

# Table of Contents

[Overview](#)

[Aggregate](#)

[AVG](#)

[CHECKSUM\\_AGG](#)

[COUNT](#)

[COUNT\\_BIG](#)

[GROUPING](#)

[GROUPING\\_ID](#)

[MAX](#)

[MIN](#)

[STDEV](#)

[STDEVP](#)

[SUM](#)

[VAR](#)

[VARP](#)

[Analytic](#)

[CUME\\_DIST](#)

[FIRST\\_VALUE](#)

[LAG](#)

[LAST\\_VALUE](#)

[LEAD](#)

[PERCENTILE\\_CONT](#)

[PERCENTILE\\_DISC](#)

[PERCENT\\_RANK](#)

[Collation](#)

[Collation - COLLATIONPROPERTY](#)

[Collation - TERTIARY\\_WEIGHTS](#)

[Configuration](#)

[@@DBTS](#)

@@LANGID  
@@LANGUAGE  
@@LOCK\_TIMEOUT  
@@MAX\_CONNECTIONS  
@@MAX\_PRECISION  
@@NESTLEVEL  
@@OPTIONS  
@@REMSERVER  
@@SERVERNAME  
@@SERVICENAME  
@@SPID  
@@TEXTSIZE  
@@VERSION

## Conversion

CAST and CONVERT  
PARSE  
TRY\_CAST  
TRY\_CONVERT  
TRY\_PARSE

## Cryptographic

ASYMKEY\_ID  
ASYMKEYPROPERTY  
CERTPROPERTY  
CERT\_ID  
CRYPT\_GEN\_RANDOM  
DECRYPTBYASYMKEY  
DECRYPTBYCERT  
DECRYPTBYKEY  
DECRYPTBYKEYAUTOASYMKEY  
DECRYPTBYKEYAUTOCERT  
DECRYPTBYPASSPHRASE  
ENCRYPTBYASYMKEY

ENCRYPTBYCERT  
ENCRYPTBYKEY  
ENCRYPTBYPASSPHRASE  
HASHBYTES  
IS\_OBJECTSIGNED  
KEY\_GUID  
KEY\_ID  
KEY\_NAME  
SIGNBYASYMKEY  
SIGNBYCERT  
SYMKEYPROPERTY  
VERIFYSIGNEDBYCERT  
VERIFYSIGNEDBYASYMKEY

#### Cursor

@@CURSOR\_ROWS  
@@FETCH\_STATUS  
CURSOR\_STATUS

#### Data type

DATALENGTH  
IDENT\_CURRENT  
IDENT\_INCR  
IDENT\_SEED  
IDENTITY (Function)  
SQL\_VARIANT\_PROPERTY

#### Date and time

@@DATEFIRST  
CURRENT\_TIMESTAMP  
DATEADD  
DATEDIFF  
DATEDIFF\_BIG  
DATEFROMPARTS  
DATENAME

DATEPART  
DATETIME2FROMPARTS  
DATETIMEFROMPARTS  
DATETIMEOFFSETFROMPARTS  
DAY  
EOMONTH  
GETDATE  
GETUTCDATE  
ISDATE  
MONTH  
SMALLDATETIMEFROMPARTS  
SWITCHOFFSET  
SYSDATETIME  
SYSDATETIMEOFFSET  
SYSUTCDATETIME  
TIMEFROMPARTS  
TODATETIMEOFFSET  
YEAR  
JSON  
ISJSON  
JSON\_VALUE  
JSON\_QUERY  
JSON\_MODIFY

Mathematical

ABS  
ACOS  
ASIN  
ATAN  
ATN2  
CEILING  
COS  
COT

DEGREES  
EXP  
FLOOR  
LOG  
LOG10  
PI  
POWER  
RADIANS  
RAND  
ROUND  
SIGN  
SIN  
SQRT  
SQUARE  
TAN  
Logical  
CHOOSE  
IIF  
Metadata  
@@PROCID  
APP\_NAME  
APPLOCK\_MODE  
APPLOCK\_TEST  
ASSEMBLYPROPERTY  
COL\_LENGTH  
COL\_NAME  
COLUMNPROPERTY  
DATABASE\_PRINCIPAL\_ID  
DATABASEPROPERTYEX  
DB\_ID  
DB\_NAME  
FILE\_ID

FILE\_INDEX  
FILE\_NAME  
FILEGROUP\_ID  
FILEGROUP\_NAME  
FILEGROUOPROPERTY  
FILEPROPERTY  
FULLTEXTCATALOGPROPERTY  
FULLTEXTSERVICEPROPERTY  
INDEX\_COL  
INDEXKEY\_PROPERTY  
INDEXPROPERTY  
NEXT VALUE FOR  
OBJECT\_DEFINITION  
OBJECT\_ID  
OBJECT\_NAME  
OBJECT\_SCHEMA\_NAME  
OBJECTPROPERTY  
OBJECTPROPERTYEX  
ORIGINAL\_DB\_NAME  
PARSENAME  
SCHEMA\_ID  
SCHEMA\_NAME  
SCOPE\_IDENTITY  
SERVERPROPERTY  
STATS\_DATE  
TYPE\_ID  
TYPE\_NAME  
TYPEPROPERTY  
VERSION  
ODBC Scalar  
Ranking  
DENSE\_RANK

NTILE  
RANK  
ROW\_NUMBER

Replication

PUBLISHINGSERVERNAME

Rowset

OPENDATASOURCE  
OPENJSON  
OPENQUERY  
OPENROWSET  
OPENXML

Security

CERTENCODED  
CERTPRIVATEKEY  
CURRENT\_USER  
HAS\_DBACCESS  
HAS\_PERMS\_BY\_NAME  
IS\_MEMBER  
IS\_ROLEMEMBER  
IS\_SRVROLEMEMBER  
LOGINPROPERTY  
ORIGINAL\_LOGIN  
PERMISSIONS  
PWDENCRYPT  
PWDCOMPARE  
SESSION\_USER  
SESSIONPROPERTY  
SUSER\_ID  
SUSER\_NAME  
SUSER\_SID  
SUSER\_SNAME  
SYSTEM\_USER

USER  
USER\_ID  
USER\_NAME  
String  
ASCII  
CHAR  
CHARINDEX  
CONCAT  
CONCAT\_WS  
DIFFERENCE  
FORMAT  
LEFT  
LEN  
LOWER  
LTRIM  
NCHAR  
PATINDEX  
QUOTENAME  
REPLACE  
REPLICATE  
REVERSE  
RIGHT  
RTRIM  
SOUNDEX  
SPACE  
STR  
STRING\_AGG  
STRING\_ESCAPE  
STRING\_SPLIT  
STUFF  
SUBSTRING  
TRANSLATE

TRIM  
UNICODE  
UPPER  
System  
\$PARTITION  
@@ERROR  
@@IDENTITY  
@@PACK\_RECEIVED  
@@ROWCOUNT  
@@TRANCOUNT  
BINARY\_CHECKSUM  
CHECKSUM  
COMPRESS  
CONNECTIONPROPERTY  
CONTEXT\_INFO  
CURRENT\_REQUEST\_ID  
CURRENT\_TRANSACTION\_ID  
DECOMPRESS  
ERROR\_LINE  
ERROR\_MESSAGE  
ERROR\_NUMBER  
ERROR\_PROCEDURE  
ERROR\_SEVERITY  
ERROR\_STATE  
FORMATMESSAGE  
GET\_FILESTREAM\_TRANSACTION\_CONTEXT  
GETANSINULL  
HOST\_ID  
HOST\_NAME  
ISNULL  
ISNUMERIC  
MIN\_ACTIVE\_ROWVERSION

NEWID  
NEWSEQUENTIALID  
ROWCOUNT\_BIG  
SESSION\_CONTEXT  
SESSION\_ID  
XACT\_STATE

System Statistical  
    @@CONNECTIONS  
    @@CPU\_BUSY  
    @@IDLE  
    @@IO\_BUSY  
    @@PACK\_SENT  
    @@PACKET\_ERRORS  
    @@TIMETICKS  
    @@TOTAL\_ERRORS  
    @@TOTAL\_READ  
    @@TOTAL\_WRITE

Text and Image  
    TEXTPTR  
    TEXTVALID

Trigger  
    COLUMNS\_UPDATED  
    EVENTDATA  
    TRIGGER\_NESTLEVEL  
    UPDATE()

# What are the SQL database functions?

6/28/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Learn about the categories of built-in functions you can use with SQL databases. You can use the built-in functions or create your own user-defined functions.

## Aggregate functions

Aggregate functions perform a calculation on a set of values and return a single value. They are allowed in the select list or the HAVING clause of a SELECT statement. You can use an aggregation in combination with the GROUP BY clause to calculate the aggregation on categories of rows. Use the OVER clause to calculate the aggregation on a specific range of value. The OVER clause cannot follow the GROUPING or GROUPING\_ID aggregations.

All aggregate functions are deterministic, which means they always return the same value when they run on the same input values. For more information, see [Deterministic and Nondeterministic Functions](#).

## Analytic functions

Analytic functions compute an aggregate value based on a group of rows. However, unlike aggregate functions, analytic functions can return multiple rows for each group. You can use analytic functions to compute moving averages, running totals, percentages, or top-N results within a group.

## Ranking functions

Ranking functions return a ranking value for each row in a partition. Depending on the function that is used, some rows might receive the same value as other rows. Ranking functions are nondeterministic.

## Rowset functions

Rowset functions Return an object that can be used like table references in an SQL statement.

## Scalar functions

Operate on a single value and then return a single value. Scalar functions can be used wherever an expression is valid.

### Categories of scalar functions

FUNCTION CATEGORY	DESCRIPTION
Configuration Functions	Return information about the current configuration.
Conversion Functions	Support data type casting and converting.
Cursor Functions	Return information about cursors.

FUNCTION CATEGORY	DESCRIPTION
Date and Time Data Types and Functions	Perform operations on a date and time input values and return string, numeric, or date and time values.
JSON Functions	Validate, query, or change JSON data.
Logical Functions	Perform logical operations.
Mathematical Functions	Perform calculations based on input values provided as parameters to the functions, and return numeric values.
Metadata Functions	Return information about the database and database objects.
Security Functions	Return information about users and roles.
String Functions	Perform operations on a string ( <b>char</b> or <b>varchar</b> ) input value and return a string or numeric value.
System Functions	Perform operations and return information about values, objects, and settings in an instance of SQL Server.
System Statistical Functions	Return statistical information about the system.
Text and Image Functions	Perform operations on text or image input values or columns, and return information about the value.

## Function Determinism

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. For more information, see [Deterministic and Nondeterministic Functions](#)

## Function Collation

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 non-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. For more information, see [Collation Precedence \(Transact-SQL\)](#).

## See Also

[CREATE FUNCTION \(Transact-SQL\)](#)

[Deterministic and Nondeterministic Functions](#)

[Using Stored Procedures \(MDX\)](#)

# Aggregate Functions (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Aggregate functions perform a calculation on a set of values and return a single value. Except for COUNT, aggregate functions ignore null values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement.

All aggregate functions are deterministic. This means aggregate functions return the same value any time that they are called by using a specific set of input values. For more information about function determinism, see [Deterministic and Nondeterministic Functions](#). The [OVER clause](#) may follow all aggregate functions except GROUPING and GROUPING\_ID.

Aggregate functions can be used as expressions only in the following:

- The select list of a SELECT statement (either a subquery or an outer query).
- A HAVING clause.

Transact-SQL provides the following aggregate functions:

AVG	MIN
CHECKSUM_AGG	SUM
COUNT	STDEV
COUNT_BIG	STDEVP
GROUPING	VAR
GROUPING_ID	VARP
MAX	

## See also

[Built-in Functions \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# AVG (Transact-SQL)

7/31/2017 • 5 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the average of the values in a group. Null values are ignored.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
AVG ( [ ALL | DISTINCT ] expression )
      OVER ( [ partition_by_clause ] order_by_clause )
```

## Arguments

**ALL**

Applies the aggregate function to all values. ALL is the default.

**DISTINCT**

Specifies that AVG be performed only on each unique instance of a value, regardless of how many times the value occurs.

**expression**

Is an [expression](#) of the exact numeric or approximate numeric data type category, except for the **bit** data type. Aggregate functions and subqueries are not permitted.

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

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. *order\_by\_clause* is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return types

The return type is determined by the type of the evaluated result of *expression*.

EXPRESSION RESULT	RETURN TYPE
<b>tinyint</b>	<b>int</b>
<b>smallint</b>	<b>int</b>
<b>int</b>	<b>int</b>
<b>bigint</b>	<b>bigint</b>
<b>decimal</b> category (p, s)	<b>decimal(38, s)</b> divided by <b>decimal(10, 0)</b>

EXPRESSION RESULT	RETURN TYPE
<b>money</b> and <b>smallmoney</b> category	<b>money</b>
<b>float</b> and <b>real</b> category	<b>float</b>

## Remarks

If the data type of *expression* is an alias data type, the return type is also of the alias data type. However, if the base data type of the alias data type is promoted, for example from **tinyint** to **int**, the return value is of the promoted data type and not the alias data type.

AVG () computes the average of a set of values by dividing the sum of those values by the count of nonnull values. If the sum exceeds the maximum value for the data type of the return value an error will be returned.

AVG is a deterministic function when used without the OVER and ORDER BY clauses. It is nondeterministic when specified with the OVER and ORDER BY clauses. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Using the SUM and AVG functions for calculations

The following example calculates the average vacation hours and the sum of sick leave hours that the vice presidents of Adventure Works Cycles have used. Each of these aggregate functions produces a single summary value for all the retrieved rows. The example uses the AdventureWorks2012 database.

```
SELECT AVG(VacationHours)AS 'Average vacation hours',
       SUM(SickLeaveHours) AS 'Total sick leave hours'
  FROM HumanResources.Employee
 WHERE JobTitle LIKE 'Vice President%';
```

Here is the result set.

Average vacation hours	Total sick leave hours
-----	-----
25	97
(1 row(s) affected)	

### B. Using the SUM and AVG functions with a GROUP BY clause

When used with a **GROUP BY** clause, each aggregate function produces a single value for each group, instead of for the whole table. The following example produces summary values for each sales territory in the AdventureWorks2012 database. The summary lists the average bonus received by the sales people in each territory and the sum of year-to-date sales for each territory.

```
SELECT TerritoryID, AVG(Bonus)as 'Average bonus', SUM(SalesYTD) as 'YTD sales'
  FROM Sales.SalesPerson
 GROUP BY TerritoryID;
 GO
```

Here is the result set.

TerritoryID	Average Bonus	YTD Sales
NULL	0.00	1252127.9471
1	4133.3333	4502152.2674
2	4100.00	3763178.1787
3	2500.00	3189418.3662
4	2775.00	6709904.1666
5	6700.00	2315185.611
6	2750.00	4058260.1825
7	985.00	3121616.3202
8	75.00	1827066.7118
9	5650.00	1421810.9242
10	5150.00	4116871.2277

(11 row(s) affected)

### C. Using AVG with DISTINCT

The following statement returns the average list price of products in the AdventureWorks2012 database. By specifying DISTINCT, only unique values are considered in the calculation.

```
SELECT AVG(DISTINCT ListPrice)
FROM Production.Product;
```

Here is the result set.

```
-----
437.4042
(1 row(s) affected)
```

### D. Using AVG without DISTINCT

Without DISTINCT, the `AVG` function finds the average list price of all products in the `Product` table in the AdventureWorks2012 database including any duplicate values.

```
SELECT AVG(ListPrice)
FROM Production.Product;
```

Here is the result set.

```
-----
438.6662
(1 row(s) affected)
```

### E. Using the OVER clause

The following example uses the AVG function with the OVER clause to provide a moving average of yearly sales for each territory in the `Sales.SalesPerson` table in the AdventureWorks2012 database. The data is partitioned by `TerritoryID` and logically ordered by `SalesYTD`. This means that the AVG function is computed for each territory based on the sales year. Notice that for `TerritoryID` 1, there are two rows for sales year 2005 representing the two sales people with sales that year. The average sales for these two rows is computed and then the third row representing sales for the year 2006 is included in the computation.

```

SELECT BusinessEntityID, TerritoryID
    ,DATEPART(yy,ModifiedDate) AS SalesYear
    ,CONVERT(varchar(20),SalesYTD,1) AS SalesYTD
    ,CONVERT(varchar(20),AVG(SalesYTD) OVER (PARTITION BY TerritoryID
        ORDER BY DATEPART(yy,ModifiedDate)
        ),1) AS MovingAvg
    ,CONVERT(varchar(20),SUM(SalesYTD) OVER (PARTITION BY TerritoryID
        ORDER BY DATEPART(yy,ModifiedDate)
        ),1) AS CumulativeTotal
FROM Sales.SalesPerson
WHERE TerritoryID IS NULL OR TerritoryID < 5
ORDER BY TerritoryID,SalesYear;

```

Here is the result set.

BusinessEntityID	TerritoryID	SalesYear	SalesYTD	MovingAvg	CumulativeTotal
274	NULL	2005	559,697.56	559,697.56	559,697.56
287	NULL	2006	519,905.93	539,801.75	1,079,603.50
285	NULL	2007	172,524.45	417,375.98	1,252,127.95
283	1	2005	1,573,012.94	1,462,795.04	2,925,590.07
280	1	2005	1,352,577.13	1,462,795.04	2,925,590.07
284	1	2006	1,576,562.20	1,500,717.42	4,502,152.27
275	2	2005	3,763,178.18	3,763,178.18	3,763,178.18
277	3	2005	3,189,418.37	3,189,418.37	3,189,418.37
276	4	2005	4,251,368.55	3,354,952.08	6,709,904.17
281	4	2005	2,458,535.62	3,354,952.08	6,709,904.17

(10 row(s) affected)

In this example, the OVER clause does not include PARTITION BY. This means that the function will be applied to all rows returned by the query. The ORDER BY clause specified in the OVER clause determines the logical order to which the AVG function is applied. The query returns a moving average of sales by year for all sales territories specified in the WHERE clause. The ORDER BY clause specified in the SELECT statement determines the order in which the rows of the query are displayed.

```

SELECT BusinessEntityID, TerritoryID
    ,DATEPART(yy,ModifiedDate) AS SalesYear
    ,CONVERT(varchar(20),SalesYTD,1) AS SalesYTD
    ,CONVERT(varchar(20),AVG(SalesYTD) OVER (ORDER BY DATEPART(yy,ModifiedDate)
        ),1) AS MovingAvg
    ,CONVERT(varchar(20),SUM(SalesYTD) OVER (ORDER BY DATEPART(yy,ModifiedDate)
        ),1) AS CumulativeTotal
FROM Sales.SalesPerson
WHERE TerritoryID IS NULL OR TerritoryID < 5
ORDER BY SalesYear;

```

Here is the result set.

BusinessEntityID	TerritoryID	SalesYear	SalesYTD	MovingAvg	CumulativeTotal
274	NULL	2005	559,697.56	2,449,684.05	17,147,788.35
275	2	2005	3,763,178.18	2,449,684.05	17,147,788.35
276	4	2005	4,251,368.55	2,449,684.05	17,147,788.35
277	3	2005	3,189,418.37	2,449,684.05	17,147,788.35
280	1	2005	1,352,577.13	2,449,684.05	17,147,788.35
281	4	2005	2,458,535.62	2,449,684.05	17,147,788.35
283	1	2005	1,573,012.94	2,449,684.05	17,147,788.35
284	1	2006	1,576,562.20	2,138,250.72	19,244,256.47
287	NULL	2006	519,905.93	2,138,250.72	19,244,256.47
285	NULL	2007	172,524.45	1,941,678.09	19,416,780.93

## See also

[Aggregate Functions \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# CHECKSUM\_AGG (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the checksum of the values in a group. Null values are ignored. Can be followed by the [OVER clause](#).

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CHECKSUM_AGG ( [ ALL | DISTINCT ] expression )
```

## Arguments

### ALL

Applies the aggregate function to all values. ALL is the default.

### DISTINCT

Specifies that CHECKSUM\_AGG returns the checksum of unique values.

### *expression*

Is an integer [expression](#). Aggregate functions and subqueries are not allowed.

## Return types

Returns the checksum of all *expression* values as [int](#).

## Remarks

CHECKSUM\_AGG can be used to detect changes in a table.

The order of the rows in the table does not affect the result of CHECKSUM\_AGG. Also, CHECKSUM\_AGG functions may be used with the DISTINCT keyword and the GROUP BY clause.

If one of the values in the expression list changes, the checksum of the list also generally changes. However, there is a small chance that the checksum will not change.

CHECKSUM\_AGG has similar functionality with other aggregate functions. For more information, see [Aggregate Functions \(Transact-SQL\)](#).

## Examples

The following example uses `CHECKSUM_AGG` to detect changes in the `Quantity` column of the `ProductInventory` table in the AdventureWorks2012 database.

```
--Get the checksum value before the column value is changed.  
SELECT CHECKSUM_AGG(CAST(Quantity AS int))  
FROM Production.ProductInventory;  
GO
```

Here is the result set.

```
-----  
262
```

```
UPDATE Production.ProductInventory  
SET Quantity=125  
WHERE Quantity=100;  
GO  
--Get the checksum of the modified column.  
SELECT CHECKSUM_AGG(CAST(Quantity AS int))  
FROM Production.ProductInventory;
```

Here is the result set.

```
-----  
287
```

## See also

[CHECKSUM \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# COUNT (Transact-SQL)

7/31/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the number of items in a group. COUNT works like the [COUNT\\_BIG](#) function. The only difference between the two functions is their return values. COUNT always returns an **int** data type value. COUNT\_BIG always returns a **bigint** data type value.



## Syntax

```
-- Syntax for SQL Server and Azure SQL Database

COUNT ( { [ [ ALL | DISTINCT ] expression ] | * } )
    [ OVER (
        [ partition_by_clause ]
        [ order_by_clause ]
        [ ROW_or_RANGE_clause ]
    ) ]
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse

-- Aggregation Function Syntax
COUNT ( { [ [ ALL | DISTINCT ] expression ] | * } )

-- Analytic Function Syntax
COUNT ( { expression | * } ) OVER ( [ <partition_by_clause> ] )
```

## Arguments

### ALL

Applies the aggregate function to all values. ALL is the default.

### DISTINCT

Specifies that COUNT returns the number of unique nonnull values.

### *expression*

Is an [expression](#) of any type except **text**, **image**, or **ntext**. Aggregate functions and subqueries are not permitted.

\*

Specifies that all rows should be counted to return the total number of rows in a table. COUNT(\*) takes no parameters and cannot be used with DISTINCT. COUNT(\*) does not require an *expression* parameter because, by definition, it does not use information about any particular column. COUNT(\*) returns the number of rows in a specified table without getting rid of duplicates. It counts each row separately. This includes rows that contain null values.

### OVER ( [ *partition\_by\_clause* ] [ *order\_by\_clause* ] [ *ROW\_or\_RANGE\_clause* ] )

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. For more information, see [OVER Clause](#)

(Transact-SQL).

## Return types

**int**

## Remarks

COUNT(\*) returns the number of items in a group. This includes NULL values and duplicates.

COUNT(ALL *expression*) evaluates *expression* for each row in a group and returns the number of nonnull values.

COUNT(DISTINCT *expression*) evaluates *expression* for each row in a group and returns the number of unique, nonnull values.

For return values greater than 2<sup>31</sup>-1, COUNT produces an error. Use COUNT\_BIG instead.

COUNT is a deterministic function when used without the OVER and ORDER BY clauses. It is nondeterministic when specified with the OVER and ORDER BY clauses. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Using COUNT and DISTINCT

The following example lists the number of different titles that an employee who works at Adventure Works Cycles can hold.

```
SELECT COUNT(DISTINCT Title)
FROM HumanResources.Employee;
GO
```

Here is the result set.

```
-----
```

```
67
```

```
(1 row(s) affected)
```

### B. Using COUNT(\*)

The following example finds the total number of employees who work at Adventure Works Cycles.

```
SELECT COUNT(*)
FROM HumanResources.Employee;
GO
```

Here is the result set.

```
-----
```

```
290
```

```
(1 row(s) affected)
```

### C. Using COUNT(\*) with other aggregates

The following example shows that COUNT(\*) can be combined with other aggregate functions in the select list. The example uses the AdventureWorks2012 database.

```

SELECT COUNT(*), AVG(Bonus)
FROM Sales.SalesPerson
WHERE SalesQuota > 25000;
GO

```

Here is the result set.

```
-----
```

```
14 3472.1428
```

```
(1 row(s) affected)
```

### C. Using the OVER clause

The following example uses the MIN, MAX, AVG and COUNT functions with the OVER clause to provide aggregated values for each department in the `HumanResources.Department` table in the AdventureWorks2012 database.

```

SELECT DISTINCT Name
    , MIN(Rate) OVER (PARTITION BY edh.DepartmentID) AS MinSalary
    , MAX(Rate) OVER (PARTITION BY edh.DepartmentID) AS MaxSalary
    , AVG(Rate) OVER (PARTITION BY edh.DepartmentID) AS AvgSalary
    , COUNT(edh.BusinessEntityID) OVER (PARTITION BY edh.DepartmentID) AS EmployeesPerDept
FROM HumanResources.EmployeePayHistory AS eph
JOIN HumanResources.EmployeeDepartmentHistory AS edh
    ON eph.BusinessEntityID = edh.BusinessEntityID
JOIN HumanResources.Department AS d
    ON d.DepartmentID = edh.DepartmentID
WHERE edh.EndDate IS NULL
ORDER BY Name;

```

Here is the result set.

Name	MinSalary	MaxSalary	AvgSalary	EmployeesPerDept
Document Control	10.25	17.7885	14.3884	5
Engineering	32.6923	63.4615	40.1442	6
Executive	39.06	125.50	68.3034	4
Facilities and Maintenance	9.25	24.0385	13.0316	7
Finance	13.4615	43.2692	23.935	10
Human Resources	13.9423	27.1394	18.0248	6
Information Services	27.4038	50.4808	34.1586	10
Marketing	13.4615	37.50	18.4318	11
Production	6.50	84.1346	13.5537	195
Production Control	8.62	24.5192	16.7746	8
Purchasing	9.86	30.00	18.0202	14
Quality Assurance	10.5769	28.8462	15.4647	6
Research and Development	40.8654	50.4808	43.6731	4
Sales	23.0769	72.1154	29.9719	18
Shipping and Receiving	9.00	19.2308	10.8718	6
Tool Design	8.62	29.8462	23.5054	6

```
(16 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D. Using COUNT and DISTINCT

The following example lists the number of different titles that an employee who works at a specific company can

hold.

```
USE ssawPDW;

SELECT COUNT(DISTINCT Title)
FROM dbo.DimEmployee;
```

Here is the result set.

```
-----
```

67

### E. Using COUNT(\*)

The following example returns the total number of rows in the `dbo.DimEmployee` table.

```
USE ssawPDW;

SELECT COUNT(*)
FROM dbo.DimEmployee;
```

Here is the result set.

```
-----
```

296

### F. Using COUNT(\*) with other aggregates

The following example combines `COUNT(*)` with other aggregate functions in the SELECT list. The query returns the number of sales representatives with a annual sales quota greater than \$500,000 and the average sales quota.

```
USE ssawPDW;

SELECT COUNT(EmployeeKey) AS TotalCount, AVG(SalesAmountQuota) AS [Average Sales Quota]
FROM dbo.FactSalesQuota
WHERE SalesAmountQuota > 500000 AND CalendarYear = 2001;
```

Here is the result set.

TotalCount	Average Sales Quota
-----	-----
10	683800.0000

### G. Using COUNT with HAVING

The following example uses COUNT with the HAVING clause to return the departments in a company that have more than 15 employees.

```
USE ssawPDW;

SELECT DepartmentName,
       COUNT(EmployeeKey)AS EmployeesInDept
FROM dbo.DimEmployee
GROUP BY DepartmentName
HAVING COUNT(EmployeeKey) > 15;
```

Here is the result set.

```
DepartmentName EmployeesInDept
```

```
----- -----
```

```
Sales 18
```

```
Production 179
```

## H. Using COUNT with OVER

The following example uses COUNT with the OVER clause to return the number of products that are contained in each of the specified sales orders.

```
USE ssawPDW;

SELECT DISTINCT COUNT(ProductKey) OVER(PARTITION BY SalesOrderNumber) AS ProductCount
    ,SalesOrderNumber
FROM dbo.FactInternetSales
WHERE SalesOrderNumber IN (N'S053115',N'S055981');
```

Here is the result set.

```
ProductCount SalesOrderID
```

```
----- -----
```

```
3 S053115
```

```
1 S055981
```

## See also

[Aggregate Functions \(Transact-SQL\)](#)

[COUNT\\_BIG \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# COUNT\_BIG (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the number of items in a group. COUNT\_BIG works like the COUNT function. The only difference between the two functions is their return values. COUNT\_BIG always returns a **bigint** data type value. COUNT always returns an **int** data type value.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server and Azure SQL Database  
  
COUNT_BIG ( { [ ALL | DISTINCT ] expression } | * )  
[ OVER ( [ partition_by_clause ] [ order_by_clause ] ) ]
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse  
  
-- Aggregation Function Syntax  
COUNT_BIG ( { [ ALL | DISTINCT ] expression } | * )  
  
-- Analytic Function Syntax  
COUNT_BIG ( { expression | * } ) OVER ( [ <partition_by_clause> ] )
```

## Arguments

### ALL

Applies the aggregate function to all values. ALL is the default.

### DISTINCT

Specifies that COUNT\_BIG returns the number of unique nonnull values.

### *expression*

Is an [expression](#) of any type. Aggregate functions and subqueries are not permitted.

### \\*

Specifies that all rows should be counted to return the total number of rows in a table. COUNT\_BIG(\\*) takes no parameters and cannot be used with DISTINCT. COUNT\_BIG(\*) does not require an \*expression parameter because, by definition, it does not use information about any particular column. COUNT\_BIG(\\*) returns the number of rows in a specified table without getting rid of duplicates. It counts each row separately. This includes rows that contain null values.

### ALL

Applies the aggregate function to all values. ALL is the default.

### DISTINCT

Specifies that AVG be performed only on each unique instance of a value, regardless of how many times the value occurs.

### *expression*

Is an [expression](#) of the exact numeric or approximate numeric data type category, except for the **bit** data type. Aggregate functions and subqueries are not permitted.

`OVER ( [ partition_by_clause ] [ order_by_clause ] )`

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return types

**bigint**

## Remarks

`COUNT_BIG(*)` returns the number of items in a group. This includes NULL values and duplicates.

`COUNT_BIG (ALL expression)` evaluates *expression* for each row in a group and returns the number of nonnull values.

`COUNT_BIG (DISTINCT expression)` evaluates *expression* for each row in a group and returns the number of unique, nonnull values.

`COUNT_BIG` is a deterministic function when used without the OVER and ORDER BY clauses. It is nondeterministic when specified with the OVER and ORDER BY clauses. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

For examples, see [COUNT \(Transact-SQL\)](#).

## See also

[Aggregate Functions \(Transact-SQL\)](#)

[COUNT \(Transact-SQL\)](#)

[int, bigint, smallint, and tinyint \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# GROUPING (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse  
✗ Parallel Data Warehouse

Indicates whether a specified column expression in a GROUP BY list is aggregated or not. GROUPING returns 1 for aggregated or 0 for not aggregated in the result set. GROUPING can be used only in the SELECT <select> list, HAVING, and ORDER BY clauses when GROUP BY is specified.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
GROUPING ( <column_expression> )
```

## Arguments

<column\_expression>

Is a column or an expression that contains a column in a [GROUP BY](#) clause.

## Return Types

**tinyint**

## Remarks

GROUPING is used to distinguish the null values that are returned by ROLLUP, CUBE or GROUPING SETS from standard null values. The NULL returned as the result of a ROLLUP, CUBE or GROUPING SETS operation is a special use of NULL. This acts as a column placeholder in the result set and means all.

## Examples

The following example groups `SalesQuota` and aggregates `SaleYTD` amounts in the AdventureWorks2012 database. The `GROUPING` function is applied to the `SalesQuota` column.

```
SELECT SalesQuota, SUM(SalesYTD) 'TotalSalesYTD', GROUPING(SalesQuota) AS 'Grouping'  
FROM Sales.SalesPerson  
GROUP BY SalesQuota WITH ROLLUP;  
GO
```

The result set shows two null values under `SalesQuota`. The first `NULL` represents the group of null values from this column in the table. The second `NULL` is in the summary row added by the ROLLUP operation. The summary row shows the `TotalSalesYTD` amounts for all `SalesQuota` groups and is indicated by `1` in the `Grouping` column.

Here is the result set.

SalesQuota	TotalSalesYTD	Grouping
		1

----- ----- -----

```
NULL 1533087.5999 0
```

```
250000.00 33461260.59 0
```

```
300000.00 9299677.9445 0
```

```
NULL 44294026.1344 1
```

```
(4 row(s) affected)
```

## See Also

[GROUPING\\_ID \(Transact-SQL\)](#)

[GROUP BY \(Transact-SQL\)](#)

# GROUPING\_ID (Transact-SQL)

7/31/2017 • 9 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Is a function that computes the level of grouping. GROUPING\_ID can be used only in the SELECT <select> list, HAVING, or ORDER BY clauses when GROUP BY is specified.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
GROUPING_ID ( <column_expression>[ ,...n ] )
```

## Arguments

<column\_expression>

Is a *column\_expression* in a GROUP BY clause.

## Return Type

**int**

## Remarks

The GROUPING\_ID <column\_expression> must exactly match the expression in the GROUP BY list. For example, if you are grouping by DATEPART (yyyy, <column name>), use GROUPING\_ID (DATEPART (yyyy, <column name>)); or if you are grouping by <column name>, use GROUPING\_ID (<column name>).

## Comparing GROUPING\_ID () to GROUPING ()

GROUPING\_ID (<column\_expression> [ ,...n ] ) inputs the equivalent of the GROUPING (<column\_expression>) return for each column in its column list in each output row as a string of ones and zeros. GROUPING\_ID interprets that string as a base-2 number and returns the equivalent integer. For example consider the following statement:

```
SELECT a, b, c, SUM(d), ``GROUPING_ID(a,b,c)`` FROM T GROUP BY <group by list> . The following table shows the GROUPING_ID () input and output values.
```

COLUMNS AGGREGATED	GROUPING_ID (A, B, C) INPUT = GROUPING(A) + GROUPING(B) + GROUPING(C)	GROUPING_ID () OUTPUT
a	100	4
b	010	2
c	001	1
ab	110	6

COLUMNS AGGREGATED	GROUPING_ID (A, B, C) INPUT = GROUPING(A) + GROUPING(B) + GROUPING(C)	GROUPING_ID () OUTPUT
ac	101	5
bc	011	3
abc	111	7

## Technical Definition of GROUPING\_ID ()

Each GROUPING\_ID argument must be an element of the GROUP BY list. GROUPING\_ID () returns an **integer** bitmap whose lowest N bits may be lit. A lit **bit** indicates the corresponding argument is not a grouping column for the given output row. The lowest-order **bit** corresponds to argument N, and the N-1<sup>th</sup> lowest-order **bit** corresponds to argument 1.

## GROUPING\_ID () Equivalents

For a single grouping query, GROUPING (<column\_expression>) is equivalent to GROUPING\_ID (<column\_expression>), and both return 0.

For example, the following statements are equivalent:

Statement A:

```
SELECT GROUPING_ID(A,B)
FROM T
GROUP BY CUBE(A,B)
```

Statement B:

```
SELECT 3 FROM T GROUP BY ()
UNION ALL
SELECT 1 FROM T GROUP BY A
UNION ALL
SELECT 2 FROM T GROUP BY B
UNION ALL
SELECT 0 FROM T GROUP BY A,B
```

## Examples

### A. Using GROUPING\_ID to identify grouping levels

The following example returns the count of employees by [Name] and [Title], [Name], and company total in the AdventureWorks2012 database. `GROUPING_ID()` is used to create a value for each row in the [Title] column that identifies its level of aggregation.

```

SELECT D.Name
    ,CASE
        WHEN GROUPING_ID(D.Name, E.JobTitle) = 0 THEN E.JobTitle
        WHEN GROUPING_ID(D.Name, E.JobTitle) = 1 THEN N'Total: ' + D.Name
        WHEN GROUPING_ID(D.Name, E.JobTitle) = 3 THEN N'Company Total:'
        ELSE N'Unknown'
    END AS N'Job Title'
    ,COUNT(E.BusinessEntityID) AS N'Employee Count'
FROM HumanResources.Employee E
    INNER JOIN HumanResources.EmployeeDepartmentHistory DH
        ON E.BusinessEntityID = DH.BusinessEntityID
    INNER JOIN HumanResources.Department D
        ON D.DepartmentID = DH.DepartmentID
WHERE DH.EndDate IS NULL
    AND D.DepartmentID IN (12,14)
GROUP BY ROLLUP(D.Name, E.JobTitle);

```

## B. Using GROUPING\_ID to filter a result set

### Simple Example

In the following code, to return only the rows that have a count of employees by title, remove the comment characters from `HAVING GROUPING_ID(D.Name, E.JobTitle); = 0` in the AdventureWorks2012 database. To return only rows with a count of employees by department, remove the comment characters from

```
HAVING GROUPING_ID(D.Name, E.JobTitle) = 1;
```

```

SELECT D.Name
    ,E.JobTitle
    ,GROUPING_ID(D.Name, E.JobTitle) AS 'Grouping Level'
    ,COUNT(E.BusinessEntityID) AS N'Employee Count'
FROM HumanResources.Employee AS E
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS DH
        ON E.BusinessEntityID = DH.BusinessEntityID
    INNER JOIN HumanResources.Department AS D
        ON D.DepartmentID = DH.DepartmentID
WHERE DH.EndDate IS NULL
    AND D.DepartmentID IN (12,14)
GROUP BY ROLLUP(D.Name, E.JobTitle)
--HAVING GROUPING_ID(D.Name, E.JobTitle) = 0; --All titles
--HAVING GROUPING_ID(D.Name, E.JobTitle) = 1; --Group by Name;

```

Here is the unfiltered result set.

NAME	TITLE	GROUPING LEVEL	EMPLOYEE COUNT	NAME
Document Control	Control Specialist	0	2	Document Control
Document Control	Document Control Assistant	0	2	Document Control
Document Control	Document Control Manager	0	1	Document Control
Document Control	NULL	1	5	Document Control
Facilities and Maintenance	Facilities Administrative Assistant	0	1	Facilities and Maintenance

NAME	TITLE	GROUPING LEVEL	EMPLOYEE COUNT	NAME
Facilities and Maintenance	Facilities Manager	0	1	Facilities and Maintenance
Facilities and Maintenance	Janitor	0	4	Facilities and Maintenance
Facilities and Maintenance	Maintenance Supervisor	0	1	Facilities and Maintenance
Facilities and Maintenance	NULL	1	7	Facilities and Maintenance
NULL	NULL	3	12	NULL

### Complex Example

The following example uses `GROUPING_ID()` to filter a result set that contains multiple grouping levels by grouping level. Similar code can be used to create a view that has several grouping levels and a stored procedure that calls the view by passing a parameter that filters the view by grouping level. The example uses the AdventureWorks2012 database.

```

DECLARE @Grouping nvarchar(50);
DECLARE @GroupingLevel smallint;
SET @Grouping = N'CountryRegionCode Total';

SELECT @GroupingLevel = (
    CASE @Grouping
        WHEN N'Grand Total' THEN 15
        WHEN N'SalesPerson Total' THEN 14
        WHEN N'Store Total' THEN 13
        WHEN N'Store SalesPerson Total' THEN 12
        WHEN N'CountryRegionCode Total' THEN 11
        WHEN N'Group Total' THEN 7
        ELSE N'Unknown'
    END);

SELECT
    T.[Group]
    ,T.CountryRegionCode
    ,S.Name AS N'Store'
    ,(SELECT P.FirstName + ' ' + P.LastName
        FROM Person.Person AS P
        WHERE P.BusinessEntityID = H.SalesPersonID)
    AS N'Sales Person'
    ,SUM(TotalDue)AS N'TotalSold'
    ,CAST(GROUPING(T.[Group])AS char(1)) +
        CAST(GROUPING(T.CountryRegionCode)AS char(1)) +
        CAST(GROUPING(S.Name)AS char(1)) +
        CAST(GROUPING(H.SalesPersonID)AS char(1))
        AS N'GROUPING base-2'
    ,GROUPING_ID((T.[Group])
        ,(T.CountryRegionCode),(S.Name),(H.SalesPersonID)
        ) AS N'GROUPING_ID'
    ,CASE
        WHEN GROUPING_ID(
            (T.[Group]),(T.CountryRegionCode)
            ,(S.Name),(H.SalesPersonID)
            ) = 15 THEN N'Grand Total'
        WHEN GROUPING_ID(
            (T.[Group]),(T.CountryRegionCode)
            ,(S.Name),(H.SalesPersonID)
            ) = 14 THEN N'SalesPerson Total'
    END

```

```

    ) = 14 THEN N'SalesPerson Total'
WHEN GROUPING_ID(
    (T.[Group]),(T.CountryRegionCode)
    ,(S.Name),(H.SalesPersonID)
    ) = 13 THEN N'Store Total'
WHEN GROUPING_ID(
    (T.[Group]),(T.CountryRegionCode)
    ,(S.Name),(H.SalesPersonID)
    ) = 12 THEN N'Store SalesPerson Total'
WHEN GROUPING_ID(
    (T.[Group]),(T.CountryRegionCode)
    ,(S.Name),(H.SalesPersonID)
    ) = 11 THEN N'CountryRegionCode Total'
WHEN GROUPING_ID(
    (T.[Group]),(T.CountryRegionCode)
    ,(S.Name),(H.SalesPersonID)
    ) = 7 THEN N'Group Total'
ELSE N'Error'
END AS N'Level'
FROM Sales.Customer AS C
INNER JOIN Sales.Store AS S
    ON C.StoreID = S.BusinessEntityID
INNER JOIN Sales.SalesTerritory AS T
    ON C.TerritoryID = T.TerritoryID
INNER JOIN Sales.SalesOrderHeader AS H
    ON C.CustomerID = H.CustomerID
GROUP BY GROUPING SETS ((S.Name,H.SalesPersonID)
    ,(H.SalesPersonID),(S.Name)
    ,(T.[Group]),(T.CountryRegionCode),()
    )
HAVING GROUPING_ID(
    (T.[Group]),(T.CountryRegionCode),(S.Name),(H.SalesPersonID)
    ) = @GroupingLevel
ORDER BY
    GROUPING_ID(S.Name,H.SalesPersonID),GROUPING_ID((T.[Group])
    ,(T.CountryRegionCode)
    ,(S.Name)
    ,(H.SalesPersonID))ASC;

```

## C. Using GROUPING\_ID () with ROLLUP and CUBE to identify grouping levels

The code in the following examples show using `GROUPING()` to compute the `Bit Vector(base-2)` column.

`GROUPING_ID()` is used to compute the corresponding `Integer Equivalent` column. The column order in the `GROUPING_ID()` function is the opposite of the column order of the columns that are concatenated by the `GROUPING()` function.

In these examples, `GROUPING_ID()` is used to create a value for each row in the `Grouping Level` column to identify the level of grouping. Grouping levels are not always a consecutive list of integers that start with 1 (0, 1, 2,...n).

### NOTE

GROUPING and GROUPING\_ID can be used n a HAVING clause to filter a result set.

### ROLLUP Example

In this example, all grouping levels do not appear as they do in the following CUBE example. If the order of the columns in the `ROLLUP` list is changed, the level values in the `Grouping Level` column will also have to be changed. The example uses the AdventureWorks2012 database.

```

SELECT DATEPART(yyyy,OrderDate) AS N'Year'
    ,DATEPART(mm,OrderDate) AS N'Month'
    ,DATEPART(dd,OrderDate) AS N'Day'
    ,SUM(TotalDue) AS N'Total Due'
    ,CAST(GROUPING(DATEPART(dd,OrderDate))AS char(1)) +
        CAST(GROUPING(DATEPART(mm,OrderDate))AS char(1)) +
        CAST(GROUPING(DATEPART(yyyy,OrderDate))AS char(1))
    AS N'Bit Vector(base-2)'
    ,GROUPING_ID(DATEPART(yyyy,OrderDate)
        ,DATEPART(mm,OrderDate)
        ,DATEPART(dd,OrderDate))
    AS N'Integer Equivalent'
,CASE
    WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
        ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
        ) = 0 THEN N'Year Month Day'
    WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
        ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
        ) = 1 THEN N'Year Month'
    WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
        ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
        ) = 2 THEN N'not used'
    WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
        ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
        ) = 3 THEN N'Year'
    WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
        ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
        ) = 4 THEN N'not used'
    WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
        ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
        ) = 5 THEN N'not used'
    WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
        ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
        ) = 6 THEN N'not used'
    WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
        ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
        ) = 7 THEN N'Grand Total'
    ELSE N'Error'
END AS N'Grouping Level'
FROM Sales.SalesOrderHeader
WHERE DATEPART(yyyy,OrderDate) IN(N'2007',N'2008')
    AND DATEPART(mm,OrderDate) IN(1,2)
    AND DATEPART(dd,OrderDate) IN(1,2)
GROUP BY ROLLUP(DATEPART(yyyy,OrderDate)
    ,DATEPART(mm,OrderDate)
    ,DATEPART(dd,OrderDate))
ORDER BY GROUPING_ID(DATEPART(mm,OrderDate)
    ,DATEPART(yyyy,OrderDate)
    ,DATEPART(dd,OrderDate)
    )
    ,DATEPART(yyyy,OrderDate)
    ,DATEPART(mm,OrderDate)
    ,DATEPART(dd,OrderDate);

```

Here is a partial result set.

YEAR	MONTH	DAY	TOTAL DUE	BIT VECTOR (BASE-2)	INTEGER EQUIVALENT	GROUPING LEVEL
2007	1	1	1497452.606 6	000	0	Year Month Day
2007	1	2	21772.3494	000	0	Year Month Day

YEAR	MONTH	DAY	TOTAL DUE	BIT VECTOR (BASE-2)	INTEGER EQUIVALENT	GROUPING LEVEL
2007	2	1	2705653.591 3	000	0	Year Month Day
2007	2	2	21684.4068	000	0	Year Month Day
2008	1	1	1908122.096 7	000	0	Year Month Day
2008	1	2	46458.0691	000	0	Year Month Day
2008	2	1	3108771.972 9	000	0	Year Month Day
2008	2	2	54598.5488	000	0	Year Month Day
2007	1	NULL	1519224.956	100	1	Year Month
2007	2	NULL	2727337.998 1	100	1	Year Month
2008	1	NULL	1954580.165 8	100	1	Year Month
2008	2	NULL	3163370.521 7	100	1	Year Month
2007	NULL	NULL	4246562.954 1	110	3	Year
2008	NULL	NULL	5117950.687 5	110	3	Year
NULL	NULL	NULL	9364513.641 6	111	7	Grand Total

#### CUBE Example

In this example, the `GROUPING_ID()` function is used to create a value for each row in the `Grouping Level` column to identify the level of grouping.

Unlike `ROLLUP` in the previous example, `CUBE` outputs all grouping levels. If the order of the columns in the `CUBE` list is changed, the level values in the `Grouping Level` column will also have to be changed. The example uses the AdventureWorks2012 database

```

SELECT DATEPART(yyyy,OrderDate) AS N'Year'
    ,DATEPART(mm,OrderDate) AS N'Month'
    ,DATEPART(dd,OrderDate) AS N'Day'
    ,SUM(TotalDue) AS N'Total Due'
    ,CAST(GROUPING(DATEPART(dd,OrderDate))AS char(1)) +
        CAST(GROUPING(DATEPART(mm,OrderDate))AS char(1)) +
        CAST(GROUPING(DATEPART(yyyy,OrderDate))AS char(1))
        AS N'bit Vector(base-2)'
    ,GROUPING_ID(DATEPART(yyyy,OrderDate)
        ,DATEPART(mm,OrderDate)
        ,DATEPART(dd,OrderDate))
        AS N'Integer Equivalent'
    ,CASE
        WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
            ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
            ) = 0 THEN N'Year Month Day'
        WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
            ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
            ) = 1 THEN N'Year Month'
        WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
            ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
            ) = 2 THEN N'Year Day'
        WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
            ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
            ) = 3 THEN N'Year'
        WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
            ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
            ) = 4 THEN N'Month Day'
        WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
            ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
            ) = 5 THEN N'Month'
        WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
            ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
            ) = 6 THEN N'Day'
        WHEN GROUPING_ID(DATEPART(yyyy,OrderDate)
            ,DATEPART(mm,OrderDate),DATEPART(dd,OrderDate)
            ) = 7 THEN N'Grand Total'
        ELSE N'Error'
    END AS N'Grouping Level'
FROM Sales.SalesOrderHeader
WHERE DATEPART(yyyy,OrderDate) IN(N'2007',N'2008')
    AND DATEPART(mm,OrderDate) IN(1,2)
    AND DATEPART(dd,OrderDate) IN(1,2)
GROUP BY CUBE(DATEPART(yyyy,OrderDate)
    ,DATEPART(mm,OrderDate)
    ,DATEPART(dd,OrderDate))
ORDER BY GROUPING_ID(DATEPART(yyyy,OrderDate)
    ,DATEPART(mm,OrderDate)
    ,DATEPART(dd,OrderDate)
    )
    ,DATEPART(yyyy,OrderDate)
    ,DATEPART(mm,OrderDate)
    ,DATEPART(dd,OrderDate);

```

Here is a partial result set.

YEAR	MONTH	DAY	TOTAL DUE	BIT VECTOR (BASE-2)	INTEGER EQUIVALENT	GROUPING LEVEL
2007	1	1	1497452.606 6	000	0	Year Month Day
2007	1	2	21772.3494	000	0	Year Month Day

YEAR	MONTH	DAY	TOTAL DUE	BIT VECTOR (BASE-2)	INTEGER EQUIVALENT	GROUPING LEVEL
2007	2	1	2705653.591 3	000	0	Year Month Day
2007	2	2	21684.4068	000	0	Year Month Day
2008	1	1	1908122.096 7	000	0	Year Month Day
2008	1	2	46458.0691	000	0	Year Month Day
2008	2	1	3108771.972 9	000	0	Year Month Day
2008	2	2	54598.5488	000	0	Year Month Day
2007	1	NULL	1519224.956	100	1	Year Month
2007	2	NULL	2727337.998 1	100	1	Year Month
2008	1	NULL	1954580.165 8	100	1	Year Month
2008	2	NULL	3163370.521 7	100	1	Year Month
2007	NULL	1	4203106.197 9	010	2	Year Day
2007	NULL	2	43456.7562	010	2	Year Day
2008	NULL	1	5016894.069 6	010	2	Year Day
2008	NULL	2	101056.6179	010	2	Year Day
2007	NULL	NULL	4246562.954 1	110	3	Year
2008	NULL	NULL	5117950.687 5	110	3	Year
NULL	1	1	3405574.703 3	001	4	Month Day
NULL	1	2	68230.4185	001	4	Month Day
NULL	2	1	5814425.564 2	001	4	Month Day

YEAR	MONTH	DAY	TOTAL DUE	BIT VECTOR (BASE-2)	INTEGER EQUIVALENT	GROUPING LEVEL
NULL	2	2	76282.9556	001	4	Month Day
NULL	1	NULL	3473805.1218	101	5	Month
NULL	2	NULL	5890708.5198	101	5	Month
NULL	NULL	1	9220000.2675	011	6	Day
NULL	NULL	2	144513.3741	011	6	Day
NULL	NULL	NULL	9364513.6416	111	7	Grand Total

## See Also

[GROUPING \(Transact-SQL\)](#)

[GROUP BY \(Transact-SQL\)](#)

# MAX (Transact-SQL)

8/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns the maximum value in the expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
MAX ( [ ALL | DISTINCT ] expression )
      OVER ( [ partition_by_clause ] order_by_clause )
```

## Arguments

### ALL

Applies the aggregate function to all values. ALL is the default.

### DISTINCT

Specifies that each unique value is considered. DISTINCT is not meaningful with MAX and is available for ISO compatibility only.

### *expression*

Is a constant, column name, or function, and any combination of arithmetic, bitwise, and string operators. MAX can be used with **numeric**, **character**, **uniqueidentifier**, and **datetime** columns, but not with **bit** columns. Aggregate functions and subqueries are not permitted.

For more information, see [Expressions \(Transact-SQL\)](#).

### OVER ( [ *partition\_by\_clause* ] *order\_by\_clause* )

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. *order\_by\_clause* is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

Returns a value same as *expression*.

## Remarks

MAX ignores any null values.

For character columns, MAX finds the highest value in the collating sequence.

MAX is a deterministic function when used without the OVER and ORDER BY clauses. It is nondeterministic when specified with the OVER and ORDER BY clauses. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

## A. Simple example

The following example returns the highest (maximum) tax rate in the AdventureWorks2012 database.

```
SELECT MAX(TaxRate)
FROM Sales.SalesTaxRate;
GO
```

Here is the result set.

```
-----
19.60
Warning, null value eliminated from aggregate.

(1 row(s) affected)
```

## B. Using the OVER clause

The following example uses the MIN, MAX, AVG, and COUNT functions with the OVER clause to provide aggregated values for each department in the `HumanResources.Department` table in the AdventureWorks2012 database.

```
SELECT DISTINCT Name
    , MIN(Rate) OVER (PARTITION BY edh.DepartmentID) AS MinSalary
    , MAX(Rate) OVER (PARTITION BY edh.DepartmentID) AS MaxSalary
    , AVG(Rate) OVER (PARTITION BY edh.DepartmentID) AS AvgSalary
    , COUNT(edh.BusinessEntityID) OVER (PARTITION BY edh.DepartmentID) AS EmployeesPerDept
FROM HumanResources.EmployeePayHistory AS eph
JOIN HumanResources.EmployeeDepartmentHistory AS edh
    ON eph.BusinessEntityID = edh.BusinessEntityID
JOIN HumanResources.Department AS d
    ON d.DepartmentID = edh.DepartmentID
WHERE edh.EndDate IS NULL
ORDER BY Name;
```

Here is the result set.

Name	MinSalary	MaxSalary	AvgSalary	EmployeesPerDept
<hr/>				
--				
Document Control	10.25	17.7885	14.3884	5
Engineering	32.6923	63.4615	40.1442	6
Executive	39.06	125.50	68.3034	4
Facilities and Maintenance	9.25	24.0385	13.0316	7
Finance	13.4615	43.2692	23.935	10
Human Resources	13.9423	27.1394	18.0248	6
Information Services	27.4038	50.4808	34.1586	10
Marketing	13.4615	37.50	18.4318	11
Production	6.50	84.1346	13.5537	195
Production Control	8.62	24.5192	16.7746	8
Purchasing	9.86	30.00	18.0202	14
Quality Assurance	10.5769	28.8462	15.4647	6
Research and Development	40.8654	50.4808	43.6731	4
Sales	23.0769	72.1154	29.9719	18
Shipping and Receiving	9.00	19.2308	10.8718	6
Tool Design	8.62	29.8462	23.5054	6

(16 row(s) affected)

## C. Using MAX with character data

The following example returns the database name that sorts as the last name alphabetically. The example uses `WHERE database_id < 5`, to consider only the system databases.

```
SELECT MAX(name) FROM sys.databases WHERE database_id < 5;
```

The last system database is `tempdb`.

## See Also

[Aggregate Functions \(Transact-SQL\)](#)  
[OVER Clause \(Transact-SQL\)](#)

# MIN (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the minimum value in the expression. May be followed by the [OVER clause](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server and Azure SQL Database  
  
MIN ( [ ALL | DISTINCT ] expression )  
      OVER ( [ partition_by_clause ] order_by_clause )
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse  
  
-- Aggregation Function Syntax  
MIN ( [ ALL | DISTINCT ] expression )  
  
-- Aggregation Function Syntax  
MIN ( expression ) OVER ( [ <partition_by_clause> ] [ <order_by_clause> ] )
```

## Arguments

### ALL

Applies the aggregate function to all values. ALL is the default.

### DISTINCT

Specifies that each unique value is considered. DISTINCT is not meaningful with MIN and is available for ISO compatibility only.

### *expression*

Is a constant, column name, or function, and any combination of arithmetic, bitwise, and string operators. MIN can be used with **numeric**, **char**, **varchar**, **uniqueidentifier**, or **datetime** columns, but not with **bit** columns.

Aggregate functions and subqueries are not permitted.

For more information, see [Expressions \(Transact-SQL\)](#).

### OVER ( [ *partition\_by\_clause* ] *order\_by\_clause* )

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. *order\_by\_clause* is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

Returns a value same as *expression*.

## Remarks

MIN ignores any null values.

With character data columns, MIN finds the value that is lowest in the sort sequence.

MIN is a deterministic function when used without the OVER and ORDER BY clauses. It is nondeterministic when specified with the OVER and ORDER BY clauses. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Simple example

The following example returns the lowest (minimum) tax rate. The example uses the AdventureWorks2012 database

```
SELECT MIN(TaxRate)
FROM Sales.SalesTaxRate;
GO
```

Here is the result set.

```
-----
5.00
(1 row(s) affected)
```

### B. Using the OVER clause

The following example uses the MIN, MAX, AVG and COUNT functions with the OVER clause to provide aggregated values for each department in the `HumanResources.Department` table in the AdventureWorks2012 database.

```
SELECT DISTINCT Name
    , MIN(Rate) OVER (PARTITION BY edh.DepartmentID) AS MinSalary
    , MAX(Rate) OVER (PARTITION BY edh.DepartmentID) AS MaxSalary
    , AVG(Rate) OVER (PARTITION BY edh.DepartmentID) AS AvgSalary
    , COUNT(edh.BusinessEntityID) OVER (PARTITION BY edh.DepartmentID) AS EmployeesPerDept
FROM HumanResources.EmployeePayHistory AS eph
JOIN HumanResources.EmployeeDepartmentHistory AS edh
    ON eph.BusinessEntityID = edh.BusinessEntityID
JOIN HumanResources.Department AS d
    ON d.DepartmentID = edh.DepartmentID
WHERE edh.EndDate IS NULL
ORDER BY Name;
```

Here is the result set.

Name	MinSalary	MaxSalary	AvgSalary
EmployeesPerDept			
--			
Document Control	10.25	17.7885	14.3884
Engineering	32.6923	63.4615	40.1442
Executive	39.06	125.50	68.3034
Facilities and Maintenance	9.25	24.0385	13.0316
Finance	13.4615	43.2692	23.935
Human Resources	13.9423	27.1394	18.0248
Information Services	27.4038	50.4808	34.1586
Marketing	13.4615	37.50	18.4318
Production	6.50	84.1346	13.5537
Production Control	8.62	24.5192	16.7746
Purchasing	9.86	30.00	18.0202
Quality Assurance	10.5769	28.8462	15.4647
Research and Development	40.8654	50.4808	43.6731
Sales	23.0769	72.1154	29.9719
Shipping and Receiving	9.00	19.2308	10.8718
Tool Design	8.62	29.8462	23.5054

(16 row(s) affected)

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Using MIN

The following example uses the MIN aggregate function to return the price of the least expensive (minimum) product in a specified set of sales orders.

```
-- Uses AdventureWorks

SELECT DISTINCT MIN(UnitPrice)
FROM dbo.FactResellerSales
WHERE SalesOrderNumber IN (N'S043659', N'S043660', N'S043664');
```

Here is the result set.

```
-----
5.1865
```

### D. Using MIN with OVER

The following examples use the MIN OVER() analytic function to return the price of the least expensive product in each sales order. The result set is partitioned by the `SalesOrderID` column.

```
-- Uses AdventureWorks

SELECT DISTINCT MIN(UnitPrice) OVER(PARTITION BY SalesOrderNumber) AS LeastExpensiveProduct,
           SalesOrderNumber
  FROM dbo.FactResellerSales
 WHERE SalesOrderNumber IN (N'S043659', N'S043660', N'S043664')
 ORDER BY SalesOrderNumber;
```

Here is the result set.

```
LeastExpensiveProduct SalesOrderID
-----
-----
```

5.1865 S043659

419.4589 S043660

28.8404 S043664

## See Also

[Aggregate Functions \(Transact-SQL\)](#)

[MAX \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# STDEV (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the statistical standard deviation of all values in the specified expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server and Azure SQL Database
```

```
STDEV ( [ ALL | DISTINCT ] expression )
      OVER ( [ partition_by_clause ] order_by_clause )
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse
```

```
-- Aggregate Function Syntax
STDEV ( [ ALL | DISTINCT ] expression )
```

```
-- Analytic Function Syntax
STDEV (expression) OVER ( [ partition_by_clause ] order_by_clause )
```

## Arguments

### ALL

Applies the function to all values. ALL is the default.

### DISTINCT

Specifies that each unique value is considered.

### expression

Is a numeric [expression](#). Aggregate functions and subqueries are not permitted. *expression* is an expression of the exact numeric or approximate numeric data type category, except for the **bit** data type.

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

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. *order\_by\_clause* is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

### float

## Remarks

If STDEV is used on all items in a SELECT statement, each value in the result set is included in the calculation. STDEV can be used with numeric columns only. Null values are ignored.

STDEV is a deterministic function when used without the OVER and ORDER BY clauses. It is nondeterministic when specified with the OVER and ORDER BY clauses. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A: Using STDEV

The following example returns the standard deviation for all bonus values in the `SalesPerson` table in the AdventureWorks2012 database.

```
SELECT STDEV(Bonus)
FROM Sales.SalesPerson;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### B: Using STDEV

The following example returns the standard deviation of the sales quota values in the table `dbo.FactSalesQuota`. The first column contains the standard deviation of all distinct values and the second column contains the standard deviation of all values including any duplicates values.

```
-- Uses AdventureWorks

SELECT STDEV(DISTINCT SalesAmountQuota)AS Distinct_Values, STDEV(SalesAmountQuota) AS All_Values
FROM dbo.FactSalesQuota;
```

Here is the result set.

Distinct_Values	All_Values
-----	-----
398974.27	398450.57

### C. Using STDEV with OVER

The following example returns the standard deviation of the sales quota values for each quarter in a calendar year. Notice that the ORDER BY in the OVER clause orders the STDEV and the ORDER BY of the SELECT statement orders the result set.

```
-- Uses AdventureWorks

SELECT CalendarYear AS Year, CalendarQuarter AS Quarter, SalesAmountQuota AS SalesQuota,
       STDEV(SalesAmountQuota) OVER (ORDER BY CalendarYear, CalendarQuarter) AS StdDeviation
  FROM dbo.FactSalesQuota
 WHERE EmployeeKey = 272 AND CalendarYear = 2002
 ORDER BY CalendarQuarter;
```

Here is the result set.

Year	Quarter	SalesQuota	StdDeviation
-----	-----	-----	-----
2002	1	91000.0000	null
2002	2	140000.0000	34648.23

2002 3 70000.0000 35921.21

2002 4 154000.0000 39752.36

## See Also

[Aggregate Functions \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# STDEVP (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the statistical standard deviation for the population for all values in the specified expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server and Azure SQL Database
```

```
STDEVP ( [ ALL | DISTINCT ] expression )
    OVER ( [ partition_by_clause ] order_by_clause )
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse
```

```
-- Aggregate Function Syntax
STDEVP ( [ ALL | DISTINCT ] expression )

-- Analytic Function Syntax
STDEVP (expression) OVER ( [ partition_by_clause ] order_by_clause)
```

## Arguments

### ALL

Applies the function to all values. ALL is the default.

### DISTINCT

Specifies that each unique value is considered.

### expression

Is a numeric [expression](#). Aggregate functions and subqueries are not permitted. *expression* is an expression of the exact numeric or approximate numeric data type category, except for the **bit** data type.

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

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. *order\_by\_clause* is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

### float

## Remarks

If STDEVP is used on all items in a SELECT statement, each value in the result set is included in the calculation. STDEVP can be used with numeric columns only. Null values are ignored.

STDEVP is a deterministic function when used without the OVER and ORDER BY clauses. It is nondeterministic when specified with the OVER and ORDER BY clauses. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A: Using STDEVP

The following example returns the standard deviation for the population for all bonus values in the `SalesPerson` table in the AdventureWorks2012 database.

```
SELECT STDEVP(Bonus)
FROM Sales.SalesPerson;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### B: Using STDEVP

The following example returns the `STDEVP` of the sales quota values in the table `dbo.FactSalesQuota`. The first column contains the standard deviation of all distinct values and the second column contains the standard deviation of all values including any duplicates values.

```
-- Uses AdventureWorks

SELECT STDEVP(DISTINCT SalesAmountQuota)AS Distinct_Values, STDEVP(SalesAmountQuota) AS All_Values
FROM dbo.FactSalesQuota;SELECT STDEVP(DISTINCT Quantity)AS Distinct_Values, STDEVP(Quantity) AS All_Values
FROM ProductInventory;
```

Here is the result set.

Distinct_Values	All_Values
-----	-----
397676.79	397226.44

### C. Using STDEVP with OVER

The following example returns the `STDEVP` of the sales quota values for each quarter in a calendar year. Notice that the `ORDER BY` in the `OVER` clause orders the `STDEVP` and the `ORDER BY` of the `SELECT` statement orders the result set.

```
-- Uses AdventureWorks

SELECT CalendarYear AS Year, CalendarQuarter AS Quarter, SalesAmountQuota AS SalesQuota,
       STDEVP(SalesAmountQuota) OVER (ORDER BY CalendarYear, CalendarQuarter) AS StdDeviation
  FROM dbo.FactSalesQuota
 WHERE EmployeeKey = 272 AND CalendarYear = 2002
 ORDER BY CalendarQuarter;
```

Here is the result set.

Year	Quarter	SalesQuota	StdDeviation
-----	-----	-----	-----
2002	1	91000.0000	0.00

2002 2 140000.0000 24500.00
-----------------------------

2002 3 70000.0000 29329.55
----------------------------

2002 4 154000.0000 34426.55
-----------------------------

## See Also

[Aggregate Functions \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# SUM (Transact-SQL)

3/24/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the sum of all the values, or only the DISTINCT values, in the expression. SUM can be used with numeric columns only. Null values are ignored.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server and Azure SQL Database

SUM ( [ ALL | DISTINCT ] expression )
    OVER ( [ partition_by_clause ] order_by_clause )
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse

SUM ( [ ALL | DISTINCT ] expression )
```

## Arguments

### ALL

Applies the aggregate function to all values. ALL is the default.

### DISTINCT

Specifies that SUM return the sum of unique values.

### *expression*

Is a constant, column, or function, and any combination of arithmetic, bitwise, and string operators. *expression* is an expression of the exact numeric or approximate numeric data type category, except for the **bit** data type. Aggregate functions and subqueries are not permitted. For more information, see [Expressions \(Transact-SQL\)](#).

### *OVER ( [ partition\_by\_clause ] order\_by\_clause )*

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. *order\_by\_clause* is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

Returns the summation of all *expression* values in the most precise *expression* data type.

EXPRESSION RESULT	RETURN TYPE
<b>tinyint</b>	<b>int</b>
<b>smallint</b>	<b>int</b>

EXPRESSION RESULT	RETURN TYPE
<b>int</b>	<b>int</b>
<b>bigint</b>	<b>bigint</b>
<b>decimal</b> category (p, s)	<b>decimal(38, s)</b>
<b>money</b> and <b>smallmoney</b> category	<b>money</b>
<b>float</b> and <b>real</b> category	<b>float</b>

## Remarks

SUM is a deterministic function when used without the OVER and ORDER BY clauses. It is nondeterministic when specified with the OVER and ORDER BY clauses. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Using SUM to return summary data

The following examples shows using the SUM function to return summary data in the AdventureWorks2012 database.

```
SELECT Color, SUM(ListPrice), SUM(StandardCost)
FROM Production.Product
WHERE Color IS NOT NULL
    AND ListPrice != 0.00
    AND Name LIKE 'Mountain%'
GROUP BY Color
ORDER BY Color;
GO
```

Here is the result set.

Color
-----
Black 27404.84 5214.9616
Silver 26462.84 14665.6792
White 19.00 6.7926
(3 row(s) affected)

### B. Using the OVER clause

The following example uses the SUM function with the OVER clause to provide a cumulative total of yearly sales for each territory in the `Sales.SalesPerson` table in the AdventureWorks2012 database. The data is partitioned by `TerritoryID` and logically ordered by `SalesYTD`. This means that the SUM function is computed for each territory based on the sales year. Notice that for `TerritoryID` 1, there are two rows for sales year 2005 representing the two sales people with sales that year. The cumulative sales for these two rows is computed and then the third row representing sales for the year 2006 is included in the computation.

```

SELECT BusinessEntityID, TerritoryID
    ,DATEPART(yy,ModifiedDate) AS SalesYear
    ,CONVERT(varchar(20),SalesYTD,1) AS SalesYTD
    ,CONVERT(varchar(20),AVG(SalesYTD) OVER (PARTITION BY TerritoryID
        ORDER BY DATEPART(yy,ModifiedDate)
        ),1) AS MovingAvg
    ,CONVERT(varchar(20),SUM(SalesYTD) OVER (PARTITION BY TerritoryID
        ORDER BY DATEPART(yy,ModifiedDate)
        ),1) AS CumulativeTotal
FROM Sales.SalesPerson
WHERE TerritoryID IS NULL OR TerritoryID < 5
ORDER BY TerritoryID,SalesYear;

```

Here is the result set.

BusinessEntityID	TerritoryID	SalesYear	SalesYTD	MovingAvg	CumulativeTotal
274	NULL	2005	559,697.56	559,697.56	559,697.56
287	NULL	2006	519,905.93	539,801.75	1,079,603.50
285	NULL	2007	172,524.45	417,375.98	1,252,127.95
283	1	2005	1,573,012.94	1,462,795.04	2,925,590.07
280	1	2005	1,352,577.13	1,462,795.04	2,925,590.07
284	1	2006	1,576,562.20	1,500,717.42	4,502,152.27
275	2	2005	3,763,178.18	3,763,178.18	3,763,178.18
277	3	2005	3,189,418.37	3,189,418.37	3,189,418.37
276	4	2005	4,251,368.55	3,354,952.08	6,709,904.17
281	4	2005	2,458,535.62	3,354,952.08	6,709,904.17

(10 row(s) affected)

In this example, the OVER clause does not include PARTITION BY. This means that the function will be applied to all rows returned by the query. The ORDER BY clause specified in the OVER clause determines the logical order to which the SUM function is applied. The query returns a cumulative total of sales by year for all sales territories specified in the WHERE clause. The ORDER BY clause specified in the SELECT statement determines the order in which the rows of the query are displayed.

```

SELECT BusinessEntityID, TerritoryID
    ,DATEPART(yy,ModifiedDate) AS SalesYear
    ,CONVERT(varchar(20),SalesYTD,1) AS SalesYTD
    ,CONVERT(varchar(20),AVG(SalesYTD) OVER (ORDER BY DATEPART(yy,ModifiedDate)
        ),1) AS MovingAvg
    ,CONVERT(varchar(20),SUM(SalesYTD) OVER (ORDER BY DATEPART(yy,ModifiedDate)
        ),1) AS CumulativeTotal
FROM Sales.SalesPerson
WHERE TerritoryID IS NULL OR TerritoryID < 5
ORDER BY SalesYear;

```

Here is the result set.

BusinessEntityID	TerritoryID	SalesYear	SalesYTD	MovingAvg	CumulativeTotal
274	NULL	2005	559,697.56	2,449,684.05	17,147,788.35
275	2	2005	3,763,178.18	2,449,684.05	17,147,788.35
276	4	2005	4,251,368.55	2,449,684.05	17,147,788.35
277	3	2005	3,189,418.37	2,449,684.05	17,147,788.35
280	1	2005	1,352,577.13	2,449,684.05	17,147,788.35
281	4	2005	2,458,535.62	2,449,684.05	17,147,788.35
283	1	2005	1,573,012.94	2,449,684.05	17,147,788.35
284	1	2006	1,576,562.20	2,138,250.72	19,244,256.47
287	NULL	2006	519,905.93	2,138,250.72	19,244,256.47
285	NULL	2007	172,524.45	1,941,678.09	19,416,780.93
(10 row(s) affected)					

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. A simple SUM example

The following example returns the total number of each product sold in the year 2003.

```
-- Uses AdventureWorks

SELECT ProductKey, SUM(SalesAmount) AS TotalPerProduct
FROM dbo.FactInternetSales
WHERE OrderDateKey >= '20030101'
    AND OrderDateKey < '20040101'
GROUP BY ProductKey
ORDER BY ProductKey;
```

Here is a partial result set.

ProductKey	TotalPerProduct
214	31421.0200
217	31176.0900
222	29986.4300
225	7956.1500

### D. Calculating group totals with more than one column

The following example calculates the sum of the `ListPrice` and `StandardCost` for each color listed in the `Product` table.

```
-- Uses AdventureWorks

SELECT Color, SUM(ListPrice)AS TotalList,
       SUM(StandardCost) AS TotalCost
FROM dbo.DimProduct
GROUP BY Color
ORDER BY Color;
```

The first part of the result set is shown below:

Color	TotalList	TotalCost
Black	1000000.0000	1000000.0000

Black 101295.7191 57490.5378

Blue 24082.9484 14772.0524

Grey 125.0000 51.5625

Multi 880.7468 526.4095

NA 3162.3564 1360.6185

## See Also

[Aggregate Functions \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# VAR (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the statistical variance of all values in the specified expression. May be followed by the [OVER clause](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server and Azure SQL Database  
  
VAR ( [ ALL | DISTINCT ] expression )  
     OVER ( [ partition_by_clause ] order_by_clause )
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse  
  
-- Aggregate Function Syntax  
VAR ( [ ALL | DISTINCT ] expression )  
  
-- Analytic Function Syntax  
VAR (expression) OVER ( [ partition_by_clause ] order_by_clause )
```

## Arguments

### ALL

Applies the function to all values. ALL is the default.

### DISTINCT

Specifies that each unique value is considered.

### expression

Is an [expression](#) of the exact numeric or approximate numeric data type category, except for the **bit** data type. Aggregate functions and subqueries are not permitted.

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

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. *order\_by\_clause* is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

### float

## Remarks

If VAR is used on all items in a SELECT statement, each value in the result set is included in the calculation. VAR can be used with numeric columns only. Null values are ignored.

VAR is a deterministic function when used without the OVER and ORDER BY clauses. It is nondeterministic when specified with the OVER and ORDER BY clauses. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A: Using VAR

The following example returns the variance for all bonus values in the `SalesPerson` table in the AdventureWorks2012 database.

```
SELECT VAR(Bonus)
FROM Sales.SalesPerson;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### B: Using VAR

The following example returns the statistical variance of the sales quota values in the table `dbo.FactSalesQuota`. The first column contains the variance of all distinct values and the second column contains the variance of all values including any duplicates values.

```
-- Uses AdventureWorks

SELECT VAR(DISTINCT SalesAmountQuota)AS Distinct_Values, VAR(SalesAmountQuota) AS All_Values
FROM dbo.FactSalesQuota;
```

Here is the result set.

Distinct_Values	All_Values
-----	-----
159180469909.18	158762853821.10

### C. Using VAR with OVER

The following example returns the statistical variance of the sales quota values for each quarter in a calendar year. Notice that the ORDER BY in the OVER clause orders the statistical variance and the ORDER BY of the SELECT statement orders the result set.

```
-- Uses AdventureWorks

SELECT CalendarYear AS Year, CalendarQuarter AS Quarter, SalesAmountQuota AS SalesQuota,
       VAR(SalesAmountQuota) OVER (ORDER BY CalendarYear, CalendarQuarter) AS Variance
  FROM dbo.FactSalesQuota
 WHERE EmployeeKey = 272 AND CalendarYear = 2002
 ORDER BY CalendarQuarter;
```

Here is the result set.

Year	Quarter	SalesQuota	Variance
-----	-----	-----	-----
2002	1	91000.0000	null
2002	2	140000.0000	1200500000.00

```
2002 3 70000.0000 1290333333.33
```

```
2002 4 154000.0000 1580250000.00
```

## See Also

[Aggregate Functions \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# VARP (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the statistical variance for the population for all values in the specified expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server and Azure SQL Database  
  
VARP ( [ ALL | DISTINCT ] expression )  
      OVER ( [ partition_by_clause ] order_by_clause ) nh
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse  
  
-- Aggregate Function Syntax  
VARP ( [ ALL | DISTINCT ] expression )  
  
-- Analytic Function Syntax  
VARP (expression) OVER ( [ partition_by_clause ] order_by_clause )
```

## Arguments

### ALL

Applies the function to all values. ALL is the default.

### DISTINCT

Specifies that each unique value is considered.

### expression

Is an [expression](#) of the exact numeric or approximate numeric data type category, except for the **bit** data type. Aggregate functions and subqueries are not permitted.

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

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. *order\_by\_clause* is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

### float

## Remarks

If VARP is used on all items in a SELECT statement, each value in the result set is included in the calculation. VARP can be used with numeric columns only. Null values are ignored.

VARP is a deterministic function when used without the OVER and ORDER BY clauses. It is nondeterministic when specified with the OVER and ORDER BY clauses. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A: Using VARP

The following example returns the variance for the population for all bonus values in the `SalesPerson` table in the AdventureWorks2012 database.

```
SELECT VARP(Bonus)
FROM Sales.SalesPerson;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### B: Using VARP

The following example returns the `VARP` of the sales quota values in the table `dbo.FactSalesQuota`. The first column contains the variance of all distinct values and the second column contains the variance of all values including any duplicates values.

```
-- Uses AdventureWorks

SELECT VARP(DISTINCT SalesAmountQuota)AS Distinct_Values, VARP(SalesAmountQuota) AS All_Values
FROM dbo.FactSalesQuota;
```

Here is the result set.

Distinct_Values	All_Values
-----	-----
158146830494.18	157788848582.94

### C. Using VARP with OVER

The following example returns the `VARP` of the sales quota values for each quarter in a calendar year. Notice that the ORDER BY in the OVER clause orders the statistical variance and the ORDER BY of the SELECT statement orders the result set.

```
-- Uses AdventureWorks

SELECT CalendarYear AS Year, CalendarQuarter AS Quarter, SalesAmountQuota AS SalesQuota,
       VARP(SalesAmountQuota) OVER (ORDER BY CalendarYear, CalendarQuarter) AS Variance
  FROM dbo.FactSalesQuota
 WHERE EmployeeKey = 272 AND CalendarYear = 2002
 ORDER BY CalendarQuarter;
```

Here is the result set.

Year	Quarter	SalesQuota	Variance
-----	-----	-----	-----
2002	1	91000.0000	0.00
2002	2	140000.0000	600250000.00

2002	3	70000.0000	860222222.22
------	---	------------	--------------

2002	4	154000.0000	1185187500.00
------	---	-------------	---------------

## See Also

[Aggregate Functions \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# Analytic Functions (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

SQL Server supports the following analytic functions. Analytic functions compute an aggregate value based on a group of rows. However, unlike aggregate functions, they can return multiple rows for each group. You can use analytic functions to compute moving averages, running totals, percentages or top-N results within a group.

<a href="#">CUME_DIST (Transact-SQL)</a>	<a href="#">LEAD (Transact-SQL)</a>
<a href="#">FIRST_VALUE (Transact-SQL)</a>	<a href="#">PERCENTILE_CONT (Transact-SQL)</a>
<a href="#">LAG (Transact-SQL)</a>	<a href="#">PERCENTILE_DISC (Transact-SQL)</a>
<a href="#">LAST_VALUE (Transact-SQL)</a>	<a href="#">PERCENT_RANK (Transact-SQL)</a>

## See also

[OVER Clause \(Transact-SQL\)](#)

# CUME\_DIST (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Calculates the cumulative distribution of a value in a group of values in SQL Server. That is, CUME\_DIST computes the relative position of a specified value in a group of values. For a row  $r$ , assuming ascending ordering, the CUME\_DIST of  $r$  is the number of rows with values lower than or equal to the value of  $r$ , divided by the number of rows evaluated in the partition or query result set. CUME\_DIST is similar to the PERCENT\_RANK function.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CUME_DIST( )  
OVER ( [ partition_by_clause ] order_by_clause )
```

## Arguments

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

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. *order\_by\_clause* is required. The <rows or range clause> of the OVER syntax cannot be specified in a CUME\_DIST function. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return types

**float(53)**

## Remarks

The range of values returned by CUME\_DIST is greater than 0 and less than or equal to 1. Tie values always evaluate to the same cumulative distribution value. NULL values are included by default and are treated as the lowest possible values.

CUME\_DIST is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

The following example uses the CUME\_DIST function to compute the salary percentile for each employee within a given department. The value returned by the CUME\_DIST function represents the percent of employees that have a salary less than or equal to the current employee in the same department. The PERCENT\_RANK function computes the percent rank of the employee's salary within a department. The PARTITION BY clause is specified to partition the rows in the result set by department. The ORDER BY clause in the OVER clause logically orders the rows in each partition. The ORDER BY clause in the SELECT statement determines the display order of the result set.

```

USE AdventureWorks2012;
GO
SELECT Department, LastName, Rate,
       CUME_DIST () OVER (PARTITION BY Department ORDER BY Rate) AS CumeDist,
       PERCENT_RANK() OVER (PARTITION BY Department ORDER BY Rate ) AS PctRank
FROM HumanResources.vEmployeeDepartmentHistory AS edh
     INNER JOIN HumanResources.EmployeePayHistory AS e
       ON e.BusinessEntityID = edh.BusinessEntityID
WHERE Department IN (N'Information Services',N'Document Control')
ORDER BY Department, Rate DESC;

```

Here is the result set.

Department	LastName	Rate	CumeDist	PctRank
---				
Document Control	Arifin	17.7885	1	1
Document Control	Norred	16.8269	0.8	0.5
Document Control	Kharatishvili	16.8269	0.8	0.5
Document Control	Chai	10.25	0.4	0
Document Control	Berge	10.25	0.4	0
Information Services	Trenary	50.4808	1	1
Information Services	Conroy	39.6635	0.9	0.88888888888889
Information Services	Ajenstat	38.4615	0.8	0.6666666666666667
Information Services	Wilson	38.4615	0.8	0.6666666666666667
Information Services	Sharma	32.4519	0.6	0.4444444444444444
Information Services	Connelly	32.4519	0.6	0.4444444444444444
Information Services	Berg	27.4038	0.4	0
Information Services	Meyyappan	27.4038	0.4	0
Information Services	Bacon	27.4038	0.4	0
Information Services	Bueno	27.4038	0.4	0
(15 row(s) affected)				

## See also

[PERCENT\\_RANK \(Transact-SQL\)](#)

# FIRST\_VALUE (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse

Parallel Data Warehouse

Returns the first value in an ordered set of values in SQL Server 2017.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
FIRST_VALUE ( [scalar_expression] )
OVER ( [ partition_by_clause ] order_by_clause [ rows_range_clause ] )
```

## Arguments

*scalar\_expression*

Is the value to be returned. *scalar\_expression* can be a column, subquery, or other arbitrary expression that results in a single value. Other analytic functions are not permitted.

*OVER ( [ partition\_by\_clause ] order\_by\_clause [ rows\_range\_clause ] )*

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. *order\_by\_clause* is required. *rows\_range\_clause* further limits the rows within the partition by specifying start and end points. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

Is the same type as *scalar\_expression*.

## General Remarks

FIRST\_VALUE is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Using FIRST\_VALUE over a query result set

The following example uses FIRST\_VALUE to return the name of the product that is the least expensive in a given product category.

```
USE AdventureWorks2012;
GO
SELECT Name, ListPrice,
       FIRST_VALUE(Name) OVER (ORDER BY ListPrice ASC) AS LeastExpensive
  FROM Production.Product
 WHERE ProductSubcategoryID = 37;
```

Here is the result set.

Name	ListPrice	LeastExpensive
Patch Kit/8 Patches	2.29	Patch Kit/8 Patches
Road Tire Tube	3.99	Patch Kit/8 Patches
Touring Tire Tube	4.99	Patch Kit/8 Patches
Mountain Tire Tube	4.99	Patch Kit/8 Patches
LL Road Tire	21.49	Patch Kit/8 Patches
ML Road Tire	24.99	Patch Kit/8 Patches
LL Mountain Tire	24.99	Patch Kit/8 Patches
Touring Tire	28.99	Patch Kit/8 Patches
ML Mountain Tire	29.99	Patch Kit/8 Patches
HL Road Tire	32.60	Patch Kit/8 Patches
HL Mountain Tire	35.00	Patch Kit/8 Patches

## B. Using FIRST\_VALUE over partitions

The following example uses FIRST\_VALUE to return the employee with the fewest number of vacation hours compared to other employees with the same job title. The PARTITION BY clause partitions the employees by job title and the FIRST\_VALUE function is applied to each partition independently. The ORDER BY clause specified in the OVER clause determines the logical order in which the FIRST\_VALUE function is applied to the rows in each partition. The ROWS UNBOUNDED PRECEDING clause specifies the starting point of the window is the first row of each partition.

```
USE AdventureWorks2012;
GO
SELECT JobTitle, LastName, VacationHours,
       FIRST_VALUE(LastName) OVER (PARTITION BY JobTitle
                                     ORDER BY VacationHours ASC
                                     ROWS UNBOUNDED PRECEDING
                               ) AS FewestVacationHours
FROM HumanResources.Employee AS e
INNER JOIN Person.Person AS p
      ON e.BusinessEntityID = p.BusinessEntityID
ORDER BY JobTitle;
```

Here is a partial result set.

JobTitle	LastName	VacationHours	FewestVacationHours
Accountant	Moreland	58	Moreland
Accountant	Seamans	59	Moreland
Accounts Manager	Liu	57	Liu
Accounts Payable Specialist	Tomic	63	Tomic
Accounts Payable Specialist	Sheperdigian	64	Tomic
Accounts Receivable Specialist	Poe	60	Poe
Accounts Receivable Specialist	Spoon	61	Poe
Accounts Receivable Specialist	Walton	62	Poe

## See Also

[OVER Clause \(Transact-SQL\)](#)

# LAG (Transact-SQL)

9/27/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2012) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse

✓ Parallel Data Warehouse

Accesses data from a previous row in the same result set without the use of a self-join in SQL Server 2017. LAG provides access to a row at a given physical offset that comes before the current row. Use this analytic function in a SELECT statement to compare values in the current row with values in a previous row.

 [Transact-SQL Syntax Conventions \(Transact-SQL\)](#)

## Syntax

```
LAG (scalar_expression [,offset] [,default])
     OVER ( [ partition_by_clause ] order_by_clause )
```

## Arguments

*scalar\_expression*

The value to be returned based on the specified offset. It is an expression of any type that returns a single (scalar) value. *scalar\_expression* cannot be an analytic function.

*offset*

The number of rows back from the current row from which to obtain a value. If not specified, the default is 1. *offset* can be a column, subquery, or other expression that evaluates to a positive integer or can be implicitly converted to **bigint**. *offset* cannot be a negative value or an analytic function.

*default*

The value to return when *scalar\_expression* at *offset* is NULL. If a default value is not specified, NULL is returned. *default* can be a column, subquery, or other expression, but it cannot be an analytic function. *default* must be type-compatible with *scalar\_expression*.

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

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the order of the data before the function is applied. If *partition\_by\_clause* is specified, it determines the order of the data in the partition. The *order\_by\_clause* is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

The data type of the specified *scalar\_expression*. NULL is returned if *scalar\_expression* is nullable or *default* is set to NULL.

## General Remarks

LAG is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

## A. Compare values between years

The following example uses the LAG function to return the difference in sales quotas for a specific employee over previous years. Notice that because there is no lag value available for the first row, the default of zero (0) is returned.

```
USE AdventureWorks2012;
GO
SELECT BusinessEntityID, YEAR(QuotaDate) AS SalesYear, SalesQuota AS CurrentQuota,
       LAG(SalesQuota, 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS PreviousQuota
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275 and YEAR(QuotaDate) IN ('2005','2006');
```

Here is the result set.

BusinessEntityID	SalesYear	CurrentQuota	PreviousQuota
275	2005	367000.00	0.00
275	2005	556000.00	367000.00
275	2006	502000.00	556000.00
275	2006	550000.00	502000.00
275	2006	1429000.00	550000.00
275	2006	1324000.00	1429000.00

## B. Compare values within partitions

The following example uses the LAG function to compare year-to-date sales between employees. The PARTITION BY clause is specified to divide the rows in the result set by sales territory. The LAG function is applied to each partition separately and computation restarts for each partition. The ORDER BY clause in the OVER clause orders the rows in each partition. The ORDER BY clause in the SELECT statement sorts the rows in the whole result set. Notice that because there is no lag value available for the first row of each partition, the default of zero (0) is returned.

```
USE AdventureWorks2012;
GO
SELECT TerritoryName, BusinessEntityID, SalesYTD,
       LAG (SalesYTD, 1, 0) OVER (PARTITION BY TerritoryName ORDER BY SalesYTD DESC) AS PrevRepSales
FROM Sales.vSalesPerson
WHERE TerritoryName IN (N'Northwest', N'Canada')
ORDER BY TerritoryName;
```

Here is the result set.

TerritoryName	BusinessEntityID	SalesYTD	PrevRepSales
Canada	282	2604540.7172	0.00
Canada	278	1453719.4653	2604540.7172
Northwest	284	1576562.1966	0.00
Northwest	283	1573012.9383	1576562.1966
Northwest	280	1352577.1325	1573012.9383

## C. Specifying arbitrary expressions

The following example demonstrates specifying a variety of arbitrary expressions in the LAG function syntax.

```

CREATE TABLE T (a int, b int, c int);
GO
INSERT INTO T VALUES (1, 1, -3), (2, 2, 4), (3, 1, NULL), (4, 3, 1), (5, 2, NULL), (6, 1, 5);

SELECT b, c,
       LAG(2*c, b*(SELECT MIN(b) FROM T), -c/2.0) OVER (ORDER BY a) AS i
FROM T;

```

Here is the result set.

b	c	i
1	-3	1
2	4	-2
1	NULL	8
3	1	-6
2	NULL	NULL
1	5	NULL

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D: Compare values between quarters

The following example demonstrates the LAG function. The query uses the LAG function to return the difference in sales quotas for a specific employee over previous calendar quarters. Notice that because there is no lag value available for the first row, the default of zero (0) is returned.

```

-- Uses AdventureWorks

SELECT CalendarYear, CalendarQuarter, SalesAmountQuota AS SalesQuota,
       LAG(SalesAmountQuota,1,0) OVER (ORDER BY CalendarYear, CalendarQuarter) AS PrevQuota,
       SalesAmountQuota - LAG(SalesAmountQuota,1,0) OVER (ORDER BY CalendarYear, CalendarQuarter) AS Diff
FROM dbo.FactSalesQuota
WHERE EmployeeKey = 272 AND CalendarYear IN (2001, 2002)
ORDER BY CalendarYear, CalendarQuarter;

```

Here is the result set.

Year	Quarter	SalesQuota	PrevQuota	Diff
2001	3	28000.0000	0.0000	28000.0000
2001	4	7000.0000	28000.0000	-21000.0000
2001	1	91000.0000	7000.0000	84000.0000
2002	2	140000.0000	91000.0000	49000.0000
2002	3	7000.0000	140000.0000	-70000.0000
2002	4	154000.0000	7000.0000	84000.0000

## See Also

[LEAD \(Transact-SQL\)](#)

# LAST\_VALUE (Transact-SQL)

3/24/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the last value in an ordered set of values in SQL Server 2017.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
LAST_VALUE ( [scalar_expression]
    OVER ( [ partition_by_clause ] order_by_clause rows_range_clause )
```

## Arguments

### *scalar\_expression*

Is the value to be returned. *scalar\_expression* can be a column, subquery, or other expression that results in a single value. Other analytic functions are not permitted.

### *OVER ( [ partition\_by\_clause ] order\_by\_clause [ rows\_range\_clause ] )*

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group.

*order\_by\_clause* determines the order of the data before the function is applied. The *order\_by\_clause* is required.

*rows\_range\_clause* further limits the rows within the partition by specifying start and end points. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

Is the same type as *scalar\_expression*.

## General Remarks

LAST\_VALUE is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Using LAST\_VALUE over partitions

The following example returns the hire date of the last employee in each department for the given salary (Rate). The PARTITION BY clause partitions the employees by department and the LAST\_VALUE function is applied to each partition independently. The ORDER BY clause specified in the OVER clause determines the logical order in which the LAST\_VALUE function is applied to the rows in each partition.

```

USE AdventureWorks2012;
GO
SELECT Department, LastName, Rate, HireDate,
       LAST_VALUE(HireDate) OVER (PARTITION BY Department ORDER BY Rate) AS LastValue
  FROM HumanResources.vEmployeeDepartmentHistory AS edh
 INNER JOIN HumanResources.EmployeePayHistory AS eph
        ON eph.BusinessEntityID = edh.BusinessEntityID
 INNER JOIN HumanResources.Employee AS e
        ON e.BusinessEntityID = edh.BusinessEntityID
 WHERE Department IN (N'Information Services',N'Document Control');

```

Here is the result set.

Department	LastName	Rate	HireDate	LastValue
Document Control	Chai	10.25	2003-02-23	2003-03-13
Document Control	Berge	10.25	2003-03-13	2003-03-13
Document Control	Norred	16.8269	2003-04-07	2003-01-17
Document Control	Kharatishvili	16.8269	2003-01-17	2003-01-17
Document Control	Arifin	17.7885	2003-02-05	2003-02-05
Information Services	Berg	27.4038	2003-03-20	2003-01-24
Information Services	Meyyappan	27.4038	2003-03-07	2003-01-24
Information Services	Bacon	27.4038	2003-02-12	2003-01-24
Information Services	Bueno	27.4038	2003-01-24	2003-01-24
Information Services	Sharma	32.4519	2003-01-05	2003-03-27
Information Services	Connelly	32.4519	2003-03-27	2003-03-27
Information Services	Ajenstat	38.4615	2003-02-18	2003-02-23
Information Services	Wilson	38.4615	2003-02-23	2003-02-23
Information Services	Conroy	39.6635	2003-03-08	2003-03-08
Information Services	Trenary	50.4808	2003-01-12	2003-01-12

## B. Using FIRST\_VALUE and LAST\_VALUE in a computed expression

The following example uses the FIRST\_VALUE and LAST\_VALUE functions in computed expressions to show the difference between the sales quota value for the current quarter and the first and last quarter of the year respectively for a given number of employees. The FIRST\_VALUE function returns the sales quota value for the first quarter of the year, and subtracts it from the sales quota value for the current quarter. It is returned in the derived column entitled DifferenceFromFirstQuarter. For the first quarter of a year, the value of the DifferenceFromFirstQuarter column is 0. The LAST\_VALUE function returns the sales quota value for the last quarter of the year, and subtracts it from the sales quota value for the current quarter. It is returned in the derived column entitled DifferenceFromLastQuarter. For the last quarter of a year, the value of the DifferenceFromLastQuarter column is 0.

The clause "RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING" is required in this example for the non-zero values to be returned in the DifferenceFromLastQuarter column, as shown below. The default range is "RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW". In this example, using that default range (or not including a range, resulting in the default being used) would result in zeroes being returned in the DifferenceFromLastQuarter column. For more information, see [OVER Clause \(Transact-SQL\)](#).

```

USE AdventureWorks2012;
SELECT BusinessEntityID, DATEPART(QUARTER,QuotaDate)AS Quarter, YEAR(QuotaDate) AS SalesYear,
SalesQuota AS QuotaThisQuarter,
SalesQuota - FIRST_VALUE(SalesQuota)
OVER (PARTITION BY BusinessEntityID, YEAR(QuotaDate)
ORDER BY DATEPART(QUARTER,QuotaDate) ) AS DifferenceFromFirstQuarter,
SalesQuota - LAST_VALUE(SalesQuota)
OVER (PARTITION BY BusinessEntityID, YEAR(QuotaDate)
ORDER BY DATEPART(QUARTER,QuotaDate)
RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING ) AS DifferenceFromLastQuarter
FROM Sales.SalesPersonQuotaHistory
WHERE YEAR(QuotaDate) > 2005
AND BusinessEntityID BETWEEN 274 AND 275
ORDER BY BusinessEntityID, SalesYear, Quarter;

```

Here is the result set.

BusinessEntityID	Quarter	SalesYear	QuotaThisQuarter	DifferenceFromFirstQuarter	DifferenceFromLastQuarter
<hr/>					
274	1	2006	91000.00	0.00	-63000.00
274	2	2006	140000.00	49000.00	-14000.00
274	3	2006	70000.00	-21000.00	-84000.00
274	4	2006	154000.00	63000.00	0.00
274	1	2007	107000.00	0.00	-9000.00
274	2	2007	58000.00	-49000.00	-58000.00
274	3	2007	263000.00	156000.00	147000.00
274	4	2007	116000.00	9000.00	0.00
274	1	2008	84000.00	0.00	-103000.00
274	2	2008	187000.00	103000.00	0.00
275	1	2006	502000.00	0.00	-822000.00
275	2	2006	550000.00	48000.00	-774000.00
275	3	2006	1429000.00	927000.00	105000.00
275	4	2006	1324000.00	822000.00	0.00
275	1	2007	729000.00	0.00	-489000.00
275	2	2007	1194000.00	465000.00	-24000.00
275	3	2007	1575000.00	846000.00	357000.00
275	4	2007	1218000.00	489000.00	0.00
275	1	2008	849000.00	0.00	-20000.00
275	2	2008	869000.00	20000.00	0.00

(20 row(s) affected)

# LEAD (Transact-SQL)

9/27/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2012) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Accesses data from a subsequent row in the same result set without the use of a self-join in SQL Server 2017.

LEAD provides access to a row at a given physical offset that follows the current row. Use this analytic function in a SELECT statement to compare values in the current row with values in a following row.

 [Transact-SQL Syntax Conventions \(Transact-SQL\)](#)

## Syntax

```
LEAD ( scalar_expression [ ,offset ] , [ default ] )
      OVER ( [ partition_by_clause ] order_by_clause )
```

## Arguments

### *scalar\_expression*

The value to be returned based on the specified offset. It is an expression of any type that returns a single (scalar) value. *scalar\_expression* cannot be an analytic function.

### *offset*

The number of rows forward from the current row from which to obtain a value. If not specified, the default is 1. *offset* can be a column, subquery, or other expression that evaluates to a positive integer or can be implicitly converted to **bigint**. *offset* cannot be a negative value or an analytic function.

### *default*

The value to return when *scalar\_expression* at *offset* is NULL. If a default value is not specified, NULL is returned. *default* can be a column, subquery, or other expression, but it cannot be an analytic function. *default* must be type-compatible with *scalar\_expression*.

### *OVER ( [ partition\_by\_clause ] order\_by\_clause )*

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the order of the data before the function is applied. When *partition\_by\_clause* is specified, it determines the order of the data in each partition. The *order\_by\_clause* is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

The data type of the specified *scalar\_expression*. NULL is returned if *scalar\_expression* is nullable or *default* is set to NULL.

LEAD is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Compare values between years

The query uses the LEAD function to return the difference in sales quotas for a specific employee over subsequent

years. Notice that because there is no lead value available for the last row, the default of zero (0) is returned.

```
USE AdventureWorks2012;
GO
SELECT BusinessEntityID, YEAR(QuotaDate) AS SalesYear, SalesQuota AS CurrentQuota,
       LEAD(SalesQuota, 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS NextQuota
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275 and YEAR(QuotaDate) IN ('2005','2006');
```

Here is the result set.

BusinessEntityID	SalesYear	CurrentQuota	NextQuota
275	2005	367000.00	556000.00
275	2005	556000.00	502000.00
275	2006	502000.00	550000.00
275	2006	550000.00	1429000.00
275	2006	1429000.00	1324000.00
275	2006	1324000.00	0.00

## B. Compare values within partitions

The following example uses the LEAD function to compare year-to-date sales between employees. The PARTITION BY clause is specified to partition the rows in the result set by sales territory. The LEAD function is applied to each partition separately and computation restarts for each partition. The ORDER BY clause specified in the OVER clause orders the rows in each partition before the function is applied. The ORDER BY clause in the SELECT statement orders the rows in the whole result set. Notice that because there is no lead value available for the last row of each partition, the default of zero (0) is returned.

```
USE AdventureWorks2012;
GO
SELECT TerritoryName, BusinessEntityID, SalesYTD,
       LEAD (SalesYTD, 1, 0) OVER (PARTITION BY TerritoryName ORDER BY SalesYTD DESC) AS NextRepSales
FROM Sales.vSalesPerson
WHERE TerritoryName IN (N'Northwest', N'Canada')
ORDER BY TerritoryName;
```

Here is the result set.

