FUNCTIONS IN SQLSERVER 2005

By Raj Kumar Beniwal

C#Corner

Functions in SQL Server: Practical Guide

This free book is provided by courtesy of C# Corner and Mindcracker Network and its authors. Feel free to share this book with your friends and co-workers. Please do not reproduce, republish, edit or copy this book.

Raj Kumar Beniwal
Project Manager, Author C# Corner

Sam Hobbs
Editor, C# Corner

ABOUT THE AUTHOR

Raj Kumar Beniwal is two times Microsoft MVP(Client App Dev) and four times C# Corner MVP, working as asst project manager with lots of hands on experience using ASP.NET 2.0/3.5, AJAX, MVC, C#, Visual Basic .NET, SQL Server 2005/2008, Oracle, WPF, WCF, XAML, HTML5, JQuery, Windows Phone, Web Api, LightSwitch 2012 and Silverlight.

He has over 8 years of IT experience working most on Microsoft technologies. He holds Master's degree in Computer Science. When he is not writing code, he likes to write articles and play cricket.

Table of Contents

1 Introduction of Functions in SQL Server 2005

1.1 Built-in Functions

2 String Functions

2.1 LEN (string)

2.2 LTRIM (string)

2.3 RTRIM (string)

2.4 LEFT (string, length)

2.5 RIGTH (string, length)

2.6 SUBSTRING (string, start, length)

2.7 REPLACE (search, find, replace)

2.8 REVERSE (string)

2.9 CHARINDEX (find, search [, start])

2.10 PATINDEX (find, search [, start])

2.11 LOWER (string)

2.12 UPPER (string)

2.13 SPACE (integer)

3 Numeric Functions

3.1 ROUND (number, length, [function])

3.2 ISNUMERIC(expressions)

3.3 ABS (number)

3.4 CEILING (number)

3.5 FLOOR (number)

3.6 SQUARE (float_number)

3.7 SQRT (float_number)

3.8 RAND ([seed])

4 Date/Time Functions

4.1 GetDate ()

4.2 GETUTCDATE()

4.3 DAY (date)

4.4 MONTH (date)

4.5 YEAR (date)

4.6 DATENAME (datepart, date)

4.7 DATEPART(datepart, date)

4.8 DATEADD (datepart, number, date)

4.9 DATEDIFF (datepart, startdate, enddate)

4.10 ISDATE (expression)

5 More Functions

5.1 CASE

5.2 COALESCE

5.3 ISNULL

5.4 GROUPING

5.5 ROW_Number()

5.6 RANK ()

5.7 DENSE_RANK ()

5.8 NTILE (integer_expression)

5.9 ISNUMERIC

Introduction of Functions in SQL Server 2005

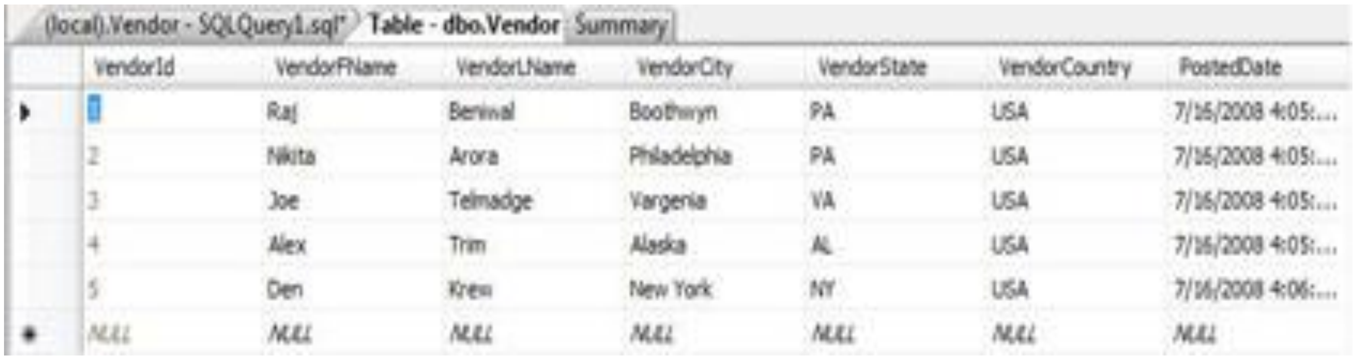
SQL Server built-in functions are either deterministic or nondeterministic. Functions are deterministic when they always return the same result any time they are called by using a specific set of input values. Functions are nondeterministic when they could return different results every time they are called, even with the same specific set of input values.

Functions that take a character string input and return a character string output use the collation of the input string for the output. Functions that take no character inputs and return a character string use the default collation of the current database for the output. Functions that take multiple character string inputs and return a character string use the rules of collation precedence to set the collation of the output string

In this Book I am going to explain about some SQL Server 2005 functions with examples. A function performs an operation and returns a value. A function consists of the function name, followed by a set of parenthesis that contains any parameter or arguments required by the function. If a function requires two or more arguments you can separate them with commas.

Here are going to discuss about some string functions, numeric functions and date/time functions.

Note - I am using my own database table for examples. See database table in figure 1.



VendorId	VendorFName	VendorLName	VendorCity	VendorState	VendorCountry	PostedDate
1	Raj	Beniwal	Boothwyn	PA	USA	7/16/2008 4:05:...
2	Nikita	Arora	Philadelphia	PA	USA	7/16/2008 4:05:...
3	Joe	Telmadge	Vargenia	VA	USA	7/16/2008 4:05:...
4	Alex	Trim	Alaska	AL	USA	7/16/2008 4:05:...
5	Den	Krew	New York	NY	USA	7/16/2008 4:06:...
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 1.

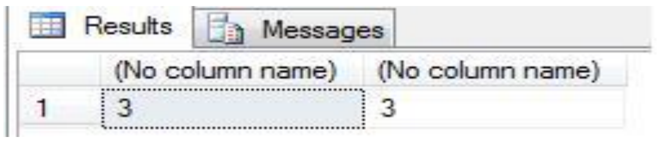
String Functions

1. **LEN (string)** - Returns the number of characters of the specified string expression, excluding trailing blanks.

Example:

```
use Vendor

GO

Select LEN('Raj'), LEN('Raj    ') FROM VENDOR WHERE VendorFName='Raj'

GO
```

LEN doesn't count length of spaces. So result looks like this.

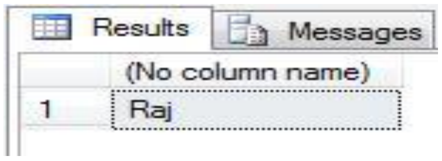


```
Results   Messages
   (No column name)   (No column name)
1  3                  3
```

2. **LTRIM (string) -** LTRIM function to return a character expression after removing leading spaces.

