value\_expression , ... [ n ] ] order\_by\_clause )

PARTITION BY value\_expression

将 [FROM](https://msdn.microsoft.com/zh-cn/library/ms177634(v=sql.120).aspx) 子句生成的结果集划分为应用 ROW\_NUMBER 函数的分区。 value\_expression 指定对结果集进行分区所依据的列。 如果未指定 PARTITION BY，则此函数将查询结果集的所有行视为单个组。

order\_by\_clause

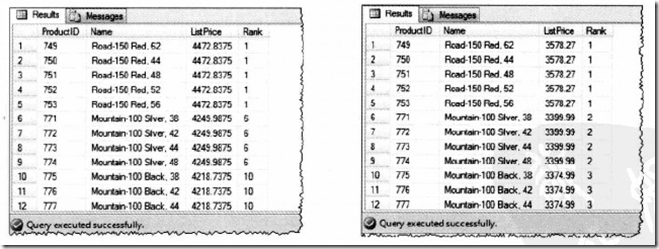
ORDER BY 子句可确定在特定分区中为行分配唯一 ROW\_NUMBER 的顺序。 它是必需的。

**RANK() / DENSE\_RANK()**

这两个函数与ROW\_NUMBER函数类似，因为它们都返回一个基于OVER( ORDER BY )子句的值。排列值对于所提供的ORDER BY了句中的重复结果而言也是重复的，而且唯一性足仅仅基于ORDER BY列表中的唯一值的。这些函数用不同的方法来处理重复的值，RANK()数保留列表中行的位置序号，对于每个重复的值，该函数会跳过下面与其相邻的值，于是就可以将下一个不重复的值保留在正确的位置上。 DENSE\_RANNK()函数的工作方式与RANK()数相同，不过它不会跳过每个连接后的值，这样就不会有值被跳过了，但是在连接处排列序号位置将会去失。

RANK ( ) OVER ( [ partition\_by\_clause ] order\_by\_clause )

DENSE\_RANK ( ) OVER ( [ <partition\_by\_clause> ] < order\_by\_clause > )

**[](http://images2015.cnblogs.com/blog/455983/201509/455983-20150912001751012-594977882.png)**

**NTILE()**

这个函数也用于对结果进行排列，并返回一个整型的排列值，但是它不会对结果以唯一的排列顺序进行枚举，而足将结果切分为有限数量的排列组。比如，一个表有10 000行，使用1000 为参数值调用ITILE()函数，会将结果分成以10为单位的1000个组，每个组赋予相同的排列值。

NTILE (integer\_expression) OVER ( [ <partition\_by\_clause> ] < order\_by\_clause > )

如果分区的行数不能被integer\_expression整除，则将导致一个成员有两种大小不同的分组。 按照 OVER 子句指定的顺序，较大的组排在较小的组前面。 例如，如果总行数是 53，组数是 5，则前三个组每组包含 11 行，其余两个组每组包含 10 行。 另一方面，如果总行数可被组数整除，则行数将在组之间平均分布。 例如，如果总行数为 50，有五个组，则每组将包含 10 行。