TerritoryName	BusinessEntityID	SalesYTD	NextRepSales
Canada	282	2604540.7172	1453719.4653
Canada	278	1453719.4653	0.00
Northwest	284	1576562.1966	1573012.9383
Northwest	283	1573012.9383	1352577.1325
Northwest	280	1352577.1325	0.00

## C. Specifying arbitrary expressions

The following example demonstrates specifying a variety of arbitrary expressions in the LEAD function syntax.

```

CREATE TABLE T (a int, b int, c int);
GO
INSERT INTO T VALUES (1, 1, -3), (2, 2, 4), (3, 1, NULL), (4, 3, 1), (5, 2, NULL), (6, 1, 5);

SELECT b, c,
       LEAD(2*c, b*(SELECT MIN(b) FROM T), -c/2.0) OVER (ORDER BY a) AS i
  FROM T;

```

Here is the result set.

b	c	i
1	-3	8
2	4	2
1	NULL	2
3	1	0
2	NULL	NULL
1	5	-2

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D: Compare values between quarters

The following example demonstrates the LEAD function. The query obtains the difference in sales quota values for a specified employee over subsequent calendar quarters. Notice that because there is no lead value available after the last row, the default of zero (0) is used.

```

-- Uses AdventureWorks

SELECT CalendarYear AS Year, CalendarQuarter AS Quarter, SalesAmountQuota AS SalesQuota,
       LEAD(SalesAmountQuota,1,0) OVER (ORDER BY CalendarYear, CalendarQuarter) AS NextQuota,
       SalesAmountQuota - LEAD(SalesAmountQuota,1,0) OVER (ORDER BY CalendarYear, CalendarQuarter) AS Diff
  FROM dbo.FactSalesQuota
 WHERE EmployeeKey = 272 AND CalendarYear IN (2001,2002)
 ORDER BY CalendarYear, CalendarQuarter;

```

Here is the result set.

Year	Quarter	SalesQuota	NextQuota	Diff
2001	3	28000.0000	7000.0000	21000.0000
2001	4	7000.0000	91000.0000	-84000.0000
2001	1	91000.0000	140000.0000	-49000.0000
2002	2	140000.0000	7000.0000	7000.0000
2002	3	7000.0000	154000.0000	84000.0000
2002	4	154000.0000	0.0000	154000.0000

## See Also

[LAG \(Transact-SQL\)](#)

# PERCENTILE\_CONT (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Calculates a percentile based on a continuous distribution of the column value in SQL Server. The result is interpolated and might not be equal to any of the specific values in the column.

[Transact-SQL Syntax Conventions \(Transact-SQL\)](#)

## Syntax

```
PERCENTILE_CONT ( numeric_literal )
    WITHIN GROUP ( ORDER BY order_by_expression [ ASC | DESC ] )
    OVER ( [ <partition_by_clause> ] )
```

## Arguments

### *numeric\_literal*

The percentile to compute. The value must range between 0.0 and 1.0.

### WITHIN GROUP ( ORDER BY *order\_by\_expression* [ ASC | DESC ] )

Specifies a list of numeric values to sort and compute the percentile over. Only one *order\_by\_expression* is allowed. The expression must evaluate to an exact numeric type (**int**, **bigint**, **smallint**, **tinyint**, **numeric**, **bit**, **decimal**, **smalldatetime**, **money**) or an approximate numeric type (**float**, **real**). Other data types are not allowed. The default sort order is ascending.

### OVER ( <partition\_by\_clause> )

Divides the result set produced by the FROM clause into partitions to which the percentile function is applied. For more information, see [OVER Clause \(Transact-SQL\)](#). The <ORDER BY clause> and <rows or range clause> of the OVER syntax cannot be specified in a PERCENTILE\_CONT function.

## Return Types

**float(53)**

## Compatibility Support

Under compatibility level 110 and higher, WITHIN GROUP is a reserved keyword. For more information, see [ALTER DATABASE Compatibility Level \(Transact-SQL\)](#).

## General Remarks

Any nulls in the data set are ignored.

PERCENTILE\_CONT is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Basic syntax example

The following example uses PERCENTILE\_CONT and PERCENTILE\_DISC to find the median employee salary in each department. Note that these functions may not return the same value. This is because PERCENTILE\_CONT interpolates the appropriate value, whether or not it exists in the data set, while PERCENTILE\_DISC always returns an actual value from the set.

```
USE AdventureWorks2012;

SELECT DISTINCT Name AS DepartmentName
    ,PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY ph.Rate)
        OVER (PARTITION BY Name) AS MedianCont
    ,PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY ph.Rate)
        OVER (PARTITION BY Name) AS MedianDisc
FROM HumanResources.Department AS d
INNER JOIN HumanResources.EmployeeDepartmentHistory AS dh
    ON dh.DepartmentID = d.DepartmentID
INNER JOIN HumanResources.EmployeePayHistory AS ph
    ON ph.BusinessEntityID = dh.BusinessEntityID
WHERE dh.EndDate IS NULL;
```

Here is a partial result set.

DepartmentName	MedianCont	MedianDisc
----------------	------------	------------

-----	-----	-----
-------	-------	-------

Document Control	16.8269	16.8269
------------------	---------	---------

Engineering	34.375	32.6923
-------------	--------	---------

Executive	54.32695	48.5577
-----------	----------	---------

Human Resources	17.427850	16.5865
-----------------	-----------	---------

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### B. Basic syntax example

The following example uses PERCENTILE\_CONT and PERCENTILE\_DISC to find the median employee salary in each department. Note that these functions may not return the same value. This is because PERCENTILE\_CONT interpolates the appropriate value, whether or not it exists in the data set, while PERCENTILE\_DISC always returns an actual value from the set.

```
-- Uses AdventureWorks

SELECT DISTINCT DepartmentName
    ,PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY BaseRate)
        OVER (PARTITION BY DepartmentName) AS MedianCont
    ,PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY BaseRate)
        OVER (PARTITION BY DepartmentName) AS MedianDisc
FROM dbo.DimEmployee;
```

Here is a partial result set.

DepartmentName	MedianCont	MedianDisc
----------------	------------	------------

-----	-----	-----
-------	-------	-------

Document Control	16.826900	16.8269
------------------	-----------	---------

Engineering	34.375000	32.6923
-------------	-----------	---------

Human Resources 17.427850 16.5865

Shipping and Receiving 9.250000 9.0000

## See Also

[PERCENTILE\\_DISC \(Transact-SQL\)](#)

# PERCENTILE\_DISC (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2012) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Computes a specific percentile for sorted values in an entire rowset or within distinct partitions of a rowset in SQL Server. For a given percentile value  $P$ , PERCENTILE\_DISC sorts the values of the expression in the ORDER BY clause and returns the value with the smallest CUME\_DIST value (with respect to the same sort specification) that is greater than or equal to  $P$ . For example, PERCENTILE\_DISC (0.5) will compute the 50th percentile (that is, the median) of an expression. PERCENTILE\_DISC calculates the percentile based on a discrete distribution of the column values; the result is equal to a specific value in the column.

 [Transact-SQL Syntax Conventions \(Transact-SQL\)](#)

## Syntax

```
PERCENTILE_DISC ( numeric_literal ) WITHIN GROUP ( ORDER BY order_by_expression [ ASC | DESC ] )
OVER ( [ <partition_by_clause> ] )
```

## Arguments

*literal*

The percentile to compute. The value must range between 0.0 and 1.0.

**WITHIN GROUP ( ORDER BY *order\_by\_expression* [ ASC | DESC ] )**

Specifies a list of values to sort and compute the percentile over. Only one *order\_by\_expression* is allowed. The default sort order is ascending. The list of values can be of any of the data types that are valid for the sort operation.

**OVER ( <partition\_by\_clause> )**

Divides the result set produced by the FROM clause into partitions to which the percentile function is applied. For more information, see [OVER Clause \(Transact-SQL\)](#). The <ORDER BY clause> and <rows or range clause> cannot be specified in a PERCENTILE\_DISC function.

## Return Types

The return type is determined by the *order\_by\_expression* type.

## Compatibility Support

Under compatibility level 110 and higher, WITHIN GROUP is a reserved keyword. For more information, see [ALTER DATABASE Compatibility Level \(Transact-SQL\)](#).

## General Remarks

Any nulls in the data set are ignored.

PERCENTILE\_DISC is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

# Examples

## A. Basic syntax example

The following example uses PERCENTILE\_CONT and PERCENTILE\_DISC to find the median employee salary in each department. Note that these functions may not return the same value. This is because PERCENTILE\_CONT interpolates the appropriate value, whether or not it exists in the data set, while PERCENTILE\_DISC always returns an actual value from the set.

```
USE AdventureWorks2012;

SELECT DISTINCT Name AS DepartmentName
    ,PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY ph.Rate)
        OVER (PARTITION BY Name) AS MedianCont
    ,PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY ph.Rate)
        OVER (PARTITION BY Name) AS MedianDisc
FROM HumanResources.Department AS d
INNER JOIN HumanResources.EmployeeDepartmentHistory AS dh
    ON dh.DepartmentID = d.DepartmentID
INNER JOIN HumanResources.EmployeePayHistory AS ph
    ON ph.BusinessEntityID = dh.BusinessEntityID
WHERE dh.EndDate IS NULL;
```

Here is a partial result set.

DepartmentName	MedianCont	MedianDisc
-----	-----	-----
Document Control	16.8269	16.8269
Engineering	34.375	32.6923
Executive	54.32695	48.5577
Human Resources	17.427850	16.5865

# Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

## B. Basic syntax example

The following example uses PERCENTILE\_CONT and PERCENTILE\_DISC to find the median employee salary in each department. Note that these functions may not return the same value. This is because PERCENTILE\_CONT interpolates the appropriate value, whether or not it exists in the data set, while PERCENTILE\_DISC always returns an actual value from the set.

```
-- Uses AdventureWorks

SELECT DISTINCT DepartmentName
    ,PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY BaseRate)
        OVER (PARTITION BY DepartmentName) AS MedianCont
    ,PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY BaseRate)
        OVER (PARTITION BY DepartmentName) AS MedianDisc
FROM dbo.DimEmployee;
```

Here is a partial result set.

DepartmentName	MedianCont	MedianDisc
-----	-----	-----

Document Control 16.826900 16.8269

Engineering 34.375000 32.6923

Human Resources 17.427850 16.5865

Shipping and Receiving 9.250000 9.0000

## See Also

[PERCENTILE\\_CONT \(Transact-SQL\)](#)

# PERCENT\_RANK (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2012) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✗ Parallel Data Warehouse

Calculates the relative rank of a row within a group of rows in SQL Server 2017. Use PERCENT\_RANK to evaluate the relative standing of a value within a query result set or partition. PERCENT\_RANK is similar to the CUME\_DIST function.

## Syntax

```
PERCENT_RANK( )
    OVER ( [ partition_by_clause ] order_by_clause )
```

## Arguments

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

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the logical order in which the operation is performed. The *order\_by\_clause* is required. The <rows or range clause> of the OVER syntax cannot be specified in a PERCENT\_RANK function. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

**float(53)**

## General Remarks

The range of values returned by PERCENT\_RANK is greater than 0 and less than or equal to 1. The first row in any set has a PERCENT\_RANK of 0. NULL values are included by default and are treated as the lowest possible values.

PERCENT\_RANK is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

The following example uses the CUME\_DIST function to compute the salary percentile for each employee within a given department. The value returned by the CUME\_DIST function represents the percent of employees that have a salary less than or equal to the current employee in the same department. The PERCENT\_RANK function computes the rank of the employee's salary within a department as a percentage. The PARTITION BY clause is specified to partition the rows in the result set by department. The ORDER BY clause in the OVER clause orders the rows in each partition. The ORDER BY clause in the SELECT statement sorts the rows in the whole result set.

```

USE AdventureWorks2012;
GO
SELECT Department, LastName, Rate,
       CUME_DIST () OVER (PARTITION BY Department ORDER BY Rate) AS CumeDist,
       PERCENT_RANK() OVER (PARTITION BY Department ORDER BY Rate ) AS PctRank
FROM HumanResources.vEmployeeDepartmentHistory AS edh
INNER JOIN HumanResources.EmployeePayHistory AS e
ON e.BusinessEntityID = edh.BusinessEntityID
WHERE Department IN (N'Information Services',N'Document Control')
ORDER BY Department, Rate DESC;

```

Here is the result set.

Department	LastName	Rate	CumeDist	PctRank
---				
Document Control	Arifin	17.7885	1	1
Document Control	Norred	16.8269	0.8	0.5
Document Control	Kharatishvili	16.8269	0.8	0.5
Document Control	Chai	10.25	0.4	0
Document Control	Berge	10.25	0.4	0
Information Services	Trenary	50.4808	1	1
Information Services	Conroy	39.6635	0.9	0.88888888888889
Information Services	Ajenstat	38.4615	0.8	0.6666666666666667
Information Services	Wilson	38.4615	0.8	0.6666666666666667
Information Services	Sharma	32.4519	0.6	0.4444444444444444
Information Services	Connelly	32.4519	0.6	0.4444444444444444
Information Services	Berg	27.4038	0.4	0
Information Services	Meyyappan	27.4038	0.4	0
Information Services	Bacon	27.4038	0.4	0
Information Services	Bueno	27.4038	0.4	0
(15 row(s) affected)				

## See Also

[CUME\\_DIST \(Transact-SQL\)](#)

# Collation Functions - COLLATIONPROPERTY (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the property of a specified collation in SQL Server 2017.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
COLLATIONPROPERTY( collation_name , property )
```

## Arguments

*collation\_name*

Is the name of the collation. *collation\_name* is **nvarchar(128)**, and has no default.

*property*

Is the property of the collation. *property* is **varchar(128)**, and can be any one of the following values:

PROPERTY NAME	DESCRIPTION
<b>CodePage</b>	Non-Unicode code page of the collation.
<b>LCID</b>	Windows LCID of the collation.
<b>ComparisonStyle</b>	Windows comparison style of the collation. Returns 0 for all binary collations .
<b>Version</b>	<p>The version of the collation, derived from the version field of the collation ID. Returns 2, 1, or 0.</p> <p>Collations with "100" in the name) return 2.</p> <p>Collations with "90" in the name) return 1.</p> <p>All other collations return 0.</p>

## Return types

**sql\_variant**

## Examples

```
SELECT COLLATIONPROPERTY('Traditional_Spanish_CS_AS_WS', 'CodePage');
```

Here is the result set.

```
1252
```

Azure SQL Data Warehouse and Parallel Data Warehouse

```
SELECT COLLATIONPROPERTY('Traditional_Spanish_CS_AS_KS_WS', 'CodePage')
```

Here is the result set.

```
1252
```

## See also

[sys.fn\\_helpcollations \(Transact-SQL\)](#)

# Collation Functions - TERTIARY\_WEIGHTS (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a binary string of weights for each character in a non-Unicode string expression defined with an SQL tertiary collation.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
TERTIARY_WEIGHTS( non_Unicode_character_string_expression )
```

## Arguments

*non\_Unicode\_character\_string\_expression*

Is a string [expression](#) of type **char**, **varchar**, or **varchar(max)** defined on a tertiary SQL collation. For a list of these collations, see Remarks.

## Return types

TERTIARY\_WEIGHTS returns **varbinary** when *non\_Unicode\_character\_string\_expression* is **char** or **varchar**, and returns **varbinary(max)** when *non\_Unicode\_character\_string\_expression* is **varchar(max)**.

## Remarks

TERTIARY\_WEIGHTS returns NULL when *non\_Unicode\_character\_string\_expression* is not defined with an SQL tertiary collation. The following table shows the SQL tertiary collations.

SORT ORDER ID	SQL COLLATION
33	SQL_Latin1_General_Pref_CP437_CI_AS
34	SQL_Latin1_General_CP437_CI_AI
43	SQL_Latin1_General_Pref_CP850_CI_AS
44	SQL_Latin1_General_CP850_CI_AI
49	SQL_1xCompat_CP850_CI_AS
53	SQL_Latin1_General_Pref_CP1_CI_AS
54	SQL_Latin1_General_CP1_CI_AI

SORT ORDER ID	SQL COLLATION
56	SQL_AltDiction_Pref_CP850_CI_AS
57	SQL_AltDiction_CP850_CI_AI
58	SQL_Scandinavian_Pref_CP850_CI_AS
82	SQL_Latin1_General_CP1250_CI_AS
84	SQL_Czech_CP1250_CI_AS
86	SQL_Hungarian_CP1250_CI_AS
88	SQL_Polish_CP1250_CI_AS
90	SQL_Romanian_CP1250_CI_AS
92	SQL_Croatian_CP1250_CI_AS
94	SQL_Slovak_CP1250_CI_AS
96	SQL_Slovenian_CP1250_CI_AS
106	SQL_Latin1_General_CP1251_CI_AS
108	SQL_Ukrainian_CP1251_CI_AS
113	SQL_Latin1_General_CP1253_CS_AS
114	SQL_Latin1_General_CP1253_CI_AS
130	SQL_Latin1_General_CP1254_CI_AS
146	SQL_Latin1_General_CP1256_CI_AS
154	SQL_Latin1_General_CP1257_CI_AS
156	SQL_Estonian_CP1257_CI_AS
158	SQL_Latvian_CP1257_CI_AS
160	SQL_Lithuanian_CP1257_CI_AS
183	SQL_Danish_Pref_CP1_CI_AS
184	SQL_SwedishPhone_Pref_CP1_CI_AS
185	SQL_SwedishStd_Pref_CP1_CI_AS
186	SQL_Icelandic_Pref_CP1_CI_AS

TERTIARY\_WEIGHTS is intended for use in the definition of a computed column that is defined on the values of a

**char**, **varchar**, or **varchar(max)** column. Defining an index on both the computed column and the **char**, **varchar**, or **varchar(max)** column can improve performance when the **char**, **varchar**, or **varchar(max)** column is specified in the ORDER BY clause of a query.

## Examples

The following example creates a computed column in a table that applies the `TERTIARY_WEIGHTS` function to the values of a `char` column.

```
CREATE TABLE TertColTable
(Col1 char(15) COLLATE SQL_Latin1_General_Pref_CP437_CI_AS,
Col2 AS TERTIARY_WEIGHTS(Col1));
GO
```

## See also

[ORDER BY Clause \(Transact-SQL\)](#)

# Configuration Functions (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following scalar functions return information about current configuration option settings:

<a href="#">@@DATEFIRST</a>	<a href="#">@@OPTIONS</a>
<a href="#">@@DBTS</a>	<a href="#">@@REMSERVER</a>
<a href="#">@@LANGID</a>	<a href="#">@@SERVERNAME</a>
<a href="#">@@LANGUAGE</a>	<a href="#">@@SERVICENAME</a>
<a href="#">@@LOCK_TIMEOUT</a>	<a href="#">@@SPID</a>
<a href="#">@@MAX_CONNECTIONS</a>	<a href="#">@@TEXTSIZE</a>
<a href="#">@@MAX_PRECISION</a>	<a href="#">@@VERSION</a>
<a href="#">@@NESTLEVEL</a>	

All configuration functions are nondeterministic. This means these functions do not always return the same results every time they are called, even with the same set of input values. For more information about function determinism, see [Deterministic and Nondeterministic Functions](#).

## See also

[Functions \(Transact-SQL\)](#)

# @@DBTS (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the value of the current **timestamp** data type for the current database. This timestamp is guaranteed to be unique in the database.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@DBTS
```

## Return types

**varbinary**

## Remarks

@@DBTS returns the last-used timestamp value of the current database. A new timestamp value is generated when a row with a **timestamp** column is inserted or updated.

The @@DBTS function is not affected by changes in the transaction isolation levels.

## Examples

The following example returns the current **timestamp** from the AdventureWorks2012 database.

```
USE AdventureWorks2012;
GO
SELECT @@DBTS;
```

## See also

[Configuration Functions \(Transact-SQL\)](#)

[Cursor Concurrency \(ODBC\)](#)

[Data Types \(Transact-SQL\)](#)

[MIN\\_ACTIVE\\_ROWVERSION \(Transact-SQL\)](#)

# @@LANGID (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the local language identifier (ID) of the language that is currently being used.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@LANGID
```

## Return Types

**smallint**

## Remarks

To view information about language settings, including language ID numbers, run **sp\_helplanguage** without a parameter specified.

## Examples

The following example sets the language for the current session to `Italian`, and then uses `@@LANGID` to return the ID for Italian.

```
SET LANGUAGE 'Italian'  
SELECT @@LANGID AS 'Language ID'
```

Here is the result set.

```
Changed language setting to Italiano.  
Language ID  
-----  
6
```

## See Also

[Configuration Functions \(Transact-SQL\)](#)

[SET LANGUAGE \(Transact-SQL\)](#)

[sp\\_helplanguage \(Transact-SQL\)](#)

# @@LANGUAGE (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns the name of the language currently being used.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
@@LANGUAGE
```

## Return Types

**nvarchar**

## Remarks

To view information about language settings, including valid official language names, run **sp\_helplanguage** without a parameter specified.

## Examples

The following example returns the language for the current session.

```
SELECT @@LANGUAGE AS 'Language Name';
```

Here is the result set.

Language Name
-----
us_english

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the language for the current session.

```
SELECT @@LANGUAGE AS 'Language Name';
```

Here is the result set.

Language Name
-----
us_english

## See Also

[Configuration Functions \(Transact-SQL\)](#)

[SET LANGUAGE \(Transact-SQL\)](#)

[sp\\_HELPLANGUAGE \(Transact-SQL\)](#)

# @@LOCK\_TIMEOUT (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the current lock time-out setting in milliseconds for the current session.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@LOCK_TIMEOUT
```

## Return Types

**integer**

## Remarks

SET LOCK\_TIMEOUT allows an application to set the maximum time that a statement waits on a blocked resource. When a statement has waited longer than the LOCK\_TIMEOUT setting, the blocked statement is automatically canceled, and an error message is returned to the application.

@@LOCK\_TIMEOUT returns a value of -1 if SET LOCK\_TIMEOUT has not yet been run in the current session.

## Examples

This example shows the result set when a LOCK\_TIMEOUT value is not set.

```
SELECT @@LOCK_TIMEOUT AS [Lock Timeout];
GO
```

Here is the result set:

```
Lock Timeout
-----
-1
```

This example sets LOCK\_TIMEOUT to 1800 milliseconds and then calls @@LOCK\_TIMEOUT.

```
SET LOCK_TIMEOUT 1800;
SELECT @@LOCK_TIMEOUT AS [Lock Timeout];
GO
```

Here is the result set:

Lock Timeout

-----

1800

## See Also

[Configuration Functions \(Transact-SQL\)](#)

[SET LOCK\\_TIMEOUT \(Transact-SQL\)](#)

# @@MAX\_CONNECTIONS (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the maximum number of simultaneous user connections allowed on an instance of SQL Server. The number returned is not necessarily the number currently configured.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@MAX_CONNECTIONS
```

## Return Types

**integer**

## Remarks

The actual number of user connections allowed also depends on the version of SQL Server that is installed and the limitations of your applications and hardware.

To reconfigure SQL Server for fewer connections, use **sp\_configure**.

## Examples

The following example shows returning the maximum number of user connections on an instance of SQL Server. The example assumes that SQL Server has not been reconfigured for fewer user connections.

```
SELECT @@MAX_CONNECTIONS AS 'Max Connections';
```

Here is the result set.

Max Connections
-----
32767

## See Also

[sp\\_configure](#)  
[Configuration Functions](#)  
[Configure the user connections Server Configuration Option](#)

# @@MAX\_PRECISION (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the precision level used by **decimal** and **numeric** data types as currently set in the server.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@MAX_PRECISION
```

## Return Types

**tinyint**

## Remarks

By default, the maximum precision returns 38.

## Examples

```
SELECT @@MAX_PRECISION AS 'Max Precision'
```

## See Also

[Configuration Functions \(Transact-SQL\)](#)

[decimal and numeric \(Transact-SQL\)](#)

[Precision, Scale, and Length \(Transact-SQL\)](#)

# @@NESTLEVEL (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the nesting level of the current stored procedure execution (initially 0) on the local server.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@NESTLEVEL
```

## Return Types

**int**

## Remarks

Each time a stored procedure calls another stored procedure or executes managed code by referencing a common language runtime (CLR) routine, type, or aggregate, the nesting level is incremented. When the maximum of 32 is exceeded, the transaction is terminated.

When @@NESTLEVEL is executed within a Transact-SQL string, the value returned is 1 + the current nesting level. When @@NESTLEVEL is executed dynamically by using sp\_executesql the value returned is 2 + the current nesting level.

## Examples

### A. Using @@NESTLEVEL in a procedure

The following example creates two procedures: one that calls the other, and one that displays the @@NESTLEVEL setting of each.

```
USE AdventureWorks2012;
GO
IF OBJECT_ID ('N'usp_OuterProc', N'P')IS NOT NULL
    DROP PROCEDURE usp_OuterProc;
GO
IF OBJECT_ID ('N'usp_InnerProc', N'P')IS NOT NULL
    DROP PROCEDURE usp_InnerProc;
GO
CREATE PROCEDURE usp_InnerProc AS
    SELECT @@NESTLEVEL AS 'Inner Level';
GO
CREATE PROCEDURE usp_OuterProc AS
    SELECT @@NESTLEVEL AS 'Outer Level';
    EXEC usp_InnerProc;
GO
EXECUTE usp_OuterProc;
GO
```

Here is the result set.

Outer Level

-----

1

Inner Level

-----

2

## B. Calling @@NESTLEVEL

The following example shows the difference in values returned by `SELECT`, `EXEC`, and `sp_executesql` when each of them calls `@@NESTLEVEL`.

```
CREATE PROC usp_NestLevelValues AS
    SELECT @@NESTLEVEL AS 'Current Nest Level';
    EXEC ('SELECT @@NESTLEVEL AS OneGreater');
    EXEC sp_executesql N'SELECT @@NESTLEVEL as TwoGreater' ;
    GO
    EXEC usp_NestLevelValues;
    GO
```

Here is the result set.

Current Nest Level

-----

1

(1 row(s) affected)

OneGreater

-----

2

(1 row(s) affected)

TwoGreater

-----

3

(1 row(s) affected)

## See Also

[Configuration Functions \(Transact-SQL\)](#)

[Create a Stored Procedure](#)

[@@TRANCOUNT \(Transact-SQL\)](#)

# @@OPTIONS (Transact-SQL)

9/18/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns information about the current SET options.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@OPTIONS
```

## Return Types

**integer**

## Remarks

The options can come from use of the **SET** command or from the **sp\_configure user options** value. Session values configured with the **SET** command override the **sp\_configure** options. Many tools (such as Management Studio automatically configure set options. Each user has an @@OPTIONS function that represents the configuration.

You can change the language and query-processing options for a specific user session by using the **SET** statement. @@OPTIONS can only detect the options which are set to ON or OFF.

The @@OPTIONS function returns a bitmap of the options, converted to a base 10 (decimal) integer. The bit settings are stored in the locations described in a table in the topic [Configure the user options Server Configuration Option](#).

To decode the @@OPTIONS value, convert the integer returned by @@OPTIONS to binary, and then look up the values on the table at [Configure the user options Server Configuration Option](#). For example, if `SELECT @@OPTIONS;` returns the value `5496`, use the Windows programmer calculator (**calc.exe**) to convert decimal `5496` to binary. The result is `1010101111000`. The right most characters (binary 1, 2, and 4) are 0, indicating that the first three items in the table are off. Consulting the table, you see that those are **DISABLE\_DEF\_CNST\_CHK** and **IMPLICIT\_TRANSACTIONS**, and **CURSOR\_CLOSE\_ON\_COMMIT**. The next item (**ANSI\_WARNINGS** in the `1000` position) is on. Continue working left though the bit map, and down in the list of options. When the left-most options are 0, they are truncated by the type conversion. The bit map `1010101111000` is actually `001010101111000` to represent all 15 options.

## Examples

### A. Demonstration of how changes affect behavior

The following example demonstrates the difference in concatenation behavior with two different setting of the **CONCAT\_NULL\_YIELDS\_NULL** option.

```
SELECT @@OPTIONS AS OriginalOptionsValue;
SET CONCAT_NULL_YIELDS_NULL OFF;
SELECT 'abc' + NULL AS ResultWhen_OFF, @@OPTIONS AS OptionsValueWhen_OFF;

SET CONCAT_NULL_YIELDS_NULL ON;
SELECT 'abc' + NULL AS ResultWhen_ON, @@OPTIONS AS OptionsValueWhen_ON;
```

## B. Testing a client NOCOUNT setting

The following example sets `NOCOUNT` `ON` and then tests the value of `@@OPTIONS`. The `NOCOUNT` `ON` option prevents the message about the number of rows affected from being sent back to the requesting client for every statement in a session. The value of `@@OPTIONS` is set to `512` (0x0200). This represents the NOCOUNT option. This example tests whether the NOCOUNT option is enabled on the client. For example, it can help track performance differences on a client.

```
SET NOCOUNT ON
IF @@OPTIONS & 512 > 0
RAISERROR ('Current user has SET NOCOUNT turned on.', 1, 1)
```

## See Also

[Configuration Functions \(Transact-SQL\)](#)

[sp\\_configure \(Transact-SQL\)](#)

[Configure the user options Server Configuration Option](#)

# @@REMSERVER (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

## IMPORTANT

This feature will be removed in the next version of Microsoft SQL Server. Do not use this feature in new development work, and modify applications that currently use this feature as soon as possible. Use linked servers and linked server stored procedures instead.

Returns the name of the remote SQL Server database server as it appears in the login record.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@REMSERVER
```

## Return Types

**nvarchar(128)**

## Remarks

@@REMSERVER enables a stored procedure to check the name of the database server from which the procedure is run.

## Examples

The following example creates the procedure `usp_CheckServer` that returns the name of the remote server.

```
CREATE PROCEDURE usp_CheckServer
AS
SELECT @@REMSERVER;
```

The following stored procedure is created on the local server `SEATTLE1`. The user logs on to a remote server, `LONDON2`, and runs `usp_CheckServer`.

```
EXEC SEATTLE1...usp_CheckServer;
```

Here is the result set.

```
-----
LONDON2
```

## See Also

[Configuration Functions \(Transact-SQL\)](#)

[Remote Servers](#)

# @@SERVERNAME (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the name of the local server that is running SQL Server.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@SERVERNAME
```

## Return Types

**nvarchar**

## Remarks

SQL Server Setup sets the server name to the computer name during installation. To change the name of the server, use **sp\_addserver**, and then restart SQL Server.

With multiple instances of SQL Server installed, @@SERVERNAME returns the following local server name information if the local server name has not been changed since setup.

INSTANCE	SERVER INFORMATION
Default instance	'servername'
Named instance	'servername\instancename'
failover clustered instance - default instance	'virtualservername'
failover clustered instance - named instance	'virtualservername\instancename'

Although the @@SERVERNAME function and the SERVERNAME property of SERVERPROPERTY function may return strings with similar formats, the information can be different. The SERVERNAME property automatically reports changes in the network name of the computer.

In contrast, @@SERVERNAME does not report such changes. @@SERVERNAME reports changes made to the local server name using the **sp\_addserver** or **sp\_dropserver** stored procedure.

## Examples

The following example shows using `@@SERVERNAME`.

```
SELECT @@SERVERNAME AS 'Server Name'
```

Here is a sample result set.

Server Name
ACCTG

## See Also

[Configuration Functions \(Transact-SQL\)](#)

[SERVERPROPERTY \(Transact-SQL\)](#)

[sp\\_addserver \(Transact-SQL\)](#)

# @@SERVICENAME (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the name of the registry key under which SQL Server is running. @@SERVICENAME returns 'MSSQLSERVER' if the current instance is the default instance; this function returns the instance name if the current instance is a named instance.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@SERVICENAME
```

## Return Types

**nvarchar**

## Remarks

SQL Server runs as a service named MSSQLServer.

## Examples

The following example shows using `@@SERVICENAME`.

```
SELECT @@SERVICENAME AS 'Service Name';
```

Here is the result set.

Service Name
MSSQLSERVER

## See Also

[Configuration Functions \(Transact-SQL\)](#)

[Manage the Database Engine Services](#)

# @@SPID (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns the session ID of the current user process.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
@@SPID
```

## Return Types

**smallint**

## Remarks

@@SPID can be used to identify the current user process in the output of **sp\_who**.

## Examples

This example returns the session ID, login name, and user name for the current user process.

```
SELECT @@SPID AS 'ID', SYSTEM_USER AS 'Login Name', USER AS 'User Name';
```

Here is the result set.

ID	Login Name	User Name
54	SEATTLE\joanna	dbo

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

This example returns the SQL Data Warehouse session ID, the SQL Server Control node session ID, login name, and user name for the current user process.

```
SELECT SESSION_ID() AS ID, @@SPID AS 'Control ID', SYSTEM_USER AS 'Login Name', USER AS 'User Name';
```

## See Also

[Configuration Functions](#)

[sp\\_lock \(Transact-SQL\)](#)

[sp\\_who](#)

# @@TEXTSIZE (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the current value of the `TEXTSIZE` option.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@TEXTSIZE
```

## Return Types

**integer**

## Examples

The following example uses `SELECT` to display the `@@TEXTSIZE` value before and after it is changed with the `SET ``TEXTSIZE` statement.

```
-- Set the TEXTSIZE option to the default size of 4096 bytes.  
SET TEXTSIZE 0  
SELECT @@TEXTSIZE AS 'Text Size'  
SET TEXTSIZE 2048  
SELECT @@TEXTSIZE AS 'Text Size'
```

Here is the result set.

Text Size
-----
4096
Text Size
-----
2048

## See Also

[Configuration Functions \(Transact-SQL\)](#)

[SET TEXTSIZE \(Transact-SQL\)](#)

# @@Version - Transact SQL Configuration Functions

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns system and build information for the current installation of SQL Server.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
@@VERSION
```

## Return Types

**nvarchar**

## Remarks

The @@VERSION results are presented as one nvarchar string. You can use the [SERVERPROPERTY \(Transact-SQL\)](#) function to retrieve the individual property values.

For SQL Server, the following information is returned.

- SQL Server version
- Processor architecture
- SQL Server build date
- Copyright statement
- SQL Server edition
- Operating system version

For Azure SQL Database, the following information is returned.

- Edition- "Windows Azure SQL Database"
- Product level- "(CTP)" or "(RTM)"
- Product version
- Build date
- Copyright statement

## Examples

### A: Return the current version of SQL Server

The following example shows returning the version information for the current installation.

```
SELECT @@VERSION AS 'SQL Server Version';
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### B. Return the current version of SQL Data Warehouse

```
SELECT @@VERSION AS 'SQL Server PDW Version';
```

## See Also

[SERVERPROPERTY \(Transact-SQL\)](#)

# Conversion Functions (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following functions support data type casting and converting.

## In This Section

[CAST and CONVERT \(Transact-SQL\)](#)

[PARSE \(Transact-SQL\)](#)

[TRY\\_CAST \(Transact-SQL\)](#)

[TRY\\_CONVERT \(Transact-SQL\)](#)

[TRY\\_PARSE \(Transact-SQL\)](#)

## See also

[Functions](#)

[Data Types \(Transact-SQL\)](#)

# CAST and CONVERT (Transact-SQL)

9/8/2017 • 23 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Converts an expression of one data type to another.

For example, the following examples change the input datatype, into two other datatypes, with different levels of precision.

```
SELECT 9.5 AS Original, CAST(9.5 AS int) AS int,
       CAST(9.5 AS decimal(6,4)) AS decimal;
SELECT 9.5 AS Original, CONVERT(int, 9.5) AS int,
       CONVERT(decimal(6,4), 9.5) AS decimal;
```

Here is the result set.

ORIGINAL	INT	DECIMAL
9.5	9	9.5000

## TIP

Many [examples](#) are at the bottom of this topic.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for CAST:
CAST ( expression AS data_type [ ( length ) ] )

-- Syntax for CONVERT:
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

## Arguments

*expression*

Is any valid [expression](#).

*data\_type*

Is the target data type. This includes **xml**, **bigint**, and **sql\_variant**. Alias data types cannot be used.

*length*

Is an optional integer that specifies the length of the target data type. The default value is 30.

*style*

Is an integer expression that specifies how the CONVERT function is to translate *expression*. If *style* is NULL, NULL is returned. The range is determined by *data\_type*.

# Return types

Returns *expression* translated to *data\_type*.

## Date and Time Styles

When *expression* is a date or time data type, *style* can be one of the values shown in the following table. Other values are processed as 0. Beginning with SQL Server 2012, the only styles that are supported when converting from date and time types to **datetimeoffset** are 0 or 1. All other conversion styles return error 9809.

### NOTE

SQL Server supports the date format in Arabic style by using the Kuwaiti algorithm.

WITHOUT CENTURY (YY) <sup>(1)</sup>	WITH CENTURY (YYYY)	STANDARD	INPUT/OUTPUT <sup>(3)</sup>
-	<b>0 or 100</b> <sup>(1,2)</sup>	Default for datetime and smalldatetime	mon dd yyyy hh:miAM (or PM)
<b>1</b>	<b>101</b>	U.S.	1 = mm/dd/yy 101 = mm/dd/yyyy
<b>2</b>	<b>102</b>	ANSI	2 = yy.mm.dd 102 = yyyy.mm.dd
<b>3</b>	<b>103</b>	British/French	3 = dd/mm/yy 103 = dd/mm/yyyy
<b>4</b>	<b>104</b>	German	4 = dd.mm.yy 104 = dd.mm.yyyy
<b>5</b>	<b>105</b>	Italian	5 = dd-mm-yy 105 = dd-mm-yyyy
<b>6</b>	<b>106</b> <sup>(1)</sup>	-	6 = dd mon yy 106 = dd mon yyyy
<b>7</b>	<b>107</b> <sup>(1)</sup>	-	7 = Mon dd, yy 107 = Mon dd, yyyy
<b>8</b>	<b>108</b>	-	hh:mi:ss
-	<b>9 or 109</b> <sup>(1,2)</sup>	Default + milliseconds	mon dd yyyy hh:mi:ss:mmmAM (or PM)
<b>10</b>	<b>110</b>	USA	10 = mm-dd-yy 110 = mm-dd-yyyy
<b>11</b>	<b>111</b>	JAPAN	11 = yy/mm/dd 111 = yyyy/mm/dd
<b>12</b>	<b>112</b>	ISO	12 = yymmdd 112 = yyymmdd

WITHOUT CENTURY (YY) ( )	WITH CENTURY (YYYY)	STANDARD	INPUT/OUTPUT ( )
-	<b>13 or 113</b> <sup>(1,2)</sup>	Europe default + milliseconds	dd mon yyyy hh:mi:ss:mmm(24h)
<b>14</b>	<b>114</b>	-	hh:mi:ss:mmm(24h)
-	<b>20 or 120</b> <sup>(2)</sup>	ODBC canonical	yyyy-mm-dd hh:mi:ss(24h)
-	<b>21 or 121</b> <sup>(2)</sup>	ODBC canonical (with milliseconds) default for time, date, datetime2, and datetimeoffset	yyyy-mm-dd hh:mi:ss.mmm(24h)
-	<b>126</b> <sup>(4)</sup>	ISO8601	yyyy-mm- ddThh:mi:ss.mmm (no spaces)  Note: When the value for milliseconds (mmm) is 0, the millisecond value is not displayed. For example, the value '2012-11-07T18:26:20.000' is displayed as '2012-11-07T18:26:20'.
-	<b>127</b> <sup>(6, 7)</sup>	ISO8601 with time zone Z.	yyyy-mm- ddThh:mi:ss.mmmZ (no spaces)  Note: When the value for milliseconds (mmm) is 0, the milliseconds value is not displayed. For example, the value '2012-11-07T18:26:20.000' is displayed as '2012-11-07T18:26:20'.
-	<b>130</b> <sup>(1,2)</sup>	Hijri <sup>(5)</sup>	dd mon yyyy hh:mi:ss:mmmAM  In this style, mon represents a multi-token Hijri unicode representation of the full month's name. This value does not render correctly on a default US installation of SSMS.
-	<b>131</b> <sup>(2)</sup>	Hijri <sup>(5)</sup>	dd/mm/yyyy hh:mi:ss:mmmAM

<sup>1</sup> These style values return nondeterministic results. Includes all (yy) (without century) styles and a subset of (yyyy) (with century) styles.

<sup>2</sup> The default values (*style\* 0\** or **100, 9** or **109, 13** or **113, 20** or **120**, and **21** or **121**) always return the century (yyyy).

<sup>3</sup> Input when you convert to **datetime**; output when you convert to character data.

<sup>4</sup> Designed for XML use. For conversion from **datetime** or **smalldatetime** to character data, the output format is as described in the previous table.

<sup>5</sup> Hijri is a calendar system with several variations. SQL Server uses the Kuwaiti algorithm.

#### IMPORTANT

By default, SQL Server interprets two-digit years based on a cutoff year of 2049. That is, the two-digit year 49 is interpreted as 2049 and the two-digit year 50 is interpreted as 1950. Many client applications, such as those based on Automation objects, use a cutoff year of 2030. SQL Server provides the two digit year cutoff configuration option that changes the cutoff year used by SQL Server and allows for the consistent treatment of dates. We recommend specifying four-digit years.

<sup>6</sup> Only supported when casting from character data to **datetime** or **smalldatetime**. When character data that represents only date or only time components is cast to the **datetime** or **smalldatetime** data types, the unspecified time component is set to 00:00:00.000, and the unspecified date component is set to 1900-01-01.

<sup>7</sup>The optional time zone indicator, Z, is used to make it easier to map XML **datetime** values that have time zone information to SQL Server **datetime** values that have no time zone. Z is the indicator for time zone UTC-0. Other time zones are indicated with HH:MM offset in the + or - direction. For example:

2006-12-12T23:45:12-08:00 .

When you convert to character data from **smalldatetime**, the styles that include seconds or milliseconds show zeros in these positions. You can truncate unwanted date parts when you convert from **datetime** or **smalldatetime** values by using an appropriate **char** or **varchar** data type length.

When you convert to **datetimeoffset** from character data with a style that includes a time, a time zone offset is appended to the result.

## float and real styles

When *expression* is **float** or **real**, *style* can be one of the values shown in the following table. Other values are processed as 0.

VALUE	OUTPUT
<b>0</b> (default)	A maximum of 6 digits. Use in scientific notation, when appropriate.
<b>1</b>	Always 8 digits. Always use in scientific notation.
<b>2</b>	Always 16 digits. Always use in scientific notation.
<b>3</b>	Always 17 digits. Use for lossless conversion. With this style, every distinct float or real value is guaranteed to convert to a distinct character string. <b>Applies to:</b> Azure SQL Database, and starting in SQL Server 2016.
<b>126, 128, 129</b>	Included for legacy reasons and might be deprecated in a future release.

## money and smallmoney styles

When *expression* is **money** or **smallmoney**, *style* can be one of the values shown in the following table.

Other values are processed as 0.

VALUE	OUTPUT
<b>0</b> (default)	No commas every three digits to the left of the decimal point, and two digits to the right of the decimal point; for example, 4235.98.
<b>1</b>	Commas every three digits to the left of the decimal point, and two digits to the right of the decimal point; for example, 3,510.92.
<b>2</b>	No commas every three digits to the left of the decimal point, and four digits to the right of the decimal point; for example, 4235.9819.
<b>126</b>	Equivalent to style 2 when converting to char(n) or varchar(n)

## xml styles

When *expression* is **xml**, *style* can be one of the values shown in the following table. Other values are processed as 0.

VALUE	OUTPUT
<b>0</b> (default)	Use default parsing behavior that discards insignificant white space and does not allow for an internal DTD subset. <b>Note:</b> When you convert to the <b>xml</b> data type, SQL Server insignificant white space is handled differently than in XML 1.0. For more information, see <a href="#">Create Instances of XML Data</a> .
<b>1</b>	Preserve insignificant white space. This style setting sets the default <b>xml:space</b> handling to behave the same as if <b>xml:space="preserve"</b> has been specified instead.
<b>2</b>	Enable limited internal DTD subset processing.  If enabled, the server can use the following information that is provided in an internal DTD subset to perform nonvalidating parse operations. <ul style="list-style-type: none"><li>- Defaults for attributes are applied.</li><li>- Internal entity references are resolved and expanded.</li><li>- The DTD content model is checked for syntactical correctness.</li></ul> The parser ignores external DTD subsets. It also does not evaluate the XML declaration to see whether the <b>standalone</b> attribute is set <b>yes</b> or <b>no</b> , but instead parses the XML instance as if it is a stand-alone document.
<b>3</b>	Preserve insignificant white space and enable limited internal DTD subset processing.

## Binary styles

When *expression* is **binary(n)**, **varbinary(n)**, **char(n)**, or **varchar(n)**, *style* can be one of the values shown in

the following table. Style values that are not listed in the table return an error.

VALUE	OUTPUT
<b>0</b> (default)	Translates ASCII characters to binary bytes or binary bytes to ASCII characters. Each character or byte is converted 1:1. If the <i>data_type</i> is a binary type, the characters 0x are added to the left of the result.
<b>1, 2</b>	If the <i>data_type</i> is a binary type, the expression must be a character expression. The <i>expression</i> must be composed of an even number of hexadecimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, a, b, c, d, e, f). If the <i>style</i> is set to 1 the characters 0x must be the first two characters in the expression. If the expression contains an odd number of characters or if any of the characters are invalid an error is raised. If the length of the converted expression is greater than the length of the <i>data_type</i> the result is right truncated. Fixed length <i>data_types</i> that are larger than the converted result has zeros added to the right of the result. If the <i>data_type</i> is a character type, the expression must be a binary expression. Each binary character is converted into two hexadecimal characters. If the length of the converted expression is greater than the <i>data_type</i> length, it will be right truncated. If the <i>data_type</i> is a fix sized character type and the length of the converted result is less than its length of the <i>data_type</i> ; spaces are added to the right of the converted expression to maintain an even number of hexadecimal digits. The characters 0x will be added to the left of the converted result for <i>style</i> 1.

## Implicit conversions

Implicit conversions are those conversions that occur without specifying either the CAST or CONVERT function. Explicit conversions are those conversions that require the CAST or CONVERT function to be specified. The following illustration shows all explicit and implicit data type conversions that are allowed for SQL Server system-supplied data types. These include **xml**, **bigint**, and **sql\_variant**. There is no implicit conversion on assignment from the **sql\_variant** data type, but there is implicit conversion to **sql\_variant**.

### TIP

This chart is available as a downloadable PDF file at the [Microsoft Download Center](#).

From	To	binary	varbinary	char	varchar	nchar	nvarchar	datetime	smalldatetime	date	time	datetimeoffset	datetime2	decimal	numeric	float	real	bigint	int(INT4)	smallint(INT2)	tinyint(INT1)	money	smallmoney	bit	timestamp	uniqueidentifier	image	ntext	text	sql_variant	xml	CLR UDT	hierarchyid							
binary	binary	●	●	●	●	●	●	●	●	●	●	■	■	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
varbinary	binary	●		●	●	●	●	●	●	●	●	■	■	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
char	binary	■	■		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
varchar	binary	■	■	■		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
nchar	binary	■	■	■	■		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
nvarchar	binary	■	■	■	■	■		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
datetime	binary	■	■	■	■	■	■		●	●	●	●	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■									
smalldatetime	binary	■	■	■	■	■	■	■		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
date	binary	■	■	■	■	■	■	■	■		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
time	binary	■	■	■	■	■	■	■	■	■		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
datetimeoffset	binary	■	■	■	■	■	■	■	■	■	■		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
datetime2	binary	■	■	■	■	■	■	■	■	■	■	■		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●									
decimal	binary	●	●	●	●	●	●	●	●	●	●	●	●		◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆									
numeric	binary	●	●	●	●	●	●	●	●	●	●	●	●	●		◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆	◆◆								
float	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●							
real	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●						
bigint	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●						
int(INT4)	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●					
smallint(INT2)	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				
tinyint(INT1)	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				
money	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●					
smallmoney	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				
bit	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●			
timestamp	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
uniqueidentifier	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
image	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
ntext	binary	✗	✗	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
text	binary	✗	✗	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
sql_variant	binary	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
xml	binary	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CLR UDT	binary	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
hierarchyid	binary	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■

When you convert between **datetimeoffset** and the character types **char**, **varchar**, **nchar**, and **nvarchar** the converted time zone offset part should always be double digits for both HH and MM for example, -08:00.

#### NOTE

Because Unicode data always uses an even number of bytes, use caution when you convert **binary** or **varbinary** to or from Unicode supported data types. For example, the following conversion does not return a hexadecimal value of 41; it returns 4100:

```
SELECT CAST(CAST(0x41 AS nvarchar) AS varbinary)
```

## Large-value data types

Large-value data types exhibit the same implicit and explicit conversion behavior as their smaller counterparts, specifically the **varchar**, **nvarchar** and **varbinary** data types. However, you should consider the following guidelines:

- Conversion from **image** to **varbinary(max)** and vice-versa is an implicit conversion, and so are conversions between **text** and **varchar(max)**, and **ntext** and **nvarchar(max)**.
- Conversion from large-value data types, such as **varchar(max)**, to a smaller counterpart data type, such

as **varchar**, is an implicit conversion, but truncation occurs if the large value is too big for the specified length of the smaller data type.

- Conversion from **varchar**, **nvarchar**, or **varbinary** to their corresponding large-value data types is performed implicitly.
- Conversion from the **sql\_variant** data type to the large-value data types is an explicit conversion.
- Large-value data types cannot be converted to the **sql\_variant** data type.

For more information about how to convert from the **xml** data type, see [Create Instances of XML Data](#).

## xml data type

When you explicitly or implicitly cast the **xml** data type to a string or binary data type, the content of the **xml** data type is serialized based on a set of rules. For information about these rules, see [Define the Serialization of XML Data](#). For information about how to convert from other data types to the **xml** data type, see [Create Instances of XML Data](#).

## text and image data types

Automatic data type conversion is not supported for the **text** and **image** data types. You can explicitly convert **text** data to character data, and **image** data to **binary** or **varbinary**, but the maximum length is 8000 bytes. If you try an incorrect conversion such as trying to convert a character expression that includes letters to an **int**, SQL Server returns an error message.

## Output Collation

When the output of CAST or CONVERT is a character string, and the input is a character string, the output has the same collation and collation label as the input. If the input is not a character string, the output has the default collation of the database, and a collation label of coercible-default. For more information, see [Collation Precedence \(Transact-SQL\)](#).

To assign a different collation to the output, apply the COLLATE clause to the result expression of the CAST or CONVERT function. For example:

```
SELECT CAST('abc' AS varchar(5)) COLLATE French_CS_AS
```

## Truncating and rounding results

When you convert character or binary expressions (**char**, **nchar**, **nvarchar**, **varchar**, **binary**, or **varbinary**) to an expression of a different data type, data can be truncated, only partially displayed, or an error is returned because the result is too short to display. Conversions to **char**, **varchar**, **nchar**, **nvarchar**, **binary**, and **varbinary** are truncated, except for the conversions shown in the following table.

FROM DATA TYPE	TO DATA TYPE	RESULT
<b>int</b> , <b>smallint</b> , or <b>tinyint</b>	<b>char</b>	*
	<b>varchar</b>	*
	<b>nchar</b>	E
	<b>nvarchar</b>	E
<b>money</b> , <b>smallmoney</b> , <b>numeric</b> , <b>decimal</b> , <b>float</b> , or <b>real</b>	<b>char</b>	E

FROM DATA TYPE	TO DATA TYPE	RESULT
	<b>varchar</b>	E
	<b>nchar</b>	E
	<b>nvarchar</b>	E

\* = Result length too short to display. E = Error returned because result length is too short to display.

SQL Server guarantees that only roundtrip conversions, conversions that convert a data type from its original data type and back again, yield the same values from version to version. The following example shows such a roundtrip conversion:

```
DECLARE @myval decimal (5, 2);
SET @myval = 193.57;
SELECT CAST(CAST(@myval AS varbinary(20)) AS decimal(10,5));
-- Or, using CONVERT
SELECT CONVERT(decimal(10,5), CONVERT(varbinary(20), @myval));
```

#### NOTE

Do not try to construct **binary** values and then convert them to a data type of the numeric data type category. SQL Server does not guarantee that the result of a **decimal** or **numeric** data type conversion to **binary** will be the same between versions of SQL Server.

The following example shows a resulting expression that is too small to display.

```
USE AdventureWorks2012;
GO
SELECT p.FirstName, p.LastName, SUBSTRING(p.Title, 1, 25) AS Title,
       CAST(e.SickLeaveHours AS char(1)) AS [Sick Leave]
  FROM HumanResources.Employee e JOIN Person.Person p
    ON e.BusinessEntityID = p.BusinessEntityID
 WHERE NOT e.BusinessEntityID >5;
```

Here is the result set.

FirstName	LastName	Title	Sick Leave
Ken	Sanchez	NULL	*
Terri	Duffy	NULL	*
Roberto	Tamburello	NULL	*
Rob	Walters	NULL	*
Gail	Erickson	Ms.	*

(5 row(s) affected)

When you convert data types that differ in decimal places, sometimes the result value is truncated and at other times it is rounded. The following table shows the behavior.

FROM	TO	BEHAVIOR
<b>numeric</b>	<b>numeric</b>	Round

FROM	TO	BEHAVIOR
<b>numeric</b>	<b>int</b>	Truncate
<b>numeric</b>	<b>money</b>	Round
<b>money</b>	<b>int</b>	Round
<b>money</b>	<b>numeric</b>	Round
<b>float</b>	<b>int</b>	Truncate
<b>float</b>	<b>numeric</b>	Round  Conversion of <b>float</b> values that use scientific notation to <b>decimal</b> or <b>numeric</b> is restricted to values of precision 17 digits only. Any value with precision higher than 17 rounds to zero.
<b>float</b>	<b>datetime</b>	Round
<b>datetime</b>	<b>int</b>	Round

For example, the values 10.6496 and -10.6496 may be truncated or rounded during conversion to **int** or **numeric** types:

```
SELECT CAST(10.6496 AS int) as trunc1,
       CAST(-10.6496 AS int) as trunc2,
       CAST(10.6496 AS numeric) as round1,
       CAST(-10.6496 AS numeric) as round2;
```

Results of the query are shown in the following table:

TRUNC1	TRUNC2	ROUND1	ROUND2
10	-10	11	-11

When you convert data types in which the target data type has fewer decimal places than the source data type, the value is rounded. For example, the result of the following conversion is \$10.3497 :

```
SELECT CAST(10.3496847 AS money);
```

SQL Server returns an error message when nonnumeric **char**, **nchar**, **varchar**, or **nvarchar** data is converted to **int**, **float**, **numeric**, or **decimal**. SQL Server also returns an error when an empty string (" ") is converted to **numeric** or **decimal**.

## Certain datetime conversions are nondeterministic

The following table lists the styles for which the string-to-datetime conversion is nondeterministic.

All styles below 100 <sup>1</sup>	106
-----------------------------------	-----

107	109
113	130

<sup>1</sup> With the exception of styles 20 and 21

## Supplementary characters (surrogate pairs)

Beginning in SQL Server 2012, if you use supplementary character (SC) collations, a CAST operation from **nchar** or **nvarchar** to an **nchar** or **nvarchar** type of smaller length will not truncate inside a surrogate pair; it truncates before the supplementary character. For example, the following code fragment leaves `@x` holding just `'ab'`. There is not enough space to hold the supplementary character.

```
DECLARE @x NVARCHAR(10) = 'ab' + NCHAR(0x10000);
SELECT CAST (@x AS NVARCHAR(3));
```

When using SC collations the behavior of `CONVERT`, is analogous to that of `CAST`.

## Compatibility support

In earlier versions of SQL Server, the default style for CAST and CONVERT operations on **time** and **datetime2** data types is 121 except when either type is used in a computed column expression. For computed columns, the default style is 0. This behavior impacts computed columns when they are created, used in queries involving auto-parameterization, or used in constraint definitions.

Under compatibility level 110 and higher, the default style for CAST and CONVERT operations on **time** and **datetime2** data types is always 121. If your query relies on the old behavior, use a compatibility level less than 110, or explicitly specify the 0 style in the affected query.

Upgrading the database to compatibility level 110 and higher will not change user data that has been stored to disk. You must manually correct this data as appropriate. For example, if you used SELECT INTO to create a table from a source that contained a computed column expression described above, the data (using style 0) would be stored rather than the computed column definition itself. You would need to manually update this data to match style 121.

## Examples

### A. Using both CAST and CONVERT

Each example retrieves the name of the product for those products that have a `3` in the first digit of their list price and converts their `ListPrice` to `int`.

```
-- Use CAST
USE AdventureWorks2012;
GO
SELECT SUBSTRING(Name, 1, 30) AS ProductName, ListPrice
FROM Production.Product
WHERE CAST(ListPrice AS int) LIKE '3%';
GO

-- Use CONVERT.
USE AdventureWorks2012;
GO
SELECT SUBSTRING(Name, 1, 30) AS ProductName, ListPrice
FROM Production.Product
WHERE CONVERT(int, ListPrice) LIKE '3%';
GO
```

## B. Using CAST with arithmetic operators

The following example calculates a single column computation (`Computed`) by dividing the total year-to-date sales (`SalesYTD`) by the commission percentage (`CommissionPCT`). This result is converted to an `int` data type after being rounded to the nearest whole number.

```
USE AdventureWorks2012;
GO
SELECT CAST(ROUND(SalesYTD/CommissionPCT, 0) AS int) AS Computed
FROM Sales.SalesPerson
WHERE CommissionPCT != 0;
GO
```

Here is the result set.

Computed
-----
379753754
346698349
257144242
176493899
281101272
0
301872549
212623750
298948202
250784119
239246890
101664220
124511336
97688107
(14 row(s) affected)

## C. Using CAST to concatenate

The following example concatenates noncharacter, nonbinary expressions by using `CAST`.

```
USE AdventureWorks2012;
GO
SELECT 'The list price is ' + CAST(ListPrice AS varchar(12)) AS ListPrice
FROM Production.Product
WHERE ListPrice BETWEEN 350.00 AND 400.00;
GO
```

Here is the result set.

```
ListPrice  
-----  
The list price is 357.06  
The list price is 364.09  
(5 row(s) affected)
```

#### D. Using CAST to produce more readable text

The following example uses `CAST` in the select list to convert the `Name` column to a `char(10)` column.

```
USE AdventureWorks2012;  
GO  
SELECT DISTINCT CAST(p.Name AS char(10)) AS Name, s.UnitPrice  
FROM Sales.SalesOrderDetail AS s  
JOIN Production.Product AS p  
    ON s.ProductID = p.ProductID  
WHERE Name LIKE 'Long-Sleeve Logo Jersey, M';  
GO
```

Here is the result set.

Name	UnitPrice
-----	-----
Long-Sleev	31.2437
Long-Sleev	32.4935
Long-Sleev	49.99
(3 row(s) affected)	

#### E. Using CAST with the LIKE clause

The following example converts the `money` column `SalesYTD` to an `int` and then to a `char(20)` column so that it can be used with the `LIKE` clause.

```
USE AdventureWorks2012;  
GO  
SELECT p.FirstName, p.LastName, s.SalesYTD, s.BusinessEntityID  
FROM Person.Person AS p  
JOIN Sales.SalesPerson AS s  
    ON p.BusinessEntityID = s.BusinessEntityID  
WHERE CAST(CAST(s.SalesYTD AS int) AS char(20)) LIKE '2%';  
GO
```

Here is the result set.

FirstName	LastName	SalesYTD	SalesPersonID
-----	-----	-----	-----
Tsvi	Reiter	2811012.7151	279
Syed	Abbas	219088.8836	288
Rachel	Valdez	2241204.0424	289
(3 row(s) affected)			

#### F. Using CONVERT or CAST with typed XML

The following are several examples that show using `CONVERT` to convert to typed XML by using the [XML Data Type and Columns \(SQL Server\)](#).

This example converts a string with white space, text and markup into typed XML and removes all insignificant white space (boundary white space between nodes):

```
CONVERT(XML, '<root><child/></root>')
```

This example converts a similar string with white space, text and markup into typed XML and preserves insignificant white space (boundary white space between nodes):

```
CONVERT(XML, '<root>          <child/>          </root>', 1)
```

This example casts a string with white space, text, and markup into typed XML:

```
CAST('<Name><FName>Carol</FName><LName>Elliot</LName></Name>' AS XML)
```

For more examples, see [Create Instances of XML Data](#).

## G. Using CAST and CONVERT with datetime data

The following example displays the current date and time, uses `CAST` to change the current date and time to a character data type, and then uses `CONVERT` display the date and time in the `ISO 8901` format.

```
SELECT
    GETDATE() AS UnconvertedDateTime,
    CAST(GETDATE() AS nvarchar(30)) AS UsingCast,
    CONVERT(nvarchar(30), GETDATE(), 126) AS UsingConvertTo_ISO8601 ;
GO
```

Here is the result set.

UnconvertedDateTime	UsingCast	UsingConvertTo_ISO8601
2006-04-18 09:58:04.570	Apr 18 2006  9:58AM	2006-04-18T09:58:04.570

(1 row(s) affected)

The following example is approximately the opposite of the previous example. The example displays a date and time as character data, uses `CAST` to change the character data to the `datetime` data type, and then uses `CONVERT` to change the character data to the `datetime` data type.

```
SELECT
    '2006-04-25T15:50:59.997' AS UnconvertedText,
    CAST('2006-04-25T15:50:59.997' AS datetime) AS UsingCast,
    CONVERT(datetime, '2006-04-25T15:50:59.997', 126) AS UsingConvertFrom_ISO8601 ;
GO
```

Here is the result set.

UnconvertedText	UsingCast	UsingConvertFrom_ISO8601
2006-04-25T15:50:59.997	2006-04-25 15:50:59.997	2006-04-25 15:50:59.997

(1 row(s) affected)

## H. Using CONVERT with binary and character data

The following examples show the results of converting binary and character data by using different styles.

```
--Convert the binary value 0xE616d65 to a character value.  
SELECT CONVERT(char(8), 0xE616d65, 0) AS [Style 0, binary to character];
```

Here is the result set.

```
Style 0, binary to character  
-----  
Name  
(1 row(s) affected)
```

The following example shows how Style 1 can force the result to be truncated. The truncation is caused by including the characters 0x in the result.

```
SELECT CONVERT(char(8), 0xE616d65, 1) AS [Style 1, binary to character];
```

Here is the result set.

```
Style 1, binary to character  
-----  
0xE616D  
(1 row(s) affected)
```

The following example shows that Style 2 does not truncate the result because the characters 0x are not included in the result.

```
SELECT CONVERT(char(8), 0xE616d65, 2) AS [Style 2, binary to character];
```

Here is the result set.

```
Style 2, binary to character  
-----  
4E616D65  
(1 row(s) affected)
```

Convert the character value 'Name' to a binary value.

```
SELECT CONVERT(binary(8), 'Name', 0) AS [Style 0, character to binary];
```

Here is the result set.

```
Style 0, character to binary  
-----  
0xE616D6500000000  
(1 row(s) affected)
```

```
SELECT CONVERT(binary(4), '0xE616D65', 1) AS [Style 1, character to binary];
```

Here is the result set.

```
Style 1, character to binary
```

```
-----
```

```
0xE616D65
```

```
(1 row(s) affected)
```

```
SELECT CONVERT(binary(4), '4E616D65', 2) AS [Style 2, character to binary];
```

Here is the result set.

```
Style 2, character to binary
```

```
-----
```

```
0xE616D65
```

```
(1 row(s) affected)
```

## I. Converting date and time data types

The following example demonstrates the conversion of date, time, and datetime data types.

```
DECLARE @d1 date, @t1 time, @dt1 datetime;
SET @d1 = GETDATE();
SET @t1 = GETDATE();
SET @dt1 = GETDATE();
SET @d1 = GETDATE();
-- When converting date to datetime the minutes portion becomes zero.
SELECT @d1 AS [date], CAST (@d1 AS datetime) AS [date as datetime];
-- When converting time to datetime the date portion becomes zero
-- which converts to January 1, 1900.
SELECT @t1 AS [time], CAST (@t1 AS datetime) AS [time as datetime];
-- When converting datetime to date or time non-applicable portion is dropped.
SELECT @dt1 AS [datetime], CAST (@dt1 AS date) AS [datetime as date],
      CAST (@dt1 AS time) AS [datetime as time];
```

# Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

## J. Using CAST and CONVERT

This example retrieves the name of the product for those products that have a `3` in the first digit of their list price and converts their `ListPrice` to `int`. Uses AdventureWorksDW.

```
SELECT EnglishProductName AS ProductName, ListPrice
FROM dbo.DimProduct
WHERE CAST(ListPrice AS int) LIKE '3%';
```

This example shows the same query, using `CONVERT` instead of `CAST`. Uses AdventureWorksDW.

```
SELECT EnglishProductName AS ProductName, ListPrice
FROM dbo.DimProduct
WHERE CONVERT(int, ListPrice) LIKE '3%';
```

## K. Using CAST with arithmetic operators

The following example calculates a single column computation by dividing the product unit price (`UnitPrice`) by the discount percentage (`UnitPriceDiscountPct`). This result is converted to an `int` data type after being rounded to the nearest whole number. Uses AdventureWorksDW.

```

SELECT ProductKey, UnitPrice, UnitPriceDiscountPct,
       CAST(ROUND (UnitPrice*UnitPriceDiscountPct,0) AS int) AS DiscountPrice
  FROM dbo.FactResellerSales
 WHERE SalesOrderNumber = 'SO47355'
   AND UnitPriceDiscountPct > .02;

```

Here is the result set.

ProductKey	UnitPrice	UnitPriceDiscountPct	DiscountPrice
323	430.6445	0.05	22
213	18.5043	0.05	1
456	37.4950	0.10	4
456	37.4950	0.10	4
216	18.5043	0.05	1

## L. Using CAST to concatenate

The following example concatenates noncharacter expressions by using CAST. Uses AdventureWorksDW.

```

SELECT 'The list price is ' + CAST(ListPrice AS varchar(12)) AS ListPrice
  FROM dbo.DimProduct
 WHERE ListPrice BETWEEN 350.00 AND 400.00;

```

Here is the result set.

ListPrice
The list price is 357.06
The list price is 364.09

## M. Using CAST to produce more readable text

The following example uses CAST in the SELECT list to convert the `Name` column to a `char(10)` column. Uses AdventureWorksDW.

```

SELECT DISTINCT CAST(EnglishProductName AS char(10)) AS Name, ListPrice
  FROM dbo.DimProduct
 WHERE EnglishProductName LIKE 'Long-Sleeve Logo Jersey, M';

```

Here is the result set.

Name	UnitPrice
Long-Sleeve	31.2437
Long-Sleeve	32.4935
Long-Sleeve	49.99

## N. Using CAST with the LIKE clause

The following example converts the `money` column `ListPrice` to an `int` type and then to a `char(20)` type so that it can be used with the LIKE clause. Uses AdventureWorksDW.

```
SELECT EnglishProductName AS Name, ListPrice
FROM dbo.DimProduct
WHERE CAST(CAST(ListPrice AS int) AS char(20)) LIKE '2%';
```

## O. Using CAST and CONVERT with datetime data

The following example displays the current date and time, uses CAST to change the current date and time to a character data type, and then uses CONVERT display the date and time in the ISO 8601 format. Uses AdventureWorksDW.

```
SELECT TOP(1)
    SYSDATETIME() AS UnconvertedDateTime,
    CAST(SYSDATETIME() AS nvarchar(30)) AS UsingCast,
    CONVERT(nvarchar(30), SYSDATETIME(), 126) AS UsingConvertTo_ISO8601
FROM dbo.DimCustomer;
```

Here is the result set.

UnconvertedDateTime	UsingCast	UsingConvertTo_ISO8601
07/20/2010 1:44:31 PM	2010-07-20 13:44:31.5879025	2010-07-20T13:44:31.5879025

The following example is approximately the opposite of the previous example. The example displays a date and time as character data, uses CAST to change the character data to the **datetime** data type, and then uses CONVERT to change the character data to the **datetime** data type. Uses AdventureWorksDW.

```
SELECT TOP(1)
    '2010-07-25T13:50:38.544' AS UnconvertedText,
    CAST('2010-07-25T13:50:38.544' AS datetime) AS UsingCast,
    CONVERT(datetime, '2010-07-25T13:50:38.544', 126) AS UsingConvertFrom_ISO8601
FROM dbo.DimCustomer;
```

Here is the result set.

UnconvertedText	UsingCast	UsingConvertFrom_ISO8601
2010-07-25T13:50:38.544	07/25/2010 1:50:38 PM	07/25/2010 1:50:38 PM

## See also

[Data Type Conversion \(Database Engine\)](#)

[SELECT \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

[Write International Transact-SQL Statements](#)

# PARSE (Transact-SQL)

7/5/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the result of an expression, translated to the requested data type in SQL Server.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
PARSE ( string_value AS data_type [ USING culture ] )
```

## Arguments

*string\_value*

**nvarchar(4000)** value representing the formatted value to parse into the specified data type.

*string\_value* must be a valid representation of the requested data type, or PARSE raises an error.

*data\_type*

Literal value representing the data type requested for the result.

*culture*

Optional string that identifies the culture in which *string\_value* is formatted.

If the *culture* argument is not provided, then the language of the current session is used. This language is set either implicitly, or explicitly by using the SET LANGUAGE statement. *culture* accepts any culture supported by the .NET Framework; it is not limited to the languages explicitly supported by SQL Server. If the *culture* argument is not valid, PARSE raises an error.

## Return Types

Returns the result of the expression, translated to the requested data type.

## Remarks

Null values passed as arguments to PARSE are treated in two ways:

1. If a null constant is passed, an error is raised. A null value cannot be parsed into a different data type in a culturally aware manner.
2. If a parameter with a null value is passed at run time, then a null is returned, to avoid canceling the whole batch.

Use PARSE only for converting from string to date/time and number types. For general type conversions, continue to use CAST or CONVERT. Keep in mind that there is a certain performance overhead in parsing the string value.

PARSE relies on the presence of the .NET Framework Common Language Runtime (CLR).

This function will not be remoted since it depends on the presence of the CLR. Remoting a function that requires the CLR would cause an error on the remote server.

### More information about the `data_type` parameter

The values for the `data_type` parameter are restricted to the types shown in the following table, together with styles. The style information is provided to help determine what types of patterns are allowed. For more information on styles, see the .NET Framework documentation for the **System.Globalization.NumberStyles** and **DateTimeStyles** enumerations.

CATEGORY	TYPE	.NET FRAMEWORK TYPE	STYLES USED
Numeric	bigint	Int64	NumberStyles.Number
Numeric	int	Int32	NumberStyles.Number
Numeric	smallint	Int16	NumberStyles.Number
Numeric	tinyint	Byte	NumberStyles.Number
Numeric	decimal	Decimal	NumberStyles.Number
Numeric	numeric	Decimal	NumberStyles.Number
Numeric	float	Double	NumberStyles.Float
Numeric	real	Single	NumberStyles.Float
Numeric	smallmoney	Decimal	NumberStyles.Currency
Numeric	money	Decimal	NumberStyles.Currency
Date and Time	date	DateTime	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal
Date and Time	time	TimeSpan	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal
Date and Time	datetime	DateTime	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal
Date and Time	smalldatetime	DateTime	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal
Date and Time	datetime2	DateTime	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal

CATEGORY	TYPE	.NET FRAMEWORK TYPE	STYLES USED
Date and Time	datetimeoffset	DateTimeOffset	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal

## More information about the culture parameter

The following table shows the mappings from SQL Server languages to .NET Framework cultures.

FULL NAME	ALIAS	LCID	SPECIFIC CULTURE
us_english	English	1033	en-US
Deutsch	German	1031	de-DE
Français	French	1036	fr-FR
日本語	Japanese	1041	ja-JP
Dansk	Danish	1030	da-DK
Español	Spanish	3082	es-ES
Italiano	Italian	1040	it-IT
Nederlands	Dutch	1043	nl-NL
Norsk	Norwegian	2068	nn-NO
Português	Portuguese	2070	pt-PT
Suomi	Finnish	1035	fi
Svenska	Swedish	1053	sv-SE
čeština	Czech	1029	cs-CZ
magyar	Hungarian	1038	hu-HU
polski	Polish	1045	pl-PL
română	Romanian	1048	ro-RO
hrvatski	Croatian	1050	hr-HR
slovenčina	Slovak	1051	sk-SK
slovenski	Slovenian	1060	sl-SI
ελληνικά	Greek	1032	el-GR

FULL NAME	ALIAS	LCID	SPECIFIC CULTURE
български	Bulgarian	1026	bg-BG
русский	Russian	1049	Ru-RU
Türkçe	Turkish	1055	Tr-TR
British	British English	2057	en-GB
eesti	Estonian	1061	Et-EE
latviešu	Latvian	1062	lv-LV
lietuvių	Lithuanian	1063	lt-LT
Português (Brasil)	Brazilian	1046	pt-BR
繁體中文	Traditional Chinese	1028	zh-TW
한국어	Korean	1042	Ko-KR
简体中文	Simplified Chinese	2052	zh-CN
Arabic	Arabic	1025	ar-SA
ไทย	Thai	1054	Th-TH

## Examples

### A. PARSE into datetime2

```
SELECT PARSE('Monday, 13 December 2010' AS datetime2 USING 'en-US') AS Result;
```

Here is the result set.

```
Result
-----
2010-12-13 00:00:00.0000000

(1 row(s) affected)
```

### B. PARSE with currency symbol

```
SELECT PARSE('€345,98' AS money USING 'de-DE') AS Result;
```

Here is the result set.

```
Result
```

```
-----
```

```
345.98
```

```
(1 row(s) affected)
```

### C. PARSE with implicit setting of language

```
-- The English language is mapped to en-US specific culture  
SET LANGUAGE 'English';  
SELECT PARSE('12/16/2010' AS datetime2) AS Result;
```

Here is the result set.

```
Result
```

```
-----
```

```
2010-12-16 00:00:00.0000000
```

```
(1 row(s) affected)
```

# TRY\_CAST (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a value cast to the specified data type if the cast succeeds; otherwise, returns null.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
TRY_CAST ( expression AS data_type [ ( length ) ] )
```

## Arguments

*expression*

The value to be cast. Any valid expression.

*data\_type*

The data type into which to cast *expression*.

*length*

Optional integer that specifies the length of the target data type.

The range of acceptable values is determined by the value of *data\_type*.

## Return Types

Returns a value cast to the specified data type if the cast succeeds; otherwise, returns null.

## Remarks

**TRY\_CAST** takes the value passed to it and tries to convert it to the specified *data\_type*. If the cast succeeds, **TRY\_CAST** returns the value as the specified *data\_type*; if an error occurs, null is returned. However if you request a conversion that is explicitly not permitted, then **TRY\_CAST** fails with an error.

**TRY\_CAST** is not a new reserved keyword and is available in all compatibility levels. **TRY\_CAST** has the same semantics as **TRY\_CONVERT** when connecting to remote servers.

## Examples

### A. TRY\_CAST returns null

The following example demonstrates that TRY\_CAST returns null when the cast fails.

```
SELECT
    CASE WHEN TRY_CAST('test' AS float) IS NULL
    THEN 'Cast failed'
    ELSE 'Cast succeeded'
END AS Result;
GO
```

Here is the result set.

```
Result
-----
Cast failed

(1 row(s) affected)
```

The following example demonstrates that the expression must be in the expected format.

```
SET DATEFORMAT dmy;
SELECT TRY_CAST('12/31/2010' AS datetime2) AS Result;
GO
```

Here is the result set.

```
Result
-----
NULL

(1 row(s) affected)
```

## B. TRY\_CAST fails with an error

The following example demonstrates that TRY\_CAST returns an error when the cast is explicitly not permitted.

```
SELECT TRY_CAST(4 AS xml) AS Result;
GO
```

The result of this statement is an error, because an integer cannot be cast into an xml data type.

```
Explicit conversion from data type int to xml is not allowed.
```

## C. TRY\_CAST succeeds

This example demonstrates that the expression must be in the expected format.

```
SET DATEFORMAT mdy;
SELECT TRY_CAST('12/31/2010' AS datetime2) AS Result;
GO
```

Here is the result set.

**Result**

-----  
2010-12-31 00:00:00.0000000

(1 row(s) affected)

## See Also

[TRY\\_CONVERT \(Transact-SQL\)](#)

[CAST and CONVERT \(Transact-SQL\)](#)

# TRY\_CONVERT (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a value cast to the specified data type if the cast succeeds; otherwise, returns null.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
TRY_CONVERT ( data_type [ ( length ) ], expression [, style ] )
```

## Arguments

*data\_type* [ ( *length* ) ]

The data type into which to cast *expression*.

*expression*

The value to be cast.

*style*

Optional integer expression that specifies how the **TRY\_CONVERT** function is to translate *expression*.

*style* accepts the same values as the *style* parameter of the **CONVERT** function. For more information, see [CAST and CONVERT \(Transact-SQL\)](#).

The range of acceptable values is determined by the value of *data\_type*. If *style* is null, then **TRY\_CONVERT** returns null.

## Return Types

Returns a value cast to the specified data type if the cast succeeds; otherwise, returns null.

## Remarks

**TRY\_CONVERT** takes the value passed to it and tries to convert it to the specified *data\_type*. If the cast succeeds, **TRY\_CONVERT** returns the value as the specified *data\_type*; if an error occurs, null is returned. However if you request a conversion that is explicitly not permitted, then **TRY\_CONVERT** fails with an error.

**TRY\_CONVERT** is a reserved keyword in compatibility level 110 and higher.

This function is capable of being remoted to servers that have a version of SQL Server 2012 and above. It will not be remoted to servers that have a version below SQL Server 2012.

## Examples

### A. TRY\_CONVERT returns null

The following example demonstrates that TRY\_CONVERT returns null when the cast fails.

```
SELECT
    CASE WHEN TRY_CONVERT(float, 'test') IS NULL
    THEN 'Cast failed'
    ELSE 'Cast succeeded'
END AS Result;
GO
```

Here is the result set.

```
Result
-----
Cast failed

(1 row(s) affected)
```

The following example demonstrates that the expression must be in the expected format.

```
SET DATEFORMAT dmy;
SELECT TRY_CONVERT(datetime2, '12/31/2010') AS Result;
GO
```

Here is the result set.

```
Result
-----
NULL

(1 row(s) affected)
```

## B. TRY\_CONVERT fails with an error

The following example demonstrates that TRY\_CONVERT returns an error when the cast is explicitly not permitted.

```
SELECT TRY_CONVERT(xml, 4) AS Result;
GO
```

The result of this statement is an error, because an integer cannot be cast into an xml data type.

```
Explicit conversion from data type int to xml is not allowed.
```

## C. TRY\_CONVERT succeeds

This example demonstrates that the expression must be in the expected format.

```
SET DATEFORMAT mdy;
SELECT TRY_CONVERT(datetime2, '12/31/2010') AS Result;
GO
```

Here is the result set.

Result

-----  
2010-12-31 00:00:00.0000000

(1 row(s) affected)

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

# TRY\_PARSE (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the result of an expression, translated to the requested data type, or null if the cast fails in SQL Server. Use TRY\_PARSE only for converting from string to date/time and number types.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
TRY_PARSE ( string_value AS data_type [ USING culture ] )
```

## Arguments

*string\_value*

**nvarchar(4000)** value representing the formatted value to parse into the specified data type.

*string\_value* must be a valid representation of the requested data type, or TRY\_PARSE returns null.

*data\_type*

Literal representing the data type requested for the result.

*culture*

Optional string that identifies the culture in which *string\_value* is formatted.

If the *culture* argument is not provided, the language of the current session is used. This language is set either implicitly or explicitly by using the SET LANGUAGE statement. *culture* accepts any culture supported by the .NET Framework; it is not limited to the languages explicitly supported by SQL Server. If the *culture* argument is not valid, PARSE raises an error.

## Return Types

Returns the result of the expression, translated to the requested data type, or null if the cast fails.

## Remarks

Use TRY\_PARSE only for converting from string to date/time and number types. For general type conversions, continue to use CAST or CONVERT. Keep in mind that there is a certain performance overhead in parsing the string value.

TRY\_PARSE relies on the presence of the .NET Framework Common Language Runtime (CLR).

This function will not be remoted since it depends on the presence of the CLR. Remoting a function that requires the CLR would cause an error on the remote server.

### More information about the *data\_type* parameter

The values for the *data\_type* parameter are restricted to the types shown in the following table, together with styles. The style information is provided to help determine what types of patterns are allowed. For more

information on styles, see the .NET Framework documentation for the **System.Globalization.NumberStyles** and **DateTimeStyles** enumerations.

CATEGORY	TYPE	.NET TYPE	STYLES USED
Numeric	bigint	Int64	NumberStyles.Number
Numeric	int	Int32	NumberStyles.Number
Numeric	smallint	Int16	NumberStyles.Number
Numeric	tinyint	Byte	NumberStyles.Number
Numeric	decimal	Decimal	NumberStyles.Number
Numeric	numeric	Decimal	NumberStyles.Number
Numeric	float	Double	NumberStyles.Float
Numeric	real	Single	NumberStyles.Float
Numeric	smallmoney	Decimal	NumberStyles.Currency
Numeric	money	Decimal	NumberStyles.Currency
Date and Time	date	DateTime	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal
Date and Time	time	TimeSpan	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal
Date and Time	datetime	DateTime	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal
Date and Time	smalldatetime	DateTime	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal
Date and Time	datetime2	DateTime	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal
Date and Time	datetimeoffset	DateTimeOffset	DateTimeStyles.AllowWhiteSpaces   DateTimeStyles.AssumeUniversal

#### More information about the culture parameter

The following table shows the mappings from SQL Server languages to .NET Framework cultures.

FULL NAME	ALIAS	LCID	SPECIFIC CULTURE
us_english	English	1033	en-US
Deutsch	German	1031	de-DE
Français	French	1036	fr-FR
日本語	Japanese	1041	ja-JP
Dansk	Danish	1030	da-DK
Español	Spanish	3082	es-ES
Italiano	Italian	1040	it-IT
Nederlands	Dutch	1043	nl-NL
Norsk	Norwegian	2068	nn-NO
Português	Portuguese	2070	pt-PT
Suomi	Finnish	1035	fi
Svenska	Swedish	1053	sv-SE
čeština	Czech	1029	Cs-CZ
magyar	Hungarian	1038	Hu-HU
polski	Polish	1045	Pl-PL
română	Romanian	1048	Ro-RO
hrvatski	Croatian	1050	hr-HR
slovenčina	Slovak	1051	Sk-SK
slovenski	Slovenian	1060	Sl-SI
ελληνικά	Greek	1032	El-GR
български	Bulgarian	1026	bg-BG
русский	Russian	1049	Ru-RU
Türkçe	Turkish	1055	Tr-TR
British	British English	2057	en-GB
eesti	Estonian	1061	Et-EE

FULL NAME	ALIAS	LCID	SPECIFIC CULTURE
latviešu	Latvian	1062	lv-LV
lietuvių	Lithuanian	1063	lt-LT
Português (Brasil)	Brazilian	1046	pt-BR
繁體中文	Traditional Chinese	1028	zh-TW
한국어	Korean	1042	Ko-KR
简体中文	Simplified Chinese	2052	zh-CN
Arabic	Arabic	1025	ar-SA
ไทย	Thai	1054	Th-TH

## Examples

### A. Simple example of TRY\_PARSE

```
SELECT TRY_PARSE('Jabberwockie' AS datetime2 USING 'en-US') AS Result;
```

Here is the result set.

```
Result
-----
NULL

(1 row(s) affected)
```

### B. Detecting nulls with TRY\_PARSE

```
SELECT
    CASE WHEN TRY_PARSE('Aragorn' AS decimal USING 'sr-Latn-CS') IS NULL
        THEN 'True'
        ELSE 'False'
    END
AS Result;
```

Here is the result set.

```
Result
-----
True

(1 row(s) affected)
```

### C. Using IIF with TRY\_PARSE and implicit culture setting

```
SET LANGUAGE English;
SELECT IIF(TRY_PARSE('01/01/2011' AS datetime2) IS NULL, 'True', 'False') AS Result;
```

Here is the result set.

```
Result
-----
False

(1 row(s) affected)
```

## See Also

[PARSE \(Transact-SQL\)](#)

[Conversion Functions \(Transact-SQL\)](#)

[TRY\\_CONVERT \(Transact-SQL\)](#)

[CAST and CONVERT \(Transact-SQL\)](#)

# Cryptographic functions (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

The following functions support encryption, decryption, digital signing, and the validation of digital signatures.

## Symmetric encryption and decryption

<a href="#">ENCRYPTBYKEY</a>	<a href="#">DECRYPTBYKEY</a>
<a href="#">ENCRYPTBYPASSPHRASE</a>	<a href="#">DECRYPTBYPASSPHRASE</a>
<a href="#">KEY_ID</a>	<a href="#">KEY_GUID</a>
<a href="#">DECRYPTBYKEYAUTOASYMKEY</a>	<a href="#">KEY_NAME</a>
<a href="#">SYMKEYPROPERTY</a>	

## Asymmetric encryption and decryption

<a href="#">ENCRYPTBYASYMKEY</a>	<a href="#">DECRYPTBYASYMKEY</a>
<a href="#">ENCRYPTBYCert</a>	<a href="#">DECRYPTBYCERT</a>
<a href="#">ASYMKEYPROPERTY</a>	<a href="#">ASYMKEY_ID</a>

## Signing and signature verification

<a href="#">SIGNBYASYMKEY</a>	<a href="#">VERIFY SIGNED BY ASMKEY</a>
<a href="#">SIGNBYCERT</a>	<a href="#">VERIFY SIGNED BY CERT</a>
<a href="#">IS_OBJECTSIGNED</a>	

## Symmetric decryption with automatic key handling

<a href="#">DecryptByKeyAutoCert</a>	
--------------------------------------	--

## Encryption hashing

HASHBYTES

## Copying certificates

[CERTENCODED \(Transact-SQL\)](#)

[CERTPRIVATEKEY \(Transact-SQL\)](#)

## See also

[Functions](#)

[Encryption Hierarchy](#)

[Permissions Hierarchy \(Database Engine\)](#)

[CREATE CERTIFICATE \(Transact-SQL\)](#)

[CREATE SYMMETRIC KEY \(Transact-SQL\)](#)

[CREATE ASYMMETRIC KEY \(Transact-SQL\)](#)

[Security Catalog Views \(Transact-SQL\)](#)

# ASYMKEY\_ID (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the ID of an asymmetric key.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
ASYMKEY_ID ( 'Asym_Key_Name' )
```

## Arguments

*Asym\_Key\_Name*

Is the name of an asymmetric key in the database.

## Return types

**int**

## Permissions

Requires some permission on the asymmetric key and that the caller has not been denied VIEW permission on the asymmetric key.

## Examples

The following example returns the ID of asymmetric key `ABerglundKey11`.

```
SELECT ASYMKEY_ID('ABerglundKey11');
GO
```

## See also

[CREATE ASYMMETRIC KEY \(Transact-SQL\)](#)

[ALTER ASYMMETRIC KEY \(Transact-SQL\)](#)

[DROP ASYMMETRIC KEY \(Transact-SQL\)](#)

[SIGNBYASYMKEY \(Transact-SQL\)](#)

[VERIFY SIGNED BY ASYMMETRIC KEY \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

[sys.asymmetric\\_keys \(Transact-SQL\)](#)

[Security Catalog Views \(Transact-SQL\)](#)

[ASYMKEYPROPERTY \(Transact-SQL\)](#)

[KEY\\_ID \(Transact-SQL\)](#)

# ASYMKEYPROPERTY (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the properties of an asymmetric key.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
ASYMKEYPROPERTY (Key_ID , 'algorithm_desc' | 'string_sid' | 'sid')
```

## Arguments

### *Key\_ID*

Is the Key\_ID of an asymmetric key in the database. To find the Key\_ID when you only know the key name, use ASYMKEY\_ID. *Key\_ID* is data type **int**.

### 'algorithm\_desc'

Specifies that the output returns the algorithm description of the asymmetric key. Only available for asymmetric keys created from an EKM module.

### 'string\_sid'

Specifies that the output returns the SID of the asymmetric key in **nvarchar()** format.

### 'sid'

Specifies that the output returns the SID of the asymmetric key in binary format.

## Return types

### **sql\_variant**

## Permissions

Requires some permission on the asymmetric key and that the caller has not been denied VIEW permission on the asymmetric key.

## Examples

The following example returns the properties of the asymmetric key with Key\_ID 256.

```
SELECT
ASYMKEYPROPERTY(256, 'algorithm_desc') AS Algorithm,
ASYMKEYPROPERTY(256, 'string_sid') AS String_SID,
ASYMKEYPROPERTY(256, 'sid') AS SID ;
GO
```

## See also

[CREATE ASYMMETRIC KEY \(Transact-SQL\)](#)

[ALTER ASYMMETRIC KEY \(Transact-SQL\)](#)

[DROP ASYMMETRIC KEY \(Transact-SQL\)](#)

[SIGNBYASYMKEY \(Transact-SQL\)](#)

[VERIFYSIGNEDBYASYMKEY \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

[sys.asymmetric\\_keys \(Transact-SQL\)](#)

[Security Catalog Views \(Transact-SQL\)](#)

[ASYMKEY\\_ID \(Transact-SQL\)](#)

[SYMKEYPROPERTY \(Transact-SQL\)](#)

# CERTPROPERTY (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the value of a specified certificate property.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CertProperty ( Cert_ID , '<PropertyName>' )  
  
<PropertyName> ::=  
    Expiry_Date | Start_Date | Issuer_Name  
    | Cert_Serial_Number | Subject | SID | String_SID
```

## Arguments

*Cert\_ID*

Is the ID of the certificate. *Cert\_ID* is an int.

*Expiry\_Date*

Is the date of expiration of the certificate.

*Start\_Date*

Is the date when the certificate becomes valid.

*Issuer\_Name*

Is the issuer name of the certificate.

*Cert\_Serial\_Number*

Is the certificate serial number.

*Subject*

Is the subject of the certificate.

*SID*

Is the SID of the certificate. This is also the SID of any login or user mapped to this certificate.

*String\_SID*

Is the SID of the certificate as a character string. This is also the SID of any login or user mapped to the certificate.

## Return types

The property specification must be enclosed in single quotation marks.

The return type depends on the property that is specified in the function call. All return values are wrapped in the return type of **sql\_variant**.

- *Expiry\_Date* and *Start\_Date* return **datetime**.
- *Cert\_Serial\_Number*, *Issuer\_Name*, *Subject*, and *String\_SID* return **nvarchar**.
- *SID* returns **varbinary**.

## Remarks

Information about certificates is visible in the [sys.certificates](#) catalog view.

## Permissions

Requires some permission on the certificate and that the caller has not been denied VIEW DEFINITION permission on the certificate.

## Examples

The following example returns the certificate subject.

```
-- First create a certificate.  
CREATE CERTIFICATE Marketing19 WITH  
    START_DATE = '04/04/2004' ,  
    EXPIRY_DATE = '07/07/2007' ,  
    SUBJECT = 'Marketing Print Division';  
GO  
  
-- Now use CertProperty to examine certificate  
-- Marketing19's properties.  
DECLARE @CertSubject sql_variant;  
set @CertSubject = CertProperty( Cert_ID('Marketing19'), 'Subject');  
PRINT CONVERT(nvarchar, @CertSubject);  
GO
```

## See also

[CREATE CERTIFICATE \(Transact-SQL\)](#)

[ALTER CERTIFICATE \(Transact-SQL\)](#)

[CERT\\_ID \(Transact-SQL\)](#) [Encryption Hierarchy](#) [sys.certificates \(Transact-SQL\)](#) [Security Catalog Views \(Transact-SQL\)](#)

# CERT\_ID (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the ID of a certificate.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
Cert_ID ( 'cert_name' )
```

## Arguments

*'cert\_name'*

Is the name of a certificate in the database.

## Return types

**int**

## Remarks

Certificate names are visible in the [sys.certificates](#) catalog view.

## Permissions

Requires some permission on the certificate and that the caller has not been denied VIEW DEFINITION permission on the certificate.

## Examples

The following example returns the ID of a certificate called `ABerglundCert3`.

```
SELECT Cert_ID('ABerglundCert3');
GO
```

## See also

[sys.certificates \(Transact-SQL\)](#)  
[CREATE CERTIFICATE \(Transact-SQL\)](#)  
[Encryption Hierarchy](#)

# CRYPT\_GEN\_RANDOM (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a cryptographic random number generated by the Crypto API (CAPI). The output is a hexadecimal number of the specified number of bytes.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CRYPT_GEN_RANDOM ( length [ , seed ] )
```

## Arguments

*length*

The length of the number being created. Maximum is 8000. *length* is type **int**.

*seed*

Optional data to use as a random seed. There must be at least *length* bytes of data. *seed* is **varbinary(8000)**.

## Returned Types

**varbinary(8000)**

## Permissions

This function is public and does not require any special permissions.

## Examples

### A. Generating a random number

The following example generates a random number 50 bytes long.

```
SELECT CRYPT_GEN_RANDOM(50) ;
```

The following example generates a random number 4 bytes long using a 4-byte seed.

```
SELECT CRYPT_GEN_RANDOM(4, 0x25F18060) ;
```

## See also

[RAND \(Transact-SQL\)](#)

# DECRYPTBYASYMKEY (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Decrypts data with an asymmetric key.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DecryptByAsymKey (Asym_Key_ID , { 'ciphertext' | @ciphertext }
[ , 'Asym_Key_Password' ] )
```

## Arguments

*Asym\_Key\_ID*

Is the ID of an asymmetric key in the database. *Asym\_Key\_ID* is **int**.

*ciphertext*

Is a string of data that has been encrypted with the asymmetric key.

*@ciphertext*

Is a variable of type **varbinary** that contains data that has been encrypted with the asymmetric key.

*Asym\_Key\_Password*

Is the password that was used to encrypt the asymmetric key in the database.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

## Remarks

Encryption/decryption with an asymmetric key is very costly compared to encryption/decryption with a symmetric key. We do not recommend using an asymmetric key when you work with large datasets, such as user data in tables.

## Permissions

Requires CONTROL permission on the asymmetric key.

## Examples

The following example decrypts ciphertext that was encrypted with asymmetric key `JanainaAsymKey02`, which was stored in `AdventureWorks2012.ProtectedData04`. The returned data is decrypted using asymmetric key `JanainaAsymKey02`, which has been decrypted with password `pGFD4bb925DGvbd2439587y`. The plaintext is converted to type **nvarchar**.

```
SELECT CONVERT(nvarchar(max),
    DecryptByAsymKey( AsymKey_Id('JanainaAsymKey02'),
    ProtectedData, N'pGFD4bb925DGvbd2439587y' ))
AS DecryptedData
FROM [AdventureWorks2012].[Sales].[ProtectedData04]
WHERE Description = N'encrypted by asym key''JanainaAsymKey02''' ;
GO
```

## See Also

- [ENCRYPTBYASYMKEY \(Transact-SQL\)](#)
- [CREATE ASYMMETRIC KEY \(Transact-SQL\)](#)
- [ALTER ASYMMETRIC KEY \(Transact-SQL\)](#)
- [DROP ASYMMETRIC KEY \(Transact-SQL\)](#)
- [Choose an Encryption Algorithm](#)
- [Encryption Hierarchy](#)

# DECRYPTBYCERT (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse

Parallel Data Warehouse

Decrypts data with the private key of a certificate.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DecryptByCert ( certificate_ID , { 'ciphertext' | @ciphertext }
    [ , { 'cert_password' | @cert_password } ] )
```

## Arguments

*certificate\_ID*

Is the ID of a certificate in the database. *certificate\_ID* is **int**.

*ciphertext*

Is a string of data that has been encrypted with the public key of the certificate.

@*ciphertext*

Is a variable of type **varbinary** that contains data that has been encrypted with the certificate.

*cert\_password*

Is the password that was used to encrypt the private key of the certificate. Must be Unicode.

@*cert\_password*

Is a variable of type **nchar** or **nvarchar** that contains the password that was used to encrypt the private key of the certificate. Must be Unicode.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

## Remarks

This function decrypts data with the private key of a certificate. Cryptographic transformations that use asymmetric keys consume significant resources. Therefore, EncryptByCert and DecryptByCert are not suited for routine encryption of user data.

## Permissions

Requires CONTROL permission on the certificate.

## Examples

The following example selects rows from `[AdventureWorks2012].[ProtectedData04]` that are marked as `data encrypted by certificate JanainaCert02`. The example decrypts the ciphertext with the private key of

certificate `JanainaCert02`, which it first decrypts with the password of the certificate, `pGFD4bb925DGvbd2439587y`. The decrypted data is converted from **varbinary** to **nvarchar**.

```
SELECT convert(nvarchar(max), DecryptByCert(Cert_Id('JanainaCert02'),
    ProtectedData, N'pGFD4bb925DGvbd2439587y'))
FROM [AdventureWorks2012].[ProtectedData04]
WHERE Description
    = N'data encrypted by certificate '' JanainaCert02'''';
GO
```

## See Also

[ENCRYPTBYCERT \(Transact-SQL\)](#)  
[CREATE CERTIFICATE \(Transact-SQL\)](#)  
[ALTER CERTIFICATE \(Transact-SQL\)](#)  
[DROP CERTIFICATE \(Transact-SQL\)](#)  
[BACKUP CERTIFICATE \(Transact-SQL\)](#)  
[Encryption Hierarchy](#)

# DECRYPTBYKEY (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✗ Parallel Data Warehouse

Decrypts data by using a symmetric key.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
DecryptByKey ( { 'ciphertext' | @ciphertext }
    [ , add_authenticator, { authenticator | @authenticator } ] )
```

## Arguments

*ciphertext*

Is data that has been encrypted with the key. *ciphertext* is **varbinary**.

**@ciphertext**

Is a variable of type **varbinary** that contains data that has been encrypted with the key.

*add\_authenticator*

Indicates whether an authenticator was encrypted together with the plaintext. Must be the same value passed to EncryptByKey when encrypting the data. *add\_authenticator* is **int**.

*authenticator*

Is data from which to generate an authenticator. Must match the value that was supplied to EncryptByKey.

*authenticator* is **sysname**.

**@authenticator**

Is a variable that contains data from which to generate an authenticator. Must match the value that was supplied to EncryptByKey.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

## Remarks

DecryptByKey uses a symmetric key. This symmetric key must already be open in the database. There can be multiple keys open at the same time. You do not have to open the key immediately before decrypting the ciphertext.

Symmetric encryption and decryption is relatively fast, and is suitable for working with large amounts of data.

## Permissions

Requires the symmetric key to have been opened in the current session. For more information, see [OPEN SYMMETRIC KEY \(Transact-SQL\)](#).

# Examples

## A. Decrypting by using a symmetric key

The following example decrypts ciphertext by using a symmetric key.

```
-- First, open the symmetric key with which to decrypt the data.  
OPEN SYMMETRIC KEY SSN_Key_01  
    DECRYPTION BY CERTIFICATE HumanResources037;  
GO  
  
-- Now list the original ID, the encrypted ID, and the  
-- decrypted ciphertext. If the decryption worked, the original  
-- and the decrypted ID will match.  
SELECT NationalIDNumber, EncryptedNationalID  
    AS 'Encrypted ID Number',  
    CONVERT(nvarchar, DecryptByKey(EncryptedNationalID))  
    AS 'Decrypted ID Number'  
    FROM HumanResources.Employee;  
GO
```

## B. Decrypting by using a symmetric key and an authenticating hash

The following example decrypts data that was encrypted together with an authenticator.