Example:

```
use Vendor

GO

use Vendor

SELECT LTRIM('   Raj')

FROM VENDOR WHERE VendorFName='Raj'

Go
```

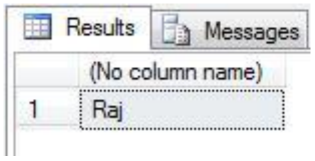


```
Results | Messages
   (No column name)
1  Raj
```

3. **RTRIM (string)** - RTRIM function to return a character expression after removing trailing spaces.

Example:

```
use Vendor

GO

use Vendor

Select RTRIM('Raj    ')

FROM VENDOR WHERE VendorFName='Raj'

GO
```

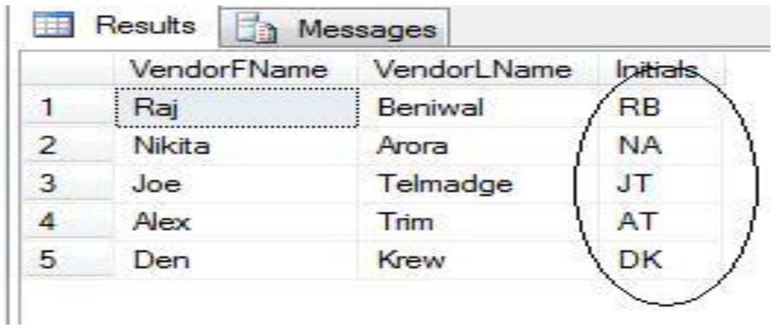


```
Results | Messages
   (No column name)
1  Raj
```

4. **LEFT (string, length) -** Returns the specified number of characters from the beginning of the string.

Example:

```
use Vendor

SELECT VendorFName, VendorLName, LEFT(VendorFName, 1) + LEFT (VendorLName, 1) AS
Initials FROM Vendor
```



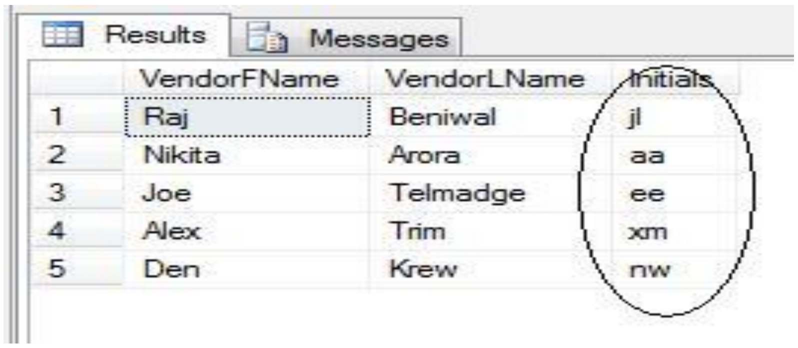
Results | Messages

	VendorFName	VendorLName	Initials
1	Raj	Beniwal	RB
2	Nikita	Arora	NA
3	Joe	Telmadge	JT
4	Alex	Trim	AT
5	Den	Krew	DK

5. **RIGTH (string, length)** - Returns the specified number of characters from the end of the string.

Example:

```
use Vendor

SELECT VendorFName, VendorLName, RIGHT(VendorFName, 1) + RIGHT (VendorLName, 1) AS
Initials FROM Vendor
```



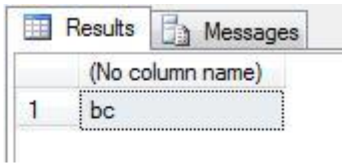
Results | Messages

	VendorFName	VendorLName	Initials
1	Raj	Beniwal	jl
2	Nikita	Arora	aa
3	Joe	Telmadge	ee
4	Alex	Trim	xm
5	Den	Krew	nw

6. **SUBSTRING (string, start, length) -** Returns the specified number of characters from the string starting at the specified position.

Example:

```
use Vendor

GO

SELECT SUBSTRING('beniwal', 2, 2) FROM VENDOR WHERE VendorFName='Raj'

GO
```

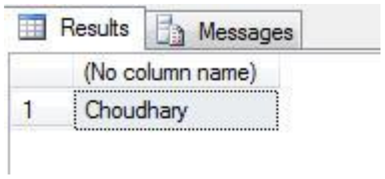


```
(No column name)
1  bc
```

7. **REPLACE (search, find, replace)** - Returns the search string with all occurrences of the find string replaced with the replace string.

Example:

```
use Vendor

GO

use Vendor

SELECT REPLACE('Beniwal', 'Beniwal', 'Choudhary')

FROM VENDOR WHERE VendorFName='Raj'

GO
```



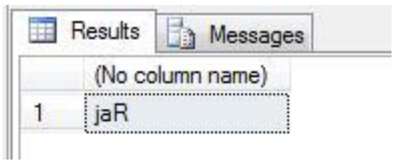
```
(No column name)
1  Choudhary
```

8. **REVERSE (string) -** Returns the string with the character in reverse order.

Example:

```
use Vendor

GO
```

```
use Vendor

SELECT REVERSE('Raj')

FROM VENDOR WHERE VendorFName='Raj'

GO
```

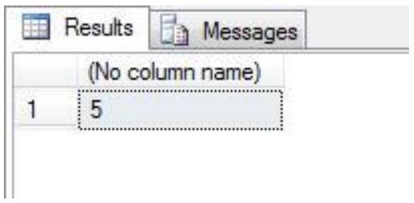


```
Results  Messages
   (No column name)
1  jaR
```

9. **CHARINDEX (find, search [, start]) -** Returns an integer that represents the position of the first occurrence of the find string in the search string starting at the specified position. If the starting position isn't specified, the search starts at the beginning of the string. If the staring isn't found, the functions returns zero.

Example:

```
use Vendor

GO

use Vendor

SELECT CHARINDEX('w', 'Beniwal')

FROM VENDOR WHERE VendorFName='Raj'

GO
```

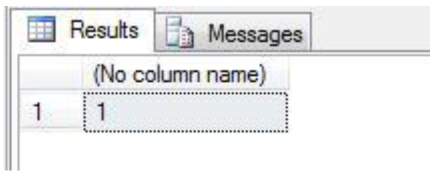


```
Results  Messages
   (No column name)
1  5
```

10. **PATINDEX (find, search [, start]) -** PATINDEX is useful with text data types; it can be used in a WHERE clause in addition to IS NULL, IS NOT NULL, and LIKE (the only other comparisons that are valid on text in a WHERE clause). If either pattern or expression is NULL, PATINDEX returns NULL when the database compatibility level is 70. If the database compatibility level is 65 or earlier, PATINDEX returns NULL only when both pattern and expression are NULL.