```
-- First, open the symmetric key with which to decrypt the data  
OPEN SYMMETRIC KEY CreditCards_Key11  
    DECRYPTION BY CERTIFICATE Sales09;  
GO  
  
-- Now list the original card number, the encrypted card number,  
-- and the decrypted ciphertext. If the decryption worked,  
-- the original number will match the decrypted number.  
SELECT CardNumber, CardNumber_Encrypted  
    AS 'Encrypted card number', CONVERT(nvarchar,  
    DecryptByKey(CardNumber_Encrypted, 1 ,  
    HashBytes('SHA1', CONVERT(varbinary, CreditCardID))))  
    AS 'Decrypted card number' FROM Sales.CreditCard;  
GO
```

## See Also

[ENCRYPTBYKEY \(Transact-SQL\)](#)

[CREATE SYMMETRIC KEY \(Transact-SQL\)](#)

[ALTER SYMMETRIC KEY \(Transact-SQL\)](#)

[DROP SYMMETRIC KEY \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

[Choose an Encryption Algorithm](#)

# DECRYPTBYKEYAUTOASYMKEY (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Decrypts using a symmetric key that is automatically decrypted using an asymmetric key.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DecryptByKeyAutoAsymKey ( akey_ID , akey_password
    , { 'ciphertext' | @ciphertext }
    [ , { add_authenticator | @add_authenticator }
    [ , { authenticator | @authenticator } ] ] )
```

## Arguments

*akey\_ID*

Is the ID of the asymmetric key that is used to protect the symmetric key. *akey\_ID* is **int**.

*akey\_password*

Is the password that protects the private key of the asymmetric key. Can be NULL if the private key is protected by the database master key. *akey\_password* is **nvarchar**.

*'ciphertext'*

Is the data that was encrypted with the key. *ciphertext* is **varbinary**.

*@ciphertext*

Is a variable of type **varbinary** that contains data that was encrypted with the key.

*add\_authenticator*

Indicates whether an authenticator was encrypted together with the plaintext. Must be the same value that is passed to EncryptByKey when encrypting the data. Is 1 if an authenticator was used. *add\_authenticator* is **int**.

*@add\_authenticator*

Indicates whether an authenticator was encrypted together with the plaintext. Must be the same value that is passed to EncryptByKey when encrypting the data.

*authenticator*

Is the data from which to generate an authenticator. Must match the value that was supplied to EncryptByKey. *authenticator* is **sysname**.

*@authenticator*

Is a variable that contains data from which to generate an authenticator. Must match the value that was supplied to EncryptByKey.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

## Remarks

DecryptByKeyAutoAsymKey combines the functionality of OPEN SYMMETRIC KEY and DecryptByKey. In a single operation, it decrypts a symmetric key and uses that key to decrypt ciphertext.

## Permissions

Requires VIEW DEFINITION permission on the symmetric key and CONTROL permission on the asymmetric key.

## Examples

The following example shows how `DecryptByKeyAutoAsymKey` can be used to simplify code that performs a decryption. This code should be run on an **AdventureWorks2012** database that does not already have a database master key.

```
--Create the keys and certificate.  
USE AdventureWorks2012;  
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'mzkvdMlk979438teag$$ds987yghn)(*&4fdg^';  
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'mzkvdMlk979438teag$$ds987yghn)(*&4fdg^';  
CREATE ASYMMETRIC KEY SSN_AKey  
    WITH ALGORITHM = RSA_2048 ;  
GO  
CREATE SYMMETRIC KEY SSN_Key_02 WITH ALGORITHM = DES  
    ENCRYPTION BY ASYMMETRIC KEY SSN_AKey;  
GO  
--  
--Add a column of encrypted data.  
ALTER TABLE HumanResources.Employee  
    ADD EncryptedNationalIDNumber2 varbinary(128);  
OPEN SYMMETRIC KEY SSN_Key_02  
    DECRYPTION BY ASYMMETRIC KEY SSN_AKey;  
UPDATE HumanResources.Employee  
SET EncryptedNationalIDNumber2  
    = EncryptByKey(Key_GUID('SSN_Key_02'), NationalIDNumber);  
GO  
--Close the key used to encrypt the data.  
CLOSE SYMMETRIC KEY SSN_Key_02;  
--  
--There are two ways to decrypt the stored data.  
--  
--OPTION ONE, using DecryptByKey()  
--1. Open the symmetric key.  
--2. Decrypt the data.  
--3. Close the symmetric key.  
OPEN SYMMETRIC KEY SSN_Key_02  
    DECRYPTION BY ASYMMETRIC KEY SSN_AKey;  
SELECT NationalIDNumber, EncryptedNationalIDNumber2  
    AS 'Encrypted ID Number',  
    CONVERT(nvarchar, DecryptByKey(EncryptedNationalIDNumber2))  
    AS 'Decrypted ID Number'  
    FROM HumanResources.Employee;  
CLOSE SYMMETRIC KEY SSN_Key_02;  
--  
--OPTION TWO, using DecryptByKeyAutoAsymKey()  
SELECT NationalIDNumber, EncryptedNationalIDNumber2  
    AS 'Encrypted ID Number',  
    CONVERT(nvarchar, DecryptByKeyAutoAsymKey ( AsymKey_ID('SSN_AKey') , NULL ,EncryptedNationalIDNumber2))  
    AS 'Decrypted ID Number'  
    FROM HumanResources.Employee;  
GO
```

## See Also

[OPEN SYMMETRIC KEY \(Transact-SQL\)](#)

[ENCRYPTBYKEY \(Transact-SQL\)](#)

[DECRYPTBYKEY \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

# DECRYPTBYKEYAUTOCERT (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Decrypts by using a symmetric key that is automatically decrypted with a certificate.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DecryptByKeyAutoCert ( cert_ID , cert_password
    , { 'ciphertext' | @ciphertext }
    [ , { add_authenticator | @add_authenticator }
    [ , { authenticator | @authenticator } ] ] )
```

## Arguments

*cert\_ID*

Is the ID of the certificate that is used to protect the symmetric key. *cert\_ID* is **int**.

*cert\_password*

Is the password that protects the private key of the certificate. Can be NULL if the private key is protected by the database master key. *cert\_password* is **nvarchar**.

*'ciphertext'*

Is the data that was encrypted with the key. *ciphertext* is **varbinary**.

*@ciphertext*

Is a variable of type **varbinary** that contains data that was encrypted with the key.

*add\_authenticator*

Indicates whether an authenticator was encrypted together with the plaintext. Must be the same value that is passed to EncryptByKey when encrypting the data. Is 1 if an authenticator was used. *add\_authenticator* is **int**.

*@add\_authenticator*

Indicates whether an authenticator was encrypted together with the plaintext. Must be the same value that is passed to EncryptByKey when encrypting the data.

*authenticator*

Is the data from which to generate an authenticator. Must match the value that was supplied to EncryptByKey. *authenticator* is **sysname**.

*@authenticator*

Is a variable that contains data from which to generate an authenticator. Must match the value that was supplied to EncryptByKey.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

## Remarks

DecryptByKeyAutoCert combines the functionality of OPEN SYMMETRIC KEY and DecryptByKey. In a single operation, it decrypts a symmetric key and uses that key to decrypt cipher text.

## Permissions

Requires VIEW DEFINITION permission on the symmetric key and CONTROL permission on the certificate.

## Examples

The following example shows how `DecryptByKeyAutoCert` can be used to simplify code that performs a decryption. This code should be run on an **AdventureWorks2012** database that does not already have a database master key.

```
--Create the keys and certificate.
USE AdventureWorks2012;
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'mzkvd1k979438teag$$ds987yghn)(*^&4fdg^';
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'mzkvd1k979438teag$$ds987yghn)(*^&4fdg^';
CREATE CERTIFICATE HumanResources037
    WITH SUBJECT = 'Sammamish HR',
    EXPIRY_DATE = '10/31/2009';
CREATE SYMMETRIC KEY SSN_Key_01 WITH ALGORITHM = DES
    ENCRYPTION BY CERTIFICATE HumanResources037;
GO
----Add a column of encrypted data.
ALTER TABLE HumanResources.Employee
    ADD EncryptedNationalIDNumber varbinary(128);
OPEN SYMMETRIC KEY SSN_Key_01
    DECRYPTION BY CERTIFICATE HumanResources037 ;
UPDATE HumanResources.Employee
SET EncryptedNationalIDNumber
    = EncryptByKey(Key_GUID('SSN_Key_01'), NationalIDNumber);
GO
--
--Close the key used to encrypt the data.
CLOSE SYMMETRIC KEY SSN_Key_01;
--
--There are two ways to decrypt the stored data.
--
--OPTION ONE, using DecryptByKey()
--1. Open the symmetric key
--2. Decrypt the data
--3. Close the symmetric key
OPEN SYMMETRIC KEY SSN_Key_01
    DECRYPTION BY CERTIFICATE HumanResources037;
SELECT NationalIDNumber, EncryptedNationalIDNumber
    AS 'Encrypted ID Number',
    CONVERT(nvarchar, DecryptByKey(EncryptedNationalIDNumber))
    AS 'Decrypted ID Number'
    FROM HumanResources.Employee;
CLOSE SYMMETRIC KEY SSN_Key_01;
--
--OPTION TWO, using DecryptByKeyAutoCert()
SELECT NationalIDNumber, EncryptedNationalIDNumber
    AS 'Encrypted ID Number',
    CONVERT(nvarchar, DecryptByKeyAutoCert ( cert_ID('HumanResources037') , NULL ,EncryptedNationalIDNumber))
    AS 'Decrypted ID Number'
    FROM HumanResources.Employee;
```

## See Also

[OPEN SYMMETRIC KEY \(Transact-SQL\)](#)

[ENCRYPTBYKEY \(Transact-SQL\)](#)

[DECRYPTBYKEY \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

# DECRYPTBYPASSPHRASE (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse

✗ Parallel Data Warehouse

Decrypts data that was encrypted with a passphrase.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
DecryptByPassPhrase ( { 'passphrase' | @passphrase }
    , { 'ciphertext' | @ciphertext }
    [ , { add_authenticator | @add_authenticator }
        , { authenticator | @authenticator } ] )
```

## Arguments

*passphrase*

Is the passphrase that will be used to generate the key for decryption.

*@passphrase*

Is a variable of type **nvarchar**, **char**, **varchar**, or **nchar** that contains the passphrase that will be used to generate the key for decryption.

*'ciphertext'*

Is the ciphertext to be decrypted.

*@ciphertext*

Is a variable of type **varbinary** that contains the ciphertext. The maximum size is 8,000 bytes.

*add\_authenticator*

Indicates whether an authenticator was encrypted together with the plaintext. Is 1 if an authenticator was used. **int**.

*@add\_authenticator*

Indicates whether an authenticator was encrypted together with the plaintext. Is 1 if an authenticator was used. **int**.

*authenticator*

Is the authenticator data. **sysname**.

*@authenticator*

Is a variable that contains data from which to derive an authenticator.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

## Remarks

No permissions are required for executing this function.

Returns NULL if the wrong passphrase or authenticator information is used.

The passphrase is used to generate a decryption key, which will not be persisted.

If an authenticator was included when the ciphertext was encrypted, the authenticator must be provided at decryption time. If the authenticator value provided at decryption time does not match the authenticator value encrypted with the data, the decryption will fail.

## Examples

The following example decrypts the record updated in [EncryptByPassPhrase](#).

```
USE AdventureWorks2012;
-- Get the pass phrase from the user.
DECLARE @PassphraseEnteredByUser nvarchar(128);
SET @PassphraseEnteredByUser
= 'A little learning is a dangerous thing!';

-- Decrypt the encrypted record.
SELECT CardNumber, CardNumber_EncryptedbyPassphrase
    AS 'Encrypted card number', CONVERT(nvarchar,
    DecryptByPassphrase(@PassphraseEnteredByUser, CardNumber_EncryptedbyPassphrase, 1
        , CONVERT(varbinary, CreditCardID)))
    AS 'Decrypted card number' FROM Sales.CreditCard
    WHERE CreditCardID = '3681';
GO
```

## See Also

[Choose an Encryption Algorithm](#)

[ENCRYPTBYPASSPHRASE \(Transact-SQL\)](#)

# ENCRYPTBYASYMKEY (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data

Warehouse Parallel Data Warehouse

Encrypts data with an asymmetric key.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
EncryptByAsymKey ( Asym_Key_ID , { 'plaintext' | @plaintext } )
```

## Arguments

*Asym\_Key\_ID*

Is the ID of an asymmetric key in the database. **int**.

*cleartext*

Is a string of data that will be encrypted with the asymmetric key.

**@plaintext**

Is a variable of type **nvarchar**, **char**, **varchar**, **binary**, **varbinary**, or **nchar** that contains data to be encrypted with the asymmetric key.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

## Remarks

Encryption and decryption with an asymmetric key is very costly compared with encryption and decryption with a symmetric key. We recommend that you not encrypt large datasets, such as user data in tables, using an asymmetric key. Instead, you should encrypt the data using a strong symmetric key and encrypt the symmetric key using an asymmetric key.

**EncryptByAsymKey** return **NULL** if the input exceeds a certain number of bytes, depending on the algorithm. The limits are: a 512 bit RSA key can encrypt up to 53 bytes, a 1024 bit key can encrypt up to 117 bytes, and a 2048 bit key can encrypt up to 245 bytes. (Note that in SQL Server, both certificates and asymmetric keys are wrappers over RSA keys.)

## Examples

The following example encrypts the text stored in `@cleartext` with the asymmetric key `JanainaAsymKey02`. The encrypted data is inserted into the `ProtectedData04` table.

```
INSERT INTO AdventureWorks2012.Sales.ProtectedData04
VALUES( N'Data encrypted by asymmetric key ''JanainaAsymKey02''',
EncryptByAsymKey(AsymKey_ID('JanainaAsymKey02'), @cleartext) );
GO
```

## See Also

[DECRYPTBYASYMKEY \(Transact-SQL\)](#)

[CREATE ASYMMETRIC KEY \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

# ENCRYPTBYCERT (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data

Warehouse Parallel Data Warehouse

Encrypts data with the public key of a certificate.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
EncryptByCert ( certificate_ID , { 'cleartext' | @cleartext } )
```

## Arguments

*certificate\_ID*

The ID of a certificate in the database. **int**.

*cleartext*

A string of data that will be encrypted with the certificate.

**@cleartext**

A variable of type **nvarchar**, **char**, **varchar**, **binary**, **varbinary**, or **nchar** containing data that will be encrypted with the public key of the certificate.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

## Remarks

This function encrypts data with the public key of a certificate. The ciphertext can only be decrypted with the corresponding private key. Such asymmetric transformations are very costly compared to encryption and decryption using a symmetric key. Asymmetric encryption is therefore not recommended when working with large datasets such as user data in tables.

## Examples

This example encrypts the plaintext stored in `@cleartext` with the certificate called `JanainaCert02`. The encrypted data is inserted into table `ProtectedData04`.

```
INSERT INTO [AdventureWorks2012].[ProtectedData04]
VALUES ( N'Data encrypted by certificate ''Shipping04''' ,
EncryptByCert(Cert_ID('JanainaCert02'), @cleartext) );
GO
```

## See Also

[DECRYPTBYCERT \(Transact-SQL\)](#)

[CREATE CERTIFICATE \(Transact-SQL\)](#)

[ALTER CERTIFICATE \(Transact-SQL\)](#)

[DROP CERTIFICATE \(Transact-SQL\)](#)

[BACKUP CERTIFICATE \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

# ENCRYPTBYKEY (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Encrypts data by using a symmetric key.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
EncryptByKey ( key_GUID , { 'cleartext' | @cleartext }
    [, { add_authenticator | @add_authenticator }
    , { authenticator | @authenticator } ] )
```

## Arguments

*key\_GUID*

Is the GUID of the key to be used to encrypt the *cleartext*. **uniqueidentifier**.

*'cleartext'*

Is the data that is to be encrypted with the key.

*@cleartext*

Is a variable of type **nvarchar**, **char**, **varchar**, **binary**, **varbinary**, or **nchar** that contains data that is to be encrypted with the key.

*add\_authenticator*

Indicates whether an authenticator will be encrypted together with the *cleartext*. Must be 1 when using an authenticator. **int**.

*@add\_authenticator*

Indicates whether an authenticator will be encrypted together with the *cleartext*. Must be 1 when using an authenticator. **int**.

*authenticator*

Is the data from which to derive an authenticator. **sysname**.

*@authenticator*

Is a variable that contains data from which to derive an authenticator.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

Returns NULL if the key is not open, if the key does not exist, or if the key is a deprecated RC4 key and the database is not in compatibility level 110 or higher.

## Remarks

EncryptByKey uses a symmetric key. This key must be open. If the symmetric key is already open in the current

session, you do not have to open it again in the context of the query.

The authenticator helps you deter whole-value substitution of encrypted fields. For example, consider the following table of payroll data.

EMPLOYEE_ID	STANDARD_TITLE	BASE_PAY
345	Copy Room Assistant	Fskj%7^edhn00
697	Chief Financial Officer	M0x8900f56543
694	Data Entry Supervisor	Cvc97824%^34f

Without breaking the encryption, a malicious user can infer significant information from the context in which the ciphertext is stored. Because a Chief Financial Officer is paid more than a Copy Room Assistant, it follows that the value encrypted as M0x8900f56543 must be greater than the value encrypted as Fskj%7^edhn00. If so, any user with ALTER permission on the table can give the Copy Room Assistant a raise by replacing the data in his Base\_Pay field with a copy of the data stored in the Base\_Pay field of the Chief Financial Officer. This whole-value substitution attack bypasses encryption altogether.

Whole-value substitution attacks can be thwarted by adding contextual information to the plaintext before encrypting it. This contextual information is used to verify that the plaintext data has not been moved.

If an authenticator parameter is specified when data is encrypted, the same authenticator is required to decrypt the data by using DecryptByKey. At encryption time, a hash of the authenticator is encrypted together with the plaintext. At decryption time, the same authenticator must be passed to DecryptByKey. If the two do not match, the decryption will fail. This indicates that the value has been moved since it was encrypted. We recommend using a column containing a unique and unchanging value as the authenticator. If the authenticator value changes, you might lose access to the data.

Symmetric encryption and decryption is relatively fast, and is suitable for working with large amounts of data.

#### IMPORTANT

Using the SQL Server encryption functions together with the ANSI\_PADDING OFF setting could cause data loss because of implicit conversions. For more information about ANSI\_PADDING, see [SET ANSI\\_PADDING \(Transact-SQL\)](#).

## Examples

The functionality illustrated in the following examples relies on keys and certificates created in [How To: Encrypt a Column of Data](#).

### A. Encrypting a string with a symmetric key

The following example adds a column to the `Employee` table, and then encrypts the value of the Social Security number that is stored in column `NationalIDNumber`.

```
USE AdventureWorks2012;
GO

-- Create a column in which to store the encrypted data.
ALTER TABLE HumanResources.Employee
    ADD EncryptedNationalIDNumber varbinary(128);
GO

-- Open the symmetric key with which to encrypt the data.
OPEN SYMMETRIC KEY SSN_Key_01
    DECRYPTION BY CERTIFICATE HumanResources037;

-- Encrypt the value in column NationalIDNumber with symmetric key
-- SSN_Key_01. Save the result in column EncryptedNationalIDNumber.
UPDATE HumanResources.Employee
SET EncryptedNationalIDNumber
    = EncryptByKey(Key_GUID('SSN_Key_01'), NationalIDNumber);
GO
```

## B. Encrypting a record together with an authentication value

```
USE AdventureWorks2012;

-- Create a column in which to store the encrypted data.
ALTER TABLE Sales.CreditCard
    ADD CardNumber_Encrypted varbinary(128);
GO

-- Open the symmetric key with which to encrypt the data.
OPEN SYMMETRIC KEY CreditCards_Key11
    DECRYPTION BY CERTIFICATE Sales09;

-- Encrypt the value in column CardNumber with symmetric
-- key CreditCards_Key11.
-- Save the result in column CardNumber_Encrypted.
UPDATE Sales.CreditCard
SET CardNumber_Encrypted = EncryptByKey(Key_GUID('CreditCards_Key11'),
    CardNumber, 1, CONVERT( varbinary, CreditCardID ) );
GO
```

## See Also

[DECRYPTBYKEY \(Transact-SQL\)](#)  
[CREATE SYMMETRIC KEY \(Transact-SQL\)](#)  
[ALTER SYMMETRIC KEY \(Transact-SQL\)](#)  
[DROP SYMMETRIC KEY \(Transact-SQL\)](#)  
[Encryption Hierarchy](#)  
[HASHBYTES \(Transact-SQL\)](#)

# ENCRYPTBYPASSPHRASE (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Encrypt data with a passphrase using the TRIPLE DES algorithm with a 128 key bit length.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
EncryptByPassPhrase ( { 'passphrase' | @passphrase }
    , { 'cleartext' | @cleartext }
    [ , { add_authenticator | @add_authenticator }
    , { authenticator | @authenticator } ] )
```

## Arguments

*passphrase*

A passphrase from which to generate a symmetric key.

*@passphrase*

A variable of type **nvarchar**, **char**, **varchar**, **binary**, **varbinary**, or **nchar** containing a passphrase from which to generate a symmetric key.

*cleartext*

The cleartext to be encrypted.

*@cleartext*

A variable of type **nvarchar**, **char**, **varchar**, **binary**, **varbinary**, or **nchar** containing the cleartext. Maximum size is 8,000 bytes.

*add\_authenticator*

Indicates whether an authenticator will be encrypted together with the cleartext. 1 if an authenticator will be added.

**int.**

*@add\_authenticator*

Indicates whether a hash will be encrypted together with the cleartext.

*authenticator*

Data from which to derive an authenticator. **sysname**.

*@authenticator*

A variable containing data from which to derive an authenticator.

## Return Types

**varbinary** with maximum size of 8,000 bytes.

## Remarks

A passphrase is a password that includes spaces. The advantage of using a passphrase is that it is easier to remember a meaningful phrase or sentence than to remember a comparably long string of characters.

This function does not check password complexity.

## Examples

The following example updates a record in the `SalesCreditCard` table and encrypts the value of the credit card number stored in column `CardNumber_EncryptedbyPassphrase`, using the primary key as an authenticator.

```
USE AdventureWorks2012;
GO
-- Create a column in which to store the encrypted data.
ALTER TABLE Sales.CreditCard
    ADD CardNumber_EncryptedbyPassphrase varbinary(256);
GO
-- First get the passphrase from the user.
DECLARE @PassphraseEnteredByUser nvarchar(128);
SET @PassphraseEnteredByUser
    = 'A little learning is a dangerous thing!';

-- Update the record for the user's credit card.
-- In this case, the record is number 3681.
UPDATE Sales.CreditCard
SET CardNumber_EncryptedbyPassphrase = EncryptByPassPhrase(@PassphraseEnteredByUser
    , CardNumber, 1, CONVERT( varbinary, CreditCardID))
WHERE CreditCardID = '3681';
GO
```

## See Also

[DECRYPTBYPASSPHRASE \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

# HASHBYTES (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the MD2, MD4, MD5, SHA, SHA1, or SHA2 hash of its input in SQL Server.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
HASHBYTES ( '<algorithm>', { @input | 'input' } )  
<algorithm> ::= MD2 | MD4 | MD5 | SHA | SHA1 | SHA2_256 | SHA2_512
```

## Arguments

'<algorithm>'

Identifies the hashing algorithm to be used to hash the input. This is a required argument with no default. The single quotation marks are required. Beginning with SQL Server 2016, all algorithms other than SHA2\_256, and SHA2\_512 are deprecated. Older algorithms (not recommended) will continue working, but they will raise a deprecation event.

**@input**

Specifies a variable containing the data to be hashed. **@input** is **varchar**, **nvarchar**, or **varbinary**.

'*input*'

Specifies an expression that evaluates to a character or binary string to be hashed.

The output conforms to the algorithm standard: 128 bits (16 bytes) for MD2, MD4, and MD5; 160 bits (20 bytes) for SHA and SHA1; 256 bits (32 bytes) for SHA2\_256, and 512 bits (64 bytes) for SHA2\_512.

**Applies to:** SQL Server 2012 through SQL Server 2017

For SQL Server 2014 and earlier, allowed input values are limited to 8000 bytes.

## Return Value

**varbinary** (maximum 8000 bytes)

## Examples

### A: Return the hash of a variable

The following example returns the **SHA1** hash of the **nvarchar** data stored in variable **@HashThis**.

```
DECLARE @HashThis nvarchar(4000);  
SET @HashThis = CONVERT(nvarchar(4000), 'ds1fdkjLK85k1dhnv$n000#knf');  
SELECT HASHBYTES('SHA1', @HashThis);
```

### B: Return the hash of a table column

The following example returns the SHA1 hash of the values in column `c1` in the table `Test1`.

```
CREATE TABLE dbo.Test1 (c1 nvarchar(50));
INSERT dbo.Test1 VALUES ('This is a test.');
INSERT dbo.Test1 VALUES ('This is test 2.');
SELECT HASHBYTES('SHA1', c1) FROM dbo.Test1;
```

Here is the result set.

```
-----
0xE7AAB0B4FF0FD2DFB4F0233E2EE7A26CD08F173
0xF643A82F948DEFB922B12E50B950CEE130A934D6

(2 row(s) affected)
```

## See Also

[Choose an Encryption Algorithm](#)

# IS\_OBJECTSIGNED (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Indicates whether an object is signed by a specified certificate or asymmetric key.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
IS_OBJECTSIGNED (
    'OBJECT', @object_id, @class, @thumbprint
)
```

## Arguments

### 'OBJECT'

The type of securable class.

### @object\_id

The object\_id of the object being tested. @object\_id is type **int**.

### @class

The class of the object:

- 'certificate'

- 'asymmetric key'

@class is **sysname**.

### @thumbprint

The SHA thumbprint of the object. @thumbprint is type **varbinary(32)**.

## Returned Types

**int**

## Remarks

IS\_OBJECTSIGNED returns the following values.

RETURN VALUE	DESCRIPTION
NULL	The object is not signed, or the object is not valid.
0	The object is signed, but the signature is not valid.
1	The object is signed.

# Permissions

Requires VIEW DEFINITION on the certificate or asymmetric key.

## Examples

### A. Displaying extended properties on a database

The following example tests if the spt\_fallback\_db table in the **master** database is signed by the schema signing certificate.

```
USE master;
-- Declare a variable to hold a thumbprint and an object name
DECLARE @thumbprint varbinary(20), @objectname sysname;

-- Populate the thumbprint variable with the thumbprint of
-- the master database schema signing certificate
SELECT @thumbprint = thumbprint
FROM sys.certificates
WHERE name LIKE '%SchemaSigningCertificate%';

-- Populate the object name variable with a table name in master
SELECT @objectname = 'spt_fallback_db';

-- Query to see if the table is signed by the thumbprint
SELECT @objectname AS [object name],
IS_OBJECTSIGNED(
'OBJECT', OBJECT_ID(@objectname), 'certificate', @thumbprint
) AS [Is the object signed?];
```

## See Also

[sys.fn\\_check\\_object\\_signatures \(Transact-SQL\)](#)

# KEY\_GUID (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the GUID of a symmetric key in the database.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
Key_GUID( 'Key_Name' )
```

## Arguments

*'Key\_Name'*

The name of a symmetric key in the database.

## Return Types

**uniqueidentifier**

## Remarks

If an identity value was specified when the key was created, its GUID is an MD5 hash of that identity value. If no identity value was specified, the server generated the GUID.

If the key is a temporary key, the key name must start with a number sign (#).

## Permissions

Because temporary keys are only available in the session in which they are created, no permissions are required to access them. To access a key that is not temporary, the caller requires some permission on the key and must not have been denied VIEW permission on the key.

## Examples

The following example returns the GUID of a symmetric key called `ABerglundKey1`.

```
SELECT Key_GUID('ABerglundKey1');
```

## See Also

[CREATE SYMMETRIC KEY \(Transact-SQL\)](#)

[sys.symmetric\\_keys \(Transact-SQL\)](#)

[sys.key\\_encryptions \(Transact-SQL\)](#)

# KEY\_ID (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the ID of a symmetric key in the current database.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
Key_ID ( 'Key_Name' )
```

## Arguments

*'Key\_Name'*

The name of a symmetric key in the database.

## Return Types

**int**

## Remarks

The name of a temporary key must start with a number sign (#).

## Permissions

Because temporary keys are only available in the session in which they are created, no permissions are required to access them. To access a key that is not temporary, the caller needs some permission on the key and must not have been denied VIEW permission on the key.

## Examples

### A. Returning the ID of a symmetric key

The following example returns the ID of a key called `ABerglundKey1`.

```
SELECT KEY_ID('ABerglundKey1');
```

### B. Returning the ID of a temporary symmetric key

The following example returns the ID of a temporary symmetric key. Note that `#` is prepended to the key name.

```
SELECT KEY_ID('#ABerglundKey2');
```

## See Also

[KEY\\_GUID \(Transact-SQL\)](#)

[CREATE SYMMETRIC KEY \(Transact-SQL\)](#)

[sys.symmetric\\_keys \(Transact-SQL\)](#)

[sys.key\\_encryptions \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

# KEY\_NAME (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the name of the symmetric key from either a symmetric key GUID or cipher text.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
KEY_NAME ( ciphertext | key_guid )
```

## Arguments

*ciphertext*

Is the text encrypted by the symmetric key. *ciphertext* is type **varbinary(8000)**.

*key\_guid*

Is the GUID of the symmetric key. *key\_guid* is type **uniqueidentifier**.

## Returned Types

**varchar(128)**

## Permissions

Beginning in SQL Server 2005, the visibility of metadata is limited to securables that a user either owns or on which the user has been granted some permission. For more information, see [Metadata Visibility Configuration](#).

## Examples

### A. Displaying the name of a symmetric key using the key\_guid

The **master** database contains a symmetric key named ##MS\_ServiceMasterKey##. The following example gets the GUID of that key from the sys.symmetric\_keys dynamic management view, assigns it to a variable, and then passes that variable to the KEY\_NAME function to demonstrate how to return the name that corresponds to the GUID.

```
USE master;
GO
DECLARE @guid uniqueidentifier ;
SELECT @guid = key_guid FROM sys.symmetric_keys
WHERE name = '##MS_ServiceMasterKey##' ;
-- Demonstration of passing a GUID to KEY_NAME to receive a name
SELECT KEY_NAME(@guid) AS [Name of Key];
```

### B. Displaying the name of a symmetric key using the cipher text

The following example demonstrates the entire process of creating a symmetric key and populating data into a table. The example then shows how KEY\_NAME returns the name of the key when passed the encrypted text.

```

-- Create a symmetric key
CREATE SYMMETRIC KEY TestSymKey
    WITH ALGORITHM = AES_128,
    KEY_SOURCE = 'The square of the hypotenuse is equal to the sum of the squares of the sides',
    IDENTITY_VALUE = 'Pythagoras'
    ENCRYPTION BY PASSWORD = 'pGFD4bb925DGvbd2439587y' ;
GO
-- Create a table for the demonstration
CREATE TABLE DemoKey
(IDCol int IDENTITY PRIMARY KEY,
SecretCol varbinary(256) NOT NULL)
GO
-- Open the symmetric key if not already open
OPEN SYMMETRIC KEY TestSymKey
    DECRYPTION BY PASSWORD = 'pGFD4bb925DGvbd2439587y';
GO
-- Insert a row into the DemoKey table
DECLARE @key_GUID uniqueidentifier;
SELECT @key_GUID = key_guid FROM sys.symmetric_keys
WHERE name LIKE 'TestSymKey' ;
INSERT INTO DemoKey(SecretCol)
VALUES ( ENCRYPTBYKEY (@key_GUID, 'EncryptedText'))
GO
-- Verify the DemoKey data
SELECT * FROM DemoKey;
GO
-- Decrypt the data
DECLARE @ciphertext varbinary(256);
SELECT @ciphertext = SecretCol
FROM DemoKey WHERE IDCol = 1 ;
SELECT CAST (
DECRYPTBYKEY( @ciphertext)
AS varchar(100) ) AS SecretText ;
-- Use KEY_NAME to view the name of the key
SELECT KEY_NAME(@ciphertext) AS [Name of Key] ;

```

## See Also

[sys.symmetric\\_keys \(Transact-SQL\)](#)

[ENCRYPTBYKEY \(Transact-SQL\)](#)

[DECRYPTBYKEYAUTOASYMKEY \(Transact-SQL\)](#)

# SIGNBYASYMKEY (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Signs plaintext with an asymmetric key

[Transact-SQL Syntax Conventions](#)

## Syntax

```
SignByAsymKey( Asym_Key_ID , @plaintext [ , 'password' ] )
```

## Arguments

*Asym\_Key\_ID*

Is the ID of an asymmetric key in the current database. *Asym\_Key\_ID* is **int**.

**@plaintext**

Is a variable of type **nvarchar**, **char**, **varchar**, or **nchar** containing data that will be signed with the asymmetric key.

*password*

Is the password with which the private key is protected. *password* is **nvarchar(128)**.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

## Remarks

Requires CONTROL permission on the asymmetric key.

## Examples

The following example creates a table, `SignedData04`, in which to store plaintext and its signature. It next inserts a record in the table, signed with asymmetric key `PrimeKey`, which is first decrypted with password `'pGFD4bb925DGvbd2439587y'`.

```
-- Create a table in which to store the data
CREATE TABLE [SignedData04](Description nvarchar(max), Data nvarchar(max), DataSignature varbinary(8000));
GO
-- Store data together with its signature
DECLARE @clear_text_data nvarchar(max);
set @clear_text_data = N'Important numbers 2, 3, 5, 7, 11, 13, 17,
    19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79,
    83, 89, 97';
INSERT INTO [SignedData04]
VALUES( N'data encrypted by asymmetric key ''PrimeKey'''',
    @clear_text_data, SignByAsymKey( AsymKey_Id( 'PrimeKey' ),
    @clear_text_data, N'pGFD4bb925DGvbd2439587y' ) );
GO
```

## See Also

[ASYMKEY\\_ID \(Transact-SQL\)](#)  
[VERIFYSIGNEDBYASYMKEY \(Transact-SQL\)](#)  
[CREATE ASYMMETRIC KEY \(Transact-SQL\)](#)  
[ALTER ASYMMETRIC KEY \(Transact-SQL\)](#)  
[DROP ASYMMETRIC KEY \(Transact-SQL\)](#)  
[Encryption Hierarchy](#)

# SIGNBYCERT (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Signs text with a certificate and returns the signature.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
SignByCert ( certificate_ID , @cleartext [ , 'password' ] )
```

## Arguments

*certificate\_ID*

Is the ID of a certificate in the current database. *certificate\_ID* is **int**.

*@cleartext*

Is a variable of type **nvarchar**, **char**, **varchar**, or **nchar** that contains data that will be signed.

'*password*'

Is the password with which the certificate's private key was encrypted. *password* is **nvarchar(128)**.

## Return Types

**varbinary** with a maximum size of 8,000 bytes.

## Remarks

Requires CONTROL permission on the certificate.

## Examples

The following example signs the text in `@SensitiveData` with certificate `ABerglundCert07`, having first decrypted the certificate with password "pGFD4bb925DGvbd2439587y". It then inserts the cleartext and the signature in table `SignedData04`.

```
DECLARE @SensitiveData nvarchar(max);
SET @SensitiveData = N'Saddle Price Points are
    2, 3, 5, 7, 11, 13, 17, 19, 23, 29';
INSERT INTO [SignedData04]
    VALUES( N'data signed by certificate ''ABerglundCert07''' ,
        @SensitiveData, SignByCert( Cert_Id( 'ABerglundCert07' ),
        @SensitiveData, N'pGFD4bb925DGvbd2439587y' ) );
GO
```

## See Also

[VERIFYSIGNEDBYCERT \(Transact-SQL\)](#)

[CERT\\_ID \(Transact-SQL\)](#)

[CREATE CERTIFICATE \(Transact-SQL\)](#)

[ALTER CERTIFICATE \(Transact-SQL\)](#)

[DROP CERTIFICATE \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

# SYMKEYPROPERTY (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the algorithm of a symmetric key created from an EKM module.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
SYMKEYPROPERTY ( Key_ID , 'algorithm_desc' | 'string_sid' | 'sid' )
```

## Arguments

### *Key\_ID*

Is the Key\_ID of a symmetric key in the database. To find the Key\_ID when you only know the key name, use SYMKEY\_ID. *Key\_ID* is data type **int**.

### 'algorithm\_desc'

Specifies that the output returns the algorithm description of the symmetric key. Only available for symmetric keys created from an EKM module.

## Return Types

### **sql\_variant**

## Permissions

Requires some permission on the symmetric key and that the caller has not been denied VIEW permission on the symmetric key.

## Examples

The following example returns the algorithm of the symmetric key with Key\_ID 256.

```
SELECT SYMKEYPROPERTY(256, 'algorithm_desc') AS Algorithm ;
GO
```

## See Also

[ASYMKEY\\_ID \(Transact-SQL\)](#)

[ALTER SYMMETRIC KEY \(Transact-SQL\)](#)

[DROP SYMMETRIC KEY \(Transact-SQL\)](#)

[Encryption Hierarchy](#)

[sys.symmetric\\_keys \(Transact-SQL\)](#)

[Security Catalog Views \(Transact-SQL\)](#)

[KEY\\_ID \(Transact-SQL\)](#)

[ASYMKEYPROPERTY \(Transact-SQL\)](#)

# VERIFYSIGNEDBYCERT (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Tests whether digitally signed data has been changed since it was signed.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
VerifySignedByCert( Cert_ID , signed_data , signature )
```

## Arguments

*Cert\_ID*

Is the ID of a certificate in the database. *Cert\_ID* is **int**.

*signed\_data*

Is a variable of type **nvarchar**, **char**, **varchar**, or **nchar** that contains data that has been signed with a certificate.

*signature*

Is the signature that was attached to the signed data. *signature* is **varbinary**.

## Return Types

**int**

Returns 1 when signed data is unchanged; otherwise 0.

## Remarks

**VerifySignedBycert** decrypts the signature of the data by using the public key of the specified certificate, and compares the decrypted value to a newly computed MD5 hash of the data. If the values match, the signature is confirmed to be valid.

## Permissions

Requires **VIEW DEFINITION** permission on the certificate.

## Examples

### A. Verifying that signed data has not been tampered with

The following example tests whether the information in `signed_Data` has been changed since it was signed with the certificate called `Shipping04`. The signature is stored in `DataSignature`. The certificate, `Shipping04`, is passed to `Cert_ID`, which returns the ID of the certificate in the database. If `VerifySignedByCert` returns 1, the signature is correct. If `VerifySignedByCert` returns 0, the data in `Signed_Data` is not the data that was used to generate `DataSignature`. In this case, either `signed_Data` has been changed since it was signed or `Signed_Data` was signed with a different certificate.

```
SELECT Data, VerifySignedByCert( Cert_Id( 'Shipping04' ),
    Signed_Data, DataSignature ) AS IsSignatureValid
FROM [AdventureWorks2012].[SignedData04]
WHERE Description = N'data signed by certificate ''Shipping04''' ;
GO
```

## B. Returning only records that have a valid signature

This query returns only records that have not been changed since they were signed using certificate `Shipping04`.

```
SELECT Data FROM [AdventureWorks2012].[SignedData04]
WHERE VerifySignedByCert( Cert_Id( 'Shipping04' ), Data,
    DataSignature ) = 1
AND Description = N'data signed by certificate ''Shipping04''' ;
GO
```

## See Also

[CERT\\_ID \(Transact-SQL\)](#)  
[SIGNBYCERT \(Transact-SQL\)](#)  
[CREATE CERTIFICATE \(Transact-SQL\)](#)  
[ALTER CERTIFICATE \(Transact-SQL\)](#)  
[DROP CERTIFICATE \(Transact-SQL\)](#)  
[BACKUP CERTIFICATE \(Transact-SQL\)](#)  
[Encryption Hierarchy](#)

# VERIFYSIGNEDBYASYMKEY (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Tests whether digitally signed data has been changed since it was signed.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
VerifySignedByAsymKey( Asym_Key_ID , clear_text , signature )
```

## Arguments

*Asym\_Key\_ID*

Is the ID of an asymmetric key certificate in the database.

*clear\_text*

Is clear text data that is being verified.

*signature*

Is the signature that was attached to the signed data. *signature* is **varbinary**.

## Return Types

**int**

Returns 1 when the signatures match; otherwise 0.

## Remarks

**VerifySignedByAsymKey** decrypts the signature of the data by using the public key of the specified asymmetric key, and compares the decrypted value to a newly computed MD5 hash of the data. If the values match, the signature is confirmed to be valid.

## Permissions

Requires VIEW DEFINITION permission on the asymmetric key.

## Examples

### A. Testing for data with a valid signature

The following example returns 1 if the selected data has not been changed since it was signed with asymmetric key `willisKey74`. The example returns 0 if the data has been tampered with.

```
SELECT Data,
       VerifySignedByAsymKey( AsymKey_Id( 'WillisKey74' ), SignedData,
                               DataSignature ) as IsSignatureValid
  FROM [AdventureWorks2012].[SignedData04]
 WHERE Description = N'data encrypted by asymmetric key ''WillisKey74'''';
GO
RETURN;
```

## B. Returning a result set that contains data with a valid signature

The following example returns rows in `SignedData04` that contain data that has not been changed since it was signed with asymmetric key `WillisKey74`. The example calls the function `AsymKey_ID` to obtain the ID of the asymmetric key from the database.

```
SELECT Data
  FROM [AdventureWorks2012].[SignedData04]
 WHERE VerifySignedByAsymKey( AsymKey_Id( 'WillisKey74' ), Data,
                               DataSignature ) = 1
   AND Description = N'data encrypted by asymmetric key ''WillisKey74'''';
GO
```

## See Also

[ASYMKEY\\_ID \(Transact-SQL\)](#)  
[SIGNBYASYMKEY \(Transact-SQL\)](#)  
[CREATE ASYMMETRIC KEY \(Transact-SQL\)](#)  
[ALTER ASYMMETRIC KEY \(Transact-SQL\)](#)  
[DROP ASYMMETRIC KEY \(Transact-SQL\)](#)  
[Encryption Hierarchy](#)

# Cursor Functions (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

The following scalar functions return information about cursors:

<code>@@CURSOR_ROWS</code>	<code>CURSOR_STATUS</code>
<code>@@FETCH_STATUS</code>	

All cursor functions are nondeterministic. This means these functions do not always return the same results every time they are called, even with the same set of input values. For more information about function determinism, see [Deterministic and Nondeterministic Functions](#).

## See also

[Built-in Functions \(Transact-SQL\)](#)

# @@CURSOR\_ROWS (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the number of qualifying rows currently in the last cursor opened on the connection. To improve performance, SQL Server can populate large keyset and static cursors asynchronously. @@CURSOR\_ROWS can be called to determine that the number of the rows that qualify for a cursor are retrieved at the time @@CURSOR\_ROWS is called.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@CURSOR_ROWS
```

## Return types

**integer**

## Return Value

RETURN VALUE	DESCRIPTION
$-m$	The cursor is populated asynchronously. The value returned ( $-m$ ) is the number of rows currently in the keyset.
$-1$	The cursor is dynamic. Because dynamic cursors reflect all changes, the number of rows that qualify for the cursor is constantly changing. It can never be definitely stated that all qualified rows have been retrieved.
$0$	No cursors have been opened, no rows qualified for the last opened cursor, or the last-opened cursor is closed or deallocated.
$n$	The cursor is fully populated. The value returned ( $n$ ) is the total number of rows in the cursor.

## Remarks

The number returned by @@CURSOR\_ROWS is negative if the last cursor was opened asynchronously. Keyset-driver or static cursors are opened asynchronously if the value for sp\_configure cursor threshold is greater than 0 and the number of rows in the cursor result set is greater than the cursor threshold.

## Examples

The following example declares a cursor and uses `SELECT` to display the value of `@@CURSOR_ROWS`. The setting has a value of `0` before the cursor is opened and a value of `-1` to indicate that the cursor keyset is populated

asynchronously.

```
USE AdventureWorks2012;
GO
SELECT @@CURSOR_ROWS;
DECLARE Name_Cursor CURSOR FOR
SELECT LastName ,@@CURSOR_ROWS FROM Person.Person;
OPEN Name_Cursor;
FETCH NEXT FROM Name_Cursor;
SELECT @@CURSOR_ROWS;
CLOSE Name_Cursor;
DEALLOCATE Name_Cursor;
GO
```

Here are the result sets.

-----

0

LastName

-----

Sanchez

-----

-1

## See also

[Cursor Functions \(Transact-SQL\)](#)

[OPEN \(Transact-SQL\)](#)

# @@FETCH\_STATUS (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the status of the last cursor FETCH statement issued against any cursor currently opened by the connection.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@FETCH_STATUS
```

## Return Type

**integer**

## Return Value

RETURN VALUE	DESCRIPTION
0	The FETCH statement was successful.
-1	The FETCH statement failed or the row was beyond the result set.
-2	The row fetched is missing.
-9	The cursor is not performing a fetch operation.

## Remarks

Because @@FETCH\_STATUS is global to all cursors on a connection, use @@FETCH\_STATUS carefully. After a FETCH statement is executed, the test for @@FETCH\_STATUS must occur before any other FETCH statement is executed against another cursor. The value of @@FETCH\_STATUS is undefined before any fetches have occurred on the connection.

For example, a user executes a FETCH statement from one cursor, and then calls a stored procedure that opens and processes the results from another cursor. When control is returned from the called stored procedure, @@FETCH\_STATUS reflects the last FETCH executed in the stored procedure, not the FETCH statement executed before the stored procedure is called.

To retrieve the last fetch status of a specific cursor, query the **fetch\_status** column of the **sys.dm\_exec\_cursors** dynamic management function.

## Examples

The following example uses `@@FETCH_STATUS` to control cursor activities in a `WHILE` loop.

```
DECLARE Employee_Cursor CURSOR FOR
SELECT BusinessEntityID, JobTitle
FROM AdventureWorks2012.HumanResources.Employee;
OPEN Employee_Cursor;
FETCH NEXT FROM Employee_Cursor;
WHILE @@FETCH_STATUS = 0
    BEGIN
        FETCH NEXT FROM Employee_Cursor;
    END;
CLOSE Employee_Cursor;
DEALLOCATE Employee_Cursor;
GO
```

## See Also

[Cursor Functions \(Transact-SQL\)](#)

[FETCH \(Transact-SQL\)](#)

# CURSOR\_STATUS (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

A scalar function that allows the caller of a stored procedure to determine whether or not the procedure has returned a cursor and result set for a given parameter.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CURSOR_STATUS
(
    { 'local' , 'cursor_name' }
    | { 'global' , 'cursor_name' }
    | { 'variable' , 'cursor_variable' }
)
```

## Arguments

'local'

Specifies a constant that indicates the source of the cursor is a local cursor name.

'cursor\_name'

Is the name of the cursor. A cursor name must conform to the rules for identifiers.

'global'

Specifies a constant that indicates the source of the cursor is a global cursor name.

'variable'

Specifies a constant that indicates the source of the cursor is a local variable.

'cursor\_variable'

Is the name of the cursor variable. A cursor variable must be defined using the **cursor** data type.

## Return types

**smallint**

RETURN VALUE	CURSOR NAME	CURSOR VARIABLE
1	The result set of the cursor has at least one row.  For insensitive and keyset cursors, the result set has at least one row.  For dynamic cursors, the result set can have zero, one, or more rows.	The cursor allocated to this variable is open.  For insensitive and keyset cursors, the result set has at least one row.  For dynamic cursors, the result set can have zero, one, or more rows.

RETURN VALUE	CURSOR NAME	CURSOR VARIABLE
0	The result set of the cursor is empty.*	The cursor allocated to this variable is open, but the result set is definitely empty.*
-1	The cursor is closed.	The cursor allocated to this variable is closed.
-2	Not applicable.	<p>Can be:</p> <p>No cursor was assigned to this OUTPUT variable by the previously called procedure.</p> <p>A cursor was assigned to this OUTPUT variable by the previously called procedure, but it was in a closed state upon completion of the procedure. Therefore, the cursor is deallocated and not returned to the calling procedure.</p> <p>There is no cursor assigned to a declared cursor variable.</p>
-3	A cursor with the specified name does not exist.	A cursor variable with the specified name does not exist, or if one exists it has not yet had a cursor allocated to it.

\*Dynamic cursors never return this result.

## Examples

The following example uses the `CURSOR_STATUS` function to show the status of a cursor before and after it is opened and closed.

```
CREATE TABLE #TMP
(
    ii int
)
GO

INSERT INTO #TMP(ii) VALUES(1)
INSERT INTO #TMP(ii) VALUES(2)
INSERT INTO #TMP(ii) VALUES(3)

GO

--Create a cursor.
DECLARE cur CURSOR
FOR SELECT * FROM #TMP

--Display the status of the cursor before and after opening
--closing the cursor.

SELECT CURSOR_STATUS('global','cur') AS 'After declare'
OPEN cur
SELECT CURSOR_STATUS('global','cur') AS 'After Open'
CLOSE cur
SELECT CURSOR_STATUS('global','cur') AS 'After Close'

--Remove the cursor.
DEALLOCATE cur

--Drop the table.
DROP TABLE #TMP
```

Here is the result set.

After declare

-----

-1

After Open

-----

1

After Close

-----

-1

## See also

[Cursor Functions \(Transact-SQL\)](#)

[Data Types \(Transact-SQL\)](#)

# Data Type Functions (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following scalar functions return information about various data type values.

## In this section

<a href="#">DATALENGTH (Transact-SQL)</a>	<a href="#">IDENT_SEED (Transact-SQL)</a>
<a href="#">IDENT_CURRENT (Transact-SQL)</a>	<a href="#">IDENTITY (Function) (Transact-SQL)</a>
<a href="#">IDENT_INCR (Transact-SQL)</a>	<a href="#">SQL_VARIANT_PROPERTY (Transact-SQL)</a>

# DATALENGTH (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the number of bytes used to represent any expression.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DATALENGTH ( expression )
```

## Arguments

*expression*

Is an [expression](#) of any data type.

## Return types

**bigint** if *expression* is of the **varchar(max)**, **nvarchar(max)** or **varbinary(max)** data types; otherwise **int**.

## Remarks

DATALENGTH is especially useful with **varchar**, **varbinary**, **text**, **image**, **nvarchar**, and **ntext** data types because these data types can store variable-length data.

The DATALENGTH of NULL is NULL.

### NOTE

Compatibility levels can affect return values. For more information about compatibility levels, see [ALTER DATABASE Compatibility Level \(Transact-SQL\)](#).

## Examples

The following example finds the length of the `Name` column in the `Product` table.

```
USE AdventureWorks2012;
GO
SELECT length = DATALENGTH(Name), Name
FROM Production.Product
ORDER BY Name;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example finds the length of the `Name` column in the `Product` table.

```
-- Uses AdventureWorks

SELECT length = DATALENGTH(EnglishProductName), EnglishProductName
FROM dbo.DimProduct
ORDER BY EnglishProductName;
GO
```

## See also

- [LEN \(Transact-SQL\)](#)
- [CAST and CONVERT \(Transact-SQL\)](#)
- [Data Types \(Transact-SQL\)](#)
- [System Functions \(Transact-SQL\)](#)

# IDENT\_CURRENT (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the last identity value generated for a specified table or view. The last identity value generated can be for any session and any scope.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
IDENT_CURRENT( 'table_name' )
```

## Arguments

*table\_name*

Is the name of the table whose identity value is returned. *table\_name* is **varchar**, with no default.

## Return Types

**numeric(38,0)**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

In SQL Server, a user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as IDENT\_CURRENT may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

IDENT\_CURRENT is similar to the SQL Server 2000 identity functions SCOPE\_IDENTITY and @@IDENTITY. All three functions return last-generated identity values. However, the scope and session on which *last* is defined in each of these functions differ:

- IDENT\_CURRENT returns the last identity value generated for a specific table in any session and any scope.
- @@IDENTITY returns the last identity value generated for any table in the current session, across all scopes.
- SCOPE\_IDENTITY returns the last identity value generated for any table in the current session and the current scope.

When the IDENT\_CURRENT value is NULL (because the table has never contained rows or has been truncated), the IDENT\_CURRENT function returns the seed value.

Failed statements and transactions can change the current identity for a table and create gaps in the identity column values. The identity value is never rolled back even though the transaction that tried to insert the

value into the table is not committed. For example, if an INSERT statement fails because of an IGNORE\_DUP\_KEY violation, the current identity value for the table is still incremented.

Be cautious about using IDENT\_CURRENT to predict the next generated identity value. The actual generated value may be different from IDENT\_CURRENT plus IDENT\_INCR because of insertions performed by other sessions.

## Examples

### A. Returning the last identity value generated for a specified table

The following example returns the last identity value generated for the `Person.Address` table in the `AdventureWorks2012` database.

```
USE AdventureWorks2012;
GO
SELECT IDENT_CURRENT ('Person.Address') AS Current_Identity;
GO
```

### B. Comparing identity values returned by IDENT\_CURRENT, @@IDENTITY and SCOPE\_IDENTITY

The following example shows the different identity values that are returned by `IDENT_CURRENT`, `@@IDENTITY`, and `SCOPE_IDENTITY`.

```

USE AdventureWorks2012;
GO
IF OBJECT_ID(N't6', N'U') IS NOT NULL
    DROP TABLE t6;
GO
IF OBJECT_ID(N't7', N'U') IS NOT NULL
    DROP TABLE t7;
GO
CREATE TABLE t6(id int IDENTITY);
CREATE TABLE t7(id int IDENTITY(100,1));
GO
CREATE TRIGGER t6ins ON t6 FOR INSERT
AS
BEGIN
    INSERT t7 DEFAULT VALUES
END;
GO
--End of trigger definition

SELECT id FROM t6;
--IDs empty.

SELECT id FROM t7;
--ID is empty.

--Do the following in Session 1
INSERT t6 DEFAULT VALUES;
SELECT @@IDENTITY;
/*Returns the value 100. This was inserted by the trigger.*/

SELECT SCOPE_IDENTITY();
/* Returns the value 1. This was inserted by the
INSERT statement two statements before this query.*/

SELECT IDENT_CURRENT('t7');
/* Returns value inserted into t7, that is in the trigger.*/

SELECT IDENT_CURRENT('t6');
/* Returns value inserted into t6. This was the INSERT statement four statements before this query.*/

-- Do the following in Session 2.
SELECT @@IDENTITY;
/* Returns NULL because there has been no INSERT action
up to this point in this session.*/

SELECT SCOPE_IDENTITY();
/* Returns NULL because there has been no INSERT action
up to this point in this scope in this session.*/

SELECT IDENT_CURRENT('t7');
/* Returns the last value inserted into t7.*/

```

## See Also

[@@IDENTITY \(Transact-SQL\)](#)  
[SCOPE\\_IDENTITY \(Transact-SQL\)](#)  
[IDENT\\_INCR \(Transact-SQL\)](#)  
[IDENT\\_SEED \(Transact-SQL\)](#)  
[Expressions \(Transact-SQL\)](#)  
[System Functions \(Transact-SQL\)](#)

# IDENT\_INCR (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the increment value (returned as **numeric** (@@MAXPRECISION,0)) specified during the creation of an identity column in a table or view that has an identity column.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
IDENT_INCR ( 'table_or_view' )
```

## Arguments

'*table\_or\_view*'

Is an [expression](#) specifying the table or view to check for a valid identity increment value. *table\_or\_view* can be a character string constant enclosed in quotation marks, a variable, a function, or a column name. *table\_or\_view* is **char**, **nchar**, **varchar**, or **nvarchar**.

## Return Types

**numeric**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

In SQL Server, a user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as IDENT\_INCR may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Examples

### A. Returning the increment value for a specified table

The following example returns the increment value for the `Person.Address` table in the AdventureWorks2012 database.

```
USE AdventureWorks2012;
GO
SELECT IDENT_INCR('Person.Address') AS Identity_Increment;
GO
```

### B. Returning the increment value from multiple tables

The following example returns the tables in the AdventureWorks2012 database that include an identity column with an increment value.

```
USE AdventureWorks2012;
GO
SELECT TABLE_SCHEMA, TABLE_NAME,
IDENT_INCR(TABLE_SCHEMA + '.' + TABLE_NAME) AS IDENT_INCR
FROM INFORMATION_SCHEMA.TABLES
WHERE IDENT_INCR(TABLE_SCHEMA + '.' + TABLE_NAME) IS NOT NULL;
```

Here is a partial result set.

TABLE_SCHEMA	TABLE_NAME	IDENT_INCR
--------------	------------	------------

Person	Address	1
--------	---------	---

Production	ProductReview	1
------------	---------------	---

Production	TransactionHistory	1
------------	--------------------	---

Person	AddressType	1
--------	-------------	---

Production	ProductSubcategory	1
------------	--------------------	---

Person	vAdditionalContactInfo	1
--------	------------------------	---

dbo	AWBuildVersion	1
-----	----------------	---

Production	BillOfMaterials	1
------------	-----------------	---

## See Also

[Expressions \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

[IDENT\\_CURRENT \(Transact-SQL\)](#)

[IDENT\\_SEED \(Transact-SQL\)](#)

[DBCC CHECKIDENT \(Transact-SQL\)](#)

[sys.identity\\_columns \(Transact-SQL\)](#)

# IDENT\_SEED (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the original seed value (returned as **numeric(@@MAXPRECISION,0)**) that was specified when an identity column in a table or a view was created. Changing the current value of an identity column by using DBCC CHECKIDENT does not change the value returned by this function.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
IDENT_SEED ( 'table_or_view' )
```

## Arguments

'*table\_or\_view*'

Is an [expression](#) that specifies the table or view to check for a identity seed value. *table\_or\_view* can be a character string constant enclosed in quotation marks, a variable, a function, or a column name. *table\_or\_view* is **char**, **nchar**, **varchar**, or **nvarchar**.

## Return Types

**numeric**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

In SQL Server, a user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as IDENT\_SEED may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Examples

### A. Returning the seed value from a specified table

The following example returns the seed value for the `Person.Address` table in the AdventureWorks2012 database.

```
USE AdventureWorks2012;
GO
SELECT IDENT_SEED('Person.Address') AS Identity_Seed;
GO
```

### B. Returning the seed value from multiple tables

The following example returns the tables in the AdventureWorks2012 database that include an identity column with a seed value.

```
USE AdventureWorks2012;
GO
SELECT TABLE_SCHEMA, TABLE_NAME,
    IDENT_SEED(TABLE_SCHEMA + '.' + TABLE_NAME) AS IDENT_SEED
FROM INFORMATION_SCHEMA.TABLES
WHERE IDENT_SEED(TABLE_SCHEMA + '.' + TABLE_NAME) IS NOT NULL;
GO
```

Here is a partial result set.

TABLE_SCHEMA	TABLE_NAME	IDENT_SEED
--------------	------------	------------

-----	-----	-----
-------	-------	-------

Person Address 1
------------------

Production ProductReview 1
----------------------------

Production TransactionHistory 100000
--------------------------------------

Person AddressType 1
----------------------

Production ProductSubcategory 1
---------------------------------

Person vAdditionalContactInfo 1
---------------------------------

dbo AWBuildVersion 1
----------------------

## See Also

[Expressions \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

[IDENT\\_CURRENT \(Transact-SQL\)](#)

[IDENT\\_INCR \(Transact-SQL\)](#)

[DBCC CHECKIDENT \(Transact-SQL\)](#)

[sys.identity\\_columns \(Transact-SQL\)](#)

# IDENTITY (Function) (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Is used only in a SELECT statement with an INTO *table* clause to insert an identity column into a new table.

Although similar, the IDENTITY function is not the IDENTITY property that is used with CREATE TABLE and ALTER TABLE.

## NOTE

To create an automatically incrementing number that can be used in multiple tables or that can be called from applications without referencing any table, see [Sequence Numbers](#).



## Syntax

```
IDENTITY (data_type [ , seed , increment ] ) AS column_name
```

## Arguments

### *data\_type*

Is the data type of the identity column. Valid data types for an identity column are any data types of the integer data type category, except for the **bit** data type, or **decimal** data type.

### *seed*

Is the integer value to be assigned to the first row in the table. Each subsequent row is assigned the next identity value, which is equal to the last IDENTITY value plus the *increment* value. If neither *seed* nor *increment* is specified, both default to 1.

### *increment*

Is the integer value to add to the *seed* value for successive rows in the table.

### *column\_name*

Is the name of the column that is to be inserted into the new table.

## Return Types

Returns the same as *data\_type*.

## Remarks

Because this function creates a column in a table, a name for the column must be specified in the select list in one of the following ways:

```
--(1)
SELECT IDENTITY(int, 1,1) AS ID_Num
INTO NewTable
FROM OldTable;

--(2)
SELECT ID_Num = IDENTITY(int, 1, 1)
INTO NewTable
FROM OldTable;
```

## Examples

The following example inserts all rows from the `Contact` table from the AdventureWorks2012database into a new table called `NewContact`. The IDENTITY function is used to start identification numbers at 100 instead of 1 in the `NewContact` table.

```
USE AdventureWorks2012;
GO
IF OBJECT_ID (N'Person.NewContact', N'U') IS NOT NULL
    DROP TABLE Person.NewContact;
GO
ALTER DATABASE AdventureWorks2012 SET RECOVERY BULK_LOGGED;
GO
SELECT  IDENTITY(smallint, 100, 1) AS ContactNum,
        FirstName AS First,
        LastName AS Last
INTO Person.NewContact
FROM Person.Person;
GO
ALTER DATABASE AdventureWorks2012 SET RECOVERY FULL;
GO
SELECT ContactNum, First, Last FROM Person.NewContact;
GO
```

## See Also

[CREATE TABLE \(Transact-SQL\)](#)  
[@@IDENTITY \(Transact-SQL\)](#)  
[IDENTITY \(Property\) \(Transact-SQL\)](#)  
[SELECT @local\\_variable \(Transact-SQL\)](#)  
[DBCC CHECKIDENT \(Transact-SQL\)](#)  
[sys.identity\\_columns \(Transact-SQL\)](#)

# SQL\_VARIANT\_PROPERTY (Transact-SQL)

9/13/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the base data type and other information about a **sql\_variant** value.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SQL_VARIANT_PROPERTY ( expression , property )
```

## Arguments

*expression*

Is an expression of type **sql\_variant**.

*property*

Contains the name of the **sql\_variant** property for which information is to be provided. *property* is **varchar(128)**, and can be any one of the following values:

VALUE	DESCRIPTION	BASE TYPE OF SQL_VARIANT RETURNED

VALUE	DESCRIPTION	BASE TYPE OF SQL_VARIANT RETURNED
<b>BaseType</b>	<p>SQL Server data type, such as:</p> <p><b>bigint</b>  <b>binary</b>  <b>char</b>  <b>date</b>  <b>datetime</b>  <b>datetime2</b>  <b>datetimeoffset</b>  <b>decimal</b>  <b>float</b>  <b>int</b>  <b>money</b>  <b>nchar</b>  <b>numeric</b>  <b>nvarchar</b>  <b>real</b>  <b>smalldatetime</b>  <b>smallint</b>  <b>smallmoney</b>  <b>time</b>  <b>tinyint</b>  <b>uniqueidentifier</b>  <b>varbinary</b>  <b>varchar</b></p>	<p><b>sysname</b></p> <p>NULL = Input is not valid.</p>

VALUE	DESCRIPTION	BASE TYPE OF SQL_VARIANT RETURNED
<b>Precision</b>	<p>Number of digits of the numeric base data type:</p> <p><b>datetime</b> = 23</p> <p><b>smalldatetime</b> = 16</p> <p><b>float</b> = 53</p> <p><b>real</b> = 24</p> <p><b>decimal</b> (p,s) and <b>numeric</b> (p,s) = p</p> <p><b>money</b> = 19</p> <p><b>smallmoney</b> = 10</p> <p><b>bigint</b> = 19</p> <p><b>int</b> = 10</p> <p><b>smallint</b> = 5</p> <p><b>tinyint</b> = 3</p> <p><b>bit</b> = 1</p> <p>All other types = 0</p>	<b>int</b> NULL = Input is not valid.
<b>Scale</b>	<p>Number of digits to the right of the decimal point of the numeric base data type:</p> <p><b>decimal</b> (p,s) and <b>numeric</b> (p,s) = s</p> <p><b>money</b> and <b>smallmoney</b> = 4</p> <p><b>datetime</b> = 3</p> <p>all other types = 0</p>	<b>int</b> NULL = Input is not valid.
<b>TotalBytes</b>	<p>Number of bytes required to hold both the metadata and data of the value. This information would be useful in checking the maximum size of data in a <b>sql_variant</b> column. If the value is larger than 900, index creation fails.</p>	<b>int</b> NULL = Input is not valid.
<b>Collation</b>	<p>Represents the collation of the particular <b>sql_variant</b> value.</p>	<b>sysname</b> NULL = Input is not valid.
<b>MaxLength</b>	<p>Maximum data type length, in bytes. For example, <b>MaxLength</b> of <b>nvarchar(50)</b> is 100, <b>MaxLength</b> of <b>int</b> is 4.</p>	<b>int</b> NULL = Input is not valid.

## Return Types

## sql\_variant

# Examples

### A. Using a sql\_variant in a table

The following example retrieves `SQL_VARIANT_PROPERTY` information about the `colA` value `46279.1` where `colB` = `1689`, given that `tableA` has `colA` that is of type `sql_variant` and `colB`.

```
CREATE TABLE tableA(colA sql_variant, colB int)
INSERT INTO tableA values ( cast (46279.1 as decimal(8,2)), 1689)
SELECT SQL_VARIANT_PROPERTY(colA,'BaseType') AS 'Base Type',
       SQL_VARIANT_PROPERTY(colA,'Precision') AS 'Precision',
       SQL_VARIANT_PROPERTY(colA,'Scale') AS 'Scale'
FROM   tableA
WHERE   colB = 1689
```

Here is the result set. Note that each of these three values is a **sql\_variant**.

Base Type	Precision	Scale
decimal	8	2

(1 row(s) affected)

### B. Using a sql\_variant as a variable

The following example retrieves `SQL_VARIANT_PROPERTY` information about a variable named `@v1`.

```
DECLARE @v1 sql_variant;
SET @v1 = 'ABC';
SELECT @v1;
SELECT SQL_VARIANT_PROPERTY(@v1, 'BaseType');
SELECT SQL_VARIANT_PROPERTY(@v1, 'MaxLength');
```

## See Also

[sql\\_variant \(Transact-SQL\)](#)

# Date and Time Data Types and Functions (Transact-SQL)

9/5/2017 • 7 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following sections in this topic provide an overview of all Transact-SQL date and time data types and functions.

- [Date and Time Data Types](#)
- [Date and Time Functions](#)
  - [Function That Get System Date and Time Values](#)
  - [Functions That Get Date and Time Parts](#)
  - [Functions That Get Date and Time Values from Their Parts](#)
  - [Functions That Get Date and Time Difference](#)
  - [Functions That Modify Date and Time Values](#)
  - [Functions That Set or Get Session Format Functions](#)
  - [Functions That Validate Date and Time Values](#)
- [Date and Time–Related Topics](#)

## Date and Time data types

The Transact-SQL date and time data types are listed in the following table:

DATA TYPE	FORMAT	RANGE	ACCURACY	STORAGE SIZE (BYTES)	USER-DEFINED FRACTIONAL SECOND PRECISION	TIME ZONE OFFSET
time	hh:mm:ss[.nnnnnn]	00:00:00.000 0000 through 23:59:59.999 9999	100 nanoseconds	3 to 5	Yes	No
date	YYYY-MM-DD	0001-01-01 through 9999-12-31	1 day	3	No	No
smalldatetime	YYYY-MM-DD hh:mm:ss	1900-01-01 through 2079-06-06	1 minute	4	No	No
datetime	YYYY-MM-DD hh:mm:ss[.nnn]	1753-01-01 through 9999-12-31	0.00333 second	8	No	No

Data Type	Format	Range	Accuracy	Storage Size (Bytes)	User-Defined Fractional Second Precision	Time Zone Offset
<code>datetime2</code>	YYYY-MM-DD hh:mm:ss[.nnnnnn]	0001-01-01 00:00:00.000 0000 through 9999-12-31 23:59:59.999 9999	100 nanoseconds	6 to 8	Yes	No
<code>datetimeoffset</code>	YYYY-MM-DD hh:mm:ss[.nnnnnn] [+ -]hh:mm	0001-01-01 00:00:00.000 0000 through 9999-12-31 23:59:59.999 9999 (in UTC)	100 nanoseconds	8 to 10	Yes	Yes

#### NOTE

The Transact-SQL `rowversion` data type is not a date or time data type. `timestamp` is a deprecated synonym for `rowversion`.

## Date and Time functions

The Transact-SQL date and time functions are listed in the following tables. For more information about determinism, see [Deterministic and Nondeterministic Functions](#).

### Functions that get system Date and Time values

All system date and time values are derived from the operating system of the computer on which the instance of SQL Server is running.

#### Higher-Precision System Date and Time Functions

SQL Server 2017 obtains the date and time values by using the `GetSystemTimeAsFileTime()` Windows API. The accuracy depends on the computer hardware and version of Windows on which the instance of SQL Server is running. The precision of this API is fixed at 100 nanoseconds. The accuracy can be determined by using the `GetSystemTimeAdjustment()` Windows API.

Function	Syntax	Return Value	Return Data Type	Determinism
<code>SYSDATETIME</code>	<code>SYSDATETIME ()</code>	Returns a <b><code>datetime2(7)</code></b> value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is not included.	<b><code>datetime2(7)</code></b>	Nondeterministic

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
SYSDATETIMEOFFSET	SYSDATETIMEOFFSET() ()	Returns a <b>datetimeoffset(7)</b> value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is included.	<b>datetimeoffset(7)</b>	Nondeterministic
SYSUTCDATETIME	SYSUTCDATETIME ()	Returns a <b>datetime2(7)</b> value that contains the date and time of the computer on which the instance of SQL Server is running. The date and time is returned as UTC time (Coordinated Universal Time).	<b>datetime2(7)</b>	Nondeterministic

#### Lower-Precision system Date and Time functions

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	Returns a <b>datetime</b> value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is not included.	<b>datetime</b>	Nondeterministic
GETDATE	GETDATE ()	Returns a <b>datetime</b> value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is not included.	<b>datetime</b>	Nondeterministic
GETUTCDATE	GETUTCDATE ()	Returns a <b>datetime</b> value that contains the date and time of the computer on which the instance of SQL Server is running. The date and time is returned as UTC time (Coordinated Universal Time).	<b>datetime</b>	Nondeterministic

## Functions that get Date and Time parts

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
DATENAME	DATENAME ( <i>datepart</i> , <i>date</i> )	Returns a character string that represents the specified <i>datepart</i> of the specified date.	nvarchar	Nondeterministic
DATEPART	DATEPART ( <i>datepart</i> , <i>date</i> )	Returns an integer that represents the specified <i>datepart</i> of the specified <i>date</i> .	int	Nondeterministic
DAY	DAY ( <i>date</i> )	Returns an integer that represents the day part of the specified <i>date</i> .	int	Deterministic
MONTH	MONTH ( <i>date</i> )	Returns an integer that represents the month part of a specified <i>date</i> .	int	Deterministic
YEAR	YEAR ( <i>date</i> )	Returns an integer that represents the year part of a specified <i>date</i> .	int	Deterministic

## Functions that get Date and Time values from their parts

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
DATEFROMPARTS	DATEFROMPARTS ( <i>year</i> , <i>month</i> , <i>day</i> )	Returns a <b>date</b> value for the specified year, month, and day.	date	Deterministic
DATETIME2FROMPARTS	DATETIME2FROMPARTS ( <i>year</i> , <i>month</i> , <i>day</i> , <i>hour</i> , <i>minute</i> , <i>seconds</i> , <i>fractions</i> , <i>precision</i> )	Returns a <b>datetime2</b> value for the specified date and time and with the specified precision.	datetime2( <i>precision</i> )	Deterministic
DATETIMEFROMPARTS	DATETIMEFROMPARTS ( <i>year</i> , <i>month</i> , <i>day</i> , <i>hour</i> , <i>minute</i> , <i>seconds</i> , <i>milliseconds</i> )	Returns a <b>datetime</b> value for the specified date and time.	datetime	Deterministic
DATETIMEOFFSETFROMPARTS	DATETIMEOFFSETFROMPARTS ( <i>year</i> , <i>month</i> , <i>day</i> , <i>hour</i> , <i>minute</i> , <i>seconds</i> , <i>fractions</i> , <i>hour_offset</i> , <i>minute_offset</i> , <i>precision</i> )	Returns a <b>datetimeoffset</b> value for the specified date and time and with the specified offsets and precision.	datetimeoffset( <i>precision</i> )	Deterministic

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
<a href="#">SMALLDATETIMEFROMPARTS</a>	SMALLDATETIMEFROMPARTS ( <i>year, month, day, hour, minute</i> )	Returns a <b>smalldatetime</b> value for the specified date and time.	<b>smalldatetime</b>	Deterministic
<a href="#">TIMEFROMPARTS</a>	TIMEFROMPARTS ( <i>hour, minute, seconds, fractions, precision</i> )	Returns a <b>time</b> value for the specified time and with the specified precision.	<b>time(</b> <i>precision</i> <b>)</b>	Deterministic

### Functions that get Date and Time difference

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
<a href="#">DATEDIFF</a>	DATEDIFF ( <i>datepart, startdate, enddate</i> )	Returns the number of date or time <i>datepart</i> boundaries that are crossed between two specified dates.	<b>int</b>	Deterministic
<a href="#">DATEDIFF_BIG</a>	DATEDIFF_BIG ( <i>datepart, startdate, enddate</i> )	Returns the number of date or time <i>datepart</i> boundaries that are crossed between two specified dates.	<b>bigint</b>	Deterministic

### Functions that modify Date and Time values

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
<a href="#">DATEADD</a>	DATEADD ( <i>datepart, number, date</i> )	Returns a new <b>datetime</b> value by adding an interval to the specified <i>datepart</i> of the specified <i>date</i> .	The data type of the <i>date</i> argument	Deterministic
<a href="#">EOMONTH</a>	EOMONTH ( <i>start_date [, month_to_add ]</i> )	Returns the last day of the month that contains the specified date, with an optional offset.	Return type is the type of <i>start_date</i> or <b>date</b> .	Deterministic
<a href="#">SWITCHOFFSET</a>	SWITCHOFFSET ( <i>DATETIMEOFFSET, time_zone</i> )	SWITCHOFFSET changes the time zone offset of a DATETIMEOFFSET value and preserves the UTC value.	<b>datetimeoffset</b> with the fractional precision of the <i>DATETIMEOFFSET</i>	Deterministic

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
<a href="#">TODATETIMEOFFSET</a>	TODATETIMEOFFSET ( <i>expression</i> , <i>time_zone</i> )	TODATETIMEOFFSET transforms a datetime2 value into a datetimeoffset value. The datetime2 value is interpreted in local time for the specified <i>time_zone</i> .	<b>datetimeoffset</b> with the fractional precision of the <b>datetime</b> argument	Deterministic

### Functions that get or set session format

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
<a href="#">@@DATEFIRST</a>	@@DATEFIRST	Returns the current value, for the session, of SET DATEFIRST.	<b>tinyint</b>	Nondeterministic
<a href="#">SET DATEFIRST</a>	SET DATEFIRST { <i>number</i>   @ <i>number_var</i> }	Sets the first day of the week to a number from 1 through 7.	Not applicable	Not applicable
<a href="#">SET DATEFORMAT</a>	SET DATEFORMAT { <i>format</i>   @ <i>format_var</i> }	Sets the order of the dateparts (month/day/year) for entering <b>datetime</b> or <b>smalldatetime</b> data.	Not applicable	Not applicable
<a href="#">@@LANGUAGE</a>	@@LANGUAGE	Returns the name of the language that is currently being used. @@LANGUAGE is not a date or time function. However, the language setting can affect the output of date functions.	Not applicable	Not applicable
<a href="#">SET LANGUAGE</a>	SET LANGUAGE { [ N ] ' <i>language</i> '   @ <i>language_var</i> }	Sets the language environment for the session and system messages. SET LANGUAGE is not a date or time function. However, the language setting affects the output of date functions.	Not applicable	Not applicable

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
<a href="#">sp_HELPLANGUAGE</a>	<code>sp_HELPLANGUAGE [ [ @language = ] 'language' ]</code>	Returns information about date formats of all supported languages. <b>sp_HELPLANGUAGE</b> is not a date or time stored procedure. However, the language setting affects the output of date functions.	Not applicable	Not applicable

### Functions that validate Date and Time values

FUNCTION	SYNTAX	RETURN VALUE	RETURN DATA TYPE	DETERMINISM
<a href="#">ISDATE</a>	<code>ISDATE ( expression )</code>	Determines whether a <b>datetime</b> or <b>smalldatetime</b> input expression is a valid date or time value.	<b>int</b>	ISDATE is deterministic only if you use it with the CONVERT function, when the CONVERT style parameter is specified, and when style is not equal to 0, 100, 9, or 109.

## Date and time-related topics

TOPIC	DESCRIPTION
<a href="#">CAST and CONVERT (Transact-SQL)</a>	Provides information about the conversion of date and time values to and from string literals and other date and time formats.
<a href="#">Write International Transact-SQL Statements</a>	Provides guidelines for portability of databases and database applications that use Transact-SQL statements from one language to another, or that support multiple languages.
<a href="#">ODBC Scalar Functions (Transact-SQL)</a>	Provides information about ODBC scalar functions that can be used in Transact-SQL statements. This includes ODBC date and time functions.
<a href="#">AT TIME ZONE (Transact-SQL)</a>	Provides time zone conversion.

## See also

[Functions](#)

[Data Types \(Transact-SQL\)](#)

# @@DATEFIRST (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns the current value, for a session, of [SET DATEFIRST](#).

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).



## Syntax

```
@@DATEFIRST
```

## Return Type

**tinyint**

## Remarks

SET DATEFIRST specifies the first day of the week. The U.S. English default is 7, Sunday.

This language setting affects the interpretation of character strings as they are converted to date values for storage in the database, and the display of date values that are stored in the database. This setting does not affect the storage format of date data. In the following example, the language is first set to `Italian`. The statement

`SELECT @@DATEFIRST;` returns `1`. The language is then set to `us_english`. The statement `SELECT @@DATEFIRST;` returns `7`.

```
SET LANGUAGE Italian;
GO
SELECT @@DATEFIRST;
GO
SET LANGUAGE us_english;
GO
SELECT @@DATEFIRST;
```

## Examples

The following example sets the first day of the week to `5` (Friday), and assumes the current day, `Today`, to be Saturday. The `SELECT` statement returns the `DATEFIRST` value and the number of the current day of the week.

```
SET DATEFIRST 5;
SELECT @@DATEFIRST AS 'First Day'
    ,DATEPART(dw, SYSDATETIME()) AS 'Today';
```

Here is the result set.

First Day	Today
5	2

## Example

Azure SQL Data Warehouse and Parallel Data Warehouse

```
SELECT @@DATEFIRST;
```

## See also

[Configuration Functions \(Transact-SQL\)](#)

# CURRENT\_TIMESTAMP (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the current database system timestamp as a **datetime** value without the database time zone offset. This value is derived from the operating system of the computer on which the instance of SQL Server is running.

## NOTE

SYSDATETIME and SYSUTCDATE have more fractional seconds precision than GETDATE and GETUTCDATE.

SYSDATETIMEOFFSET includes the system time zone offset. SYSDATETIME, SYSUTCDATE, and SYSDATETIMEOFFSET can be assigned to a variable of any of the date and time types.

This function is the ANSI SQL equivalent to [GETDATE](#).

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions](#).



## Syntax

```
CURRENT_TIMESTAMP
```

## Arguments

Takes no arguments.

## Return Type

**datetime**

## Remarks

Transact-SQL statements can refer to CURRENT\_TIMESTAMP anywhere they can refer to a **datetime** expression.

CURRENT\_TIMESTAMP is a nondeterministic function. Views and expressions that reference this column cannot be indexed.

## Examples

The following examples use the six SQL Server system functions that return current date and time to return the date, the time, or both. The values are returned in series so their fractional seconds might differ.

### A. Get the Current System Date and Time

```

SELECT SYSDATETIME()
      ,SYSDATETIMEOFFSET()
      ,SYSUTCDATETIME()
      ,CURRENT_TIMESTAMP
      ,GETDATE()
      ,GETUTCDATE();
/* Returned:
SYSDATETIME()      2007-04-30 13:10:02.0474381
SYSDATETIMEOFFSET()2007-04-30 13:10:02.0474381 -07:00
SYSUTCDATETIME()   2007-04-30 20:10:02.0474381
CURRENT_TIMESTAMP   2007-04-30 13:10:02.047
GETDATE()          2007-04-30 13:10:02.047
GETUTCDATE()       2007-04-30 20:10:02.047

```

## B. Get the Current System Date

```

SELECT CONVERT (date, SYSDATETIME())
      ,CONVERT (date, SYSDATETIMEOFFSET())
      ,CONVERT (date, SYSUTCDATETIME())
      ,CONVERT (date, CURRENT_TIMESTAMP)
      ,CONVERT (date, GETDATE())
      ,CONVERT (date, GETUTCDATE());

/* Returned
SYSDATETIME()      2007-05-03
SYSDATETIMEOFFSET()2007-05-03
SYSUTCDATETIME()   2007-05-04
CURRENT_TIMESTAMP   2007-05-03
GETDATE()          2007-05-03
GETUTCDATE()       2007-05-04
*/

```

## C. Get the Current System Time

```

SELECT CONVERT (time, SYSDATETIME())
      ,CONVERT (time, SYSDATETIMEOFFSET())
      ,CONVERT (time, SYSUTCDATETIME())
      ,CONVERT (time, CURRENT_TIMESTAMP)
      ,CONVERT (time, GETDATE())
      ,CONVERT (time, GETUTCDATE());

/* Returned
SYSDATETIME()      13:18:45.3490361
SYSDATETIMEOFFSET()13:18:45.3490361
SYSUTCDATETIME()   20:18:45.3490361
CURRENT_TIMESTAMP   13:18:45.3470000
GETDATE()          13:18:45.3470000
GETUTCDATE()       20:18:45.3470000
*/

```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

```
SELECT CURRENT_TIMESTAMP;
```

## See also

[CAST and CONVERT \(Transact-SQL\)](#)

# DATEADD (Transact-SQL)

7/31/2017 • 7 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns a specified *date* with the specified *number* interval (signed integer) added to a specified *datepart* of that *date*.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).



## Syntax

```
DATEADD (datepart , number , date )
```

## Arguments

*datepart*

Is the part of *date* to which an **integer** *number* is added. The following table lists all valid *datepart* arguments. User-defined variable equivalents are not valid.

DATEPART	ABBREVIATIONS
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw, w
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs

DATEPART	ABBREVIATIONS
<b>nanosecond</b>	<b>ns</b>

### number

Is an expression that can be resolved to an **int** that is added to a *datepart* of *date*. User-defined variables are valid. If you specify a value with a decimal fraction, the fraction is truncated and not rounded.

### date

Is an expression that can be resolved to a **time**, **date**, **smalldatetime**, **datetime**, **datetime2**, or **datetimeoffset** value. *date* can be an expression, column expression, user-defined variable, or string literal. If the expression is a string literal, it must resolve to a **datetime**. To avoid ambiguity, use four-digit years. For information about two-digit years, see [Configure the two digit year cutoff Server Configuration Option](#).

## Return types

The return data type is the data type of the *date* argument, except for string literals. The return data type for a string literal is **datetime**. An error will be raised if the string literal seconds scale is more than three positions (.nnn) or contains the time zone offset part.

## Return Value

### datepart Argument

**dayofyear**, **day**, and **weekday** return the same value.

Each *datepart* and its abbreviations return the same value.

If *datepart* is **month** and the *date* month has more days than the return month and the *date* day does not exist in the return month, the last day of the return month is returned. For example, September has 30 days; therefore, the two following statements return 2006-09-30 00:00:00.000:

```
SELECT DATEADD(month, 1, '2006-08-30');
SELECT DATEADD(month, 1, '2006-08-31');
```

### number Argument

The *number* argument cannot exceed the range of **int**. In the following statements, the argument for *number* exceeds the range of **int** by 1. The following error message is returned:

```
Msg 8115, Level 16, State 2, Line 1. Arithmetic overflow error converting expression to data type int."
```

```
SELECT DATEADD(year,2147483648, '2006-07-31');
SELECT DATEADD(year,-2147483649, '2006-07-31');
```

### date Argument

The *date* argument cannot be incremented to a value outside the range of its data type. In the following statements, the *number* value that is added to the *date* value exceeds the range of the *date* data type. The following error message is returned:

```
"Msg 517, Level 16, State 1, Line 1 Adding a value to a 'datetime' column caused overflow."
```

```
SELECT DATEADD(year,2147483647, '2006-07-31');
SELECT DATEADD(year,-2147483647, '2006-07-31');
```

## Return Values for a smalldatetime date and a second or Fractional Seconds datepart

The seconds part of a **smalldatetime** value is always 00. If *date* is **smalldatetime**, the following apply:

- If *datepart* is **second** and *number* is between -30 and +29, no addition is performed.
- If *datepart* is **second** and *number* is less than -30 or more than +29, addition is performed beginning at one minute.
- If *datepart* is **millisecond** and *number* is between -30001 and +29998, no addition is performed.
- If *datepart* is **millisecond** and *number* is less than -30001 or more than +29998, addition is performed beginning at one minute.

## Remarks

DATEADD can be used in the SELECT <list>, WHERE, HAVING, GROUP BY and ORDER BY clauses.

## Fractional seconds precision

Addition for a *datepart* of **microsecond** or **nanosecond** for *date* data types **smalldatetime**, **date**, and **datetime** is not allowed.

Milliseconds have a scale of 3 (.123), microseconds have a scale of 6 (.123456), And nanoseconds have a scale of 9 (.123456789). The **time**, **datetime2**, and **datetimeoffset** data types have a maximum scale of 7 (.1234567). If *datepart* is **nanosecond**, *number* must be 100 before the fractional seconds of *date* increase. A *number* between 1 and 49 is rounded down to 0 and a number from 50 to 99 is rounded up to 100.

The following statements add a *datepart* of **millisecond**, **microsecond**, or **nanosecond**.

```
DECLARE @datetime2 datetime2 = '2007-01-01 13:10:10.111111';
SELECT '1 millisecond', DATEADD(millisecond,1,@datetime2)
UNION ALL
SELECT '2 milliseconds', DATEADD(millisecond,2,@datetime2)
UNION ALL
SELECT '1 microsecond', DATEADD(microsecond,1,@datetime2)
UNION ALL
SELECT '2 microseconds', DATEADD(microsecond,2,@datetime2)
UNION ALL
SELECT '49 nanoseconds', DATEADD(nanosecond,49,@datetime2)
UNION ALL
SELECT '50 nanoseconds', DATEADD(nanosecond,50,@datetime2)
UNION ALL
SELECT '150 nanoseconds', DATEADD(nanosecond,150,@datetime2);
```

Here is the result set.

1 millisecond	2007-01-01 13:10:10.1121111
2 milliseconds	2007-01-01 13:10:10.1131111
1 microsecond	2007-01-01 13:10:10.1111121
2 microseconds	2007-01-01 13:10:10.1111131
49 nanoseconds	2007-01-01 13:10:10.1111111
50 nanoseconds	2007-01-01 13:10:10.1111112
150 nanoseconds	2007-01-01 13:10:10.1111113

# Time zone offset

Addition is not allowed for time zone offset.

## Examples

### A. Incrementing datepart by an interval of 1

Each of the following statements increments *datepart* by an interval of 1.

```
DECLARE @datetime2 datetime2 = '2007-01-01 13:10:10.111111';
SELECT 'year', DATEADD(year,1,@datetime2)
UNION ALL
SELECT 'quarter',DATEADD(quarter,1,@datetime2)
UNION ALL
SELECT 'month',DATEADD(month,1,@datetime2)
UNION ALL
SELECT 'dayofyear',DATEADD(dayofyear,1,@datetime2)
UNION ALL
SELECT 'day',DATEADD(day,1,@datetime2)
UNION ALL
SELECT 'week',DATEADD(week,1,@datetime2)
UNION ALL
SELECT 'weekday',DATEADD(weekday,1,@datetime2)
UNION ALL
SELECT 'hour',DATEADD(hour,1,@datetime2)
UNION ALL
SELECT 'minute',DATEADD(minute,1,@datetime2)
UNION ALL
SELECT 'second',DATEADD(second,1,@datetime2)
UNION ALL
SELECT 'millisecond',DATEADD(millisecond,1,@datetime2)
UNION ALL
SELECT 'microsecond',DATEADD(microsecond,1,@datetime2)
UNION ALL
SELECT 'nanosecond',DATEADD(nanosecond,1,@datetime2);
```

Here is the result set.

Year	2008-01-01 13:10:10.111111
quarter	2007-04-01 13:10:10.111111
month	2007-02-01 13:10:10.111111
dayofyear	2007-01-02 13:10:10.111111
day	2007-01-02 13:10:10.111111
week	2007-01-08 13:10:10.111111
weekday	2007-01-02 13:10:10.111111
hour	2007-01-01 14:10:10.111111
minute	2007-01-01 13:11:10.111111
second	2007-01-01 13:10:11.111111
millisecond	2007-01-01 13:10:10.1121111
microsecond	2007-01-01 13:10:10.1111121
nanosecond	2007-01-01 13:10:10.1111111

### B. Incrementing more than one level of datepart in one statement

Each of the following statements increments *datepart* by a *number* large enough to also increment the next higher *datepart* of *date*.

```

DECLARE @datetime2 datetime2;
SET @datetime2 = '2007-01-01 01:01:01.111111';
--Statement                                         Result
-----
SELECT DATEADD(quarter,4,@datetime2);      --2008-01-01 01:01:01.110
SELECT DATEADD(month,13,@datetime2);        --2008-02-01 01:01:01.110
SELECT DATEADD(dayofyear,365,@datetime2);   --2008-01-01 01:01:01.110
SELECT DATEADD(day,365,@datetime2);         --2008-01-01 01:01:01.110
SELECT DATEADD(week,5,@datetime2);          --2007-02-05 01:01:01.110
SELECT DATEADD(weekday,31,@datetime2);       --2007-02-01 01:01:01.110
SELECT DATEADD(hour,23,@datetime2);         --2007-01-02 00:01:01.110
SELECT DATEADD(minute,59,@datetime2);        --2007-01-01 02:00:01.110
SELECT DATEADD(second,59,@datetime2);        --2007-01-01 01:02:00.110
SELECT DATEADD(millisecond,1,@datetime2);    --2007-01-01 01:01:01.110

```

## C. Using expressions as arguments for the number and date parameters

The following examples use different types of expressions as arguments for the *number* and *date* parameters. The examples use the AdventureWorks database.

### Specifying a column as date

The following example adds `2` days to each value in the `OrderDate` column to derive a new column named `PromisedShipDate`.

```

SELECT SalesOrderID
      ,OrderDate
      ,DATEADD(day,2,OrderDate) AS PromisedShipDate
  FROM Sales.SalesOrderHeader;

```

Here is a partial result set.

SalesOrderID	OrderDate	PromisedShipDate
43659	2005-07-01 00:00:00.000	2005-07-03 00:00:00.000
43660	2005-07-01 00:00:00.000	2005-07-03 00:00:00.000
43661	2005-07-01 00:00:00.000	2005-07-03 00:00:00.000
...		
43702	2005-07-02 00:00:00.000	2005-07-04 00:00:00.000
43703	2005-07-02 00:00:00.000	2005-07-04 00:00:00.000
43704	2005-07-02 00:00:00.000	2005-07-04 00:00:00.000
43705	2005-07-02 00:00:00.000	2005-07-04 00:00:00.000
43706	2005-07-03 00:00:00.000	2005-07-05 00:00:00.000
...		
43711	2005-07-04 00:00:00.000	2005-07-06 00:00:00.000
43712	2005-07-04 00:00:00.000	2005-07-06 00:00:00.000
...		
43740	2005-07-11 00:00:00.000	2005-07-13 00:00:00.000
43741	2005-07-12 00:00:00.000	2005-07-14 00:00:00.000

### Specifying user-defined variables as number and date

The following example specifies user-defined variables as arguments for *number* and *date*.

```

DECLARE @days int = 365,
        @datetime datetime = '2000-01-01 01:01:01.111' /* 2000 was a leap year */;
SELECT DATEADD(day, @days, @datetime);

```

Here is the result set.

```
-----  
2000-12-31 01:01:01.110
```

```
(1 row(s) affected)
```

### Specifying scalar system function as date

The following example specifies `SYSDATETIME()` for *date*.

```
SELECT DATEADD(month, 1, SYSDATETIME());
```

Here is the result set.

```
-----  
2013-02-06 14:29:59.6727944
```

```
(1 row(s) affected)
```

### Specifying scalar subqueries and scalar functions as number and date

The following example uses scalar subqueries, `MAX(ModifiedDate)`, as arguments for *number* and *date*.

`(SELECT TOP 1 BusinessEntityID FROM Person.Person)` is an artificial argument for the *number* parameter to show how to select a *number* argument from a value list.

```
SELECT DATEADD(month,(SELECT TOP 1 BusinessEntityID FROM Person.Person),  
(SELECT MAX(ModifiedDate) FROM Person.Person));
```

### Specifying numeric expressions and scalar system functions as number and date

The following example uses a numeric expression `-(10/2)`, unary operators `( - )`, an arithmetic operator `( / )`, and scalar system functions `( SYSDATETIME )` as arguments for *number* and *date*.

```
SELECT DATEADD(month,-(10/2), SYSDATETIME());
```

### Specifying ranking functions as number

The following example uses a ranking function as arguments for *number*.

```
SELECT p.FirstName, p.LastName  
,DATEADD(day,ROW_NUMBER() OVER (ORDER BY  
a.PostalCode),SYSDATETIME()) AS 'Row Number'  
FROM Sales.SalesPerson AS s  
INNER JOIN Person.Person AS p  
ON s.BusinessEntityID = p.BusinessEntityID  
INNER JOIN Person.Address AS a  
ON a.AddressID = p.BusinessEntityID  
WHERE TerritoryID IS NOT NULL  
AND SalesYTD <> 0;
```

### Specifying an aggregate window function as number

The following example uses an aggregate window function as an argument for *number*.

```
SELECT SalesOrderID, ProductID, OrderQty  
,DATEADD(day,SUM(OrderQty)  
OVER(PARTITION BY SalesOrderID),SYSDATETIME()) AS 'Total'  
FROM Sales.SalesOrderDetail  
WHERE SalesOrderID IN(43659,43664);  
GO
```

## See also

[CAST and CONVERT \(Transact-SQL\)](#)

# DATEDIFF (Transact-SQL)

9/27/2017 • 6 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the count (signed integer) of the specified *datepart* boundaries crossed between the specified *startdate* and *enddate*.

For larger differences, see [DATEDIFF\\_BIG \(Transact-SQL\)](#). For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).



## Syntax

```
DATEDIFF ( datepart , startdate , enddate )
```

## Arguments

*datepart*

Is the part of *startdate* and *enddate* that specifies the type of boundary crossed. The following table lists all valid *datepart* arguments. User-defined variable equivalents are not valid.

DATEPART	ABBREVIATIONS
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns

#### *startdate*

Is an expression that can be resolved to a **time**, **date**, **smalldatetime**, **datetime**, **datetime2**, or **datetimeoffset** value. *date* can be an expression, column expression, user-defined variable or string literal. *startdate* is subtracted from *enddate*.

To avoid ambiguity, use four-digit years. For information about two digits years, see [Configure the two digit year cutoff Server Configuration Option](#).

#### *enddate*

See *startdate*.

## Return Type

### **int**

## Return Value

- Each *datepart* and its abbreviations return the same value.

If the return value is out of range for **int** (-2,147,483,648 to +2,147,483,647), an error is returned. For **millisecond**, the maximum difference between *startdate* and *enddate* is 24 days, 20 hours, 31 minutes and 23.647 seconds. For **second**, the maximum difference is 68 years.

If *startdate* and *enddate* are both assigned only a time value and the *datepart* is not a time *datepart*, 0 is returned.

A time zone offset component of *startdate* or *enddate* is not used in calculating the return value.

Because **smalldatetime** is accurate only to the minute, when a **smalldatetime** value is used for *startdate* or *enddate*, seconds and milliseconds are always set to 0 in the return value.

If only a time value is assigned to a variable of a date data type, the value of the missing date part is set to the default value: 1900-01-01. If only a date value is assigned to a variable of a time or date data type, the value of the missing time part is set to the default value: 00:00:00. If either *startdate* or *enddate* have only a time part and the other only a date part, the missing time and date parts are set to the default values.

If *startdate* and *enddate* are of different date data types and one has more time parts or fractional seconds precision than the other, the missing parts of the other are set to 0.

## datepart Boundaries

The following statements have the same *startdate* and the same *enddate*. Those dates are adjacent and differ in time by .0000001 second. The difference between the *startdate* and *enddate* in each statement crosses one calendar or time boundary of its *datepart*. Each statement returns 1. If different years are used for this example and if both *startdate* and *enddate* are in the same calendar week, the return value for **week** would be 0.

```
SELECT DATEDIFF(year, '2005-12-31 23:59:59.999999'  
    , '2006-01-01 00:00:00.000000');  
  
SELECT DATEDIFF(quarter, '2005-12-31 23:59:59.999999'  
    , '2006-01-01 00:00:00.000000');  
  
SELECT DATEDIFF(month, '2005-12-31 23:59:59.999999'  
    , '2006-01-01 00:00:00.000000');  
  
SELECT DATEDIFF(dayofyear, '2005-12-31 23:59:59.999999'
```

```

, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF(day, '2005-12-31 23:59:59.999999'
, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF(week, '2005-12-31 23:59:59.999999'
, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF(hour, '2005-12-31 23:59:59.999999'
, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF(minute, '2005-12-31 23:59:59.999999'
, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF(second, '2005-12-31 23:59:59.999999'
, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF(millisecond, '2005-12-31 23:59:59.999999'
, '2006-01-01 00:00:00.0000000');

```

## Remarks

DATEDIFF can be used in the select list, WHERE, HAVING, GROUP BY and ORDER BY clauses.

DATEDIFF implicitly casts string literals as a **datetime2** type. This means that DATEDIFF does not support the format YDM when the date is passed as a string. You must explicitly cast the string to a **datetime** or **smalldatetime** type to use the YDM format.

Specifying SET DATEFIRST has no effect on DATEDIFF. DATEDIFF always uses Sunday as the first day of the week to ensure the function is deterministic.

## Examples

The following examples use different types of expressions as arguments for the *startdate* and *enddate* parameters.

### A. Specifying columns for startdate and enddate

The following example calculates the number of day boundaries that are crossed between dates in two columns in a table.

```

CREATE TABLE dbo.Duration
(
    startDate datetime2
    ,endDate datetime2
);
INSERT INTO dbo.Duration(startDate,endDate)
    VALUES('2007-05-06 12:10:09','2007-05-07 12:10:09');
SELECT DATEDIFF(day,startDate,endDate) AS 'Duration'
FROM dbo.Duration;
-- Returns: 1

```

### B. Specifying user-defined variables for startdate and enddate

The following example uses user-defined variables as arguments for *startdate* and *enddate*.

```
DECLARE @startdate datetime2 = '2007-05-05 12:10:09.3312722';
DECLARE @enddate datetime2 = '2007-05-04 12:10:09.3312722';
SELECT DATEDIFF(day, @startdate, @enddate);
```

### C. Specifying scalar system functions for startdate and enddate

The following example uses scalar system functions as arguments for *startdate* and *enddate*.

```
SELECT DATEDIFF(millisecond, GETDATE(), SYSDATETIME());
```

### D. Specifying scalar subqueries and scalar functions for startdate and enddate

The following example uses scalar subqueries and scalar functions as arguments for *startdate* and *enddate*.

```
USE AdventureWorks2012;
GO
SELECT DATEDIFF(day,(SELECT MIN(OrderDate) FROM Sales.SalesOrderHeader),
    (SELECT MAX(OrderDate) FROM Sales.SalesOrderHeader));
```

### E. Specifying constants for startdate and enddate

The following example uses character constants as arguments for *startdate* and *enddate*.

```
SELECT DATEDIFF(day, '2007-05-07 09:53:01.0376635'
    , '2007-05-08 09:53:01.0376635');
```

### F. Specifying numeric expressions and scalar system functions for enddate

The following example uses a numeric expression, `(GETDATE ()+ 1)`, and scalar system functions, `GETDATE` and `SYSDATETIME`, as arguments for *enddate*.

```
USE AdventureWorks2012;
GO
SELECT DATEDIFF(day, '2007-05-07 09:53:01.0376635', GETDATE()+ 1)
    AS NumberOfDays
FROM Sales.SalesOrderHeader;
GO
USE AdventureWorks2012;
GO
SELECT DATEDIFF(day, '2007-05-07 09:53:01.0376635', DATEADD(day,1,SYSDATETIME())) AS NumberOfDays
FROM Sales.SalesOrderHeader;
GO
```

### G. Specifying ranking functions for startdate

The following example uses a ranking function as an argument for *startdate*.

```

USE AdventureWorks2012;
GO
SELECT p.FirstName, p.LastName
,DATEDIFF(day,ROW_NUMBER() OVER (ORDER BY
a.PostalCode),SYSDATETIME()) AS 'Row Number'
FROM Sales.SalesPerson s
INNER JOIN Person.Person p
ON s.BusinessEntityID = p.BusinessEntityID
INNER JOIN Person.Address a
ON a.AddressID = p.BusinessEntityID
WHERE TerritoryID IS NOT NULL
AND SalesYTD <> 0;

```

## H. Specifying an aggregate window function for startdate

The following example uses an aggregate window function as an argument for *startdate*.

```

USE AdventureWorks2012;
GO
SELECT soh.SalesOrderID, sod.ProductID, sod.OrderQty,soh.OrderDate
,DATEDIFF(day,MIN(soh.OrderDate)
OVER(PARTITION BY soh.SalesOrderID),SYSDATETIME() ) AS 'Total'
FROM Sales.SalesOrderDetail sod
INNER JOIN Sales.SalesOrderHeader soh
ON sod.SalesOrderID = soh.SalesOrderID
WHERE soh.SalesOrderID IN(43659,58918);
GO

```

# Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following examples use different types of expressions as arguments for the *startdate* and *enddate* parameters.

## I. Specifying columns for startdate and enddate

The following example calculates the number of day boundaries that are crossed between dates in two columns in a table.

```

CREATE TABLE dbo.Duration (
    startDate datetime2
    ,endDate datetime2
    );
INSERT INTO dbo.Duration(startDate,endDate)
VALUES('2007-05-06 12:10:09','2007-05-07 12:10:09');
SELECT TOP(1) DATEDIFF(day,startDate,endDate) AS Duration
FROM dbo.Duration;
-- Returns: 1

```

## J. Specifying scalar subqueries and scalar functions for startdate and enddate

The following example uses scalar subqueries and scalar functions as arguments for *startdate* and *enddate*.

```

-- Uses AdventureWorks

SELECT TOP(1) DATEDIFF(day,(SELECT MIN(HireDate) FROM dbo.DimEmployee),
(SELECT MAX(HireDate) FROM dbo.DimEmployee))
FROM dbo.DimEmployee;

```

## K. Specifying constants for startdate and enddate

The following example uses character constants as arguments for *startdate* and *enddate*.

```
-- Uses AdventureWorks

SELECT TOP(1) DATEDIFF(day, '2007-05-07 09:53:01.0376635'
    , '2007-05-08 09:53:01.0376635') FROM DimCustomer;
```

## L. Specifying ranking functions for startdate

The following example uses a ranking function as an argument for *startdate*.

```
-- Uses AdventureWorks

SELECT FirstName, LastName
,DATEDIFF(day,ROW_NUMBER() OVER (ORDER BY
    DepartmentName),SYSDATETIME()) AS RowNumber
FROM dbo.DimEmployee;
```

## M. Specifying an aggregate window function for startdate

The following example uses an aggregate window function as an argument for *startdate*.

```
-- Uses AdventureWorks

SELECT FirstName, LastName, DepartmentName
,DATEDIFF(year,MAX(HireDate)
    OVER (PARTITION BY DepartmentName),SYSDATETIME()) AS SomeValue
FROM dbo.DimEmployee
```

## See also

[DATEDIFF\\_BIG \(Transact-SQL\)](#)  
[CAST and CONVERT \(Transact-SQL\)](#)

# DATEDIFF\_BIG (Transact-SQL)

7/31/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2016) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Returns the count (signed big integer) of the specified *datepart* boundaries crossed between the specified *startdate* and *enddate*.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
DATEDIFF_BIG ( datepart , startdate , enddate )
```

## Arguments

### *datepart*

Is the part of *startdate* and *enddate* that specifies the type of boundary crossed. The following table lists all valid *datepart* arguments. User-defined variable equivalents are not valid.

DATEPART	ABBREVIATIONS
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs

DATEPART	ABBREVIATIONS
<b>nanosecond</b>	<b>ns</b>

#### *startdate*

Is an expression that can be resolved to a **time**, **date**, **smalldatetime**, **datetime**, **datetime2**, or **datetimeoffset** value. **date** can be an expression, column expression, user-defined variable or string literal. *startdate* is subtracted from *enddate*.

To avoid ambiguity, use four-digit years. For information about two digits years, see [Configure the two digit year cutoff Server Configuration Option](#).

#### *enddate*

See *startdate*.

## Return Type

Signed

**bigint**

## Return Value

Returns count (signed bigint) of the specified datepart boundaries crossed between the specified *startdate* and *enddate*.

- Each *datepart* and its abbreviations return the same value.

If the return value is out of range for **bigint** (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807), an error is returned. For **millisecond**, the maximum difference between *startdate* and *enddate* is 24 days, 20 hours, 31 minutes and 23.647 seconds. For **second**, the maximum difference is 68 years.

If *startdate* and *enddate* are both assigned only a time value and the *datepart* is not a time *datepart*, 0 is returned.

A time zone offset component of *startdate* or *enddate* is not used in calculating the return value.

Because **smalldatetime** is accurate only to the minute, when a **smalldatetime** value is used for *startdate* or *enddate*, seconds and milliseconds are always set to 0 in the return value.

If only a time value is assigned to a variable of a date data type, the value of the missing date part is set to the default value: 1900-01-01. If only a date value is assigned to a variable of a time or date data type, the value of the missing time part is set to the default value: 00:00:00. If either *startdate* or *enddate* have only a time part and the other only a date part, the missing time and date parts are set to the default values.

If *startdate* and *enddate* are of different date data types and one has more time parts or fractional seconds precision than the other, the missing parts of the other are set to 0.

## datepart boundaries

The following statements have the same *startdate* and the same *enddate*. Those dates are adjacent and differ in time by .0000001 second. The difference between the *startdate* and *enddate* in each statement crosses one calendar or time boundary of its *datepart*. Each statement returns 1. If different years are used for this example and if both *startdate* and *enddate* are in the same calendar week, the return value for **week** would be 0.

```
SELECT DATEDIFF_BIG(year, '2005-12-31 23:59:59.999999'
                      , '2006-01-01 00:00:00.000000');
```

```
SELECT DATEDIFF_BIG(quarter, '2005-12-31 23:59:59.999999'
```

```
, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF_BIG(month, '2005-12-31 23:59:59.9999999'

, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF_BIG(dayofyear, '2005-12-31 23:59:59.9999999'

, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF_BIG(day, '2005-12-31 23:59:59.9999999'

, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF_BIG(week, '2005-12-31 23:59:59.9999999'

, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF_BIG(hour, '2005-12-31 23:59:59.9999999'

, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF_BIG(minute, '2005-12-31 23:59:59.9999999'

, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF_BIG(second, '2005-12-31 23:59:59.9999999'

, '2006-01-01 00:00:00.0000000');

SELECT DATEDIFF_BIG(millisecond, '2005-12-31 23:59:59.9999999'

, '2006-01-01 00:00:00.0000000');
```

## Remarks

DATEDIFF\_BIG can be used in the select list, WHERE, HAVING, GROUP BY and ORDER BY clauses.

DATEDIFF\_BIG implicitly casts string literals as a **datetime2** type. This means that DATEDIFF\_BIG does not support the format YDM when the date is passed as a string. You must explicitly cast the string to a **datetime** or **smalldatetime** type to use the YDM format.

Specifying SET DATEFIRST has no effect on DATEDIFF\_BIG. DATEDIFF\_BIG always uses Sunday as the first day of the week to ensure the function is deterministic.

## Examples

The following examples use different types of expressions as arguments for the *startdate* and *enddate* parameters.

### Specifying columns for startdate and enddate

The following example calculates the number of day boundaries that are crossed between dates in two columns in a table.

```
CREATE TABLE dbo.Duration
(
    startDate datetime2
    ,endDate datetime2
);
INSERT INTO dbo.Duration(startDate,endDate)
VALUES('2007-05-06 12:10:09','2007-05-07 12:10:09');
SELECT DATEDIFF_BIG(day,startDate,endDate) AS 'Duration'
FROM dbo.Duration;
-- Returns: 1
```

For many additional examples, see the closely related examples in [DATEDIFF \(Transact-SQL\)](#).

## See also

[CAST and CONVERT \(Transact-SQL\)](#)

[DATEDIFF \(Transact-SQL\)](#)

# DATEFROMPARTS (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a **date** value for the specified year, month, and day.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DATEFROMPARTS ( year, month, day )
```

## Arguments

*year*

Integer expression specifying a year.

*month*

Integer expression specifying a month, from 1 to 12.

*day*

Integer expression specifying a day.

## Return types

**date**

## Remarks

**DATEFROMPARTS** returns a **date** value with the date portion set to the specified year, month and day, and the time portion set to the default. If the arguments are not valid, then an error is raised. If required arguments are null, then null is returned.

This function is capable of being remoted to SQL Server 2012 servers and above. It will not be remoted to servers with a version below SQL Server 2012.

## Examples

The following example demonstrates the **DATEFROMPARTS** function.

```
SELECT DATEFROMPARTS ( 2010, 12, 31 ) AS Result;
```

Here is the result set.

```
Result
-----
2010-12-31

(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example demonstrates the **DATEFROMPARTS** function.

```
SELECT DATEFROMPARTS ( 2010, 12, 31 ) AS Result;
```

Here is the result set.

```
Result
-----
2010-12-31

(1 row(s) affected)
```

## See also

[date \(Transact-SQL\)](#)

# DATENAME (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a character string that represents the specified *datepart* of the specified *date*.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).



## Syntax

```
DATENAME ( datepart , date )
```

## Arguments

### *datepart*

Is the part of the *date* to return. The following table lists all valid *datepart* arguments. User-defined variable equivalents are not valid.

DATEPART	ABBREVIATIONS
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw, w
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs

DATEPART	ABBREVIATIONS
<b>nanosecond</b>	<b>ns</b>
<b>TZoffset</b>	<b>tz</b>
<b>ISO_WEEK</b>	<b>ISOWK, ISOWW</b>

### date

Is an expression that can be resolved to a **time**, **date**, **smalldatetime**, **datetime**, **datetime2**, or **datetimeoffset** value. **date** can be an expression, column expression, user-defined variable, or string literal.

To avoid ambiguity, use four-digit years. For information about two-digit years, see [Configure the two digit year cutoff Server Configuration Option](#).

## Return Type

**nvarchar**

## Return Value

- Each *datepart* and its abbreviations return the same value.

The return value depends on the language environment set by using **SET LANGUAGE** and by the [Configure the default language Server Configuration Option](#) of the login. The return value is dependant on **SET DATEFORMAT** if *date* is a string literal of some formats. **SET DATEFORMAT** does not affect the return value when the date is a column expression of a date or time data type.

When the *date* parameter has a **date** data type argument, the return value depends on the setting specified by using **SET DATEFIRST**.

## TZoffset datepart Argument

If *datepart* argument is **TZoffset (tz)** and the *date* argument has no time zone offset, 0 is returned.

## smalldatetime date Argument

When *date* is **smalldatetime**, seconds are returned as 00.

## Default Returned for a datepart That Is Not in the date Argument

If the data type of the *date* argument does not have the specified *datepart*, the default for that *datepart* will be returned only when a literal is specified for *date*.

For example, the default year-month-day for any **date** data type is 1900-01-01. The following statement has date part arguments for *datepart*, a time argument for *date*, and returns `1900, January, 1, 1, Monday`.

```
SELECT DATENAME(year, '12:10:30.123')
      ,DATENAME(month, '12:10:30.123')
      ,DATENAME(day, '12:10:30.123')
      ,DATENAME(dayofyear, '12:10:30.123')
      ,DATENAME(weekday, '12:10:30.123');
```

If *date* is specified as a variable or table column and the data type for that variable or column does not have the specified *datepart*, error 9810 is returned. The following code example fails because the date part year is not a valid for the **time** data type that is declared for the variable `@t`.

```
DECLARE @t time = '12:10:30.123';
SELECT DATENAME(year, @t);
```

## Remarks

DATENAME can be used in the select list, WHERE, HAVING, GROUP BY, and ORDER BY clauses.

In SQL Server 2017, DATENAME implicitly casts string literals as a **datetime2** type. This means that DATENAME does not support the format YDM when the date is passed as a string. You must explicitly cast the string to a **datetime** or **smalldatetime** type to use the YDM format.

## Examples

The following example returns the date parts for the specified date.

```
SELECT DATENAME(datepart,'2007-10-30 12:15:32.1234567 +05:10');
```

Here is the result set.

DATEPART	RETURN VALUE
year, yyyy, yy	2007
quarter, qq, q	4
month, mm, m	October
dayofyear, dy, y	303
day, dd, d	30
week, wk, ww	44
weekday, dw	Tuesday
hour, hh	12
minute, n	15
second, ss, s	32
millisecond, ms	123
microsecond, mcs	123456
nanosecond, ns	123456700
TZoffset, tz	310
ISO_WEEK, ISOWK, ISOWW	44

The following example returns the date parts for the specified date.

```
SELECT DATENAME(datepart,'2007-10-30 12:15:32.1234567 +05:10');
```

Here is the result set.

DATEPART	RETURN VALUE
<b>year, yyyy, yy</b>	2007
<b>quarter, qq, q</b>	4
<b>month, mm, m</b>	October
<b>dayofyear, dy, y</b>	303
<b>day, dd, d</b>	30
<b>week, wk, ww</b>	44
<b>weekday, dw</b>	Tuesday
<b>hour, hh</b>	12
<b>minute, n</b>	15
<b>second, ss, s</b>	32
<b>millisecond, ms</b>	123
<b>microsecond, mcs</b>	123456
<b>nanosecond, ns</b>	123456700
<b>TZoffset, tz</b>	310
<b>ISO_WEEK, ISOWK, ISOOWW</b>	44

## See also

[CAST and CONVERT \(Transact-SQL\)](#)

# DATEPART (Transact-SQL)

9/27/2017 • 5 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns an integer that represents the specified *datepart* of the specified *date*.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
DATEPART ( datepart , date )
```

## Arguments

*datepart*

Is the part of *date* (a date or time value) for which an **integer** will be returned. The following table lists all valid *datepart* arguments. User-defined variable equivalents are not valid.

<b>DATEPART</b>	<b>ABBREVIATIONS</b>
<b>year</b>	<b>yy, yyyy</b>
<b>quarter</b>	<b>qq, q</b>
<b>month</b>	<b>mm, m</b>
<b>dayofyear</b>	<b>dy, y</b>
<b>day</b>	<b>dd, d</b>
<b>week</b>	<b>wk, ww</b>
<b>weekday</b>	<b>dw</b>
<b>hour</b>	<b>hh</b>
<b>minute</b>	<b>mi, n</b>
<b>second</b>	<b>ss, s</b>
<b>millisecond</b>	<b>ms</b>
<b>microsecond</b>	<b>mcs</b>

DATEPART	ABBREVIATIONS
<b>nanosecond</b>	<b>ns</b>
<b>TZoffset</b>	<b>tz</b>
<b>ISO_WEEK</b>	<b>isowk, isoww</b>

### *date*

Is an expression that can be resolved to a **time**, **date**, **smalldatetime**, **datetime**, **datetime2**, or **datetimeoffset** value. *date* can be an expression, column expression, user-defined variable, or string literal.

To avoid ambiguity, use four-digit years. For information about two digits years, see [Configure the two digit year cutoff Server Configuration Option](#).

## Return Type

**int**

## Return Value

Each *datepart* and its abbreviations return the same value.

The return value depends on the language environment set by using **SET LANGUAGE** and by the [Configure the default language Server Configuration Option](#) of the login. If *date* is a string literal for some formats, the return value depends on the format specified by using **SET DATEFORMAT**. **SET DATEFORMAT** does not affect the return value when the date is a column expression of a date or time data type.

The following table lists all *datepart* arguments with corresponding return values for the statement

```
SELECT DATEPART(datepart, '2007-10-30 12:15:32.1234567 +05:10') . The data type of the date argument is
datetimeoffset(7). The nanoseconddatepart return value has a scale of 9 (.123456700) and the last two
positions are always 00.
```

DATEPART	RETURN VALUE
<b>year, yyyy, yy</b>	2007
<b>quarter, qq, q</b>	4
<b>month, mm, m</b>	10
<b>dayofyear, dy, y</b>	303
<b>day, dd, d</b>	30
<b>week, wk, ww</b>	45
<b>weekday, dw</b>	1
<b>hour, hh</b>	12
<b>minute, n</b>	15
<b>second, ss, s</b>	32

DATEPART	RETURN VALUE
millisecond, ms	123
microsecond, mcs	123456
nanosecond, ns	123456700
TZoffset, tz	310

## Week and weekday datepart arguments

When *datepart* is **week** (**wk**, **ww**) or **weekday** (**dw**), the return value depends on the value that is set by using [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 following table lists the return value for **week** and **weekday** *datepart* for '2007-04-21' for each SET DATEFIRST argument. January 1 is a Monday in the year 2007. April 21 is a Saturday in the year 2007. SET DATEFIRST 7, Sunday, is the default for U.S. English.

SET DATEFIRST	WEEK	WEEKDAY
ARGUMENT	RETURNED	RETURNED
1	16	6
2	17	5
3	17	4
4	17	3
5	17	2
6	17	1
7	16	7

## year, month, and day datepart Arguments

The values that are returned for DATEPART (**year**, *date*), DATEPART (**month**, *date*), and DATEPART (**day**, *date*) are the same as those returned by the functions [YEAR](#), [MONTH](#), and [DAY](#), respectively.

## ISO\_WEEK datepart

ISO 8601 includes the ISO week-date system, a numbering system for weeks. Each week is associated with the year in which Thursday occurs. For example, week 1 of 2004 (2004W01) ran from Monday 29 December 2003 to Sunday, 4 January 2004. The highest week number in a year might be 52 or 53. This style of numbering is typically used in European countries/regions, but rare elsewhere.

The numbering system in different countries/regions might not comply with the ISO standard. There are at least six possibilities as shown in the following table

FIRST DAY OF WEEK	FIRST WEEK OF YEAR CONTAINS	WEEKS ASSIGNED TWO TIMES	USED BY/IN
Sunday	1 January, First Saturday, 1–7 days of year	Yes	United States
Monday	1 January, First Sunday, 1–7 days of year	Yes	Most of Europe and the United Kingdom
Monday	4 January, First Thursday, 4–7 days of year	No	ISO 8601, Norway, and Sweden
Monday	7 January, First Monday, 7 days of year	No	
Wednesday	1 January, First Tuesday, 1–7 days of year	Yes	
Saturday	1 January, First Friday, 1–7 days of year	Yes	

## TZoffset

The **TZoffset** (**tz**) is returned as the number of minutes (signed). The following statement returns a time zone offset of 310 minutes.

```
SELECT DATEPART (TZoffset, '2007-05-10 00:00:01.1234567 +05:10');
```

The TZoffset value is rendered as follows:

- For datetimeoffset and datetime2, TZoffset returns the time offset in minutes, where the offset for datetime2 is always 0 minutes.
- For data types that can be implicitly converted to datetimeoffset or datetime2, with the exception of the other date/time data types, it returns the time offset in minutes.
- Parameters of all other types result in an error.

## smalldatetime date Argument

When *date* is **smalldatetime**, seconds are returned as 00.

## Default Returned for a datepart That Is Not in a date Argument

If the data type of the *date* argument does not have the specified *datepart*, the default for that *datepart* will be returned only when a literal is specified for *date*.

For example, the default year-month-day for any **date** data type is 1900-01-01. The following statement has date part arguments for *datepart*, a time argument for *date*, and returns `1900, 1, 1, 1, 2`.

```
SELECT DATEPART(year, '12:10:30.123')
      ,DATEPART(month, '12:10:30.123')
      ,DATEPART(day, '12:10:30.123')
      ,DATEPART(dayofyear, '12:10:30.123')
      ,DATEPART(weekday, '12:10:30.123');
```

If *date* is specified as a variable or table column and the data type for that variable or column does not have the specified *datepart*, error 9810 is returned. The following code example fails because the date part year is not a valid for the **time** data type that is declared for the variable *@t*.

```
DECLARE @t time = '12:10:30.123';
SELECT DATEPART(year, @t);
```

## Fractional seconds

Fractional seconds are returned as shown in the following statements:

```
SELECT DATEPART(millisecond, '00:00:01.1234567'); -- Returns 123
SELECT DATEPART(microsecond, '00:00:01.1234567'); -- Returns 123456
SELECT DATEPART(nanosecond, '00:00:01.1234567'); -- Returns 123456700
```

## Remarks

DATEPART can be used in the select list, WHERE, HAVING, GROUP BY and ORDER BY clauses.

In SQL Server 2017, DATEPART implicitly casts string literals as a **datetime2** type. This means that DATEPART does not support the format YDM when the date is passed as a string. You must explicitly cast the string to a **datetime** or **smalldatetime** type to use the YDM format.

## Examples

The following example returns the base year. The base year is useful for date calculations. In the example, the date is specified as a number. Notice that SQL Server interprets 0 as January 1, 1900.

```
SELECT DATEPART(year, 0), DATEPART(month, 0), DATEPART(day, 0);
-- Returns: 1900    1    1 */
```

The following example returns the day part of the date `12/20/1974`.

```
-- Uses AdventureWorks

SELECT TOP(1) DATEPART (day,'12/20/1974') FROM dbo.DimCustomer;
```

Here is the result set.

-----  
20

The following example returns the year part of the date 12/20/1974 .

```
-- Uses AdventureWorks  
  
SELECT TOP(1) DATEPART (year,'12/20/1974') FROM dbo.DimCustomer;
```

Here is the result set.

-----  
1974

## See also

[CAST and CONVERT \(Transact-SQL\)](#)

# DATETIME2FROMPARTS (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a **datetime2** value for the specified date and time and with the specified precision.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DATETIME2FROMPARTS ( year, month, day, hour, minute, seconds, fractions, precision )
```

## Arguments

*year*

Integer expression specifying a year.

*month*

Integer expression specifying a month.

*day*

Integer expression specifying a day.

*hour*

Integer expression specifying hours.

*minute* Integer expression specifying minutes.

*seconds*

Integer expression specifying seconds.

*fractions*

Integer expression specifying fractions.

*precision*

Integer literal specifying the precision of the **datetime2** value to be returned.

## Return types

**datetime2(** *precision* **)**

## Remarks

**DATETIME2FROMPARTS** returns a fully initialized **datetime2** value. If the arguments are not valid, an error is raised. If required arguments are null, then null is returned. However, if the *precision* argument is null, then an error is raised.

The *fractions* argument depends on the *precision* argument. For example, if *precision* is 7, then each fraction represents 100 nanoseconds; if *precision* is 3, then each fraction represents a millisecond. If the value of *precision* is zero, then the value of *fractions* must also be zero; otherwise, an error is raised.

This function is capable of being remoted to SQL Server 2017 servers and above. It will not be remoted to servers that have a version below SQL Server 2017.

## Examples

### A. Simple example without fractions of a second

```
SELECT DATETIME2FROMPARTS ( 2010, 12, 31, 23, 59, 59, 0, 0 ) AS Result;
```

Here is the result set.

```
Result
-----
2010-12-31 23:59:59.0000000

(1 row(s) affected)
```

### B. Example with fractions of a second

The following example demonstrates the use of the *fractions* and *precision* parameters:

1. When *fractions* has a value of 5 and *precision* has a value of 1, then the value of *fractions* represents 5/10 of a second.
2. When *fractions* has a value of 50 and *precision* has a value of 2, then the value of *fractions* represents 50/100 of a second.
3. When *fractions* has a value of 500 and *precision* has a value of 3, then the value of *fractions* represents 500/1000 of a second.

```
SELECT DATETIME2FROMPARTS ( 2011, 8, 15, 14, 23, 44, 5, 1 );
SELECT DATETIME2FROMPARTS ( 2011, 8, 15, 14, 23, 44, 50, 2 );
SELECT DATETIME2FROMPARTS ( 2011, 8, 15, 14, 23, 44, 500, 3 );
GO
```

Here is the result set.

```
-----
2011-08-15 14:23:44.5

(1 row(s) affected)

-----
2011-08-15 14:23:44.50

(1 row(s) affected)

-----
2011-08-15 14:23:44.500

(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Simple example without fractions of a second

```
SELECT DATETIME2FROMPARTS ( 2010, 12, 31, 23, 59, 59, 0, 0 ) AS Result;
```

Here is the result set.

```
Result
-----
2010-12-31 23:59:59.0000000
(1 row(s) affected)
```

#### D. Example with fractions of a second

The following example demonstrates the use of the *fractions* and *precision* parameters:

1. When *fractions* has a value of 5 and *precision* has a value of 1, then the value of *fractions* represents 5/10 of a second.
2. When *fractions* has a value of 50 and *precision* has a value of 2, then the value of *fractions* represents 50/100 of a second.
3. When *fractions* has a value of 500 and *precision* has a value of 3, then the value of *fractions* represents 500/1000 of a second.

```
SELECT DATETIME2FROMPARTS ( 2011, 8, 15, 14, 23, 44, 5, 1 );
SELECT DATETIME2FROMPARTS ( 2011, 8, 15, 14, 23, 44, 50, 2 );
SELECT DATETIME2FROMPARTS ( 2011, 8, 15, 14, 23, 44, 500, 3 );
GO
```

Here is the result set.

```
-----
2011-08-15 14:23:44.5
(1 row(s) affected)

-----
2011-08-15 14:23:44.50
(1 row(s) affected)

-----
2011-08-15 14:23:44.500
(1 row(s) affected)
```

## See also

[datetime2 \(Transact-SQL\)](#)

# DATETIMEFROMPARTS (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a **datetime** value for the specified date and time.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DATETIMEFROMPARTS ( year, month, day, hour, minute, seconds, milliseconds )
```

## Arguments

*year*

Integer expression specifying a year.

*month*

Integer expression specifying a month.

*day*

Integer expression specifying a day.

*hour*

Integer expression specifying hours.

*minute*

Integer expression specifying minutes.

*seconds*

Integer expression specifying seconds.

*milliseconds*

Integer expression specifying milliseconds.

## Return types

**datetime**

## Remarks

**DATETIMEFROMPARTS** returns a fully initialized **datetime** value. If the arguments are not valid, then an error is raised. If required arguments are null, then a null is returned.

This function is capable of being remoted to SQL Server 2017 servers and above. It will not be remoted to servers that have a version below SQL Server 2017.

## Examples

```
SELECT DATETIMEFROMPARTS ( 2010, 12, 31, 23, 59, 59, 0 ) AS Result;
```

Here is the result set.

```
Result
-----
2010-12-31 23:59:59.000

(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

```
SELECT DATETIMEFROMPARTS ( 2010, 12, 31, 23, 59, 59, 0 ) AS Result;
```

Here is the result set.

```
Result
-----
2010-12-31 23:59:59.000

(1 row(s) affected)
```

## See also

[datetime \(Transact-SQL\)](#)

# DATETIMEOFFSETFROMPARTS (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a **datetimeoffset** value for the specified date and time and with the specified offsets and precision.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DATETIMEOFFSETFROMPARTS ( year, month, day, hour, minute, seconds, fractions, hour_offset, minute_offset,  
precision )
```

## Arguments

*year*

Integer expression specifying a year.

*month*

Integer expression specifying a month.

*day*

Integer expression specifying a day.

*hour*

Integer expression specifying hours.

*minute*

Integer expression specifying minutes.

*seconds*

Integer expression specifying seconds.

*fractions*

Integer expression specifying fractions.

*hour\_offset*

Integer expression specifying the hour portion of the time zone offset.

*minute\_offset*

Integer expression specifying the minute portion of the time zone offset.

*precision*

Integer literal specifying the precision of the **datetimeoffset** value to be returned.

## Return types

**datetimeoffset(*precision*)**

## Remarks

**DATETIMEOFFSETFROMPARTS** returns a fully initialized **datetimeoffset** data type. The offset arguments are used to represent the time zone offset. If the offset arguments are omitted, then the time zone offset is assumed to be 00:00, that is, there is no time zone offset. If the offset arguments are specified, then both arguments must be present and both must be positive or negative. If *minute\_offset* is specified without *hour\_offset*, an error is raised. If other arguments are not valid, then an error is raised. If required arguments are null, then a null is returned. However, if the *precision* argument is null, then an error is raised.

The *fractions* argument depends on the *precision* argument. For example, if *precision* is 7, then each fraction represents 100 nanoseconds; if *precision* is 3, then each fraction represents a millisecond. If the value of *precision* is zero, then the value of *fractions* must also be zero; otherwise, an error is raised.

This function is capable of being remoted to SQL Server 2017 servers and above. It will not be remoted to servers that have a version below SQL Server 2017.

## Examples

### A. Simple example without fractions of a second

```
SELECT DATETIMEOFFSETFROMPARTS ( 2010, 12, 31, 14, 23, 23, 0, 12, 0, 7 ) AS Result;
```

Here is the result set.

Result
----- 2010-12-07 00:00:00.0000000 +00:00 (1 row(s) affected)

### B. Example with fractions of a second

The following example demonstrates the use of the *fractions* and *precision* parameters:

1. When *fractions* has a value of 5 and *precision* has a value of 1, then the value of *fractions* represents 5/10 of a second.
2. When *fractions* has a value of 50 and *precision* has a value of 2, then the value of *fractions* represents 50/100 of a second.
3. When *fractions* has a value of 500 and *precision* has a value of 3, then the value of *fractions* represents 500/1000 of a second.

```
SELECT DATETIMEOFFSETFROMPARTS ( 2011, 8, 15, 14, 30, 00, 5, 12, 30, 1 );  
SELECT DATETIMEOFFSETFROMPARTS ( 2011, 8, 15, 14, 30, 00, 50, 12, 30, 2 );  
SELECT DATETIMEOFFSETFROMPARTS ( 2011, 8, 15, 14, 30, 00, 500, 12, 30, 3 );  
GO
```

Here is the result set.

```
-----  
2011-08-15 14:30:00.5 +12:30
```

```
(1 row(s) affected)
```

```
-----  
2011-08-15 14:30:00.50 +12:30
```

```
(1 row(s) affected)
```

```
-----  
2011-08-15 14:30:00.500 +12:30
```

```
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Simple example without fractions of a second

```
SELECT DATETIMEOFFSETFROMPARTS ( 2010, 12, 31, 14, 23, 23, 0, 12, 0, 7 ) AS Result;
```

Here is the result set.

```
Result
```

```
-----  
2010-12-07 00:00:00.0000000 +00:00
```

```
(1 row(s) affected)
```

### D. Example with fractions of a second

The following example demonstrates the use of the *fractions* and *precision* parameters:

1. When *fractions* has a value of 5 and *precision* has a value of 1, then the value of *fractions* represents 5/10 of a second.
2. When *fractions* has a value of 50 and *precision* has a value of 2, then the value of *fractions* represents 50/100 of a second.
3. When *fractions* has a value of 500 and *precision* has a value of 3, then the value of *fractions* represents 500/1000 of a second.

```
SELECT DATETIMEOFFSETFROMPARTS ( 2011, 8, 15, 14, 30, 00, 5, 12, 30, 1 );  
SELECT DATETIMEOFFSETFROMPARTS ( 2011, 8, 15, 14, 30, 00, 50, 12, 30, 2 );  
SELECT DATETIMEOFFSETFROMPARTS ( 2011, 8, 15, 14, 30, 00, 500, 12, 30, 3 );  
GO
```

Here is the result set.

```
-----  
2011-08-15 14:30:00.5 +12:30  
  
(1 row(s) affected)  
  
-----  
2011-08-15 14:30:00.50 +12:30  
  
(1 row(s) affected)  
-----  
2011-08-15 14:30:00.500 +12:30  
  
(1 row(s) affected)
```

## See also

[datetimeoffset \(Transact-SQL\)](#)

[AT TIME ZONE \(Transact-SQL\)](#)

# DAY (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns an integer representing the day (day of the month) of the specified *date*.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
DAY ( date )
```

## Arguments

*date*

Is an expression that can be resolved to a **time**, **date**, **smalldatetime**, **datetime**, **datetime2**, or **datetimeoffset** value. The *date* argument can be an expression, column expression, user-defined variable or string literal.

## Return Type

**int**

## Return Value

DAY returns the same value as [DATEPART \(day, date\)](#).

If *date* contains only a time part, the return value is 1, the base day.

## Examples

The following statement returns `30`. This is the number of the day.

```
SELECT DAY('2015-04-30 01:01:01.1234567');
```

The following statement returns `1900, 1, 1`. The argument for *date* is the number `0`. SQL Server interprets `0` as January 1, 1900.

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

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns `30`. This is the number of the day.

```
-- Uses AdventureWorks

SELECT TOP 1 DAY('2010-07-30T01:01:01.1234')
FROM dbo.DimCustomer;
```

The following example returns `1900, 1, 1`. The argument for *date* is the number `0`. SQL Server interprets `0` as January 1, 1900.

```
-- Uses AdventureWorks

SELECT TOP 1 YEAR(0), MONTH(0), DAY(0) FROM dbo.DimCustomer;
```

## See also

[CAST and CONVERT \(Transact-SQL\)](#)

# EOMONTH (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2012) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse

✓ Parallel Data Warehouse

Returns the last day of the month that contains the specified date, with an optional offset.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
EOMONTH ( start_date [, month_to_add ] )
```

## Arguments

*start\_date*

Date expression specifying the date for which to return the last day of the month.

*month\_to\_add*

Optional integer expression specifying the number of months to add to *start\_date*.

If this argument is specified, then **EOMONTH** adds the specified number of months to *start\_date*, and then returns the last day of the month for the resulting date. If this addition overflows the valid range of dates, then an error is raised.

## Return Type

**date**

## Remarks

This function can be remoted to SQL Server 2012 servers and higher. It cannot be remoted to servers with a version lower than SQL Server 2012.

## Examples

### A. EOMONTH with explicit datetime type

```
DECLARE @date DATETIME = '12/1/2011';
SELECT EOMONTH ( @date ) AS Result;
GO
```

Here is the result set.

Result
-----
2011-12-31
(1 row(s) affected)

## B. EOMONTH with string parameter and implicit conversion

```
DECLARE @date VARCHAR(255) = '12/1/2011';
SELECT EOMONTH ( @date ) AS Result;
GO
```

Here is the result set.

```
Result
-----
2011-12-31

(1 row(s) affected)
```

## C. EOMONTH with and without the month\_to\_add parameter

```
DECLARE @date DATETIME = GETDATE();
SELECT EOMONTH ( @date ) AS 'This Month';
SELECT EOMONTH ( @date, 1 ) AS 'Next Month';
SELECT EOMONTH ( @date, -1 ) AS 'Last Month';
GO
```

Here is the result set.

```
This Month
-----
2011-12-31

(1 row(s) affected)

Next Month
-----
2012-01-31

(1 row(s) affected)

Last Month
-----
2011-11-30

(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D. EOMONTH with explicit datetime type

```
DECLARE @date DATETIME = '12/1/2011';
SELECT EOMONTH ( @date ) AS Result;
GO
```

Here is the result set.

```
Result
-----
2011-12-31

(1 row(s) affected)
```

## E. EOMONTH with string parameter and implicit conversion

```
DECLARE @date VARCHAR(255) = '12/1/2011';
SELECT EOMONTH ( @date ) AS Result;
GO
```

Here is the result set.

```
Result
-----
2011-12-31

(1 row(s) affected)
```

## F. EOMONTH with and without the month\_to\_add parameter

```
DECLARE @date DATETIME = GETDATE();
SELECT EOMONTH ( @date ) AS 'This Month';
SELECT EOMONTH ( @date, 1 ) AS 'Next Month';
SELECT EOMONTH ( @date, -1 ) AS 'Last Month';
GO
```

Here is the result set.

```
This Month
-----
2011-12-31

(1 row(s) affected)

Next Month
-----
2012-01-31

(1 row(s) affected)

Last Month
-----
2011-11-30

(1 row(s) affected)
```

# GETDATE (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the current database system timestamp as a **datetime** value without the database time zone offset. This value is derived from the operating system of the computer on which the instance of SQL Server is running.

## NOTE

SYSDATETIME and SYSUTCDATETIME have more fractional seconds precision than GETDATE and GETUTCDATE. SYSDATETIMEOFFSET includes the system time zone offset. SYSDATETIME, SYSUTCDATETIME, and SYSDATETIMEOFFSET can be assigned to a variable of any of the date and time types.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
GETDATE ( )
```

## Return Type

**datetime**

## Remarks

Transact-SQL statements can refer to GETDATE anywhere they can refer to a **datetime** expression.

GETDATE is a nondeterministic function. Views and expressions that reference this function in a column cannot be indexed.

Using SWITCHOFFSET with the function GETDATE() can cause the query to run slowly because the query optimizer is unable to obtain accurate cardinality estimates for the GETDATE value. We recommend that you precompute the GETDATE value and then specify that value in the query as shown in the following example. In addition, use the OPTION (RECOMPILE) query hint to force the query optimizer to recompile a query plan the next time the same query is executed. The optimizer will then have accurate cardinality estimates for GETDATE() and will produce a more efficient query plan.

```
DECLARE @dt datetimeoffset = switchoffset (CONVERT(datetimeoffset, GETDATE()), '-04:00');
SELECT * FROM t
WHERE c1 > @dt OPTION (RECOMPILE);
```

## Examples

The following examples use the six SQL Server system functions that return current date and time to return the date, time, or both. The values are returned in series; therefore, their fractional seconds might be different.

## A. Getting the current system date and time

```
SELECT SYSDATETIME()
      ,SYSDATETIMEOFFSET()
      ,SYSUTCDATETIME()
      ,CURRENT_TIMESTAMP
      ,GETDATE()
      ,GETUTCDATE();
```

Here is the result set.

```
SYSDATETIME() 2007-04-30 13:10:02.0474381
```

```
SYSDATETIMEOFFSET()2007-04-30 13:10:02.0474381 -07:00
```

```
SYSUTCDATETIME() 2007-04-30 20:10:02.0474381
```

```
CURRENT_TIMESTAMP 2007-04-30 13:10:02.047
```

```
GETDATE() 2007-04-30 13:10:02.047
```

```
GETUTCDATE() 2007-04-30 20:10:02.047
```

## B. Getting the current system date

```
SELECT CONVERT (date, SYSDATETIME())
      ,CONVERT (date, SYSDATETIMEOFFSET())
      ,CONVERT (date, SYSUTCDATETIME())
      ,CONVERT (date, CURRENT_TIMESTAMP)
      ,CONVERT (date, GETDATE())
      ,CONVERT (date, GETUTCDATE());
```

Here is the result set.

```
SYSDATETIME() 2007-05-03
```

```
SYSDATETIMEOFFSET() 2007-05-03
```

```
SYSUTCDATETIME() 2007-05-04
```

```
CURRENT_TIMESTAMP 2007-05-03
```

```
GETDATE() 2007-05-03
```

```
GETUTCDATE() 2007-05-04
```

## C. Getting the current system time

```
SELECT CONVERT (time, SYSDATETIME())
      ,CONVERT (time, SYSDATETIMEOFFSET())
      ,CONVERT (time, SYSUTCDATETIME())
      ,CONVERT (time, CURRENT_TIMESTAMP)
      ,CONVERT (time, GETDATE())
      ,CONVERT (time, GETUTCDATE());
```

Here is the result set.

```
SYSDATETIME() 13:18:45.3490361
```

```
SYSDATETIMEOFFSET()13:18:45.3490361
```

```
SYSUTCDATETIME() 20:18:45.3490361
```

```
CURRENT_TIMESTAMP 13:18:45.3470000
```

```
GETDATE() 13:18:45.3470000
```

```
GETUTCDATE() 20:18:45.3470000
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following examples use the three SQL Server system functions that return current date and time to return the date, time, or both. The values are returned in series; therefore, their fractional seconds might be different.

### D. Getting the current system date and time

```
SELECT SYSDATETIME()
      ,CURRENT_TIMESTAMP
      ,GETDATE();
```

### E. Getting the current system date

```
SELECT CONVERT (date, SYSDATETIME())
      ,CONVERT (date, CURRENT_TIMESTAMP)
      ,CONVERT (date, GETDATE());
```

### F. Getting the current system time

```
SELECT CONVERT (time, SYSDATETIME())
      ,CONVERT (time, CURRENT_TIMESTAMP)
      ,CONVERT (time, GETDATE());
```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

# GETUTCDATE (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the current database system timestamp as a **datetime** value. The database time zone offset is not included. This value represents the current UTC time (Coordinated Universal Time). This value is derived from the operating system of the computer on which the instance of SQL Server is running.

## NOTE

SYSDATETIME and SYSUTCDATETIME have more fractional seconds precision than GETDATE and GETUTCDATE. SYSDATETIMEOFFSET includes the system time zone offset. SYSDATETIME, SYSUTCDATETIME, and SYSDATETIMEOFFSET can be assigned to a variable of any of the date and time types.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).



## Syntax

```
GETUTCDATE()
```

## Return Types

**datetime**

## Remarks

Transact-SQL statements can refer to GETUTCDATE anywhere they can refer to a **datetime** expression.

GETUTCDATE is a nondeterministic function. Views and expressions that reference this function in a column cannot be indexed.

## Examples

The following examples use the six SQL Server system functions that return current date and time to return the date, time or both. The values are returned in series; therefore, their fractional seconds might be different.

### A. Getting the current system date and time

```

SELECT 'SYSDATETIME()'      ', SYSDATETIME();
SELECT 'SYSDATETIMEOFFSET()', SYSDATETIMEOFFSET();
SELECT 'SYSUTCDATETIME()'   ', SYSUTCDATETIME();
SELECT 'CURRENT_TIMESTAMP'  ', CURRENT_TIMESTAMP;
SELECT 'GETDATE()'          ', GETDATE();
SELECT 'GETUTCDATE()'       ', GETUTCDATE();
/* Returned:
SYSDATETIME()              2007-05-03 18:34:11.9351421
SYSDATETIMEOFFSET()         2007-05-03 18:34:11.9351421 -07:00
SYSUTCDATETIME()           2007-05-04 01:34:11.9351421
CURRENT_TIMESTAMP           2007-05-03 18:34:11.933
GETDATE()                  2007-05-03 18:34:11.933
GETUTCDATE()               2007-05-04 01:34:11.933
*/

```

## B. Getting the current system date

```

SELECT 'SYSDATETIME()'      ', CONVERT (date, SYSDATETIME());
SELECT 'SYSDATETIMEOFFSET()', CONVERT (date, SYSDATETIMEOFFSET());
SELECT 'SYSUTCDATETIME()'   ', CONVERT (date, SYSUTCDATETIME());
SELECT 'CURRENT_TIMESTAMP'  ', CONVERT (date, CURRENT_TIMESTAMP);
SELECT 'GETDATE()'          ', CONVERT (date, GETDATE());
SELECT 'GETUTCDATE()'       ', CONVERT (date, GETUTCDATE());

/* Returned:
SYSDATETIME()              2007-05-03
SYSDATETIMEOFFSET()         2007-05-03
SYSUTCDATETIME()           2007-05-04
CURRENT_TIMESTAMP           2007-05-03
GETDATE()                  2007-05-03
GETUTCDATE()               2007-05-04
*/

```

## C. Getting the current system time

```

SELECT 'SYSDATETIME()'      ', CONVERT (time, SYSDATETIME());
SELECT 'SYSDATETIMEOFFSET()', CONVERT (time, SYSDATETIMEOFFSET());
SELECT 'SYSUTCDATETIME()'   ', CONVERT (time, SYSUTCDATETIME());
SELECT 'CURRENT_TIMESTAMP'  ', CONVERT (time, CURRENT_TIMESTAMP);
SELECT 'GETDATE()'          ', CONVERT (time, GETDATE());
SELECT 'GETUTCDATE()'       ', CONVERT (time, GETUTCDATE());
/* Returned
SYSDATETIME()              18:25:01.6958841
SYSDATETIMEOFFSET()         18:25:01.6958841
SYSUTCDATETIME()           01:25:01.6958841
CURRENT_TIMESTAMP           18:25:01.6930000
GETDATE()                  18:25:01.6930000
GETUTCDATE()               01:25:01.6930000
*/

```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following examples use the six system functions that return current date and time to return the date, time or both. The values are returned in series; therefore, their fractional seconds might be different.

## D. Getting the current system date and time

```

SELECT 'SYSDATETIME()'      ', SYSDATETIME();
SELECT 'SYSDATETIMEOFFSET()', SYSDATETIMEOFFSET();
SELECT 'SYSUTCDATETIME()'   ', SYSUTCDATETIME();
SELECT 'CURRENT_TIMESTAMP'  ', CURRENT_TIMESTAMP;
SELECT 'GETDATE()'          ', GETDATE();
SELECT 'GETUTCDATE()'       ', GETUTCDATE();

/* Returned:
SYSDATETIME()              2007-05-03 18:34:11.9351421
SYSDATETIMEOFFSET()         2007-05-03 18:34:11.9351421 -07:00
SYSUTCDATETIME()           2007-05-04 01:34:11.9351421
CURRENT_TIMESTAMP           2007-05-03 18:34:11.933
GETDATE()                  2007-05-03 18:34:11.933
GETUTCDATE()               2007-05-04 01:34:11.933
*/

```

## E. Getting the current system date

```

SELECT 'SYSDATETIME()'      ', CONVERT (date, SYSDATETIME());
SELECT 'SYSDATETIMEOFFSET()', CONVERT (date, SYSDATETIMEOFFSET());
SELECT 'SYSUTCDATETIME()'   ', CONVERT (date, SYSUTCDATETIME());
SELECT 'CURRENT_TIMESTAMP'  ', CONVERT (date, CURRENT_TIMESTAMP);
SELECT 'GETDATE()'          ', CONVERT (date, GETDATE());
SELECT 'GETUTCDATE()'       ', CONVERT (date, GETUTCDATE());

/* Returned:
SYSDATETIME()              2007-05-03
SYSDATETIMEOFFSET()         2007-05-03
SYSUTCDATETIME()           2007-05-04
CURRENT_TIMESTAMP           2007-05-03
GETDATE()                  2007-05-03
GETUTCDATE()               2007-05-04
*/

```

## F. Getting the current system time

```

SELECT 'SYSDATETIME()'      ', CONVERT (time, SYSDATETIME());
SELECT 'SYSDATETIMEOFFSET()', CONVERT (time, SYSDATETIMEOFFSET());
SELECT 'SYSUTCDATETIME()'   ', CONVERT (time, SYSUTCDATETIME());
SELECT 'CURRENT_TIMESTAMP'  ', CONVERT (time, CURRENT_TIMESTAMP);
SELECT 'GETDATE()'          ', CONVERT (time, GETDATE());
SELECT 'GETUTCDATE()'       ', CONVERT (time, GETUTCDATE());

/* Returned
SYSDATETIME()              18:25:01.6958841
SYSDATETIMEOFFSET()         18:25:01.6958841
SYSUTCDATETIME()           01:25:01.6958841
CURRENT_TIMESTAMP           18:25:01.6930000
GETDATE()                  18:25:01.6930000
GETUTCDATE()               01:25:01.6930000
*/

```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)  
[AT TIME ZONE \(Transact-SQL\)](#)

# ISDATE (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns 1 if the *expression* is a valid **date**, **time**, or **datetime** value; otherwise, 0.

ISDATE returns 0 if the *expression* is a **datetime2** value.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#). Note that the range for datetime data is 1753-01-01 through 9999-12-31, while the range for date data is 0001-01-01 through 9999-12-31.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ISDATE ( expression )
```

## Arguments

*expression*

Is a character string or [expression](#) that can be converted to a character string. The expression must be less than 4,000 characters. Date and time data types, except datetime and smalldatetime, are not allowed as the argument for ISDATE.

## Return Type

**int**

## Remarks

ISDATE is deterministic only if you use it with the [CONVERT](#) function, if the CONVERT style parameter is specified, and style is not equal to 0, 100, 9, or 109.

The return value of ISDATE depends on the settings set by [SET DATEFORMAT](#), [SET LANGUAGE](#) and [Configure the default language Server Configuration Option](#).

## ISDATE expression Formats

For examples of valid formats for which ISDATE will return 1, see the section "Supported String Literal Formats for datetime" in the [datetime](#) and [smalldatetime](#) topics. For additional examples, also see the Input/Output column of the "Arguments" section of [CAST and CONVERT](#).

The following table summarizes input expression formats that are not valid and that return 0 or an error.

ISDATE EXPRESSION	ISDATE RETURN VALUE
NULL	0

ISDATE EXPRESSION	ISDATE RETURN VALUE
Values of data types listed in <a href="#">Data Types</a> in any data type category other than character strings, Unicode character strings, or date and time.	0
Values of <b>text</b> , <b>ntext</b> , or <b>image</b> data types.	0
Any value that has a seconds precision scale greater than 3, (.0000 through .000000...n). ISDATE will return 0 if the expression is a <b>datetime2</b> value, but will return 1 if the expression is a valid <b>datetime</b> value.	0
Any value that mixes a valid date with an invalid value, for example 1995-10-1a.	0

## Examples

### A. Using ISDATE to test for a valid datetime expression

The following example shows you how to use `ISDATE` to test whether a character string is a valid **datetime**.

```
IF ISDATE('2009-05-12 10:19:41.177') = 1
    PRINT 'VALID'
ELSE
    PRINT 'INVALID';
```

### B. Showing the effects of the SET DATEFORMAT and SET LANGUAGE settings on return values

The following statements show the values that are returned as a result of the settings of `SET DATEFORMAT` and `SET LANGUAGE`.

```

/* Use these sessions settings. */
SET LANGUAGE us_english;
SET DATEFORMAT mdy;
/* Expression in mdy dateformat */
SELECT ISDATE('04/15/2008'); --Returns 1.
/* Expression in mdy dateformat */
SELECT ISDATE('04-15-2008'); --Returns 1.
/* Expression in mdy dateformat */
SELECT ISDATE('04.15.2008'); --Returns 1.
/* Expression in myd dateformat */
SELECT ISDATE('04/2008/15'); --Returns 1.

SET DATEFORMAT mdy;
SELECT ISDATE('15/04/2008'); --Returns 0.
SET DATEFORMAT mdy;
SELECT ISDATE('15/2008/04'); --Returns 0.
SET DATEFORMAT mdy;
SELECT ISDATE('2008/15/04'); --Returns 0.
SET DATEFORMAT mdy;
SELECT ISDATE('2008/04/15'); --Returns 1.

SET DATEFORMAT dmy;
SELECT ISDATE('15/04/2008'); --Returns 1.
SET DATEFORMAT dym;
SELECT ISDATE('15/2008/04'); --Returns 1.
SET DATEFORMAT ydm;
SELECT ISDATE('2008/15/04'); --Returns 1.
SET DATEFORMAT ymd;
SELECT ISDATE('2008/04/15'); --Returns 1.

SET LANGUAGE English;
SELECT ISDATE('15/04/2008'); --Returns 0.
SET LANGUAGE Hungarian;
SELECT ISDATE('15/2008/04'); --Returns 0.
SET LANGUAGE Swedish;
SELECT ISDATE('2008/15/04'); --Returns 0.
SET LANGUAGE Italian;
SELECT ISDATE('2008/04/15'); --Returns 1.

/* Return to these sessions settings. */
SET LANGUAGE us_english;
SET DATEFORMAT mdy;

```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Using ISDATE to test for a valid datetime expression

The following example shows you how to use `ISDATE` to test whether a character string is a valid **datetime**.

```

IF ISDATE('2009-05-12 10:19:41.177') = 1
    SELECT 'VALID';
ELSE
    SELECT 'INVALID';

```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

# MONTH (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns an integer that represents the month of the specified *date*.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).

[Transact-SQL Syntax Conventions](#)

## Syntax

```
MONTH ( date )
```

## Arguments

*date*

Is an expression that can be resolved to a **time**, **date**, **smalldatetime**, **datetime**, **datetime2**, or **datetimeoffset** value. The *date* argument can be an expression, column expression, user-defined variable, or string literal.

## Return Type

**int**

## Return Value

MONTH returns the same value as [DATEPART \(month, date\)](#).

If *date* contains only a time part, the return value is 1, the base month.

## Examples

The following statement returns `4`. This is the number of the month.

```
SELECT MONTH('2007-04-30T01:01:01.1234567 -07:00');
```

The following statement returns `1900, 1, 1`. The argument for *date* is the number `0`. SQL Server interprets `0` as January 1, 1900.

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

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns `4`. This is the number of the month.

```
-- Uses AdventureWorks

SELECT TOP 1 MONTH('2007-04-30T01:01:01.1234')
FROM dbo.DimCustomer;
```

The following example returns `1900, 1, 1`. The argument for *date* is the number `0`. SQL Server interprets `0` as January 1, 1900.

```
-- Uses AdventureWorks

SELECT TOP 1 YEAR(0), MONTH(0), DAY(0) FROM dbo.DimCustomer;
```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

# SMALLDATETIMEFROMPARTS (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a **smalldatetime** value for the specified date and time.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
SMALLDATETIMEFROMPARTS ( year, month, day, hour, minute )
```

## Arguments

*year*

Integer expression specifying a year.

*month*

Integer expression specifying a month.

*day*

Integer expression specifying a day.

*hour*

Integer expression specifying hours.

*minute*

Integer expression specifying minutes.

## Return Types

**smalldatetime**

## Remarks

This function acts like a constructor for a fully initialized **smalldatetime** value. If the arguments are not valid, then an error is thrown. If required arguments are null, then null is returned.

This function is capable of being remoted to SQL Server 2017 servers and above. It will not be remoted to servers that have a version below SQL Server 2017.

## Examples

```
SELECT SMALLDATETIMEFROMPARTS ( 2010, 12, 31, 23, 59 ) AS Result
```

Here is the result set.

```
Result
-----
2011-01-01 00:00:00

(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

```
SELECT SMALLDATETIMEFROMPARTS ( 2010, 12, 31, 23, 59 ) AS Result
```

Here is the result set.

```
Result
-----
2011-01-01 00:00:00

(1 row(s) affected)
```

# SWITCHOFFSET (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a **datetimeoffset** value that is changed from the stored time zone offset to a specified new time zone offset.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SWITCHOFFSET ( DATETIMEOFFSET, time_zone )
```

## Arguments

**DATETIMEOFFSET**

Is an expression that can be resolved to a **datetimeoffset(n)** value.

**time\_zone**

Is a character string in the format [+|-]TZH:TZM or a signed integer (of minutes) that represents the time zone offset, and is assumed to be daylight-saving aware and adjusted.

## Return Type

**datetimeoffset** with the fractional precision of the **DATETIMEOFFSET** argument.

## Remarks

Use SWITCHOFFSET to select a **datetimeoffset** value into a time zone offset that is different from the time zone offset that was originally stored. SWITCHOFFSET does not update the stored **time\_zone** value.

SWITCHOFFSET can be used to update a **datetimeoffset** column.

Using SWITCHOFFSET with the function GETDATE() can cause the query to run slowly. This is because the query optimizer is unable to obtain accurate cardinality estimates for the datetime value. To resolve this problem, use the OPTION (RECOMPILE) query hint to force the query optimizer to recompile a query plan the next time the same query is executed. The optimizer will then have accurate cardinality estimates and will produce a more efficient query plan. For more information about the RECOMPILE query hint, see [Query Hints \(Transact-SQL\)](#).

```
DECLARE @dt datetimeoffset = switchoffset (CONVERT(datetimeoffset, GETDATE()), '-04:00');
SELECT * FROM t
WHERE c1 > @dt OPTION (RECOMPILE);
```

## Examples

The following example uses **SWITCHOFFSET** to display a different time zone offset than the value stored in the

database.

```
CREATE TABLE dbo.test
(
    ColDatetimeoffset datetimeoffset
);
GO
INSERT INTO dbo.test
VALUES ('1998-09-20 7:45:50.71345 -5:00');
GO
SELECT SWITCHOFFSET (ColDatetimeoffset, '-08:00')
FROM dbo.test;
GO
--Returns: 1998-09-20 04:45:50.7134500 -08:00
SELECT ColDatetimeoffset
FROM dbo.test;
--Returns: 1998-09-20 07:45:50.7134500 -05:00
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example uses `SWITCHOFFSET` to display a different time zone offset than the value stored in the database.

```
CREATE TABLE dbo.test
(
    ColDatetimeoffset datetimeoffset
);
GO
INSERT INTO dbo.test
VALUES ('1998-09-20 7:45:50.71345 -5:00');
GO
SELECT SWITCHOFFSET (ColDatetimeoffset, '-08:00')
FROM dbo.test;
GO
--Returns: 1998-09-20 04:45:50.7134500 -08:00
SELECT ColDatetimeoffset
FROM dbo.test;
--Returns: 1998-09-20 07:45:50.7134500 -05:00
```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

[AT TIME ZONE \(Transact-SQL\)](#)

# SYSDATETIME (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a **datetime2(7)** value that contains the date and time of the computer on which the instance of SQL Server is running.

## NOTE

SYSDATETIME and SYSUTCDATETIME have more fractional seconds precision than GETDATE and GETUTCDATE. SYSDATETIMEOFFSET includes the system time zone offset. SYSDATETIME, SYSUTCDATETIME, and SYSDATETIMEOFFSET can be assigned to a variable of any of the date and time types.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SYSDATETIME ( )
```

## Return Type

**datetime2(7)**

## Remarks

Transact-SQL statements can refer to SYSDATETIME anywhere they can refer to a **datetime2(7)** expression.

SYSDATETIME is a nondeterministic function. Views and expressions that reference this function in a column cannot be indexed.

## NOTE

SQL Server obtains the date and time values by using the GetSystemTimeAsFileTime() Windows API. The accuracy depends on the computer hardware and version of Windows on which the instance of SQL Server is running. The precision of this API is fixed at 100 nanoseconds. The accuracy can be determined by using the GetSystemTimeAdjustment() Windows API.

## Examples

The following examples use the six SQL Server system functions that return current date and time to return the date, time or both. The values are returned in series; therefore, their fractional seconds might be different.

### A. Getting the current system date and time

```

SELECT SYSDATETIME()
      ,SYSDATETIMEOFFSET()
      ,SYSUTCDATETIME()
      ,CURRENT_TIMESTAMP
      ,GETDATE()
      ,GETUTCDATE();
/* Returned:
SYSDATETIME()      2007-04-30 13:10:02.0474381
SYSDATETIMEOFFSET()2007-04-30 13:10:02.0474381 -07:00
SYSUTCDATETIME()   2007-04-30 20:10:02.0474381
CURRENT_TIMESTAMP   2007-04-30 13:10:02.047
GETDATE()          2007-04-30 13:10:02.047
GETUTCDATE()       2007-04-30 20:10:02.047
*/

```

## B. Getting the current system date

```

SELECT CONVERT (date, SYSDATETIME())
      ,CONVERT (date, SYSDATETIMEOFFSET())
      ,CONVERT (date, SYSUTCDATETIME())
      ,CONVERT (date, CURRENT_TIMESTAMP)
      ,CONVERT (date, GETDATE())
      ,CONVERT (date, GETUTCDATE());

/* All returned 2007-04-30 */

```

## C. Getting the current system time

```

SELECT CONVERT (time, SYSDATETIME())
      ,CONVERT (time, SYSDATETIMEOFFSET())
      ,CONVERT (time, SYSUTCDATETIME())
      ,CONVERT (time, CURRENT_TIMESTAMP)
      ,CONVERT (time, GETDATE())
      ,CONVERT (time, GETUTCDATE());

/* Returned
SYSDATETIME()      13:18:45.3490361
SYSDATETIMEOFFSET()13:18:45.3490361
SYSUTCDATETIME()   20:18:45.3490361
CURRENT_TIMESTAMP   13:18:45.3470000
GETDATE()          13:18:45.3470000
GETUTCDATE()       20:18:45.3470000
*/

```

# Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

## D: Getting the current system date and time

```
SELECT SYSDATETIME();
```

Here is the result set.

-----

7/20/2013 2:49:59 PM

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)



# SYSDATETIMEOFFSET (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a **datetimeoffset(7)** value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is included.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SYSDATETIMEOFFSET ( )
```

## Return Type

**datetimeoffset(7)**

## Remarks

Transact-SQL statements can refer to SYSDATETIMEOFFSET anywhere they can refer to a **datetimeoffset** expression.

SYSDATETIMEOFFSET is a nondeterministic function. Views and expressions that reference this function in a column cannot be indexed.

### NOTE

SQL Server obtains the date and time values by using the GetSystemTimeAsFileTime() Windows API. The accuracy depends on the computer hardware and version of Windows on which the instance of SQL Server is running. The precision of this API is fixed at 100 nanoseconds. The accuracy can be determined by using the GetSystemTimeAdjustment() Windows API.

## Examples

The following examples use the six SQL Server system functions that return current date and time to return the date, time, or both. The values are returned in series; therefore, their fractional seconds might be different.

### A. Showing the formats that are returned by the date and time functions

The following example shows the different formats that are returned by the date and time functions.

```
SELECT SYSDATETIME() AS SYSDATETIME  
,SYSDATETIMEOFFSET() AS SYSDATETIMEOFFSET  
,SYSUTCDATETIME() AS SYSUTCDATETIME  
,CURRENT_TIMESTAMP AS CURRENT_TIMESTAMP  
,GETDATE() AS GETDATE  
,GETUTCDATE() AS GETUTCDATE;
```

Here is the result set.

```
SYSDATETIME() 2007-04-30 13:10:02.0474381
```

```
SYSDATETIMEOFFSET()2007-04-30 13:10:02.0474381 -07:00
```

```
SYSUTCDATETIME() 2007-04-30 20:10:02.0474381
```

```
CURRENT_TIMESTAMP 2007-04-30 13:10:02.047
```

```
GETDATE() 2007-04-30 13:10:02.047
```

```
GETUTCDATE() 2007-04-30 20:10:02.047
```

## B. Converting date and time to date

The following example shows you how to convert date and time values to `date`.

```
SELECT CONVERT (date, SYSDATETIME())
      ,CONVERT (date, SYSDATETIMEOFFSET())
      ,CONVERT (date, SYSUTCDATETIME())
      ,CONVERT (date, CURRENT_TIMESTAMP)
      ,CONVERT (date, GETDATE())
      ,CONVERT (date, GETUTCDATE());
```

Here is the result set.

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

## C. Converting date and time to times

The following example shows you how to convert date and time values to `time`.

```
SELECT CONVERT (time, SYSDATETIME()) AS SYSDATETIME()
      ,CONVERT (time, SYSDATETIMEOFFSET()) AS SYSDATETIMEOFFSET()
      ,CONVERT (time, SYSUTCDATETIME()) AS SYSUTCDATETIME()
      ,CONVERT (time, CURRENT_TIMESTAMP) AS CURRENT_TIMESTAMP
      ,CONVERT (time, GETDATE()) AS GETDATE()
      ,CONVERT (time, GETUTCDATE()) AS GETUTCDATE();
```

Here is the result set.

```
SYSDATETIME() 13:18:45.3490361
```

```
SYSDATETIMEOFFSET()13:18:45.3490361
```

```
SYSUTCDATETIME() 20:18:45.3490361
```

```
CURRENT_TIMESTAMP 13:18:45.3470000
```

```
GETDATE() 13:18:45.3470000
```

```
GETUTCDATE() 20:18:45.3470000
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following examples use the six SQL Server system functions that return current date and time to return the date, time, or both. The values are returned in series; therefore, their fractional seconds might be different.

### D. Showing the formats that are returned by the date and time functions

The following example shows the different formats that are returned by the date and time functions.

```
SELECT SYSDATETIME() AS SYSDATETIME  
,SYSDATETIMEOFFSET() AS SYSDATETIMEOFFSET  
,SYSUTCDATETIME() AS SYSUTCDATETIME  
,CURRENT_TIMESTAMP AS CURRENT_TIMESTAMP  
,GETDATE() AS GETDATE  
,GETUTCDATE() AS GETUTCDATE;
```

Here is the result set.

```
SYSDATETIME() 2007-04-30 13:10:02.0474381
```

```
SYSDATETIMEOFFSET()2007-04-30 13:10:02.0474381 -07:00
```

```
SYSUTCDATETIME() 2007-04-30 20:10:02.0474381
```

```
CURRENT_TIMESTAMP 2007-04-30 13:10:02.047
```

```
GETDATE() 2007-04-30 13:10:02.047
```

```
GETUTCDATE() 2007-04-30 20:10:02.047
```

### E. Converting date and time to date

The following example shows you how to convert date and time values to `date`.

```
SELECT CONVERT (date, SYSDATETIME())  
,CONVERT (date, SYSDATETIMEOFFSET())  
,CONVERT (date, SYSUTCDATETIME())  
,CONVERT (date, CURRENT_TIMESTAMP)  
,CONVERT (date, GETDATE())  
,CONVERT (date, GETUTCDATE());
```

Here is the result set.

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

### F. Converting date and time to times

The following example shows you how to convert date and time values to `time`.

```
SELECT CONVERT (time, SYSDATETIME()) AS SYSDATETIME()
      ,CONVERT (time, SYSDATETIMEOFFSET()) AS SYSDATETIMEOFFSET()
      ,CONVERT (time, SYSUTCDATETIME()) AS SYSUTCDATETIME()
      ,CONVERT (time, CURRENT_TIMESTAMP) AS CURRENT_TIMESTAMP
      ,CONVERT (time, GETDATE()) AS GETDATE()
      ,CONVERT (time, GETUTCDATE()) AS GETUTCDATE();
```

Here is the result set.

SYSDATETIME() 13:18:45.3490361

SYSDATETIMEOFFSET()13:18:45.3490361

SYSUTCDATETIME() 20:18:45.3490361

CURRENT\_TIMESTAMP 13:18:45.3470000

GETDATE() 13:18:45.3470000

GETUTCDATE() 20:18:45.3470000

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

[Date and Time Data Types and Functions \(Transact-SQL\)](#)

# SYSUTCDATETIME (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a **datetime2** value that contains the date and time of the computer on which the instance of SQL Server is running. The date and time is returned as UTC time (Coordinated Universal Time). The fractional second precision specification has a range from 1 to 7 digits. The default precision is 7 digits.

## NOTE

SYSDATETIME and SYSUTCDATE have more fractional seconds precision than GETDATE and GETUTCDATE. SYSDATETIMEOFFSET includes the system time zone offset. SYSDATETIME, SYSUTCDATE, and SYSDATETIMEOFFSET can be assigned to a variable of any one of the date and time types.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SYSUTCDATETIME ( )
```

## Return Type

**datetime2**

## Remarks

Transact-SQL statements can refer to SYSUTCDATETIME anywhere they can refer to a **datetime2** expression.

SYSUTCDATETIME is a nondeterministic function. Views and expressions that reference this function in a column cannot be indexed.

## NOTE

SQL Server obtains the date and time values by using the GetSystemTimeAsFileTime() Windows API. The accuracy depends on the computer hardware and version of Windows on which the instance of SQL Server is running. The precision of this API is fixed at 100 nanoseconds. The accuracy can be determined by using the GetSystemTimeAdjustment() Windows API.

## Examples

The following examples use the six SQL Server system functions that return current date and time to return the date, time, or both. The values are returned in series; therefore, their fractional seconds might be different.

### A. Showing the formats that are returned by the date and time functions

The following example shows the different formats that are returned by the date and time functions.

```
SELECT SYSDATETIME() AS SYSDATETIME  
,SYSDATETIMEOFFSET() AS SYSDATETIMEOFFSET  
,SYSUTCDATETIME() AS SYSUTCDATETIME  
,CURRENT_TIMESTAMP AS CURRENT_TIMESTAMP  
,GETDATE() AS GETDATE  
,GETUTCDATE() AS GETUTCDATE;
```

Here is the result set.

```
SYSDATETIME() 2007-04-30 13:10:02.0474381
```

```
SYSDATETIMEOFFSET()2007-04-30 13:10:02.0474381 -07:00
```

```
SYSUTCDATETIME() 2007-04-30 20:10:02.0474381
```

```
CURRENT_TIMESTAMP 2007-04-30 13:10:02.047
```

```
GETDATE() 2007-04-30 13:10:02.047
```

```
GETUTCDATE() 2007-04-30 20:10:02.047
```

## B. Converting date and time to date

The following example shows you how to convert date and time values to `date`.

```
SELECT CONVERT (date, SYSDATETIME())  
,CONVERT (date, SYSDATETIMEOFFSET())  
,CONVERT (date, SYSUTCDATETIME())  
,CONVERT (date, CURRENT_TIMESTAMP)  
,CONVERT (date, GETDATE())  
,CONVERT (date, GETUTCDATE());
```

Here is the result set.

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

## C. Converting date and time values to time

The following example shows you how to convert date and time values to `time`.

```
DECLARE @DATETIME DATETIME = GetDate();
```

```
DECLARE @TIME TIME
```

```
SELECT @TIME = CONVERT(time, @DATETIME)
```

```
SELECT @TIME AS 'Time', @DATETIME AS 'Date Time'
```

Here is the result set.

```
Time Date Time
```

```
13:49:33.6330000 2009-04-22 13:49:33.633
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following examples use the six SQL Server system functions that return current date and time to return the date, time, or both. The values are returned in series; therefore, their fractional seconds might be different.

### D. Showing the formats that are returned by the date and time functions

The following example shows the different formats that are returned by the date and time functions.

```
SELECT SYSDATETIME() AS SYSDATETIME  
,SYSDATETIMEOFFSET() AS SYSDATETIMEOFFSET  
,SYSUTCDATETIME() AS SYSUTCDATETIME  
,CURRENT_TIMESTAMP AS CURRENT_TIMESTAMP  
,GETDATE() AS GETDATE  
,GETUTCDATE() AS GETUTCDATE;
```

Here is the result set.

```
SYSDATETIME() 2007-04-30 13:10:02.0474381
```

```
SYSDATETIMEOFFSET()2007-04-30 13:10:02.0474381 -07:00
```

```
SYSUTCDATETIME() 2007-04-30 20:10:02.0474381
```

```
CURRENT_TIMESTAMP 2007-04-30 13:10:02.047
```

```
GETDATE() 2007-04-30 13:10:02.047
```

```
GETUTCDATE() 2007-04-30 20:10:02.047
```

### E. Converting date and time to date

The following example shows you how to convert date and time values to `date`.

```
SELECT CONVERT (date, SYSDATETIME())  
,CONVERT (date, SYSDATETIMEOFFSET())  
,CONVERT (date, SYSUTCDATETIME())  
,CONVERT (date, CURRENT_TIMESTAMP)  
,CONVERT (date, GETDATE())  
,CONVERT (date, GETUTCDATE());
```

Here is the result set.

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

```
2007-04-30
```

### F. Converting date and time values to time

The following example shows you how to convert date and time values to `time`.

```
DECLARE @DATETIME DATETIME = GetDate();
```

```
DECLARE @TIME TIME
```

```
SELECT @TIME = CONVERT(time, @DATETIME)
```

```
SELECT @TIME AS 'Time', @DATETIME AS 'Date Time'
```

Here is the result set.

Time	Date	Time
------	------	------

13:49:33.6330000	2009-04-22	13:49:33.633
------------------	------------	--------------

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

[Date and Time Data Types and Functions \(Transact-SQL\)](#)

[AT TIME ZONE \(Transact-SQL\)](#)

# TIMEFROMPARTS (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a **time** value for the specified time and with the specified precision.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
TIMEFROMPARTS ( hour, minute, seconds, fractions, precision )
```

## Arguments

*hour*

Integer expression specifying hours.

*minute*

Integer expression specifying minutes.

*seconds*

Integer expression specifying seconds.

*fractions*

Integer expression specifying fractions.

*precision*

Integer literal specifying the precision of the **time** value to be returned.

## Return Types

**time(** *precision* **)**

## Remarks

**TIMEFROMPARTS** returns a fully initialized time value. If the arguments are invalid, then an error is raised. If any of the parameters are null, null is returned. However, if the *precision* argument is null, then an error is raised.

The *fractions* argument depends on the *precision* argument. For example, if *precision* is 7, then each fraction represents 100 nanoseconds; if *precision* is 3, then each fraction represents a millisecond. If the value of *precision* is zero, then the value of *fractions* must also be zero; otherwise, an error is raised.

This function can be remoted to SQL Server 2012 servers and higher. It cannot be remoted to servers that have a version lower than SQL Server 2012.

## Examples

### A. Simple example without fractions of a second

```
SELECT TIMEFROMPARTS ( 23, 59, 59, 0, 0 ) AS Result;
```

Here is the result set.

```
Result
-----
23:59:59.0000000
(1 row(s) affected)
```

## B. Example with fractions of a second

The following example demonstrates the use of the *fractions* and *precision* parameters:

1. When *fractions* has a value of 5 and *precision* has a value of 1, then the value of *fractions* represents 5/10 of a second.
2. When *fractions* has a value of 50 and *precision* has a value of 2, then the value of *fractions* represents 50/100 of a second.
3. When *fractions* has a value of 500 and *precision* has a value of 3, then the value of *fractions* represents 500/1000 of a second.

```
SELECT TIMEFROMPARTS ( 14, 23, 44, 5, 1 );
SELECT TIMEFROMPARTS ( 14, 23, 44, 50, 2 );
SELECT TIMEFROMPARTS ( 14, 23, 44, 500, 3 );
GO
```

Here is the result set.

```
-----
14:23:44.5
(1 row(s) affected)

-----
14:23:44.50
(1 row(s) affected)

-----
14:23:44.500
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Simple example without fractions of a second

```
SELECT TIMEFROMPARTS ( 23, 59, 59, 0, 0 ) AS Result;
```

Here is the result set.

```
Result
-----
23:59:59.0000000
(1 row(s) affected)
```

#### D. Example with fractions of a second

The following example demonstrates the use of the *fractions* and *precision* parameters:

1. When *fractions* has a value of 5 and *precision* has a value of 1, then the value of *fractions* represents 5/10 of a second.
2. When *fractions* has a value of 50 and *precision* has a value of 2, then the value of *fractions* represents 50/100 of a second.
3. When *fractions* has a value of 500 and *precision* has a value of 3, then the value of *fractions* represents 500/1000 of a second.

```
SELECT TIMEFROMPARTS ( 14, 23, 44, 5, 1 );
SELECT TIMEFROMPARTS ( 14, 23, 44, 50, 2 );
SELECT TIMEFROMPARTS ( 14, 23, 44, 500, 3 );
GO
```

Here is the result set.

```
-----
14:23:44.5
(1 row(s) affected)

-----
14:23:44.50
(1 row(s) affected)

-----
14:23:44.500
(1 row(s) affected)
```

# TODATETIMEOFFSET (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a **datetimeoffset** value that is translated from a **datetime2** expression.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
TODATETIMEOFFSET ( expression , time_zone )
```

## Arguments

*expression*

Is an [expression](#) that resolves to a **datetime2** value.

**NOTE**

The expression cannot be of type **text**, **ntext**, or **image** because these types cannot be implicitly converted to **varchar** or **nvarchar**.

*time\_zone*

Is an expression that represents the time zone offset in minutes (if an integer), for example -120, or hours and minutes (if a string), for example '+13.00'. The range is +14 to -14 (in hours). The expression is interpreted in local time for the specified *time\_zone*.

**NOTE**

If *expression* is a character string, it must be in the format {+|-}TZH:THM.

## Return Type

**datetimeoffset**. The fractional precision is the same as the *datetime* argument.

## Examples

### A. Changing the time zone offset of the current date and time

The following example changes the zone offset of the current date and time to time zone **-07:00**.

```
DECLARE @todaysDateTime datetime2;
SET @todaysDateTime = GETDATE();
SELECT TODATETIMEOFFSET (@todaysDateTime, '-07:00');
-- RETURNS 2007-08-30 15:51:34.7030000 -07:00
```

### B. Changing the time zone offset in minutes

The following example changes the current time zone to `-120` minutes.

```
DECLARE @todaysDate datetime2;
SET @todaysDate = GETDATE();
SELECT TODATETIMEOFFSET (@todaysDate, -120);
-- RETURNS 2007-08-30 15:52:37.8770000 -02:00
```

### C. Adding a 13-hour time zone offset

The following example adds a 13-hour time zone offset to a date and time.

```
DECLARE @dateTime datetimeoffset(7)= '2007-08-28 18:00:30';
SELECT TODATETIMEOFFSET (@dateTime, '+13:00');
-- RETURNS 2007-08-28 18:00:30.0000000 +13:00
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D. Changing the time zone offset of the current date and time

The following example changes the zone offset of the current date and time to time zone `-07:00`.

```
DECLARE @todaysDateTime datetime2;
SET @todaysDateTime = GETDATE();
SELECT TODATETIMEOFFSET (@todaysDateTime, '-07:00');
-- RETURNS 2007-08-30 15:51:34.7030000 -07:00
```

### E. Changing the time zone offset in minutes

The following example changes the current time zone to `-120` minutes.

```
DECLARE @todaysDate datetime2;
SET @todaysDate = GETDATE();
SELECT TODATETIMEOFFSET (@todaysDate, -120);
-- RETURNS 2007-08-30 15:52:37.8770000 -02:00
```

### F. Adding a 13-hour time zone offset

The following example adds a 13-hour time zone offset to a date and time.

```
DECLARE @dateTime datetimeoffset(7)= '2007-08-28 18:00:30';
SELECT TODATETIMEOFFSET (@dateTime, '+13:00');
-- RETURNS 2007-08-28 18:00:30.0000000 +13:00
```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

[Date and Time Data Types and Functions \(Transact-SQL\)](#)

[AT TIME ZONE \(Transact-SQL\)](#)

# YEAR (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns an integer that represents the year of the specified *date*.

For an overview of all Transact-SQL date and time data types and functions, see [Date and Time Data Types and Functions \(Transact-SQL\)](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
YEAR ( date )
```

## Arguments

*date*

Is an expression that can be resolved to a **time**, **date**, **smalldatetime**, **datetime**, **datetime2**, or **datetimeoffset** value. The *date* argument can be an expression, column expression, user-defined variable or string literal.

## Return Types

**int**

## Return Value

YEAR returns the same value as [DATEPART \(year, date\)](#).

If *date* only contains a time part, the return value is 1900, the base year.

## Examples

The following statement returns `2007`. This is the number of the year.

```
SELECT YEAR('2007-04-30T01:01:01.1234567-07:00');
```

The following statement returns `1900, 1, 1`. The argument for *date* is the number `0`. SQL Server interprets `0` as January 1, 1900.

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

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following statement returns `2010`. This is the number of the year.

```
SELECT YEAR('2010-07-20T01:01:01.1234');
```

The following statement returns `1900, 1, 1`. The argument for *date* is the number `0`. SQL Server interprets `0` as January 1, 1900.

```
SELECT TOP 1 YEAR(0), MONTH(0), DAY(0);
```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

# JSON Functions (Transact-SQL)

7/28/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Use the functions described on the pages in this section to validate or change JSON text or to extract simple or complex values.

FUNCTION	DESCRIPTION
<a href="#">ISJSON</a>	Tests whether a string contains valid JSON.
<a href="#">JSON_VALUE</a>	Extracts a scalar value from a JSON string.
<a href="#">JSON_QUERY</a>	Extracts an object or an array from a JSON string.
<a href="#">JSON_MODIFY</a>	Updates the value of a property in a JSON string and returns the updated JSON string.

For more info about the built-in support for JSON in SQL Server, see [JSON Data \(SQL Server\)](#).

## See Also

[Validate, Query, and Change JSON Data with Built-in Functions \(SQL Server\)](#)

[JSON Path Expressions \(SQL Server\)](#)

[JSON Data \(SQL Server\)](#)

# ISJSON (Transact-SQL)

7/28/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2016) Azure SQL Database Azure SQL Data

Warehouse Parallel Data Warehouse

Tests whether a string contains valid JSON.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
ISJSON ( expression )
```

## Arguments

*expression*

The string to test.

## Return Value

Returns 1 if the string contains valid JSON; otherwise, returns 0. Returns null if *expression* is null.

Does not return errors.

## Remarks

**ISJSON** does not check the uniqueness of keys at the same level.

## Examples

### Example 1

The following example runs a statement block conditionally if the parameter value `@param` contains valid JSON.

```
DECLARE @param <data type>
SET @param = <value>

IF (ISJSON(@param) > 0)
BEGIN
    -- Do something with the valid JSON value of @param.
END
```

### Example 2

The following example returns rows in which the column `json_col` contains valid JSON.

```
SELECT id, json_col
FROM tab1
WHERE ISJSON(json_col) > 0
```

## See Also

[JSON Data \(SQL Server\)](#)

# JSON\_VALUE (Transact-SQL)

7/28/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Extracts a scalar value from a JSON string.

To extract an object or an array from a JSON string instead of a scalar value, see [JSON\\_QUERY \(Transact-SQL\)](#). For info about the differences between **JSON\_VALUE** and **JSON\_QUERY**, see [Compare JSON\\_VALUE and JSON\\_QUERY](#).



## Syntax

```
JSON_VALUE ( expression , path )
```

## Arguments

*expression*

An expression. Typically the name of a variable or a column that contains JSON text.

If **JSON\_VALUE** finds JSON that is not valid in *expression* before it finds the value identified by *path*, the function returns an error. If **JSON\_VALUE** doesn't find the value identified by *path*, it scans the entire text and returns an error if it finds JSON that is not valid anywhere in *expression*.

*path*

A JSON path that specifies the property to extract. For more info, see [JSON Path Expressions \(SQL Server\)](#).

In SQL Server 2017 and in Azure SQL Database, you can provide a variable as the value of *path*.

If the format of *path* isn't valid, **JSON\_VALUE** returns an error .

## Return Value

Returns a single text value of type nvarchar(4000). The collation of the returned value is the same as the collation of the input expression.

If the value is greater than 4000 characters:

- In lax mode, **JSON\_VALUE** returns null.
- In strict mode, **JSON\_VALUE** returns an error.

If you have to return scalar values greater than 4000 characters, use **OPENJSON** instead of **JSON\_VALUE**.

For more info, see [OPENJSON \(Transact-SQL\)](#).

## Remarks

### Lax mode and strict mode

Consider the following JSON text:

```

DECLARE @jsonInfo NVARCHAR(MAX)

SET @jsonInfo=N'{
    "info": {
        "type":1,
        "address": {
            "town":"Bristol",
            "county":"Avon",
            "country":"England"
        },
        "tags":["Sport", "Water polo"]
    },
    "type": "Basic"
}'

```

The following table compares the behavior of **JSON\_VALUE** in lax mode and in strict mode. For more info about the optional path mode specification (lax or strict), see [JSON Path Expressions \(SQL Server\)](#).

PATH	RETURN VALUE IN LAX MODE	RETURN VALUE IN STRICT MODE	MORE INFO
\$	NULL	Error	Not a scalar value. Use <b>JSON_QUERY</b> instead.
\$.info.type	N'1'	N'1'	N/a
\$.info.address.town	N'Bristol'	N'Bristol'	N/a
\$.info."address"	NULL	Error	Not a scalar value. Use <b>JSON_QUERY</b> instead.
\$.info.tags	NULL	Error	Not a scalar value. Use <b>JSON_QUERY</b> instead.
\$.info.type[0]	NULL	Error	Not an array.
\$.info.none	NULL	Error	Property does not exist.

## Examples

### Example 1

The following example uses the values of the JSON properties `town` and `state` in query results. Since **JSON\_VALUE** preserves the collation of the source, the sort order of the results depends on the collation of the `jsonInfo` column.

```

SELECT FirstName,LastName,
    JSON_VALUE(jsonInfo,'$.info.address[0].town') AS Town
FROM Person.Person
WHERE JSON_VALUE(jsonInfo,'$.info.address[0].state') LIKE 'US%'
ORDER BY JSON_VALUE(jsonInfo,'$.info.address[0].town')

```

### Example 2

The following example extracts the value of the JSON property `town` into a local variable.

```
DECLARE @jsonInfo NVARCHAR(MAX)
DECLARE @town NVARCHAR(32)

SET @jsonInfo=N'<array of address info>'

SET @town=JSON_VALUE(@jsonInfo,'$.info.address.town')
```

### Example 3

The following example creates computed columns based on the values of JSON properties.

```
CREATE TABLE dbo.Store
(
    StoreID INT IDENTITY(1,1) NOT NULL,
    Address VARCHAR(500),
    jsonContent NVARCHAR(8000),
    Longitude AS JSON_VALUE(jsonContent, '$.address[0].longitude'),
    Latitude AS JSON_VALUE(jsonContent, '$.address[0].latitude')
)
```

## See Also

[JSON Path Expressions \(SQL Server\)](#)

[JSON Data \(SQL Server\)](#)

# JSON\_QUERY (Transact-SQL)

7/28/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Extracts an object or an array from a JSON string.

To extract a scalar value from a JSON string instead of an object or an array, see [JSON\\_VALUE \(Transact-SQL\)](#). For info about the differences between **JSON\_VALUE** and **JSON\_QUERY**, see [Compare JSON\\_VALUE and JSON\\_QUERY](#).

[Transact-SQL Syntax Conventions](#)

## Syntax

```
JSON_QUERY ( expression [ , path ] )
```

## Arguments

*expression*

An expression. Typically the name of a variable or a column that contains JSON text.

If **JSON\_QUERY** finds JSON that is not valid in *expression* before it finds the value identified by *path*, the function returns an error. If **JSON\_QUERY** doesn't find the value identified by *path*, it scans the entire text and returns an error if it finds JSON that is not valid anywhere in *expression*.

*path*

A JSON path that specifies the object or the array to extract.

In SQL Server 2017 and in Azure SQL Database, you can provide a variable as the value of *path*.

The JSON path can specify lax or strict mode for parsing. If you don't specify the parsing mode, lax mode is the default. For more info, see [JSON Path Expressions \(SQL Server\)](#).

The default value for *path* is '\$'. As a result, if you don't provide a value for *path*, **JSON\_QUERY** returns the input *expression*.

If the format of *path* isn't valid, **JSON\_QUERY** returns an error.

## Return Value

Returns a JSON fragment of type nvarchar(max). The collation of the returned value is the same as the collation of the input expression.

If the value is not an object or an array:

- In lax mode, **JSON\_QUERY** returns null.
- In strict mode, **JSON\_QUERY** returns an error.

## Remarks

## Lax mode and strict mode

Consider the following JSON text:

```
{  
    "info": {  
        "type": 1,  
        "address": {  
            "town": "Bristol",  
            "county": "Avon",  
            "country": "England"  
        },  
        "tags": ["Sport", "Water polo"]  
    },  
    "type": "Basic"  
}
```

The following table compares the behavior of **JSON\_QUERY** in lax mode and in strict mode. For more info about the optional path mode specification (lax or strict), see [JSON Path Expressions \(SQL Server\)](#).

PATH	RETURN VALUE IN LAX MODE	RETURN VALUE IN STRICT MODE	MORE INFO
\$	Returns the entire JSON text.	Returns the entire JSON text.	N/a
\$.info.type	NULL	Error	Not an object or array. Use <b>JSON_VALUE</b> instead.
\$.info.address.town	NULL	Error	Not an object or array. Use <b>JSON_VALUE</b> instead.
\$.info."address"	N'{ "town":"Bristol", "county":"Avon", "country":"England" }'	N'{ "town":"Bristol", "county":"Avon", "country":"England" }'	N/a
\$.info.tags	N'[ "Sport", "Water polo"]'	N'[ "Sport", "Water polo"]'	N/a
\$.info.type[0]	NULL	Error	Not an array.
\$.info.none	NULL	Error	Property does not exist.

## Using **JSON\_QUERY** with **FOR JSON**

**JSON\_QUERY** returns a valid JSON fragment. As a result, **FOR JSON** doesn't escape special characters in the **JSON\_QUERY** return value.

If you're returning results with **FOR JSON**, and you're including data that's already in JSON format (in a column or as the result of an expression), wrap the JSON data with **JSON\_QUERY** without the *path* parameter.

## Examples

### Example 1

The following example shows how to return a JSON fragment from a `CustomFields` column in query results.

```
SELECT PersonID,FullName,  
    JSON_QUERY(CustomFields,'$.OtherLanguages') AS Languages  
FROM Application.People
```

## Example 2

The following example shows how to include JSON fragments in the output of the FOR JSON clause.

```
SELECT StockItemID, StockItemName,  
    JSON_QUERY(Tags) as Tags,  
    JSON_QUERY(CONCAT('["',ValidFrom,'"","',ValidTo,'""]')) ValidityPeriod  
FROM Warehouse.StockItems  
FOR JSON PATH
```

## See Also

[JSON Path Expressions \(SQL Server\)](#)

[JSON Data \(SQL Server\)](#)

# JSON\_MODIFY (Transact-SQL)

7/28/2017 • 5 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Updates the value of a property in a JSON string and returns the updated JSON string.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
JSON_MODIFY ( expression , path , newValue )
```

## Arguments

*expression*

An expression. Typically the name of a variable or a column that contains JSON text.

**JSON\_MODIFY** returns an error if *expression* doesn't contain valid JSON.

*path*

A JSON path expression that specifies the property to update.

*path* has the following syntax:

```
[append] [ lax | strict ] $.<json path>
```

- *append*

Optional modifier that specifies that the new value should be appended to the array referenced by *<json path>*.

- *lax*

Specifies that the property referenced by *<json path>* does not have to exist. If the property is not present, JSON\_MODIFY tries to insert the new value on the specified path. Insertion may fail if the property can't be inserted on the path. If you don't specify *lax* or *strict*, *lax* is the default mode.

- *strict*

Specifies that the property referenced by *<json path>* must be in the JSON expression. If the property is not present, JSON\_MODIFY returns an error.

- *<json path>*

Specifies the path for the property to update. For more info, see [JSON Path Expressions \(SQL Server\)](#).

In SQL Server 2017 and in Azure SQL Database, you can provide a variable as the value of *path*.

**JSON\_MODIFY** returns an error if the format of *path* isn't valid.

*newValue*

The new value for the property specified by *path*.

In lax mode, JSON\_MODIFY deletes the specified key if the new value is NULL.

JSON\_MODIFY escapes all special characters in the new value if the type of the value is NVARCHAR or VARCHAR.

A text value is not escaped if it is properly formatted JSON produced by FOR JSON, JSON\_QUERY, or JSON\_MODIFY.

## Return Value

Returns the updated value of *expression* as properly formatted JSON text.

## Remarks

The JSON\_MODIFY function lets you either update the value of an existing property, insert a new key:value pair, or delete a key based on a combination of modes and provided values.

The following table compares the behavior of **JSON\_MODIFY** in lax mode and in strict mode. For more info about the optional path mode specification (lax or strict), see [JSON Path Expressions \(SQL Server\)](#).

EXISTING VALUE	PATH EXISTS	LAX MODE	STRICT MODE
Not NULL	Yes	Update the existing value.	Update the existing value.
Not NULL	No	Try to create a new key:value pair on the specified path.  This may fail. For example, if you specify the path <code>\$ .user.setting.theme</code> , JSON_MODIFY does not insert the key <code>theme</code> if the <code>\$ .user</code> or <code>\$ .user.settings</code> objects do not exist, or if settings is an array or a scalar value.	Error – INVALID_PROPERTY
NULL	Yes	Delete the existing property.	Set the existing value to null.
NULL	No	No action. The first argument is returned as the result.	Error – INVALID_PROPERTY

In lax mode, JSON\_MODIFY tries to create a new key:value pair, but in some cases it might fail.

## Examples

### Example - Basic operations

The following example shows basic operations that can be done with JSON text.

#### Query

```

DECLARE @info NVARCHAR(100)='{"name":"John","skills":["C#","SQL"]}' 

PRINT @info

-- Update name

SET @info=JSON_MODIFY(@info,'$.name','Mike')

PRINT @info

-- Insert surname

SET @info=JSON_MODIFY(@info,'$.surname','Smith')

PRINT @info

-- Delete name

SET @info=JSON_MODIFY(@info,'$.name',NULL)

PRINT @info

-- Add skill

SET @info=JSON_MODIFY(@info,'append $.skills','Azure')

PRINT @info

```

## Results

```
{
  "name": "John",
  "skills": ["C#", "SQL"]
} {
  "name": "Mike",
  "skills": ["C#", "SQL"]
} {
  "name": "Mike",
  "skills": ["C#", "SQL"],
  "surname": "Smith"
} {
  "skills": ["C#", "SQL"],
  "surname": "Smith"
} {
  "skills": ["C#", "SQL", "Azure"],
  "surname": "Smith"
}
```

## Example - Multiple updates

With JSON\_MODIFY you can update only one property. If you have to do multiple updates, you can use multiple JSON\_MODIFY calls.

## Query

```

DECLARE @info NVARCHAR(100)='{"name":"John","skills":["C#","SQL"]}' 

PRINT @info

-- Multiple updates

SET @info=JSON_MODIFY(JSON_MODIFY(JSON_MODIFY(@info,'$.name','Mike'),'$.surname','Smith'),'append
$.skills','Azure')

PRINT @info

```

## Results

```
{
  "name": "John",
  "skills": ["C#", "SQL"]
} {
  "name": "Mike",
  "skills": ["C#", "SQL", "Azure"],
  "surname": "Smith"
}
```

### Example - Rename a key

The following example shows how to rename a property in JSON text with the JSON\_MODIFY function. First you can take the value of an existing property and insert it as a new key:value pair. Then you can delete the old key by setting the value of the old property to NULL.

## Query

```

DECLARE @product NVARCHAR(100)='{"price":49.99}'

PRINT @product

-- Rename property

SET @product=
JSON_MODIFY(
  JSON_MODIFY(@product,'$.Price',CAST(JSON_VALUE(@product,'$.price') AS NUMERIC(4,2))),
  '$.price',
  NULL
)

PRINT @product

```

## Results

```
{
  "price": 49.99
} {
  "Price": 49.99
}
```

If you don't cast the new value to a numeric type, JSON\_MODIFY treats it as text and surrounds it with double quotes.

### Example - Increment a value

The following example shows how to increment the value of a property in JSON text with the JSON\_MODIFY function. First you can take the value of the existing property and insert it as a new key:value pair. Then you can

delete the old key by setting the value of the old property to NULL.

## Query

```
DECLARE @stats NVARCHAR(100)='{"click_count": 173}'  
  
PRINT @stats  
  
-- Increment value  
  
SET @stats=JSON_MODIFY(@stats,'$.click_count',  
CAST(JSON_VALUE(@stats,'$.click_count') AS INT)+1)  
  
PRINT @stats
```

## Results

```
{  
    "click_count": 173  
} {  
    "click_count": 174  
}
```

## Example - Modify a JSON object

JSON\_MODIFY treats the *newValue* argument as plain text even if it contains properly formatted JSON text. As a result, the JSON output of the function is surrounded with double quotes and all special characters are escaped, as shown in the following example.

## Query

```
DECLARE @info NVARCHAR(100)='{"name":"John","skills":["C#","SQL"]}'  
  
PRINT @info  
  
-- Update skills array  
  
SET @info=JSON_MODIFY(@info,'$.skills','[ "C#", "T-SQL", "Azure" ]')  
  
PRINT @info
```

## Results

```
{  
    "name": "John",  
    "skills": [ "C#", "SQL" ]  
} {  
    "name": "John",  
    "skills": "[ \"C#\", \"T-SQL\", \"Azure\" ]"  
}
```

To avoid automatic escaping, provide *newValue* by using the JSON\_QUERY function. JSON\_MODIFY knows that the value returned by JSON\_MODIFY is properly formatted JSON, so it doesn't escape the value.

## Query

```
DECLARE @info NVARCHAR(100)='{"name":"John","skills":["C#","SQL"]}'  
  
PRINT @info  
  
-- Update skills array  
  
SET @info=JSON_MODIFY(@info,'$.skills',JSON_QUERY('["C#","T-SQL","Azure"]'))  
  
PRINT @info
```

## Results

```
{  
    "name": "John",  
    "skills": ["C#", "SQL"]  
} {  
    "name": "John",  
    "skills": ["C#", "T-SQL", "Azure"]  
}
```

## Example - Update a JSON column

The following example updates the value of a property in a table column that contains JSON.

```
UPDATE Employee  
SET jsonCol=JSON_MODIFY(jsonCol,"$.info.address.town",'London')  
WHERE EmployeeID=17
```

## See Also

[JSON Path Expressions \(SQL Server\)](#)

[JSON Data \(SQL Server\)](#)

# Mathematical Functions (Transact-SQL)

7/6/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following scalar functions perform a calculation, usually based on input values that are provided as arguments, and return a numeric value:

ABS	DEGREES	RAND
ACOS	EXP	ROUND
ASIN	FLOOR	SIGN
ATAN	LOG	SIN
ATN2	LOG10	SQRT
CEILING	PI	SQUARE
COS	POWER	TAN
COT	RADIANS	

## NOTE

Arithmetic functions, such as ABS, CEILING, DEGREES, FLOOR, POWER, RADIANS, and SIGN, return a value having the same data type as the input value. Trigonometric and other functions, including EXP, LOG, LOG10, SQUARE, and SQRT, cast their input values to **float** and return a **float** value.

All mathematical functions, except for RAND, are deterministic functions. This means they return the same results each time they are called with a specific set of input values. RAND is deterministic only when a seed parameter is specified. For more information about function determinism, see [Deterministic and Nondeterministic Functions](#).

## See Also

[Arithmetic Operators \(Transact-SQL\)](#)

[Built-in Functions \(Transact-SQL\)](#)

# ABS (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

A mathematical function that returns the absolute (positive) value of the specified numeric expression. (`ABS` changes negative values to positive values. `ABS` has no effect on zero or positive values.)

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ABS ( numeric_expression )
```

## Arguments

*numeric\_expression*

Is an expression of the exact numeric or approximate numeric data type category.

## Return Types

Returns the same type as *numeric\_expression*.

## Examples

The following example shows the results of using the `ABS` function on three different numbers.

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

Here is the result set.

```
---- ---- ----  
1.0 .0 1.0
```

The `ABS` function can produce an overflow error when the absolute value of a number is greater than the largest number that can be represented by the specified data type. For example, the `int` data type can hold only values that range from `-2,147,483,648` to `2,147,483,647`. Computing the absolute value for the signed integer `-2,147,483,648` causes an overflow error because its absolute value is greater than the positive range for the `int` data type.

```
DECLARE @i int;  
SET @i = -2147483648;  
SELECT ABS(@i);  
GO
```

Here is the error message:

"Msg 8115, Level 16, State 2, Line 3"

"Arithmetic overflow error converting expression to data type int."

## See also

[CAST and CONVERT \(Transact-SQL\)](#)

[Data Types \(Transact-SQL\)](#)

[Mathematical Functions \(Transact-SQL\)](#)

[Built-in Functions \(Transact-SQL\)](#)

# ACOS (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

A mathematical function that returns the angle, in radians, whose cosine is the specified **float** expression; also called arccosine.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ACOS ( float_expression )
```

## Arguments

*float\_expression*

Is an expression of the type **float** or of a type that can be implicitly converted to **float**, with a value from -1 through 1. Values outside this range return NULL and report a domain error.

## Return Types

**float**

## Examples

The following example returns the ACOS of the specified number.

```
SET NOCOUNT OFF;
DECLARE @cos float;
SET @cos = -1.0;
SELECT 'The ACOS of the number is: ' + CONVERT(varchar, ACOS(@cos));
```

Here is the result set.

```
-----
The ACOS of the number is: 3.14159
(1 row(s) affected)
```

## Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the ACOS of the specified number.

```
DECLARE @cos float;
SET @cos = -1.0;
SELECT 'The ACOS of the number is: ' + CONVERT(varchar, ACOS(@cos));
```

Here is the result set.

```
-----  
The ACOS of the number is: 3.14159
```

```
(1 row(s) affected)
```

## See also

[Mathematical Functions \(Transact-SQL\)](#)

[Functions](#)

# ASIN (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the angle, in radians, whose sine is the specified **float** expression. This is also called arcsine.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ASIN ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of the type **float** or of a type that can be implicitly converted to float, with a value from -1 through 1. Values outside this range return NULL and report a domain error.