Example:

```
use Vendor

GO

use Vendor

SELECT PATINDEX('%Krew%', VendorLName)

FROM VENDOR WHERE VendorId=5

GO
```



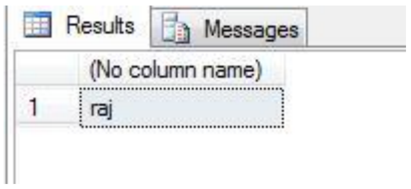
```
   (No column name)
1  1
```

11. **LOWER (string) -** Returns the string converted to lowercase letters.

Example:

```
use Vendor

GO

use Vendor

SELECT LOWER('Raj')

FROM VENDOR WHERE VendorFName='Raj'
```

```
GO
```

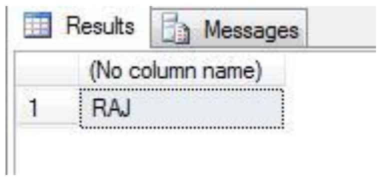


```
(No column name)
1   raj
```

12. **UPPER (string) -** Returns the string converted to uppercase letters.

Example:

```
use Vendor

GO

use Vendor

SELECT UPPER('Raj')

FROM VENDOR WHERE VendorFName='Raj'

GO
```



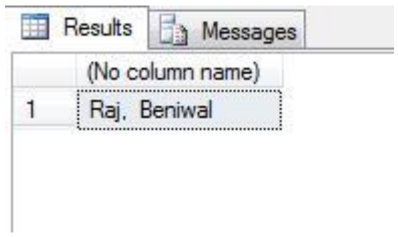
```
(No column name)
1   RAJ
```

13. **SPACE (integer) -** Returns the string with the specified number of space characters (blanks).

Example:

```
use Vendor

GO

use Vendor
```

```
SELECT VendorFName + ',' + SPACE(2) + VendorLName

FROM VENDOR WHERE VendorFName='Raj'

GO
```



```
	(No column name)
1	Raj,  Beniwal
```

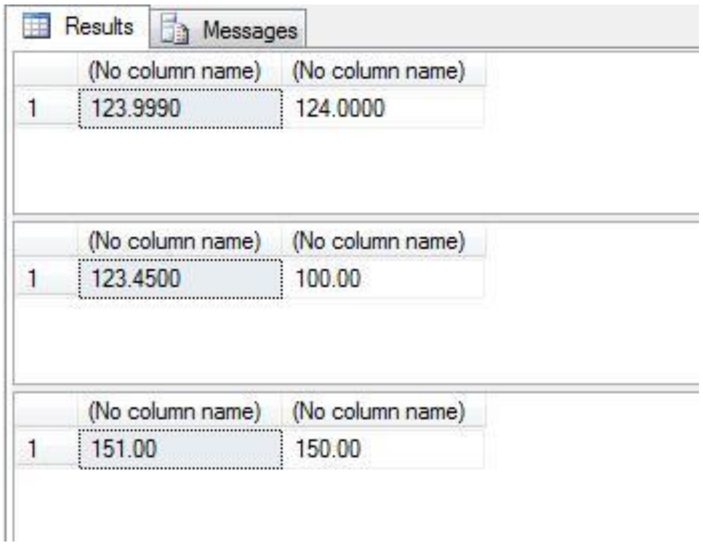
Numeric Functions

1 **ROUND (number, length, [function]) -** Returns the number rounded to the precision specified by length. If length is positive, the digits to the right of the decimal point are rounded. If it's negative the digits to the left of the decimal point are rounded. To truncate the number rather than round it code a non zero value for function.

Example:

```
USE Vendor

GO

--Used Round the estimates

SELECT ROUND(123.9994, 3), ROUND(123.9995, 3)



--Use ROUND and rounding approximations

SELECT ROUND(123.4545, 2), ROUND(123.45, -2)



--Use ROUND to truncate

SELECT ROUND(150.75, 0), ROUND(150.75, 0, 1)
```

```
GO
```

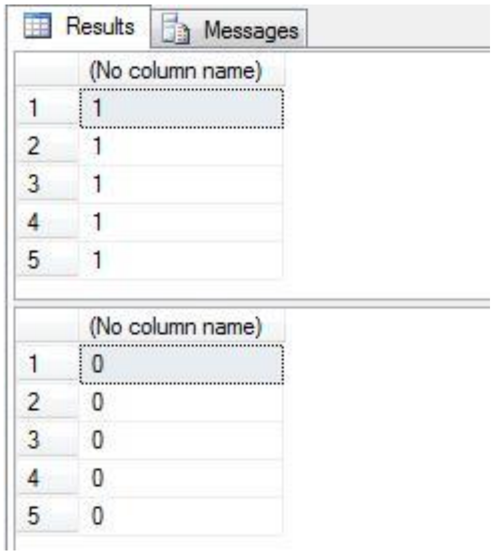


```
Results | Messages

   (No column name)  (No column name)
1  123.9990          124.0000

   (No column name)  (No column name)
1  123.4500          100.00

   (No column name)  (No column name)
1  151.00            150.00
```

2 **ISNUMERIC(expressions) -** Returns a value of 1 (true) if the expression is a numeric value; returns a values of 0 (false) otherwise.

Example:

```
USE Vendor

GO

SELECT IsNumeric(VendorId) FROM Vendor

SELECT ISNumeric(VendorFName) FROM Vendor

GO
```

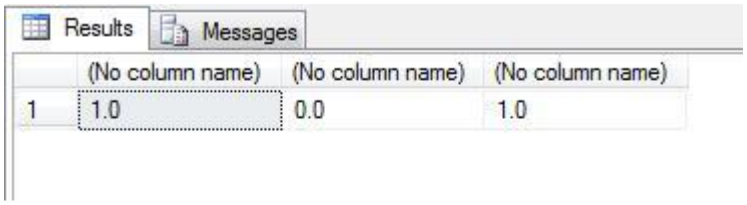


```
   (No column name)
1  1
2  1
3  1
4  1
5  1

   (No column name)
1  0
2  0
3  0
4  0
5  0
```

3 **ABS (number) -** Returns the absolute value of number.

Example:

```
USE Vendor

GO

SELECT ABS(-1.0), ABS(0.0), ABS(1.0)

GO
```



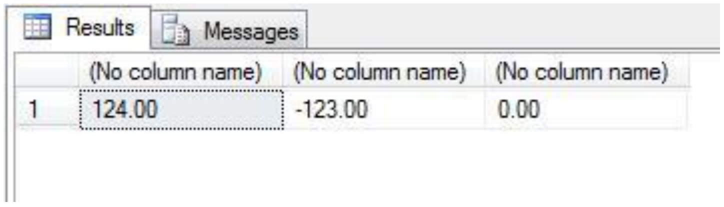
```
   (No column name)  (No column name)  (No column name)
1  1.0               0.0               1.0
```

4 **CEILING (number) -** Returns the smallest integer that is greater than or equal to the number.