## Return types

**float**

## Examples

The following example takes a **float** expression and returns the ASIN of the specified angle.

```
/* The first value will be -1.01. This fails because the value is
outside the range.*/
DECLARE @angle float
SET @angle = -1.01
SELECT 'The ASIN of the angle is: ' + CONVERT(varchar, ASIN(@angle))
GO

-- The next value is -1.00.
DECLARE @angle float
SET @angle = -1.00
SELECT 'The ASIN of the angle is: ' + CONVERT(varchar, ASIN(@angle))
GO

-- The next value is 0.1472738.
DECLARE @angle float
SET @angle = 0.1472738
SELECT 'The ASIN of the angle is: ' + CONVERT(varchar, ASIN(@angle))
GO
```

Here is the result set.

```
-----  
.Net SqlClient Data Provider: Msg 3622, Level 16, State 1, Line 3  
A domain error occurred.
```

```
-----  
The ASIN of the angle is: -1.5708
```

```
(1 row(s) affected)
```

```
-----  
The ASIN of the angle is: 0.147811
```

```
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the arcsine of 1.00.

```
SELECT ASIN(1.00) AS asinCalc;
```

The following example returns an error, because it requests the arcsine for a value outside the allowed range.

```
SELECT ASIN(1.1472738) AS asinCalc;
```

## See also

[CEILING \(Transact-SQL\)](#)

[Mathematical Functions \(Transact-SQL\)](#)

[SET ARITHIGNORE \(Transact-SQL\)](#)

[SET ARITHABORT \(Transact-SQL\)](#)

# ATAN (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the angle in radians whose tangent is a specified **float** expression. This is also called arctangent.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
ATAN ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of the type **float** or of a type that can be implicitly converted to **float**.

## Return types

**float**

## Examples

The following example takes a **float** expression and returns the ATAN of the specified angle.

```
SELECT 'The ATAN of -45.01 is: ' + CONVERT(varchar, ATAN(-45.01))
SELECT 'The ATAN of -181.01 is: ' + CONVERT(varchar, ATAN(-181.01))
SELECT 'The ATAN of 0 is: ' + CONVERT(varchar, ATAN(0))
SELECT 'The ATAN of 0.1472738 is: ' + CONVERT(varchar, ATAN(0.1472738))
SELECT 'The ATAN of 197.1099392 is: ' + CONVERT(varchar, ATAN(197.1099392))
GO
```

Here is the result set.

```
-----  
The ATAN of -45.01 is: -1.54858
```

```
(1 row(s) affected)
```

```
-----  
The ATAN of -181.01 is: -1.56527
```

```
(1 row(s) affected)
```

```
-----  
The ATAN of 0 is: 0
```

```
(1 row(s) affected)
```

```
-----  
The ATAN of 0.1472738 is: 0.146223
```

```
(1 row(s) affected)
```

```
-----  
The ATAN of 197.1099392 is: 1.56572
```

```
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example takes a **float** expression and returns the arctangent of the specified angle.

```
SELECT ATAN(45.87) AS atanCalc1,  
       ATAN(-181.01) AS atanCalc2,  
       ATAN(0) AS atanCalc3,  
       ATAN(0.1472738) AS atanCalc4,  
       ATAN(197.1099392) AS atanCalc5;
```

Here is the result set.

```
atanCalc1 atanCalc2 atanCalc3 atanCalc4 atanCalc5
```

```
----- ----- ----- ----- -----
```

```
1.55 -1.57 0.00 0.15 1.57
```

## See also

[CEILING \(Transact-SQL\)](#)

[Mathematical Functions \(Transact-SQL\)](#)

# ATN2 (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the angle, in radians, between the positive x-axis and the ray from the origin to the point ( $y, x$ ), where  $x$  and  $y$  are the values of the two specified float expressions.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ATN2 ( float_expression , float_expression )
```

## Arguments

*float\_expression* is an [expression](#) of the **float** data type.

## Return types

**float**

## Examples

The following example calculates the `ATN2` for the specified `x` and `y` components.

```
DECLARE @x float = 35.175643, @y float = 129.44;
SELECT 'The ATN2 of the angle is: ' + CONVERT(varchar,ATN2(@x,@y ));
GO
```

Here is the result set.

```
The ATN2 of the angle is: 0.265345
(1 row(s) affected)
```

## Examples:

Azure SQL Data Warehouse and Parallel Data Warehouse

The following example calculates the `ATN2` for the specified `x` and `y` components.

```
DECLARE @x float = 35.175643, @y float = 129.44;
SELECT 'The ATN2 of the angle is: ' + CONVERT(varchar,ATN2(@x,@y ));
GO
```

Here is the result set.

```
The ATN2 of the angle is: 0.265345
(1 row(s) affected)
```

## See also

[CAST and CONVERT \(Transact-SQL\)](#)

[float and real \(Transact-SQL\)](#)

[Mathematical Functions \(Transact-SQL\)](#)

# CEILING (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the smallest integer greater than, or equal to, the specified numeric expression.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CEILING ( numeric_expression )
```

## Arguments

*numeric\_expression*

Is an [expression](#) of the exact numeric or approximate numeric data type category, except for the **bit** data type.

## Return types

Returns the same type as *numeric\_expression*.

## Examples

The following example shows positive numeric, negative, and zero values with the CEILING function.

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

Here is the result set.

```
-----
124.00      -123.00      0.00
(1 row(s) affected)
```

Azure SQL Data Warehouse and Parallel Data Warehouse

The following example shows use of positive numeric, negative, and zero values with the CEILING function.

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

Here is the result set.

```
-----
124.00      -123.00      0.00
```

## See also

## System Functions (Transact-SQL)

# COS (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Is a mathematical function that returns the trigonometric cosine of the specified angle, in radians, in the specified expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
COS ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of type **float**.

## Return types

**float**

## Examples

The following example returns the COS of the specific angle.

```
DECLARE @angle float;
SET @angle = 14.78;
SELECT 'The COS of the angle is: ' + CONVERT(varchar,COS(@angle));
GO
```

Here is the result set.

```
The COS of the angle is: -0.599465
(1 row(s) affected)
```

## Examples

Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the COS of the specific angle.

```
SELECT COS(14.76) AS cosCalc1, COS(-0.1472738) AS cosCalc2;
```

Here is the result set.

cosCalc1	cosCalc2
----------	----------

-----

-0.58 0.99

## See also

[Mathematical Functions \(Transact-SQL\)](#)

# COT (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

A mathematical function that returns the trigonometric cotangent of the specified angle, in radians, in the specified **float** expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
COT ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of type **float** or of a type that can be implicitly converted to **float**.

## Return types

**float**

## Examples

The following example returns the COT for the specific angle.

```
DECLARE @angle float;
SET @angle = 124.1332;
SELECT 'The COT of the angle is: ' + CONVERT(varchar,COT(@angle));
GO
```

Here is the result set.

```
The COT of the angle is: -0.040312
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the COT for the specific angle.

```
DECLARE @angle float;
SET @angle = 124.1332;
SELECT 'The COT of the angle is: ' + CONVERT(varchar,COT(@angle));
GO
```

Here is the result set.

```
The COT of the angle is: -0.040312
```

```
(1 row(s) affected)
```

## See also

[Mathematical Functions \(Transact-SQL\)](#)

# DEGREES (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the corresponding angle in degrees for an angle specified in radians.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
DEGREES ( numeric_expression )
```

## Arguments

*numeric\_expression*

Is an [expression](#) of the exact numeric or approximate numeric data type category, except for the **bit** data type.

## Return Code Values

Returns the same type as *numeric\_expression*.

## Examples

The following example returns the number of degrees in an angle of PI/2 radians.

```
SELECT 'The number of degrees in PI/2 radians is: ' +
CONVERT(varchar, DEGREES((PI()/2)));
GO
```

Here is the result set.

```
The number of degrees in PI/2 radians is 90
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the number of degrees in an angle of PI/2 radians.

```
SELECT 'The number of degrees in PI/2 radians is: ' +
CONVERT(varchar, DEGREES((PI()/2)));
GO
```

Here is the result set.

```
The number of degrees in PI/2 radians is 90
```

```
(1 row(s) affected)
```

## See Also

[Mathematical Functions \(Transact-SQL\)](#)

# EXP (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns the exponential value of the specified **float** expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
EXP ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of type **float** or of a type that can be implicitly converted to **float**.

## Return Types

**float**

## Remarks

The constant **e** (2.718281...), is the base of natural logarithms.

The exponent of a number is the constant **e** raised to the power of the number. For example EXP(1.0) = e^1.0 = 2.71828182845905 and EXP(10) = e^10 = 22026.4657948067.

The exponential of the natural logarithm of a number is the number itself: EXP (LOG (n)) = n. And the natural logarithm of the exponential of a number is the number itself: LOG (EXP (n)) = n.

## Examples

### A. Finding the exponent of a number

The following example declares a variable and returns the exponential value of the specified variable (`10`) with a text description.

```
DECLARE @var float  
SET @var = 10  
SELECT 'The EXP of the variable is: ' + CONVERT(varchar,EXP(@var))  
GO
```

Here is the result set.

```
-----  
The EXP of the variable is: 22026.5  
(1 row(s) affected)
```

## B. Finding exponentials and natural logarithms

The following example returns the exponential value of the natural logarithm of 20 and the natural logarithm of the exponential of 20. Because these functions are inverse functions of one another, the return value in both cases is 20.

```
SELECT EXP( LOG(20)), LOG( EXP(20))
GO
```

Here is the result set.

```
-----
20          20
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Finding the exponent of a number

The following example returns the exponential value of the specified value (10).

```
SELECT EXP(10);
```

Here is the result set.

```
-----
22026.4657948067
```

### D. Finding exponential values and natural logarithms

The following example returns the exponential value of the natural logarithm of 20 and the natural logarithm of the exponential of 20. Because these functions are inverse functions of one another, the return value in both cases is 20.

```
SELECT EXP( LOG(20)), LOG( EXP(20));
```

Here is the result set.

```
-----
20          20
```

## See Also

[Mathematical Functions \(Transact-SQL\)](#)

[LOG \(Transact-SQL\)](#)

[LOG10 \(Transact-SQL\)](#)

# FLOOR (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse

Parallel Data Warehouse

Returns the largest integer less than or equal to the specified numeric expression.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server, Azure SQL Data Warehouse, Parallel Data Warehouse  
FLOOR ( numeric_expression )
```

## Arguments

*numeric\_expression*

Is an expression of the exact numeric or approximate numeric data type category, except for the **bit** data type.

## Return Types

Returns the same type as *numeric\_expression*.

## Examples

The following example shows positive numeric, negative numeric, and currency values with the `FLOOR` function.

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

The result is the integer part of the calculated value in the same data type as *numeric\_expression*.

-----	-----	-----
123	-124	123.0000

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example shows positive numeric, negative numeric, and values with the `FLOOR` function.

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

The result is the integer part of the calculated value in the same data type as *numeric\_expression*.

-----	-----	-----
-------	-------	-------

123	-124	123
-----	------	-----

## See Also

## Mathematical Functions (Transact-SQL)

# LOG (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the natural logarithm of the specified **float** expression in SQL Server.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server  
  
LOG ( float_expression [, base ] )
```

```
-- Syntax for Azure SQL Database, Azure SQL Data Warehouse, Parallel Data Warehouse  
  
LOG ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of type **float** or of a type that can be implicitly converted to **float**.

*base*

Optional integer argument that sets the base for the logarithm.

**Applies to:** SQL Server 2012 through SQL Server 2017

## Return Types

**float**

## Remarks

By default, **LOG()** returns the natural logarithm. Starting with SQL Server 2012, you can change the base of the logarithm to another value by using the optional *base* parameter.

The natural logarithm is the logarithm to the base **e**, where **e** is an irrational constant approximately equal to 2.718281828.

The natural logarithm of the exponential of a number is the number itself:  $\text{LOG}(\text{EXP}(n)) = n$ . And the exponential of the natural logarithm of a number is the number itself:  $\text{EXP}(\text{LOG}(n)) = n$ .

## Examples

### A. Calculating the logarithm for a number.

The following example calculates the **LOG** for the specified **float** expression.

```
DECLARE @var float = 10;
SELECT 'The LOG of the variable is: ' + CONVERT(varchar, LOG(@var));
GO
```

Here is the result set.

```
-----
The LOG of the variable is: 2.30259
(1 row(s) affected)
```

## B. Calculating the logarithm of the exponent of a number.

The following example calculates the `LOG` for the exponent of a number.

```
SELECT LOG (EXP (10));
```

Here is the result set.

```
-----
10
(1 row(s) affected)
```

# Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

## C. Calculating the logarithm for a number

The following example calculates the `LOG` for the specified **float** expression.

```
SELECT LOG(10);
```

Here is the result set.

```
-----
2.30
```

## D. Calculating the logarithm of the exponent of a number

The following example calculates the `LOG` for the exponent of a number.

```
SELECT LOG(EXP (10));
```

Here is the result set.

```
-----
10.00
```

## See Also

[Mathematical Functions \(Transact-SQL\)](#)

[EXP \(Transact-SQL\)](#)

[LOG10 \(Transact-SQL\)](#)

# LOG10 (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the base-10 logarithm of the specified **float** expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
LOG10 ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of type **float** or of a type that can be implicitly converted to **float**.

## Return Types

**float**

## Remarks

The LOG10 and POWER functions are inversely related to one another. For example,  $10 ^ \text{LOG10}(n) = n$ .

## Examples

### A. Calculating the base 10 logarithm for a variable.

The following example calculates the `LOG10` of the specified variable.

```
DECLARE @var float;
SET @var = 145.175643;
SELECT 'The LOG10 of the variable is: ' + CONVERT(varchar,LOG10(@var));
GO
```

Here is the result set.

```
The LOG10 of the variable is: 2.16189
(1 row(s) affected)
```

### B. Calculating the result of raising a base-10 logarithm to a specified power.

The following example returns the result of raising a base-10 logarithm to a specified power.

```
SELECT POWER (10, LOG10(5));
```

Here is the result set.

```
-----  
5
```

```
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C: Calculating the base 10 logarithm for a value.

The following example calculates the `LOG10` of the specified value.

```
SELECT LOG10(145.175642);
```

Here is the result set.

```
-----  
2.16
```

## See Also

[Mathematical Functions \(Transact-SQL\)](#)

[POWER \(Transact-SQL\)](#)

[LOG \(Transact-SQL\)](#)

# PI (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the constant value of PI.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
PI( )
```

## Return Types

**float**

## Examples

The following example returns the value of `PI`.

```
SELECT PI();
GO
```

Here is the result set.

```
-----
3.14159265358979
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the value of `PI`.

```
SELECT PI();
GO
```

Here is the result set.

```
-----
3.14159265358979
(1 row(s) affected)
```

## See Also

## Mathematical Functions (Transact-SQL)

# POWER (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the value of the specified expression to the specified power.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
POWER ( float_expression , y )
```

## Arguments

*float\_expression*

Is an [expression](#) of type **float** or of a type that can be implicitly converted to **float**.

*y*

Is the power to which to raise *float\_expression*. *y* can be an expression of the exact numeric or approximate numeric data type category, except for the **bit** data type.

## Return Types

Returns the same type as submitted in *float\_expression*. For example, if a **decimal**(2,0) is submitted as *float\_expression*, the result returned is **decimal**(2,0).

## Examples

### A. Using POWER to return the cube of a number

The following example demonstrates raising a number to the power of 3 (the cube of the number).

```
DECLARE @input1 float;
DECLARE @input2 float;
SET @input1= 2;
SET @input2 = 2.5;
SELECT POWER(@input1, 3) AS Result1, POWER(@input2, 3) AS Result2;
```

Here is the result set.

Result1	Result2
8	15.625
(1 row(s) affected)	

### B. Using POWER to show results of data type conversion

The following example shows how the *float\_expression* preserves the data type which can return unexpected results.

```

SELECT
POWER(CAST(2.0 AS float), -100.0) AS FloatResult,
POWER(2, -100.0) AS IntegerResult,
POWER(CAST(2.0 AS int), -100.0) AS IntegerResult,
POWER(2.0, -100.0) AS Decimal1Result,
POWER(2.00, -100.0) AS Decimal2Result,
POWER(CAST(2.0 AS decimal(5,2)), -100.0) AS Decimal2Result;
GO

```

Here is the result set.

FloatResult	IntegerResult	IntegerResult	Decimal1Result	Decimal2Result	Decimal2Result
7.88860905221012E-31	0	0	0.0	0.00	0.00

## C. Using POWER

The following example returns `POWER` results for `2`.

```

DECLARE @value int, @counter int;
SET @value = 2;
SET @counter = 1;

WHILE @counter < 5
BEGIN
    SELECT POWER(@value, @counter)
    SET NOCOUNT ON
    SET @counter = @counter + 1
    SET NOCOUNT OFF
END;
GO

```

Here is the result set.

```

-----
2
(1 row(s) affected)

-----
4
(1 row(s) affected)

-----
8
(1 row(s) affected)

-----
16
(1 row(s) affected)

```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D: Using POWER to return the cube of a number

The following example shows returns `POWER` results for `2.0` to the 3rd power.

```
SELECT POWER(2.0, 3);
```

Here is the result set.

```
-----
```

```
8.0
```

## See Also

[decimal and numeric \(Transact-SQL\)](#)

[float and real \(Transact-SQL\)](#)

[int, bigint, smallint, and tinyint \(Transact-SQL\)](#)

[Mathematical Functions \(Transact-SQL\)](#)

[money and smallmoney \(Transact-SQL\)](#)

# RADIANS (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns radians when a numeric expression, in degrees, is entered.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
RADIANS ( numeric_expression )
```

## Arguments

*numeric\_expression*

Is an [expression](#) of the exact numeric or approximate numeric data type category, except for the **bit** data type.

## Return Types

Returns the same type as *numeric\_expression*.

## Examples

### A. Using RADIANS to show 0.0

The following example returns a result of `0.0` because the numeric expression to convert to radians is too small for the `RADIANS` function.

```
SELECT RADIANS(1e-307)
GO
```

Here is the result set.

```
-----
0.0
(1 row(s) affected)
```

### B. Using RADIANS to return the equivalent angle of a float expression.

The following example takes a `float` expression and returns the `RADIANS` of the specified angle.

```

-- First value is -45.01.
DECLARE @angle float
SET @angle = -45.01
SELECT 'The RADIANS of the angle is: ' +
    CONVERT(varchar, RADIANS(@angle))
GO
-- Next value is -181.01.
DECLARE @angle float
SET @angle = -181.01
SELECT 'The RADIANS of the angle is: ' +
    CONVERT(varchar, RADIANS(@angle))
GO
-- Next value is 0.00.
DECLARE @angle float
SET @angle = 0.00
SELECT 'The RADIANS of the angle is: ' +
    CONVERT(varchar, RADIANS(@angle))
GO
-- Next value is 0.1472738.
DECLARE @angle float
SET @angle = 0.1472738
SELECT 'The RADIANS of the angle is: ' +
    CONVERT(varchar, RADIANS(@angle))
GO
-- Last value is 197.1099392.
DECLARE @angle float
SET @angle = 197.1099392
SELECT 'The RADIANS of the angle is: ' +
    CONVERT(varchar, RADIANS(@angle))
GO

```

Here is the result set.

```

-----
The RADIANS of the angle is: -0.785573
(1 row(s) affected)
-----
The RADIANS of the angle is: -3.15922
(1 row(s) affected)
-----
The RADIANS of the angle is: 0
(1 row(s) affected)
-----
The RADIANS of the angle is: 0.00257041
(1 row(s) affected)
-----
The RADIANS of the angle is: 3.44022
(1 row(s) affected)

```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Using RADIANS to show 0.0

The following example returns a result of `0.0` because the numeric expression to convert to radians is too small for the `RADIANS` function.

```

SELECT RADIANS(1e-307)
GO

```

Here is the result set.

```
-----  
0.0  
(1 row(s) affected)
```

## See Also

- [CAST and CONVERT \(Transact-SQL\)](#)
- [decimal and numeric \(Transact-SQL\)](#)
- [float and real \(Transact-SQL\)](#)
- [int, bigint, smallint, and tinyint \(Transact-SQL\)](#)
- [Mathematical Functions \(Transact-SQL\)](#)
- [money and smallmoney \(Transact-SQL\)](#)

# RAND (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Returns a pseudo-random **float** value from 0 through 1, exclusive.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
RAND ( [ seed ] )
```

## Arguments

*seed*

Is an integer [expression](#) (**tinyint**, **smallint**, or **int**) that gives the seed value. If *seed* is not specified, the SQL Server Database Engine assigns a seed value at random. For a specified seed value, the result returned is always the same.

## Return Types

**float**

## Remarks

Repetitive calls of RAND() with the same seed value return the same results.

For one connection, if RAND() is called with a specified seed value, all subsequent calls of RAND() produce results based on the seeded RAND() call. For example, the following query will always return the same sequence of numbers.

```
SELECT RAND(100), RAND(), RAND()
```

## Examples

The following example produces four different random numbers that are generated by the RAND function.

```
DECLARE @counter smallint;
SET @counter = 1;
WHILE @counter < 5
BEGIN
    SELECT RAND() Random_Number
    SET @counter = @counter + 1
END;
GO
```

## See Also

## Mathematical Functions (Transact-SQL)

# ROUND (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a numeric value, rounded to the specified length or precision.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server and Azure SQL Database
```

```
ROUND ( numeric_expression , length [ ,function ] )
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse
```

```
ROUND (numeric_expression , length )
```

## Arguments

### *numeric\_expression*

Is an [expression](#) of the exact numeric or approximate numeric data type category, except for the **bit** data type.

### *length*

Is the precision to which *numeric\_expression* is to be rounded. *length* must be an expression of type **tinyint**, **smallint**, or **int**. When *length* is a positive number, *numeric\_expression* is rounded to the number of decimal positions specified by *length*. When *length* is a negative number, *numeric\_expression* is rounded on the left side of the decimal point, as specified by *length*.

### *function*

Is the type of operation to perform. *function* must be **tinyint**, **smallint**, or **int**. When *function* is omitted or has a value of 0 (default), *numeric\_expression* is rounded. When a value other than 0 is specified, *numeric\_expression* is truncated.

## Return Types

Returns the following data types.

EXPRESSION RESULT	RETURN TYPE
<b>tinyint</b>	<b>int</b>
<b>smallint</b>	<b>int</b>
<b>int</b>	<b>int</b>
<b>bigint</b>	<b>bigint</b>

EXPRESSION RESULT	RETURN TYPE
<b>decimal</b> and <b>numeric</b> category (p, s)	<b>decimal(p, s)</b>
<b>money</b> and <b>smallmoney</b> category	<b>money</b>
<b>float</b> and <b>real</b> category	<b>float</b>

## Remarks

ROUND always returns a value. If *length* is negative and larger than the number of digits before the decimal point, ROUND returns 0.

EXAMPLE	RESULT
ROUND(748.58, -4)	0

ROUND returns a rounded *numeric\_expression*, regardless of data type, when *length* is a negative number.

EXAMPLES	RESULT
ROUND(748.58, -1)	750.00
ROUND(748.58, -2)	700.00
ROUND(748.58, -3)	Results in an arithmetic overflow, because 748.58 defaults to decimal(5,2), which cannot return 1000.00.
To round up to 4 digits, change the data type of the input. For example:  SELECT ROUND(CAST (748.58 AS decimal (6,2)), -3);	1000.00

## Examples

### A. Using ROUND and estimates

The following example shows two expressions that demonstrate by using `ROUND` the last digit is always an estimate.

```
SELECT ROUND(123.9994, 3), ROUND(123.9995, 3);
GO
```

Here is the result set.

```
-----
```

```
123.9990    124.0000
```

### B. Using ROUND and rounding approximations

The following example shows rounding and approximations.

```
SELECT ROUND(123.4545, 2);
GO
SELECT ROUND(123.45, -2);
GO
```

Here is the result set.

```
-----
123.4500
(1 row(s) affected)
-----
100.00
(1 row(s) affected)
```

### C. Using ROUND to truncate

The following example uses two `SELECT` statements to demonstrate the difference between rounding and truncation. The first statement rounds the result. The second statement truncates the result.

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

Here is the result set.

```
-----
151.00
(1 row(s) affected)
-----
150.00
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D. Using ROUND and estimates

The following example shows two expressions that demonstrate by using `ROUND` the last digit is always an estimate.

```
SELECT ROUND(123.994999, 3), ROUND(123.995444, 3);
```

Here is the result set.

```
-----
123.995000 123.995444
```

### E. Using ROUND and rounding approximations

The following example shows rounding and approximations.

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

Here is the result set.

```
----- -----
```

```
123.45 100.00
```

## See Also

[CEILING \(Transact-SQL\)](#)

[Data Types \(Transact-SQL\)](#)

[Expressions \(Transact-SQL\)](#)

[FLOOR \(Transact-SQL\)](#)

[Mathematical Functions \(Transact-SQL\)](#)

# SIGN (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the positive (+1), zero (0), or negative (-1) sign of the specified expression.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
SIGN ( numeric_expression )
```

## Arguments

*numeric\_expression*

Is an [expression](#) of the exact numeric or approximate numeric data type category, except for the **bit** data type.

## Return Types

SPECIFIED EXPRESSION	RETURN TYPE
<b>bigint</b>	<b>bigint</b>
<b>int/smallint/tinyint</b>	<b>int</b>
<b>money/smallmoney</b>	<b>money</b>
<b>numeric/decimal</b>	<b>numeric/decimal</b>
<b>Other types</b>	<b>float</b>

## Examples

The following example returns the SIGN values of numbers from -1 to 1.

```
DECLARE @value real
SET @value = -1
WHILE @value < 2
BEGIN
    SELECT SIGN(@value)
    SET NOCOUNT ON
    SELECT @value = @value + 1
    SET NOCOUNT OFF
END
SET NOCOUNT OFF
GO
```

Here is the result set.

```
(1 row(s) affected)
```

```
-----  
-1.0
```

```
(1 row(s) affected)
```

```
-----  
0.0
```

```
(1 row(s) affected)
```

```
-----  
1.0
```

```
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the SIGN values of three numbers.

```
SELECT SIGN(-125), SIGN(0), SIGN(564);
```

Here is the result set.

```
-----
```

```
-1 0 1
```

## See Also

[Mathematical Functions \(Transact-SQL\)](#)

# SIN (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the trigonometric sine of the specified angle, in radians, and in an approximate numeric, **float**, expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SIN ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of type **float** or of a type that can be implicitly converted to float.

## Return Types

**float**

## Examples

The following example calculates the SIN for a specified angle.

```
DECLARE @angle float;
SET @angle = 45.175643;
SELECT 'The SIN of the angle is: ' + CONVERT(varchar,SIN(@angle));
GO
```

Here is the result set.

```
The SIN of the angle is: 0.929607
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example calculates the sine for a specified angle.

```
SELECT SIN(45.175643);
```

Here is the result set.

```
-----
0.929607
```

## See Also

[Mathematical Functions \(Transact-SQL\)](#)

# SQRT (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the square root of the specified float value.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
SQRT ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of type **float** or of a type that can be implicitly converted to float.

## Return Types

**float**

## Examples

The following example returns the square root of numbers between `1.00` and `10.00`.

```
DECLARE @myvalue float;
SET @myvalue = 1.00;
WHILE @myvalue < 10.00
BEGIN
    SELECT SQRT(@myvalue);
    SET @myvalue = @myvalue + 1
END;
GO
```

Here is the result set.

```
-----  
1.0  
-----  
1.4142135623731  
-----  
1.73205080756888  
-----  
2.0  
-----  
2.23606797749979  
-----  
2.44948974278318  
-----  
2.64575131106459  
-----  
2.82842712474619  
-----  
3.0
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the square root of numbers `1.00` and `10.00`.

```
SELECT SQRT(1.00), SQRT(10.00);
```

Here is the result set.

```
-----  
1.00 3.16
```

## See Also

[Mathematical Functions \(Transact-SQL\)](#)

# SQUARE (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the square of the specified float value.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server, Azure SQL Data Warehouse, Parallel Data Warehouse

SQUARE ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of type **float** or of a type that can be implicitly converted to float.

## Return Types

**float**

## Examples

The following example returns the volume of a cylinder having a radius of `1` inch and a height of `5` inches.

```
DECLARE @h float, @r float;
SET @h = 5;
SET @r = 1;
SELECT PI()* SQUARE(@r)* @h AS 'Cyl Vol';
```

Here is the result set.

```
Cyl Vol
-----
15.707963267948966
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the square of each value in the `volume` column in the `containers` table.

```
-- Uses AdventureWorks

CREATE TABLE Containers (
    ID int NOT NULL,
    Name varchar(20),
    Volume float(24));

INSERT INTO Containers VALUES (1, 'Cylinder', '125.22');
INSERT INTO Containers VALUES (2, 'Cube', '23.98');

SELECT Name, SQUARE(Volume) AS VolSquared
FROM Containers;
```

Here is the result set.

Name	VolSquared
----- -----	
Cylinder	15680.05
Cube	575.04

## See Also

[Mathematical Functions \(Transact-SQL\)](#)

# TAN (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the tangent of the input expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
TAN ( float_expression )
```

## Arguments

*float\_expression*

Is an [expression](#) of type **float** or of a type that can be implicitly converted to **float**, interpreted as number of radians.

## Return Types

**float**

## Examples

The following example returns the tangent of `PI()/2`.

```
SELECT TAN(PI()/2);
```

Here is the result set.

```
-----  
1.6331778728383844E+16
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the tangent of `.45`.

```
SELECT TAN(.45);
```

Here is the result set.

```
-----  
0.48
```

## See Also

## Mathematical Functions (Transact-SQL)

# Logical Functions - CHOOSE (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the item at the specified index from a list of values in SQL Server.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CHOOSE ( index, val_1, val_2 [, val_n ] )
```

## Arguments

### *index*

Is an integer expression that represents a 1-based index into the list of the items following it.

If the provided index value has a numeric data type other than **int**, then the value is implicitly converted to an integer. If the index value exceeds the bounds of the array of values, then CHOOSE returns null.

### *val\_1 ... val\_n*

List of comma separated values of any data type.

## Return Types

Returns the data type with the highest precedence from the set of types passed to the function. For more information, see [Data Type Precedence \(Transact-SQL\)](#).

## Remarks

CHOOSE acts like an index into an array, where the array is composed of the arguments that follow the index argument. The index argument determines which of the following values will be returned.

## Examples

The following example returns the third item from the list of values that is provided.

```
SELECT CHOOSE ( 3, 'Manager', 'Director', 'Developer', 'Tester' ) AS Result;
```

Here is the result set.

Result
-----
Developer
(1 row(s) affected)

The following example returns a simple character string based on the value in the `ProductCategoryID` column.

```
USE AdventureWorks2012;
GO
SELECT ProductCategoryID, CHOOSE (ProductCategoryID, 'A','B','C','D','E') AS Expression1
FROM Production.ProductCategory;
```

Here is the result set.

ProductCategoryID	Expression1
3	C
1	A
2	B
4	D

(4 row(s) affected)

The following example returns the quarter in which an employee was hired. The MONTH function is used to return the month value from the column `HireDate`.

```
USE AdventureWorks2012;
GO
SELECT JobTitle, HireDate, CHOOSE(MONTH(HireDate),'Winter','Winter',
'Spring','Spring','Spring','Summer','Summer',
'Summer','Autumn','Autumn','Autumn','Winter') AS
Quarter_Hired
FROM HumanResources.Employee
WHERE YEAR(HireDate) > 2005
ORDER BY YEAR(HireDate);
```

Here is the result set.

JobTitle	HireDate	Quarter_Hired
Sales Representative	2006-11-01	Autumn
European Sales Manager	2006-05-18	Spring
Sales Representative	2006-07-01	Summer
Sales Representative	2006-07-01	Summer
Sales Representative	2007-07-01	Summer
Pacific Sales Manager	2007-04-15	Spring
Sales Representative	2007-07-01	Summer

## See Also

[IIF \(Transact-SQL\)](#)

# Logical Functions - IIF (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns one of two values, depending on whether the Boolean expression evaluates to true or false in SQL Server.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
IIF ( boolean_expression, true_value, false_value )
```

## Arguments

*boolean\_expression*

A valid Boolean expression.

If this argument is not a Boolean expression, then a syntax error is raised.

*true\_value*

Value to return if *boolean\_expression* evaluates to true.

*false\_value*

Value to return if *boolean\_expression* evaluates to false.

## Return Types

Returns the data type with the highest precedence from the types in *true\_value* and *false\_value*. For more information, see [Data Type Precedence \(Transact-SQL\)](#).

## Remarks

IIF is a shorthand way for writing a CASE expression. It evaluates the Boolean expression passed as the first argument, and then returns either of the other two arguments based on the result of the evaluation. That is, the *true\_value* is returned if the Boolean expression is true, and the *false\_value* is returned if the Boolean expression is false or unknown. *true\_value* and *false\_value* can be of any type. The same rules that apply to the CASE expression for Boolean expressions, null handling, and return types also apply to IIF. For more information, see [CASE \(Transact-SQL\)](#).

The fact that IIF is translated into CASE also has an impact on other aspects of the behavior of this function. Since CASE expressions can be nested only up to the level of 10, IIF statements can also be nested only up to the maximum level of 10. Also, IIF is remoted to other servers as a semantically equivalent CASE expression, with all the behaviors of a remoted CASE expression.

## Examples

### A. Simple IIF example

```
DECLARE @a int = 45, @b int = 40;
SELECT IIF ( @a > @b, 'TRUE', 'FALSE' ) AS Result;
```

Here is the result set.

```
Result
-----
TRUE

(1 row(s) affected)
```

## B. IIF with NULL constants

```
SELECT IIF ( 45 > 30, NULL, NULL ) AS Result;
```

The result of this statement is an error.

## C. IIF with NULL parameters

```
DECLARE @P INT = NULL, @S INT = NULL;
SELECT IIF ( 45 > 30, @p, @s ) AS Result;
```

Here is the result set.

```
Result
-----
NULL

(1 row(s) affected)
```

## See Also

[CASE \(Transact-SQL\)](#)

[CHOOSE \(Transact-SQL\)](#)

# Metadata Functions (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following scalar functions return information about the database and database objects:

<code>@@PROCID</code>	<code>INDEX_COL</code>
<code>APP_NAME</code>	<code>INDEXKEY_PROPERTY</code>
<code>APPLOCK_MODE</code>	<code>INDEXPROPERTY</code>
<code>APPLOCK_TEST</code>	<code>NEXT VALUE FOR</code>
<code>ASSEMBLYPROPERTY</code>	<code>OBJECT_DEFINITION</code>
<code>COL_LENGTH</code>	<code>OBJECT_ID</code>
<code>COL_NAME</code>	<code>OBJECT_NAME</code>
<code>COLUMNPROPERTY</code>	<code>OBJECT_SCHEMA_NAME</code>
<code>DATABASE_PRINCIPAL_ID</code>	<code>OBJECTPROPERTY</code>
<code>DATABASEPROPERTYEX</code>	<code>OBJECTPROPERTYEX</code>
<code>DB_ID</code>	<code>ORIGINAL_DB_NAME</code>
<code>DB_NAME</code>	<code>PARSENAME</code>
<code>FILE_ID</code>	<code>SCHEMA_ID</code>
<code>FILE_INDEX</code>	<code>SCHEMA_NAME</code>
<code>FILE_NAME</code>	<code>SCOPE_IDENTITY</code>
<code>FILEGROUP_ID</code>	<code>SERVERPROPERTY</code>
<code>FILEGROUP_NAME</code>	<code>STATS_DATE</code>
<code>FILEGROUPOPPERTY</code>	<code>TYPE_ID</code>
<code>FILEPROPERTY</code>	<code>TYPE_NAME</code>
<code>FULLTEXTCATALOGPROPERTY</code>	<code>TYPEPROPERTY</code>

FULLTEXTSERVICEPROPERTY	VERSION
-------------------------	---------

All metadata functions are nondeterministic. This means these functions do not always return the same results every time they are called, even with the same set of input values.

## See Also

[Built-in Functions \(Transact-SQL\)](#)

# @@PROCID (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the object identifier (ID) of the current Transact-SQL module. A Transact-SQL module can be a stored procedure, user-defined function, or trigger. @@PROCID cannot be specified in CLR modules or the in-process data access provider.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@PROCID
```

## Return Types

int

## Examples

The following example uses `@@PROCID` as the input parameter in the `OBJECT_NAME` function to return the name of the stored procedure in the `RAISERROR` message.

```
USE AdventureWorks2012;
GO
IF OBJECT_ID ( 'usp_FindName', 'P' ) IS NOT NULL
DROP PROCEDURE usp_FindName;
GO
CREATE PROCEDURE usp_FindName
    @lastname varchar(40) = '%',
    @firstname varchar(20) = '%'
AS
DECLARE @Count int;
DECLARE @ProcName nvarchar(128);
SELECT LastName, FirstName
FROM Person.Person
WHERE FirstName LIKE @firstname AND LastName LIKE @lastname;
SET @Count = @@ROWCOUNT;
SET @ProcName = OBJECT_NAME(@@PROCID);
RAISERROR ('Stored procedure %s returned %d rows.', 16,10, @ProcName, @Count);
GO
EXECUTE dbo.usp_FindName 'P%', 'A%';
```

## See Also

[CREATE FUNCTION \(Transact-SQL\)](#)  
[CREATE PROCEDURE \(Transact-SQL\)](#)  
[CREATE TRIGGER \(Transact-SQL\)](#)  
[Metadata Functions \(Transact-SQL\)](#)  
[sys.objects \(Transact-SQL\)](#)

[sys.sql\\_modules](#) (Transact-SQL)

[RAISERROR](#) (Transact-SQL)

# APP\_NAME (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the application name for the current session if set by the application.

## IMPORTANT

The application name is provided by the client and is not verified in any way. Do not use **APP\_NAME** as part of a security check.



## Syntax

```
APP_NAME ( )
```

## Return Types

**nvarchar(128)**

## Remarks

Use **APP\_NAME** when you want to perform different actions for different applications. For example, formatting a date differently for different applications, or returning an informational message to certain applications.

To set an application name in Management Studio, in the **Connect to Database Engine** dialog box, click **Options**. On the **Additional Connection Parameters** tab, provide an **app** attribute in the format `;app='application_name'`

## Examples

The following example checks whether the client application that initiated this process is a

`SQL Server Management Studio` session and provides a date in either US or ANSI format.

```
USE AdventureWorks2012;
GO
IF APP_NAME() = 'Microsoft SQL Server Management Studio - Query'
PRINT 'This process was started by ' + APP_NAME() + '. The date is ' + CONVERT ( varchar(100) , GETDATE(),
101) + '.';
ELSE
PRINT 'This process was started by ' + APP_NAME() + '. The date is ' + CONVERT ( varchar(100) , GETDATE(),
102) + '.';
GO
```

## See also

[System Functions \(Transact-SQL\)](#)

## Functions

# APPLOCK\_MODE (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the lock mode held by the lock owner on a particular application resource. APPLOCK\_MODE is an application lock function, and it operates on the current database. The scope of application locks is the database.



## Syntax

```
APPLOCK_MODE( 'database_principal' , 'resource_name' , 'lock_owner' )
```

## Arguments

*'database\_principal'*

Is the user, role, or application role that can be granted permissions to objects in the database. The caller of the function must be a member of *database\_principal*, dbo, or the db\_owner fixed database role in order to call the function successfully.

*'resource\_name'*

Is a lock resource name specified by the client application. The application must ensure that the resource name is unique. The specified name is hashed internally into a value that can be stored in the SQL Server lock manager. *resource\_name* is **nvarchar(255)** with no default. *resource\_name* is binary compared, and is case-sensitive regardless of the collation settings of the current database.

*'lock\_owner'*

Is the owner of the lock, which is the *lock\_owner* value when the lock was requested. *lock\_owner* is **nvarchar(32)**, and the value can be either **Transaction** (the default) or **Session**.

## Return types

**nvarchar(32)**

## Return value

Returns the lock mode held by the lock owner on a particular application resource. Lock mode can be any one of these values:

NoLock	Update	*SharedIntentExclusive
IntentShared	IntentExclusive	*UpdateIntentExclusive
Shared	Exclusive	

\*This lock mode is a combination of other lock modes and cannot be explicitly acquired by using sp\_getapplock.

# Function properties

**Nondeterministic**

**Nonindexable**

**Nonparallelizable**

## Examples

Two users (User A and User B) with separate sessions run the following sequence of Transact-SQL statements.

User A runs:

```
USE AdventureWorks2012;
GO
BEGIN TRAN;
DECLARE @result int;
EXEC @result=sp_getapplock
    @DbPrincipal='public',
    @Resource='Form1',
    @LockMode='Shared',
    @LockOwner='Transaction';
SELECT APPLOCK_MODE('public', 'Form1', 'Transaction');
GO
```

User B then runs:

```
Use AdventureWorks2012;
GO
BEGIN TRAN;
SELECT APPLOCK_MODE('public', 'Form1', 'Transaction');
--Result set: NoLock

SELECT APPLOCK_TEST('public', 'Form1', 'Shared', 'Transaction');
--Result set: 1 (Lock is grantable.)

SELECT APPLOCK_TEST('public', 'Form1', 'Exclusive', 'Transaction');
--Result set: 0 (Lock is not grantable.)
GO
```

User A then runs:

```
EXEC sp_releaseapplock @Resource='Form1', @DbPrincipal='public';
GO
```

User B then runs:

```
SELECT APPLOCK_TEST('public', 'Form1', 'Exclusive', 'Transaction');
--Result set: '1' (The lock is grantable.)
GO
```

User A and User B then run:

```
COMMIT TRAN;
GO
```

## See also

[APPLOCK\\_TEST \(Transact-SQL\)](#)  
[sp\\_getapplock \(Transact-SQL\)](#)  
[sp\\_releaseapplock \(Transact-SQL\)](#)

# APPLOCK\_TEST (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data

Warehouse Parallel Data Warehouse

Returns information about whether or not a lock can be granted on a particular application resource for a specified lock owner without acquiring the lock. APPLOCK\_TEST is an application lock function, and it operates on the current database. The scope of application locks is the database.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
APPLOCK_TEST ( 'database_principal' , 'resource_name' , 'lock_mode' , 'lock_owner' )
```

## Arguments

'*database\_principal*'

Is the user, role, or application role that can be granted permissions to objects in the database. The caller of the function must be a member of *database\_principal*, **dbo**, or the **db\_owner** fixed database role in order to call the function successfully.

'*resource\_name*'

Is a lock resource name specified by the client application. The application must ensure that the resource is unique. The specified name is hashed internally into a value that can be stored in the SQL Server lock manager.

*resource\_name* is **nvarchar(255)** with no default. *resource\_name* is binary compared and is case-sensitive, regardless of the collation settings of the current database.

'*lock\_mode*'

Is the lock mode to be obtained for a particular resource. *lock\_mode* is **nvarchar(32)** and has no default value. The value can be any of the following: **Shared**, **Update**, **IntentShared**, **IntentExclusive**, **Exclusive**.

'*lock\_owner*'

Is the owner of the lock, which is the *lock\_owner* value when the lock was requested. *lock\_owner* is **nvarchar(32)**. The value can be **Transaction** (the default) or **Session**. If default or **Transaction** is explicitly specified, APPLOCK\_TEST must be executed from within a transaction.

## Return types

**smallint**

## Return value

Returns 0 when the lock cannot be granted to the specified owner and returns 1 if the lock can be granted.

## Function properties

**Nondeterministic**

**Nonindexable**

## Nonparallelizable

# Examples

In the following example, two users (**User A** and **User B**) with separate sessions run the following sequence of Transact-SQL statements.

**User A** runs:

```
USE AdventureWorks2012;
GO
BEGIN TRAN;
DECLARE @result int;
EXEC @result=sp_getapplock
    @DbPrincipal='public',
    @Resource='Form1',
    @LockMode='Shared',
    @LockOwner='Transaction';
SELECT APPLOCK_MODE('public', 'Form1', 'Transaction');
GO
```

**User B** then runs:

```
Use AdventureWorks2012;
GO
BEGIN TRAN;
SELECT APPLOCK_MODE('public', 'Form1', 'Transaction');
--Result set: NoLock

SELECT APPLOCK_TEST('public', 'Form1', 'Shared', 'Transaction');
--Result set: 1 (Lock is grantable.)

SELECT APPLOCK_TEST('public', 'Form1', 'Exclusive', 'Transaction');
--Result set: 0 (Lock is not grantable.)
GO
```

**User A** then runs:

```
EXEC sp_releaseapplock @Resource='Form1', @DbPrincipal='public';
GO
```

**User B** then runs:

```
SELECT APPLOCK_TEST('public', 'Form1', 'Exclusive', 'Transaction');
--Result set: '1' (The lock is grantable.)
GO
```

**User A** and **User B** then both run:

```
COMMIT TRAN;
GO
```

## See also

[APPLOCK\\_MODE \(Transact-SQL\)](#)  
[sp\\_getapplock \(Transact-SQL\)](#)

[sp\\_releaseapplock \(Transact-SQL\)](#)

# ASSEMBLYPROPERTY (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns information about a property of an assembly.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
ASSEMBLYPROPERTY('assembly_name', 'property_name')
```

## Arguments

*assembly\_name*

Is the name of the assembly.

*property\_name*

Is the name of a property about which to retrieve information. *property\_name* can be one of the following values.

VALUE	DESCRIPTION
<b>CultureInfo</b>	Locale of the assembly.
<b>PublicKey</b>	Public key or public key token of the assembly.
<b>MvID</b>	Complete, compiler-generated version identification number of the assembly.
<b>VersionMajor</b>	Major component (first part) of the four-part version identification number of the assembly.
<b>VersionMinor</b>	Minor component (second part) of the four-part version identification number of the assembly.
<b>VersionBuild</b>	Build component (third part) of the four-part version identification number of the assembly.
<b>VersionRevision</b>	Revision component (fourth part) of the four-part version identification number of the assembly.
<b>SimpleName</b>	Simple name of the assembly.
<b>Architecture</b>	Processor architecture of the assembly.
<b>CLRName</b>	Canonical string that encodes the simple name, version number, culture, public key, and architecture of the assembly. This value uniquely identifies the assembly on the common language runtime (CLR) side.

## Return type

**sql\_variant**

## Examples

The following example assumes a `HelloWorld` assembly is registered in the **AdventureWorks2012** database. For more information, see [Hello World Sample](#).

```
USE AdventureWorks2012;
GO
SELECT ASSEMBLYPROPERTY ('HelloWorld' , 'PublicKey');
```

## See also

[CREATE ASSEMBLY \(Transact-SQL\)](#)

[DROP ASSEMBLY \(Transact-SQL\)](#)

# COL\_LENGTH (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the defined length, in bytes, of a column.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
COL_LENGTH ( 'table' , 'column' )
```

## Arguments

*'table'*

Is the name of the table for which to determine column length information. *table* is an expression of type **nvarchar**.

*'column'*

Is the name of the column for which to determine length. *column* is an expression of type **nvarchar**.

## Return type

**smallint**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

In SQL Server, a user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as COL\_LENGTH may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

For columns of type **varchar** declared with the **max** specifier (**varchar(max)**), COL\_LENGTH returns the value -1.

## Examples

The following example shows the return values for a column of type `varchar(40)` and a column of type `nvarchar(40)`.

```
USE AdventureWorks2012;
GO
CREATE TABLE t1(c1 varchar(40), c2 nvarchar(40) );
GO
SELECT COL_LENGTH('t1','c1')AS 'VarChar',
       COL_LENGTH('t1','c2')AS 'NVarChar';
GO
DROP TABLE t1;
```

Here is the result set.

VarChar	NVarChar
40	80

## See also

[Expressions \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[COL\\_NAME \(Transact-SQL\)](#)

[COLUMNPROPERTY \(Transact-SQL\)](#)

# COL\_NAME (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the name of a column from a specified corresponding table identification number and column identification number.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
COL_NAME ( table_id , column_id )
```

## Arguments

*table\_id*

Is the identification number of the table that contains the column. *table\_id* is of type **int**.

*column\_id*

Is the identification number of the column. *column\_id* parameter is of type **int**.

## Return types

**sysname**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as COL\_NAME may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

The *table\_id* and *column\_id* parameters together produce a column name string.

For more information about obtaining table and column identification numbers, see [OBJECT\\_ID \(Transact-SQL\)](#).

## Examples

The following example returns the name of the first column in the `Employee` table of the `AdventureWorks2012` database.

```
USE AdventureWorks2012;
GO
SET NOCOUNT OFF;
GO
SELECT COL_NAME(OBJECT_ID('HumanResources.Employee'), 1) AS 'Column Name';
GO
```

Here is the result set.

```
Column Name
```

```
-----
```

```
BusinessEntityID
```

## Examples

Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the name of the first column in a sample `Employee` table.

```
-- Uses AdventureWorks

SELECT COL_NAME(OBJECT_ID('dbo.FactResellerSales'), 1) AS FirstColumnName,
COL_NAME(OBJECT_ID('dbo.FactResellerSales'), 2) AS SecondColumnName;
```

Here is the result set.

```
ColumnName
-----
BusinessEntityID
```

## See also

[Expressions \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[COLUMNPROPERTY \(Transact-SQL\)](#)

[COL\\_LENGTH \(Transact-SQL\)](#)

# COLUMNPROPERTY (Transact-SQL)

7/31/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns information about a column or parameter.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
COLUMNPROPERTY ( id , column , property )
```

## Arguments

*id*

Is an [expression](#) that contains the identifier (ID) of the table or procedure.

*column*

Is an expression that contains the name of the column or parameter.

*property*

Is an expression that contains the information to be returned for *id*, and can be any one of the following values.

VALUE	DESCRIPTION	VALUE RETURNED
<b>AllowsNull</b>	Allows null values.	1 = TRUE 0 = FALSE NULL = Input is not valid.
<b>ColumnId</b>	Column ID value corresponding to <b>sys.columns.column_id</b> .	Column ID  <b>Note:</b> When querying multiple columns, gaps may appear in the sequence of Column ID values.
<b>FullTextTypeColumn</b>	The TYPE COLUMN in the table that holds the document type information of the <i>column</i> .	ID of the full-text TYPE COLUMN for the column passed as the second parameter of this property.
<b>IsComputed</b>	Column is a computed column.	1 = TRUE 0 = FALSE NULL = Input is not valid.

VALUE	DESCRIPTION	VALUE RETURNED
<b>IsCursorType</b>	Procedure parameter is of type CURSOR.	1 = TRUE 0 = FALSE NULL = Input is not valid.
<b>IsDeterministic</b>	Column is deterministic. This property applies only to computed columns and view columns.	1 = TRUE 0 = FALSE NULL = Input is not valid. Not a computed column or view column.
<b>IsFulltextIndexed</b>	Column has been registered for full-text indexing.	1 = TRUE 0 = FALSE NULL = Input is not valid.
<b>IsIdentity</b>	Column uses the IDENTITY property.	1 = TRUE 0 = FALSE NULL = Input is not valid.
<b>IsIdNotForRepl</b>	Column checks for the IDENTITY_INSERT setting.	1 = TRUE 0 = FALSE NULL = Input is not valid.
<b>IsIndexable</b>	Column can be indexed.	1 = TRUE 0 = FALSE NULL = Input is not valid.
<b>IsOutParam</b>	Procedure parameter is an output parameter.	1 = TRUE 0 = FALSE NULL = Input is not valid.
<b>IsPrecise</b>	Column is precise. This property applies only to deterministic columns.	1 = TRUE 0 = FALSE NULL = Input is not valid. Not a deterministic column
<b>IsRowGuidCol</b>	Column has the <b>uniqueidentifier</b> data type and is defined with the ROWGUIDCOL property.	1 = TRUE 0 = FALSE NULL = Input is not valid.
<b>IsSystemVerified</b>	The determinism and precision properties of the column can be verified by the Database Engine. This property applies only to computed columns and columns of views.	1 = TRUE 0 = FALSE NULL = Input is not valid.

VALUE	DESCRIPTION	VALUE RETURNED
<b>IsXmlIndexable</b>	The XML column can be used in an XML index.	1 = TRUE 0 = FALSE NULL = Input is not valid.
<b>Precision</b>	Length for the data type of the column or parameter.	The length of the specified column data type  -1 = <b>xml</b> or large value types  NULL = Input is not valid.
<b>Scale</b>	Scale for the data type of the column or parameter.	The scale  NULL = Input is not valid.
<b>StatisticalSemantics</b>	Column is enabled for semantic indexing.	1 = TRUE 0 = FALSE
<b>SystemDataAccess</b>	Column is derived from a function that accesses data in the system catalogs or virtual system tables of SQL Server. This property applies only to computed columns and columns of views.	1 = TRUE (Indicates read-only access.) 0 = FALSE  NULL = Input is not valid.
<b>UserDataAdapter</b>	Column is derived from a function that accesses data in user tables, including views and temporary tables, stored in the local instance of SQL Server. This property applies only to computed columns and columns of views.	1 = TRUE (Indicates read-only access.) 0 = FALSE  NULL = Input is not valid.
<b>UsesAnsiTrim</b>	ANSI_PADDING was set ON when the table was first created. This property applies only to columns or parameters of type <b>char</b> or <b>varchar</b> .	1= TRUE 0= FALSE  NULL = Input is not valid.
<b>IsSparse</b>	Column is a sparse column. For more information, see <a href="#">Use Sparse Columns</a> .	1= TRUE 0= FALSE  NULL = Input is not valid.
<b>IsColumnSet</b>	Column is a column set. For more information, see <a href="#">Use Column Sets</a> .	1= TRUE 0= FALSE  NULL = Input is not valid.

VALUE	DESCRIPTION	VALUE RETURNED
<b>GeneratedAlwaysType</b>	Is column value generated by the system. Corresponds to <b>sys.columns.generated_always_type</b>	<p><b>Applies to:</b> SQL Server 2016 through SQL Server 2017.</p> <p>0 = Not generated always 1 = Generated always as row start 2 – Generated always as row end</p>
<b>IsHidden</b>	Is column value generated by the system. Corresponds to <b>sys.columns.is_hidden</b>	<p><b>Applies to:</b> SQL Server 2017 through SQL Server 2017.</p> <p>0 = Not hidden 1 = Hidden</p>

## Return types

**int**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as COLUMNPROPERTY may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

When you check the deterministic property of a column, first test whether the column is a computed column.

**IsDeterministic** returns NULL for noncomputed columns. Computed columns can be specified as index columns.

## Examples

The following example returns the length of the `LastName` column.

```
USE AdventureWorks2012;
GO
SELECT COLUMNPROPERTY( OBJECT_ID('Person.Person'), 'LastName', 'PRECISION')AS 'Column Length';
GO
```

Here is the result set.

Column Length
---------------

-----  

50
----

## See also

[Metadata Functions \(Transact-SQL\)](#)

ms.date: "07/24/2017" [TYPEPROPERTY \(Transact-SQL\)](#)

# DATABASE\_PRINCIPAL\_ID (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the ID number of a principal in the current database. For more information about principals, see [Principals \(Database Engine\)](#).

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DATABASE_PRINCIPAL_ID ( 'principal_name' )
```

## Arguments

*principal\_name*

Is an expression of type **sysname** that represents the principal.

When *principal\_name* is omitted, the ID of the current user is returned. The parentheses are required.

## Return types

**int**

NULL when the database principal does not exist

## Remarks

DATABASE\_PRINCIPAL\_ID can be used in a select list, a WHERE clause, or anywhere an expression is allowed. For more information, see [Expressions \(Transact-SQL\)](#).

## Examples

### A. Retrieving the ID of the current user

The following example returns the database principal ID of the current user.

```
SELECT DATABASE_PRINCIPAL_ID();
GO
```

### B. Retrieving the ID of a specified database principal

The following example returns the database principal ID for the database role `db_owner`.

```
SELECT DATABASE_PRINCIPAL_ID('db_owner');
GO
```

## See also

[Principals \(Database Engine\)](#)

## Permissions Hierarchy (Database Engine)

sys.database\_principals (Transact-SQL)

# DATABASEPROPERTYEX (Transact-SQL)

9/27/2017 • 8 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the current setting of the specified database option or property for the specified database in SQL Server.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
DATABASEPROPERTYEX ( database , property )
```

## Arguments

### *database*

Is an expression that represents the name of the database for which to return the named property information. *database* is **nvarchar(128)**.

For SQL Database, must be the name of the current database. Returns NULL for all properties if a different database name is provided.

### *property*

Is an expression that represents the name of the database property to return. *property* is **varchar(128)**, and can be one of the following values. The return type is **sql\_variant**. The following table shows the base data type for each property value.

### NOTE

If the database is not started, properties that the SQL Server retrieves by accessing the database directly instead of retrieving the value from metadata will return NULL. That is, if the database has AUTO\_CLOSE set to ON, or the database is otherwise offline.

PROPERTY	DESCRIPTION	VALUE RETURNED
Collation	Default collation name for the database.	Collation name NULL = Database is not started. Base data type: <b>nvarchar(128)</b>

PROPERTY	DESCRIPTION	VALUE RETURNED
ComparisonStyle	<p>The Windows comparison style of the collation. ComparisonStyle is a bitmap that is calculated by using the following values for the possible styles.</p> <p>Ignore case : 1 Ignore accent : 2 Ignore Kana : 65536 Ignore width : 131072</p> <p>For example, the default of 196609 is the result of combining the Ignore case, Ignore Kana, and Ignore width options.</p>	<p>Returns the comparison style.</p> <p>Returns 0 for all binary collations.</p> <p>Base data type: <b>int</b></p>
Edition	The database edition or service tier.	<p><b>Applies to:</b> Azure SQL Database, SQL Data Warehouse.</p> <p>Web = Web Edition Database Business = Business Edition Database Basic Standard Premium System (for master database)</p> <p>NULL = Database is not started.</p> <p>Base data type: <b>nvarchar(64)</b></p>
IsAnsiNullDefault	Database follows ISO rules for allowing null values.	<p>1 = TRUE 0 = FALSE NULL = Input not valid</p> <p>Base data type: <b>int</b></p>
IsAnsiNullsEnabled	All comparisons to a null evaluate to unknown.	<p>1 = TRUE 0 = FALSE NULL = Input not valid</p> <p>Base data type: <b>int</b></p>

PROPERTY	DESCRIPTION	VALUE RETURNED
IsAnsiPaddingEnabled	Strings are padded to the same length before comparison or insert.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsAnsiWarningsEnabled	Error or warning messages are issued when standard error conditions occur.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsArithmeticAbortEnabled	Queries are ended when an overflow or divide-by-zero error occurs during query execution.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsAutoClose	Database shuts down cleanly and frees resources after the last user exits.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsAutoCreateStatistics	Query optimizer creates single-column statistics, as required, to improve query performance.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsAutoCreateStatisticsIncremental	Auto created single-column statistics are incremental when possible.	<b>Applies to:</b> SQL Server 2014 through SQL Server 2017.  1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>

PROPERTY	DESCRIPTION	VALUE RETURNED
IsAutoShrink	Database files are candidates for automatic periodic shrinking.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsAutoUpdateStatistics	Query optimizer updates existing statistics when they are used by a query and might be out-of-date.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsClone	Database is a schema and statistics only copy of a user database.	<b>Applies to:</b> SQL Server 2014 Service Pack 2.  1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsCloseCursorsOnCommitEnabled	Cursors that are open when a transaction is committed are closed.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsFulltextEnabled	Database is enabled for full-text and semantic indexing.	<b>Applies to:</b> SQL Server 2008 through SQL Server 2017.  1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>  <b>Note:</b> The value of this property has no effect. User databases are always enabled for full-text search. This column will be removed in a future release of SQL Server. Do not use this column in new development work, and modify applications that currently use any of these columns as soon as possible.

PROPERTY	DESCRIPTION	VALUE RETURNED
IsInStandBy	Database is online as read-only, with restore log allowed.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsLocalCursorsDefault	Cursor declarations default to LOCAL.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsMemoryOptimizedElevateToSnapshotEnabled	Memory-optimized tables are accessed using SNAPSHOT isolation when the session setting TRANSACTION ISOLATION LEVEL is set to a lower isolation level, READ COMMITTED or READ UNCOMMITTED.	<b>Applies to:</b> SQL Server 2014 through SQL Server 2017. 1 = TRUE 0 = FALSE Base data type: <b>int</b>
IsMergePublished	The tables of a database can be published for merge replication, if replication is installed.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsNullConcat	Null concatenation operand yields NULL.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsNumericRoundAbortEnabled	Errors are generated when loss of precision occurs in expressions.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsParameterizationForced	PARAMETERIZATION database SET option is FORCED.	1 = TRUE 0 = FALSE NULL = Input not valid

PROPERTY	DESCRIPTION	VALUE RETURNED
IsQuotedIdentifiersEnabled	Double quotation marks can be used on identifiers.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsPublished	The tables of the database can be published for snapshot or transactional replication, if replication is installed.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsRecursiveTriggersEnabled	Recursive firing of triggers is enabled.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsSubscribed	Database is subscribed to a publication.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsSyncWithBackup	The database is either a published database or a distribution database, and can be restored without disrupting transactional replication.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>
IsTornPageDetectionEnabled	The SQL Server Database Engine detects incomplete I/O operations caused by power failures or other system outages.	1 = TRUE 0 = FALSE NULL = Input not valid Base data type: <b>int</b>

PROPERTY	DESCRIPTION	VALUE RETURNED
IsXTPSupported	<p>Indicates whether the database supports In-Memory OLTP, i.e., creating and using memory-optimized tables and natively compiled modules.</p> <p>Specific to SQL Server:</p> <p>IsXTPSupported is independent of the existence of any MEMORY_OPTIMIZED_DATA filegroup, which is required for creating In-Memory OLTP objects.</p>	<p><b>Applies to:</b> SQL Server ( SQL Server 2014 through SQL Server 2017), SQL Database.</p> <p><b>Applies to:</b> Azure SQL Database and SQL Server starting SQL Server 2016.</p> <p>1 = TRUE 0 = FALSE NULL = Input not valid, an error, or not applicable</p> <p>Base data type: <b>int</b></p>
LCID	The Windows locale identifier (LCID) of the collation.	<p>LCID value (in decimal format).</p> <p>Base data type: <b>int</b></p>
MaxSizeInBytes	Maximum database size in bytes.	<p><b>Applies to:</b> Azure SQL Database, SQL Data Warehouse.</p> <p>1073741824 5368709120 10737418240 21474836480 32212254720 42949672960 53687091200 NULL = Database is not started</p> <p>Base data type: <b>bigint</b></p>
Recovery	Recovery model for the database.	<p>FULL = Full recovery model BULK_LOGGED = Bulk logged model SIMPLE = Simple recovery model</p> <p>Base data type: <b>nvarchar(128)</b></p>

PROPERTY	DESCRIPTION	VALUE RETURNED
ServiceObjective	Describes the performance level of the database in SQL Database or SQL Data Warehouse.	<p>Can be one of the following:</p> <p>Null: database not started</p> <p>Shared (for Web/Business editions)</p> <p>Basic</p> <p>S0</p> <p>S1</p> <p>S2</p> <p>S3</p> <p>P1</p> <p>P2</p> <p>P3</p> <p>ElasticPool</p> <p>System (for master DB)</p> <p>Base data type: <b>nvarchar(32)</b></p>
ServiceObjectiveld	The id of the service objective in SQL Database.	<b>uniqueidentifier</b> that identifies the service objective.
SQLSortOrder	SQL Server sort order ID supported in earlier versions of SQL Server.	<p>0 = Database is using Windows collation</p> <p>&gt;0 = SQL Server sort order ID</p> <p>NULL = Input not valid or database is not started</p> <p>Base data type: <b>tinyint</b></p>

PROPERTY	DESCRIPTION	VALUE RETURNED
Status	Database status.	<p>ONLINE = Database is available for query.</p> <p><b>Note:</b> The ONLINE status may be returned while the database is being opened and is not yet recovered. To identify when a database can accept connections, query the Collation property of <b>DATABASEPROPERTYEX</b>. The database can accept connections when the database collation returns a non-null value. For Always On databases, query the database_state or database_state_desc columns of sys.dm_hadr_database_replica_states.</p> <p>OFFLINE = Database was explicitly taken offline.</p> <p>RESTORING = Database is being restored.</p> <p>RECOVERING = Database is recovering and not yet ready for queries.</p> <p>SUSPECT = Database did not recover.</p> <p>EMERGENCY = Database is in an emergency, read-only state. Access is restricted to sysadmin members</p> <p>Base data type: <b>nvarchar(128)</b></p>
Updateability	Indicates whether data can be modified.	<p>READ_ONLY = Data can be read but not modified.</p> <p>READ_WRITE = Data can be read and modified.</p> <p>Base data type: <b>nvarchar(128)</b></p>
UserAccess	Indicates which users can access the database.	<p>SINGLE_USER = Only one db_owner, dbcreator, or sysadmin user at a time</p> <p>RESTRICTED_USER = Only members of db_owner, dbcreator, and sysadmin roles</p> <p>MULTI_USER = All users</p> <p>Base data type: <b>nvarchar(128)</b></p>
Version	Internal version number of the SQL Server code with which the database was created. Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	<p>Version number = Database is open.</p> <p>NULL = Database is not started.</p> <p>Base data type: <b>int</b></p>

## Return types

## sql\_variant

# Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

In SQL Server, a user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as OBJECT\_ID may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

DATABASEPROPERTYEX returns only one property setting at a time. To display multiple property settings, use the [sys.databases](#) catalog view.

## Examples

### A. Retrieving the status of the AUTO\_SHRINK database option

The following example returns the status of the AUTO\_SHRINK database option for the `AdventureWorks` database.

```
SELECT DATABASEPROPERTYEX('AdventureWorks2014', 'IsAutoShrink');
```

Here is the result set. This indicates that AUTO\_SHRINK is off.

```
-----  
0
```

### B. Retrieving the default collation for a database

The following example returns several attributes of the `AdventureWorks` database.

```
SELECT  
    DATABASEPROPERTYEX('AdventureWorks2014', 'Collation') AS Collation,  
    DATABASEPROPERTYEX('AdventureWorks2014', 'Edition') AS Edition,  
    DATABASEPROPERTYEX('AdventureWorks2014', 'ServiceObjective') AS ServiceObjective,  
    DATABASEPROPERTYEX('AdventureWorks2014', 'MaxSizeInBytes') AS MaxSizeInBytes
```

Here is the result set.

Collation	Edition	ServiceObjective	MaxSizeInBytes
SQL_Latin1_General_CI_AS	DataWarehouse	DW1000	5368709120

## See also

[ALTER DATABASE \(Transact-SQL\)](#)

[Database States](#)

[sys.databases \(Transact-SQL\)](#)

[sys.database\\_files \(Transact-SQL\)](#)

[SERVERPROPERTY \(Transact-SQL\)](#)

# DB\_ID (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns the database identification (ID) number.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
DB_ID ( [ 'database_name' ] )
```

## Arguments

*'database\_name'*

Is the database name used to return the corresponding database ID. *database\_name* is **sysname**. If *database\_name* is omitted, the current database ID is returned.

## Return types

**int**

## Permissions

If the caller of **DB\_ID** is not the owner of the database and the database is not **master** or **tempdb**, the minimum permissions required to see the corresponding row are ALTER ANY DATABASE or VIEW ANY DATABASE server-level permission, or CREATE DATABASE permission in the **master** database. The database to which the caller is connected can always be viewed in **sys.databases**.

### IMPORTANT

By default, the public role has the VIEW ANY DATABASE permission, allowing all logins to see database information. To block a login from the ability to detect a database, REVOKE the VIEW ANY DATABASE permission from public, or DENY the VIEW ANY DATABASE permission for individual logins.

## Examples

### A. Returning the database ID of the current database

The following example returns the database ID of the current database.

```
SELECT DB_ID() AS [Database ID];
GO
```

### B. Returning the database ID of a specified database

The following example returns the database ID of the **AdventureWorks2012** database.

```
SELECT DB_ID(N'AdventureWorks2008R2') AS [Database ID];
GO
```

### C. Using DB\_ID to specify the value of a system function parameter

The following example uses `DB_ID` to return the database ID of the **AdventureWorks2012** database in the system function `sys.dm_db_index_operational_stats`. The function takes a database ID as the first parameter.

```
DECLARE @db_id int;
DECLARE @object_id int;
SET @db_id = DB_ID(N'AdventureWorks2012');
SET @object_id = OBJECT_ID(N'AdventureWorks2012.Person.Address');
IF @db_id IS NULL
BEGIN;
    PRINT N'Invalid database';
END;
ELSE IF @object_id IS NULL
BEGIN;
    PRINT N'Invalid object';
END;
ELSE
BEGIN;
    SELECT * FROM sys.dm_db_index_operational_stats(@db_id, @object_id, NULL, NULL);
END;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D. Return the ID of the current database

The following example returns the database ID of the current database.

```
SELECT DB_ID();
```

### E. Return the ID of a named database.

The following example returns the database ID of the AdventureWorksDW2012 database.

```
SELECT DB_ID('AdventureWorksDW2012');
```

## See also

[DB\\_NAME \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[sys.databases \(Transact-SQL\)](#)

[sys.dm\\_db\\_index\\_operational\\_stats \(Transact-SQL\)](#)

# DB\_NAME (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the database name.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DB_NAME ( [ database_id ] )
```

## Arguments

*database\_id*

Is the identification number (ID) of the database to be returned. *database\_id* is **int**, with no default. If no ID is specified, the current database name is returned.

## Return types

**nvarchar(128)**

## Permissions

If the caller of **DB\_NAME** is not the owner of the database and the database is not **master** or **tempdb**, the minimum permissions required to see the corresponding row are **ALTER ANY DATABASE** or **VIEW ANY DATABASE** server-level permission, or **CREATE DATABASE** permission in the **master** database. The database to which the caller is connected can always be viewed in **sys.databases**.

### IMPORTANT

By default, the public role has the **VIEW ANY DATABASE** permission, allowing all logins to see database information. To block a login from the ability to detect a database, REVOKE the **VIEW ANY DATABASE** permission from public, or DENY the **VIEW ANY DATABASE** permission for individual logins.

## Examples

### A. Returning the current database name

The following example returns the name of the current database.

```
SELECT DB_NAME() AS [Current Database];
GO
```

### B. Returning the database name of a specified database ID

The following example returns the database name for database ID 3.

```
USE master;
GO
SELECT DB_NAME(3)AS [Database Name];
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Return the current database name

```
SELECT DB_NAME() AS [Current Database];
```

### D. Return the name of a database by using the database ID

The following example returns the database name and database\_id for each database.

```
SELECT DB_NAME(database_id) AS [Database], database_id
FROM sys.databases;
```

## See also

[DB\\_ID \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[sys.databases \(Transact-SQL\)](#)

# FILE\_ID (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the file identification (ID) number for the given logical file name in the current database.

## IMPORTANT

This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Use [FILE\\_INDEX](#) instead.



## Syntax

```
FILE_ID ( file_name )
```

## Arguments

*file\_name*

Is an expression of type **sysname** that represents the name of the file for which to return the file ID.

## Return Types

**smallint**

## Remarks

*file\_name* corresponds to the logical file name displayed in the name column in the sys.master\_files or sys.database\_files catalog views.

In SQL Server, the file identification number assigned to full-text catalogs is greater than 32767. Because the return type of the FILE\_ID function is **smallint**, this function cannot be used for full-text files. Use [FILE\\_INDEX](#) instead.

## Examples

The following example returns the file ID for the `AdventureWorks_Data` file.

```
USE AdventureWorks2012;
GO
SELECT FILE_ID('AdventureWorks2012_Data')AS 'File ID';
GO
```

Here is the result set.

```
File ID  
-----  
1  
(1 row(s) affected)
```

## See Also

[Deprecated Database Engine Features in SQL Server 2016](#)

[FILE\\_NAME \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[sys.database\\_files \(Transact-SQL\)](#)

[sys.master\\_files \(Transact-SQL\)](#)

# FILE\_IDEX (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the file identification (ID) number for the specified logical file name of the data, log, or full-text file in the current database.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
FILE_IDEX ( file_name )
```

## Arguments

*file\_name*

Is an expression of type **sysname** that represents the name of the file for which to return the file ID.

## Return Types

**int**

**NULL** on error

## Remarks

*file\_name* corresponds to the logical file name displayed in the **name** column in the [sys.master\\_files](#) or [sys.database\\_files](#) catalog views.

FILE\_IDEX can be used in a select list, a WHERE clause, or anywhere an expression is allowed. For more information, see [Expressions \(Transact-SQL\)](#).

## Examples

### A. Retrieving the file id of a specified file

The following example returns the file ID for the `AdventureWorks_Data` file.

```
USE AdventureWorks2012;
GO
SELECT FILE_IDEX('AdventureWorks2012_Data')AS 'File ID';
GO
```

Here is the result set.

```
File ID  
-----  
1  
(1 row(s) affected)
```

## B. Retrieving the file id when the file name is not known

The following example returns the file ID of the `AdventureWorks` log file by selecting the logical file name from the `sys.database_files` catalog view where the file type is equal to `1` (log).

```
USE AdventureWorks2012;  
GO  
SELECT FILE_ID AS 'File ID'  
FROM sys.database_files  
WHERE type = 1;
```

Here is the result set.

```
File ID  
-----  
2
```

## C. Retrieving the file id of a full-text catalog file

The following example returns the file ID of a full-text file by selecting the logical file name from the `sys.database_files` catalog view where the file type is equal to `4` (full-text). This example will return NULL if a full-text catalog does not exist.

```
SELECT FILE_ID AS 'File_ID'  
FROM sys.master_files  
WHERE type = 4;
```

## See Also

[Metadata Functions \(Transact-SQL\)](#)  
[sys.database\\_files \(Transact-SQL\)](#)  
[sys.master\\_files \(Transact-SQL\)](#)

# FILE\_NAME (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Returns the logical file name for the given file identification (ID) number.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
FILE_NAME ( file_id )
```

## Arguments

*file\_id*

Is the file identification number for which to return the file name. *file\_id* is **int**.

## Return Types

**nvarchar(128)**

## Remarks

*file\_ID* corresponds to the *file\_id* column in the *sys.master\_files* or *sys.database\_files* catalog views.

## Examples

The following example returns the file names for `file ID 1` and `file \ ID` in the AdventureWorks2012 database.

```
SELECT FILE_NAME(1) AS 'File Name 1', FILE_NAME(2) AS 'File Name 2';
GO
```

Here is the result set.

File Name 1	File Name 2
-----	-----

AdventureWorks2012_Data	AdventureWorks2012_Log
-------------------------	------------------------

(1 row(s) affected)
---------------------

## See Also

[FILE\\_INDEX \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[sys.database\\_files](#) (Transact-SQL)

[sys.master\\_files](#) (Transact-SQL)

# FILEGROUP\_ID (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the filegroup identification (ID) number for a specified filegroup name.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
FILEGROUP_ID ( 'filegroup_name' )
```

## Arguments

'*filegroup\_name*'

Is an expression of type **sysname** that represents the filegroup name for which to return the filegroup ID.

## Return Types

**int**

## Remarks

*filegroup\_name* corresponds to the **name** column in the **sys.filegroups** catalog view.

## Examples

The following example returns the filegroup ID for the filegroup named **PRIMARY** in the AdventureWorks2012 database.

```
SELECT FILEGROUP_ID('PRIMARY') AS [Filegroup ID];
GO
```

Here is the result set.

Filegroup ID
-----
1
(1 row(s) affected)

## See Also

[FILEGROUP\\_NAME \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[sys.filegroups \(Transact-SQL\)](#)

# FILEGROUP\_NAME (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the filegroup name for the specified filegroup identification (ID) number.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
FILEGROUP_NAME ( filegroup_id )
```

## Arguments

*filegroup\_id*

Is the filegroup ID number for which to return the filegroup name. *filegroup\_id* is **smallint**.

## Return Types

**nvarchar(128)**

## Remarks

*filegroup\_id* corresponds to the **data\_space\_id** column in the **sys.filegroups** catalog view.

## Examples

The following example returns the filegroup name for the filegroup ID 1 in the AdventureWorks2012 database.

```
SELECT FILEGROUP_NAME(1) AS [Filegroup Name];
GO
```

Here is the result set.

Filegroup Name
-----
PRIMARY
(1 row(s) affected)

## See Also

[Metadata Functions \(Transact-SQL\)](#)

[SELECT \(Transact-SQL\)](#)

[sys.filegroups \(Transact-SQL\)](#)

# FILEGROUPPROPERTY (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the specified filegroup property value when supplied with a filegroup and property name.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
FILEGROUPPROPERTY ( filegroup_name , property )
```

## Arguments

*filegroup\_name*

Is an expression of type **sysname** that represents the name of the filegroup for which to return the named property information.

*property*

Is an expression of type **varchar(128)** contains the name of the filegroup property to return. *property* can be one of these values.

VALUE	DESCRIPTION	VALUE RETURNED
<b>IsReadOnly</b>	Filegroup is read-only.	1 = True 0 = False NULL = Input is not valid.
<b>IsUserDefinedFG</b>	Filegroup is a user-defined filegroup.	1 = True 0 = False NULL = Input is not valid.
<b>IsDefault</b>	Filegroup is the default filegroup.	1 = True 0 = False NULL = Input is not valid.

## Return Types

**int**

## Remarks

*filegroup\_name* corresponds to the **name** column in the **sys.filegroups** catalog view.

## Examples

This example returns the setting for the `IsDefault` property for the primary filegroup in the AdventureWorks2012 database.

```
SELECT FILEGROUPPROPERTY('PRIMARY', 'IsDefault') AS 'Default Filegroup';
GO
```

Here is the result set.

```
Default Filegroup
```

```
-----
```

```
1
```

```
(1 row(s) affected)
```

## See Also

[FILEGROUP\\_ID \(Transact-SQL\)](#)  
[FILEGROUP\\_NAME \(Transact-SQL\)](#)  
[Metadata Functions \(Transact-SQL\)](#)  
[SELECT \(Transact-SQL\)](#)  
[sys.filegroups \(Transact-SQL\)](#)

# FILEPROPERTY (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the specified file name property value when a file name in the current database and a property name are specified. Returns NULL for files that are not in the current database.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
FILEPROPERTY ( file_name , property )
```

## Arguments

### *file\_name*

Is an expression that contains the name of the file associated with the current database for which to return property information. *file\_name* is **nchar(128)**.

### *property*

Is an expression that contains the name of the file property to return. *property* is **varchar(128)**, and can be one of the following values.

VALUE	DESCRIPTION	VALUE RETURNED
<b>IsReadOnly</b>	Filegroup is read-only.	1 = True 0 = False NULL = Input is not valid.
<b>IsPrimaryFile</b>	File is the primary file.	1 = True 0 = False NULL = Input is not valid.
<b>IsLogFile</b>	File is a log file.	1 = True 0 = False NULL = Input is not valid.
<b>SpaceUsed</b>	Amount of space that is used by the specified file.	Number of pages allocated in the file

## Return Types

**int**

## Remarks

*file\_name* corresponds to the **name** column in the **sys.master\_files** or **sys.database\_files** catalog view.

## Examples

The following example returns the setting for the `IsPrimaryFile` property for the `AdventureWorks_Data` file name in AdventureWorks2012 the database.

```
SELECT FILEPROPERTY('AdventureWorks2012_Data', 'IsPrimaryFile')AS [Primary File];
GO
```

Here is the result set.

```
Primary File
-----
1
(1 row(s) affected)
```

## See Also

[FILEGROUOPROPERTY \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[sp\\_spaceused \(Transact-SQL\)](#)

[sys.database\\_files \(Transact-SQL\)](#)

[sys.master\\_files \(Transact-SQL\)](#)

# FULLTEXTCATALOGPROPERTY (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns information about full-text catalog properties in SQL Server 2017.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
FULLTEXTCATALOGPROPERTY ('catalog_name' , 'property')
```

## Arguments

### NOTE

The following properties will be removed in a future release of SQL Server: **LogSize** and **PopulateStatus**. Avoid using these properties in new development work, and plan to modify applications that currently use any of them.

*catalog\_name*

Is an expression containing the name of the full-text catalog.

*property*

Is an expression containing the name of the full-text catalog property. The table lists the properties and provides descriptions of the information returned.

PROPERTY	DESCRIPTION
<b>AccentSensitivity</b>	Accent-sensitivity setting. 0 = Accent insensitive 1 = Accent sensitive
<b>IndexSize</b>	Logical size of the full-text catalog in megabytes (MB). Includes the size of semantic key phrase and document similarity indexes.  For more information, see "Remarks," later in this topic.
<b>ItemCount</b>	Number of indexed items including all full-text, keyphrase and document similarity indexes in a catalog
<b>LogSize</b>	Supported for backward compatibility only. Always returns 0.  Size, in bytes, of the combined set of error logs associated with a Microsoft Search Service full-text catalog.

PROPERTY	DESCRIPTION
<b>MergeStatus</b>	Whether a master merge is in progress.  0 = master merge not in progress  1 = master merge in progress
<b>PopulateCompletionAge</b>	The difference in seconds between the completion of the last full-text index population and 01/01/1990 00:00:00.  Only updated for full and incremental crawls. Returns 0 if no population has occurred.
<b>PopulateStatus</b>	0 = Idle  1 = Full population in progress  2 = Paused  3 = Throttled  4 = Recovering  5 = Shutdown  6 = Incremental population in progress  7 = Building index  8 = Disk is full. Paused.  9 = Change tracking
<b>UniqueKeyCount</b>	Number of unique keys in the full-text catalog.
<b>ImportStatus</b>	Whether the full-text catalog is being imported.  0 = The full-text catalog is not being imported.  1 = The full-text catalog is being imported.

## Return Types

**int**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

In SQL Server 2017, a user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as FULLTEXTCATALOGPROPERTY may return NULL if the user does not have any permission on the object. For more information, see [sp\\_help\\_fulltext\\_catalogs \(Transact-SQL\)](#).

## Remarks

FULLTEXTCATALOGPROPERTY ('catalog\_name','**IndexSize**') looks at only fragments with status 4 or 6 as shown

in [sys.fulltext\\_index\\_fragments](#). These fragments are part of the logical index. Therefore, the **IndexSize** property returns only the logical index size. During an index merge, however, the actual index size might be double its logical size. To find the actual size that is being consumed by a full-text index during a merge, use the [sp\\_spaceused](#) system stored procedure. That procedure looks at all fragments associated with a full-text index. If you restrict the growth of the full-text catalog file and do not allow enough space for the merge process, the full-text population may fail. In this case, [FULLTEXTCATALOGPROPERTY \('catalog\\_name', 'IndexSize'\)](#) returns 0 and the following error is written to the full-text log:

```
Error: 30059, Severity: 16, State: 1. A fatal error occurred during a full-text population and caused the population to be cancelled. Population type is: FULL; database name is FTS_Test (id: 13); catalog name is t1_cat (id: 5); table name t1 (id: 2105058535). Fix the errors that are logged in the full-text crawl log. Then, resume the population. The basic Transact-SQL syntax for this is: ALTER FULLTEXT INDEX ON table_name RESUME POPULATION.
```

It is important that applications do not wait in a tight loop, checking for the **PopulateStatus** property to become idle (indicating that population has completed) because this takes CPU cycles away from the database and full-text search processes, and causes timeouts. In addition, it is usually a better option to check the corresponding **PopulateStatus** property at the table level, [TableFullTextPopulateStatus](#) in the [OBJECTPROPERTYEX](#) system function. This and other new full-text properties in [OBJECTPROPERTYEX](#) provide more granular information about full-text indexing tables. For more information, see [OBJECTPROPERTYEX \(Transact-SQL\)](#).

## Examples

The following example returns the number of full-text indexed items in a full-text catalog named `Cat_Desc`.

```
USE AdventureWorks2012;
GO
SELECT fulltextcatalogproperty('Cat_Desc', 'ItemCount');
GO
```

## See Also

[FULLTEXTSERVICEPROPERTY \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[sp\\_help\\_fulltext\\_catalogs \(Transact-SQL\)](#)

# FULLTEXTSERVICEPROPERTY (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Returns information related to the properties of the Full-Text Engine. These properties can be set and retrieved by using **sp\_fulltext\_service**.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
FULLTEXTSERVICEPROPERTY ('property')
```

## Arguments

*property*

Is an expression containing the name of the full-text service-level property. The table lists the properties and provides descriptions of the information returned.

### NOTE

The following properties will be removed in a future release of Microsoft SQL Server: **ConnectTimeout**, **DataTimeout**, and **ResourceUsage**. Avoid using these properties in new development work, and plan to modify applications that currently use any of them.

PROPERTY	VALUE
<b>ResourceUsage</b>	Returns 0. Supported for backward compatibility only.
<b>ConnectTimeout</b>	Returns 0. Supported for backward compatibility only.
<b>IsFulltextInstalled</b>	The full-text component is installed with the current instance of SQL Server.  0 = Full-text is not installed.  1 = Full-text is installed.  NULL = Invalid input, or error.
<b>DataTimeout</b>	Returns 0. Supported for backward compatibility only.

PROPERTY	VALUE
<b>LoadOSResources</b>	<p>Indicates whether operating system word breakers and filters are registered and used with this instance of SQL Server. By default, this property is disabled to prevent inadvertent behavior changes by updates made to the operating system (OS). Enabling use of OS resources provides access to resources for languages and document types registered with Microsoft Indexing Service, but that do not have an instance-specific resource installed. If you enable the loading of OS resources, ensure that the OS resources are trusted signed binaries; otherwise, they cannot be loaded when <b>VerifySignature</b> is set to 1.</p> <p>0 = Use only filters and word breakers specific to this instance of SQL Server.</p> <p>1 = Load OS filters and word breakers.</p>
<b>VerifySignature</b>	<p>Specifies whether only signed binaries are loaded by the Microsoft Search Service. By default, only trusted, signed binaries are loaded.</p> <p>0 = Do not verify whether or not binaries are signed.</p> <p>1 = Verify that only trusted, signed binaries are loaded.</p>

## Return Types

**int**

## Examples

The following example checks whether only signed binaries are loaded, and the return value indicates that this verification is not occurring.

```
SELECT fulltextserviceproperty('VerifySignature');
```

Here is the result set.

```
-----
0
```

Note that to set signature verification back to its default value, 1, you can use the following `sp_fulltext_service` statement:

```
EXEC sp_fulltext_service @action='verify_signature', @value=1;
GO
```

## See Also

[FULLTEXTCATALOGPROPERTY \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[sp\\_fulltext\\_service \(Transact-SQL\)](#)

# INDEX\_COL (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the indexed column name. Returns NULL for XML indexes.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
INDEX_COL ( '[ database_name . [ schema_name ] .| schema_name ]  
           table_or_view_name', index_id , key_id )
```

## Arguments

*database\_name*

Is the name of the database.

*schema\_name*

Is the name of the schema to which the index belongs.

*table\_or\_view\_name*

Is the name of the table or indexed view. *table\_or\_view\_name* must be delimited by single quotation marks and can be fully qualified by database name and schema name.

*index\_id*

Is the ID of the index. *index\_ID* is **int**.

*key\_id*

Is the index key column position. *key\_ID* is **int**.

## Return Types

**nvarchar(128)**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as INDEX\_COL may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Examples

### A. Using INDEX\_COL to return an index column name

The following example returns the column names of the two key columns in the index

```
PK_SalesOrderDetail_SalesOrderID_LineNumber .
```

```
USE AdventureWorks2012;
GO
SELECT
    INDEX_COL (N'AdventureWorks2012.Sales.SalesOrderDetail', 1,1) AS
        [Index Column 1],
    INDEX_COL (N'AdventureWorks2012.Sales.SalesOrderDetail', 1,2) AS
        [Index Column 2]
;
GO
```

Here is the result set:

Index Column 1	Index Column 2
SalesOrderID	SalesOrderDetailID

## See Also

[Expressions \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[sys.indexes \(Transact-SQL\)](#)

[sys.index\\_columns \(Transact-SQL\)](#)

# INDEXKEY\_PROPERTY (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns information about the index key. Returns NULL for XML indexes.

## IMPORTANT

This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Instead, use [sys.index\\_columns \(Transact-SQL\)](#).



## Syntax

```
INDEXKEY_PROPERTY ( object_ID ,index_ID ,key_ID ,property )
```

## Arguments

### *object\_ID*

Is the object identification number of the table or indexed view. *object\_ID* is **int**.

### *index\_ID*

Is the index identification number. *index\_ID* is **int**.

### *key\_ID*

Is the index key column position. *key\_ID* is **int**.

### *property*

Is the name of the property for which information will be returned. *property* is a character string and can be one of the following values.

VALUE	DESCRIPTION
<b>ColumnId</b>	Column ID at the <i>key_ID</i> position of the index.
<b>IsDescending</b>	Order in which the index column is stored. 1 = Descending 0 = Ascending

## Return Types

### **int**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as INDEXKEY\_PROPERTY may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Examples

In the following example, both properties are returned for index ID `1`, key column `1` in the `Production.Location` table.

```
USE AdventureWorks2012;
GO
SELECT
    INDEXKEY_PROPERTY(OBJECT_ID('Production.Location', 'U'),
    1,1,'ColumnId') AS [Column ID],
    INDEXKEY_PROPERTY(OBJECT_ID('Production.Location', 'U'),
    1,1,'IsDescending') AS [Asc or Desc order];
```

Here is the result set:

Column ID	Asc or Desc order
1	0

(1 row(s) affected)

## See Also

[INDEX\\_COL \(Transact-SQL\)](#)  
[INDEXPROPERTY \(Transact-SQL\)](#)  
[sys.objects \(Transact-SQL\)](#)  
[sys.indexes \(Transact-SQL\)](#)  
[sys.index\\_columns \(Transact-SQL\)](#)

# INDEXPROPERTY (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the named index or statistics property value of a specified table identification number, index or statistics name, and property name. Returns NULL for XML indexes.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
INDEXPROPERTY ( object_ID , index_or_statistics_name , property )
```

## Arguments

### *object\_ID*

Is an expression that contains the object identification number of the table or indexed view for which to provide index property information. *object\_ID* is **int**.

### *index\_or\_statistics\_name*

Is an expression that contains the name of the index or statistics for which to return property information.

*index\_or\_statistics\_name* is **nvarchar(128)**.

### *property*

Is an expression that contains the name of the database property to return. *property* is **varchar(128)**, and can be one of these values.

### NOTE

Unless noted otherwise, NULL is returned when *property* is not a valid property name, *object\_ID* is not a valid object ID, *object\_ID* is an unsupported object type for the specified property, or the caller does not have permission to view the object's metadata.

PROPERTY	DESCRIPTION	VALUE
<b>IndexDepth</b>	Depth of the index.	Number of index levels. NULL = XML index or input is not valid.
<b>IndexFillFactor</b>	Fill factor value used when the index was created or last rebuilt.	Fill factor
<b>IndexID</b>	Index ID of the index on a specified table or indexed view.	Index ID
<b>IsAutoStatistics</b>	Statistics were generated by the AUTO_CREATE_STATISTICS option of ALTER DATABASE.	1 = True 0 = False or XML index.

PROPERTY	DESCRIPTION	VALUE
<b>IsClustered</b>	Index is clustered.	1 = True 0 = False or XML index.
<b>IsDisabled</b>	Index is disabled.	1 = True 0 = False NULL = Input is not valid.
<b>IsFulltextKey</b>	Index is the full-text and semantic indexing key for a table.	<b>Applies to:</b> SQL Server 2008 through SQL Server 2017. 1 = True 0 = False or XML index. NULL = Input is not valid.
<b>IsHypothetical</b>	Index is hypothetical and cannot be used directly as a data access path. Hypothetical indexes hold column-level statistics and are maintained and used by Database Engine Tuning Advisor.	1 = True 0 = False or XML index NULL = Input is not valid.
<b>IsPadIndex</b>	Index specifies space to leave open on each interior node.	<b>Applies to:</b> SQL Server 2008 through SQL Server 2017. 1 = True 0 = False or XML index.
<b>IsPageLockDisallowed</b>	Page-locking value set by the ALLOW_PAGE_LOCKS option of ALTER INDEX.	<b>Applies to:</b> SQL Server 2008 through SQL Server 2017. 1 = Page locking is disallowed. 0 = Page locking is allowed. NULL = Input is not valid.
<b>IsRowLockDisallowed</b>	Row-locking value set by the ALLOW_ROW_LOCKS option of ALTER INDEX.	<b>Applies to:</b> SQL Server 2008 through SQL Server 2017. 1 = Row locking is disallowed. 0 = Row locking is allowed. NULL = Input is not valid.
<b>IsStatistics</b>	<i>index_or_statistics_name</i> is statistics created by the CREATE STATISTICS statement or by the AUTO_CREATE_STATISTICS option of ALTER DATABASE.	1 = True 0 = False or XML index.

PROPERTY	DESCRIPTION	VALUE
<b>IsUnique</b>	Index is unique.	1 = True 0 = False or XML index.
<b>IsColumnstore</b>	Index is an xVelocity memory optimized columnstore index.	<b>Applies to:</b> SQL Server 2012 through SQL Server 2017. 1 = True 0 = False

## Return Types

**int**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as INDEXPROPERTY may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Examples

The following example returns the values for the **IsClustered**, **IndexDepth**, and **IndexFillFactor** properties for the `PK Employee \ BusinessEntityID` index of the `Employee` table in the AdventureWorks2012 database.

```
SELECT
    INDEXPROPERTY(OBJECT_ID('HumanResources.Employee'),
        'PK_Employee_BusinessEntityID','IsClustered')AS [Is Clustered],
    INDEXPROPERTY(OBJECT_ID('HumanResources.Employee'),
        'PK_Employee_BusinessEntityID','IndexDepth') AS [Index Depth],
    INDEXPROPERTY(OBJECT_ID('HumanResources.Employee'),
        'PK_Employee_BusinessEntityID','IndexFillFactor') AS [Fill Factor];
```

Here is the result set:

Is Clustered	Index Depth	Fill Factor
1	2	0

(1 row(s) affected)

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example examines the properties of one of the indexes on the `FactResellerSales` table.

```
-- Uses AdventureWorks

SELECT
INDEXPROPERTY(OBJECT_ID('dbo.FactResellerSales'),
'ClusteredIndex_6d10fa223e5e4c1fbba087e29e16a7a2','IsClustered') AS [Is Clustered],
INDEXPROPERTY(OBJECT_ID('dbo.FactResellerSales'),
'ClusteredIndex_6d10fa223e5e4c1fbba087e29e16a7a2','IsColumnstore') AS [Is Columnstore Index],
INDEXPROPERTY(OBJECT_ID('dbo.FactResellerSales'),
'ClusteredIndex_6d10fa223e5e4c1fbba087e29e16a7a2','IndexFillFactor') AS [Fill Factor];
GO
```

## See Also

[CREATE INDEX \(Transact-SQL\)](#)

[Statistics](#)

[sys.indexes \(Transact-SQL\)](#)

[sys.index\\_columns \(Transact-SQL\)](#)

[sys.stats \(Transact-SQL\)](#)

[sys.stats\\_columns \(Transact-SQL\)](#)

# NEXT VALUE FOR (Transact-SQL)

3/24/2017 • 8 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Generates a sequence number from the specified sequence object.

For a complete discussion of both creating and using sequences, see [Sequence Numbers](#). Use [sp\\_sequence\\_get\\_range](#) to generate reserve a range of sequence numbers.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
NEXT VALUE FOR [ database_name . ] [ schema_name . ] sequence_name
      [ OVER (over_order_by_clause) ]
```

## Arguments

*database\_name*

The name of the database that contains the sequence object.

*schema\_name*

The name of the schema that contains the sequence object.

*sequence\_name*

The name of the sequence object that generates the number.

*over\_order\_by\_clause*

Determines the order in which the sequence value is assigned to the rows in a partition. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

Returns a number using the type of the sequence.

## Remarks

The **NEXT VALUE FOR** function can be used in stored procedures and triggers.

When the **NEXT VALUE FOR** function is used in a query or default constraint, if the same sequence object is used more than once, or if the same sequence object is used both in the statement supplying the values, and in a default constraint being executed, the same value will be returned for all columns referencing the same sequence within a row in the result set.

The **NEXT VALUE FOR** function is nondeterministic, and is only allowed in contexts where the number of generated sequence values is well defined. Below is the definition of how many values will be used for each referenced sequence object in a given statement:

- **SELECT** - For each referenced sequence object, a new value is generated once per row in the result of the statement.

- **INSERT ... VALUES** - For each referenced sequence object, a new value is generated once for each inserted row in the statement.
- **UPDATE** - For each referenced sequence object, a new value is generated for each row being updated by the statement.
- Procedural statements (such as **DECLARE**, **SET**, etc.) - For each referenced sequence object, a new value is generated for each statement.

## Limitations and Restrictions

The **NEXT VALUE FOR** function cannot be used in the following situations:

- When a database is in read-only mode.
- As an argument to a table-valued function.
- As an argument to an aggregate function.
- In subqueries including common table expressions and derived tables.
- In views, in user-defined functions, or in computed columns.
- In a statement using the **DISTINCT**, **UNION**, **UNION ALL**, **EXCEPT** or **INTERSECT** operator.
- In a statement using the **ORDER BY** clause unless **NEXT VALUE FOR ... OVER (ORDER BY ...)** is used.
- In the following clauses: **FETCH**, **OVER**, **OUTPUT**, **ON**, **PIVOT**, **UNPIVOT**, **GROUP BY**, **HAVING**, **COMPUTE**, **COMPUTE BY**, or **FOR XML**.
- In conditional expressions using **CASE**, **CHOOSE**, **COALESCE**, **IIF**, **ISNULL**, or **NULLIF**.
- In a **VALUES** clause that is not part of an **INSERT** statement.
- In the definition of a check constraint.
- In the definition of a rule or default object. (It can be used in a default constraint.)
- As a default in a user-defined table type.
- In a statement using **TOP**, **OFFSET**, or when the **ROWCOUNT** option is set.
- In the **WHERE** clause of a statement.
- In a **MERGE** statement. (Except when the **NEXT VALUE FOR** function is used in a default constraint in the target table and default is used in the **CREATE** statement of the **MERGE** statement.)

## Using a Sequence Object in a Default Constraint

When using the **NEXT VALUE FOR** function in a default constraint, the following rules apply:

- A single sequence object may be referenced from default constraints in multiple tables.
- The table and the sequence object must reside in the same database.
- The user adding the default constraint must have **REFERENCES** permission on the sequence object.
- A sequence object that is referenced from a default constraint cannot be dropped before the default constraint is dropped.
- The same sequence number is returned for all columns in a row if multiple default constraints use the same sequence object, or if the same sequence object is used both in the statement supplying the values, and in a

default constraint being executed.

- References to the **NEXT VALUE FOR** function in a default constraint cannot specify the **OVER** clause.
- A sequence object that is referenced in a default constraint can be altered.
- In the case of an `INSERT ... SELECT` or `INSERT ... EXEC` statement where the data being inserted comes from a query using an **ORDER BY** clause, the values being returned by the **NEXT VALUE FOR** function will be generated in the order specified by the **ORDER BY** clause.

## Using a Sequence Object with an OVER ORDER BY Clause

The **NEXT VALUE FOR** function supports generating sorted sequence values by applying the **OVER** clause to the **NEXT VALUE FOR** call. By using the **OVER** clause, a user is guaranteed that the values being returned are generated in the order of the **OVER** clause's **ORDER BY** subclause. The following additional rules apply when using the **NEXT VALUE FOR** function with the **OVER** clause:

- Multiple calls to the **NEXT VALUE FOR** function for the same sequence generator in a single statement must all use the same **OVER** clause definition.
- Multiple calls to the **NEXT VALUE FOR** function that reference different sequence generators in a single statement can have different **OVER** clause definitions.
- An **OVER** clause applied to the **NEXT VALUE FOR** function does not support the **PARTITION BY** sub clause.
- If all calls to the **NEXT VALUE FOR** function in a **SELECT** statement specifies the **OVER** clause, an **ORDER BY** clause may be used in the **SELECT** statement.
- The **OVER** clause is allowed with the **NEXT VALUE FOR** function when used in a **SELECT** statement or `INSERT ... SELECT ...` statement. Use of the **OVER** clause with the **NEXT VALUE FOR** function is not allowed in **UPDATE** or **MERGE** statements.
- If another process is accessing the sequence object at the same time, the numbers returned could have gaps.

## Metadata

For information about sequences, query the [sys.sequences](#) catalog view.

## Security

### Permissions

Requires **UPDATE** permission on the sequence object or the schema of the sequence. For an example of granting permission, see example F later in this topic.

### Ownership Chaining

Sequence objects support ownership chaining. If the sequence object has the same owner as the calling stored procedure, trigger, or table (having a sequence object as a default constraint), no permission check is required on the sequence object. If the sequence object is not owned by the same user as the calling stored procedure, trigger, or table, a permission check is required on the sequence object.

When the **NEXT VALUE FOR** function is used as a default value in a table, users require both **INSERT** permission on the table, and **UPDATE** permission on the sequence object, to insert data using the default.

- If the default constraint has the same owner as the sequence object, no permissions are required on the sequence object when the default constraint is called.

- If the default constraint and the sequence object are not owned by the same user, permissions are required on the sequence object even if it is called through the default constraint.

## Audit

To audit the **NEXT VALUE FOR** function, monitor the SCHEMA\_OBJECT\_ACCESS\_GROUP.

## Examples

For examples of both creating sequences and using the **NEXT VALUE FOR** function to generate sequence numbers, see [Sequence Numbers](#).

The following examples use a sequence named `CountBy1` in a schema named `Test`. Execute the following statement to create the `Test.CountBy1` sequence. Examples C and E use the **AdventureWorks2012** database, so the `CountBy1` sequence is created in that database.

```
USE AdventureWorks2012 ;
GO

CREATE SCHEMA Test;
GO

CREATE SEQUENCE Test.CountBy1
    START WITH 1
    INCREMENT BY 1 ;
GO
```

### A. Using a sequence in a select statement

The following example creates a sequence named `CountBy1` that increases by one every time that it is used.

```
SELECT NEXT VALUE FOR Test.CountBy1 AS FirstUse;
SELECT NEXT VALUE FOR Test.CountBy1 AS SecondUse;
```

Here is the result set.

`FirstUse`

`1`

`SecondUse`

`2`

### B. Setting a variable to the next sequence value

The following example demonstrates three ways to set a variable to the next value of a sequence number.

```
DECLARE @myvar1 bigint = NEXT VALUE FOR Test.CountBy1
DECLARE @myvar2 bigint ;
DECLARE @myvar3 bigint ;
SET @myvar2 = NEXT VALUE FOR Test.CountBy1 ;
SELECT @myvar3 = NEXT VALUE FOR Test.CountBy1 ;
SELECT @myvar1 AS myvar1, @myvar2 AS myvar2, @myvar3 AS myvar3 ;
GO
```

### C. Using a sequence with a ranking window function

```

USE AdventureWorks2012 ;
GO

SELECT NEXT VALUE FOR Test.CountBy1 OVER (ORDER BY LastName) AS ListNumber,
       FirstName, LastName
FROM Person.Contact ;
GO

```

#### D. Using the NEXT VALUE FOR function in the definition of a default constraint

Using the **NEXT VALUE FOR** function in the definition of a default constraint is supported. For an example of using **NEXT VALUE FOR** in a **CREATE TABLE** statement, see Example CSequence Numbers. The following example uses **ALTER TABLE** to add a sequence as a default to a current table.

```

CREATE TABLE Test.MyTable
(
    IDColumn nvarchar(25) PRIMARY KEY,
    name varchar(25) NOT NULL
) ;
GO

CREATE SEQUENCE Test.CounterSeq
    AS int
    START WITH 1
    INCREMENT BY 1 ;
GO

ALTER TABLE Test.MyTable
    ADD
        DEFAULT N'AdvWorks_' +
        CAST(NEXT VALUE FOR Test.CounterSeq AS NVARCHAR(20))
        FOR IDColumn;
GO

INSERT Test.MyTable (name)
VALUES ('Larry') ;
GO

SELECT * FROM Test.MyTable;
GO

```

#### E. Using the NEXT VALUE FOR function in an INSERT statement

The following example creates a table named **TestTable** and then uses the **NEXT VALUE FOR** function to insert a row.

```

CREATE TABLE Test.TestTable
    (CounterColumn int PRIMARY KEY,
     Name nvarchar(25) NOT NULL) ;
GO

INSERT Test.TestTable (CounterColumn,Name)
    VALUES (NEXT VALUE FOR Test.CountBy1, 'Syed') ;
GO

SELECT * FROM Test.TestTable;
GO

```

#### E. Using the NEXT VALUE FOR function with SELECT ... INTO

The following example uses the **SELECT ... INTO** statement to create a table named **Production.NewLocation** and uses the **NEXT VALUE FOR** function to number each row.

```
USE AdventureWorks2012 ;
GO

SELECT NEXT VALUE FOR Test.CountBy1 AS LocNumber, Name
    INTO Production.NewLocation
    FROM Production.Location ;
GO

SELECT * FROM Production.NewLocation ;
GO
```

## F. Granting permission to execute NEXT VALUE FOR

The following example grants **UPDATE** permission to a user named `AdventureWorks\Larry` permission to execute `NEXT VALUE FOR` using the `Test.CounterSeq` sequence.

```
GRANT UPDATE ON OBJECT::Test.CounterSeq TO [AdventureWorks\Larry] ;
```

## See Also

[CREATE SEQUENCE \(Transact-SQL\)](#)

[ALTER SEQUENCE \(Transact-SQL\)](#)

[Sequence Numbers](#)

# OBJECT\_DEFINITION (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the Transact-SQL source text of the definition of a specified object.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
OBJECT_DEFINITION ( object_id )
```

## Arguments

*object\_id*

Is the ID of the object to be used. *object\_id* is **int**, and assumed to represent an object in the current database context.

## Return Types

**nvarchar(max)**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as OBJECT\_DEFINITION may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

The SQL Server Database Engine assumes that *object\_id* is in the current database context. The collation of the object definition always matches that of the calling database context.

OBJECT\_DEFINITION applies to the following object types:

- C = Check constraint
- D = Default (constraint or stand-alone)
- P = SQL stored procedure
- FN = SQL scalar function
- R = Rule
- RF = Replication filter procedure
- TR = SQL trigger (schema-scoped DML trigger, or DDL trigger at either the database or server scope)

- IF = SQL inline table-valued function
- TF = SQL table-valued function
- V = View

## Permissions

System object definitions are publicly visible. The definition of user objects is visible to the object owner or grantees that have any one of the following permissions: ALTER, CONTROL, TAKE OWNERSHIP, or VIEW DEFINITION. These permissions are implicitly held by members of the **db\_owner**, **db\_ddladmin**, and **db\_securityadmin** fixed database roles.

## Examples

### A. Returning the source text of a user-defined object

The following example returns the definition of a user-defined trigger, `uAddress`, in the `Person` schema. The built-in function `OBJECT_ID` is used to return the object ID of the trigger to the `OBJECT_DEFINITION` statement.

```
USE AdventureWorks2012;
GO
SELECT OBJECT_DEFINITION (OBJECT_ID(N'Person.uAddress')) AS [Trigger Definition];
GO
```

### B. Returning the source text of a system object

The following example returns the definition of the system stored procedure `sys.sp_columns`.

```
USE AdventureWorks2012;
GO
SELECT OBJECT_DEFINITION (OBJECT_ID(N'sys.sp_columns')) AS [Object Definition];
GO
```

## See Also

- [Metadata Functions \(Transact-SQL\)](#)
- [OBJECT\\_NAME \(Transact-SQL\)](#)
- [OBJECT\\_ID \(Transact-SQL\)](#)
- [sp\\_helptext \(Transact-SQL\)](#)
- [sys.sql\\_modules \(Transact-SQL\)](#)
- [sys.server\\_sql\\_modules \(Transact-SQL\)](#)

# OBJECT\_ID (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the database object identification number of a schema-scoped object.

## IMPORTANT

Objects that are not schema-scoped, such as DDL triggers, cannot be queried by using OBJECT\_ID. For objects that are not found in the `sys.objects` catalog view, obtain the object identification numbers by querying the appropriate catalog view. For example, to return the object identification number of a DDL trigger, use

```
SELECT OBJECT_ID FROM sys.triggers WHERE name = 'DatabaseTriggerLog'` ``.
```



## Syntax

```
OBJECT_ID ( '[ database_name . [ schema_name ] . | schema_name . ]  
object_name' [ , 'object_type' ] )
```

## Arguments

'*object\_name*'

Is the object to be used. *object\_name* is either **varchar** or **nvarchar**. If *object\_name* is **varchar**, it is implicitly converted to **nvarchar**. Specifying the database and schema names is optional.

'*object\_type*'

Is the schema-scoped object type. *object\_type* is either **varchar** or **nvarchar**. If *object\_type* is **varchar**, it is implicitly converted to **nvarchar**. For a list of object types, see the **type** column in [sys.objects \(Transact-SQL\)](#).

## Return Types

**int**

## Exceptions

For a spatial index, OBJECT\_ID returns NULL.

Returns NULL on error.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as OBJECT\_ID may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

When the parameter to a system function is optional, the current database, host computer, server user, or database user is assumed. Built-in functions must always be followed by parentheses.

When a temporary table name is specified, the database name must come before the temporary table name, unless the current database is **tempdb**. For example: `SELECT OBJECT_ID('tempdb..#mytemptable')`.

System functions can be used in the select list, in the WHERE clause, and anywhere an expression is allowed. For more information, see [Expressions \(Transact-SQL\)](#) and [WHERE \(Transact-SQL\)](#).

## Examples

### A. Returning the object ID for a specified object

The following example returns the object ID for the `Production.WorkOrder` table in the AdventureWorks2012 database.

```
USE master;
GO
SELECT OBJECT_ID(N'AdventureWorks2012.Production.WorkOrder') AS 'Object ID';
GO
```

### B. Verifying that an object exists

The following example checks for the existence of a specified table by verifying that the table has an object ID. If the table exists, it is deleted. If the table does not exist, the `DROP TABLE` statement is not executed.

```
USE AdventureWorks2012;
GO
IF OBJECT_ID ('Ndbo.AWBuildVersion', N'U') IS NOT NULL
DROP TABLE dbo.AWBuildVersion;
GO
```

### C. Using OBJECT\_ID to specify the value of a system function parameter

The following example returns information for all indexes and partitions of the `Person.Address` table in the AdventureWorks2012 database by using the [sys.dm\\_db\\_index\\_operational\\_stats](#) function.

#### IMPORTANT

When you are using the Transact-SQL functions DB\_ID and OBJECT\_ID to return a parameter value, always make sure that a valid ID is returned. If the database or object name cannot be found, such as when they do not exist or are spelled incorrectly, both functions will return NULL. The [sys.dm\\_db\\_index\\_operational\\_stats](#) function interprets NULL as a wildcard value that specifies all databases or all objects. Because this can be an unintentional operation, the examples in this section demonstrate the safe way to determine database and object IDs.

```
DECLARE @db_id int;
DECLARE @object_id int;
SET @db_id = DB_ID(N'AdventureWorks2012');
SET @object_id = OBJECT_ID(N'AdventureWorks2012.Person.Address');
IF @db_id IS NULL
BEGIN;
    PRINT N'Invalid database';
END;
ELSE IF @object_id IS NULL
BEGIN;
    PRINT N'Invalid object';
END;
ELSE
BEGIN;
    SELECT * FROM sys.dm_db_index_operational_stats(@db_id, @object_id, NULL, NULL);
END;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D: Returning the object ID for a specified object

The following example returns the object ID for the `FactFinance` table in the **AdventureWorksPDW2012** database.

```
SELECT OBJECT_ID(AdventureWorksPDW2012.dbo.FactFinance) AS 'Object ID';
```

## See Also

[Metadata Functions \(Transact-SQL\)](#)

[sys.objects \(Transact-SQL\)](#)

[sys.dm\\_db\\_index\\_operational\\_stats \(Transact-SQL\)](#)

[OBJECT\\_DEFINITION \(Transact-SQL\)](#)

[OBJECT\\_NAME \(Transact-SQL\)](#)

# OBJECT\_NAME (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the database object name for schema-scoped objects. For a list of schema-scoped objects, see [sys.objects \(Transact-SQL\)](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
OBJECT_NAME ( object_id [ , database_id ] )
```

## Arguments

*object\_id*

Is the ID of the object to be used. *object\_id* is **int** and is assumed to be a schema-scoped object in the specified database, or in the current database context.

*database\_id*

Is the ID of the database where the object is to be looked up. *database\_id* is **int**.

## Return Types

**sysname**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object. If the target database has the AUTO\_CLOSE option set to ON, the function will open the database.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as OBJECT\_NAME may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Permissions

Requires ANY permission on the object. To specify a database ID, CONNECT permission to the database is also required, or the guest account must be enabled.

## Remarks

System functions can be used in the select list, in the WHERE clause, and anywhere an expression is allowed. For more information, see [Expressions](#) and [WHERE](#).

The value returned by this system function uses the collation of the current database.

By default, the SQL Server Database Engine assumes that *object\_id* is in the context of the current database. A query that references an *object\_id* in another database returns NULL or incorrect results. For example, in the

following query the context of the current database is AdventureWorks2012. The Database Engine tries to return an object name for the specified object ID in that database instead of the database specified in the FROM clause of the query. Therefore, incorrect information is returned.

```
USE AdventureWorks2012;
GO
SELECT DISTINCT OBJECT_NAME(object_id)
FROM master.sys.objects;
GO
```

You can resolve object names in the context of another database by specifying a database ID. The following example specifies the database ID for the `master` database in the `OBJECT_SCHEMA_NAME` function and returns the correct results.

```
USE AdventureWorks2012;
GO
SELECT DISTINCT OBJECT_SCHEMA_NAME(object_id, 1) AS schema_name
FROM master.sys.objects;
GO
```

## Examples

### A. Using `OBJECT_NAME` in a `WHERE` clause

The following example returns columns from the `sys.objects` catalog view for the object specified by `OBJECT_NAME` in the `WHERE` clause of the `SELECT` statement.

```
USE AdventureWorks2012;
GO
DECLARE @MyID int;
SET @MyID = (SELECT OBJECT_ID('AdventureWorks2012.Production.Product',
    'U'));
SELECT name, object_id, type_desc
FROM sys.objects
WHERE name = OBJECT_NAME(@MyID);
GO
```

### B. Returning the object schema name and object name

The following example returns the object schema name, object name, and SQL text for all cached query plans that are not ad hoc or prepared statements.

```
SELECT DB_NAME(st.dbid) AS database_name,
    OBJECT_SCHEMA_NAME(st.objectid, st.dbid) AS schema_name,
    OBJECT_NAME(st.objectid, st.dbid) AS object_name,
    st.text AS query_text
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS st
WHERE st.objectid IS NOT NULL;
GO
```

### C. Returning three-part object names

The following example returns the database, schema, and object name along with all other columns in the `sys.dm_db_index_operational_stats` dynamic management view for all objects in all databases.

```
SELECT QUOTENAME(DB_NAME(database_id))
+ N' . '
+ QUOTENAME(OBJECT_SCHEMA_NAME(object_id, database_id))
+ N' . '
+ QUOTENAME(OBJECT_NAME(object_id, database_id))
,
*
FROM sys.dm_db_index_operational_stats(null, null, null, null);
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D. Using OBJECT\_NAME in a WHERE clause

The following example returns columns from the `sys.objects` catalog view for the object specified by `OBJECT_NAME` in the `WHERE` clause of the `SELECT` statement. (Your object number (274100017 in the example below) will be different. To test this example, look up a valid object number by executing `SELECT name, object_id FROM sys.objects;` in your database.)

```
SELECT name, object_id, type_desc
FROM sys.objects
WHERE name = OBJECT_NAME(274100017);
```

## See Also

[Metadata Functions \(Transact-SQL\)](#)

[OBJECT\\_DEFINITION \(Transact-SQL\)](#)

[OBJECT\\_ID \(Transact-SQL\)](#)

# OBJECT\_SCHEMA\_NAME (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the database schema name for schema-scoped objects. For a list of schema-scoped objects, see [sys.objects \(Transact-SQL\)](#).



## Syntax

```
OBJECT_SCHEMA_NAME ( object_id [ , database_id ] )
```

## Arguments

### *object\_id*

Is the ID of the object to be used. *object\_id* is **int** and is assumed to be a schema-scoped object in the specified database, or in the current database context.

### *database\_id*

Is the ID of the database where the object is to be looked up. *database\_id* is **int**.

## Return Types

### **sysname**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object. If the target database has the AUTO\_CLOSE option set to ON, the function will open the database.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as OBJECT\_SCHEMA\_NAME may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Permissions

Requires ANY permission on the object. To specify a database ID, CONNECT permission to the database is also required, or the guest account must be enabled.

## Remarks

System functions can be used in the select list, in the WHERE clause, and anywhere an expression is allowed. For more information, see [Expressions](#) and [WHERE](#).

The result set returned by this system function uses the collation of the current database.

If *database\_id* is not specified, the SQL Server Database Engine assumes that *object\_id* is in the context of the

current database. A query that references an *object\_id* in another database returns NULL or incorrect results. For example, in the following query the context of the current database is AdventureWorks2012. The Database Engine tries to return an object schema name for the specified object ID in that database instead of the database specified in the FROM clause of the query. Therefore, incorrect information is returned.

```
SELECT DISTINCT OBJECT_SCHEMA_NAME(object_id)
FROM master.sys.objects;
```

The following example specifies the database ID for the `master` database in the `OBJECT_SCHEMA_NAME` function and returns the correct results.

```
SELECT DISTINCT OBJECT_SCHEMA_NAME(object_id, 1) AS schema_name
FROM master.sys.objects;
```

## Examples

### A. Returning the object schema name and object name

The following example returns the object schema name, object name, and SQL text for all cached query plans that are not ad hoc or prepared statements.

```
SELECT DB_NAME(st.dbid) AS database_name,
       OBJECT_SCHEMA_NAME(st.objectid, st.dbid) AS schema_name,
       OBJECT_NAME(st.objectid, st.dbid) AS object_name,
       st.text AS query_statement
  FROM sys.dm_exec_query_stats AS qs
 CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS st
 WHERE st.objectid IS NOT NULL;
 GO
```

### B. Returning three-part object names

The following example returns the database, schema, and object name along with all other columns in the `sys.dm_db_index_operational_stats` dynamic management view for all objects in all databases.

```
SELECT QUOTENAME(DB_NAME(database_id))
      + N'.' +
      + QUOTENAME(OBJECT_SCHEMA_NAME(object_id, database_id))
      + N'.' +
      + QUOTENAME(OBJECT_NAME(object_id, database_id))
      ,
      *
  FROM sys.dm_db_index_operational_stats(null, null, null, null);
 GO
```

## See Also

[Metadata Functions \(Transact-SQL\)](#)

[OBJECT\\_DEFINITION \(Transact-SQL\)](#)

[OBJECT\\_ID \(Transact-SQL\)](#)

[OBJECT\\_NAME \(Transact-SQL\)](#)

[Securables](#)

# OBJECTPROPERTY (Transact-SQL)

9/27/2017 • 14 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns information about schema-scoped objects in the current database. For a list of schema-scoped objects, see [sys.objects \(Transact-SQL\)](#). This function cannot be used for objects that are not schema-scoped, such as data definition language (DDL) triggers and event notifications.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
OBJECTPROPERTY ( id , property )
```

## Arguments

*id*

Is an expression that represents the ID of the object in the current database. *id* is **int** and is assumed to be a schema-scoped object in the current database context.

*property*

Is an expression that represents the information to be returned for the object specified by *id*. *property* can be one of the following values.

### NOTE

Unless noted otherwise, NULL is returned when *property* is not a valid property name, *id* is not a valid object ID, *id* is an unsupported object type for the specified *property*, or the caller does not have permission to view the object's metadata.

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
CnstIsClustKey	Constraint	PRIMARY KEY constraint with a clustered index. 1 = True 0 = False
CnstIsColumn	Constraint	CHECK, DEFAULT, or FOREIGN KEY constraint on a single column. 1 = True 0 = False

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
CnstIsDeleteCascade	Constraint	<p>FOREIGN KEY constraint with the ON DELETE CASCADE option.</p> <p>1 = True</p> <p>0 = False</p>
CnstIsDisabled	Constraint	<p>Disabled constraint.</p> <p>1 = True</p> <p>0 = False</p>
CnstIsNonclustKey	Constraint	<p>PRIMARY KEY or UNIQUE constraint with a nonclustered index.</p> <p>1 = True</p> <p>0 = False</p>
CnstIsNotRepl	Constraint	<p>Constraint is defined by using the NOT FOR REPLICATION keywords.</p> <p>1 = True</p> <p>0 = False</p>
CnstIsNotTrusted	Constraint	<p>Constraint was enabled without checking existing rows; therefore, the constraint may not hold for all rows.</p> <p>1 = True</p> <p>0 = False</p>
CnstIsUpdateCascade	Constraint	<p>FOREIGN KEY constraint with the ON UPDATE CASCADE option.</p> <p>1 = True</p> <p>0 = False</p>
ExecIsAfterTrigger	Trigger	<p>AFTER trigger.</p> <p>1 = True</p> <p>0 = False</p>
ExecIsAnsiNullsOn	Transact-SQL function, Transact-SQL procedure, Transact-SQL trigger, view	<p>Setting of ANSI_NULLS at creation time.</p> <p>1 = True</p> <p>0 = False</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
ExecIsDeleteTrigger	Trigger	<p>DELETE trigger.</p> <p>1 = True</p> <p>0 = False</p>
ExecIsFirstDeleteTrigger	Trigger	<p>First trigger fired when a DELETE is executed against the table.</p> <p>1 = True</p> <p>0 = False</p>
ExecIsFirstInsertTrigger	Trigger	<p>First trigger fired when an INSERT is executed against the table.</p> <p>1 = True</p> <p>0 = False</p>
ExecIsFirstUpdateTrigger	Trigger	<p>First trigger fired when an UPDATE is executed against the table.</p> <p>1 = True</p> <p>0 = False</p>
ExecIsInsertTrigger	Trigger	<p>INSERT trigger.</p> <p>1 = True</p> <p>0 = False</p>
ExecIsInsteadOfTrigger	Trigger	<p>INSTEAD OF trigger.</p> <p>1 = True</p> <p>0 = False</p>
ExecIsLastDeleteTrigger	Trigger	<p>Last trigger fired when a DELETE is executed against the table.</p> <p>1 = True</p> <p>0 = False</p>
ExecIsLastInsertTrigger	Trigger	<p>Last trigger fired when an INSERT is executed against the table.</p> <p>1 = True</p> <p>0 = False</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
ExecIsLastUpdateTrigger	Trigger	Last trigger fired when an UPDATE is executed against the table.  1 = True  0 = False
ExecIsQuotedIdentOn	Transact-SQL function, Transact-SQL procedure, Transact-SQL trigger, view	Setting of QUOTED_IDENTIFIER at creation time.  1 = True  0 = False
ExecIsStartup	Procedure	Startup procedure.  1 = True  0 = False
ExecIsTriggerDisabled	Trigger	Disabled trigger.  1 = True  0 = False
ExecIsTriggerNotForRepl	Trigger	Trigger defined as NOT FOR REPLICATION.  1 = True  0 = False
ExecIsUpdateTrigger	Trigger	UPDATE trigger.  1 = True  0 = False
ExecIsWithNativeCompilation	Transact-SQL Procedure	<b>Applies to:</b> SQL Server 2014 through SQL Server 2017.  Procedure is natively compiled.  1 = True  0 = False  Base data type: <b>int</b>
HasAfterTrigger	Table, view	Table or view has an AFTER trigger.  1 = True  0 = False

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
HasDeleteTrigger	Table, view	<p>Table or view has a DELETE trigger.</p> <p>1 = True</p> <p>0 = False</p>
HasInsertTrigger	Table, view	<p>Table or view has an INSERT trigger.</p> <p>1 = True</p> <p>0 = False</p>
HasInsteadOfTrigger	Table, view	<p>Table or view has an INSTEAD OF trigger.</p> <p>1 = True</p> <p>0 = False</p>
HasUpdateTrigger	Table, view	<p>Table or view has an UPDATE trigger.</p> <p>1 = True</p> <p>0 = False</p>
IsAnsiNullsOn	Transact-SQL function, Transact-SQL procedure, table, Transact-SQL trigger, view	<p>Specifies that the ANSI NULLS option setting for the table is ON. This means all comparisons against a null value evaluate to UNKNOWN. This setting applies to all expressions in the table definition, including computed columns and constraints, for as long as the table exists.</p> <p>1 = True</p> <p>0 = False</p>
IsCheckCnst	Any schema-scoped object	<p>CHECK constraint.</p> <p>1 = True</p> <p>0 = False</p>
IsConstraint	Any schema-scoped object	<p>Is a single column CHECK, DEFAULT, or FOREIGN KEY constraint on a column or table.</p> <p>1 = True</p> <p>0 = False</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
IsDefault	Any schema-scoped object	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Bound default.</p> <p>1 = True</p> <p>0 = False</p>
IsDefaultCnst	Any schema-scoped object	<p>DEFAULT constraint.</p> <p>1 = True</p> <p>0 = False</p>
IsDeterministic	Function, view	<p>The determinism property of the function or view.</p> <p>1 = Deterministic</p> <p>0 = Not Deterministic</p>
IsEncrypted	Transact-SQL function, Transact-SQL procedure, table, Transact-SQL trigger, view	<p>Indicates that the original text of the module statement was converted to an obfuscated format. The output of the obfuscation is not directly visible in any of the catalog views in SQL Server 2005. Users without access to system tables or database files cannot retrieve the obfuscated text. However, the text is available to users that can either access system tables over the <a href="#">DAC port</a> or directly access database files. Also, users that can attach a debugger to the server process can retrieve the original procedure from memory at run time.</p> <p>1 = Encrypted</p> <p>0 = Not encrypted</p> <p>Base data type: <b>int</b></p>
IsExecuted	Any schema-scoped object	<p>Object can be executed (view, procedure, function, or trigger).</p> <p>1 = True</p> <p>0 = False</p>
IsExtendedProc	Any schema-scoped object	<p>Extended procedure.</p> <p>1 = True</p> <p>0 = False</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
IsForeignKey	Any schema-scoped object	<p>FOREIGN KEY constraint.</p> <p>1 = True</p> <p>0 = False</p>
IsIndexed	Table, view	<p>Table or view that has an index.</p> <p>1 = True</p> <p>0 = False</p>
IsIndexable	Table, view	<p>Table or view on which an index can be created.</p> <p>1 = True</p> <p>0 = False</p>
IsInlineFunction	Function	<p>Inline function.</p> <p>1 = Inline function</p> <p>0 = Not inline function</p>
IsMSShipped	Any schema-scoped object	<p>Object created during installation of SQL Server.</p> <p>1 = True</p> <p>0 = False</p>
IsPrimaryKey	Any schema-scoped object	<p>PRIMARY KEY constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>NULL = Not a function, or object ID is not valid.</p>
IsProcedure	Any schema-scoped object	<p>Procedure.</p> <p>1 = True</p> <p>0 = False</p>
IsQuotedIdentOn	Transact-SQL function, Transact-SQL procedure, table, Transact-SQL trigger, view, CHECK constraint, DEFAULT definition	<p>Specifies that the quoted identifier setting for the object is ON. This means double quotation marks delimit identifiers in all expressions involved in the object definition.</p> <p>1 = ON</p> <p>0 = OFF</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
IsQueue	Any schema-scoped object	Service Broker Queue  1 = True  0 = False
IsReplProc	Any schema-scoped object	Replication procedure.  1 = True  0 = False
IsRule	Any schema-scoped object	Bound rule.  1 = True  0 = False
IsScalarFunction	Function	Scalar-valued function.  1 = Scalar-valued function  0 = Not scalar-valued function
IsSchemaBound	Function, view	A schema bound function or view created by using SCHEMABINDING.  1 = Schema-bound  0 = Not schema-bound.
IsSystemTable	Table	System table.  1 = True  0 = False
IsTable	Table	Table.  1 = True  0 = False
IsTableFunction	Function	Table-valued function.  1 = Table-valued function  0 = Not table-valued function
IsTrigger	Any schema-scoped object	Trigger.  1 = True  0 = False

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
IsUniqueCnst	Any schema-scoped object	<p>UNIQUE constraint.</p> <p>1 = True</p> <p>0 = False</p>
IsUserTable	Table	<p>User-defined table.</p> <p>1 = True</p> <p>0 = False</p>
IsView	View	<p>View.</p> <p>1 = True</p> <p>0 = False</p>
OwnerId	Any schema-scoped object	<p>Owner of the object.</p> <p><b>Note:</b> The schema owner is not necessarily the object owner. For example, child objects (those where <i>parent_object_id</i> is nonnull) will always return the same owner ID as the parent.</p> <p>Nonnull = The database user ID of the object owner.</p>
TableDeleteTrigger	Table	<p>Table has a DELETE trigger.</p> <p>&gt;1 = ID of first trigger with the specified type.</p>
TableDeleteTriggerCount	Table	<p>Table has the specified number of DELETE triggers.</p> <p>&gt;0 = The number of DELETE triggers.</p>
TableFullTextMergeStatus	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Whether a table that has a full-text index that is currently in merging.</p> <p>0 = Table does not have a full-text index, or the full-text index is not in merging.</p> <p>1 = The full-text index is in merging.</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableFullTextBackgroundUpdateIndexOn	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Table has full-text background update index (autochange tracking) enabled.</p> <p>1 = TRUE</p> <p>0 = FALSE</p>
TableFulltextCatalogId	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>ID of the full-text catalog in which the full-text index data for the table resides.</p> <p>Nonzero = Full-text catalog ID, associated with the unique index that identifies the rows in a full-text indexed table.</p> <p>0 = Table does not have a full-text index.</p>
TableFulltextChangeTrackingOn	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Table has full-text change-tracking enabled.</p> <p>1 = TRUE</p> <p>0 = FALSE</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableFulltextDocsProcessed	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Number of rows processed since the start of full-text indexing. In a table that is being indexed for full-text search, all the columns of one row are considered as part of one document to be indexed.</p> <p>0 = No active crawl or full-text indexing is completed.</p> <p>&gt; 0 = One of the following (A or B): A) The number of documents processed by insert or update operations since the start of Full, Incremental, or Manual change tracking population. B) The number of rows processed by insert or update operations since change tracking with background update index population was enabled, the full-text index schema changed, the full-text catalog rebuilt, or the instance of SQL Server restarted, and so on.</p> <p>NULL = Table does not have a full-text index.</p> <p>This property does not monitor or count deleted rows.</p>
TableFulltextFailCount	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Number of rows Full-Text Search did not index.</p> <p>0 = The population has completed.</p> <p>&gt; 0 = One of the following (A or B): A) The number of documents that were not indexed since the start of Full, Incremental, and Manual Update change tracking population. B) For change tracking with background update index, the number of rows that were not indexed since the start of the population, or the restart of the population. This could be caused by a schema change, rebuild of the catalog, server restart, and so on.</p> <p>NULL = Table does not have a full-text index.</p>
TableFulltextItemCount	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Number of rows that were successfully full-text indexed.</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableFulltextKeyColumn	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>ID of the column associated with the single-column unique index that is participating in the full-text index definition.</p> <p>0 = Table does not have a full-text index.</p>
TableFulltextPendingChanges	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Number of pending change tracking entries to process.</p> <p>0 = change tracking is not enabled.</p> <p>NULL = Table does not have a full-text index.</p>
TableFulltextPopulateStatus	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>0 = Idle.</p> <p>1 = Full population is in progress.</p> <p>2 = Incremental population is in progress.</p> <p>3 = Propagation of tracked changes is in progress.</p> <p>4 = Background update index is in progress, such as autochange tracking.</p> <p>5 = Full-text indexing is throttled or paused.</p>
TableHasActiveFulltextIndex	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Table has an active full-text index.</p> <p>1 = True</p> <p>0 = False</p>
TableHasCheckCnst	Table	<p>Table has a CHECK constraint.</p> <p>1 = True</p> <p>0 = False</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableHasClustIndex	Table	<p>Table has a clustered index.</p> <p>1 = True</p> <p>0 = False</p>
TableHasDefaultCnst	Table	<p>Table has a DEFAULT constraint.</p> <p>1 = True</p> <p>0 = False</p>
TableHasDeleteTrigger	Table	<p>Table has a DELETE trigger.</p> <p>1 = True</p> <p>0 = False</p>
TableHasForeignKey	Table	<p>Table has a FOREIGN KEY constraint.</p> <p>1 = True</p> <p>0 = False</p>
TableHasForeignRef	Table	<p>Table is referenced by a FOREIGN KEY constraint.</p> <p>1 = True</p> <p>0 = False</p>
TableHasIdentity	Table	<p>Table has an identity column.</p> <p>1 = True</p> <p>0 = False</p>
TableHasIndex	Table	<p>Table has an index of any type.</p> <p>1 = True</p> <p>0 = False</p>
TableHasInsertTrigger	Table	<p>Object has an INSERT trigger.</p> <p>1 = True</p> <p>0 = False</p>
TableHasNonclustIndex	Table	<p>Table has a nonclustered index.</p> <p>1 = True</p> <p>0 = False</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableHasPrimaryKey	Table	<p>Table has a primary key.</p> <p>1 = True</p> <p>0 = False</p>
TableHasRowGuidCol	Table	<p>Table has a ROWGUIDCOL for a <b>uniqueidentifier</b> column.</p> <p>1 = True</p> <p>0 = False</p>
TableHasTextImage	Table	<p>Table has a <b>text</b>, <b>ntext</b>, or <b>image</b> column.</p> <p>1 = True</p> <p>0 = False</p>
TableHasTimestamp	Table	<p>Table has a <b>timestamp</b> column.</p> <p>1 = True</p> <p>0 = False</p>
TableHasUniqueCnst	Table	<p>Table has a UNIQUE constraint.</p> <p>1 = True</p> <p>0 = False</p>
TableHasUpdateTrigger	Table	<p>Object has an UPDATE trigger.</p> <p>1 = True</p> <p>0 = False</p>
TableHasVarDecimalStorageFormat	Table	<p>Table is enabled for <b>vardecimal</b> storage format.</p> <p>1 = True</p> <p>0 = False</p>
TableInsertTrigger	Table	<p>Table has an INSERT trigger.</p> <p>&gt;1 = ID of first trigger with the specified type.</p>
TableInsertTriggerCount	Table	<p>Table has the specified number of INSERT triggers.</p> <p>&gt;0 = The number of INSERT triggers.</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableIsFake	Table	<p>Table is not real. It is materialized internally on demand by the SQL Server Database Engine.</p> <p>1 = True</p> <p>0 = False</p>
TableIsLockedOnBulkLoad	Table	<p>Table is locked due to a <b>bcp</b> or BULK INSERT job.</p> <p>1 = True</p> <p>0 = False</p>
TableIsMemoryOptimized	Table	<p><b>Applies to:</b> SQL Server 2014 through SQL Server 2017.</p> <p>Table is memory optimized</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p> <p>For more information, see <a href="#">In-Memory OLTP (In-Memory Optimization)</a>.</p>
TableIsPinned	Table	<p>Table is pinned to be held in the data cache.</p> <p>0 = False</p> <p>This feature is not supported in SQL Server 2005 and later.</p>
TableTextInRowLimit	Table	<p>Maximum bytes allowed for text in row.</p> <p>0 if text in row option is not set.</p>
TableUpdateTrigger	Table	<p>Table has an UPDATE trigger.</p> <p>&gt; 1 = ID of first trigger with the specified type.</p>
TableUpdateTriggerCount	Table	<p>The table has the specified number of UPDATE triggers.</p> <p>&gt; 0 = The number of UPDATE triggers.</p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableHasColumnSet	Table	<p>Table has a column set.</p> <p>0 = False</p> <p>1 = True</p> <p>For more information, see <a href="#">Use Column Sets</a>.</p>
TableTemporalType	Table	<p><b>Applies to:</b> SQL Server 2016 through SQL Server 2017.</p> <p>Specifies the type of table.</p> <p>0 = non-temporal table</p> <p>1 = history table for system-versioned table</p> <p>2 = system-versioned temporal table</p>

## Return Types

**int**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as OBJECTPROPERTY may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

The Database Engine assumes that *object\_id* is in the current database context. A query that references an *object\_id* in another database will return NULL or incorrect results. For example, in the following query the current database context is the master database. The Database Engine will try to return the property value for the specified *object\_id* in that database instead of the database specified in the query. The query returns incorrect results because the view `vEmployee` is not in the master database.

```
USE master;
GO
SELECT OBJECTPROPERTY(OBJECT_ID(N'AdventureWorks2012.HumanResources.vEmployee'), 'IsView');
GO
```

OBJECTPROPERTY(*view\_id*, 'IsIndexable') may consume significant computer resources because evaluation of IsIndexable property requires the parsing of view definition, normalization, and partial optimization. Although the IsIndexable property identifies tables or views that can be indexed, the actual creation of the index still might fail if certain index key requirements are not met. For more information, see [CREATE INDEX \(Transact-SQL\)](#).

OBJECTPROPERTY(*table\_id*, 'TableHasActiveFulltextIndex') will return a value of 1 (true) when at least one column of a table is added for indexing. Full-text indexing becomes active for population as soon as the first column is added for indexing.

When a table is created, the QUOTED IDENTIFIER option is always stored as ON in the metadata of the table, even if the option is set to OFF when the table is created. Therefore, OBJECTPROPERTY(*table\_id*, 'IsQuotedIdentOn') will always return a value of 1 (true).

## Examples

### A. Verifying that an object is a table

The following example tests whether `UnitMeasure` is a table in the **AdventureWorks2012** database.

```
USE AdventureWorks2012;
GO
IF OBJECTPROPERTY (OBJECT_ID(N'Production.UnitMeasure'), 'ISTABLE') = 1
    PRINT 'UnitMeasure is a table.'
ELSE IF OBJECTPROPERTY (OBJECT_ID(N'Production.UnitMeasure'), 'ISTABLE') = 0
    PRINT 'UnitMeasure is not a table.'
ELSE IF OBJECTPROPERTY (OBJECT_ID(N'Production.UnitMeasure'), 'ISTABLE') IS NULL
    PRINT 'ERROR: UnitMeasure is not a valid object.';
GO
```

### B. Verifying that a scalar-valued user-defined function is deterministic

The following example tests whether the user-defined scalar-valued function `ufnGetProductDealerPrice`, which returns a **money** value, is deterministic.

```
USE AdventureWorks2012;
GO
SELECT OBJECTPROPERTY(OBJECT_ID('dbo.ufnGetProductDealerPrice'), 'IsDeterministic');
GO
```

The result set shows that `ufnGetProductDealerPrice` is not a deterministic function.

-----

0

### C. Finding the objects that belong to a specific schema

The following example uses the `SchemaId` property to return all the objects that belong to the schema `Production`.

```
USE AdventureWorks2012;
GO
SELECT name, object_id, type_desc
FROM sys.objects
WHERE OBJECTPROPERTY(object_id, N'SchemaId') = SCHEMA_ID(N'Production')
ORDER BY type_desc, name;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D: Verifying that an object is a table

The following example tests whether `dbo.DimReseller` is a table in the **AdventureWorksPDW2012** database.

```
-- Uses AdventureWorks

IF OBJECTPROPERTY (OBJECT_ID(N'dbo.DimReseller'), 'ISTABLE') = 1
    SELECT 'DimReseller is a table.'
ELSE
    SELECT 'DimReseller is not a table.';
GO
```

## E: Finding the tables that belong to a specific schema

The following example returns all the tables in the dbo schema.

```
-- Uses AdventureWorks

SELECT name, object_id, type_desc
FROM sys.objects
WHERE OBJECTPROPERTY(object_id, N'SchemaId') = SCHEMA_ID(N'dbo')
ORDER BY type_desc, name;
GO
```

## See Also

[COLUMNPROPERTY \(Transact-SQL\)](#)  
[Metadata Functions \(Transact-SQL\)](#)  
[OBJECTPROPERTYEX \(Transact-SQL\)](#)  
[ALTER AUTHORIZATION \(Transact-SQL\)](#)  
[TYPEPROPERTY \(Transact-SQL\)](#)  
[sys.objects \(Transact-SQL\)](#)

# OBJECTPROPERTYEX (Transact-SQL)

9/27/2017 • 17 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns information about schema-scoped objects in the current database. For a list of these objects, see [sys.objects \(Transact-SQL\)](#). OBJECTPROPERTYEX cannot be used for objects that are not schema-scoped, such as data definition language (DDL) triggers and event notifications.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
OBJECTPROPERTYEX ( id , property )
```

## Arguments

*id*

Is an expression that represents the ID of the object in the current database. *id* is **int** and is assumed to be a schema-scoped object in the current database context.

*property*

Is an expression that contains the information to be returned for the object specified by *id*. The return type is **sql\_variant**. The following table shows the base data type for each property value.

### NOTE

Unless noted otherwise, NULL is returned when *property* is not a valid property name, *id* is not a valid object ID, *id* is an unsupported object type for the specified *property*, or the caller does not have permission to view the object's metadata.

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
BaseType	Any schema-scoped object	<p>Identifies the base type of the object. When the specified object is a SYNONYM, the base type of the underlying object is returned.</p> <p>Nonnull = Object type</p> <p>Base data type: <b>char(2)</b></p>
CnstIsClustKey	Constraint	<p>PRIMARY KEY constraint with a clustered index.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
CnstIsColumn	Constraint	<p>CHECK, DEFAULT, or FOREIGN KEY constraint on a single column.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
CnstIsDeleteCascade	Constraint	<p>FOREIGN KEY constraint with the ON DELETE CASCADE option.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
CnstIsDisabled	Constraint	<p>Disabled constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
CnstIsNonclustKey	Constraint	<p>PRIMARY KEY constraint with a nonclustered index.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
CnstIsNotRepl	Constraint	<p>Constraint is defined by using the NOT FOR REPLICATION keywords.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
CnstIsNotTrusted	Constraint	<p>Constraint was enabled without checking existing rows. Therefore, the constraint may not hold for all rows.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
CnstsUpdateCascade	Constraint	<p>FOREIGN KEY constraint with the ON UPDATE CASCADE option.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsAfterTrigger	Trigger	<p>AFTER trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsAnsiNullsOn	Transact-SQL function, Transact-SQL procedure, Transact-SQL trigger, view	<p>The setting of ANSI_NULLS at creation time.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsDeleteTrigger	Trigger	<p>DELETE trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsFirstDeleteTrigger	Trigger	<p>The first trigger fired when a DELETE is executed against the table.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsFirstInsertTrigger	Trigger	<p>The first trigger fired when an INSERT is executed against the table.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
ExecIsFirstUpdateTrigger	Trigger	<p>The first trigger fired when an UPDATE is executed against the table.</p> <p>1 = True 0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsInsertTrigger	Trigger	<p>INSERT trigger.</p> <p>1 = True 0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsInsteadOfTrigger	Trigger	<p>INSTEAD OF trigger.</p> <p>1 = True 0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsLastDeleteTrigger	Trigger	<p>Last trigger fired when a DELETE is executed against the table.</p> <p>1 = True 0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsLastInsertTrigger	Trigger	<p>Last trigger fired when an INSERT is executed against the table.</p> <p>1 = True 0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsLastUpdateTrigger	Trigger	<p>Last trigger fired when an UPDATE is executed against the table.</p> <p>1 = True 0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
ExecIsQuotedIdentOn	Transact-SQL function, Transact-SQL procedure, Transact-SQL trigger, view	<p>Setting of QUOTED_IDENTIFIER at creation time.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsStartup	Procedure	<p>Startup procedure.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsTriggerDisabled	Trigger	<p>Disabled trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsTriggerNotForRepl	Trigger	<p>Trigger defined as NOT FOR REPLICATION.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsUpdateTrigger	Trigger	<p>UPDATE trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
ExecIsWithNativeCompilation	Transact-SQL Procedure	<p><b>Applies to:</b> SQL Server 2014 through SQL Server 2017.</p> <p>Procedure is natively compiled.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
HasAfterTrigger	Table, view	<p>Table or view has an AFTER trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
HasDeleteTrigger	Table, view	<p>Table or view has a DELETE trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
HasInsertTrigger	Table, view	<p>Table or view has an INSERT trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
HasInsteadOfTrigger	Table, view	<p>Table or view has an INSTEAD OF trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
HasUpdateTrigger	Table, view	<p>Table or view has an UPDATE trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsAnsiNullsOn	Transact-SQL function, Transact-SQL procedure, table, Transact-SQL trigger, view	<p>Specifies that the ANSI NULLS option setting for the table is ON, meaning all comparisons against a null value evaluate to UNKNOWN. This setting applies to all expressions in the table definition, including computed columns and constraints, for as long as the table exists.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
IsCheckCnst	Any schema-scoped object	<p>CHECK constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsConstraint	Any schema-scoped object	<p>Constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsDefault	Any schema-scoped object	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Bound default.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsDefaultCnst	Any schema-scoped object	<p>DEFAULT constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsDeterministic	Scalar and table-valued functions, view	<p>The determinism property of the function or view.</p> <p>1 = Deterministic</p> <p>0 = Not Deterministic</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
IsEncrypted	Transact-SQL function, Transact-SQL procedure, table, Transact-SQL trigger, view	<p>Indicates that the original text of the module statement was converted to an obfuscated format. The output of the obfuscation is not directly visible in any of the catalog views in SQL Server 2005. Users without access to system tables or database files cannot retrieve the obfuscated text. However, the text is available to users that can either access system tables over the <a href="#">DAC port</a> or directly access database files. Also, users that can attach a debugger to the server process can retrieve the original procedure from memory at run time.</p> <p>1 = Encrypted 0 = Not encrypted Base data type: <b>int</b></p>
IsExecuted	Any schema-scoped object	<p>Specifies the object can be executed (view, procedure, function, or trigger).</p> <p>1 = True 0 = False Base data type: <b>int</b></p>
IsExtendedProc	Any schema-scoped object	<p>Extended procedure.</p> <p>1 = True 0 = False Base data type: <b>int</b></p>
IsForeignKey	Any schema-scoped object	<p>FOREIGN KEY constraint.</p> <p>1 = True 0 = False Base data type: <b>int</b></p>
IsIndexed	Table, view	<p>A table or view with an index.</p> <p>1 = True 0 = False Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
IsIndexable	Table, view	<p>A table or view on which an index may be created.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsInlineFunction	Function	<p>Inline function.</p> <p>1 = Inline function</p> <p>0 = Not inline function</p> <p>Base data type: <b>int</b></p>
IsMSShipped	Any schema-scoped object	<p>An object created during installation of SQL Server.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsPrecise	Computed column, function, user-defined type, view	<p>Indicates whether the object contains an imprecise computation, such as floating point operations.</p> <p>1 = Precise</p> <p>0 = Imprecise</p> <p>Base data type: <b>int</b></p>
IsPrimaryKey	Any schema-scoped object	<p>PRIMARY KEY constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsProcedure	Any schema-scoped object	<p>Procedure.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
IsQuotedIdentOn	CHECK constraint, DEFAULT definition, Transact-SQL function, Transact-SQL procedure, table, Transact-SQL trigger, view	<p>Specifies that the quoted identifier setting for the object is ON, meaning double quotation marks delimit identifiers in all expressions involved in the object definition.</p> <p>1 = True 0 = False Base data type: <b>int</b></p>
IsQueue	Any schema-scoped object	<p>Service Broker Queue</p> <p>1 = True 0 = False Base data type: <b>int</b></p>
IsReplProc	Any schema-scoped object	<p>Replication procedure.</p> <p>1 = True 0 = False Base data type: <b>int</b></p>
IsRule	Any schema-scoped object	<p>Bound rule.</p> <p>1 = True 0 = False Base data type: <b>int</b></p>
IsScalarFunction	Function	<p>Scalar-valued function.</p> <p>1 = Scalar-valued function 0 = Not scalar-valued function Base data type: <b>int</b></p>
IsSchemaBound	Function, Procedure, view	<p>A schema bound function or view created by using SCHEMABINDING.</p> <p>1 = Schema-bound 0 = Not schema-bound Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
IsSystemTable	Table	<p>System table.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsSystemVerified	Computed column, function, user-defined type, view	<p>The precision and determinism properties of the object can be verified by SQL Server.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsTable	Table	<p>Table.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsTableFunction	Function	<p>Table-valued function.</p> <p>1 = Table-valued function</p> <p>0 = Not table-valued function</p> <p>Base data type: <b>int</b></p>
IsTrigger	Any schema-scoped object	<p>Trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsUniqueCnst	Any schema-scoped object	<p>UNIQUE constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
IsUserTable	Table	<p>User-defined table.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
IsView	View	<p>View.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
OwnerId	Any schema-scoped object	<p>Owner of the object.</p> <p><b>Note:</b> The schema owner is not necessarily the object owner. For example, child objects (those where <i>parent_object_id</i> is nonnull) will always return the same owner ID as the parent.</p> <p>Nonnull = Database user ID of the object owner.</p> <p>NULL = Unsupported object type, or object ID is not valid.</p> <p>Base data type: <b>int</b></p>
Schemaid	Any schema-scoped object	<p>The ID of the schema associated with the object.</p> <p>Nonnull = Schema ID of the object.</p> <p>Base data type: <b>int</b></p>
SystemDataAccess	Function, view	<p>Object accesses system data, system catalogs or virtual system tables, in the local instance of SQL Server.</p> <p>0 = None</p> <p>1 = Read</p> <p>Base data type: <b>int</b></p>
TableDeleteTrigger	Table	<p>Table has a DELETE trigger.</p> <p>&gt;1 = ID of first trigger with the specified type.</p> <p>Base data type: <b>int</b></p>
TableDeleteTriggerCount	Table	<p>The table has the specified number of DELETE triggers.</p> <p>Nonnull = Number of DELETE triggers</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableFullTextMergeStatus	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Whether a table that has a full-text index that is currently in merging.</p> <p>0 = Table does not have a full-text index, or the full-text index is not in merging.</p> <p>1 = The full-text index is in merging.</p>
TableFullTextBackgroundUpdateIndexOn	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>The table has full-text background update index (autochange tracking) enabled.</p> <p>1 = TRUE</p> <p>0 = FALSE</p> <p>Base data type: <b>int</b></p>
TableFulltextCatalogId	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>ID of the full-text catalog in which the full-text index data for the table resides.</p> <p>Nonzero = Full-text catalog ID, associated with the unique index that identifies the rows in a full-text indexed table.</p> <p>0 = Table does not have a full-text index.</p> <p>Base data type: <b>int</b></p>
TableFullTextChangeTrackingOn	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Table has full-text change-tracking enabled.</p> <p>1 = TRUE</p> <p>0 = FALSE</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableFulltextDocsProcessed	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Number of rows processed since the start of full-text indexing. In a table that is being indexed for full-text search, all the columns of one row are considered as part of one document to be indexed.</p> <p>0 = No active crawl or full-text indexing is completed.</p> <p>&gt; 0 = One of the following (A or B): A) The number of documents processed by insert or update operations since the start of full, incremental, or manual change tracking population; B) The number of rows processed by insert or update operations since change tracking with background update index population was enabled, the full-text index schema changed, the full-text catalog rebuilt, or the instance of SQL Server restarted, and so on.</p> <p>NULL = Table does not have a full-text index.</p> <p>Base data type: <b>int</b></p> <p><b>Note</b> This property does not monitor or count deleted rows.</p>
TableFulltextFailCount	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>The number of rows that full-text search did not index.</p> <p>0 = The population has completed.</p> <p>&gt; 0 = One of the following (A or B): A) The number of documents that were not indexed since the start of Full, Incremental, and Manual Update change tracking population; B) For change tracking with background update index, the number of rows that were not indexed since the start of the population, or the restart of the population. This could be caused by a schema change, rebuild of the catalog, server restart, and so on</p> <p>NULL = Table does not have a Full-Text index.</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableFulltextItemCount	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Nonnull = Number of rows that were full-text indexed successfully.</p> <p>NULL = Table does not have a full-text index.</p> <p>Base data type: <b>int</b></p>
TableFulltextKeyColumn	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>ID of the column associated with the single-column unique index that is part of the definition of a full-text index and semantic index.</p> <p>0 = Table does not have a full-text index.</p> <p>Base data type: <b>int</b></p>
TableFulltextPendingChanges	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Number of pending change tracking entries to process.</p> <p>0 = change tracking is not enabled.</p> <p>NULL = Table does not have a full-text index.</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableFulltextPopulateStatus	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>0 = Idle.</p> <p>1 = Full population is in progress.</p> <p>2 = Incremental population is in progress.</p> <p>3 = Propagation of tracked changes is in progress.</p> <p>4 = Background update index is in progress, such as autochange tracking.</p> <p>5 = Full-text indexing is throttled or paused.</p> <p>6 = An error has occurred. Examine the crawl log for details. For more information, see the <a href="#">Troubleshooting Errors in a Full-Text Population (Crawl)</a> section of <a href="#">Populate Full-Text Indexes</a>.</p> <p>Base data type: <b>int</b></p>
TableFullTextSemanticExtraction	Table	<p><b>Applies to:</b> SQL Server 2012 through SQL Server 2017.</p> <p>Table is enabled for semantic indexing.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasActiveFulltextIndex	Table	<p><b>Applies to:</b> SQL Server 2008 through SQL Server 2017.</p> <p>Table has an active full-text index.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasCheckCnst	Table	<p>Table has a CHECK constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableHasClustIndex	Table	<p>Table has a clustered index.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasDefaultCnst	Table	<p>Table has a DEFAULT constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasDeleteTrigger	Table	<p>Table has a DELETE trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasForeignKey	Table	<p>Table has a FOREIGN KEY constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasForeignRef	Table	<p>Table is referenced by a FOREIGN KEY constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasIdentity	Table	<p>Table has an identity column.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasIndex	Table	<p>Table has an index of any type.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableHasInsertTrigger	Table	<p>Object has an INSERT trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasNonclustIndex	Table	<p>The table has a nonclustered index.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasPrimaryKey	Table	<p>Table has a primary key.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasRowGuidCol	Table	<p>Table has a ROWGUIDCOL for a <b>uniqueidentifier</b> column.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasTextImage	Table	<p>Table has a <b>text</b>, <b>ntext</b>, or <b>image</b> column.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasTimestamp	Table	<p>Table has a <b>timestamp</b> column.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasUniqueCnst	Table	<p>Table has a UNIQUE constraint.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableHasUpdateTrigger	Table	<p>The object has an UPDATE trigger.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableHasVarDecimalStorageFormat	Table	<p>Table is enabled for <b>vardecimal</b> storage format.</p> <p>1 = True</p> <p>0 = False</p>
TableInsertTrigger	Table	<p>Table has an INSERT trigger.</p> <p>&gt;1 = ID of first trigger with the specified type.</p> <p>Base data type: <b>int</b></p>
TableInsertTriggerCount	Table	<p>The table has the specified number of INSERT triggers.</p> <p>&gt;0 = The number of INSERT triggers.</p> <p>Base data type: <b>int</b></p>
TableIsFake	Table	<p>Table is not real. It is materialized internally on demand by the Database Engine.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>
TableIsLockedOnBulkLoad	Table	<p>Table is locked because a <b>bcp</b> or BULK INSERT job.</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableIsMemoryOptimized	Table	<p><b>Applies to:</b> SQL Server 2014 through SQL Server 2017.</p> <p>Table is memory optimized</p> <p>1 = True</p> <p>0 = False</p> <p>Base data type: <b>int</b></p> <p>For more information, see <a href="#">In-Memory OLTP (In-Memory Optimization)</a>.</p>
TableIsPinned	Table	<p>Table is pinned to be held in the data cache.</p> <p>0 = False</p> <p>This feature is not supported in SQL Server 2005 and later versions.</p>
TableTextInRowLimit	Table	<p>Table has text in row option set.</p> <p>&gt; 0 = Maximum bytes allowed for text in row.</p> <p>0 = text in row option is not set.</p> <p>Base data type: <b>int</b></p>
TableUpdateTrigger	Table	<p>Table has an UPDATE trigger.</p> <p>&gt; 1 = ID of first trigger with the specified type.</p> <p>Base data type: <b>int</b></p>
TableUpdateTriggerCount	Table	<p>Table has the specified number of UPDATE triggers.</p> <p>&gt; 0 = The number of UPDATE triggers.</p> <p>Base data type: <b>int</b></p>
User DataAccess	Function, View	<p>Indicates the object accesses user data, user tables, in the local instance of SQL Server.</p> <p>1 = Read</p> <p>0 = None</p> <p>Base data type: <b>int</b></p>

PROPERTY NAME	OBJECT TYPE	DESCRIPTION AND VALUES RETURNED
TableHasColumnSet	Table	<p>Table has a column set.</p> <p>0 = False</p> <p>1 = True</p> <p>For more information, see <a href="#">Use Column Sets</a>.</p>
Cardinality	Table (system or user-defined), view, or index	<p><b>Applies to:</b> SQL Server 2012 through SQL Server 2017.</p> <p>The number of rows in the specified object.</p>
TableTemporalType	Table	<p><b>Applies to:</b> SQL Server 2016 through SQL Server 2017.</p> <p>Specifies the type of table.</p> <p>0 = non-temporal table</p> <p>1 = history table for system-versioned table</p> <p>2 = system-versioned temporal table</p>

## Return Types

**sql\_variant**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

A user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as OBJECTPROPERTYEX may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

The Database Engine assumes that *object\_id* is in the current database context. A query that references an *object\_id* in another database will return NULL or incorrect results. For example, in the following query the current database context is the master database. The Database Engine will try to return the property value for the specified *object\_id* in that database instead of the database that is specified in the query. The query returns incorrect results because the view `vEmployee` is not in the master database.

```
USE master;
GO
SELECT OBJECTPROPERTYEX(OBJECT_ID(N'AdventureWorks2012.HumanResources.vEmployee'), 'IsView');
GO
```

OBJECTPROPERTYEX(*view\_id*, 'IsIndexable') may consume significant computer resources because evaluation of IsIndexable property requires the parsing of view definition, normalization, and partial optimization. Although the

`IsIndexable` property identifies tables or views that can be indexed, the actual creation of the index still might fail if certain index key requirements are not met. For more information, see [CREATE INDEX \(Transact-SQL\)](#).

`OBJECTPROPERTYEX (table_id, 'TableHasActiveFulltextIndex')` will return a value of 1 (true) when at least one column of a table is added for indexing. Full-text indexing becomes active for population as soon as the first column is added for indexing.

Restrictions on metadata visibility are applied to the result set. For more information, see [Metadata Visibility Configuration](#).

## Examples

### A. Finding the base type of an object

The following example creates a SYNONYM `MyEmployeeTable` for the `Employee` table in the **AdventureWorks2012** database and then returns the base type of the SYNONYM.

```
USE AdventureWorks2012;
GO
CREATE SYNONYM MyEmployeeTable FOR HumanResources.Employee;
GO
SELECT OBJECTPROPERTYEX ( object_id(N'MyEmployeeTable'), N'BaseType')AS [Base Type];
GO
```

The result set shows that the base type of the underlying object, the `Employee` table, is a user table.

Base Type

-----

U

### B. Returning a property value

The following example returns the number of UPDATE triggers on the specified table.

```
USE AdventureWorks2012;
GO
SELECT OBJECTPROPERTYEX(OBJECT_ID(N'HumanResources.Employee'), N'TABLEUPDATETRIGGERCOUNT');
GO
```

### C. Finding tables that have a FOREIGN KEY constraint

The following example uses the `TableHasForeignKey` property to return all the tables that have a FOREIGN KEY constraint.

```
USE AdventureWorks2012;
GO
SELECT name, object_id, schema_id, type_desc
FROM sys.objects
WHERE OBJECTPROPERTYEX(object_id, N'TableHasForeignKey') = 1
ORDER BY name;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D: Finding the base type of an object

The following example returns the base type of `dbo.DimReseller` object.

```
-- Uses AdventureWorks  
  
SELECT OBJECTPROPERTYEX ( object_id(N'dbo.DimReseller'), N'BaseType')AS BaseType;
```

The result set shows that the base type of the underlying object, the `dbo.DimReseller` table, is a user table.

```
BaseType  
-----  
U
```

## See Also

[CREATE SYNONYM \(Transact-SQL\)](#)  
[Metadata Functions \(Transact-SQL\)](#)  
[OBJECT\\_DEFINITION \(Transact-SQL\)](#)  
[OBJECT\\_ID \(Transact-SQL\)](#)  
[OBJECT\\_NAME \(Transact-SQL\)](#)  
[sys.objects \(Transact-SQL\)](#)  
[ALTER AUTHORIZATION \(Transact-SQL\)](#)  
[TYPEPROPERTY \(Transact-SQL\)](#)

# ORIGINAL\_DB\_NAME (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the database name that is specified by the user in the database connection string. This is the database that is specified by using the **sqlcmd-d** option (USE *database*) or the ODBC data source expression (initial catalog =*databasename*).

This database is not the same as the default user database.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
ORIGINAL_DB_NAME ()
```

## Remarks

If the initial database is not specified, the function returns an empty string.

## See Also

[sqlcmd Utility](#)  
[osql Utility](#)  
[SQL Server Native Client \(ODBC\)](#)

# PARSENAME (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the specified part of an object name. The parts of an object that can be retrieved are the object name, owner name, database name, and server name.

## NOTE

The PARSENAME function does not indicate whether an object by the specified name exists. PARSENAME just returns the specified part of the specified object name.



## Syntax

```
PARSENAME ( 'object_name' , object_piece )
```

## Arguments

### 'object\_name'

Is the name of the object for which to retrieve the specified object part. *object\_name* is **sysname**. This parameter is an optionally-qualified object name. If all parts of the object name are qualified, this name can have four parts: the server name, the database name, the owner name, and the object name.

### object\_piece

Is the object part to return. *object\_piece* is of type **int**, and can have these values:

1 = Object name

2 = Schema name

3 = Database name

4 = Server name

## Return Types

### nchar

## Remarks

PARSENAME returns NULL if one of the following conditions is true:

- Either *object\_name* or *object\_piece* is NULL.
- A syntax error occurs.

The requested object part has a length of 0 and is not a valid Microsoft SQL Server identifier. A zero-length object name renders the complete qualified name as not valid.

## Examples

The following example uses `PARSENAME` to return information about the `Person` table in the `AdventureWorks2012` database.

```
USE AdventureWorks2012;
SELECT PARSENAME('AdventureWorks2012..Person', 1) AS 'Object Name';
SELECT PARSENAME('AdventureWorks2012..Person', 2) AS 'Schema Name';
SELECT PARSENAME('AdventureWorks2012..Person', 3) AS 'Database Name';
SELECT PARSENAME('AdventureWorks2012..Person', 4) AS 'Server Name';
GO
```

Here is the result set.

Object Name
-----

Person
(1 row(s) affected)

Schema Name
-----

(null)
(1 row(s) affected)

Database Name
-----

AdventureWorks2012
(1 row(s) affected)

Server Name
-----

(null)
(1 row(s) affected)

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example uses `PARSENAME` to return information about the `Person` table in the `AdventureWorks2012` database.

```
-- Uses AdventureWorks

SELECT PARSENAME('AdventureWorksPDW2012.dbo.DimCustomer', 1) AS 'Object Name';
SELECT PARSENAME('AdventureWorksPDW2012.dbo.DimCustomer', 2) AS 'Schema Name';
SELECT PARSENAME('AdventureWorksPDW2012.dbo.DimCustomer', 3) AS 'Database Name';
SELECT PARSENAME('AdventureWorksPDW2012.dbo.DimCustomer', 4) AS 'Server Name';
GO
```

Here is the result set.

Object Name

-----

DimCustomer

(1 row(s) affected)

Schema Name

-----

dbo

(1 row(s) affected)

Database Name

-----

AdventureWorksPDW2012

(1 row(s) affected)

Server Name

-----

(null)

(1 row(s) affected)

## See Also

[ALTER TABLE \(Transact-SQL\)](#)

[CREATE TABLE \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

# SCHEMA\_ID (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the schema ID associated with a schema name.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SCHEMA_ID ( [ schema_name ] )
```

## Arguments

TERM	DEFINITION
<i>schema_name</i>	Is the name of the schema. <i>schema_name</i> is a <b>sysname</b> . If <i>schema_name</i> is not specified, SCHEMA_ID will return the ID of the default schema of the caller.

## Return Types

**int**

NULL will be returned if *schema\_name* is not a valid schema.

## Remarks

SCHEMA\_ID will return IDs of system schemas and user-defined schemas. SCHEMA\_ID can be called in a select list, in a WHERE clause, and anywhere an expression is allowed.

## Examples

### A. Returning the default schema ID of a caller

```
SELECT SCHEMA_ID();
GO
```

### B. Returning the schema ID of a named schema

```
USE AdventureWorks2012;
GO
SELECT SCHEMA_ID('HumanResources');
GO
```

Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Returning the default schema ID of a caller

```
SELECT SCHEMA_ID();
```

### D. Returning the schema ID of a named schema

```
SELECT SCHEMA_ID('dbo');
```

## See Also

[Metadata Functions \(Transact-SQL\)](#)

[SCHEMA\\_NAME \(Transact-SQL\)](#)

[sys.schemas \(Transact-SQL\)](#)

# SCHEMA\_NAME (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the schema name associated with a schema ID.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SCHEMA_NAME ( [ schema_id ] )
```

## Arguments

TERM	DEFINITION
<i>schema_id</i>	The ID of the schema. <i>schema_id</i> is an <b>int</b> . If <i>schema_id</i> is not defined, SCHEMA_NAME will return the name of the default schema of the caller.

## Return Types

**sysname**

Returns NULL when *schema\_id* is not a valid ID.

## Remarks

SCHEMA\_NAME returns names of system schemas and user-defined schemas. SCHEMA\_NAME can be called in a select list, in a WHERE clause, and anywhere an expression is allowed.

## Examples

### A. Returning the name of the default schema of the caller

```
SELECT SCHEMA_NAME();  
GO
```

### B. Returning the name of a schema by using an ID

```
USE AdventureWorks2012;  
GO  
SELECT SCHEMA_NAME(5);  
GO
```

Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Returning the name of the default schema of the caller

```
SELECT SCHEMA_NAME();
```

### D. Returning the name of a schema by using an ID

```
SELECT SCHEMA_NAME(1);
```

## See Also

[Expressions \(Transact-SQL\)](#)

[SCHEMA\\_ID \(Transact-SQL\)](#)

[sys.schemas \(Transact-SQL\)](#)

[sys.database\\_principals \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[WHERE \(Transact-SQL\)](#)

# SCOPE\_IDENTITY (Transact-SQL)

7/6/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the last identity value inserted into an identity column in the same scope. A scope is a module: a stored procedure, trigger, function, or batch. Therefore, if two statements are in the same stored procedure, function, or batch, they are in the same scope.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
SCOPE_IDENTITY()
```

## Return Types

**numeric(38,0)**

## Remarks

SCOPE\_IDENTITY, IDENT\_CURRENT, and @@IDENTITY are similar functions because they return values that are inserted into identity columns.

IDENT\_CURRENT is not limited by scope and session; it is limited to a specified table. IDENT\_CURRENT returns the value generated for a specific table in any session and any scope. For more information, see [IDENT\\_CURRENT \(Transact-SQL\)](#).

SCOPE\_IDENTITY and @@IDENTITY return the last identity values that are generated in any table in the current session. However, SCOPE\_IDENTITY returns values inserted only within the current scope; @@IDENTITY is not limited to a specific scope.

For example, there are two tables, T1 and T2, and an INSERT trigger is defined on T1. When a row is inserted to T1, the trigger fires and inserts a row in T2. This scenario illustrates two scopes: the insert on T1, and the insert on T2 by the trigger.

Assuming that both T1 and T2 have identity columns, @@IDENTITY and SCOPE\_IDENTITY return different values at the end of an INSERT statement on T1. @@IDENTITY returns the last identity column value inserted across any scope in the current session. This is the value inserted in T2. SCOPE\_IDENTITY() returns the IDENTITY value inserted in T1. This was the last insert that occurred in the same scope. The SCOPE\_IDENTITY() function returns the null value if the function is invoked before any INSERT statements into an identity column occur in the scope.

Failed statements and transactions can change the current identity for a table and create gaps in the identity column values. The identity value is never rolled back even though the transaction that tried to insert the value into the table is not committed. For example, if an INSERT statement fails because of an IGNORE\_DUP\_KEY violation, the current identity value for the table is still incremented.

## Examples

### A. Using @@IDENTITY and SCOPE\_IDENTITY with triggers

The following example creates two tables, `TZ` and `TY`, and an INSERT trigger on `TZ`. When a row is inserted to table `TZ`, the trigger (`Ztrig`) fires and inserts a row in `TY`.

```
USE tempdb;
GO
CREATE TABLE TZ (
    Z_id int IDENTITY(1,1)PRIMARY KEY,
    Z_name varchar(20) NOT NULL;

INSERT TZ
VALUES ('Lisa'),('Mike'),('Carla');

SELECT * FROM TZ;
```

Result set: This is how table TZ looks.

Z_id	Z_name
1	Lisa
2	Mike
3	Carla

```
CREATE TABLE TY (
    Y_id int IDENTITY(100,5)PRIMARY KEY,
    Y_name varchar(20) NULL;

INSERT TY (Y_name)
VALUES ('boathouse'), ('rocks'), ('elevator');

SELECT * FROM TY;
```

Result set: This is how TY looks:

Y_id	Y_name
100	boathouse
105	rocks
110	elevator

Create the trigger that inserts a row in table TY when a row is inserted in table TZ.

```
CREATE TRIGGER Ztrig
ON TZ
FOR INSERT AS
BEGIN
    INSERT TY VALUES ('')
END;
```

FIRE the trigger and determine what identity values you obtain with the `@@IDENTITY` and `SCOPE_IDENTITY` functions.

```
INSERT TZ VALUES ('Rosalie');

SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY];
GO
SELECT @@IDENTITY AS [@@IDENTITY];
GO
```

Here is the result set.

```
/*SCOPE_IDENTITY returns the last identity value in the same scope. This was the insert on table TZ.*/
SCOPE_IDENTITY
4

/*@@IDENTITY returns the last identity value inserted to TY by the trigger.
This fired because of an earlier insert on TZ.*/
@@IDENTITY
115
```

## B. Using @@IDENTITY and SCOPE\_IDENTITY() with replication

The following examples show how to use `@@IDENTITY` and `SCOPE_IDENTITY()` for inserts in a database that is published for merge replication. Both tables in the examples are in the **AdventureWorks2012** sample database: `Person.ContactType` is not published, and `Sales.Customer` is published. Merge replication adds triggers to tables that are published. Therefore, `@@IDENTITY` can return the value from the insert into a replication system table instead of the insert into a user table.

The `Person.ContactType` table has a maximum identity value of 20. If you insert a row into the table, `@@IDENTITY` and `SCOPE_IDENTITY()` return the same value.

```
USE AdventureWorks2012;
GO
INSERT INTO Person.ContactType ([Name]) VALUES ('Assistant to the Manager');
GO
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY];
GO
SELECT @@IDENTITY AS [@@IDENTITY];
GO
```

Here is the result set.

```
SCOPE_IDENTITY
21
@@IDENTITY
21
```

The `Sales.Customer` table has a maximum identity value of 29483. If you insert a row into the table, `@@IDENTITY` and `SCOPE_IDENTITY()` return different values. `SCOPE_IDENTITY()` returns the value from the insert into the user table, whereas `@@IDENTITY` returns the value from the insert into the replication system table. Use `SCOPE_IDENTITY()` for applications that require access to the inserted identity value.

```
INSERT INTO Sales.Customer ([TerritoryID],[PersonID]) VALUES (8,NULL);
GO
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY];
GO
SELECT @@IDENTITY AS [@@IDENTITY];
GO
```

Here is the result set.

```
SCOPE_IDENTITY
29484
@@IDENTITY
89
```

## See Also

[@@IDENTITY \(Transact-SQL\)](#)

# SERVERPROPERTY (Transact-SQL)

3/24/2017 • 8 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns property information about the server instance.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SERVERPROPERTY ( 'propertyname' )
```

## Arguments

### *propertyname*

Is an expression that contains the property information to be returned for the server. *propertyname* can be one of the following values.

PROPERTY	VALUES RETURNED
BuildClrVersion	Version of the Microsoft .NET Framework common language runtime (CLR) that was used while building the instance of SQL Server.  NULL = Input is not valid, an error, or not applicable.  Base data type: <b>nvarchar(128)</b>
Collation	Name of the default collation for the server.  NULL = Input is not valid, or an error.  Base data type: <b>nvarchar(128)</b>
CollationID	ID of the SQL Server collation.  Base data type: <b>int</b>
ComparisonStyle	Windows comparison style of the collation.  Base data type: <b>int</b>

PROPERTY	VALUES RETURNED
ComputerNamePhysicalNetBIOS	<p>NetBIOS name of the local computer on which the instance of SQL Server is currently running.</p> <p>For a clustered instance of SQL Server on a failover cluster, this value changes as the instance of SQL Server fails over to other nodes in the failover cluster.</p> <p>On a stand-alone instance of SQL Server, this value remains constant and returns the same value as the MachineName property.</p> <p><b>Note:</b> If the instance of SQL Server is in a failover cluster and you want to obtain the name of the failover clustered instance, use the MachineName property.</p> <p>NULL = Input is not valid, an error, or not applicable.</p> <p>Base data type: <b>nvarchar(128)</b></p>
Edition	<p>Installed product edition of the instance of SQL Server. Use the value of this property to determine the features and the limits, such as <a href="#">Compute Capacity Limits by Edition of SQL Server</a>. 64-bit versions of the Database Engine append (64-bit) to the version.</p> <p>Returns:</p> <p>'Enterprise Edition'</p> <p>'Enterprise Edition: Core-based Licensing'</p> <p>'Enterprise Evaluation Edition'</p> <p>'Business Intelligence Edition'</p> <p>'Developer Edition'</p> <p>'Express Edition'</p> <p>'Express Edition with Advanced Services'</p> <p>'Standard Edition'</p> <p>'Web Edition'</p> <p>'SQL Azure' indicates SQL Database or SQL Data Warehouse</p> <p>Base data type: <b>nvarchar(128)</b></p>

PROPERTY	VALUES RETURNED
EditionID	<p>EditionID represents the installed product edition of the instance of SQL Server. Use the value of this property to determine features and limits, such as <a href="#">Compute Capacity Limits by Edition of SQL Server</a>.</p> <p>1804890536 = Enterprise  1872460670 = Enterprise Edition: Core-based Licensing  610778273 = Enterprise Evaluation  284895786 = Business Intelligence  -2117995310 = Developer  -1592396055 = Express  -133711905 = Express with Advanced Services  -1534726760 = Standard  1293598313 = Web  1674378470 = SQL Database or SQL Data Warehouse  Base data type: <b>bigint</b></p>
EngineEdition	<p>Database Engine edition of the instance of SQL Server installed on the server.</p> <p>1 = Personal or Desktop Engine (Not available in SQL Server 2005 and later versions.)  2 = Standard (This is returned for Standard, Web, and Business Intelligence.)  3 = Enterprise (This is returned for Evaluation, Developer, and both Enterprise editions.)  4 = Express (This is returned for Express, Express with Tools and Express with Advanced Services)  5 = SQL Database  6 = SQL Data Warehouse  Base data type: <b>int</b></p>
HadrManagerStatus	<p><b>Applies to:</b> SQL Server 2012 through SQL Server 2017.</p> <p>Indicates whether the Always On availability groups manager has started.</p> <p>0 = Not started, pending communication.  1 = Started and running.  2 = Not started and failed.  NULL = Input is not valid, an error, or not applicable.</p>

PROPERTY	VALUES RETURNED
InstanceDefaultDataPath	<p><b>Applies to:</b> SQL Server 2012 through current version in updates beginning in late 2015.</p> <p>Name of the default path to the instance data files.</p>
InstanceDefaultLogPath	<p><b>Applies to:</b> SQL Server 2012 through current version in updates beginning in late 2015.</p> <p>Name of the default path to the instance log files.</p>
InstanceName	<p>Name of the instance to which the user is connected.</p> <p>Returns NULL if the instance name is the default instance, if the input is not valid, or error.</p> <p>NULL = Input is not valid, an error, or not applicable.</p> <p>Base data type: <b>nvarchar(128)</b></p>
IsAdvancedAnalyticsInstalled	<p>Returns 1 if the Advanced Analytics feature was installed during setup; 0 if Advanced Analytics was not installed.</p>
IsClustered	<p>Server instance is configured in a failover cluster.</p> <p>1 = Clustered.</p> <p>0 = Not Clustered.</p> <p>NULL = Input is not valid, an error, or not applicable.</p> <p>Base data type: <b>int</b></p>
IsFullTextInstalled	<p>The full-text and semantic indexing components are installed on the current instance of SQL Server.</p> <p>1 = Full-text and semantic indexing components are installed.</p> <p>0 = Full-text and semantic indexing components are not installed.</p> <p>NULL = Input is not valid, an error, or not applicable.</p> <p>Base data type: <b>int</b></p>

PROPERTY	VALUES RETURNED
IsHadrEnabled	<p><b>Applies to:</b> SQL Server 2012 through SQL Server 2017.</p> <p>Always On availability groups is enabled on this server instance.</p> <p>0 = The Always On availability groups feature is disabled.</p> <p>1 = The Always On availability groups feature is enabled.</p> <p>NULL = Input is not valid, an error, or not applicable.</p> <p>Base data type: <b>int</b></p> <p>For availability replicas to be created and run on an instance of SQL Server, Always On availability groups must be enabled on the server instance. For more information, see <a href="#">Enable and Disable AlwaysOn Availability Groups (SQL Server)</a>.</p> <p><b>Note:</b> The IsHadrEnabled property pertains only to Always On availability groups. Other high availability or disaster recovery features, such as database mirroring or log shipping, are unaffected by this server property.</p>
IsIntegratedSecurityOnly	<p>Server is in integrated security mode.</p> <p>1 = Integrated security (Windows Authentication)</p> <p>0 = Not integrated security. (Both Windows Authentication and SQL Server Authentication.)</p> <p>NULL = Input is not valid, an error, or not applicable.</p> <p>Base data type: <b>int</b></p>
IsLocalDB	<p><b>Applies to:</b> SQL Server 2012 through SQL Server 2017.</p> <p>Server is an instance of SQL Server Express LocalDB.</p> <p>NULL = Input is not valid, an error, or not applicable.</p>
IsPolybaseInstalled	<p><b>Applies to:</b> SQL Server 2017.</p> <p>Returns whether the server instance has the PolyBase feature installed.</p> <p>0 = PolyBase is not installed.</p> <p>1 = PolyBase is installed.</p> <p>Base data type: <b>int</b></p>

PROPERTY	VALUES RETURNED
IsSingleUser	<p>Server is in single-user mode.</p> <p>1 = Single user.</p> <p>0 = Not single user</p> <p>NULL = Input is not valid, an error, or not applicable.</p> <p>Base data type: <b>int</b></p>
IsXTPSupported	<p><b>Applies to:</b> SQL Server ( SQL Server 2014 through SQL Server 2017), SQL Database.</p> <p>Server supports In-Memory OLTP.</p> <p>1= Server supports In-Memory OLTP.</p> <p>0= Server does not supports In-Memory OLTP.</p> <p>NULL = Input is not valid, an error, or not applicable.</p> <p>Base data type: <b>int</b></p>
LCID	<p>Windows locale identifier (LCID) of the collation.</p> <p>Base data type: <b>int</b></p>
LicenseType	<p>Unused. License information is not preserved or maintained by the SQL Server product. Always returns DISABLED.</p> <p>Base data type: <b>nvarchar(128)</b></p>
MachineName	<p>Windows computer name on which the server instance is running.</p> <p>For a clustered instance, an instance of SQL Server running on a virtual server on Microsoft Cluster Service, it returns the name of the virtual server.</p> <p>NULL = Input is not valid, an error, or not applicable.</p> <p>Base data type: <b>nvarchar(128)</b></p>
NumLicenses	<p>Unused. License information is not preserved or maintained by the SQL Server product. Always returns NULL.</p> <p>Base data type: <b>int</b></p>
ProcessID	<p>Process ID of the SQL Server service. ProcessID is useful in identifying which Sqlservr.exe belongs to this instance.</p> <p>NULL = Input is not valid, an error, or not applicable.</p> <p>Base data type: <b>int</b></p>
ProductBuild	<p><b>Applies to:</b> SQL Server 2014 beginning October, 2015.</p> <p>The build number.</p>

PROPERTY	VALUES RETURNED
ProductBuildType	<p><b>Applies to:</b> SQL Server 2012 through current version in updates beginning in late 2015.</p> <p>Type of build of the current build.</p> <p>Returns one of the following:</p> <p>OD = On Demand release a specific customer.</p> <p>GDR = General Distribution Release released through windows update.</p> <p>NULL = Not applicable.</p>
ProductLevel	<p>Level of the version of the instance of SQL Server.</p> <p>Returns one of the following:</p> <p>'RTM' = Original release version</p> <p>'SPn' = Service pack version</p> <p>'CTPn', = Community Technology Preview version</p> <p>Base data type: <b>nvarchar(128)</b></p>
ProductMajorVersion	<p><b>Applies to:</b> SQL Server 2012 through current version in updates beginning in late 2015.</p> <p>The major version.</p>
ProductMinorVersion	<p><b>Applies to:</b> SQL Server 2012 through current version in updates beginning in late 2015.</p> <p>The minor version.</p>
ProductUpdateLevel	<p><b>Applies to:</b> SQL Server 2012 through current version in updates beginning in late 2015.</p> <p>Update level of the current build. CU indicates a cumulative update.</p> <p>Returns one of the following:</p> <p>CUn = Cumulative Update</p> <p>NULL = Not applicable.</p>
ProductUpdateReference	<p><b>Applies to:</b> SQL Server 2012 through current version in updates beginning in late 2015.</p> <p>KB article for that release.</p>
ProductVersion	<p>Version of the instance of SQL Server, in the form of '<i>major.minor.build.revision</i>'.</p> <p>Base data type: <b>nvarchar(128)</b></p>

PROPERTY	VALUES RETURNED
ResourceLastUpdateDateTime	Returns the date and time that the Resource database was last updated.  Base data type: <b>datetime</b>
ResourceVersion	Returns the version Resource database.  Base data type: <b>nvarchar(128)</b>
ServerName	Both the Windows server and instance information associated with a specified instance of SQL Server.  NULL = Input is not valid, or an error.  Base data type: <b>nvarchar(128)</b>
SqlCharSet	The SQL character set ID from the collation ID.  Base data type: <b>tinyint</b>
SqlCharSetName	The SQL character set name from the collation.  Base data type: <b>nvarchar(128)</b>
SqlSortOrder	The SQL sort order ID from the collation  Base data type: <b>tinyint</b>
SqlSortOrderName	The SQL sort order name from the collation.  Base data type: <b>nvarchar(128)</b>
FilestreamShareName	The name of the share used by FILESTREAM.  NULL = Input is not valid, an error, or not applicable.
FilestreamConfiguredLevel	The configured level of FILESTREAM access. For more information, see <a href="#">filestream access level</a> .
FilestreamEffectiveLevel	The effective level of FILESTREAM access. This value can be different than the FilestreamConfiguredLevel if the level has changed and either an instance restart or a computer restart is pending. For more information, see <a href="#">filestream access level</a> .

## Return Types

**sql\_variant**

## Remarks

### ServerName Property

The `ServerName` property of the `SERVERPROPERTY` function and `@@SERVERNAME` return similar information. The `ServerName` property provides the Windows server and instance name that together make up the unique server instance. `@@SERVERNAME` provides the currently configured local server name.

The `ServerName` property and `@@SERVERNAME` return the same information if the default server name at the time of installation has not been changed. The local server name can be configured by executing the following:

```
EXEC sp_dropserver 'current_server_name';
GO
EXEC sp_addserver 'new_server_name', 'local';
GO
```

If the local server name has been changed from the default server name at installation time, `@@SERVERNAME` returns the new name.

## Version Properties

The `SERVERPROPERTY` function returns individual properties that relate to the version information whereas the `@@VERSION` function combines the output into one string. If your application requires individual property strings, you can use the `SERVERPROPERTY` function to return them instead of parsing the `@@VERSION` results.

## Permissions

All users can query the server properties.

## Examples

The following example uses the `SERVERPROPERTY` function in a `SELECT` statement to return information about the current instance of SQL Server.

```
SELECT
    SERVERPROPERTY('MachineName') AS ComputerName,
    SERVERPROPERTY('ServerName') AS InstanceName,
    SERVERPROPERTY('Edition') AS Edition,
    SERVERPROPERTY('ProductVersion') AS ProductVersion,
    SERVERPROPERTY('ProductLevel') AS ProductLevel;
GO
```

## See Also

[Editions and Components of SQL Server 2016](#)

# STATS\_DATE (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the date of the most recent update for statistics on a table or indexed view.

For more information about updating statistics, see [Statistics](#).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
STATS_DATE ( object_id , stats_id )
```

## Arguments

*object\_id*

ID of the table or indexed view with the statistics.

*stats\_id*

ID of the statistics object.

## Return Types

Returns **datetime** on success. Returns **NULL** on error.

## Remarks

System functions can be used in the select list, in the WHERE clause, and anywhere an expression can be used.

## Permissions

Requires membership in the db\_owner fixed database role or permission to view the metadata for the table or indexed view.

## Examples

### A. Return the dates of the most recent statistics for a table

The following example returns the date of the most recent update for each statistics object on the `Person.Address` table.

```
USE AdventureWorks2012;
GO
SELECT name AS stats_name,
       STATS_DATE(object_id, stats_id) AS statistics_update_date
  FROM sys.stats
 WHERE object_id = OBJECT_ID('Person.Address');
GO
```

If statistics correspond to an index, the `stats_id` value in the `sys.stats` catalog view is the same as the `index_id` value in the `sys.indexes` catalog view, and the following query returns the same results as the preceding query. If statistics do not correspond to an index, they are in the `sys.stats` results but not in the `sys.indexes` results.

```
USE AdventureWorks2012;
GO
SELECT name AS index_name,
       STATS_DATE(object_id, index_id) AS statistics_update_date
FROM sys.indexes
WHERE object_id = OBJECT_ID('Person.Address');
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### B. Learn when a named statistics was last updated

The following example creates statistics on the `LastName` column of the `DimCustomer` table. It then runs a query to show the date of the statistics. Then it updates the statistics and runs the query again to show the updated date.

```
--First, create a statistics object
USE AdventureWorksPDW2012;
GO
CREATE STATISTICS Customer_LastName_Stats
ON AdventureWorksPDW2012.dbo.DimCustomer (LastName)
WITH SAMPLE 50 PERCENT;
GO

--Return the date when Customer_LastName_Stats was last updated
USE AdventureWorksPDW2012;
GO
SELECT stats_id, name AS stats_name,
       STATS_DATE(object_id, stats_id) AS statistics_date
FROM sys.stats s
WHERE s.object_id = OBJECT_ID('dbo.DimCustomer')
      AND s.name = 'Customer_LastName_Stats';
GO

--Update Customer_LastName_Stats so it will have a different timestamp in the next query
GO
UPDATE STATISTICS dbo.dimCustomer (Customer_LastName_Stats);

--Return the date when Customer_LastName_Stats was last updated.
SELECT stats_id, name AS stats_name,
       STATS_DATE(object_id, stats_id) AS statistics_date
FROM sys.stats s
WHERE s.object_id = OBJECT_ID('dbo.DimCustomer')
      AND s.name = 'Customer_LastName_Stats';
GO
```

### C. View the date of the last update for all statistics on a table

This example returns the date for when each statistics object on the `DimCustomer` table was last updated.

```
--Return the dates all statistics on the table were last updated.
SELECT stats_id, name AS stats_name,
       STATS_DATE(object_id, stats_id) AS statistics_date
FROM sys.stats s
WHERE s.object_id = OBJECT_ID('dbo.DimCustomer');
GO
```

If statistics correspond to an index, the *stats\_id* value in the [sys.stats](#) catalog view is the same as the *index\_id* value in the [sys.indexes](#) catalog view, and the following query returns the same results as the preceding query. If statistics do not correspond to an index, they are in the sys.stats results but not in the sys.indexes results.

```
USE AdventureWorksPDW2012;
GO
SELECT name AS index_name,
       STATS_DATE(object_id, index_id) AS statistics_update_date
FROM sys.indexes
WHERE object_id = OBJECT_ID('dbo.DimCustomer');
GO
```

## See Also

[System Functions \(Transact-SQL\)](#)

[UPDATE STATISTICS \(Transact-SQL\)](#)

[sp\\_autostats \(Transact-SQL\)](#)

[Statistics](#)

# TYPE\_ID (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the ID for a specified data type name.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
TYPE_ID ( [ schema_name ] type_name )
```

## Arguments

*type\_name*

Is the name of the data type. *type\_name* is of type **nvarchar**. *type\_name* can be a system or user-defined data type.

## Return Types

**int**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

In SQL Server, a user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as TYPE\_ID may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

TYPE\_ID returns NULL if the type name is not valid, or if the caller does not have sufficient permission to reference the type.

## Examples

### A. Looking up the TYPE\_ID values for single- and two-part type names

The following example returns type ID for single- and two-part type names.

```
USE tempdb;
GO
CREATE TYPE NewType FROM int;
GO
CREATE SCHEMA NewSchema;
GO
CREATE TYPE NewSchema.NewType FROM int;
GO
SELECT TYPE_ID('NewType') AS [1 Part Data Type ID],
      TYPE_ID('NewSchema.NewType') AS [2 Part Data Type ID];
GO
```

## B. Looking up the TYPE ID of a system data type

The following example returns the `TYPE_ID` for the `datetime` system data type.

```
SELECT TYPE_NAME(TYPE_ID('datetime')) AS [TYPE_NAME]
      ,TYPE_ID('datetime') AS [TYPE_ID];
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C: Looking up the TYPE ID of a system data type

The following example returns the `TYPE_ID` for the `datetime` system data type.

```
SELECT TYPE_NAME(TYPE_ID('datetime')) AS typeName,
      TYPE_ID('datetime') AS typeID FROM table1;
```

## See Also

[TYPE\\_NAME \(Transact-SQL\)](#)

[TYPEPROPERTY \(Transact-SQL\)](#)

[sys.types \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

# TYPE\_NAME (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the unqualified type name of a specified type ID.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
TYPE_NAME ( type_id )
```

## Arguments

*type\_id*

Is the ID of the type that will be used. *type\_id* is an **int**, and it can refer to a type in any schema that the caller has permission to access.

## Return Types

**sysname**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

In SQL Server, a user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as **TYPE\_NAME** may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Remarks

**TYPE\_NAME** will return NULL when *type\_id* is not valid or when the caller does not have sufficient permission to reference the type.

**TYPE\_NAME** works for system data types and also for user-defined data types. The type can be contained in any schema, but an unqualified type name is always returned. This means the name does not have the *schema*. prefix.

System functions can be used in the select list, in the WHERE clause, and anywhere an expression is allowed. For more information, see [Expressions \(Transact-SQL\)](#) and [WHERE \(Transact-SQL\)](#).

## Examples

The following example returns the object name, column name, and type name for each column in the **Vendor** table of the **AdventureWorks2012** database.

```
SELECT o.name AS obj_name, c.name AS col_name,
       TYPE_NAME(c.user_type_id) AS type_name
  FROM sys.objects AS o
 JOIN sys.columns AS c  ON o.object_id = c.object_id
 WHERE o.name = 'Vendor'
 ORDER BY col_name;
GO
```

Here is the result set.

obj_name	col_name	type_name
Vendor	AccountNumber	AccountNumber
Vendor	ActiveFlag	Flag
Vendor	BusinessEntityID	int
Vendor	CreditRating	tinyint
Vendor	ModifiedDate	datetime
Vendor	Name	Name
Vendor	PreferredVendorStatus	Flag
Vendor	PurchasingWebServiceURL	nvarchar

(8 row(s) affected)

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the `TYPE_ID` for the data type with id `1`.

```
SELECT TYPE_NAME(36) AS Type36, TYPE_NAME(239) AS Type239;
GO
```

For a list of types, query `sys.types`.

```
SELECT * FROM sys.types;
GO
```

## See Also

[TYPE\\_ID \(Transact-SQL\)](#)  
[TYPEPROPERTY \(Transact-SQL\)](#)  
[sys.types \(Transact-SQL\)](#)  
[Metadata Functions \(Transact-SQL\)](#)

# TYPEPROPERTY (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns information about a data type.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
TYPEPROPERTY (type , property)
```

## Arguments

*type*

Is the name of the data type.

*property*

Is the type of information to be returned for the data type. *property* can be one of the following values.

PROPERTY	DESCRIPTION	VALUE RETURNED
<b>AllowsNull</b>	Data type allows for null values.	1 = True 0 = False NULL = Data type not found.
<b>OwnerId</b>	Owner of the type.  Note: The schema owner is not necessarily the type owner.	Nonnull = The database user ID of the type owner.  NULL = Unsupported type, or type ID is not valid.
<b>Precision</b>	Precision for the data type.	The number of digits or characters.  -1 = <b>xml</b> or large value data type  NULL = Data type not found.
<b>Scale</b>	Scale for the data type.	The number of decimal places for the data type.  NULL = Data type is not <b>numeric</b> or not found.

PROPERTY	DESCRIPTION	VALUE RETURNED
<b>UsesAnsiTrim</b>	ANSI padding setting was ON when the data type was created.	1 = True 0 = False  NULL = Data type not found, or it is not a binary or string data type.

## Return Types

**int**

## Exceptions

Returns NULL on error or if a caller does not have permission to view the object.

In SQL Server, a user can only view the metadata of securables that the user owns or on which the user has been granted permission. This means that metadata-emitting, built-in functions such as `TYPEPROPERTY` may return NULL if the user does not have any permission on the object. For more information, see [Metadata Visibility Configuration](#).

## Examples

### A. Identifying the owner of a data type

The following example returns the owner of a data type.

```
SELECT TYPEPROPERTY(SCHEMA_NAME(schema_id) + '.' + name, 'OwnerId') AS owner_id, name, system_type_id,
user_type_id, schema_id
FROM sys.types;
```

### B. Returning the precision of the tinyint data type

The following example returns the precision or number of digits for the `tinyint` data type.

```
SELECT TYPEPROPERTY( 'tinyint', 'PRECISION');
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C: Returning the precision of the tinyint data type

The following example returns the precision or number of digits for the `tinyint` data type.

```
SELECT TYPEPROPERTY( 'tinyint', 'PRECISION');
```

## See Also

[TYPE\\_ID \(Transact-SQL\)](#)

[TYPE\\_NAME \(Transact-SQL\)](#)

[COLUMNPROPERTY \(Transact-SQL\)](#)

[Metadata Functions \(Transact-SQL\)](#)

[OBJECTPROPERTY \(Transact-SQL\)](#)

[ALTER AUTHORIZATION \(Transact-SQL\)](#)

[sys.types \(Transact-SQL\)](#)

# Version - Transact SQL Metadata functions

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:**  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

Returns the version of SQL Data Warehouse or Parallel Data Warehouse running on the appliance.

 [Transact-SQL Syntax Conventions \(Transact-SQL\)](#)

## Syntax

```
-- Azure SQL Data Warehouse and Parallel Data Warehouse  
VERSION()
```

## Arguments

## General Remarks

A table name must be specified in a [FROM](#) clause for this function to return results. A result row will be returned for each row in the result set for the query; use [TOP \(Transact-SQL\)](#) to limit the number of returned rows.

## Examples

The following example returns the version number.

```
SELECT VERSION();
```

## See Also

[SESSION\\_ID \(Transact-SQL\)](#)

[DB\\_NAME \(Transact-SQL\)](#)

# ODBC Scalar Functions (Transact-SQL)

3/24/2017 • 5 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

You can use [ODBC Scalar Functions](#) in Transact-SQL statements. These statements are interpreted by SQL Server. They can be used in stored procedures and user-defined functions. These include string, numeric, time, date, interval, and system functions.

## Usage

```
SELECT {fn <function_name> [ (<argument>,....n) ] }
```

## Functions

The following tables list ODBC scalar functions that are not duplicated in Transact-SQL.

### String Functions

FUNCTION	DESCRIPTION
BIT_LENGTH( string_exp ) (ODBC 3.0)	Returns the length in bits of the string expression.  Does not work only for string data types. Therefore, will not implicitly convert string_exp to string but instead will return the (internal) size of whatever data type it is given.
CONCAT( string_exp1,string_exp2 ) (ODBC 1.0)	Returns a character string that is the result of concatenating string_exp2 to string_exp1. The resulting string is DBMS-dependent. For example, if the column represented by string_exp1 contained a NULL value, DB2 would return NULL but SQL Server would return the non-NUL string.
OCTET_LENGTH( string_exp ) (ODBC 3.0)	Returns the length in bytes of the string expression. The result is the smallest integer not less than the number of bits divided by 8.  Does not work only for string data types. Therefore, will not implicitly convert string_exp to string but instead will return the (internal) size of whatever data type it is given.

### Numeric Function

FUNCTION	DESCRIPTION
TRUNCATE( numeric_exp, integer_exp ) (ODBC 2.0)	Returns numeric_exp truncated to integer_exp positions right of the decimal point. If integer_exp is negative, numeric_exp is truncated to $ integer\_exp $ positions to the left of the decimal point.

### Time, Date, and Interval Functions

FUNCTION	DESCRIPTION
CURRENT_DATE( ) (ODBC 3.0)	Returns the current date.
CURDATE( ) (ODBC 3.0)	Returns the current date.
CURRENT_TIME [ ( time-precision ) ] (ODBC 3.0)	Returns the current local time. The time-precision argument determines the seconds precision of the returned value
CURTIME() (ODBC 3.0)	Returns the current local time.
DAYNAME( date_exp ) (ODBC 2.0)	Returns a character string that contains the data source-specific name of the day (for example, Sunday through Saturday or Sun. through Sat. for a data source that uses English, or Sonntag through Samstag for a data source that uses German) for the day part of date_exp.
DAYOFMONTH( date_exp ) (ODBC 1.0)	Returns the day of the month based on the month field in date_exp as an integer value in the range of 1–31.
DAYOFWEEK( date_exp ) (ODBC 1.0)	Returns the day of the week based on the week field in date_exp as an integer value in the range of 1–7, where 1 represents Sunday.
HOUR( time_exp ) (ODBC 1.0)	Returns the hour based on the hour field in time_exp as an integer value in the range of 0–23.
MINUTE( time_exp ) (ODBC 1.0)	Returns the minute based on the minute field in time_exp as an integer value in the range of 0–59.
SECOND( time_exp ) (ODBC 1.0)	Returns the second based on the second field in time_exp as an integer value in the range of 0–59.
MONTHNAME( date_exp ) (ODBC 2.0)	Returns a character string that contains the data source-specific name of the month (for example, January through December or Jan. through Dec. for a data source that uses English, or Januar through Dezember for a data source that uses German) for the month part of date_exp.
QUARTER( date_exp ) (ODBC 1.0)	Returns the quarter in date_exp as an integer value in the range of 1–4, where 1 represents January 1 through March 31.
WEEK( date_exp ) (ODBC 1.0)	Returns the week of the year based on the week field in date_exp as an integer value in the range of 1–53.

## Examples

### A. Using an ODBC function in a stored procedure

The following example uses an ODBC function in a stored procedure:

```
CREATE PROCEDURE dbo.ODBCprocedure
(
    @string_exp nvarchar(4000)
)
AS
SELECT {fn OCTET_LENGTH( @string_exp )};
```

## B. Using an ODBC Function in a user-defined function

The following example uses an ODBC function in a user-defined function:

```
CREATE FUNCTION dbo.ODBCudf
(
    @string_exp nvarchar(4000)
)
RETURNS int
AS
BEGIN
DECLARE @len int
SET @len = (SELECT {fn OCTET_LENGTH( @string_exp )})
RETURN(@len)
END ;

SELECT dbo.ODBCudf('Returns the length.');
--Returns 38
```

## C. Using an ODBC functions in SELECT statements

The following SELECT statements use ODBC functions:

```

DECLARE @string_exp nvarchar(4000) = 'Returns the length.';
SELECT {fn BIT_LENGTH( @string_exp )};
-- Returns 304
SELECT {fn OCTET_LENGTH( @string_exp )};
-- Returns 38

SELECT {fn CONCAT( 'CONCAT ','returns a character string')};
-- Returns CONCAT returns a character string
SELECT {fn TRUNCATE( 100.123456, 4)};
-- Returns 100.123400
SELECT {fn CURRENT_DATE( )};
-- Returns 2007-04-20
SELECT {fn CURRENT_TIME(6)};
-- Returns 10:27:11.973000

DECLARE @date_exp nvarchar(30) = '2007-04-21 01:01:01.1234567';
SELECT {fn DAYNAME( @date_exp )};
-- Returns Saturday
SELECT {fn DAYOFMONTH( @date_exp )};
-- Returns 21
SELECT {fn DAYOFWEEK( @date_exp )};
-- Returns 7
SELECT {fn HOUR( @date_exp )};
-- Returns 1
SELECT {fn MINUTE( @date_exp )};
-- Returns 1
SELECT {fn SECOND( @date_exp )};
-- Returns 1
SELECT {fn MONTHNAME( @date_exp )};
-- Returns April
SELECT {fn QUARTER( @date_exp )};
-- Returns 2
SELECT {fn WEEK( @date_exp )};
-- Returns 16

```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D. Using an ODBC function in a stored procedure

The following example uses an ODBC function in a stored procedure:

```

CREATE PROCEDURE dbo.ODBCprocedure
(
    @string_exp nvarchar(4000)
)
AS
    SELECT {fn BIT_LENGTH( @string_exp )};

```

### E. Using an ODBC Function in a user-defined function

The following example uses an ODBC function in a user-defined function:

```

CREATE FUNCTION dbo.ODBCudf
(
    @string_exp nvarchar(4000)
)
RETURNS int
AS
BEGIN
DECLARE @len int
SET @len = (SELECT {fn BIT_LENGTH( @string_exp )})
RETURN(@len)
END ;
SELECT dbo.ODBCudf('Returns the length in bits.');
--Returns 432

```

## F. Using an ODBC functions in SELECT statements

The following SELECT statements use ODBC functions:

```

DECLARE @string_exp nvarchar(4000) = 'Returns the length.';
SELECT {fn BIT_LENGTH( @string_exp )};
-- Returns 304

SELECT {fn CONCAT( 'CONCAT ','returns a character string')};
-- Returns CONCAT returns a character string
SELECT {fn CURRENT_DATE( )};
-- Returns todays date
SELECT {fn CURRENT_TIME(6)};
-- Returns the time

DECLARE @date_exp nvarchar(30) = '2007-04-21 01:01:01.1234567';
SELECT {fn DAYNAME( @date_exp )};
-- Returns Saturday
SELECT {fn DAYOFMONTH( @date_exp )};
-- Returns 21
SELECT {fn DAYOFWEEK( @date_exp )};
-- Returns 7
SELECT {fn HOUR( @date_exp )};
-- Returns 1
SELECT {fn MINUTE( @date_exp )};
-- Returns 1
SELECT {fn SECOND( @date_exp )};
-- Returns 1
SELECT {fn MONTHNAME( @date_exp )};
-- Returns April
SELECT {fn QUARTER( @date_exp )};
-- Returns 2
SELECT {fn WEEK( @date_exp )};
-- Returns 16

```

## See Also

[Built-in Functions \(Transact-SQL\)](#)

# Ranking Functions (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2012) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

Ranking functions return a ranking value for each row in a partition. Depending on the function that is used, some rows might receive the same value as other rows. Ranking functions are nondeterministic.

Transact-SQL provides the following ranking functions:

RANK	NTILE
DENSE_RANK	ROW_NUMBER

## Examples

The following shows the four ranking functions used in the same query. For function specific examples, see each ranking function.