Example:

```
USE Vendor

GO
```

```
SELECT CEILING($123.45),CEILING($-123.45), CEILING($0.0)

GO
```



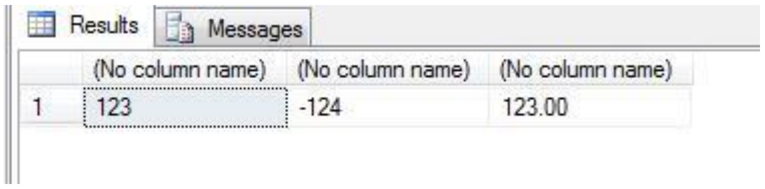
Results | Messages

	(No column name)	(No column name)	(No column name)
1	124.00	-123.00	0.00

5 **FLOOR (number)** - Is an expression of the exact numeric or approximate numeric data type category, except for the bit data type.

Example:

```
USE Vendor

GO

SELECT FLOOR(123.45), FLOOR(-123.45), FLOOR($123.45)

GO
```



Results | Messages

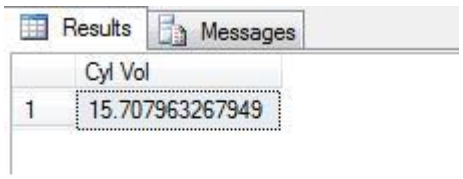
	(No column name)	(No column name)	(No column name)
1	123	-124	123.00

6 **SQUARE (float_number)** - Returns the square of the given expression.

Example:

```
USE Vendor

GO

DECLARE @h float, @r float
```

```
SET @h = 5

SET @r = 1

SELECT PI()* SQUARE(@r)* @h AS 'Cyl Vol'

GO
```

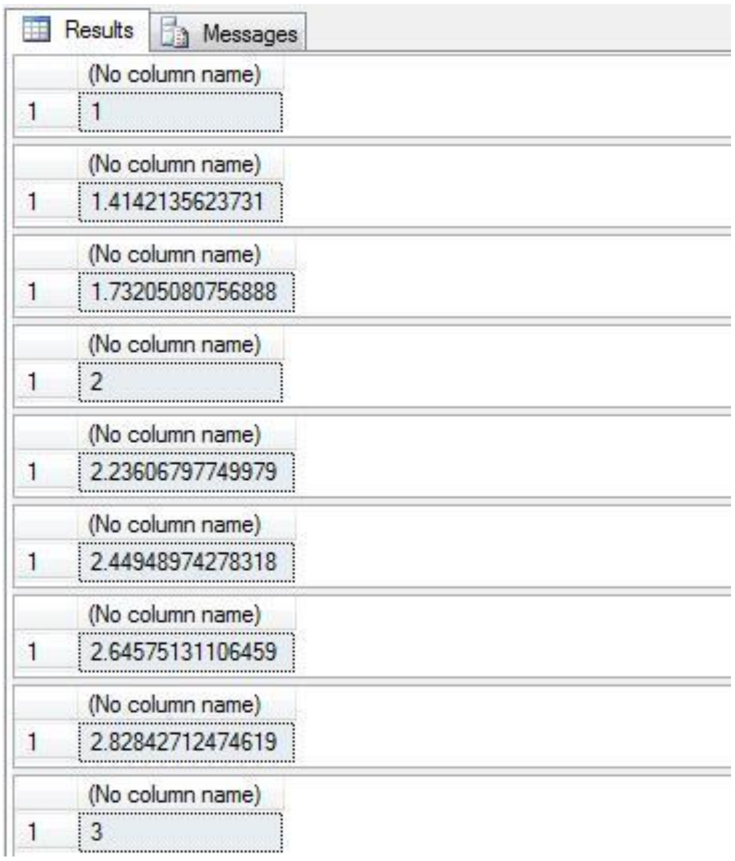


```
   Cyl Vol
1  15.707963267949
```

7 **SQRT (float_number) -** Returns a square root of a floating-point number.

Example:

```
USE Vendor

GO

DECLARE @myvalue float

SET @myvalue = 1.00

WHILE @myvalue < 10.00

  BEGIN

    SELECT SQRT(@myvalue)

    SELECT @myvalue = @myvalue + 1

  END

GO
```



```
Results   Messages
   (No column name)
1  1
   (No column name)
1  1.4142135623731
   (No column name)
1  1.73205080756888
   (No column name)
1  2
   (No column name)
1  2.23606797749979
   (No column name)
1  2.44948974278318
   (No column name)
1  2.64575131106459
   (No column name)
1  2.82842712474619
   (No column name)
1  3
```

8 **RAND ([seed]) -** Returns a random float value from 0 through 1.

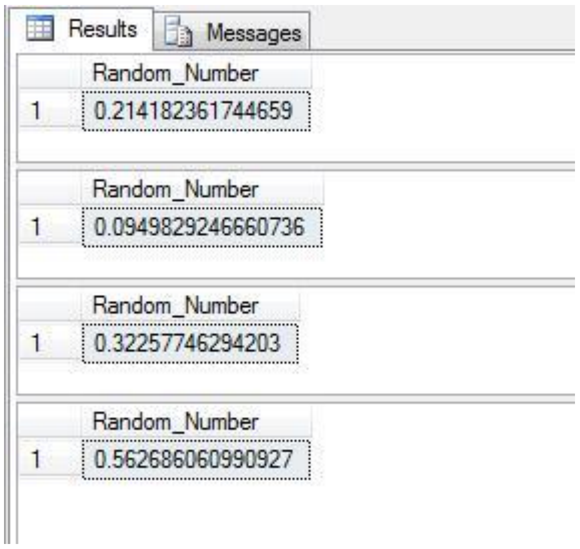
Seed

Is an integer expression (tinyint, smallint, or int) that specifies the seed value. If seed is not specified, Microsoft SQL Server 2000 assigns a seed value at random. For a given seed value, the result returned is always the same.

Example:

```
USE Vendor

GO

DECLARE @counter smallint

SET @counter = 1

WHILE @counter < 5
```

```
 BEGIN

   SELECT RAND() Random_Number

   SET NOCOUNT ON

   SET @counter = @counter + 1

   SET NOCOUNT OFF

 END

GO
```



```
   Random_Number
1  0.214182361744659

   Random_Number
1  0.0949829246660736

   Random_Number
1  0.32257746294203

   Random_Number
1  0.562686060990927
```

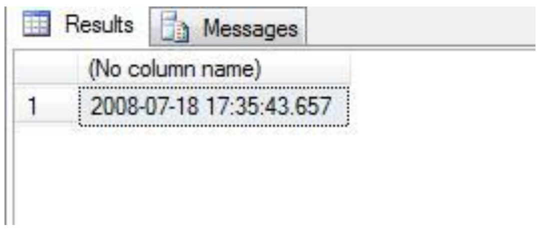
Date/Time Functions

1 GetDate () - Returns the current system date and time in the Microsoft SQL Server standard internal format for date time values.