```
USE AdventureWorks2012;
GO
SELECT p.FirstName, p.LastName
    ,ROW_NUMBER() OVER (ORDER BY a.PostalCode) AS "Row Number"
    ,RANK() OVER (ORDER BY a.PostalCode) AS Rank
    ,DENSE_RANK() OVER (ORDER BY a.PostalCode) AS "Dense Rank"
    ,NTILE(4) OVER (ORDER BY a.PostalCode) AS Quartile
    ,s.SalesYTD
    ,a.PostalCode
FROM Sales.SalesPerson AS s
    INNER JOIN Person.Person AS p
        ON s.BusinessEntityID = p.BusinessEntityID
    INNER JOIN Person.Address AS a
        ON a.AddressID = p.BusinessEntityID
    WHERE TerritoryID IS NOT NULL AND SalesYTD <> 0;
```

Here is the result set.

FIRSTNAME	LASTNAME	ROW NUMBER	RANK	DENSE RANK	QUARTILE	SALESYTD	POSTALCODE
Michael	Blythe	1	1	1	1	4557045.04 59	98027
Linda	Mitchell	2	1	1	1	5200475.23 13	98027
Jillian	Carson	3	1	1	1	3857163.63 32	98027
Garrett	Vargas	4	1	1	1	1764938.98 59	98027

FIRSTNAME	LASTNAME	ROW NUMBER	RANK	DENSE RANK	QUARTILE	SALESYTD	POSTALCODE
Tsvi	Reiter	5	1	1	2	2811012.71 51	98027
Shu	Ito	6	6	2	2	3018725.48 58	98055
José	Saraiva	7	6	2	2	3189356.24 65	98055
David	Campbell	8	6	2	3	3587378.42 57	98055
Tete	Mensa-Annan	9	6	2	3	1931620.18 35	98055
Lynn	Tsoflias	10	6	2	3	1758385.92 6	98055
Rachel	Valdez	11	6	2	4	2241204.04 24	98055
Jae	Pak	12	6	2	4	5015682.37 52	98055
Ranjit	Varkey Chudukatil	13	6	2	4	3827950.23 8	98055

## See Also

[Built-in Functions \(Transact-SQL\)](#)

[OVER Clause \(Transact-SQL\)](#)

# DENSE\_RANK (Transact-SQL)

9/27/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

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.

 [Transact-SQL Syntax Conventions](#)

## 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 PARTITION BY syntax, see [OVER Clause \(Transact-SQL\)](#).

<order\_by\_clause>

Determines the order in which the DENSE\_RANK function is applied to the rows in a partition.

## Return Types

**bigint**

## Remarks

If two or more rows tie for a rank in the same partition, each tied rows receives the same rank. For example, if the two top salespeople have the same SalesYTD value, they are both ranked one. The salesperson with the next highest SalesYTD is ranked number two. This is one more than the number of distinct rows that come before this row. Therefore, the numbers returned by the DENSE\_RANK function do not have gaps and always have consecutive ranks.

The sort order used for the whole query determines the order in which the rows appear in a result. This implies that a row ranked number one does not have to be the first row in the partition.

DENSE\_RANK is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Ranking rows within a partition

The following example ranks the products in inventory the specified inventory locations according to their quantities. The result set is partitioned by `LocationID` and logically ordered by `Quantity`. Notice that products 494 and 495 have the same quantity. Because they are tied, they are both ranked one.

```

USE AdventureWorks2012;
GO
SELECT i.ProductID, p.Name, i.LocationID, i.Quantity
    ,DENSE_RANK() OVER
    (PARTITION BY i.LocationID ORDER BY i.Quantity DESC) AS Rank
FROM Production.ProductInventory AS i
INNER JOIN Production.Product AS p
    ON i.ProductID = p.ProductID
WHERE i.LocationID BETWEEN 3 AND 4
ORDER BY i.LocationID;
GO

```

Here is the result set.

ProductID	Name	LocationID	Quantity	Rank
494	Paint - Silver	3	49	1
495	Paint - Blue	3	49	1
493	Paint - Red	3	41	2
496	Paint - Yellow	3	30	3
492	Paint - Black	3	17	4
495	Paint - Blue	4	35	1
496	Paint - Yellow	4	25	2
493	Paint - Red	4	24	3
492	Paint - Black	4	14	4
494	Paint - Silver	4	12	5

(10 row(s) affected)

## B. Ranking all rows in a result set

The following example returns the top ten employees ranked by their salary. Because a PARTITION BY clause was not specified, the DENSE\_RANK function was applied to all rows in the result set.

```

USE AdventureWorks2012;
GO
SELECT TOP(10) BusinessEntityID, Rate,
    DENSE_RANK() OVER (ORDER BY Rate DESC) AS RankBySalary
FROM HumanResources.EmployeePayHistory;

```

Here is the result set.

BusinessEntityID	Rate	RankBySalary
1	125.50	1
25	84.1346	2
273	72.1154	3
2	63.4615	4
234	60.0962	5
263	50.4808	6
7	50.4808	6
234	48.5577	7
285	48.101	8
274	48.101	8

## C. Four ranking functions used in the same query

The following shows the four ranking functions used in the same query. For function specific examples, see each ranking function.

```

USE AdventureWorks2012;
GO
SELECT p.FirstName, p.LastName
    ,ROW_NUMBER() OVER (ORDER BY a.PostalCode) AS "Row Number"
    ,RANK() OVER (ORDER BY a.PostalCode) AS Rank
    ,DENSE_RANK() OVER (ORDER BY a.PostalCode) AS "Dense Rank"
    ,NTILE(4) OVER (ORDER BY a.PostalCode) AS Quartile
    ,s.SalesYTD
    ,a.PostalCode
FROM Sales.SalesPerson AS s
    INNER JOIN Person.Person AS p
        ON s.BusinessEntityID = p.BusinessEntityID
    INNER JOIN Person.Address AS a
        ON a.AddressID = p.BusinessEntityID
WHERE TerritoryID IS NOT NULL AND SalesYTD <> 0;

```

Here is the result set.

FIRSTNAME	LASTNAME	ROW NUMBER	RANK	DENSE RANK	QUARTILE	SALESYTD	POSTALCODE
Michael	Blythe	1	1	1	1	4557045.0 459	98027
Linda	Mitchell	2	1	1	1	5200475.2 313	98027
Jillian	Carson	3	1	1	1	3857163.6 332	98027
Garrett	Vargas	4	1	1	1	1764938.9 859	98027
Tsvi	Reiter	5	1	1	2	2811012.7 151	98027
Shu	Ito	6	6	2	2	3018725.4 858	98055
José	Saraiva	7	6	2	2	3189356.2 465	98055
David	Campbell	8	6	2	3	3587378.4 257	98055
Tete	Mensa-Annan	9	6	2	3	1931620.1 835	98055
Lynn	Tsoflias	10	6	2	3	1758385.9 26	98055
Rachel	Valdez	11	6	2	4	2241204.0 424	98055
Jae	Pak	12	6	2	4	5015682.3 752	98055

FIRSTNAME	LASTNAME	ROW NUMBER	RANK	DENSE RANK	QUARTILE	SALESYTD	POSTALCODE
Ranjit	Varkey Chudukatil	13	6	2	4	3827950.2 38	98055

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D: Ranking rows within a partition

The following example ranks the sales representatives in each sales territory according to their total sales. The rowset is partitioned by `SalesTerritoryGroup` and sorted by `SalesAmountQuota`.

```
-- Uses AdventureWorks

SELECT LastName, SUM(SalesAmountQuota) AS TotalSales, SalesTerritoryGroup,
       DENSE_RANK() OVER (PARTITION BY SalesTerritoryGroup ORDER BY SUM(SalesAmountQuota) DESC ) AS RankResult
  FROM dbo.DimEmployee AS e
  INNER JOIN dbo.FactSalesQuota AS sq ON e.EmployeeKey = sq.EmployeeKey
  INNER JOIN dbo.DimSalesTerritory AS st ON e.SalesTerritoryKey = st.SalesTerritoryKey
 WHERE SalesPersonFlag = 1 AND SalesTerritoryGroup != N'NA'
 GROUP BY LastName,SalesTerritoryGroup;
```

Here is the result set.

LastName	TotalSales	SalesTerritoryGroup	RankResult
----------	------------	---------------------	------------

Pak	10514000.0000	Europe	1
Varkey Chudukatil	5557000.0000	Europe	2
Valdez	2287000.0000	Europe	3
Carson	12198000.0000	North America	1
Mitchell	11786000.0000	North America	2
Blythe	11162000.0000	North America	3
Reiter	8541000.0000	North America	4
Ito	7804000.0000	North America	5
Saraiva	7098000.0000	North America	6
Vargas	4365000.0000	North America	7
Campbell	4025000.0000	North America	8
Ansman-Wolfe	3551000.0000	North America	9
Mensa-Annan	2753000.0000	North America	10
Tsoflias	1687000.0000	Pacific	1

## See Also

[RANK \(Transact-SQL\)](#)

[ROW\\_NUMBER \(Transact-SQL\)](#)

[NTILE \(Transact-SQL\)](#)

[Ranking Functions \(Transact-SQL\)](#)

[Functions](#)

# NTILE (Transact-SQL)

9/27/2017 • 5 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

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.

[Transact-SQL Syntax Conventions](#)

## 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**.

<partition\_by\_clause>

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

<order\_by\_clause>

Determines the order in which the NTILE values are assigned to the rows in a partition. An integer cannot represent a column when the <order\_by\_clause> is used in a ranking function.

## Return Types

**bigint**

## Remarks

If the number of rows in a partition is not divisible by *integer\_expression*, this will cause groups of two sizes that differ by one member. Larger groups come before smaller groups in the order specified by the OVER clause. For example if the total number of rows is 53 and the number of groups is five, the first three groups will have 11 rows and the two remaining groups will have 10 rows each. If on the other hand the total number of rows is divisible by the number of groups, the rows will be evenly distributed among the groups. For example, if the total number of rows is 50, and there are five groups, each bucket will contain 10 rows.

NTILE is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Dividing rows into groups

The following example divides rows into four groups of employees based on their year-to-date sales. Because the total number of rows is not divisible by the number of groups, the first two groups have four rows and the remaining groups have three rows each.

```

USE AdventureWorks2012;
GO
SELECT p.FirstName, p.LastName
    ,NTILE(4) OVER(ORDER BY SalesYTD DESC) AS Quartile
    ,CONVERT(nvarchar(20),s.SalesYTD,1) AS SalesYTD
    ,a.PostalCode
FROM Sales.SalesPerson AS s
INNER JOIN Person.Person AS p
    ON s.BusinessEntityID = p.BusinessEntityID
INNER JOIN Person.Address AS a
    ON a.AddressID = p.BusinessEntityID
WHERE TerritoryID IS NOT NULL
    AND SalesYTD <> 0;
GO

```

Here is the result set.

FirstName	LastName	Quartile	SalesYTD	PostalCode
Linda	Mitchell	1	4,251,368.55	98027
Jae	Pak	1	4,116,871.23	98055
Michael	Blythe	1	3,763,178.18	98027
Jillian	Carson	1	3,189,418.37	98027
Ranjit	Varkey Chudukatil	2	3,121,616.32	98055
José	Saraiva	2	2,604,540.72	98055
Shu	Ito	2	2,458,535.62	98055
Tsvi	Reiter	2	2,315,185.61	98027
Rachel	Valdez	3	1,827,066.71	98055
Tete	Mensa-Annan	3	1,576,562.20	98055
David	Campbell	3	1,573,012.94	98055
Garrett	Vargas	4	1,453,719.47	98027
Lynn	Tsoflias	4	1,421,810.92	98055
Pamela	Anzman-Wolfe	4	1,352,577.13	98027
(14 row(s) affected)				

## B. Dividing the result set by using PARTITION BY

The following example adds the `PARTITION BY` argument to the code in example A. The rows are first partitioned by `PostalCode` and then divided into four groups within each `PostalCode`. The example also declares a variable `@NTILE_Var` and uses that variable to specify the value for the `integer_expression` parameter.

```

USE AdventureWorks2012;
GO
DECLARE @NTILE_Var int = 4;

SELECT p.FirstName, p.LastName
    ,NTILE(@NTILE_Var) OVER(PARTITION BY PostalCode ORDER BY SalesYTD DESC) AS Quartile
    ,CONVERT(nvarchar(20),s.SalesYTD,1) AS SalesYTD
    ,a.PostalCode
FROM Sales.SalesPerson AS s
INNER JOIN Person.Person AS p
    ON s.BusinessEntityID = p.BusinessEntityID
INNER JOIN Person.Address AS a
    ON a.AddressID = p.BusinessEntityID
WHERE TerritoryID IS NOT NULL
    AND SalesYTD <> 0;
GO

```

Here is the result set.

FirstName	LastName	Quartile	SalesYTD	PostalCode
Linda	Mitchell	1	4,251,368.55	98027
Michael	Blythe	1	3,763,178.18	98027
Jillian	Carson	2	3,189,418.37	98027
Tsvi	Reiter	2	2,315,185.61	98027
Garrett	Vargas	3	1,453,719.47	98027
Pamela	Ansman-Wolfe	4	1,352,577.13	98027
Jae	Pak	1	4,116,871.23	98055
Ranjit	Varkey Chudukatil	1	3,121,616.32	98055
José	Saraiva	2	2,604,540.72	98055
Shu	Ito	2	2,458,535.62	98055
Rachel	Valdez	3	1,827,066.71	98055
Tete	Mensa-Annan	3	1,576,562.20	98055
David	Campbell	4	1,573,012.94	98055
Lynn	Tsoflias	4	1,421,810.92	98055

(14 row(s) affected)

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Dividing rows into groups

The following example uses the NTILE function to divide a set of salespersons into four groups based on their assigned sales quota for the year 2003. Because the total number of rows is not divisible by the number of groups, the first group has five rows and the remaining groups have four rows each.

```
-- Uses AdventureWorks

SELECT e.LastName, NTILE(4) OVER(ORDER BY SUM(SalesAmountQuota) DESC) AS Quartile,
       CONVERT (varchar(13), SUM(SalesAmountQuota), 1) AS SalesQuota
  FROM dbo.DimEmployee AS e
 INNER JOIN dbo.FactSalesQuota AS sq
    ON e.EmployeeKey = sq.EmployeeKey
   WHERE sq.CalendarYear = 2003
      AND SalesTerritoryKey IS NOT NULL AND SalesAmountQuota <> 0
 GROUP BY e.LastName
 ORDER BY Quartile, e.LastName;
```

Here is the result set.

LastName	Quartile	SalesYTD
Blythe	1	4,716,000.00
Carson	1	4,350,000.00
Mitchell	1	4,682,000.00
Pak	1	5,142,000.00
Varkey Chudukatil	1	2,940,000.00
Ito	2	2,644,000.00
Reiter	2	2,768,000.00
Saraiva	2	2,293,000.00
Vargas	2	1,617,000.00

Anzman-Wolfe 3 1,183,000.00

Campbell 3 1,438,000.00

Mensa-Annan 3 1,481,000.00

Valdez 3 1,294,000.00

Abbas 4 172,000.00

Albert 4 651,000.00

Jiang 4 544,000.00

Tsoflias 4 867,000.00

#### D. Dividing the result set by using PARTITION BY

The following example adds the PARTITION BY argument to the code in example A. The rows are first partitioned by `SalesTerritoryCountry` and then divided into two groups within each `SalesTerritoryCountry`. Notice that the ORDER BY in the OVER clause orders the NTILE and the ORDER BY of the SELECT statement orders the result set.

```
-- Uses AdventureWorks

SELECT e.LastName, NTILE(2) OVER(PARTITION BY e.SalesTerritoryKey ORDER BY SUM(SalesAmountQuota) DESC) AS
Quartile,
    CONVERT (varchar(13), SUM(SalesAmountQuota), 1) AS SalesQuota
    ,st.SalesTerritoryCountry
FROM dbo.DimEmployee AS e
INNER JOIN dbo.FactSalesQuota AS sq
    ON e.EmployeeKey = sq.EmployeeKey
INNER JOIN dbo.DimSalesTerritory AS st
    ON e.SalesTerritoryKey = st.SalesTerritoryKey
WHERE sq.CalendarYear = 2003
GROUP BY e.LastName,e.SalesTerritoryKey,st.SalesTerritoryCountry
ORDER BY st.SalesTerritoryCountry, Quartile;
```

Here is the result set.

LastName Quartile SalesYTD SalesTerritoryCountry

Tsoflias 1 867,000.00 Australia

Saraiva 1 2,293,000.00 Canada

Varkey Chudukatil 1 2,940,000.00 France

Valdez 1 1,294,000.00 Germany

Alberts 1 651,000.00 NA

Jiang 1 544,000.00 NA

Pak 1 5,142,000.00 United Kingdom

Mensa-Annan 1 1,481,000.00 United States

Campbell 1 1,438,000.00 United States

Reiter 1 2,768,000.00 United States

Blythe 1 4,716,000.00 United States

Carson 1 4,350,000.00 United States

Mitchell 1 4,682,000.00 United States

Vargas 2 1,617,000.00 Canada

Abbas 2 172,000.00 NA

Ito 2 2,644,000.00 United States

Anzman-Wolfe 2 1,183,000.00 United States

## See Also

[RANK \(Transact-SQL\)](#)

[DENSE\\_RANK \(Transact-SQL\)](#)

[ROW\\_NUMBER \(Transact-SQL\)](#)

[Ranking Functions \(Transact-SQL\)](#)

[Built-in Functions \(Transact-SQL\)](#)

# RANK (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

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.

ROW\_NUMBER and RANK are similar. ROW\_NUMBER numbers all rows sequentially (for example 1, 2, 3, 4, 5). RANK provides the same numeric value for ties (for example 1, 2, 2, 4, 5).

## NOTE

RANK is a temporary value calculated when the query is run. To persist numbers in a table, see [IDENTITY Property](#) and [SEQUENCE](#).



## Syntax

```
RANK ( ) OVER ( [ partition_by_clause ] order_by_clause )
```

## Arguments

`OVER ( [ partition_by_clause ] order_by_clause )`

*partition\_by\_clause* divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group. *order\_by\_clause* determines the order of the data before the function is applied. The *order\_by\_clause* is required. The <rows or range clause> of the OVER clause cannot be specified for the RANK function. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

**bigint**

## Remarks

If two or more rows tie for a rank, each tied rows receives the same rank. For example, if the two top salespeople have the same SalesYTD value, they are both ranked one. The salesperson with the next highest SalesYTD is ranked number three, because there are two rows that are ranked higher. Therefore, the RANK function does not always return consecutive integers.

The sort order that is used for the whole query determines the order in which the rows appear in a result set.

RANK is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Ranking rows within a partition

The following example ranks the products in inventory the specified inventory locations according to their quantities. The result set is partitioned by `LocationID` and logically ordered by `Quantity`. Notice that products 494 and 495 have the same quantity. Because they are tied, they are both ranked one.

```
USE AdventureWorks2012;
GO
SELECT i.ProductID, p.Name, i.LocationID, i.Quantity
    ,RANK() OVER
    (PARTITION BY i.LocationID ORDER BY i.Quantity DESC) AS Rank
FROM Production.ProductInventory AS i
INNER JOIN Production.Product AS p
    ON i.ProductID = p.ProductID
WHERE i.LocationID BETWEEN 3 AND 4
ORDER BY i.LocationID;
GO
```

Here is the result set.

ProductID	Name	LocationID	Quantity	Rank
494	Paint - Silver	3	49	1
495	Paint - Blue	3	49	1
493	Paint - Red	3	41	3
496	Paint - Yellow	3	30	4
492	Paint - Black	3	17	5
495	Paint - Blue	4	35	1
496	Paint - Yellow	4	25	2
493	Paint - Red	4	24	3
492	Paint - Black	4	14	4
494	Paint - Silver	4	12	5

(10 row(s) affected)

## B. Ranking all rows in a result set

The following example returns the top ten employees ranked by their salary. Because a PARTITION BY clause was not specified, the RANK function was applied to all rows in the result set.

```
USE AdventureWorks2012
SELECT TOP(10) BusinessEntityID, Rate,
    RANK() OVER (ORDER BY Rate DESC) AS RankBySalary
FROM HumanResources.EmployeePayHistory AS eph1
WHERE RateChangeDate = (SELECT MAX(RateChangeDate)
    FROM HumanResources.EmployeePayHistory AS eph2
    WHERE eph1.BusinessEntityID = eph2.BusinessEntityID)
ORDER BY BusinessEntityID;
```

Here is the result set.

BusinessEntityID	Rate	RankBySalary
1	125.50	1
2	63.4615	4
3	43.2692	8
4	29.8462	19
5	32.6923	16
6	32.6923	16
7	50.4808	6
8	40.8654	10
9	40.8654	10
10	42.4808	9

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C: Ranking rows within a partition

The following example ranks the sales representatives in each sales territory according to their total sales. The rowset is partitioned by `SalesTerritoryGroup` and sorted by `SalesAmountQuota`.

```
-- Uses AdventureWorks

SELECT LastName, SUM(SalesAmountQuota) AS TotalSales, SalesTerritoryRegion,
       RANK() OVER (PARTITION BY SalesTerritoryRegion ORDER BY SUM(SalesAmountQuota) DESC ) AS RankResult
FROM dbo.DimEmployee AS e
INNER JOIN dbo.FactSalesQuota AS sq ON e.EmployeeKey = sq.EmployeeKey
INNER JOIN dbo.DimSalesTerritory AS st ON e.SalesTerritoryKey = st.SalesTerritoryKey
WHERE SalesPersonFlag = 1 AND SalesTerritoryRegion != N'NA'
GROUP BY LastName, SalesTerritoryRegion;
```

Here is the result set.

LastName	TotalSales	SalesTerritoryGroup	RankResult
----------	------------	---------------------	------------

Tsoflias	1687000.0000	Australia	1
----------	--------------	-----------	---

Saraiva	7098000.0000	Canada	1
---------	--------------	--------	---

Vargas	4365000.0000	Canada	2
--------	--------------	--------	---

Carson	12198000.0000	Central	1
--------	---------------	---------	---

Varkey	Chudukatil	5557000.0000	France	1
--------	------------	--------------	--------	---

Valdez	2287000.0000	Germany	1
--------	--------------	---------	---

Blythe	11162000.0000	Northeast	1
--------	---------------	-----------	---

Campbell	4025000.0000	Northwest	1
----------	--------------	-----------	---

Ansman-Wolfe	3551000.0000	Northwest	2
--------------	--------------	-----------	---

Mensa-Annan	2753000.0000	Northwest	3
-------------	--------------	-----------	---

Reiter	8541000.0000	Southeast	1
--------	--------------	-----------	---

Mitchell	11786000.0000	Southwest	1
----------	---------------	-----------	---

Ito	7804000.0000	Southwest	2
-----	--------------	-----------	---

## See Also

[DENSE\\_RANK \(Transact-SQL\)](#)  
[ROW\\_NUMBER \(Transact-SQL\)](#)  
[NTILE \(Transact-SQL\)](#)  
[Ranking Functions \(Transact-SQL\)](#)  
[Built-in Functions \(Transact-SQL\)](#)

# ROW\_NUMBER (Transact-SQL)

9/11/2017 • 5 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Numbers the output of a result set. More specifically, returns the sequential number of a row within a partition of a result set, starting at 1 for the first row in each partition.

`ROW_NUMBER` and `RANK` are similar. `ROW_NUMBER` numbers all rows sequentially (for example 1, 2, 3, 4, 5). `RANK` provides the same numeric value for ties (for example 1, 2, 2, 4, 5).

## NOTE

`ROW_NUMBER` is a temporary value calculated when the query is run. To persist numbers in a table, see [IDENTITY Property](#) and [SEQUENCE](#).



## Syntax

```
ROW_NUMBER ( )
OVER ( [ PARTITION BY value_expression , ... [ n ] ] order_by_clause )
```

## Arguments

### PARTITION BY *value\_expression*

Divides the result set produced by the [FROM](#) clause into partitions to which the `ROW_NUMBER` function is applied. *value\_expression* specifies the column by which the result set is partitioned. If `PARTITION BY` is not specified, the function treats all rows of the query result set as a single group. For more information, see [OVER Clause \(Transact-SQL\)](#).

### *order\_by\_clause*

The `ORDER BY` clause determines the sequence in which the rows are assigned their unique `ROW_NUMBER` within a specified partition. It is required. For more information, see [OVER Clause \(Transact-SQL\)](#).

## Return Types

**bigint**

## General Remarks

There is no guarantee that the rows returned by a query using `ROW_NUMBER()` will be ordered exactly the same with each execution unless the following conditions are true.

1. Values of the partitioned column are unique.
2. Values of the `ORDER BY` columns are unique.
3. Combinations of values of the partition column and `ORDER BY` columns are unique.

`ROW_NUMBER()` is nondeterministic. For more information, see [Deterministic and Nondeterministic Functions](#).

## Examples

### A. Simple examples

The following query returns the four system tables in alphabetic order.

```
SELECT
    name, recovery_model_desc
FROM sys.databases
WHERE database_id < 5
ORDER BY name ASC;
```

Here is the result set.

NAME	RECOVERY_MODEL_DESC
master	SIMPLE
model	FULL
msdb	SIMPLE
tempdb	SIMPLE

To add a row number column in front of each row, add a column with the `ROW_NUMBER` function, in this case named `Row#`. You must move the `ORDER BY` clause up to the `OVER` clause.

```
SELECT
    ROW_NUMBER() OVER(ORDER BY name ASC) AS Row#,
    name, recovery_model_desc
FROM sys.databases
WHERE database_id < 5;
```

Here is the result set.

ROW#	NAME	RECOVERY_MODEL_DESC
1	master	SIMPLE
2	model	FULL
3	msdb	SIMPLE
4	tempdb	SIMPLE

Adding a `PARTITION BY` clause on the `recovery_model_desc` column, will restart the numbering when the `recovery_model_desc` value changes.

```

SELECT
    ROW_NUMBER() OVER(PARTITION BY recovery_model_desc ORDER BY name ASC)
        AS Row#,
    name, recovery_model_desc
FROM sys.databases WHERE database_id < 5;

```

Here is the result set.

ROW#	NAME	RECOVERY_MODEL_DESC
1	model	FULL
1	master	SIMPLE
2	msdb	SIMPLE
3	tempdb	SIMPLE

## B. Returning the row number for salespeople

The following example calculates a row number for the salespeople in Adventure Works Cycles based on their year-to-date sales ranking.

```

USE AdventureWorks2012;
GO
SELECT ROW_NUMBER() OVER(ORDER BY SalesYTD DESC) AS Row,
    FirstName, LastName, ROUND(SalesYTD,2,1) AS "Sales YTD"
FROM Sales.vSalesPerson
WHERE TerritoryName IS NOT NULL AND SalesYTD <> 0;

```

Here is the result set.

Row	FirstName	LastName	SalesYTD
1	Linda	Mitchell	4251368.54
2	Jae	Pak	4116871.22
3	Michael	Blythe	3763178.17
4	Jillian	Carson	3189418.36
5	Ranjit	Varkey Chudukatil	3121616.32
6	José	Saraiva	2604540.71
7	Shu	Ito	2458535.61
8	Tsvi	Reiter	2315185.61
9	Rachel	Valdez	1827066.71
10	Tete	Mensa-Annan	1576562.19
11	David	Campbell	1573012.93
12	Garrett	Vargas	1453719.46
13	Lynn	Tsoflias	1421810.92
14	Pamela	Anzman-Wolfe	1352577.13

## C. Returning a subset of rows

The following example calculates row numbers for all rows in the `SalesOrderHeader` table in the order of the `OrderDate` and returns only rows `50` to `60` inclusive.

```

USE AdventureWorks2012;
GO
WITH OrderedOrders AS
(
    SELECT SalesOrderID, OrderDate,
    ROW_NUMBER() OVER (ORDER BY OrderDate) AS RowNumber
    FROM Sales.SalesOrderHeader
)
SELECT SalesOrderID, OrderDate, RowNumber
FROM OrderedOrders
WHERE RowNumber BETWEEN 50 AND 60;

```

#### D. Using `ROW_NUMBER()` with `PARTITION`

The following example uses the `PARTITION BY` argument to partition the query result set by the column `TerritoryName`. The `ORDER BY` clause specified in the `OVER` clause orders the rows in each partition by the column `SalesYTD`. The `ORDER BY` clause in the `SELECT` statement orders the entire query result set by `TerritoryName`.

```

USE AdventureWorks2012;
GO
SELECT FirstName, LastName, TerritoryName, ROUND(SalesYTD,2,1) AS SalesYTD,
ROW_NUMBER() OVER(PARTITION BY TerritoryName ORDER BY SalesYTD DESC)
AS Row
FROM Sales.vSalesPerson
WHERE TerritoryName IS NOT NULL AND SalesYTD <> 0
ORDER BY TerritoryName;

```

Here is the result set.

FirstName	LastName	TerritoryName	SalesYTD	Row
Lynn	Tsoflias	Australia	1421810.92	1
José	Saraiva	Canada	2604540.71	1
Garrett	Vargas	Canada	1453719.46	2
Jillian	Carson	Central	3189418.36	1
Ranjit	Varkey Chudukatil	France	3121616.32	1
Rachel	Valdez	Germany	1827066.71	1
Michael	Blythe	Northeast	3763178.17	1
Tete	Mensa-Annan	Northwest	1576562.19	1
David	Campbell	Northwest	1573012.93	2
Pamela	Anzman-Wolfe	Northwest	1352577.13	3
Tsvi	Reiter	Southeast	2315185.61	1
Linda	Mitchell	Southwest	4251368.54	1
Shu	Ito	Southwest	2458535.61	2
Jae	Pak	United Kingdom	4116871.22	1

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

#### E. Returning the row number for salespeople

The following example returns the `ROW_NUMBER()` for sales representatives based on their assigned sales quota.

```
-- Uses AdventureWorks

SELECT ROW_NUMBER() OVER(ORDER BY SUM(SalesAmountQuota) DESC)
    AS RowNumber,
    FirstName, LastName,
    CONVERT(varchar(13), SUM(SalesAmountQuota),1) AS SalesQuota
FROM dbo.DimEmployee AS e
INNER JOIN dbo.FactSalesQuota AS sq
    ON e.EmployeeKey = sq.EmployeeKey
WHERE e.SalesPersonFlag = 1
GROUP BY LastName, FirstName;
```

Here is a partial result set.

RowNumber	FirstName	LastName	SalesQuota
-----			

1 Jillian Carson 12,198,000.00
--------------------------------

2 Linda Mitchell 11,786,000.00
--------------------------------

3 Michael Blythe 11,162,000.00
--------------------------------

4 Jae Pak 10,514,000.00
-------------------------

## F. Using ROW\_NUMBER() with PARTITION

The following example shows using the `ROW_NUMBER` function with the `PARTITION BY` argument. This causes the `ROW_NUMBER` function to number the rows in each partition.

```
-- Uses AdventureWorks

SELECT ROW_NUMBER() OVER(PARTITION BY SalesTerritoryKey
    ORDER BY SUM(SalesAmountQuota) DESC) AS RowNumber,
    LastName, SalesTerritoryKey AS Territory,
    CONVERT(varchar(13), SUM(SalesAmountQuota),1) AS SalesQuota
FROM dbo.DimEmployee AS e
INNER JOIN dbo.FactSalesQuota AS sq
    ON e.EmployeeKey = sq.EmployeeKey
WHERE e.SalesPersonFlag = 1
GROUP BY LastName, FirstName, SalesTerritoryKey;
```

Here is a partial result set.

RowNumber	LastName	Territory	SalesQuota
-----			

1 Campbell 1 4,025,000.00
---------------------------

2 Ansman-Wolfe 1 3,551,000.00
-------------------------------

3 Mensa-Annan 1 2,275,000.00
------------------------------

1 Blythe 2 11,162,000.00
--------------------------

1 Carson 3 12,198,000.00
--------------------------

1 Mitchell 4 11,786,000.00
----------------------------

2 Ito 4 7,804,000.00
----------------------

## See Also

[RANK \(Transact-SQL\)](#)

[DENSE\\_RANK \(Transact-SQL\)](#)

[NTILE \(Transact-SQL\)](#)

# Replication Functions - PUBLISHINGSERVERNAME

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the name of the originating Publisher for a published database participating in a database mirroring session. This function is executed at a Publisher instance of SQL Server on the publication database. Use it to determine the original Publisher of the published database.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
PUBLISHINGSERVERNAME()
```

## Return Types

**nvarchar**

## Remarks

PUBLISHINGSERVERNAME is used in all types of replication.

PUBLISHINGSERVERNAME is used when a database mirroring session exists on the publication database between the Publisher and a mirror partner instance.

This function must be executed within the context of a publication database. When PUBLISHINGSERVERNAME is executed on a publication database at the mirror server instance of SQL Server, the name of the Publisher instance from which the published database originates is returned. When this function is executed on a database at the mirror server instance that is not published or that is published from the mirror server instance after a failover, the name of the mirror server instance is returned. When this function is executed at the original Publisher instance, the name of the Publisher is returned.

## See Also

[Database Mirroring and Replication \(SQL Server\)](#)

[Replication Functions \(Transact-SQL\)](#)

# Rowset Functions (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following rowset functions return an object that can be used in place of a table reference in a Transact-SQL statement.

<a href="#">OPENDATASOURCE</a>	<a href="#">OPENJSON</a>
<a href="#">OPENROWSET</a>	<a href="#">OPENQUERY</a>
<a href="#">OPENXML</a>	

All rowset functions are nondeterministic. This means these functions do not always return the same results every time they are called, even with the same set of input values. For more information about function determinism, see [Deterministic and Nondeterministic Functions](#).

## See Also

[Built-in Functions \(Transact-SQL\)](#)

# OPENDATASOURCE (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Provides ad hoc connection information as part of a four-part object name without using a linked server name.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
OPENDATASOURCE ( provider_name, init_string )
```

## Arguments

### *provider\_name*

Is the name registered as the PROGID of the OLE DB provider used to access the data source. *provider\_name* is a **char** data type, with no default value.

### *init\_string*

Is the connection string passed to the IDataInitialize interface of the destination provider. The provider string syntax is based on keyword-value pairs separated by semicolons, such as: '*keyword1=value;keyword2=value*'.

For specific keyword-value pairs supported on the provider, see the Microsoft Data Access SDK. This documentation defines the basic syntax. The following table lists the most frequently used keywords in the *init\_string* argument.

KEYWORD	OLE DB PROPERTY	VALID VALUES AND DESCRIPTION
Data Source	DBPROP_INIT_DATASOURCE	Name of the data source to connect to. Different providers interpret this in different ways. For SQL Server Native Client OLE DB provider, this indicates the name of the server. For Jet OLE DB provider, this indicates the full path of the .mdb file or .xls file.
Location	DBPROP_INIT_LOCATION	Location of the database to connect to.
Extended Properties	DBPROP_INIT_PROVIDERSTRING	The provider-specific connect-string.
Connect timeout	DBPROP_INIT_TIMEOUT	Time-out value after which the connection try fails.
User ID	DBPROP_AUTH_USERID	User ID to be used for the connection.
Password	DBPROP_AUTH_PASSWORD	Password to be used for the connection.

KEYWORD	OLE DB PROPERTY	VALID VALUES AND DESCRIPTION
Catalog	DBPROP_INIT_CATALOG	The name of the initial or default catalog when connecting to the data source.
Integrated Security	DBPROP_AUTH_INTEGRATED	SSPI, to specify Windows Authentication

## Remarks

OPENDATASOURCE can be used to access remote data from OLE DB data sources only when the DisallowAdhocAccess registry option is explicitly set to 0 for the specified provider, and the Ad Hoc Distributed Queries advanced configuration option is enabled. When these options are not set, the default behavior does not allow for ad hoc access.

The OPENDATASOURCE function can be used in the same Transact-SQL syntax locations as a linked-server name. Therefore, OPENDATASOURCE can be used as the first part of a four-part name that refers to a table or view name in a SELECT, INSERT, UPDATE, or DELETE statement, or to a remote stored procedure in an EXECUTE statement. When executing remote stored procedures, OPENDATASOURCE should refer to another instance of SQL Server. OPENDATASOURCE does not accept variables for its arguments.

Like the OPENROWSET function, OPENDATASOURCE should only reference OLE DB data sources that are accessed infrequently. Define a linked server for any data sources accessed more than several times. Neither OPENDATASOURCE nor OPENROWSET provide all the functionality of linked-server definitions, such as security management and the ability to query catalog information. All connection information, including passwords, must be provided every time that OPENDATASOURCE is called.

### IMPORTANT

Windows Authentication is much more secure than SQL Server Authentication. You should use Windows Authentication whenever possible. OPENDATASOURCE should not be used with explicit passwords in the connection string.

The connection requirements for each provider are similar to the requirements for those parameters when creating linked servers. The details for many common providers are listed in the topic [sp\\_addlinkedserver \(Transact-SQL\)](#).

Any call to OPENDATASOURCE, OPENQUERY, or OPENROWSET in the FROM clause is evaluated separately and independently from any call to these functions used as the target of the update, even if identical arguments are supplied to the two calls. In particular, filter or join conditions applied on the result of one of those calls have no effect on the results of the other.

## Permissions

Any user can execute OPENDATASOURCE. The permissions that are used to connect to the remote server are determined from the connection string.

## Examples

The following example creates an ad hoc connection to the `Payroll` instance of SQL Server on server `London`, and queries the `AdventureWorks2012.HumanResources.Employee` table. (Use SQLNCLI and SQL Server will redirect to the latest version of SQL Server Native Client OLE DB Provider.)

```
SELECT *
FROM OPENDATASOURCE('SQLNCLI',
'Data Source=London\Payroll;Integrated Security=SSPI')
.AdventureWorks2012.HumanResources.Employee
```

The following example creates an ad hoc connection to an Excel spreadsheet in the 1997 - 2003 format.

```
SELECT * FROM OPENDATASOURCE('Microsoft.Jet.OLEDB.4.0',
'Data Source=C:\DataFolder\Documents\TestExcel.xls;Extended Properties=EXCEL 5.0')...[Sheet1$] ;
```

## See Also

[OPENROWSET \(Transact-SQL\)](#)  
[sp\\_addlinkedserver \(Transact-SQL\)](#)

# OPENJSON (Transact-SQL)

9/18/2017 • 10 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2016) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

**OPENJSON** is a table-valued function that parses JSON text and returns objects and properties from the JSON input as rows and columns. In other words, **OPENJSON** provides a rowset view over a JSON document. You can explicitly specify the columns in the rowset and the JSON property paths used to populate the columns. Since **OPENJSON** returns a set of rows, you can use **OPENJSON** in the `FROM` clause of a Transact-SQL statement just as you can use any other table, view, or table-valued function.

Use **OPENJSON** to import JSON data into SQL Server, or to convert JSON data to relational format for an app or service that can't consume JSON directly.

## NOTE

The **OPENJSON** function is available only under compatibility level 130 or higher. If your database compatibility level is lower than 130, SQL Server can't find and run the **OPENJSON** function. Other JSON functions are available at all compatibility levels.

You can check compatibility level in the `sys.databases` view or in database properties. You can change the compatibility level of a database with the following command:

```
ALTER DATABASE DatabaseName SET COMPATIBILITY_LEVEL = 130
```

Compatibility level 120 may be the default even in a new Azure SQL Database.

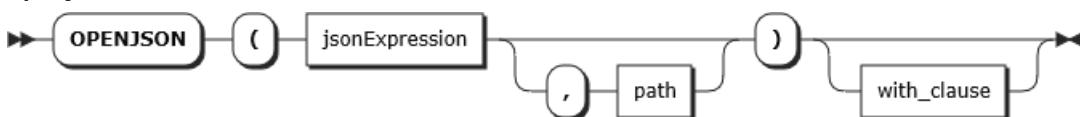
## Transact-SQL Syntax Conventions

## Syntax

```
OPENJSON( jsonExpression [ , path ] ) [ <with_clause> ]  
<with_clause> ::= WITH ( { colName type [ column_path ] [ AS JSON ] } [ ,...n ] )
```

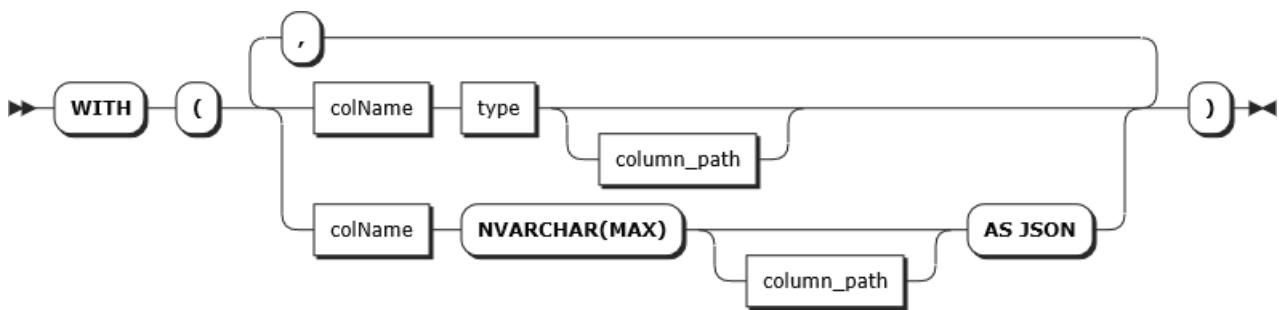
The **OPENJSON** table-valued function parses the *jsonExpression* provided as the first argument and returns one or more rows containing data from the JSON objects in the expression. *jsonExpression* can contain nested sub-objects. If you want to parse a sub-object from within *jsonExpression*, you can specify a **path** parameter for the JSON sub-object.

### openjson



By default, the **OPENJSON** table-valued function returns three columns, which contain the key name, the value, and the type of each {key:value} pair found in *jsonExpression*. As an alternative, you can explicitly specify the schema of the result set that **OPENJSON** returns by providing *with\_clause*.

### with\_clause



*with\_clause* contains a list of columns with their types for **OPENJSON** to return. By default, **OPENJSON** matches keys in *jsonExpression* with the column names in *with\_clause* (in this case, matches keys implies that it is case sensitive). If a column name does not match a key name, you can provide an optional *column\_path*, which is a [JSON Path Expression](#) that references a key within the *jsonExpression*.

## Arguments

### *jsonExpression*

Is a Unicode character expression containing JSON text.

OPENJSON iterates over the elements of the array or the properties of the object in the JSON expression and returns one row for each element or property. The following example returns each property of the object provided as *jsonExpression*:

```

DECLARE @json NVARCHAR(4000) = N'{
    "StringValue": "John",
    "IntValue": 45,
    "TrueValue": true,
    "FalseValue": false,
    "NullValue": null,
    "ArrayValue": ["a", "r", "r", "a", "y"],
    "ObjectValue": {"obj": "ect"}
}'

SELECT *
FROM OPENJSON(@json)
    
```

## Results

KEY	VALUE	TYPE
StringValue	John	1
IntValue	Doe	2
TrueValue	true	3
FalseValue	false	3
NullValue	NULL	0
ArrayValue	["a", "r", "r", "a", "y"]	4
ObjectValue	{"obj": "ect"}	5

### *path*

Is an optional JSON path expression that references an object or an array within *jsonExpression*. **OPENJSON** seeks into the JSON text at the specified position and parses only the referenced fragment. For more info, see [JSON Path Expressions \(SQL Server\)](#).

In SQL Server 2017 and in Azure SQL Database, you can provide a variable as the value of *path*.

The following example returns a nested object by specifying the *path*:

```
DECLARE @json NVARCHAR(4000) = N'{  
    "path": {  
        "to": {  
            "sub-object": ["en-GB", "en-UK", "de-AT", "es-AR", "sr-Cyr1"]  
        }  
    }';  
  
SELECT [key], value  
FROM OPENJSON(@json, '$.path.to."sub-object"')
```

## Results

KEY	VALUE
0	en-GB
1	en-UK
2	de-AT
3	es-AR
4	sr-Cyr1

When **OPENJSON** parses a JSON array, the function returns the indexes of the elements in the JSON text as keys.

The comparison used to match path steps with the properties of the JSON expression is case-sensitive and collation-unaware (that is, a BIN2 comparison).

### *with\_clause*

Explicitly defines the output schema for the **OPENJSON** function to return. The optional *with\_clause* can contain the following elements:

*colName* Is the name for the output column.

By default, **OPENJSON** uses the name of the column to match a property in the JSON text. For example, if you specify the column *name* in the schema, OPENJSON tries to populate this column with the property "name" in the JSON text. You can override this default mapping by using the *column\_path* argument.

### *type*

Is the data type for the output column.

#### NOTE

If you also use the **AS JSON** option, the column *type* must be `NVARCHAR(MAX)`.

### *column\_path*

Is the JSON path that specifies the property to return in the specified column. For more info, see the description of

the *path* parameter previously in this topic.

Use *column\_path* to override default mapping rules when the name of an output column doesn't match the name of the property.

The comparison used to match path steps with the properties of the JSON expression is case-sensitive and collation-unaware (that is, a BIN2 comparison).

For more info about paths, see [JSON Path Expressions \(SQL Server\)](#).

#### AS JSON

Use the **AS JSON** option in a column definition to specify that the referenced property contains an inner JSON object or array. If you specify the **AS JSON** option, the type of the column must be NVARCHAR(MAX).

- If you don't specify **AS JSON** for a column, the function returns a scalar value (for example, int, string, true, false) from the specified JSON property on the specified path. If the path represents an object or an array, and the property can't be found at the specified path, the function returns null in lax mode or returns an error in strict mode. This behavior is similar to the behavior of the **JSON\_VALUE** function.
- If you specify **AS JSON** for a column, the function returns a JSON fragment from the specified JSON property on the specified path. If the path represents a scalar value, and the property can't be found at the specified path, the function returns null in lax mode or returns an error in strict mode. This behavior is similar to the behavior of the **JSON\_QUERY** function.

#### NOTE

If you want to return a nested JSON fragment from a JSON property, you have to provide the **AS JSON** flag. Without this option, if the property can't be found, OPENJSON returns a NULL value instead of the referenced JSON object or array, or it returns a run-time error in strict mode.

For example, the following query returns and formats the elements of an array:

```

DECLARE @json NVARCHAR(MAX) = N'[
{
    "Order": {
        "Number": "SO43659",
        "Date": "2011-05-31T00:00:00"
    },
    "AccountNumber": "AW29825",
    "Item": {
        "Price": 2024.9940,
        "Quantity": 1
    }
},
{
    "Order": {
        "Number": "SO43661",
        "Date": "2011-06-01T00:00:00"
    },
    "AccountNumber": "AW73565",
    "Item": {
        "Price": 2024.9940,
        "Quantity": 3
    }
}
]'

SELECT *
FROM OPENJSON ( @json )
WITH (
    Number      varchar(200)      '$.Order.Number',
    Date        datetime         '$.Order.Date',
    Customer    varchar(200)      '$.AccountNumber',
    Quantity    int              '$.Item.Quantity',
    [Order]     nvarchar(MAX)    AS JSON
)

```

## Results

NUMBER	DATE	CUSTOMER	QUANTITY	ORDER
SO43659	2011-05-31T00:00:00	AW29825	1	{"Number": "SO43659", "Date": "2011-05-31T00:00:00"}
SO43661	2011-06-01T00:00:00	AW73565	3	{"Number": "SO43661", "Date": "2011-06-01T00:00:00"}

## Return Value

The columns that the OPENJSON function returns depend on the WITH option.

1. When you call OPENJSON with the default schema - that is, when you don't specify an explicit schema in the WITH clause - the function returns a table with the following columns:
  - a. **Key**. An nvarchar(4000) value that contains the name of the specified property or the index of the element in the specified array. The key column has a BIN2 collation.
  - b. **Value**. An nvarchar(max) value that contains the value of the property. The value column inherits its collation from *jsonExpression*.
  - c. **Type**. An int value that contains the type of the value. The **Type** column is returned only when you use OPENJSON with the default schema. The type column has one of the following values:

VALUE OF THE TYPE COLUMN	JSON DATA TYPE
0	null
1	string
2	int
3	true/false
4	array
5	object

Only first-level properties are returned. The statement fails if the JSON text is not properly formatted.

2. When you call OPENJSON and you specify an explicit schema in the WITH clause, the function returns a table with the schema that you defined in the WITH clause.

## Remarks

*json\_path* used in the second argument of **OPENJSON** or in *with\_clause* can start with the **lax** or **strict** keyword.

- In **lax** mode, **OPENJSON** doesn't raise an error if the object or value on the specified path can't be found. If the path can't be found, **OPENJSON** returns either an empty result set or a NULL value.
- In **strict**, mode **OPENJSON** returns an error if the path can't be found.

Some of the examples on this page explicitly specify the path mode, lax or strict. The path mode is optional. If you don't explicitly specify a path mode, lax mode is the default. For more info about path mode and path expressions, see [JSON Path Expressions \(SQL Server\)](#).

Column names in *with\_clause* are matched with keys in the JSON text. If you specify the column name `[Address.Country]`, it's matched with the key `Address.Country`. If you want to reference a nested key `Country` within the object `Address`, you have to specify the path `$.Address.Country` in column path.

*json\_path* may contain keys with alphanumeric characters. Escape the key name in *json\_path* with double quotes if you have special characters in the keys. For example, `$. "my key $1".regularKey."key with . dot"` matches value 1 in the following JSON text:

```
{
  "my key $1": {
    "regularKey":{
      "key with . dot": 1
    }
  }
}
```

## Examples

### Example 1 - Convert a JSON array to a temporary table

The following example provides a list of identifiers as a JSON array of numbers. The query converts the JSON array to a table of identifiers and filters all products with the specified ids.

```

DECLARE @pSearchOptions NVARCHAR(4000) = N'[1,2,3,4]'

SELECT *
FROM products
INNER JOIN OPENJSON(@pSearchOptions) AS productTypes
ON product.productTypeID = productTypes.value

```

This query is equivalent to the following example. However, in the example below, you have to embed numbers in the query instead of passing them as parameters.

```

SELECT *
FROM products
WHERE product.productTypeID IN (1,2,3,4)

```

### Example 2 - Merge properties from two JSON objects

The following example selects a union of all the properties of two JSON objects. The two objects have a duplicate *name* property. The example uses the key value to exclude the duplicate row from the results.

```

DECLARE @json1 NVARCHAR(MAX),@json2 NVARCHAR(MAX)

SET @json1=N'{"name": "John", "surname":"Doe"}'

SET @json2=N'{"name": "John", "age":45}'

SELECT *
FROM OPENJSON(@json1)
UNION ALL
SELECT *
FROM OPENJSON(@json2)
WHERE [key] NOT IN (SELECT [key] FROM OPENJSON(@json1))

```

### Example 3 - Join rows with JSON data stored in table cells using CROSS APPLY

In the following example, the `SalesOrderHeader` table has a `SalesReason` text column that contains an array of `SalesOrderReasons` in JSON format. The `SalesOrderReasons` objects contain properties like *Quality* and *Manufacturer*. The example creates a report that joins every sales order row to the related sales reasons. The `OPENJSON` operator expands the JSON array of sales reasons as if the reasons were stored in a separate child table. Then the `CROSS APPLY` operator joins each sales order row to the rows returned by the `OPENJSON` table-valued function.

```

SELECT SalesOrderID,OrderDate,value AS Reason
FROM Sales.SalesOrderHeader
CROSS APPLY OPENJSON(SalesReasons)

```

#### TIP

When you have to expand JSON arrays stored in individual fields and join them with their parent rows, you typically use the Transact-SQL `CROSS APPLY` operator. For more info about `CROSS APPLY`, see [FROM \(Transact-SQL\)](#).

The same query can be rewritten by using `OPENJSON` with an explicitly defined schema of rows to return:

```

SELECT SalesOrderID, OrderDate, value AS Reason
FROM Sales.SalesOrderHeader
CROSS APPLY OPENJSON (SalesReasons) WITH (value nvarchar(100) '$')

```

In this example, the `$` path references each element in the array. If you want to explicitly cast the returned value, you can use this type of query.

#### Example 4 - Combine relational rows and JSON elements with CROSS APPLY

The following query combines relational rows and JSON elements into the results shown in the following table.

```
SELECT store.title, location.street, location.lat, location.long
FROM store
CROSS APPLY OPENJSON(store.jsonCol, 'lax $.location')
    WITH (street varchar(500) , postcode varchar(500) '$.postcode' ,
    lon int '$.geo.longitude', lat int '$.geo.latitude')
    AS location
```

#### Results

TITLE	STREET	POSTCODE	LON	LAT
Whole Food Markets	17991 Redmond Way	WA 98052	47.666124	-122.10155
Sears	148th Ave NE	WA 98052	47.63024	-122.141246,17

#### Example 5 - Import JSON data into SQL Server

The following example loads an entire JSON object into a SQL Server table.

```
DECLARE @json NVARCHAR(max) = N'{  
    "id" : 2,  
    "firstName": "John",  
    "lastName": "Smith",  
    "isAlive": true,  
    "age": 25,  
    "dateOfBirth": "2015-03-25T12:00:00",  
    "spouse": null  
}';  
  
INSERT INTO Person  
SELECT *  
FROM OPENJSON(@json)  
WITH (id int,  
      firstName nvarchar(50), lastName nvarchar(50),  
      isAlive bit, age int,  
      dateOfBirth datetime2, spouse nvarchar(50))
```

## See Also

[JSON Path Expressions \(SQL Server\)](#)

[Convert JSON Data to Rows and Columns with OPENJSON \(SQL Server\)](#)

[Use OPENJSON with the Default Schema \(SQL Server\)](#)

[Use OPENJSON with an Explicit Schema \(SQL Server\)](#)

# OPENQUERY (Transact-SQL)

10/2/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Executes the specified pass-through query on the specified linked server. This server is an OLE DB data source. OPENQUERY can be referenced in the FROM clause of a query as if it were a table name. OPENQUERY can also be referenced as the target table of an INSERT, UPDATE, or DELETE statement. This is subject to the capabilities of the OLE DB provider. Although the query may return multiple result sets, OPENQUERY returns only the first one.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
OPENQUERY ( linked_server , 'query' )
```

## Arguments

*linked\_server*

Is an identifier representing the name of the linked server.

*'query'*

Is the query string executed in the linked server. The maximum length of the string is 8 KB.

## Remarks

OPENQUERY does not accept variables for its arguments.

OPENQUERY cannot be used to execute extended stored procedures on a linked server. However, an extended stored procedure can be executed on a linked server by using a four-part name. For example:

```
EXEC SeattleSales.master.dbo.xp_msver
```

Any call to OPENDATASOURCE, OPENQUERY, or OPENROWSET in the FROM clause is evaluated separately and independently from any call to these functions used as the target of the update, even if identical arguments are supplied to the two calls. In particular, filter or join conditions applied on the result of one of those calls have no effect on the results of the other.

## Permissions

Any user can execute OPENQUERY. The permissions that are used to connect to the remote server are obtained from the settings defined for the linked server.

## Examples

### A. Executing an UPDATE pass-through query

The following example uses a pass-through `UPDATE` query against the linked server created in example A.

```
UPDATE OPENQUERY (OracleSvr, 'SELECT name FROM joe.titles WHERE id = 101')
SET name = 'ADifferentName';
```

## B. Executing an INSERT pass-through query

The following example uses a pass-through `INSERT` query against the linked server created in example A.

```
INSERT OPENQUERY (OracleSvr, 'SELECT name FROM joe.titles')
VALUES ('NewTitle');
```

## C. Executing a DELETE pass-through query

The following example uses a pass-through `DELETE` query to delete the row inserted in example C.

```
DELETE OPENQUERY (OracleSvr, 'SELECT name FROM joe.titles WHERE name = ''NewTitle'''');
```

## D. Executing a SELECT pass-through query

The following example uses a pass-through `SELECT` query to select the row inserted in example C.

```
SELECT * FROM OPENQUERY (OracleSvr, 'SELECT name FROM joe.titles WHERE name = ''NewTitle'''');
```

## See Also

[DELETE \(Transact-SQL\)](#)  
[FROM \(Transact-SQL\)](#)  
[INSERT \(Transact-SQL\)](#)  
[OPENDATASOURCE \(Transact-SQL\)](#)  
[OPENROWSET \(Transact-SQL\)](#)  
[Rowset Functions \(Transact-SQL\)](#)  
[SELECT \(Transact-SQL\)](#)  
[sp\\_addlinkedserver \(Transact-SQL\)](#)  
[sp\\_serveroption \(Transact-SQL\)](#)  
[UPDATE \(Transact-SQL\)](#)  
[WHERE \(Transact-SQL\)](#)

# OPENROWSET (Transact-SQL)

7/31/2017 • 17 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✖ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

Includes all connection information that is required to access remote data from an OLE DB data source. This method is an alternative to accessing tables in a linked server and is a one-time, ad hoc method of connecting and accessing remote data by using OLE DB. For more frequent references to OLE DB data sources, use linked servers instead. For more information, see [Linked Servers \(Database Engine\)](#). The `OPENROWSET` function can be referenced in the FROM clause of a query as if it were a table name. The `OPENROWSET` function can also be referenced as the target table of an `INSERT`, `UPDATE`, or `DELETE` statement, subject to the capabilities of the OLE DB provider. Although the query might return multiple result sets, `OPENROWSET` returns only the first one.

`OPENROWSET` also supports bulk operations through a built-in BULK provider that enables data from a file to be read and returned as a rowset.

## [Transact-SQL Syntax Conventions](#)

## Syntax

```
OPENROWSET
( { 'provider_name' , { 'datasource' ; 'user_id' ; 'password'
    | 'provider_string' }
    , { [ catalog. ] [ schema. ] object
        | 'query'
    }
    | BULK 'data_file' ,
        { FORMATFILE = 'format_file_path' [ <bulk_options> ]
        | SINGLE_BLOB | SINGLE_CLOB | SINGLE_NCLOB }
} )  
  
<bulk_options> ::=  
[ , CODEPAGE = { 'ACP' | 'OEM' | 'RAW' | 'code_page' } ]  
[ , DATASOURCE = 'data_source_name' ]  
[ , ERRORFILE = 'file_name' ]  
[ , ERRORFILE_DATASOURCE = 'data_source_name' ]  
[ , FIRSTROW = first_row ]  
[ , LASTROW = last_row ]  
[ , MAXERRORS = maximum_errors ]  
[ , ROWS_PER_BATCH = rows_per_batch ]  
[ , ORDER ( { column [ ASC | DESC ] } [ ,...n ] ) [ UNIQUE ] ]  
  
-- bulk_options related to input file format  
[ , FORMAT = 'CSV' ]  
[ , FIELDQUOTE = 'quote_characters' ]  
[ , FORMATFILE = 'format_file_path' ]
```

## Arguments

### `'provider_name'`

Is a character string that represents the friendly name (or PROGID) of the OLE DB provider as specified in the registry. `provider_name` has no default value.

### `'datasource'`

Is a string constant that corresponds to a particular OLE DB data source. *datasource* is the DBPROP\_INIT\_DATASOURCE property to be passed to the IDBProperties interface of the provider to initialize the provider. Typically, this string includes the name of the database file, the name of a database server, or a name that the provider understands to locate the database or databases.

**'user\_id'**

Is a string constant that is the user name passed to the specified OLE DB provider. *user\_id* specifies the security context for the connection and is passed in as the DBPROP\_AUTH\_USERID property to initialize the provider. *user\_id* cannot be a Microsoft Windows login name.

**'password'**

Is a string constant that is the user password to be passed to the OLE DB provider. *password* is passed in as the DBPROP\_AUTH\_PASSWORD property when initializing the provider. *password* cannot be a Microsoft Windows password.

**'provider\_string'**

Is a provider-specific connection string that is passed in as the DBPROP\_INIT\_PROVIDERSTRING property to initialize the OLE DB provider. *provider\_string* typically encapsulates all the connection information required to initialize the provider. For a list of keywords that are recognized by the SQL Server Native Client OLE DB provider, see [Initialization and Authorization Properties](#).

***catalog***

Is the name of the catalog or database in which the specified object resides.

***schema***

Is the name of the schema or object owner for the specified object.

***object***

Is the object name that uniquely identifies the object to work with.

**'query'**

Is a string constant sent to and executed by the provider. The local instance of SQL Server does not process this query, but processes query results returned by the provider, a pass-through query. Pass-through queries are useful when used on providers that do not make available their tabular data through table names, but only through a command language. Pass-through queries are supported on the remote server, as long as the query provider supports the OLE DB Command object and its mandatory interfaces. For more information, see [SQL Server Native Client \(OLE DB\) Reference](#).

**BULK**

Uses the BULK rowset provider for OPENROWSET to read data from a file. In SQL Server, OPENROWSET can read from a data file without loading the data into a target table. This lets you use OPENROWSET with a simple SELECT statement.

The arguments of the BULK option allow for significant control over where to start and end reading data, how to deal with errors, and how data is interpreted. For example, you can specify that the data file be read as a single-row, single-column rowset of type **varbinary**, **varchar**, or **nvarchar**. The default behavior is described in the argument descriptions that follow.

For information about how to use the BULK option, see "Remarks," later in this topic. For information about the permissions that are required by the BULK option, see "Permissions," later in this topic.

**NOTE**

When used to import data with the full recovery model, OPENROWSET (BULK ...) does not optimize logging.

For information on preparing data for bulk import, see [Prepare Data for Bulk Export or Import \(SQL Server\)](#).

`'data_file'`

Is the full path of the data file whose data is to be copied into the target table.

**Applies to:** SQL Server 2017 CTP 1.1.

Beginning with SQL Server 2017 CTP 1.1, the `data_file` can be in Azure blob storage. For examples, see [Examples of Bulk Access to Data in Azure Blob Storage](#).

`<bulk_options>`

Specifies one or more arguments for the BULK option.

`CODEPAGE = { 'ACP'| 'OEM'| 'RAW'| 'code_page' }`

Specifies the code page of the data in the data file. CODEPAGE is relevant only if the data contains **char**, **varchar**, or **text** columns with character values more than 127 or less than 32.

**NOTE**

We recommend that you specify a collation name for each column in a format file, except when you want the 65001 option to have priority over the collation/code page specification.

CODEPAGE VALUE	DESCRIPTION
ACP	Converts columns of <b>char</b> , <b>varchar</b> , or <b>text</b> data type from the ANSI/ Microsoft Windows code page (ISO 1252) to the SQL Server code page.
OEM (default)	Converts columns of <b>char</b> , <b>varchar</b> , or <b>text</b> data type from the system OEM code page to the SQL Server code page.
RAW	No conversion occurs from one code page to another. This is the fastest option.
<i>code_page</i>	Indicates the source code page on which the character data in the data file is encoded; for example, 850.  ** Important ** Versions prior to SQL Server 2016 do not support code page 65001 (UTF-8 encoding).

`ERRORFILE ='file_name'`

Specifies the file used to collect rows that have formatting errors and cannot be converted to an OLE DB rowset.

These rows are copied into this error file "as is."

The error file is created at the start of the command execution. An error will be raised if the file already exists.

Additionally, a control file that has the extension .ERROR.txt is created. This file references each row in the error file and provides error diagnostics. After the errors have been corrected, the data can be loaded.

**Applies to:** SQL Server 2017 CTP 1.1. Beginning with SQL Server 2017, the `error_file_path` can be in Azure blob storage.

`'errorfile_data_source_name'`

**Applies to:** SQL Server 2017 CTP 1.1. Is a named external data source pointing to the Azure Blob storage location of the error file that will contain errors found during the import. The external data source must be created using the `TYPE = BLOB_STORAGE` option added in SQL Server 2017 CTP 1.1. For more information, see [CREATE EXTERNAL DATA SOURCE](#).

`FIRSTROW =first_row`

Specifies the number of the first row to load. The default is 1. This indicates the first row in the specified data file. The row numbers are determined by counting the row terminators. FIRSTROW is 1-based.

LASTROW =*last\_row*

Specifies the number of the last row to load. The default is 0. This indicates the last row in the specified data file.

MAXERRORS =*maximum\_errors*

Specifies the maximum number of syntax errors or nonconforming rows, as defined in the format file, that can occur before OPENROWSET throws an exception. Until MAXERRORS is reached, OPENROWSET ignores each bad row, not loading it, and counts the bad row as one error.

The default for *maximum\_errors* is 10.

**NOTE**

MAX\_ERRORS does not apply to CHECK constraints, or to converting **money** and **bigint** data types.

ROWS\_PER\_BATCH =*rows\_per\_batch*

Specifies the approximate number of rows of data in the data file. This value should be of the same order as the actual number of rows.

OPENROWSET always imports a data file as a single batch. However, if you specify *rows\_per\_batch* with a value > 0, the query processor uses the value of *rows\_per\_batch* as a hint for allocating resources in the query plan.

By default, ROWS\_PER\_BATCH is unknown. Specifying ROWS\_PER\_BATCH = 0 is the same as omitting ROWS\_PER\_BATCH.

ORDER ( { *column* [ ASC | DESC ] } [ ... *n* ] [ UNIQUE ] )

An optional hint that specifies how the data in the data file is sorted. By default, the bulk operation assumes the data file is unordered. Performance might improve if the order specified can be exploited by the query optimizer to generate a more efficient query plan. Examples for when specifying a sort can be beneficial include the following:

- Inserting rows into a table that has a clustered index, where the rowset data is sorted on the clustered index key.
- Joining the rowset with another table, where the sort and join columns match.
- Aggregating the rowset data by the sort columns.
- Using the rowset as a source table in the FROM clause of a query, where the sort and join columns match.

UNIQUE specifies that the data file does not have duplicate entries.

If the actual rows in the data file are not sorted according to the order that is specified, or if the UNIQUE hint is specified and duplicates keys are present, an error is returned.

Column aliases are required when ORDER is used. The column alias list must reference the derived table that is being accessed by the BULK clause. The column names that are specified in the ORDER clause refer to this column alias list. Large value types (**varchar(max)**, **nvarchar(max)**, **varbinary(max)**, and **xml**) and large object (LOB) types (**text**, **ntext**, and **image**) columns cannot be specified.

SINGLE\_BLOB

Returns the contents of *data\_file* as a single-row, single-column rowset of type **varbinary(max)**.

**IMPORTANT**

We recommend that you import XML data only using the SINGLE\_BLOB option, rather than SINGLE\_CLOB and SINGLE\_NCLOB, because only SINGLE\_BLOB supports all Windows encoding conversions.

SINGLE\_CLOB

By reading *data\_file* as ASCII, returns the contents as a single-row, single-column rowset of type **varchar(max)**, using the collation of the current database.

SINGLE\_NCLOB

By reading *data\_file* as UNICODE, returns the contents as a single-row, single-column rowset of type **nvarchar(max)**, using the collation of the current database.

### Input file format options

FORMAT = 'CSV'

**Applies to:** SQL Server 2017 CTP 1.1.

Specifies a comma separated values file compliant to the [RFC 4180](#) standard.

FORMATFILE ='*format\_file\_path*'

Specifies the full path of a format file. SQL Server supports two types of format files: XML and non-XML.

A format file is required to define column types in the result set. The only exception is when SINGLE\_CLOB, SINGLE\_BLOB, or SINGLE\_NCLOB is specified; in which case, the format file is not required.

For information about format files, see [Use a Format File to Bulk Import Data \(SQL Server\)](#).

**Applies to:** SQL Server 2017 CTP 1.1.

Beginning with SQL Server 2017 CTP 1.1, the *format\_file\_path* can be in Azure blob storage. For examples, see [Examples of Bulk Access to Data in Azure Blob Storage](#).

FIELDQUOTE = 'field\_quote'

**Applies to:** SQL Server 2017 CTP 1.1.

Specifies a character that will be used as the quote character in the CSV file. If not specified, the quote character ("") will be used as the quote character as defined in the [RFC 4180](#) standard.

## Remarks

`OPENROWSET` can be used to access remote data from OLE DB data sources only when the **DisallowAdhocAccess** registry option is explicitly set to 0 for the specified provider, and the Ad Hoc Distributed Queries advanced configuration option is enabled. When these options are not set, the default behavior does not allow for ad hoc access.

When accessing remote OLE DB data sources, the login identity of trusted connections is not automatically delegated from the server on which the client is connected to the server that is being queried. Authentication delegation must be configured.

Catalog and schema names are required if the OLE DB provider supports multiple catalogs and schemas in the specified data source. Values for *catalog* and *schema* can be omitted when the OLE DB provider does not support them. If the provider supports only schema names, a two-part name of the form *schema.object* must be specified. If the provider supports only catalog names, a three-part name of the form *catalog.schema.object* must be specified. Three-part names must be specified for pass-through queries that use the SQL Server Native Client OLE DB provider. For more information, see [Transact-SQL Syntax Conventions \(Transact-SQL\)](#).

`OPENROWSET` does not accept variables for its arguments.

Any call to `OPENDATASOURCE`, `OPENQUERY`, or `OPENROWSET` in the `FROM` clause is evaluated separately and independently from any call to these functions used as the target of the update, even if identical arguments are supplied to the two calls. In particular, filter or join conditions applied on the result of one of those calls have no effect on the results of the other.

## Using OPENROWSET with the BULK Option

The following Transact-SQL enhancements support the `OPENROWSET(BULK...)` function:

- A FROM clause that is used with `SELECT` can call `OPENROWSET(BULK...)` instead of a table name, with full `SELECT` functionality.

`OPENROWSET` with the `BULK` option requires a correlation name, also known as a range variable or alias, in the `FROM` clause. Column aliases can be specified. If a column alias list is not specified, the format file must have column names. Specifying column aliases overrides the column names in the format file, such as:

```
FROM OPENROWSET(BULK...) AS table_alias
```

```
FROM OPENROWSET(BULK...) AS table_alias(column_alias,...n)
```

#### IMPORTANT

Failure to add the `AS <table_alias>` will result in the error:

Msg 491, Level 16, State 1, Line 20

A correlation name must be specified for the bulk rowset in the from clause.

- A `SELECT...FROM OPENROWSET(BULK...)` statement queries the data in a file directly, without importing the data into a table. `SELECT...FROM OPENROWSET(BULK...)` statements can also list bulk-column aliases by using a format file to specify column names, and also data types.
- Using `OPENROWSET(BULK...)` as a source table in an `INSERT` or `MERGE` statement bulk imports data from a data file into a SQL Server table. For more information, see [Import Bulk Data by Using BULK INSERT or OPENROWSET\(BULK...\) \(SQL Server\)](#).
- When the `OPENROWSET BULK` option is used with an `INSERT` statement, the BULK clause supports table hints. In addition to the regular table hints, such as `TABLOCK`, the `BULK` clause can accept the following specialized table hints: `IGNORE_CONSTRAINTS` (ignores only the `CHECK` and `FOREIGN KEY` constraints), `IGNORE_TRIGGERS`, `KEEPDEFAULTS`, and `KEEPIDENTITY`. For more information, see [Table Hints \(Transact-SQL\)](#).

For information about how to use `INSERT...SELECT * FROM OPENROWSET(BULK...)` statements, see [Bulk Import and Export of Data \(SQL Server\)](#). For information about when row-insert operations that are performed by bulk import are logged in the transaction log, see [Prerequisites for Minimal Logging in Bulk Import](#).

#### NOTE

When you use `OPENROWSET`, it is important to understand how SQL Server handles impersonation. For information about security considerations, see [Import Bulk Data by Using BULK INSERT or OPENROWSET\(BULK...\) \(SQL Server\)](#).

### Bulk Importing SQLCHAR, SQLNCHAR or SQLBINARY Data

`OPENROWSET(BULK...)` assumes that, if not specified, the maximum length of `SQLCHAR`, `SQLNCHAR` or `SQLBINARY` data does not exceed 8000 bytes. If the data being imported is in a LOB data field that contains any `varchar(max)`, `nvarchar(max)`, or `varbinary(max)` objects that exceed 8000 bytes, you must use an XML format file that defines the maximum length for the data field. To specify the maximum length, edit the format file and declare the `MAX_LENGTH` attribute.

#### NOTE

An automatically generated format file does not specify the length or maximum length for a LOB field. However, you can edit a format file and specify the length or maximum length manually.

### Bulk Exporting or Importing SQLXML Documents

To bulk export or import `SQLXML` data, use one of the following data types in your format file.

DATA TYPE	EFFECT
SQLCHAR or SQLVARYCHAR	The data is sent in the client code page or in the code page implied by the collation).
SQLNCHAR or SQLNVARCHAR	The data is sent as Unicode.
SQLBINARY or SQLVARYBIN	The data is sent without any conversion.

## Permissions

`OPENROWSET` permissions are determined by the permissions of the user name that is being passed to the OLE DB provider. To use the `BULK` option requires `ADMINISTER BULK OPERATIONS` permission.

## Examples

### A. Using OPENROWSET with SELECT and the SQL Server Native Client OLE DB Provider

The following example uses the SQL Server Native Client OLE DB provider to access the `HumanResources.Department` table in the **AdventureWorks2012** database on the remote server `Seattle1`. (Use `SQLNCLI` and SQL Server will redirect to the latest version of SQL Server Native Client OLE DB Provider.) A `SELECT` statement is used to define the row set returned. The provider string contains the `Server` and `Trusted_Connection` keywords. These keywords are recognized by the SQL Server Native Client OLE DB provider.

```
SELECT a.*  
FROM OPENROWSET('SQLNCLI', 'Server=Seattle1;Trusted_Connection=yes;',  
    'SELECT GroupName, Name, DepartmentID  
        FROM AdventureWorks2012.HumanResources.Department  
    ORDER BY GroupName, Name') AS a;
```

### B. Using the Microsoft OLE DB Provider for Jet

The following example accesses the `Customers` table in the Microsoft Access `Northwind` database through the Microsoft OLE DB Provider for Jet.

#### NOTE

This example assumes that Access is installed. To run this example, you must install the Northwind database.

```
SELECT CustomerID, CompanyName  
FROM OPENROWSET('Microsoft.Jet.OLEDB.4.0',  
    'C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb';  
    'admin';'',Customers);  
GO
```

### C. Using OPENROWSET and another table in an INNER JOIN

The following example selects all data from the `Customers` table from the local instance of SQL Server `Northwind` database and from the `Orders` table from the Access `Northwind` database stored on the same computer.

#### NOTE

This example assumes that Access is installed. To run this example, you must install the Northwind database.

```

USE Northwind ;
GO
SELECT c.*, o.*
FROM Northwind.dbo.Customers AS c
INNER JOIN OPENROWSET('Microsoft.Jet.OLEDB.4.0',
'C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb';'admin''', Orders)
AS o
ON c.CustomerID = o.CustomerID ;
GO

```

#### D. Using OPENROWSET to bulk insert file data into a varbinary(max) column

The following example creates a small table for demonstration purposes, and inserts file data from a file named `Text1.txt` located in the `c:` root directory into a `varbinary(max)` column.

```

USE AdventureWorks2012;
GO
CREATE TABLE myTable(FileName nvarchar(60),
FileType nvarchar(60), Document varbinary(max));
GO

INSERT INTO myTable(FileName, FileType, Document)
SELECT 'Text1.txt' AS FileName,
'.txt' AS FileType,
* FROM OPENROWSET(BULK N'C:\Text1.txt', SINGLE_BLOB) AS Document;
GO

```

#### E. Using the OPENROWSET BULK provider with a format file to retrieve rows from a text file

The following example uses a format file to retrieve rows from a tab-delimited text file, `values.txt` that contains the following data:

```

1 Data Item 1
2 Data Item 2
3 Data Item 3

```

The format file, `values fmt`, describes the columns in `values.txt`:

```

9.0
2
1 SQLCHAR 0 10 "\t" 1 ID SQL_Latin1_General_Cp437_BIN
2 SQLCHAR 0 40 "\r\n" 2 Description SQL_Latin1_General_Cp437_BIN

```

This is the query that retrieves that data:

```

SELECT a.* FROM OPENROWSET( BULK 'c:\test\values.txt',
FORMATFILE = 'c:\test\values fmt') AS a;

```

#### F. Specifying a format file and code page

The following example show how to use both the format file and code page options at the same time.

```

INSERT INTO MyTable SELECT a.* FROM
OPENROWSET (BULK N'D:\data.csv', FORMATFILE =
'D:\format_no_collation.txt', CODEPAGE = '65001') AS a;

```

#### G. Accessing data from a CSV file with a format file

**Applies to:** SQL Server 2017 CTP 1.1.

```
SELECT *
FROM OPENROWSET(BULK N'D:\XChange\test-csv.csv',
    FORMATFILE = N'D:\XChange\test-csv.fmt',
    FIRSTROW=2,
    FORMAT='CSV') AS cars;
```

## H. Accessing data from a CSV file without a format file

```
SELECT * FROM OPENROWSET(
    BULK 'C:\Program Files\Microsoft SQL Server\MSSQL14.CTP1_1\MSSQL\DATA\inv-2017-01-19.csv',
    SINGLE_CLOB) AS DATA;
```

## I. Accessing data from a file stored on Azure Blob storage

**Applies to:** SQL Server 2017 CTP 1.1.

The following example uses an external data source that points to a container in an Azure storage account and a database scoped credential created for a shared access signature.

```
SELECT * FROM OPENROWSET(
    BULK 'inv-2017-01-19.csv',
    DATA_SOURCE = 'MyAzureInvoices',
    SINGLE_CLOB) AS DataFile;
```

For complete `OPENROWSET` examples including configuring the credential and external data source, see [Examples of Bulk Access to Data in Azure Blob Storage](#).

## Additional Examples

For additional examples that show using `INSERT...SELECT * FROM OPENROWSET(BULK...)`, see the following topics:

- [Examples of Bulk Import and Export of XML Documents \(SQL Server\)](#)
- [Keep Identity Values When Bulk Importing Data \(SQL Server\)](#)
- [Keep Nulls or Use Default Values During Bulk Import \(SQL Server\)](#)
- [Use a Format File to Bulk Import Data \(SQL Server\)](#)
- [Use Character Format to Import or Export Data \(SQL Server\)](#)
- [Use a Format File to Skip a Table Column \(SQL Server\)](#)
- [Use a Format File to Skip a Data Field \(SQL Server\)](#)
- [Use a Format File to Map Table Columns to Data-File Fields \(SQL Server\)](#)

## See Also

[DELETE \(Transact-SQL\)](#)

[FROM \(Transact-SQL\)](#)

[Bulk Import and Export of Data \(SQL Server\)](#)

[INSERT \(Transact-SQL\)](#)

[OPENDATASOURCE \(Transact-SQL\)](#)

[OPENQUERY \(Transact-SQL\)](#)

[Rowset Functions \(Transact-SQL\)](#)

[SELECT \(Transact-SQL\)](#)

[sp\\_addlinkedserver \(Transact-SQL\)](#)

[sp\\_serveroption \(Transact-SQL\)](#)

[UPDATE \(Transact-SQL\)](#)

[WHERE \(Transact-SQL\)](#)

# OPENXML (Transact-SQL)

3/24/2017 • 7 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

OPENXML provides a rowset view over an XML document. Because OPENXML is a rowset provider, OPENXML can be used in Transact-SQL statements in which rowset providers such as a table, view, or the OPENROWSET function can appear.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
OPENXML( idoc int [ in ] , rowpattern nvarchar [ in ] , [ flags byte [ in ] ] )
[ WITH ( SchemaDeclaration | TableName ) ]
```

## Arguments

### *idoc*

Is the document handle of the internal representation of an XML document. The internal representation of an XML document is created by calling **sp\_xml\_preparedocument**.

### *rowpattern*

Is the XPath pattern used to identify the nodes (in the XML document whose handle is passed in the *idoc* parameter) to be processed as rows.

### *flags*

Indicates the mapping that should be used between the XML data and the relational rowset, and how the spill-over column should be filled. *flags* is an optional input parameter, and can be one of the following values.

BYTE VALUE	DESCRIPTION
0	Defaults to <b>attribute-centric</b> mapping.
1	Use the <b>attribute-centric</b> mapping. Can be combined with XML_ELEMENTS. In this case, <b>attribute-centric</b> mapping is applied first, and then <b>element-centric</b> mapping is applied for all columns that are not yet dealt with.
2	Use the <b>element-centric</b> mapping. Can be combined with XML_ATTRIBUTES. In this case, <b>attribute-centric</b> mapping is applied first, and then <b>element-centric</b> mapping is applied for all columns not yet dealt with.
8	Can be combined (logical OR) with XML_ATTRIBUTES or XML_ELEMENTS. In the context of retrieval, this flag indicates that the consumed data should not be copied to the overflow property @mp:xmltext.

*SchemaDeclaration*

Is the schema definition of the form: *ColName**ColType* [*ColPattern* | *MetaProperty*] [,*ColName**ColType* [*ColPattern* | *MetaProperty*]...]

#### *ColName*

Is the column name in the rowset.

#### *ColType*

Is the SQL Server data type of the column in the rowset. If the column types differ from the underlying **xml** data type of the attribute, type coercion occurs.

#### *ColPattern*

Is an optional, general XPath pattern that describes how the XML nodes should be mapped to the columns. If *ColPattern* is not specified, the default mapping (**attribute-centric** or **element-centric** mapping as specified by *flags*) takes place.

The XPath pattern specified as *ColPattern* is used to specify the special nature of the mapping (in the case of **attribute-centric** and **element-centric** mapping) that overwrites or enhances the default mapping indicated by *flags*.

The general XPath pattern specified as *ColPattern* also supports the metaproPERTIES.

#### *MetaProperty*

Is one of the metaproPERTIES provided by OPENXML. If *MetaProperty* is specified, the column contains information provided by the metaproPERTY. The metaproPERTIES allow you to extract information (such as relative position and namespace information) about XML nodes. This provides more information than is visible in the textual representation.

#### *TableName*

Is the table name that can be given (instead of *SchemaDeclaration*) if a table with the desired schema already exists and no column patterns are required.

## Remarks

The WITH clause provides a rowset format (and additional mapping information as required) by using either *SchemaDeclaration* or specifying an existing *TableName*. If the optional WITH clause is not specified, the results are returned in an **edge** table format. Edge tables represent the fine-grained XML document structure (such as element/attribute names, the document hierarchy, the namespaces, PIs, and so on) in a single table.

The following table describes the structure of the **edge** table.

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>id</b>	<b>bigint</b>	<p>Is the unique ID of the document node.</p> <p>The root element has an ID value 0. The negative ID values are reserved.</p>
<b>parentid</b>	<b>bigint</b>	<p>Identifies the parent of the node. The parent identified by this ID is not necessarily the parent element, but it depends on the NodeType of the node whose parent is identified by this ID. For example, if the node is a text node, the parent of it may be an attribute node.</p> <p>If the node is at the top level in the XML document, its <b>ParentID</b> is NULL.</p>

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>nodetype</b>	<b>int</b>	<p>Identifies the node type. Is an integer that corresponds to the XML DOM node type numbering.</p> <p>The node types are:</p> <ul style="list-style-type: none"> <li>1 = Element node</li> <li>2 = Attribute node</li> <li>3 = Text node</li> </ul>
<b>localname</b>	<b>nvarchar</b>	Gives the local name of the element or attribute. Is NULL if the DOM object does not have a name.
<b>prefix</b>	<b>nvarchar</b>	Is the namespace prefix of the node name.
<b>namespaceuri</b>	<b>nvarchar</b>	Is the namespace URI of the node. If the value is NULL, no namespace is present.
<b>datatype</b>	<b>nvarchar</b>	Is the actual data type of the element or attribute row, otherwise is NULL. The data type is inferred from the inline DTD or from the inline schema.
<b>prev</b>	<b>bigint</b>	Is the XML ID of the previous sibling element. Is NULL if there is no direct previous sibling.
<b>text</b>	<b>ntext</b>	Contains the attribute value or the element content in text form (or is NULL if the <b>edge</b> table entry does not require a value).

## Examples

### A. Using a simple SELECT statement with OPENXML

The following example creates an internal representation of the XML image by using `sp_xml_preparedocument`. A `SELECT` statement that uses an `OPENXML` rowset provider is then executed against the internal representation of the XML document.

The *flag* value is set to `1`. This indicates **attribute-centric** mapping. Therefore, the XML attributes map to the columns in the rowset. The *rowpattern* specified as `/ROOT/Customer` identifies the `<Customers>` nodes to be processed.

The optional *ColPattern* (column pattern) parameter is not specified because the column name matches the XML attribute names.

The `OPENXML` rowset provider creates a two-column rowset (`CustomerID` and `ContactName`) from which the `SELECT` statement retrieves the necessary columns (in this case, all the columns).

```

DECLARE @idoc int, @doc varchar(1000);
SET @doc =''
<ROOT>
<Customer CustomerID="VINET" ContactName="Paul Henriot">
    <Order CustomerID="VINET" EmployeeID="5" OrderDate="1996-07-04T00:00:00">
        <OrderDetail OrderID="10248" ProductID="11" Quantity="12"/>
        <OrderDetail OrderID="10248" ProductID="42" Quantity="10"/>
    </Order>
</Customer>
<Customer CustomerID="LILAS" ContactName="Carlos Gonzlez">
    <Order CustomerID="LILAS" EmployeeID="3" OrderDate="1996-08-16T00:00:00">
        <OrderDetail OrderID="10283" ProductID="72" Quantity="3"/>
    </Order>
</Customer>
</ROOT>';
--Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc;
-- Execute a SELECT statement that uses the OPENXML rowset provider.
SELECT      *
FROM        OPENXML (@idoc, '/ROOT/Customer',1)
            WITH (CustomerID  varchar(10),
                  ContactName varchar(20));

```

Here is the result set.

CustomerID	ContactName
VINET	Paul Henriot
LILAS	Carlos Gonzlez

If the same `SELECT` statement is executed with `flags` set to `2`, indicating **element-centric** mapping, the values of `CustomerID` and `ContactName` for both of the customers in the XML document are returned as `NULL`, because there are not any elements named `CustomerID` or `ContactName` in the XML document.

Here is the result set.

CustomerID	ContactName
NULL	NULL
NULL	NULL

## B. Specifying ColPattern for mapping between columns and the XML attributes

The following query returns customer ID, order date, product ID and quantity attributes from the XML document.

The `rowpattern` identifies the `<OrderDetails>` elements. `ProductID` and `Quantity` are the attributes of the `<OrderDetails>` element. However, `OrderID`, `CustomerID`, and `OrderDate` are the attributes of the parent element (`<Orders>`).

The optional `ColPattern` is specified. This indicates the following:

- The `OrderID`, `CustomerID`, and `OrderDate` in the rowset map to the attributes of the parent of the nodes identified by `rowpattern` in the XML document.
- The `ProdID` column in the rowset maps to the `ProductID` attribute, and the `Qty` column in the rowset maps to the `Quantity` attribute of the nodes identified in `rowpattern`.

Although the **element-centric** mapping is specified by the `flags` parameter, the mapping specified in `ColPattern` overwrites this mapping.

```

DECLARE @idoc int, @doc varchar(1000);
SET @doc ='<ROOT>
<Customer CustomerID="VINET" ContactName="Paul Henriot">
    <Order OrderID="10248" CustomerID="VINET" EmployeeID="5"
        OrderDate="1996-07-04T00:00:00">
        <OrderDetail ProductID="11" Quantity="12"/>
        <OrderDetail ProductID="42" Quantity="10"/>
    </Order>
</Customer>
<Customer CustomerID="LILAS" ContactName="Carlos Gonzlez">v
    <Order OrderID="10283" CustomerID="LILAS" EmployeeID="3"
        OrderDate="1996-08-16T00:00:00">
        <OrderDetail ProductID="72" Quantity="3"/>
    </Order>
</Customer>
</ROOT>';

--Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc;

-- SELECT stmt using OPENXML rowset provider
SELECT *
FROM  OPENXML (@idoc, '/ROOT/Customer/Order/OrderDetail',2)
      WITH (OrderID      int      '../@OrderID',
            CustomerID  varchar(10)  '../@CustomerID',
            OrderDate   datetime   '../@OrderDate',
            ProdID      int      '@ProductID',
            Qty         int      '@Quantity');

```

Here is the result set.

OrderID	CustomerID	OrderDate	ProdID	Qty
10248	VINET	1996-07-04 00:00:00.000	11	12
10248	VINET	1996-07-04 00:00:00.000	42	10
10283	LILAS	1996-08-16 00:00:00.000	72	3

### C. Obtaining results in an edge table format

The sample XML document in the following example consists of `<Customers>`, `<Orders>`, and `<Order_0020_Details>` elements. First, `sp_xml_preparedocument` is called to obtain a document handle. This document handle is passed to `OPENXML`.

In the `OPENXML` statement, the `rowpattern` (`/ROOT/Customers`) identifies the `<Customers>` nodes to process. Because the `WITH` clause is not provided, `OPENXML` returns the rowset in an **edge** table format.

Finally the `SELECT` statement retrieves all the columns in the **edge** table.

```

DECLARE @idoc int, @doc varchar(1000);
SET @doc =''
<ROOT>
<Customers CustomerID="VINET" ContactName="Paul Henriot">
    <Orders CustomerID="VINET" EmployeeID="5" OrderDate=
        "1996-07-04T00:00:00">
        <Order_x0020_Details OrderID="10248" ProductID="11" Quantity="12"/>
        <Order_x0020_Details OrderID="10248" ProductID="42" Quantity="10"/>
    </Orders>
</Customers>
<Customers CustomerID="LILAS" ContactName="Carlos Gonzlez">
    <Orders CustomerID="LILAS" EmployeeID="3" OrderDate=
        "1996-08-16T00:00:00">
        <Order_x0020_Details OrderID="10283" ProductID="72" Quantity="3"/>
    </Orders>
</Customers>
</ROOT>';

--Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc;

-- SELECT statement that uses the OPENXML rowset provider.
SELECT      *
FROM        OPENXML (@idoc, '/ROOT/Customers')
EXEC sp_xml_removedocument @idoc;

```

## See Also

[Examples: Using OPENXML](#)

# Security Functions (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following functions return information that is useful in managing security. Additional functions are listed under [Cryptographic Functions \(Transact-SQL\)](#).

<a href="#">CERTENCODED (Transact-SQL)</a>	<a href="#">PWDCOMPARE (Transact-SQL)</a>
<a href="#">CERTPRIVATEKEY (Transact-SQL)</a>	<a href="#">PWDENCRYPT (Transact-SQL)</a>
<a href="#">CURRENT_USER (Transact-SQL)</a>	<a href="#">SCHEMA_ID (Transact-SQL)</a>
<a href="#">DATABASE_PRINCIPAL_ID (Transact-SQL)</a>	<a href="#">SCHEMA_NAME (Transact-SQL)</a>
<a href="#">sys.fn_builtin_permissions (Transact-SQL)</a>	<a href="#">SESSION_USER (Transact-SQL)</a>
<a href="#">sys.fn_get_audit_file (Transact-SQL)</a>	<a href="#">SUSER_ID (Transact-SQL)</a>
<a href="#">sys.fn_my_permissions (Transact-SQL)</a>	<a href="#">SUSER_SID (Transact-SQL)</a>
<a href="#">HAS_PERMS_BY_NAME (Transact-SQL)</a>	<a href="#">SUSER_SNAME (Transact-SQL)</a>
<a href="#">IS_MEMBER (Transact-SQL)</a>	<a href="#">SYSTEM_USER (Transact-SQL)</a>
<a href="#">IS_ROLEMEMBER (Transact-SQL)</a>	<a href="#">SUSER_NAME (Transact-SQL)</a>
<a href="#">IS_SRVROLEMEMBER (Transact-SQL)</a>	<a href="#">USER_ID (Transact-SQL)</a>
<a href="#">ORIGINAL_LOGIN (Transact-SQL)</a>	<a href="#">USER_NAME (Transact-SQL)</a>
<a href="#">PERMISSIONS (Transact-SQL)</a>	

For information about membership in Windows groups, see [xp\\_logininfo \(Transact-SQL\)](#) and [xp\\_enumgroups \(Transact-SQL\)](#).

## See Also

[Security Stored Procedures \(Transact-SQL\)](#)

[Cryptographic Functions \(Transact-SQL\)](#)

[Built-in Functions \(Transact-SQL\)](#)

[Security Statements](#)

# CERTENCODED (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2012) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Returns the public portion of a certificate in binary format. This function takes a certificate ID and returns the encoded certificate. The binary result can be passed to **CREATE CERTIFICATE ... WITH BINARY** to create a new certificate.

## Syntax

```
CERTENCODED ( cert_id )
```

## Arguments

*cert\_id*

Is the **certificate\_id** of the certificate. This is available from sys.certificates or by using the [CERT\\_ID \(Transact-SQL\)](#) function. *cert\_id* is type **int**

## Return types

**varbinary**

## Remarks

**CERTENCODED** and **CERTPRIVATEKEY** are used together to return different portions of a certificate in binary form.

## Permissions

**CERTENCODED** is available to public.

## Examples

### Simple Example

The following example creates a certificate named `Shipping04` and then uses the **CERTENCODED** function to return the binary encoding of the certificate.

```
CREATE DATABASE TEST1;
GO
USE TEST1
CREATE CERTIFICATE Shipping04
ENCRYPTION BY PASSWORD = 'pGFD4bb925DGvbd2439587y'
WITH SUBJECT = 'Sammamish Shipping Records',
EXPIRY_DATE = '20161031';
GO
SELECT CERTENCODED(CERT_ID('Shipping04'));
```

### B. Copying a Certificate to Another Database

The following more complicated example, creates two databases, `SOURCE_DB` and `TARGET_DB`. The goal is to create a certificate in the `SOURCE_DB`, and then copy the certificate to the `TARGET_DB`, and then demonstrate that data encrypted in `SOURCE_DB` can be decrypted in `TARGET_DB` using the copy of the certificate.

To create the example environment, create the `SOURCE_DB` and `TARGET_DB` databases, and a master key in each. Then create a certificate in `SOURCE_DB`.

```
USE master;
GO
CREATE DATABASE SOURCE_DB;
GO
USE SOURCE_DB;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'SOURCE_DB KEY Pa$$W0rd';
GO
CREATE DATABASE TARGET_DB;
GO
USE TARGET_DB
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Pa$$W0rd in TARGET_DB';
GO

-- Create a certificate in SOURCE_DB
USE SOURCE_DB;
GO
CREATE CERTIFICATE SOURCE_CERT WITH SUBJECT = 'SOURCE_CERTIFICATE';
GO
```

Now extract the binary description of the certificate.

```
DECLARE @CERTENC VARBINARY(MAX);
DECLARE @CERTPVK VARBINARY(MAX);
SELECT @CERTENC = CERTENCODED(CERT_ID('SOURCE_CERT'));
SELECT @CERTPVK = CERTPRIVATEKEY(CERT_ID('SOURCE_CERT'),
    'CertEncryptionPa$$word');
SELECT @CERTENC AS BinaryCertificate;
SELECT @CERTPVK AS EncryptedBinaryCertificate;
GO
```

Create the duplicate certificate in the `TARGET_DB` database. You must modify the following code, inserting the two binary values returned in the previous step.

```
-- Create the duplicate certificate in the TARGET_DB database
USE TARGET_DB
GO
CREATE CERTIFICATE TARGET_CERT
FROM BINARY = <insert the binary value of the @CERTENC variable>
WITH PRIVATE KEY (
    BINARY = <insert the binary value of the @CERTPVK variable>
    , DECRYPTION BY PASSWORD = 'CertEncryptionPa$$word');
-- Compare the certificates in the two databases
-- The two certificates should be the same
-- except for name and (possibly) the certificate_id
SELECT * FROM SOURCE_DB.sys.certificates
UNION
SELECT * FROM TARGET_DB.sys.certificates;
```

The following code executed as a single batch demonstrates that data encrypted in `SOURCE_DB` can be decrypted in `TARGET_DB`.