Example:

```
USE Vendor

GO

SELECT GetDate()
```

```
GO
```

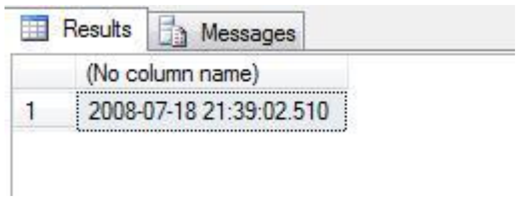


```
Results | Messages
   (No column name)
1  2008-07-18 17:35:43.657
```

2 **GETUTCDATE() -** Returns the current UTC date and time based on the system's clock and time zone setting. UTC (Universal Time Coordination) is the same as Greenwich Mean Time.

Example:

```
USE Vendor

GO

SELECT GETUTCDATE()

GO
```

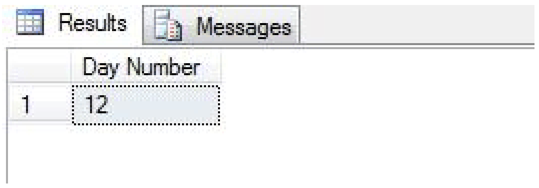


```
Results | Messages
   (No column name)
1  2008-07-18 21:39:02.510
```

3 **DAY (date) -** Returns the day of the month as an integer.

Example:

```
USE Vendor

GO

SELECT DAY('03/12/1998') AS 'Day Number'

GO
```

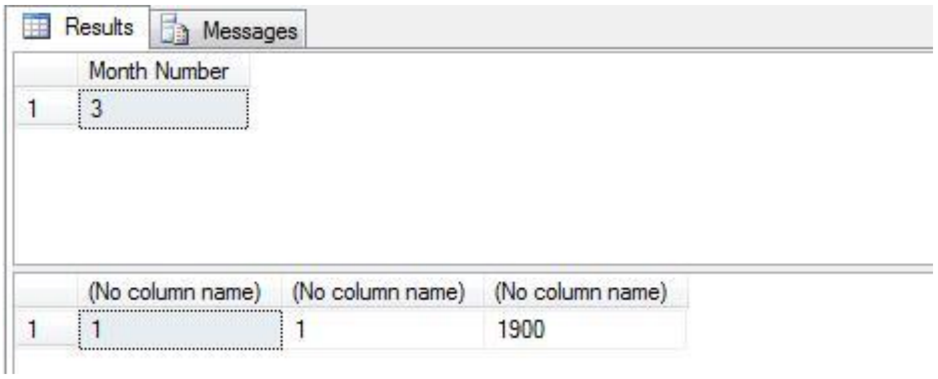


```
Results | Messages
   Day Number
1  12
```

4 **MONTH (date) -** Returns the month as an integer.

Example:

```
USE Vendor

GO

SELECT "Month Number" = MONTH('03/12/1998')

SELECT MONTH(0), DAY(0), YEAR(0)

GO
```

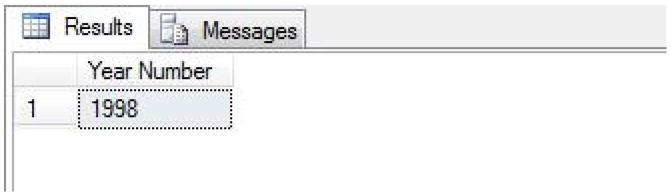


```
Results | Messages
   Month Number
1  3

   (No column name)  (No column name)  (No column name)
1  1                 1                 1900
```

5 **YEAR (date) -** Returns the 4-digit year as an integer.

Example:

```
USE Vendor

GO

SELECT "Year Number" = YEAR('03/12/1998')

GO
```

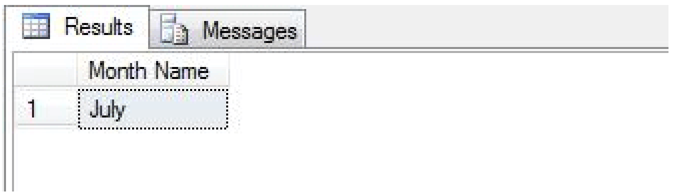

Results | Messages

```
  Year Number
1 1998
```

6 **DATENAME (datepart, date) -** Returns an integer representing the specified date part of the specified date.

Example:

```
USE Vendor

GO

SELECT DATENAME(month, getdate()) AS 'Month Name'

GO
```


Results | Messages

```
  Month Name
1 July
```

7 **DATEPART(datepart, date)**

Is the parameter that specifies the part of the date to return? The table lists date parts and abbreviations recognized by Microsoft SQL Server.

Datepart	Abbreviations
year	**yy, yyyy**
quarter	**qq, q**
month	**mm, m**

dayofyear	**dy, y**
day	**dd, d**
week	**wk, ww**
weekday	**dw**
hour	**hh**
minute	**mi, n**
second	**ss, s**
millisecond	**ms**

The week (wk, ww) datepart reflects changes made to SET DATEFIRST. January 1 of any year defines the starting number for the week datepart, for example: DATEPART(wk, 'Jan 1, xxxx') = 1, where xxxx is any year.

The weekday (dw) datepart returns a number that corresponds to the day of the week, for example: Sunday = 1, Saturday = 7. The number produced by the weekday datepart depends on the value set by SET DATEFIRST, which sets the first day of the week.

Date

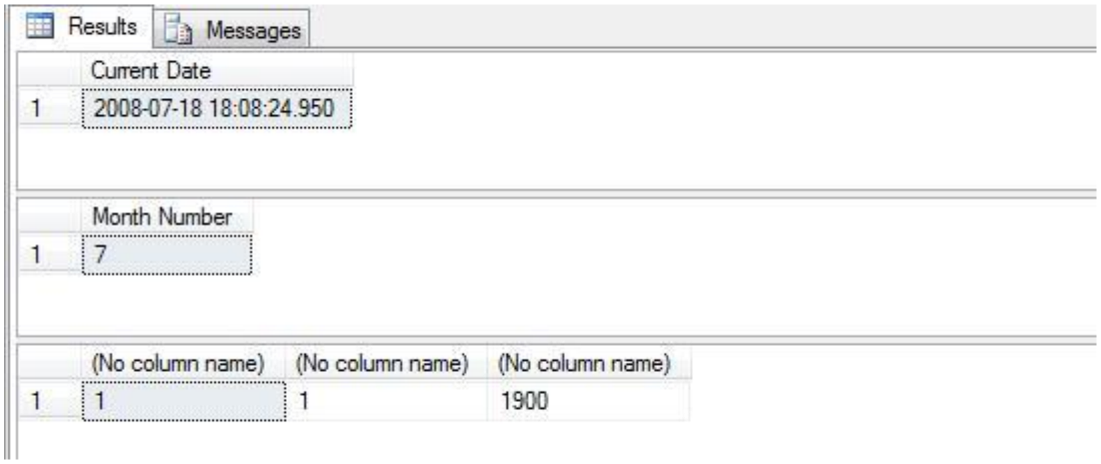
Is an expression that returns a datetime or smalldatetime value, or a character string in a date format. Use the datetime data type only for dates after January 1, 1753. Store dates as character data for earlier dates. When entering datetime values, always enclose them in quotation marks. Because smalldatetime is accurate only to the minute, when a smalldatetime value is used, seconds and milliseconds are always 0.

If you specify only the last two digits of the year, values less than or equal to the last two digits of the value of the two digit year cutoff configuration option are in the same century as the cutoff year. Values greater than the last two digits of the value of this option are in the century that precedes the cutoff year. For example, if two digit year cutoff is 2049 (default), 49 is interpreted as 2049 and 2050 is interpreted as 1950. To avoid ambiguity, use four-digit years.

For more information about specifying time values, see Time Formats. For more information about specifying dates, see datetime and smalldatetime.

Example :

```
USE Vendor

GO

SELECT GETDATE() AS 'Current Date'

SELECT DATEPART(month, GETDATE()) AS 'Month Number'

SELECT DATEPART(m, 0), DATEPART(d, 0), DATEPART(yy, 0)

GO
```

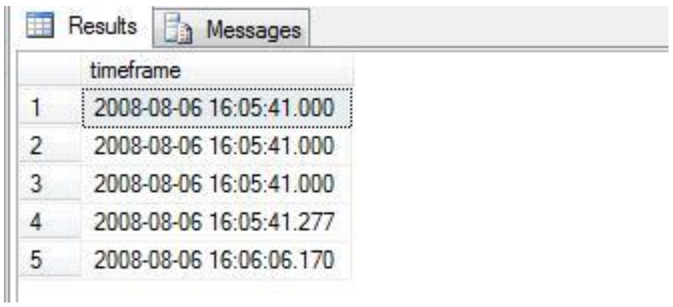


```
   Current Date
1  2008-07-18 18:08:24.950

   Month Number
1  7

   (No column name)  (No column name)  (No column name)
1  1                 1                 1900
```

8 **DATEADD (datepart, number, date) -** Returns the date that results from adding the specified number of datepart units to the date.

Example -

```
USE Vendor

GO

SELECT DATEADD(day, 21, PostedDate) AS timeframe FROM Vendor

GO
```

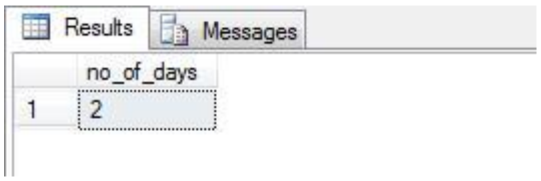


```
Results | Messages

   timeframe
1  2008-08-06 16:05:41.000
2  2008-08-06 16:05:41.000
3  2008-08-06 16:05:41.000
4  2008-08-06 16:05:41.277
5  2008-08-06 16:06:06.170
```

9 **DATEDIFF (datepart, startdate, enddate) -** Returns the number of datepart units between the specified start date and end date.

Example:

```
USE Vendor

GO

SELECT DATEDIFF(day, posteddate, getdate()) AS no_of_days

FROM Vendor WHERE VendorFName='Raj'

GO
```

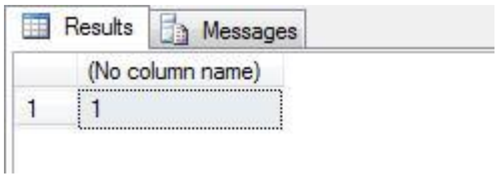


```
Results | Messages

   no_of_days
1  2
```

10 **ISDATE (expression) -** Returns a value of 1(true) if the expression is a valid date/time value; returns a value of 0(false) otherwise.

Example:

```
USE Vendor

GO

DECLARE @datestring varchar(8)

SET @datestring = '12/21/98'

SELECT ISDATE(@datestring)

GO
```



```
   (No column name)
1  1
```

More Functions

1 **CASE -** Evaluate a list of conditions and returns one of multiple possible return expressions.

CASE has two formats:

The simple CASE function compares an expression to a set of simple expressions to determine the result.

The searched case function evaluates a set of boolean expressions to determine the result.

Syntax:

Simple CASE function:

```
CASE input_expression

     WHEN when_expression THEN result_expression

          [...n]

     [
                ELSE else_result_expression

        ]

END
```

Searched CASE function:

```
CASE

        WHEN Boolean_expression THEN result_expression

                [...n]

        [

        ELSE else_result_expression

        ]

END
```

Example:

```
use Vendor

GO

SELECT VendorId, VendorFName, VendorLName,

CASE VendorId

        WHEN 1 THEN 'This is vendor id one'

        WHEN 2 THEN 'This is vendor id two'

        WHEN 3 THEN 'This is vendor id three'

        WHEN 4 THEN 'This is vendor id four'

        WHEN 5 THEN 'this is vendor id five'

END AS PrintMessage
```

```
FROM Vendor
```

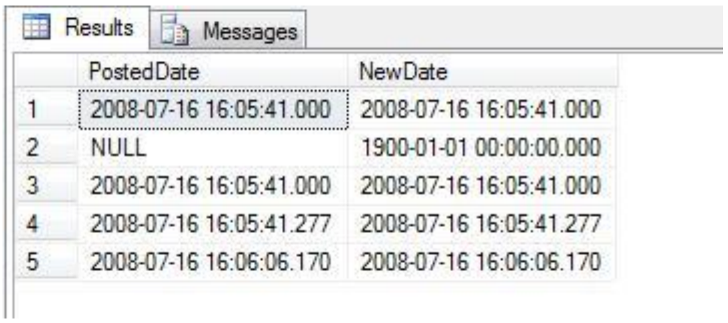
2 **COALESCE** - Returns the first nonnull expression among its arguments.

Syntax:

COALESCE (expression [...n])

Example:

```
use Vendor

GO

SELECT PostedDate, COALESCE(PostedDate, '1900-01-01') AS NewDate

FROM Vendor
```



	PostedDate	NewDate
1	2008-07-16 16:05:41.000	2008-07-16 16:05:41.000
2	NULL	1900-01-01 00:00:00.000
3	2008-07-16 16:05:41.000	2008-07-16 16:05:41.000
4	2008-07-16 16:05:41.277	2008-07-16 16:05:41.277
5	2008-07-16 16:06:06.170	2008-07-16 16:06:06.170

3 **ISNULL** - Replaces NULL with the specified replacement value.

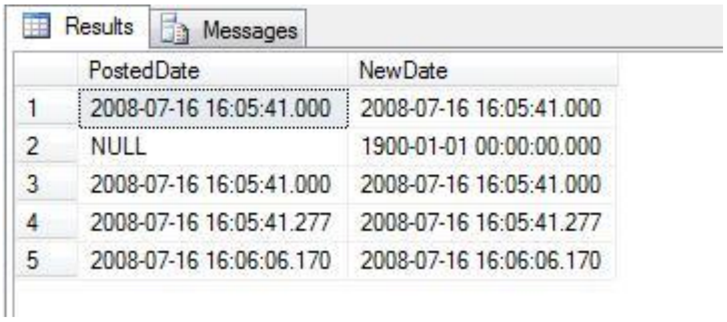
Syntax:

ISNULL (check_expression, replacement_value)

Example:

```
use Vendor

GO

SELECT PostedDate, ISNULL(PostedDate, '1900-01-01') AS NewDate
```

```
FROM Vendor

GO
```



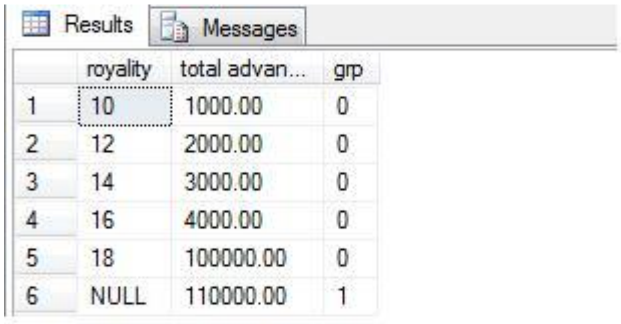
	PostedDate	NewDate
1	2008-07-16 16:05:41.000	2008-07-16 16:05:41.000
2	NULL	1900-01-01 00:00:00.000
3	2008-07-16 16:05:41.000	2008-07-16 16:05:41.000
4	2008-07-16 16:05:41.277	2008-07-16 16:05:41.277
5	2008-07-16 16:06:06.170	2008-07-16 16:06:06.170

4 **GROUPING -** Is an aggregate function that causes an additional column to be output with a value of 1 when the row is added by either the CUBE or ROLLUP operator, or 0 when the row is not the result of CUBE or ROLLUP.Grouping is allowed only in the select list associated with a GROUP BY clause that contains either the CUBE or ROLLUP operator.

Syntax: GROUPING (column_name)

Example -

```
Use Vendor

GO

SELECT royality, SUM(advance) 'total advance', GROUPING(royality) 'grp'

FROM advance

GROUP BY royality

WITH ROLLUP
```



	royality	total advan...	grp
1	10	1000.00	0
2	12	2000.00	0
3	14	3000.00	0
4	16	4000.00	0
5	18	100000.00	0
6	NULL	110000.00	1

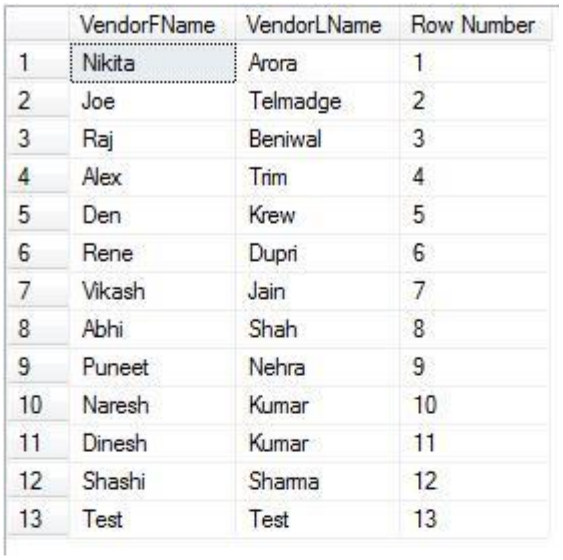
5 **ROW_Number() -** Returns the sequential number of a row within a partition of a result set, starting at 1 for the first row in each partition

Syntax: ROW_NUMBER ( ) OVER ([<partition_by_clause>] <order_by_clause>)

Note: The ORDER BY in the OVER clause orders ROW_NUMBER. If you add an ORDER BY clause to the SELECT statement that orders by a column(s) other than 'Row Number' the result set will be ordered by the outer ORDER BY.

Example:

```
Use Vendor

GO

SELECT VendorFName, VendorLName,

ROW_Number() Over(ORDER BY PostedDate) AS 'Row Number'

FROM Vendor

GO
```



	VendorFName	VendorLName	Row Number
1	Nikita	Arora	1
2	Joe	Telmadge	2
3	Raj	Beniwal	3
4	Alex	Trim	4
5	Den	Krew	5
6	Rene	Dupri	6
7	Vikash	Jain	7
8	Abhi	Shah	8
9	Puneet	Nehra	9
10	Naresh	Kumar	10
11	Dinesh	Kumar	11
12	Shashi	Sharma	12
13	Test	Test	13

6 **RANK () -** Returns the rank of each row within the partition of a result set. The rank of a row is one plus the number of ranks that come before the row in question.

Syntax: RANK ( ) OVER ([< partition_by_clause >] < order_by_clause >)

Arguments: < partition_by_clause >

Divides the result set produced by the FROM clause into partitions to which the RANK function is applied. For the syntax of PARTITION BY, see OVER Clause (Transact-SQL).

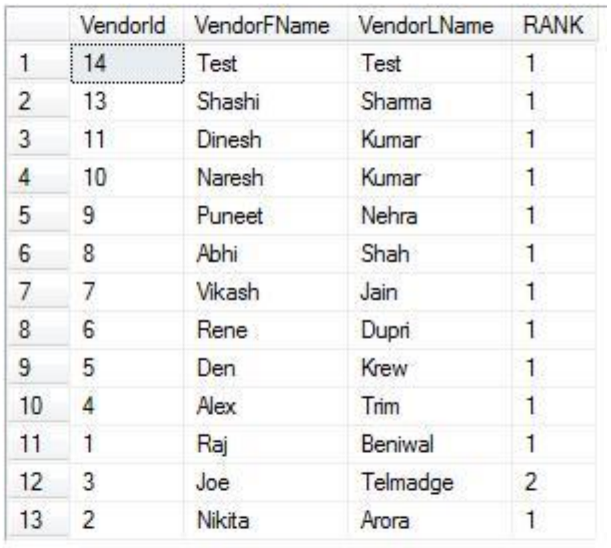
< order_by_clause >

Determines the order in which the RANK values are applied to the rows in a partition. For more information, see ORDER BY Clause (Transact-SQL). An integer cannot represent a column when the < order_by_clause > is used in a ranking function.

Example:

```
Use Vendor

GO

SELECT VendorId, VendorFName, VendorLName,

RANK() Over(PARTITION BY PostedDate ORDER BY VendorId) AS 'RANK'
```

```
FROM Vendor ORDER BY PostedDate DESC

GO
```



	VendorId	VendorFName	VendorLName	RANK
1	14	Test	Test	1
2	13	Shashi	Sharma	1
3	11	Dinesh	Kumar	1
4	10	Naresh	Kumar	1
5	9	Puneet	Nehra	1
6	8	Abhi	Shah	1
7	7	Vikash	Jain	1
8	6	Rene	Dupri	1
9	5	Den	Krew	1
10	4	Alex	Trim	1
11	1	Raj	Beniwal	1
12	3	Joe	Telmadge	2
13	2	Nikita	Arora	1

7 **DENSE_RANK ()** - Returns the rank of rows within the partition of a result set, without any gaps in the ranking. The rank of a row is one plus the number of distinct ranks that come before the row in question.

Syntax: DENSE_RANK ( ) OVER ([< partition_by_clause >] < order_by_clause >)

Arguments: < partition_by_clause >

Divides the result set produced by the FROM clause into partitions to which the DENSE_RANK function is applied. For the syntax of PARTITION BY, see OVER Clause (Transact-SQL).

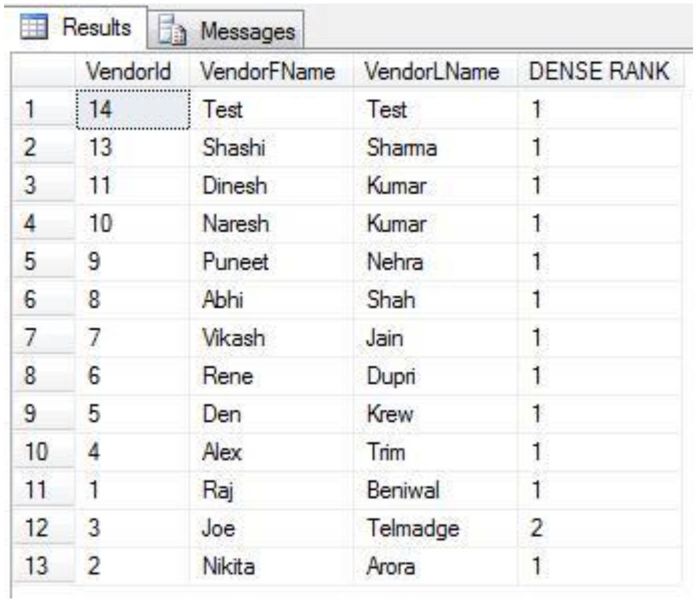
< order_by_clause >

Determines the order in which the DENSE_RANK values are applied to the rows in a partition. An integer cannot represent a column in the <order_by_clause> that is used in a ranking function.

Example :

```
Use Vendor
```

```
GO

SELECT VendorId, VendorFName, VendorLName,

DENSE_RANK() OVER(PARTITION BY PostedDate ORDER BY VendorId) AS 'DENSE RANK'

FROM Vendor ORDER BY PostedDate DESC

GO
```


Results | Messages

	VendorId	VendorFName	VendorLName	DENSE RANK
1	14	Test	Test	1
2	13	Shashi	Sharma	1
3	11	Dinesh	Kumar	1
4	10	Naresh	Kumar	1
5	9	Puneet	Nehra	1
6	8	Abhi	Shah	1
7	7	Vikash	Jain	1
8	6	Rene	Dupri	1
9	5	Den	Krew	1
10	4	Alex	Trim	1
11	1	Raj	Beniwal	1
12	3	Joe	Telmadge	2
13	2	Nikita	Arora	1

NTILE (integer_expression) - Distributes the rows in an ordered partition into a specified number of groups. The groups are numbered, starting at one. For each row, NTILE returns the number of the group to which the row belongs.

Syntax: NTILE (integer_expression) OVER ([<partition_by_clause>] < order_by_clause >)

Arguments: integer_expression

Is a positive integer constant expression that specifies the number of groups into which each partition must be divided? integer_expression can be of type int, or bigint.

Note: integer_expression can only reference columns in the PARTITION BY clause. integer_expression cannot reference columns listed in the current FROM clause.

<partition_by_clause>

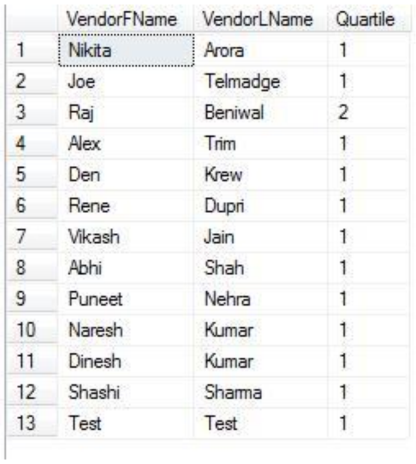
Divides the result set produced by the FROM clause into partitions to which the RANK function is applied. For the syntax of PARTITION BY, see OVER Clause (Transact-SQL).

< order_by_clause >

Determines the order in which the NTILE values are assigned to the rows in a partition. For more information, see ORDER BY Clause (Transact-SQL). An integer cannot represent a column when the <order_by_clause> is used in a ranking function.

Example :

```
Use Vendor

GO

SELECT VendorFName, VendorLName,

NTILE(4) OVER(PARTITION BY PostedDate ORDER BY VendorId DESC) AS 'Quartile'

FROM Vendor

GO
```



	VendorFName	VendorLName	Quartile
1	Nikita	Arora	1
2	Joe	Telmadge	1
3	Raj	Beniwal	2
4	Alex	Trim	1
5	Den	Krew	1
6	Rene	Dupri	1
7	Vikash	Jain	1
8	Abhi	Shah	1
9	Puneet	Nehra	1
10	Naresh	Kumar	1
11	Dinesh	Kumar	1
12	Shashi	Sharma	1
13	Test	Test	1

STUFF : It works like replace

```
select empname,STUFF(empname,4,3,'f') from EmpDetails
```

Result:

empname   (New)

jagatheesan jagfeesan

prabhakar prafkar

sibi   sibf

Rajesh   Rajf

siva   sivf

null   nulf

ISNUMERIC: This function Returns 1 if the value are numeric & zero when values are not numeric.

EXAMPLE :

```
select ISNUMERIC(name) from Employee
```

Result

new
0
0
0

Summary

In this book we have learned the some important functions like Built-in functions in Sql Server of Miscellaneous functions. Where we have seen the Rank, Dense rank methods.