```
USE SOURCE_DB;

DECLARE @CLEARTEXT nvarchar(100);
DECLARE @CIPHERTEXT varbinary(8000);
DECLARE @UNCIPHEREDTEXT_Source nvarchar(100);
SET @CLEARTEXT = N'Hello World';
SET @CIPHERTEXT = ENCRYPTBYCERT(CERT_ID('SOURCE_CERT'), @CLEARTEXT);
SET @UNCIPHEREDTEXT_Source =
    DECRYPTBYCERT(CERT_ID('SOURCE_CERT'), @CIPHERTEXT)
-- Encryption and decryption result in SOURCE_DB
SELECT @CLEARTEXT AS SourceClearText, @CIPHERTEXT AS SourceCipherText,
    @UNCIPHEREDTEXT_Source AS SourceDecryptedText;

-- SWITCH DATABASE
USE TARGET_DB;

DECLARE @UNCIPHEREDTEXT_Target nvarchar(100);
SET @UNCIPHEREDTEXT_Target = DECRYPTBYCERT(CERT_ID('TARGET_CERT'), @CIPHERTEXT);
-- Encryption and decryption result in TARGET_DB
SELECT @CLEARTEXT AS ClearTextInTarget, @CIPHERTEXT AS CipherTextInTarget, @UNCIPHEREDTEXT_Target AS
DecryptedTextInTarget;
GO
```

## See also

[Security Functions \(Transact-SQL\)](#)  
[CREATE CERTIFICATE \(Transact-SQL\)](#)  
[CERTPRIVATEKEY \(Transact-SQL\)](#)  
[sys.certificates \(Transact-SQL\)](#)

# CERTPRIVATEKEY (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2012) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Returns the private key of a certificate in binary format. This function takes three arguments.

- A certificate ID.
- An encryption password which is used to encrypt the private key bits when they are returned by the function, so that the keys are not exposed clear text to users.
- A decryption password which is optional. If a decryption password is specified, then it is used to decrypt the private key of the certificate otherwise database master key is used.

Only users that have access to certificate's private key will be able to use this function. This function returns the private key in PVK format.

## Syntax

```
CERTPRIVATEKEY
(
    cert_ID
    , 'encryption_password'
    [ , 'decryption_password' ]
)
```

## Arguments

*certificate\_ID*

Is the **certificate\_id** of the certificate. This is available from sys.certificates or by using the [CERT\\_ID \(Transact-SQL\)](#) function. *cert\_id* is type **int**

*encryption\_password*

The password used to encrypt the returned binary value.

*decryption\_password*

The password used to decrypt the returned binary value.

## Return types

**varbinary**

## Remarks

**CERTENCODED** and **CERTPRIVATEKEY** are used together to return different portions of a certificate in binary form.

## Permissions

**CERTPRIVATEKEY** is available to public.

## Examples

```
CREATE DATABASE TEST1;
GO
USE TEST1
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Use 5tr0ng P^55Words'
GO
CREATE CERTIFICATE Shipping04
WITH SUBJECT = 'Sammamish Shipping Records',
EXPIRY_DATE = '20141031';
GO
SELECT CERTPRIVATEKEY(CERT_ID('Shipping04'), 'jklalkaa/; uia3dd');
```

For a more complex example that uses **CERTPRIVATEKEY** and **CERTENCODED** to copy a certificate to another database, see example B in the topic [CERTENCODED \(Transact-SQL\)](#).

## See also

[Security Functions \(Transact-SQL\)](#)

[CREATE CERTIFICATE \(Transact-SQL\)](#) [Security Functions \(Transact-SQL\)](#) [sys.certificates \(Transact-SQL\)](#)

# CURRENT\_USER (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the name of the current user. This function is equivalent to `USER_NAME()`.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
CURRENT_USER
```

## Return types

**sysname**

## Remarks

`CURRENT_USER` returns the name of the current security context. If `CURRENT_USER` is executed after a call to `EXECUTE AS` switches context, `CURRENT_USER` will return the name of the impersonated context. If a Windows principal accessed the database by way of membership in a group, the name of the Windows principal will be returned instead of the name of the group.

To return the login of the current user, see [`SUSER\_NAME \(Transact-SQL\)`](#) and [`SYSTEM\_USER \(Transact-SQL\)`](#).

## Examples

### A. Using `CURRENT_USER` to return the current user name

The following example returns the name of the current user.

```
SELECT CURRENT_USER;
GO
```

### B. Using `CURRENT_USER` as a `DEFAULT` constraint

The following example creates a table that uses `CURRENT_USER` as a `DEFAULT` constraint for the `order_person` column on a sales row.

```

USE AdventureWorks2012;
GO
IF EXISTS (SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_NAME = 'orders22')
    DROP TABLE orders22;
GO
SET NOCOUNT ON;
CREATE TABLE orders22
(
order_id int IDENTITY(1000, 1) NOT NULL,
cust_id int NOT NULL,
order_date smalldatetime NOT NULL DEFAULT GETDATE(),
order_amt money NOT NULL,
order_person char(30) NOT NULL DEFAULT CURRENT_USER
);
GO

```

The following code inserts a record in the table. The user that is executing these statements is named `Wanida`.

```

INSERT orders22 (cust_id, order_amt)
VALUES (5105, 577.95);
GO
SET NOCOUNT OFF;
GO

```

The following query selects all information from the `orders22` table.

```

SELECT * FROM orders22;
GO

```

Here is the result set.

order_id	cust_id	order_date	order_amt	order_person
----------	---------	------------	-----------	--------------

1000	5105	2005-04-03 23:34:00	577.95	Wanida
------	------	---------------------	--------	--------

(1 row(s) affected)
---------------------

### C. Using `CURRENT_USER` from an impersonated context

In the following example, user `Wanida` executes the following Transact-SQL code.

```

SELECT CURRENT_USER;
GO
EXECUTE AS USER = 'Arnalfo';
GO
SELECT CURRENT_USER;
GO
REVERT;
GO
SELECT CURRENT_USER;
GO

```

Here is the result set.

Wanida
--------

Arnalfo
---------

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D: Using CURRENT\_USER to return the current user name

The following example returns the name of the current user.

```
SELECT CURRENT_USER;
```

## See also

[USER\\_NAME \(Transact-SQL\)](#)

[SYSTEM\\_USER \(Transact-SQL\)](#)

[sys.database\\_principals \(Transact-SQL\)](#)

[ALTER TABLE \(Transact-SQL\)](#)

[CREATE TABLE \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

# HAS\_DBACCESS (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns information about whether the user has access to the specified database.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server, Azure SQL Data Warehouse, Parallel Data Warehouse

HAS_DBACCESS ( 'database_name' )
```

## Arguments

'*database\_name*'

The name of the database for which the user wants access information. *database\_name* is **sysname**.

## Return Types

**int**

## Remarks

HAS\_DBACCESS returns 1 if the user has access to the database, 0 if the user has no access to the database, and NULL if the database name is not valid.

HAS\_DBACCESS returns 0 if the database is offline or suspect.

HAS\_DBACCESS returns 0 if the database is in single-user mode and the database is in use by another user.

## Permissions

Requires membership in the public role.

## Examples

The following example tests whether current user has access to the `AdventureWorks2012` database.

```
SELECT HAS_DBACCESS('AdventureWorks2012');
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example tests whether current user has access to the `AdventureWorksPDW2012` database.

```
SELECT HAS_DBACCESS('AdventureWorksPDW2012');
GO
```

## See Also

[IS\\_MEMBER \(Transact-SQL\)](#)

[IS\\_SRVROLEMEMBER \(Transact-SQL\)](#)

# HAS\_PERMS\_BY\_NAME (Transact-SQL)

7/31/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Evaluates the effective permission of the current user on a securable. A related function is [fn\\_my\\_permissions](#).

[Transact-SQL Syntax Conventions](#)

## Syntax

```
HAS_PERMS_BY_NAME ( securable , securable_class , permission
[ , sub-securable ] [ , sub-securable_class ] )
```

## Arguments

### *securable*

Is the name of the securable. If the securable is the server itself, this value should be set to NULL. *securable* is a scalar expression of type **sysname**. There is no default.

### *securable\_class*

Is the name of the class of securable against which the permission is tested. *securable\_class* is a scalar expression of type **nvarchar(60)**.

In Azure SQL Database, the *securable\_class* argument must be set to one of the following: **DATABASE, OBJECT, ROLE, SCHEMA, or USER**.

### *permission*

A nonnull scalar expression of type **sysname** that represents the permission name to be checked. There is no default. The permission name ANY is a wildcard.

### *sub-securable*

An optional scalar expression of type **sysname** that represents the name of the securable sub-entity against which the permission is tested. The default is NULL.

### NOTE

In versions of SQL Server through SQL Server 2017, sub-securables cannot use brackets in the form '[*sub name*]'. Use '*sub name*' instead.

### *sub-securable\_class*

An optional scalar expression of type **nvarchar(60)** that represent the class of securable subentity against which the permission is tested. The default is NULL.

In Azure SQL Database, the *sub-securable\_class* argument is valid only if the *securable\_class* argument is set to **OBJECT**. If the *securable\_class* argument is set to **OBJECT**, the *sub-securable\_class* argument must be set to **COLUMN**.

## Return Types

## int

Returns NULL when the query fails.

## Remarks

This built-in function tests whether the current principal has a particular effective permission on a specified securable. HAS\_PERMS\_BY\_NAME returns 1 when the user has effective permission on the securable, 0 when the user has no effective permission on the securable, and NULL when the securable class or permission is not valid. An effective permission is any of the following:

- A permission granted directly to the principal, and not denied.
- A permission implied by a higher-level permission held by the principal and not denied.
- A permission granted to a role or group of which the principal is a member, and not denied.
- A permission held by a role or group of which the principal is a member, and not denied.

The permission evaluation is always performed in the security context of the caller. To determine whether some other user has an effective permission, the caller must have IMPERSONATE permission on that user.

For schema-level entities, one-, two-, or three-part nonnull names are accepted. For database-level entities a one-part name is accepted, with a null value meaning "current database". For the server itself, a null value (meaning "current server") is required. This function cannot check permissions on a linked server or on a Windows user for which no server-level principal has been created.

The following query will return a list of built-in securable classes:

```
SELECT class_desc FROM sys.fn_builtin_permissions(default);
```

The following collations are used:

- Current database collation: Database-level securables that include securables not contained by a schema; one- or two-part schema-scoped securables; target database when using a three-part name.
- master database collation: Server-level securables.
- 'ANY' is not supported for column-level checks. You must specify the appropriate permission.

## Examples

### A. Do I have the server-level VIEW SERVER STATE permission?

**Applies to:** SQL Server 2008 through SQL Server 2017

```
SELECT HAS_PERMS_BY_NAME(null, null, 'VIEW SERVER STATE');
```

### B. Am I able to IMPERSONATE server principal Ps?

**Applies to:** SQL Server 2008 through SQL Server 2017

```
SELECT HAS_PERMS_BY_NAME('Ps', 'LOGIN', 'IMPERSONATE');
```

### C. Do I have any permissions in the current database?

```
SELECT HAS_PERMS_BY_NAME(db_name(), 'DATABASE', 'ANY');
```

#### D. Does database principal Pd have any permission in the current database?

Assume caller has IMPERSONATE permission on principal `Pd`.

```
EXECUTE AS user = 'Pd'  
GO  
SELECT HAS_PERMS_BY_NAME(db_name(), 'DATABASE', 'ANY');  
GO  
REVERT;  
GO
```

#### E. Can I create procedures and tables in schema S?

The following example requires `ALTER` permission in `S` and `CREATE PROCEDURE` permission in the database, and similarly for tables.

```
SELECT HAS_PERMS_BY_NAME(db_name(), 'DATABASE', 'CREATE PROCEDURE')  
& HAS_PERMS_BY_NAME('S', 'SCHEMA', 'ALTER') AS _can_create_procs,  
HAS_PERMS_BY_NAME(db_name(), 'DATABASE', 'CREATE TABLE') &  
HAS_PERMS_BY_NAME('S', 'SCHEMA', 'ALTER') AS _can_create_tables;
```

#### F. Which tables do I have SELECT permission on?

```
SELECT HAS_PERMS_BY_NAME  
(QUOTENAME(SCHEMA_NAME(schema_id)) + '.' + QUOTENAME(name),  
'OBJECT', 'SELECT') AS have_select, * FROM sys.tables
```

#### G. Do I have INSERT permission on the SalesPerson table in AdventureWorks2012?

The following example assumes `AdventureWorks2012` is my current database context, and uses a two-part name.

```
SELECT HAS_PERMS_BY_NAME('Sales.SalesPerson', 'OBJECT', 'INSERT');
```

The following example makes no assumptions about my current database context, and uses a three-part name.

```
SELECT HAS_PERMS_BY_NAME('AdventureWorks2012.Sales.SalesPerson',  
'OBJECT', 'INSERT');
```

#### H. Which columns of table T do I have SELECT permission on?

```
SELECT name AS column_name,  
HAS_PERMS_BY_NAME('T', 'OBJECT', 'SELECT', name, 'COLUMN')  
AS can_select  
FROM sys.columns AS c  
WHERE c.object_id=object_id('T');
```

## See Also

[Permissions \(Database Engine\)](#)

[Securables](#)

[Permissions Hierarchy \(Database Engine\)](#)

[sys.fn\\_builtin\\_permissions \(Transact-SQL\)](#)

## Security Catalog Views (Transact-SQL)

# IS\_MEMBER (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Indicates whether the current user is a member of the specified Microsoft Windows group or SQL Server database role.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
IS_MEMBER ( { 'group' | 'role' } )
```

## Arguments

`'group'`

**Applies to:** SQL Server 2008 through SQL Server 2017

Is the name of the Windows group that is being checked; must be in the format *Domain\Group*. *group* is **sysname**.

`'role'`

Is the name of the SQL Server role that is being checked. *role* is **sysname** and can include the database fixed roles or user-defined roles, but not server roles.

## Return Types

**int**

## Remarks

IS\_MEMBER returns the following values.

RETURN VALUE	DESCRIPTION
0	Current user is not a member of <i>group</i> or <i>role</i> .
1	Current user is a member of <i>group</i> or <i>role</i> .
NULL	Either <i>group</i> or <i>role</i> is not valid. When queried by a SQL Server login or a login using an application role, returns NULL for a Windows group.

IS\_MEMBER determines Windows group membership by examining an access token that is created by Windows. The access token does not reflect changes in group membership that are made after a user connects to an instance of SQL Server. Windows group membership cannot be queried by a SQL Server login or a SQL Server application role.

To add and remove members from a database role, use [ALTER ROLE \(Transact-SQL\)](#). To add and remove members from a server role, use [ALTER SERVER ROLE \(Transact-SQL\)](#).

This function evaluates role membership, not the underlying permission. For example, the **db\_owner** fixed database role has the **CONTROL DATABASE** permission. If the user has the **CONTROL DATABASE** permission but is not a member of the role, this function will correctly report that the user is not a member of the **db\_owner** role, even though the user has the same permissions.

Members of the **sysadmin** fixed server role enter every database as the **dbo** user. Checking permission for member of the **sysadmin** fixed server role, checks permissions for **dbo**, not the original login. Since **dbo** can't be added to a database role and doesn't exist in Windows groups, **dbo** will always return 0 (or NULL if the role doesn't exist).

## Related Functions

To determine whether another SQL Server login is a member of a database role, use [IS\\_ROLEMEMBER \(Transact-SQL\)](#). To determine whether a SQL Server login is a member of a server role, use [IS\\_SRVROLEMEMBER \(Transact-SQL\)](#).

## Examples

The following example checks whether the current user is a member of a database role or a Windows domain group.

```
-- Test membership in db_owner and print appropriate message.  
IF IS_MEMBER ('db_owner') = 1  
    PRINT 'Current user is a member of the db_owner role'  
ELSE IF IS_MEMBER ('db_owner') = 0  
    PRINT 'Current user is NOT a member of the db_owner role'  
ELSE IF IS_MEMBER ('db_owner') IS NULL  
    PRINT 'ERROR: Invalid group / role specified';  
GO  
  
-- Execute SELECT if user is a member of ADVWORKS\Shipping.  
IF IS_MEMBER ('ADVWORKS\Shipping') = 1  
    SELECT 'User ' + USER + ' is a member of ADVWORKS\Shipping.';  
GO
```

## See Also

[IS\\_SRVROLEMEMBER \(Transact-SQL\)](#)

[Principals \(Database Engine\)](#)

[Security Catalog Views \(Transact-SQL\)](#)

[Security Functions \(Transact-SQL\)](#)

# IS\_ROLEMEMBER (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2012) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Indicates whether a specified database principle is a member of the specified database role.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
IS_ROLEMEMBER ( 'role' [ , 'database_principal' ] )
```

## Arguments

'*role*'

Is the name of the database role that is being checked. *role* is **sysname**.

'*database\_principal*'

Is the name of the database user, database role, or application role to check. *database\_principal* is **sysname**, with a default of NULL. If no value is specified, the result is based on the current execution context. If the parameter contains the word NULL will return NULL.

## Return Types

**int**

RETURN VALUE	DESCRIPTION
0	<i>database_principal</i> is not a member of <i>role</i> .
1	<i>database_principal</i> is a member of <i>role</i> .
NULL	<i>database_principal</i> or <i>role</i> is not valid, or you do not have permission to view the role membership.

## Remarks

Use IS\_ROLEMEMBER to determine whether the current user can perform an action that requires the database role's permissions.

If *database\_principal* is based on a Windows login, such as Contoso\Mary5, IS\_ROLEMEMBER returns NULL, unless the *database\_principal* has been granted or denied direct access to SQL Server.

If the optional *database\_principal* parameter is not provided and if the *database\_principal* is based on a Windows domain login, it may be a member of a database role through membership in a Windows group. To resolve such indirect memberships, IS\_ROLEMEMBER requests Windows group membership information from the domain controller. If the domain controller is inaccessible or does not respond, IS\_ROLEMEMBER returns role membership information by accounting for the user and its local groups only. If the user specified is not the current user, the

value returned by `IS_ROLEMEMBER` might differ from the authenticator's (such as Active Directory) last data update to SQL Server.

If the optional `database_principal` parameter is provided, the database principal that is being queried must be present in `sys.database_principals`, or `IS_ROLEMEMBER` will return `NULL`. This indicates that the `database_principal` is not valid in this database.

When the `database_principal` parameter is based on a domain login or based on a Windows group and the domain controller is inaccessible, calls to `IS_ROLEMEMBER` will fail and might return incorrect or incomplete data.

If the domain controller is not available, the call to `IS_ROLEMEMBER` will return accurate information when the Windows principle can be authenticated locally, such as a local Windows account or a SQL Server login.

**IS\_ROLEMEMBER** always returns 0 when a Windows group is used as the database principal argument, and this Windows group is a member of another Windows group which is, in turn, a member of the specified database role.

The User Account Control (UAC) found in Windows Vista and Windows Server 2008 might also return different results. This would depend on whether the user accessed the server as a Windows group member or as a specific SQL Server user.

This function evaluates role membership, not the underlying permission. For example, the **db\_owner** fixed database role has the **CONTROL DATABASE** permission. If the user has the **CONTROL DATABASE** permission but is not a member of the role, this function will correctly report that the user is not a member of the **db\_owner** role, even though the user has the same permissions.

## Related Functions

To determine whether the current user is a member of the specified Windows group or SQL Server database role, use [IS\\_MEMBER \(Transact-SQL\)](#). To determine whether a SQL Server login is a member of a server role, use [IS\\_SRVROLEMEMBER \(Transact-SQL\)](#).

## Permissions

Requires `VIEW DEFINITION` permission on the database role.

## Examples

The following example indicates whether the current user is a member of the `db_datareader` fixed database role.

```
IF IS_ROLEMEMBER ('db_datareader') = 1
    print 'Current user is a member of the db_datareader role'
ELSE IF IS_ROLEMEMBER ('db_datareader') = 0
    print 'Current user is NOT a member of the db_datareader role'
ELSE IF IS_ROLEMEMBER ('db_datareader') IS NULL
    print 'ERROR: The database role specified is not valid.';
```

## See Also

[CREATE ROLE \(Transact-SQL\)](#)  
[ALTER ROLE \(Transact-SQL\)](#)  
[DROP ROLE \(Transact-SQL\)](#)  
[CREATE SERVER ROLE \(Transact-SQL\)](#)  
[ALTER SERVER ROLE \(Transact-SQL\)](#)  
[DROP SERVER ROLE \(Transact-SQL\)](#)  
[IS\\_MEMBER \(Transact-SQL\)](#)  
[IS\\_SRVROLEMEMBER \(Transact-SQL\)](#)

## Security Functions (Transact-SQL)

# IS\_SRVROLEMEMBER (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Indicates whether a SQL Server login is a member of the specified server role.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
IS_SRVROLEMEMBER ( 'role' [ , 'login' ] )
```

## Arguments

**'role'**

Is the name of the server role that is being checked. *role* is **sysname**.

Valid values for *role* are user-defined server roles, and the following fixed server roles:

sysadmin	serveradmin
dbcreator	setupadmin
bulkadmin	securityadmin
diskadmin	<b>Applies to:</b> SQL Server 2012 through SQL Server 2017. public
processadmin	

**'login'**

Is the name of the SQL Server login to check. *login* is **sysname**, with a default of NULL. If no value is specified, the result is based on the current Execution context. If the parameter contains the word NULL will return NULL.

## Return Types

**int**

RETURN VALUE	DESCRIPTION
0	<i>login</i> is not a member of <i>role</i> .  In Azure SQL Database, this statement always returns 0.
1	<i>login</i> is a member of <i>role</i> .

RETURN VALUE	DESCRIPTION
NULL	<i>role</i> or <i>login</i> is not valid, or you do not have permission to view the role membership.

## Remarks

Use `IS_SRVROLEMEMBER` to determine whether the current user can perform an action requiring the server role's permissions.

If a Windows login, such as `Contoso\Mary5`, is specified for *login*, **IS\_SRVROLEMEMBER** returns **NULL**, unless the login has been granted or denied direct access to SQL Server.

If the optional *login* parameter is not provided and if *login* is a Windows domain login, it may be a member of a fixed server role through membership in a Windows group. To resolve such indirect memberships, **IS\_SRVROLEMEMBER** requests Windows group membership information from the domain controller. If the domain controller is inaccessible or does not respond, **IS\_SRVROLEMEMBER** returns role membership information by accounting for the user and its local groups only. If the user specified is not the current user, the value returned by **IS\_SRVROLEMEMBER** might differ from the authenticator's (such as Active Directory) last data update to SQL Server.

If the optional *login* parameter is provided, the Windows login that is being queried must be present in `sys.server_principals`, or **IS\_SRVROLEMEMBER** will return **NULL**. This indicates that the login is not valid.

When the *login* parameter is a domain login or based on a Windows group and the domain controller is inaccessible, calls to **IS\_SRVROLEMEMBER** will fail and might return incorrect or incomplete data.

If the domain controller is not available, the call to **IS\_SRVROLEMEMBER** will return accurate information when the Windows principle can be authenticated locally, such as a local Windows account or a SQL Server login.

**IS\_SRVROLEMEMBER** always returns 0 when a Windows group is used as the *login* argument, and this Windows group is a member of another Windows group which is, in turn, a member of the specified server role.

The User Account Control (UAC) setting might also cause the return different results. This would depend on whether the user accessed the server as a Windows group member or as a specific SQL Server user.

This function evaluates role membership, not the underlying permission. For example, the **sysadmin** fixed server role has the **CONTROL SERVER** permission. If the user has the **CONTROL SERVER** permission but is not a member of the role, this function will correctly report that the user is not a member of the **sysadmin** role, even though the user has the same permissions.

## Related Functions

To determine whether the current user is a member of the specified Windows group or SQL Server database role, use [IS\\_MEMBER \(Transact-SQL\)](#). To determine whether a SQL Server login is a member of a database role, use [IS\\_ROLEMEMBER \(Transact-SQL\)](#).

## Permissions

Requires **VIEW DEFINITION** permission on the server role.

## Examples

The following example indicates whether the SQL Server login for the current user is a member of the **sysadmin** fixed server role.

```
IF IS_SRVROLEMEMBER ('sysadmin') = 1
    print 'Current user''s login is a member of the sysadmin role'
ELSE IF IS_SRVROLEMEMBER ('sysadmin') = 0
    print 'Current user''s login is NOT a member of the sysadmin role'
ELSE IF IS_SRVROLEMEMBER ('sysadmin') IS NULL
    print 'ERROR: The server role specified is not valid.';
```

The following example indicates whether the domain login Pat is a member of the **diskadmin** fixed server role.

```
SELECT IS_SRVROLEMEMBER('diskadmin', 'Contoso\Pat');
```

## See Also

[IS\\_MEMBER \(Transact-SQL\)](#)

[Security Functions \(Transact-SQL\)](#)

# LOGINPROPERTY (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns information about login policy settings.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
LOGINPROPERTY ( 'login_name' , 'property_name' )
```

## Arguments

*login\_name*

Is the name of a SQL Server login for which login property status will be returned.

*propertyname*

Is an expression that contains the property information to be returned for the login. *propertyname* can be one of the following values.

VALUE	DESCRIPTION
<b>BadPasswordCount</b>	Returns the number of consecutive attempts to log in with an incorrect password.
<b>BadPasswordTime</b>	Returns the time of the last attempt to log in with an incorrect password.
<b>DaysUntilExpiration</b>	Returns the number of days until the password expires.
<b>DefaultDatabase</b>	Returns the SQL Server login default database as stored in metadata or <b>master</b> if no database is specified. Returns NULL for non- SQL Server provisioned users (for example, Windows authenticated users).
<b>DefaultLanguage</b>	Returns the login default language as stored in metadata. Returns NULL for non- SQL Server provisioned users, for example, Windows authenticated users.
<b>HistoryLength</b>	Returns the number of passwords tracked for the login, using the password-policy enforcement mechanism. 0 if the password policy is not enforced. Resuming password policy enforcement restarts at 1.
<b>IsExpired</b>	Indicates whether the login has expired.
<b>IsLocked</b>	Indicates whether the login is locked.

VALUE	DESCRIPTION
<b>IsMustChange</b>	Indicates whether the login must change its password the next time it connects.
<b>LockoutTime</b>	Returns the date when the SQL Server login was locked out because it had exceeded the permitted number of failed login attempts.
<b>PasswordHash</b>	Returns the hash of the password.
<b>PasswordLastSetTime</b>	Returns the date when the current password was set.
<b>PasswordHashAlgorithm</b>	Returns the algorithm used to hash the password.

## Returns

Data type depends on requested value.

**IsLocked**, **IsExpired**, and **IsMustChange** are of type **int**.

- 1 if the login is in the specified state.
- 0 if the login is not in the specified state.

**BadPasswordCount** and **HistoryLength** are of type **int**.

**BadPasswordTime**, **LockoutTime**, **PasswordLastSetTime** are of type **datetime**.

**PasswordHash** is of type **varbinary**.

NULL if the login is not a valid SQL Server login.

**DaysUntilExpiration** is of type **int**.

- 0 if the login is expired or if it will expire on the day when queried.
- -1 if the local security policy in Windows never expires the password.
- NULL if the CHECK\_POLICY or CHECK\_EXPIRATION is OFF for a login, or if the operating system does not support the password policy.

**PasswordHashAlgorithm** is of type **int**.

- 0 if a SQL7.0 hash
- 1 if a SHA-1 hash
- 2 if a SHA-2 hash
- NULL if the login is not a valid SQL Server login

## Remarks

This built-in function returns information about the password policy settings of a SQL Server login. The names of the properties are not case sensitive, so property names such as **BadPasswordCount** and **badpasswordcount** are equivalent. The values of the **PasswordHash**, **PasswordHashAlgorithm**, and **PasswordLastSetTime** properties are available on all supported configurations of SQL Server, but the other properties are only available when SQL Server is running on Windows Server 2003 and both CHECK\_POLICY and CHECK\_EXPIRATION are enabled. For

more information, see [Password Policy](#).

## Permissions

Requires VIEW permission on the login. When requesting the password hash, also requires CONTROL SERVER permission.

## Examples

### A. Checking whether a login must change its password

The following example checks whether SQL Server login `John3` must change its password the next time it connects to an instance of SQL Server.

```
SELECT LOGINPROPERTY('John3', 'IsMustChange');
GO
```

### B. Checking whether a login is locked out

The following example checks whether SQL Server login `John3` is locked.

```
SELECT LOGINPROPERTY('John3', 'IsLocked');
GO
```

## See Also

[CREATE LOGIN \(Transact-SQL\)](#)  
[sys.server\\_principals \(Transact-SQL\)](#)

# ORIGINAL\_LOGIN (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the name of the login that connected to the instance of SQL Server. You can use this function to return the identity of the original login in sessions in which there are many explicit or implicit context switches.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
ORIGINAL_LOGIN( )
```

## Return Types

**sysname**

## Remarks

This function can be useful in auditing the identity of the original connecting context. Whereas functions such as [SESSION\\_USER](#) and [CURRENT\\_USER](#) return the current executing context, [ORIGINAL\\_LOGIN](#) returns the identity of the login that first connected to the instance of SQL Server in that session.

Returns NULL on Azure SQL Database.

## Examples

The following example switches the execution context of the current session from the caller of the statements to `login1`. The functions `SUSER_SNAME` and `ORIGINAL_LOGIN` are used to return the current session user (the user to whom the context was switched), and the original login account.

```
USE AdventureWorks2012;
GO
--Create a temporary login and user.
CREATE LOGIN login1 WITH PASSWORD = 'J345#)$thb';
CREATE USER user1 FOR LOGIN login1;
GO
--Execute a context switch to the temporary login account.
DECLARE @original_login sysname;
DECLARE @current_context sysname;
EXECUTE AS LOGIN = 'login1';
SET @original_login = ORIGINAL_LOGIN();
SET @current_context = SUSER_SNAME();
SELECT 'The current executing context is: '+@current_context;
SELECT 'The original login in this session was: '+@original_login
GO
-- Return to the original execution context
-- and remove the temporary principal.
REVERT;
GO
DROP LOGIN login1;
DROP USER user1;
GO
```

## See Also

[EXECUTE AS \(Transact-SQL\)](#)

[REVERT \(Transact-SQL\)](#)

# PERMISSIONS (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a value containing a bitmap that indicates the statement, object, or column permissions of the current user.

**Important** This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Use [fn\\_my\\_permissions](#) and [Has\\_Perms\\_By\\_Name](#) instead. Continued use of the PERMISSIONS function may result in slower performance.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
PERMISSIONS ( [ objectid [ , 'column' ] ] )
```

## Arguments

### *objectid*

Is the ID of a securable. If *objectid* is not specified, the bitmap value contains statement permissions for the current user; otherwise, the bitmap contains permissions on the securable for the current user. The securable specified must be in the current database. Use the [OBJECT\\_ID](#) function to determine the *objectid* value.

### '*column*'

Is the optional name of a column for which permission information is being returned. The column must be a valid column name in the table specified by *objectid*.

## Return Types

### **int**

## Remarks

PERMISSIONS can be used to determine whether the current user has the permissions required to execute a statement or to GRANT a permission to another user.

The permissions information returned is a 32-bit bitmap.

The lower 16 bits reflect permissions granted to the user, and also permissions that are applied to Windows groups or and fixed server roles of which the current user is a member. For example, a returned value of 66 (hex value 0x42), when no *objectid* is specified, indicates that the user has permission to execute the CREATE TABLE (decimal value 2) and BACKUP DATABASE (decimal value 64) statements.

The upper 16 bits reflect the permissions that the user can GRANT to other users. The upper 16 bits are interpreted exactly as those for the lower 16 bits described in the following tables, except they are shifted to the left by 16 bits (multiplied by 65536). For example, 0x8 (decimal value 8) is the bit that indicates INSERT permission when an *objectid* is specified. Whereas, 0x80000 (decimal value 524288) indicates the ability to GRANT INSERT permission, because  $524288 = 8 \times 65536$ .

Because of membership in roles, a user that does not have permission to execute a statement may still be able to grant that permission to another user.

The following table shows the bits that are used for statement permissions (*objectid* is not specified).

BIT (DEC)	BIT (HEX)	STATEMENT PERMISSION
1	0x1	CREATE DATABASE (master database only)
2	0x2	CREATE TABLE
4	0x4	CREATE PROCEDURE
8	0x8	CREATE VIEW
16	0x10	CREATE RULE
32	0x20	CREATE DEFAULT
64	0x40	BACKUP DATABASE
128	0x80	BACKUP LOG
256	0x100	Reserved

The following table shows the bits used for object permissions that are returned when only *objectid* is specified.

BIT (DEC)	BIT (HEX)	STATEMENT PERMISSION
1	0x1	SELECT ALL
2	0x2	UPDATE ALL
4	0x4	REFERENCES ALL
8	0x8	INSERT
16	0x10	DELETE
32	0x20	EXECUTE (procedures only)
4096	0x1000	SELECT ANY (at least one column)
8192	0x2000	UPDATE ANY
16384	0x4000	REFERENCES ANY

The following table shows the bits used for column-level object permissions that are returned when both *objectid* and column are specified.

BIT (DEC)	BIT (HEX)	STATEMENT PERMISSION
1	0x1	SELECT
2	0x2	UPDATE
4	0x4	REFERENCES

A NULL is returned when a specified parameter is NULL or not valid (for example, an *objectid* or column that does not exist). The bit values for permissions that do not apply (for example EXECUTE permission, bit 0x20, for a table) are undefined.

Use the bitwise AND (&) operator to determine each bit set in the bitmap that is returned by the PERMISSIONS function.

The sp\_helpprotect system stored procedure can also be used to return a list of permissions for a user in the current database.

## Examples

### A. Using the PERMISSIONS function with statement permissions

The following example determines whether the current user can execute the CREATE TABLE statement.

```
IF PERMISSIONS()&2=2
    CREATE TABLE test_table (col1 INT)
ELSE
    PRINT 'ERROR: The current user cannot create a table.';
```

### B. Using the PERMISSIONS function with object permissions

The following example determines whether the current user can insert a row of data into the Address table in the AdventureWorks2012 database.

```
IF PERMISSIONS(OBJECT_ID('AdventureWorks2012.Person.Address', 'U'))&8=8
    PRINT 'The current user can insert data into Person.Address.'
ELSE
    PRINT 'ERROR: The current user cannot insert data into Person.Address.';
```

### C. Using the PERMISSIONS function with grantable permissions

The following example determines whether the current user can grant the INSERT permission on the Address table in the AdventureWorks2012 database to another user.

```
IF PERMISSIONS(OBJECT_ID('AdventureWorks2012.Person.Address', 'U'))&0x80000=0x80000
    PRINT 'INSERT on Person.Address is grantable.'
ELSE
    PRINT 'You may not GRANT INSERT permissions on Person.Address.';
```

## See Also

[DENY \(Transact-SQL\)](#)

[GRANT \(Transact-SQL\)](#)

[OBJECT\\_ID \(Transact-SQL\)](#)

[REVOKE \(Transact-SQL\)](#)

[sp\\_helpprotect \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

# PWDENCRYPT (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the SQL Server password hash of the input value that uses the current version of the password hashing algorithm.

PWDENCRYPT is an older function and might not be supported in a future release of SQL Server. Use [HASHBYTES](#) instead. HASHBYTES provides more hashing algorithms.



## Syntax

```
PWDENCRYPT ( 'password' )
```

## Arguments

*password*

Is the password to be encrypted. *password* is **sysname**.

## Return Types

**varbinary(128)**

## Permissions

PWDENCRYPT is available to public.

## See Also

[Security Functions \(Transact-SQL\)](#)

[PWDCOMPARE \(Transact-SQL\)](#)

# PWDCOMPARE (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Hashes a password and compares the hash to the hash of an existing password. PWDCOMPARE can be used to search for blank SQL Server login passwords or common weak passwords.



## Syntax

```
PWDCOMPARE ( 'clear_text_password'  
    , password_hash  
    [ , version ] )
```

## Arguments

*'clear\_text\_password'*

Is the unencrypted password. *clear\_text\_password* is **sysname (nvarchar(128))**.

*password\_hash*

Is the encryption hash of a password. *password\_hash* is **varbinary(128)**.

*version*

Obsolete parameter that can be set to 1 if *password\_hash* represents a value from a login earlier than SQL Server 2000 that was migrated to SQL Server 2005 or later but never converted to the SQL Server 2000 system. *version* is **int**.

### Caution

This parameter is provided for backwards compatibility, but is ignored because password hash blobs now contain their own version descriptions. This feature will be removed in the next version of Microsoft SQL Server. Do not use this feature in new development work, and modify applications that currently use this feature as soon as possible.

## Return Types

**int**

Returns 1 if the hash of the *clear\_text\_password* matches the *password\_hash* parameter, and 0 if it does not.

## Remarks

The PWDCOMPARE function is not a threat against the strength of password hashes because the same test could be performed by trying to log in using the password provided as the first parameter.

**PWDCOMPARE** cannot be used with the passwords of contained database users. There is no contained database equivalent.

## Permissions

PWDENCRYPT is available to public.

CONTROL SERVER permission is required to examine the password\_hash column of sys.sql\_logins.

## Examples

### A. Identifying logins that have no passwords

The following example identifies SQL Server logins that have no passwords.

```
SELECT name FROM sys.sql_logins  
WHERE PWDCOMPARE('', password_hash) = 1 ;
```

### B. Searching for common passwords

To search for common passwords that you want to identify and change, specify the password as the first parameter. For example, execute the following statement to search for a password specified as `password`.

```
SELECT name FROM sys.sql_logins  
WHERE PWDCOMPARE('password', password_hash) = 1 ;
```

## See Also

[PWDENCRYPT \(Transact-SQL\)](#)

[Security Functions \(Transact-SQL\)](#)

# SESSION\_USER (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

SESSION\_USER returns the user name of the current context in the current database.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SESSION_USER
```

## Return Types

**nvarchar(128)**

## Remarks

Use SESSION\_USER with DEFAULT constraints in either the CREATE TABLE or ALTER TABLE statements, or use it as any standard function. SESSION\_USER can be inserted into a table when no default value is specified. This function takes no arguments. SESSION\_USER can be used in queries.

If SESSION\_USER is called after a context switch, SESSION\_USER will return the user name of the impersonated context.

## Examples

### A. Using SESSION\_USER to return the user name of the current session

The following example declares a variable as `nchar`, assigns the current value of `SESSION_USER` to that variable, and then prints the variable with a text description.

```
DECLARE @session_usr nchar(30);
SET @session_usr = SESSION_USER;
SELECT 'This session''s current user is: '+ @session_usr;
GO
```

This is the result set when the session user is `surya`:

```
-----
```

```
This session's current user is: Surya
```

```
(1 row(s) affected)
```

### B. Using SESSION\_USER with DEFAULT constraints

The following example creates a table that uses `SESSION_USER` as a `DEFAULT` constraint for the name of the person who records receipt of a shipment.

```

USE AdventureWorks2012;
GO
CREATE TABLE deliveries3
(
    order_id int IDENTITY(5000, 1) NOT NULL,
    cust_id int NOT NULL,
    order_date smalldatetime NOT NULL DEFAULT GETDATE(),
    delivery_date smalldatetime NOT NULL DEFAULT
        DATEADD(dd, 10, GETDATE()),
    received_shipment nchar(30) NOT NULL DEFAULT SESSION_USER
);
GO

```

Records added to the table will be stamped with the user name of the current user. In this example, `Wanida`, `Sylvester`, and `Alejandro` verify receipt of shipments. This can be emulated by switching user context by using `EXECUTE AS`.

```

EXECUTE AS USER = 'Wanida'
INSERT deliveries3 (cust_id)
VALUES (7510);
INSERT deliveries3 (cust_id)
VALUES (7231);
REVERT;
EXECUTE AS USER = 'Sylvester'
INSERT deliveries3 (cust_id)
VALUES (7028);
REVERT;
EXECUTE AS USER = 'Alejandro'
INSERT deliveries3 (cust_id)
VALUES (7392);
INSERT deliveries3 (cust_id)
VALUES (7452);
REVERT;
GO

```

The following query selects all information from the `deliveries3` table.

```

SELECT order_id AS 'Order #', cust_id AS 'Customer #',
       delivery_date AS 'When Delivered', received_shipment
           AS 'Received By'
  FROM deliveries3
 ORDER BY order_id;
GO

```

Here is the result set.

Order #	Customer #	When Delivered	Received By
---------	------------	----------------	-------------

---

5000 7510 2005-03-16 12:02:14 Wanida
--------------------------------------

5001 7231 2005-03-16 12:02:14 Wanida
--------------------------------------

5002 7028 2005-03-16 12:02:14 Sylvester
---

5003 7392 2005-03-16 12:02:14 Alejandro
---

5004 7452 2005-03-16 12:02:14 Alejandro
---

(5 row(s) affected)
---------------------

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C: Using SESSION\_USER to return the user name of the current session

The following example returns the session user for the current session.

```
SELECT SESSION_USER;
```

## See Also

[ALTER TABLE \(Transact-SQL\)](#)  
[CREATE TABLE \(Transact-SQL\)](#)  
[CURRENT\\_TIMESTAMP \(Transact-SQL\)](#)  
[CURRENT\\_USER \(Transact-SQL\)](#)  
[SYSTEM\\_USER \(Transact-SQL\)](#)  
[System Functions \(Transact-SQL\)](#)  
[USER \(Transact-SQL\)](#)  
[USER\\_NAME \(Transact-SQL\)](#)

# SESSIONPROPERTY (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the SET options settings of a session.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
SESSIONPROPERTY (option)
```

## Arguments

*option*

Is the current option setting for this session. *option* can be any of the following values.

OPTION	DESCRIPTION
ANSI_NULLS	Specifies whether the ISO behavior of equals (=) and not equal to (<>) against null values is applied. 1 = ON 0 = OFF
ANSI_PADDING	Controls the way the column stores values shorter than the defined size of the column, and the way the column stores values that have trailing blanks in character and binary data. 1 = ON 0 = OFF
ANSI_WARNINGS	Specifies whether the ISO standard behavior of raising error messages or warnings for certain conditions, including divide-by-zero and arithmetic overflow, is applied. 1 = ON 0 = OFF
ARITHABORT	Determines whether a query is ended when an overflow or a divide-by-zero error occurs during query execution. 1 = ON 0 = OFF

OPTION	DESCRIPTION
CONCAT_NULL_YIELDS_NULL	Controls whether concatenation results are treated as null or empty string values. 1 = ON 0 = OFF
NUMERIC_ROUNDABORT	Specifies whether error messages and warnings are generated when rounding in an expression causes a loss of precision. 1 = ON 0 = OFF
QUOTED_IDENTIFIER	Specifies whether ISO rules about how to use quotation marks to delimit identifiers and literal strings are to be followed. 1 = ON 0 = OFF
<Any other string>	NULL = Input is not valid.

## Return Types

### **sql\_variant**

## Remarks

SET options are figured by combining server-level, database-level, and user-specified options.

## Examples

The following example returns the setting for the `CONCAT_NULL_YIELDS_NULL` option.

```
SELECT SESSIONPROPERTY ('CONCAT_NULL_YIELDS_NULL')
```

## See Also

[sql\\_variant \(Transact-SQL\)](#)  
[SET ANSI\\_NULLS \(Transact-SQL\)](#)  
[SET ANSI\\_PADDING \(Transact-SQL\)](#)  
[SET ANSI\\_WARNINGS \(Transact-SQL\)](#)  
[SET ARITHABORT \(Transact-SQL\)](#)  
[SET CONCAT\\_NULL\\_YIELDS\\_NULL \(Transact-SQL\)](#)  
[SET NUMERIC\\_ROUNDABORT \(Transact-SQL\)](#)  
[SET QUOTED\\_IDENTIFIER \(Transact-SQL\)](#)

# SUSER\_ID (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the login identification number of the user.

## NOTE

Starting with SQL Server 2005, SUSER\_ID returns the value listed as **principal\_id** in the **sys.server\_principals** catalog view.



## Syntax

```
SUSER_ID ( [ 'login' ] )
```

## Arguments

'*login*'

Is the login name of the user. *login* is **nchar**. If *login* is specified as **char**, *login* is implicitly converted to **nchar**. *login* can be any SQL Server login or Windows user or group that has permission to connect to an instance of SQL Server. If *login* is not specified, the login identification number for the current user is returned. If the parameter contains the word NULL will return NULL.

## Return Types

**int**

## Remarks

SUSER\_ID returns an identification number only for logins that have been explicitly provisioned inside SQL Server. This ID is used within SQL Server to track ownership and permissions. This ID is not equivalent to the SID of the login that is returned by SUSER\_SID. If *login* is a SQL Server login, the SID maps to a GUID. If *login* is a Windows login or Windows group, the SID maps to a Windows security identifier.

SUSER\_SID returns a SUID only for a login that has an entry in the **syslogins** system table.

System functions can be used in the select list, in the WHERE clause, and anywhere an expression is allowed, and must always be followed by parentheses, even if no parameter is specified.

## Examples

The following example returns the login identification number for the `sa` login.

```
SELECT SUSER_ID('sa');
```

## See Also

[sys.server\\_principals \(Transact-SQL\)](#)

[SUSER\\_SID \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

# SUSER\_NAME (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data

Warehouse Parallel Data Warehouse

Returns the login identification name of the user.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
SUSER_NAME ( [ server_user_id ] )
```

## Arguments

*server\_user\_id*

Is the login identification number of the user. *server\_user\_id*, which is optional, is **int**. *server\_user\_id* can be the login identification number of any SQL Server login or Microsoft Windows user or group that has permission to connect to an instance of SQL Server. If *server\_user\_id* is not specified, the login identification name for the current user is returned. If the parameter contains the word NULL will return NULL.

## Return Types

**nvarchar(128)**

## Remarks

In SQL Server version 7.0, the security identification number (SID) replaced the server user identification number (SUID).

SUSER\_NAME returns a login name only for a login that has an entry in the **syslogins** system table.

SUSER\_NAME can be used in a select list, in a WHERE clause, and anywhere an expression is allowed, and must always be followed by parentheses, even if no parameter is specified.

## Examples

The following example returns the login identification name of the user with a login identification number of 1.

```
SELECT SUSER_NAME(1);
```

## See Also

[SUSER\\_ID \(Transact-SQL\)](#)

[Principals \(Database Engine\)](#)

# SUSER\_SID (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Returns the security identification number (SID) for the specified login name.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SUSER_SID ( [ 'login' ] [ , Param2 ] )
```

## Arguments

*'login'*

**Applies to:** SQL Server 2008 through SQL Server 2017

Is the login name of the user. *login* is **sysname**. *login*, which is optional, can be a SQL Server login or Microsoft Windows user or group. If *login* is not specified, information about the current security context is returned. If the parameter contains the word NULL will return NULL.

*Param2*

**Applies to:** SQL Server 2012 through SQL Server 2017

Specifies whether the login name is validated. *Param2* is of type **int** and is optional. When *Param2* is 0, the login name is not validated. When *Param2* is not specified as 0, the Windows login name is verified to be exactly the same as the login name stored in SQL Server.

## Return Types

**varbinary(85)**

## Remarks

SUSER\_SID can be used as a DEFAULT constraint in either ALTER TABLE or CREATE TABLE. SUSER\_SID can be used in a select list, in a WHERE clause, and anywhere an expression is allowed. SUSER\_SID must always be followed by parentheses, even if no parameter is specified.

When called without an argument, SUSER\_SID returns the SID of the current security context. When called without an argument within a batch that has switched context by using EXECUTE AS, SUSER\_SID returns the SID of the impersonated context. When called from an impersonated context, SUSER\_SID(ORIGINAL\_LOGIN()) returns the SID of the original context.

When the SQL Server collation and the Windows collation are different, SUSER\_SID can fail when SQL Server and Windows store the login in a different format. For example, if the Windows computer TestComputer has the login User and SQL Server stores the login as TESTCOMPUTER\User, the lookup of the login TestComputer\User might fail to resolve the login name correctly. To skip this validation of the login name, use *Param2*. Differing collations is often a cause of SQL Server error 15401:

```
Windows NT user or group '%s' not found. Check the name again.
```

## Examples

### A. Using SUSER\_SID

The following example returns the security identification number (SID) for the current security context.

```
SELECT SUSER_SID('sa');
```

### B. Using SUSER\_SID with a specific login

The following example returns the security identification number for the SQL Server `sa` login.

**Applies to:** SQL Server 2012 through SQL Server 2017

```
SELECT SUSER_SID('sa');
GO
```

### C. Using SUSER\_SID with a Windows user name

The following example returns the security identification number for the Windows user `London\Workstation1`.

**Applies to:** SQL Server 2012 through SQL Server 2017

```
SELECT SUSER_SID('London\Workstation1');
GO
```

### D. Using SUSER\_SID as a DEFAULT constraint

The following example uses `SUSER_SID` as a `DEFAULT` constraint in a `CREATE TABLE` statement.

```
USE AdventureWorks2012;
GO
CREATE TABLE sid_example
(
    login_sid   varbinary(85) DEFAULT SUSER_SID(),
    login_name  varchar(30)  DEFAULT SYSTEM_USER,
    login_dept  varchar(10)  DEFAULT 'SALES',
    login_date  datetime    DEFAULT GETDATE()
);
GO
INSERT sid_example DEFAULT VALUES;
GO
```

### E. Comparing the Windows login name to the login name stored in SQL Server

The following example shows how to use `Param2` to obtain the SID from Windows and uses that SID as an input to the `SUSER_SNAME` function. The example provides the login in the format in which it is stored in Windows (`TestComputer\User`), and returns the login in the format in which it is stored in SQL Server (`TESTCOMPUTER\User`).

**Applies to:** SQL Server 2012 through SQL Server 2017

```
SELECT SUSER_SNAME(SUSER_SID('TestComputer\User', 0));
```

## See Also

[ORIGINAL\\_LOGIN \(Transact-SQL\)](#)

[CREATE TABLE \(Transact-SQL\)](#)  
[binary and varbinary \(Transact-SQL\)](#)  
[System Functions \(Transact-SQL\)](#)

# SUSER\_SNAME (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the login name associated with a security identification number (SID).

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SUSER_SNAME ( [ server_user_sid ] )
```

## Arguments

*server\_user\_sid*

**Applies to:** SQL Server 2008 through SQL Server 2017

Is the optional login security identification number. *server\_user\_sid* is **varbinary(85)**. *server\_user\_sid* can be the security identification number of any SQL Server login or Microsoft Windows user or group. If *server\_user\_sid* is not specified, information about the current user is returned. If the parameter contains the word NULL will return NULL.

## Return Types

**nvarchar(128)**

## Remarks

SUSER\_SNAME can be used as a DEFAULT constraint in either ALTER TABLE or CREATE TABLE. SUSER\_SNAME can be used in a select list, in a WHERE clause, and anywhere an expression is allowed. SUSER\_SNAME must always be followed by parentheses, even if no parameter is specified.

When called without an argument, SUSER\_SNAME returns the name of the current security context. When called without an argument within a batch that has switched context by using EXECUTE AS, SUSER\_SNAME returns the name of the impersonated context. When called from an impersonated context, ORIGINAL\_LOGIN returns the name of the original context.

## Azure SQL Database Remarks

SUSER\_NAME always return the login name for the current security context.

The SUSER\_SNAME statement does not support execution using an impersonated security context through EXECUTE AS.

## Examples

### A. Using SUSER\_SNAME

The following example returns the login name for the current security context.

```
SELECT SUSER_SNAME();
GO
```

## B. Using SUSER\_SNAME with a Windows user security ID

The following example returns the login name associated with a Windows security identification number.

**Applies to:** SQL Server 2008 through SQL Server 2017

```
SELECT SUSER_SNAME(0x01050000000000051500000a065cf7e784b9b5fe77c87705a2e0000);
GO
```

## C. Using SUSER\_SNAME as a DEFAULT constraint

The following example uses `SUSER_SNAME` as a `DEFAULT` constraint in a `CREATE TABLE` statement.

```
USE AdventureWorks2012;
GO
CREATE TABLE sname_example
(
    login_sname sysname DEFAULT SUSER_SNAME(),
    employee_id uniqueidentifier DEFAULT NEWID(),
    login_date datetime DEFAULT GETDATE()
);
GO
INSERT sname_example DEFAULT VALUES;
GO
```

## D. Calling SUSER\_SNAME in combination with EXECUTE AS

This example shows the behavior of `SUSER_SNAME` when called from an impersonated context.

**Applies to:** SQL Server 2008 through SQL Server 2017

```
SELECT SUSER_SNAME();
GO
EXECUTE AS LOGIN = 'WanidaBenShoof';
SELECT SUSER_SNAME();
REVERT;
GO
SELECT SUSER_SNAME();
GO
```

Here is the result.

```
sa
```

```
WanidaBenShoof
```

```
sa
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### E. Using SUSER\_SNAME

The following example returns the login name for the security identification number with a value of `0x01`.

```
SELECT SUSER_SNAME(0x01);
GO
```

## F. Returning the Current Login

The following example returns the login name of the current login.

```
SELECT SUSER_SNAME() AS CurrentLogin;
GO
```

## See Also

[SUSER\\_SID \(Transact-SQL\)](#)

[Principals \(Database Engine\)](#)

[sys.server\\_principals \(Transact-SQL\)](#)

[EXECUTE AS \(Transact-SQL\)](#)

# SYSTEM\_USER (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Allows a system-supplied value for the current login to be inserted into a table when no default value is specified.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server, Azure SQL Data Warehouse, Parallel Data Warehouse  
  
SYSTEM_USER
```

## Return Types

**nchar**

## Remarks

You can use the SYSTEM\_USER function with DEFAULT constraints in the CREATE TABLE and ALTER TABLE statements. You can also use it as any standard function.

If the user name and login name are different, SYSTEM\_USER returns the login name.

If the current user is logged in to SQL Server by using Windows Authentication, SYSTEM\_USER returns the Windows login identification name in the form: *DOMAIN\user\_login\_name*. However, if the current user is logged in to SQL Server by using SQL Server Authentication, SYSTEM\_USER returns the SQL Server login identification name, such as `willisJo` for a user logged in as `WillisJo`.

SYSTEM\_USER returns the name of the currently executing context. If the EXECUTE AS statement has been used to switch context, SYSTEM\_USER returns the name of the impersonated context.

## Examples

### A. Using SYSTEM\_USER to return the current system user name

The following example declares a `char` variable, stores the current value of `SYSTEM_USER` in the variable, and then prints the value stored in the variable.

```
DECLARE @sys_usr char(30);  
SET @sys_usr = SYSTEM_USER;  
SELECT 'The current system user is: '+ @sys_usr;  
GO
```

Here is the result set.

```
-----  
The current system user is: WillisJo
```

```
(1 row(s) affected)
```

## B. Using SYSTEM\_USER with DEFAULT constraints

The following example creates a table with `SYSTEM_USER` as a `DEFAULT` constraint for the `SRep_tracking_user` column.

```
USE AdventureWorks2012;
GO
CREATE TABLE Sales.Sales_Tracking
(
    Territory_id int IDENTITY(2000, 1) NOT NULL,
    Rep_id int NOT NULL,
    Last_sale datetime NOT NULL DEFAULT GETDATE(),
    SRep_tracking_user varchar(30) NOT NULL DEFAULT SYSTEM_USER
);
GO
INSERT Sales.Sales_Tracking (Rep_id)
VALUES (151);
INSERT Sales.Sales_Tracking (Rep_id, Last_sale)
VALUES (293, '19980515');
INSERT Sales.Sales_Tracking (Rep_id, Last_sale)
VALUES (27882, '19980620');
INSERT Sales.Sales_Tracking (Rep_id)
VALUES (21392);
INSERT Sales.Sales_Tracking (Rep_id, Last_sale)
VALUES (24283, '19981130');
GO
```

The following query selects all the information from the `Sales_Tracking` table:

```
SELECT * FROM Sales_Tracking ORDER BY Rep_id;
GO
```

Here is the result set.

Territory_id	Rep_id	Last_sale	SRep_tracking_user
2000	151	Mar 4 1998 10:36AM	ArvinDak
2001	293	May 15 1998 12:00AM	ArvinDak
2003	21392	Mar 4 1998 10:36AM	ArvinDak
2004	24283	Nov 3 1998 12:00AM	ArvinDak
2002	27882	Jun 20 1998 12:00AM	ArvinDak

2000 151 Mar 4 1998 10:36AM ArvinDak
--------------------------------------

2001 293 May 15 1998 12:00AM ArvinDak
---------------------------------------

2003 21392 Mar 4 1998 10:36AM ArvinDak
--

2004 24283 Nov 3 1998 12:00AM ArvinDak
--

2002 27882 Jun 20 1998 12:00AM ArvinDak
---

```
(5 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C: Using SYSTEM\_USER to return the current system user name

The following example returns the current value of `SYSTEM_USER`.

```
SELECT SYSTEM_USER;
```

## See Also

[ALTER TABLE \(Transact-SQL\)](#)  
[CREATE TABLE \(Transact-SQL\)](#)  
[CURRENT\\_TIMESTAMP \(Transact-SQL\)](#)  
[CURRENT\\_USER \(Transact-SQL\)](#)  
[SESSION\\_USER \(Transact-SQL\)](#)  
[System Functions \(Transact-SQL\)](#)  
[USER \(Transact-SQL\)](#)

# USER (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Allows a system-supplied value for the database user name of the current user to be inserted into a table when no default value is specified.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
USER
```

## Return Types

**char**

## Remarks

USER provides the same functionality as the USER\_NAME system function.

Use USER with DEFAULT constraints in either the CREATE TABLE or ALTER TABLE statements, or use as any standard function.

USER always returns the name of the current context. When called after an EXECUTE AS statement, USER returns the name of the impersonated context.

If a Windows principal accesses the database by way of membership in a group, USER returns the name of the Windows principal instead of the name of the group.

## Examples

### A. Using USER to return the database user name

The following example declares a variable as `char`, assigns the current value of USER to it, and then prints the variable with a text description.

```
DECLARE @usr char(30)
SET @usr = user
SELECT 'The current user''s database username is: '+ @usr
GO
```

Here is the result set.

```
-----
```

```
The current user's database username is: dbo
```

```
(1 row(s) affected)
```

### B. Using USER with DEFAULT constraints

The following example creates a table by using `USER` as a `DEFAULT` constraint for the salesperson of a sales row.

```
USE AdventureWorks2012;
GO
CREATE TABLE inventory22
(
    part_id int IDENTITY(100, 1) NOT NULL,
    description varchar(30) NOT NULL,
    entry_person varchar(30) NOT NULL DEFAULT USER
)
GO
INSERT inventory22 (description)
VALUES ('Red pencil')
INSERT inventory22 (description)
VALUES ('Blue pencil')
INSERT inventory22 (description)
VALUES ('Green pencil')
INSERT inventory22 (description)
VALUES ('Black pencil')
INSERT inventory22 (description)
VALUES ('Yellow pencil')
GO
```

This is the query to select all information from the `inventory22` table:

```
SELECT * FROM inventory22 ORDER BY part_id;
GO
```

Here is the result set (note the `entry-person` value):

part_id	description	entry_person
---------	-------------	--------------

100	Red pencil	dbo
-----	------------	-----

101	Blue pencil	dbo
-----	-------------	-----

102	Green pencil	dbo
-----	--------------	-----

103	Black pencil	dbo
-----	--------------	-----

104	Yellow pencil	dbo
-----	---------------	-----

(5 row(s) affected)
---------------------

## C. Using `USER` in combination with `EXECUTE AS`

The following example illustrates the behavior of `USER` when called inside an impersonated session.

```
SELECT USER;
GO
EXECUTE AS USER = 'Mario';
GO
SELECT USER;
GO
REVERT;
GO
SELECT USER;
GO
```

Here is the result set.

dbo

Mario

dbo

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D. Using **USER** to return the database user name

The following example declares a variable as `char`, assigns the current value of `USER` to it, and then prints the variable with a text description.

```
DECLARE @usr char(30)
SET @usr = user
SELECT 'The current user''s database username is: '+ @usr
GO
```

Here is the result set.

```
-----
The current user's database username is: dbo
(1 row(s) affected)
```

## See Also

[ALTER TABLE \(Transact-SQL\)](#)  
[CREATE TABLE \(Transact-SQL\)](#)  
[CURRENT\\_TIMESTAMP \(Transact-SQL\)](#)  
[CURRENT\\_USER \(Transact-SQL\)](#)  
[Security Functions \(Transact-SQL\)](#)  
[SESSION\\_USER \(Transact-SQL\)](#)  
[SYSTEM\\_USER \(Transact-SQL\)](#)  
[USER\\_NAME \(Transact-SQL\)](#)

# USER\_ID (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the identification number for a database user.

## IMPORTANT

This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Use [DATABASE\\_PRINCIPAL\\_ID](#) instead.



## Syntax

```
USER_ID ( [ 'user' ] )
```

## Arguments

*user*

Is the username to be used. *user* is **nchar**. If a **char** value is specified, it is implicitly converted to **nchar**. The parentheses are required.

## Return Types

**int**

## Remarks

When *user* is omitted, the current user is assumed. If the parameter contains the word NULL will return NULL. When USER\_ID is called after EXECUTE AS, USER\_ID will return the ID of the impersonated context.

When a Windows principal that is not mapped to a specific database user accesses a database by way of membership in a group, USER\_ID returns 0 (the ID of public). If such a principal creates an object without specifying a schema, SQL Server will create an implicit user and schema mapped to the Windows principal. The user created in such cases cannot be used to connect to the database. Calls to USER\_ID by a Windows principal mapped to an implicit user will return the ID of the implicit user.

USER\_ID can be used in a select list, in a WHERE clause, and anywhere an expression is allowed. For more information, see [Expressions \(Transact-SQL\)](#).

## Examples

The following example returns the identification number for the `Adventureworks2012` user `Harold`.

```
USE AdventureWorks2012;
SELECT USER_ID('Harold');
GO
```

## See Also

- [USER\\_NAME \(Transact-SQL\)](#)
- [sys.database\\_principals \(Transact-SQL\)](#)
- [DATABASE\\_PRINCIPAL\\_ID \(Transact-SQL\)](#)
- [Security Functions \(Transact-SQL\)](#)

# USER\_NAME (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a database user name from a specified identification number.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
USER_NAME ( [ id ] )
```

## Arguments

*id*

Is the identification number associated with a database user. *id* is **int**. The parentheses are required.

## Return Types

**nvarchar(256)**

## Remarks

When *id* is omitted, the current user in the current context is assumed. If the parameter contains the word NULL will return NULL. When USER\_NAME is called without specifying an *id* after an EXECUTE AS statement, USER\_NAME returns the name of the impersonated user. If a Windows principal accesses the database by way of membership in a group, USER\_NAME returns the name of the Windows principal instead of the group.

## Examples

### A. Using USER\_NAME

The following example returns the user name for user ID 13.

```
SELECT USER_NAME(13);
GO
```

### B. Using USER\_NAME without an ID

The following example finds the name of the current user without specifying an ID.

```
SELECT USER_NAME();
GO
```

Here is the result set for a user that is a member of the sysadmin fixed server role.

```
-----  
dbo
```

```
(1 row(s) affected)
```

### C. Using USER\_NAME in the WHERE clause

The following example finds the row in `sysusers` in which the name is equal to the result of applying the system function `USER_NAME` to user identification number `1`.

```
SELECT name FROM sysusers WHERE name = USER_NAME(1);
GO
```

Here is the result set.

```
name
```

```
-----
```

```
dbo
```

```
(1 row(s) affected)
```

### D. Calling USER\_NAME during impersonation with EXECUTE AS

The following example shows how `USER_NAME` behaves during impersonation.

```
SELECT USER_NAME();
GO
EXECUTE AS USER = 'Zelig';
GO
SELECT USER_NAME();
GO
REVERT;
GO
SELECT USER_NAME();
GO
```

Here is the result set.

```
dbo
```

```
Zelig
```

```
dbo
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### E. Using USER\_NAME

The following example returns the user name for user ID `13`.

```
SELECT USER_NAME(13);
```

### F. Using USER\_NAME without an ID

The following example finds the name of the current user without specifying an ID.

```
SELECT USER_NAME();
```

Here is the result set for a currently logged-in user.

Here is the result set.

User7

## G. Using USER\_NAME in the WHERE clause

The following example finds the row in `sysusers` in which the name is equal to the result of applying the system function `USER_NAME` to user identification number `1`.

```
SELECT name FROM sysusers WHERE name = USER_NAME(1);
```

Here is the result set.

name

User7

## See Also

[ALTER TABLE \(Transact-SQL\)](#)  
[CREATE TABLE \(Transact-SQL\)](#)  
[CURRENT\\_TIMESTAMP \(Transact-SQL\)](#)  
[CURRENT\\_USER \(Transact-SQL\)](#)  
[SESSION\\_USER \(Transact-SQL\)](#)  
[System Functions \(Transact-SQL\)](#)  
[SYSTEM\\_USER \(Transact-SQL\)](#)

# String Functions (Transact-SQL)

8/15/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

The following scalar functions perform an operation on a string input value and return a string or numeric value:

ASCII	CHAR	CHARINDEX
CONCAT	CONCAT_WS	DIFFERENCE
FORMAT	LEFT	LEN
LOWER	LTRIM	NCHAR
PATINDEX	QUOTENAME	REPLACE
REPLICATE	REVERSE	RIGHT
RTRIM	SOUNDEX	SPACE
STR	STRING_AGG	STRING_ESCAPE
STRING_SPLIT	STUFF	SUBSTRING
TRANSLATE	TRIM	UNICODE
UPPER		

All built-in string functions except `FORMAT` are deterministic. This means they return the same value any time they are called with a specific set of input values. For more information about function determinism, see [Deterministic and Nondeterministic Functions](#).

When string functions are passed arguments that are not string values, the input type is implicitly converted to a text data type. For more information, see [Data Type Conversion \(Database Engine\)](#).

## See Also

[Built-in Functions \(Transact-SQL\)](#)

# ASCII (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the ASCII code value of the leftmost character of a character expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ASCII ( character_expression )
```

## Arguments

*character\_expression*

Is an [expression](#) of the type **char** or **varchar**.

## Return types

**int**

## Remarks

ASCII is an abbreviation for American Standard Code for Information Interchange. It is a character encoding standard used by computers. For a list of ASCII characters, see the **Printable characters** section of [ASCII](#).

## Examples

The following example assumes an ASCII character set and returns the `ASCII` value for 6 characters.

```
SELECT ASCII('A') AS A, ASCII('B') AS B,
       ASCII('a') AS a, ASCII('b') AS b,
       ASCII(1) AS [1], ASCII(2) AS [2];
```

Here is the result set.

A	B	a	b	1	2
65	66	97	98	49	50

## See also

[String Functions \(Transact-SQL\)](#)

# CHAR (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Converts an **int** ASCII code to a character.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
CHAR ( integer_expression )
```

## Arguments

*integer\_expression*

Is an integer from 0 through 255. `NULL` is returned if the integer expression is not in this range.

## Return types

**char(1)**

## Remarks

`CHAR` can be used to insert control characters into character strings. The following table shows some frequently used control characters.

CONTROL CHARACTER	VALUE
Tab	<code>char(9)</code>
Line feed	<code>char(10)</code>
Carriage return	<code>char(13)</code>

## Examples

### A. Using ASCII and CHAR to print ASCII values from a string

The following example prints the ASCII value and character for each character in the string `New Moon`.

```

SET TEXTSIZE 0;
-- Create variables for the character string and for the current
-- position in the string.
DECLARE @position int, @string char(8);
-- Initialize the current position and the string variables.
SET @position = 1;
SET @string = 'New Moon';
WHILE @position <= DATALENGTH(@string)
BEGIN
    SELECT ASCII(SUBSTRING(@string, @position, 1)),
           CHAR(ASCII(SUBSTRING(@string, @position, 1)))
    SET @position = @position + 1
END;
GO

```

Here is the result set.

```

----- -
78      N
-----
101     e
-----
119     w
-----
32
-----
77      M
-----
111     o
-----
111     o
-----
110     n

```

## B. Using CHAR to insert a control character

The following example uses `CHAR(13)` to print the name and e-mail address of an employee on separate lines when the results are returned in text. This example uses the AdventureWorks2012 database.

```

SELECT p.FirstName + ' ' + p.LastName, + CHAR(13) + pe.EmailAddress
FROM Person.Person p JOIN Person.EmailAddress pe
ON p.BusinessEntityID = pe.BusinessEntityID
AND p.BusinessEntityID = 1;
GO
```sql

[ !INCLUDE[ssResult](../../includes/ssresult-md.md)]]

`Ken Sanchez`

`ken0@adventure-works.com`

`(1 row(s) affected)

## Examples: [ !INCLUDE[ssSDWfull](../../includes/ssdwfull-md.md)] and [ !INCLUDE[ssPDW](../../includes/sspdw-md.md)]]

### C. Using ASCII and CHAR to print ASCII values from a string

The following example assumes an ASCII character set and returns the character value for 6 ASCII character numbers.

```sql
SELECT CHAR(65) AS [65], CHAR(66) AS [66],
CHAR(97) AS [97], CHAR(98) AS [98],
CHAR(49) AS [49], CHAR(50) AS [50];

```

Here is the result set.

65	66	97	98	49	50
A	B	a	b	1	2

#### D. Using CHAR to insert a control character

The following example uses `CHAR(13)` to return information about the databases on separate lines when the results are returned in text.

```

SELECT name, 'was created on ', create_date, CHAR(13), name, 'is currently ', state_desc
FROM sys.databases;
GO

```

Here is the result set.

name	create_date	name	state_desc
master	was created on 2003-04-08 09:13:36.390		
master		is currently ONLINE	
tempdb	was created on 2014-01-10 17:24:24.023		
tempdb		is currently ONLINE	
AdventureWorksPDW2012	was created on 2014-05-07 09:05:07.083		
AdventureWorksPDW2012		is currently ONLINE	

## See also

[+ \(String Concatenation\) \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

# CHARINDEX (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Searches an expression for another expression and returns its starting position if found.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
CHARINDEX ( expressionToFind , expressionToSearch [ , start_location ] )
```

## Arguments

*expressionToFind*

Is a character [expression](#) that contains the sequence to be found. *expressionToFind* is limited to 8000 characters.

*expressionToSearch*

Is a character expression to be searched.

*start\_location*

Is an **integer** or **bigint** expression at which the search starts. If *start\_location* is not specified, is a negative number, or is 0, the search starts at the beginning of *expressionToSearch*.

## Return types

**bigint** if *expressionToSearch* is of the **varchar(max)**, **nvarchar(max)**, or **varbinary(max)** data types; otherwise, **int**.

## Remarks

If either *expressionToFind* or *expressionToSearch* is of a Unicode data type (**nvarchar** or **nchar**) and the other is not, the other is converted to a Unicode data type. CHARINDEX cannot be used with **text**, **ntext**, and **image** data types.

If either *expressionToFind* or *expressionToSearch* is NULL, CHARINDEX returns NULL.

If *expressionToFind* is not found within *expressionToSearch*, CHARINDEX returns 0.

CHARINDEX performs comparisons based on the collation of the input. To perform a comparison in a specified collation, you can use COLLATE to apply an explicit collation to the input.

The starting position returned is 1-based, not 0-based.

0x0000 (**char(0)**) is an undefined character in Windows collations and cannot be included in CHARINDEX.

## Supplementary Characters (Surrogate Pairs)

When using SC collations, both *start\_location* and the return value count surrogate pairs as one character, not two. For more information, see [Collation and Unicode Support](#).

# Examples

## A. Returning the starting position of an expression

The following example returns the position at which the sequence of characters `bicycle` starts in the `DocumentSummary` column of the `Document` table in the AdventureWorks2012 database.

```
DECLARE @document varchar(64);
SELECT @document = 'Reflectors are vital safety' +
    ' components of your bicycle.';
SELECT CHARINDEX('bicycle', @document);
GO
```

Here is the result set.

```
-----
48
```

## B. Searching from a specific position

The following example uses the optional `start_location` parameter to start looking for `vital` at the fifth character of the `DocumentSummary` column in the AdventureWorks2012 database.

```
DECLARE @document varchar(64);

SELECT @document = 'Reflectors are vital safety' +
    ' components of your bicycle.';
SELECT CHARINDEX('vital', @document, 5);
GO
```

Here is the result set.

```
-----
16
(1 row(s) affected)
```

## C. Searching for a nonexistent expression

The following example shows the result set when `expressionToFind` is not found within `expressionToSearch`.

```
DECLARE @document varchar(64);

SELECT @document = 'Reflectors are vital safety' +
    ' components of your bicycle.';
SELECT CHARINDEX('bike', @document);
GO
```

Here is the result set.

```
-----
0
(1 row(s) affected)
```

## D. Performing a case-sensitive search

The following example performs a case-sensitive search for the string `'TEST'` in `'This is a Test'''`.

```
USE tempdb;
GO
--perform a case sensitive search
SELECT CHARINDEX ( 'TEST',
    'This is a Test'
    COLLATE Latin1_General_CS_AS);
```

Here is the result set.

```
-----
```

```
0
```

The following example performs a case-sensitive search for the string `'Test'` in `'This is a Test'`.

```
USE tempdb;
GO
SELECT CHARINDEX ( 'Test',
    'This is a Test'
    COLLATE Latin1_General_CS_AS);
```

Here is the result set.

```
-----
```

```
13
```

## E. Performing a case-insensitive search

The following example performs a case-insensitive search for the string `'TEST'` in `'This is a Test'`.

```
USE tempdb;
GO
SELECT CHARINDEX ( 'TEST',
    'This is a Test'
    COLLATE Latin1_General_CI_AS);
GO
```

Here is the result set.

```
-----
```

```
13
```

# Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

## F. Searching from the start of a string expression

The following example returns the first location of the `is` string in `This is a string`, starting from position 1 (the first character) in the string.

```
SELECT CHARINDEX('is', 'This is a string');
```

Here is the result set.

```
-----
```

## G. Searching from a position other than the first position

The following example returns the first location of the `is` string in `This is a string`, starting with the fourth position.

```
SELECT CHARINDEX('is', 'This is a string', 4);
```

Here is the result set.

```
-----
```

```
6
```

## H. Results when the string is not found

The following example shows the return value when the `string_pattern` is not found in the searched string.

```
SELECT TOP(1) CHARINDEX('at', 'This is a string') FROM dbo.DimCustomer;
```

Here is the result set.

```
-----
```

```
0
```

## See also

- [String Functions \(Transact-SQL\)](#)
- [+ \(String Concatenation\) \(Transact-SQL\)](#)
- [Collation and Unicode Support](#)

# CONCAT (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2012) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a string that is the result of concatenating two or more string values. (To add a separating value during concatenation, see [CONCAT\\_WS](#).)

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
CONCAT ( string_value1, string_value2 [, string_valueN ] )
```

## Arguments

*string\_value*

A string value to concatenate to the other values.

## Return types

String, the length and type of which depend on the input.

## Remarks

**CONCAT** takes a variable number of string arguments and concatenates them into a single string. It requires a minimum of two input values; otherwise, an error is raised. All arguments are implicitly converted to string types and then concatenated. Null values are implicitly converted to an empty string. If all the arguments are null, an empty string of type **varchar(1)** is returned. The implicit conversion to strings follows the existing rules for data type conversions. For more information about data type conversions, see [CAST and CONVERT \(Transact-SQL\)](#).

The return type depends on the type of the arguments. The following table illustrates the mapping.

INPUT TYPE	OUTPUT TYPE AND LENGTH
If any argument is a SQL-CLR system type, a SQL-CLR UDT, or <code>nvarchar(max)</code>	<b>nvarchar(max)</b>
Otherwise, if any argument is <b>varbinary(max)</b> or <b>varchar(max)</b>	<b>varchar(max)</b> unless one of the parameters is an <b>nvarchar</b> of any length. If so, then the result is <b>nvarchar(max)</b> .
Otherwise, if any argument is <b>nvarchar(&lt;= 4000)</b>	<b>nvarchar(&lt;= 4000)</b>
Otherwise, in all other cases	<b>varchar(&lt;= 8000)</b> unless one of the parameters is an <b>nvarchar</b> of any length. If so, then the result is <b>nvarchar(max)</b> .

When the arguments are  $\leq 4000$  for **nvarchar**, or  $\leq 8000$  for **varchar**, implicit conversions can affect the length of the result. Other data types have different lengths when they are implicitly converted to strings. For example, an **int (14)** has a string length of 12, while a **float** has a length of 32. Thus the result of concatenating two

integers has a length of no less than 24.

If none of the input arguments is of a supported large object (LOB) type, then the return type is truncated to 8000 in length, regardless of the return type. This truncation preserves space and supports efficiency in plan generation.

The CONCAT function can be executed remotely on a linked server which is version SQL Server 2012 and above. For older linked servers, the CONCAT operation will be performed locally after the non-concatenated values are returned from the linked server.

## Examples

### A. Using CONCAT

```
SELECT CONCAT ( 'Happy ', 'Birthday ', 11, '/', '25' ) AS Result;
```

Here is the result set.

```
Result
-----
Happy Birthday 11/25

(1 row(s) affected)
```

### B. Using CONCAT with NULL values

```
CREATE TABLE #temp (
    emp_name nvarchar(200) NOT NULL,
    emp_middlename nvarchar(200) NULL,
    emp_lastname nvarchar(200) NOT NULL
);
INSERT INTO #temp VALUES( 'Name', NULL, 'Lastname' );
SELECT CONCAT( emp_name, emp_middlename, emp_lastname ) AS Result
FROM #temp;
```

Here is the result set.

```
Result
-----
NameLastname

(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Using CONCAT

```
SELECT CONCAT ( 'Happy ', 'Birthday ', 11, '/', '25' ) AS Result;
```

Here is the result set.

```
Result
-----
Happy Birthday 11/25

(1 row(s) affected)
```

## See also

[String Functions \(Transact-SQL\)](#)

[CONCAT\\_WS \(Transact-SQL\)](#)

# CONCAT\_WS (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2017) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Concatenates a variable number of arguments with a delimiter specified in the 1st argument. (`CONCAT_WS` indicates *concatenate with separator*.)

## Syntax

```
CONCAT_WS ( separator, argument1, argument1 [, argumentN]... )
```

## Arguments

### separator

Is an expression of any character type (`nvarchar`, `varchar`, `nchar`, or `char`).

### argument1, argument2, argumentN

Is an expression of any type.

## Return types

String. The length and type depend on the input.

## Remarks

`CONCAT_WS` takes a variable number of arguments and concatenates them into a single string using the first argument as separator. It requires a separator and a minimum of two arguments; otherwise, an error is raised. All arguments are implicitly converted to string types and are then concatenated.

The implicit conversion to strings follows the existing rules for data type conversions. For more information about behavior and data type conversions, see [CONCAT \(Transact-SQL\)](#).

### Treatment of NULL values

`CONCAT_WS` ignores the `SET CONCAT_NULL_YIELDS_NULL {ON|OFF}` setting.

If all the arguments are null, an empty string of type `varchar(1)` is returned.

Null values are ignored during concatenation, and does not add the separator. This facilitates the common scenario of concatenating strings which often have blank values, such as a second address field. See example B.

If your scenario requires null values to be included with a separator, see example C using the `ISNULL` function.

## Examples

### A. Concatenating values with separator

The following example concatenates three columns from the `sys.databases` table, separating the values with a `-`.

```
SELECT CONCAT_WS( ' - ', database_id, recovery_model_desc, containment_desc) AS DatabaseInfo
FROM sys.databases;
```

Here is the result set.

#### DATABASEINFO

1 - SIMPLE - NONE
2 - SIMPLE - NONE
3 - FULL - NONE
4 - SIMPLE - NONE

### B. Skipping NULL values

The following example ignores `NULL` values in the arguments list.

```
SELECT CONCAT_WS(',', '1 Microsoft Way', NULL, NULL, 'Redmond', 'WA', 98052) AS Address;
```

Here is the result set.

Address
-----
1 Microsoft Way, Redmond, WA, 98052

### C. Generating CSV file from table

The following example uses a comma as the separator and adds the carriage return character to result in the column separated values format.

```
SELECT
STRING_AGG(CONCAT_WS( ',', database_id, recovery_model_desc, containment_desc), char(13)) AS DatabaseInfo
FROM sys.databases
```

Here is the result set.

DatabaseInfo
-----
1,SIMPLE,NONE
2,SIMPLE,NONE
3,FULL,NONE
4,SIMPLE,NONE

`CONCAT_WS` will ignore `NULL` values in the columns. If some of the columns are nullable, wrap it with `ISNULL` function and provide default value like in the following example:

```
SELECT
STRING_AGG(CONCAT_WS( ',', database_id, ISNULL(recovery_model_desc,''), ISNULL(containment_desc,'N/A')), char(13)) AS DatabaseInfo
FROM sys.databases;
```

## See also

[String Functions \(Transact-SQL\)](#)

[CONCAT \(Transact-SQL\)](#)

# DIFFERENCE (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns an integer value that indicates the difference between the SOUNDEX values of two character expressions.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
DIFFERENCE ( character_expression , character_expression )
```

## Arguments

*character\_expression*

Is an alphanumeric [expression](#) of character data. *character\_expression* can be a constant, variable, or column.

## Return Types

**int**

## Remarks

The integer returned is the number of characters in the SOUNDEX values that are the same. The return value ranges from 0 through 4: 0 indicates weak or no similarity, and 4 indicates strong similarity or the same values.

DIFFERENCE and SOUNDEX are collation sensitive.

## Examples

In the first part of the following example, the `SOUNDEX` values of two very similar strings are compared. For a Latin1\_General collation `DIFFERENCE` returns a value of `4`. In the second part of the following example, the `SOUNDEX` values for two very different strings are compared, and for a Latin1\_General collation `DIFFERENCE` returns a value of `0`.

```
-- Returns a DIFFERENCE value of 4, the least possible difference.  
SELECT SOUNDEX('Green'), SOUNDEX('Greene'), DIFFERENCE('Green','Greene');  
GO  
-- Returns a DIFFERENCE value of 0, the highest possible difference.  
SELECT SOUNDEX('Blotchet-Halls'), SOUNDEX('Greene'), DIFFERENCE('Blotchet-Halls', 'Greene');  
GO
```

Here is the result set.

```
-----  
G650  G650  4  
  
(1 row(s) affected)  
  
-----  
B432  G650  0  
  
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

In the first part of the following example, the `SOUNDEX` values of two very similar strings are compared. For a Latin1\_General collation `DIFFERENCE` returns a value of `4`. In the second part of the following example, the `SOUNDEX` values for two very different strings are compared, and for a Latin1\_General collation `DIFFERENCE` returns a value of `0`.

```
-- Returns a DIFFERENCE value of 4, the least possible difference.  
SELECT SOUNDEX('Green'), SOUNDEX('Greene'), DIFFERENCE('Green','Greene');  
GO  
-- Returns a DIFFERENCE value of 0, the highest possible difference.  
SELECT SOUNDEX('Blotchet-Halls'), SOUNDEX('Greene'), DIFFERENCE('Blotchet-Halls', 'Greene');  
GO
```

Here is the result set.

```
-----  
G650  G650  4  
  
(1 row(s) affected)  
  
-----  
B432  G650  0  
  
(1 row(s) affected)
```

## See Also

[SOUNDEX \(Transact-SQL\)](#)  
[String Functions \(Transact-SQL\)](#)

# FORMAT (Transact-SQL)

8/15/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a value formatted with the specified format and optional culture in SQL Server 2017. Use the FORMAT function for locale-aware formatting of date/time and number values as strings. For general data type conversions, use CAST or CONVERT.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
FORMAT ( value, format [, culture ] )
```

## Arguments

### *value*

Expression of a supported data type to format. For a list of valid types, see the table in the following Remarks section.

### *format*

**nvarchar** format pattern.

The *format* argument must contain a valid .NET Framework format string, either as a standard format string (for example, "C" or "D"), or as a pattern of custom characters for dates and numeric values (for example, "MMMM DD, yyyy (dddd)"). Composite formatting is not supported. For a full explanation of these formatting patterns, consult the .NET Framework documentation on string formatting in general, custom date and time formats, and custom number formats. A good starting point is the topic, "[Formatting Types](#)."

### *culture*

Optional **nvarchar** argument specifying a culture.

If the *culture* argument is not provided, the language of the current session is used. This language is set either implicitly, or explicitly by using the SET LANGUAGE statement. *culture* accepts any culture supported by the .NET Framework as an argument; it is not limited to the languages explicitly supported by SQL Server. If the *culture* argument is not valid, FORMAT raises an error.

## Return Types

**nvarchar** or null

The length of the return value is determined by the *format*.

## Remarks

FORMAT returns NULL for errors other than a *culture* that is not *valid*. For example, NULL is returned if the value specified in *format* is not valid.

The FORMAT function is nondeterministic.

FORMAT relies on the presence of the .NET Framework Common Language Runtime (CLR).

This function cannot be remoted since it depends on the presence of the CLR. Remoting a function that requires the CLR, could cause an error on the remote server.

FORMAT relies upon CLR formatting rules, which dictate that colons and periods must be escaped. Therefore, when the format string (second parameter) contains a colon or period, the colon or period must be escaped with backslash when an input value (first parameter) is of the **time** data type. See [D. FORMAT with time data types](#).

The following table lists the acceptable data types for the *value* argument together with their .NET Framework mapping equivalent types.

CATEGORY	TYPE	.NET TYPE
Numeric	bigint	Int64
Numeric	int	Int32
Numeric	smallint	Int16
Numeric	tinyint	Byte
Numeric	decimal	SqlDecimal
Numeric	numeric	SqlDecimal
Numeric	float	Double
Numeric	real	Single
Numeric	smallmoney	Decimal
Numeric	money	Decimal
Date and Time	date	DateTime
Date and Time	time	TimeSpan
Date and Time	datetime	DateTime
Date and Time	smalldatetime	DateTime
Date and Time	datetime2	DateTime
Date and Time	datetimeoffset	DateTimeOffset

## Examples

### A. Simple FORMAT example

The following example returns a simple date formatted for different cultures.

```

DECLARE @d DATETIME = '10/01/2011';
SELECT FORMAT ( @d, 'd', 'en-US' ) AS 'US English Result'
    ,FORMAT ( @d, 'd', 'en-gb' ) AS 'Great Britain English Result'
    ,FORMAT ( @d, 'd', 'de-de' ) AS 'German Result'
    ,FORMAT ( @d, 'd', 'zh-cn' ) AS 'Simplified Chinese (PRC) Result';

SELECT FORMAT ( @d, 'D', 'en-US' ) AS 'US English Result'
    ,FORMAT ( @d, 'D', 'en-gb' ) AS 'Great Britain English Result'
    ,FORMAT ( @d, 'D', 'de-de' ) AS 'German Result'
    ,FORMAT ( @d, 'D', 'zh-cn' ) AS 'Chinese (Simplified PRC) Result';

```

Here is the result set.

US English Result	Great Britain English Result	German Result	Simplified Chinese (PRC) Result
10/1/2011	01/10/2011	01.10.2011	2011/10/1

(1 row(s) affected)

US English Result (Simplified PRC) Result	Great Britain English Result	German Result	Chinese
Saturday, October 01, 2011	01 October 2011	Samstag, 1. Oktober 2011	2011年10月1日

(1 row(s) affected)

## B. FORMAT with custom formatting strings

The following example shows formatting numeric values by specifying a custom format. The example assumes that the current date is September 27, 2012. For more information about these and other custom formats, see [Custom Numeric Format Strings](#).

```

DECLARE @d DATETIME = GETDATE();
SELECT FORMAT( @d, 'dd/MM/yyyy', 'en-US' ) AS 'DateTime Result'
    ,FORMAT(123456789,'###-##-####') AS 'Custom Number Result';

```

Here is the result set.

DateTime Result	Custom Number Result
27/09/2012	123-45-6789

(1 row(s) affected)

## C. FORMAT with numeric types

The following example returns 5 rows from the **Sales.CurrencyRate** table in the AdventureWorks2012 database. The column **EndOfDayRate** is stored as type **money** in the table. In this example, the column is returned unformatted and then formatted by specifying the .NET Number format, General format, and Currency format types. For more information about these and other numeric formats, see [Standard Numeric Format Strings](#).

```

SELECT TOP(5)CurrencyRateID, EndOfDayRate
    ,FORMAT(EndOfDayRate, 'N', 'en-us') AS 'Number Format'
    ,FORMAT(EndOfDayRate, 'G', 'en-us') AS 'General Format'
    ,FORMAT(EndOfDayRate, 'C', 'en-us') AS 'Currency Format'
FROM Sales.CurrencyRate
ORDER BY CurrencyRateID;

```

Here is the result set.

CurrencyRateID	EndOfDayRate	Numeric Format	General Format	Currency Format
1	1.0002	1.00	1.0002	\$1.00
2	1.55	1.55	1.5500	\$1.55
3	1.9419	1.94	1.9419	\$1.94
4	1.4683	1.47	1.4683	\$1.47
5	8.2784	8.28	8.2784	\$8.28

(5 row(s) affected)

This example specifies the German culture (de-de).

```
SELECT TOP(5)CurrencyRateID, EndOfDayRate
    ,FORMAT(EndOfDayRate, 'N', 'de-de') AS 'Numeric Format'
    ,FORMAT(EndOfDayRate, 'G', 'de-de') AS 'General Format'
    ,FORMAT(EndOfDayRate, 'C', 'de-de') AS 'Currency Format'
FROM Sales.CurrencyRate
ORDER BY CurrencyRateID;
```

CurrencyRateID	EndOfDayRate	Numeric Format	General Format	Currency Format
1	1.0002	1,00	1,0002	1,00 €
2	1.55	1,55	1,5500	1,55 €
3	1.9419	1,94	1,9419	1,94 €
4	1.4683	1,47	1,4683	1,47 €
5	8.2784	8,28	8,2784	8,28 €

(5 row(s) affected)

## D. FORMAT with time data types

FORMAT returns NULL in these cases because [.] and [:] are not escaped.

```
SELECT FORMAT(cast('07:35' as time), N'hh.mm'); --> returns NULL
SELECT FORMAT(cast('07:35' as time), N'hh:mm'); --> returns NULL
```

Format returns a formatted string because the [.] and [:] are escaped.

```
SELECT FORMAT(cast('07:35' as time), N'hh\.mm'); --> returns 07.35
SELECT FORMAT(cast('07:35' as time), N'hh\::mm'); --> returns 07:35
```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

# LEFT (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns the left part of a character string with the specified number of characters.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
LEFT ( character_expression , integer_expression )
```

## Arguments

*character\_expression*

Is an [expression](#) of character or binary data. *character\_expression* can be a constant, variable, or column. *character\_expression* can be of any data type, except **text** or **ntext**, that can be implicitly converted to **varchar** or **nvarchar**. Otherwise, use the [CAST](#) function to explicitly convert *character\_expression*.

*integer\_expression*

Is a positive integer that specifies how many characters of the *character\_expression* will be returned. If *integer\_expression* is negative, an error is returned. If *integer\_expression* is type **bigint** and contains a large value, *character\_expression* must be of a large data type such as **varchar(max)**.

The *integer\_expression* parameter counts a UTF-16 surrogate character as one character.

## Return Types

Returns **varchar** when *character\_expression* is a non-Unicode character data type.

Returns **nvarchar** when *character\_expression* is a Unicode character data type.

## Remarks

When using SC collations, the *integer\_expression* parameter counts a UTF-16 surrogate pair as one character. For more information, see [Collation and Unicode Support](#).

## Examples

### A. Using LEFT with a column

The following example returns the five leftmost characters of each product name in the `Product` table of the AdventureWorks2012 database.

```
SELECT LEFT(Name, 5)
FROM Production.Product
ORDER BY ProductID;
GO
```

### B. Using LEFT with a character string

The following example uses `LEFT` to return the two leftmost characters of the character string `abcdefg`.

```
SELECT LEFT('abcdefg',2);
GO
```

Here is the result set.

```
--  
ab  
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Using LEFT with a column

The following example returns the five leftmost characters of each product name.

```
-- Uses AdventureWorks

SELECT LEFT(EnglishProductName, 5)
FROM dbo.DimProduct
ORDER BY ProductKey;
```

### D. Using LEFT with a character string

The following example uses `LEFT` to return the two leftmost characters of the character string `abcdefg`.

```
-- Uses AdventureWorks

SELECT LEFT('abcdefg',2) FROM dbo.DimProduct;
```

Here is the result set.

```
--  
ab
```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

# LEN (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the number of characters of the specified string expression, excluding trailing blanks.

## NOTE

To return the number of bytes used to represent an expression, use the [DATALENGTH](#) function.



## Syntax

```
LEN ( string_expression )
```

## Arguments

*string\_expression*

Is the string [expression](#) to be evaluated. *string\_expression* can be a constant, variable, or column of either character or binary data.

## Return Types

**bigint** if *expression* is of the **varchar(max)**, **nvarchar(max)** or **varbinary(max)** data types; otherwise, **int**.

If you are using SC collations, the returned integer value counts UTF-16 surrogate pairs as a single character. For more information, see [Collation and Unicode Support](#).

## Remarks

LEN excludes trailing blanks. If that is a problem, consider using the [DATALENGTH \(Transact-SQL\)](#) function which does not trim the string. If processing a unicode string, DATALENGTH will return twice the number of characters. The following example demonstrates LEN and DATALENGTH with a trailing space.

```
DECLARE @v1 varchar(40),
       @v2 nvarchar(40);
SELECT
    @v1 = 'Test of 22 characters ',
    @v2 = 'Test of 22 characters ';
SELECT LEN(@v1) AS [varchar LEN] , DATALENGTH(@v1) AS [varchar DATALENGTH];
SELECT LEN(@v2) AS [nvarchar LEN], DATALENGTH(@v2) AS [nvarchar DATALENGTH];
```

## Examples

The following example selects the number of characters and the data in `FirstName` for people located in `Australia`. This example uses the AdventureWorks2012 database.

```
SELECT LEN(FirstName) AS Length, FirstName, LastName
FROM Sales.vIndividualCustomer
WHERE CountryRegionName = 'Australia';
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns the number of characters in the column `FirstName` and the first and last names of employees located in `Australia`.

```
-- Uses AdventureWorks

SELECT DISTINCT LEN(FirstName) AS FNameLength, FirstName, LastName
FROM dbo.DimEmployee AS e
INNER JOIN dbo.DimGeography AS g
    ON e.SalesTerritoryKey = g.SalesTerritoryKey
WHERE EnglishCountryRegionName = 'Australia';
```

Here is the result set.

FNameLength	FirstName	LastName
-------------	-----------	----------

4	Lynn	Tsoflias
---	------	----------

## See Also

[Data Types \(Transact-SQL\)](#)  
[String Functions \(Transact-SQL\)](#)  
[DATALENGTH \(Transact-SQL\)](#)  
[LEFT \(Transact-SQL\)](#)  
[RIGHT \(Transact-SQL\)](#)

# LOWER (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a character expression after converting uppercase character data to lowercase.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
LOWER ( character_expression )
```

## Arguments

*character\_expression*

Is an [expression](#) of character or binary data. *character\_expression* can be a constant, variable, or column. *character\_expression* must be of a data type that is implicitly convertible to **varchar**. Otherwise, use [CAST](#) to explicitly convert *character\_expression*.

## Return Types

**varchar** or **nvarchar**

## Examples

The following example uses the `LOWER` function, the `UPPER` function, and nests the `UPPER` function inside the `LOWER` function in selecting product names that have prices between \$11 and \$20. This example uses the AdventureWorks2012 database.

```
SELECT LOWER(SUBSTRING(Name, 1, 20)) AS Lower,
       UPPER(SUBSTRING(Name, 1, 20)) AS Upper,
       LOWER(UPPER(SUBSTRING(Name, 1, 20))) AS LowerUpper
  FROM Production.Product
 WHERE ListPrice between 11.00 and 20.00;
 GO
```

Here is the result set.

Lower	Upper	LowerUpper
-----	-----	-----

minipump MINIPUMP minipump
----------------------------

taillights - battery TAILLIGHTS - BATTERY taillights - battery
--

(2 row(s) affected)
---------------------

**Examples: Azure SQL Data Warehouse and Parallel Data Warehouse**

The following example uses the `LOWER` function, the `UPPER` function, and nests the `UPPER` function inside the `LOWER` function in selecting product names that have prices between \$11 and \$20.

```
-- Uses AdventureWorks

SELECT LOWER(SUBSTRING(EnglishProductName, 1, 20)) AS Lower,
       UPPER(SUBSTRING(EnglishProductName, 1, 20)) AS Upper,
       LOWER(UPPER(SUBSTRING(EnglishProductName, 1, 20))) As LowerUpper
  FROM dbo.DimProduct
 WHERE ListPrice between 11.00 and 20.00;
```

Here is the result set.

Lower	Upper	LowerUpper
-----	-----	-----
minipump	MINIPUMP	minipump
taillights - battery	TAILLIGHTS - BATTERY	taillights - battery

## See Also

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

# LTRIM (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns a character expression after it removes leading blanks.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
LTRIM ( character_expression )
```

## Arguments

*character\_expression*

Is an [expression](#) of character or binary data. *character\_expression* can be a constant, variable, or column. *character\_expression* must be of a data type, except **text**, **ntext**, and **image**, that is implicitly convertible to **varchar**. Otherwise, use [CAST](#) to explicitly convert *character\_expression*.

## Return Type

**varchar** or **nvarchar**

## Examples

### A. Simple example

The following example uses LTRIM to remove leading spaces from a character expression.

```
SELECT LTRIM('      Five spaces are at the beginning of this string.') FROM sys.databases;
```

Here is the result set.

```
-----  
Five spaces are at the beginning of this string.
```

### B: Example using a variable

The following example uses `LTRIM` to remove leading spaces from a character variable.

```
DECLARE @string_to_trim varchar(60);
SET @string_to_trim = '      5 spaces are at the beginning of this string.';
SELECT
    @string_to_trim AS 'Original string',
    LTRIM(@string_to_trim) AS 'Without spaces';
GO
```

Here is the result set.

Original string Without spaces

---

5 spaces are at the beginning of this string. 5 spaces are at the beginning of this string.

## See Also

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

# NCHAR (Transact-SQL)

9/27/2017 • 6 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the Unicode character with the specified integer code, as defined by the Unicode standard.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
NCHAR ( integer_expression )
```

## Arguments

*integer\_expression*

When the collation of the database does not contain the supplementary character (SC) flag, this is a positive whole number from 0 through 65535 (0 through 0xFFFF). If a value outside this range is specified, NULL is returned. For more information about supplementary characters, see [Collation and Unicode Support](#).

When the collation of the database supports the supplementary character (SC) flag, this is a positive whole number from 0 through 1114111 (0 through 0x10FFFF). If a value outside this range is specified, NULL is returned.

## Return Types

**nchar(1)** when the default database collation does not support supplementary characters.

**nvarchar(2)** when the default database collation supports supplementary characters.

If the parameter *integer\_expression* lies in the range 0 - 0xFFFF, only one character is returned. For higher values, NCHAR returns the corresponding surrogate pair. Do not construct a surrogate pair by using

`NCHAR(<High surrogate>) + NCHAR(<Low Surrogate>)`. Instead, use a database collation that supports supplementary characters and then specify the Unicode codepoint for the surrogate pair. The following example demonstrates both the old style method of constructing a surrogate pair and the preferred method of specifying the Unicode codepoint.

```
CREATE DATABASE test COLLATE Finnish_Swedish_100_CS_AS_SC;
DECLARE @d nvarchar(10) = N'𩶻';
-- Old style method.
SELECT NCHAR(0xD84C) + NCHAR(0xDD7F);

-- Preferred method.
SELECT NCHAR(143743);

-- Alternative preferred method.
SELECT NCHAR(UNICODE(@d));
```

## Examples

### A. Using NCHAR and UNICODE

The following example uses the `UNICODE` and `NCHAR` functions to print the `UNICODE` value and the `NCHAR` (Unicode character) of the second character of the `København` character string, and to print the actual second character, `Ø`.

```
DECLARE @nstring nchar(8);
SET @nstring = N'København';
SELECT UNICODE(SUBSTRING(@nstring, 2, 1)),
       NCHAR(UNICODE(SUBSTRING(@nstring, 2, 1)));
GO
```

Here is the result set.

```
----- -
248      Ø
(1 row(s) affected)
```

## B. Using SUBSTRING, UNICODE, CONVERT, and NCHAR

The following example uses the `SUBSTRING`, `UNICODE`, `CONVERT`, and `NCHAR` functions to print the character number, the Unicode character, and the `UNICODE` value of each character in the string `København`.

```
-- The @position variable holds the position of the character currently
-- being processed. The @nstring variable is the Unicode character
-- string to process.
DECLARE @position int, @nstring nchar(9);
-- Initialize the current position variable to the first character in
-- the string.
SET @position = 1;
-- Initialize the character string variable to the string to process.
-- Notice that there is an N before the start of the string. This
-- indicates that the data following the N is Unicode data.
SET @nstring = N'København';
-- Print the character number of the position of the string you are at,
-- the actual Unicode character you are processing, and the UNICODE
-- value for this particular character.
PRINT 'Character #' + ' ' + 'Unicode Character' + ' ' + 'UNICODE Value';
WHILE @position <= DATALENGTH(@nstring)
    BEGIN
        SELECT @position,
               NCHAR(UNICODE(SUBSTRING(@nstring, @position, 1))),
               CONVERT(NCHAR(17), SUBSTRING(@nstring, @position, 1)),
               UNICODE(SUBSTRING(@nstring, @position, 1))
        SELECT @position = @position + 1
    END;
GO
```

Here is the result set.

	Character #	Unicode Character	UNICODE Value
1	K	K	75
	(1 row(s) affected)		
2	Ø	Ø	248
	(1 row(s) affected)		
3	b	b	98

(1 row(s) affected)

-----  
4 e e 101

(1 row(s) affected)

-----  
5 n n 110

(1 row(s) affected)

-----  
6 h h 104

(1 row(s) affected)

-----  
7 a a 97

(1 row(s) affected)

-----  
8 v v 118

(1 row(s) affected)

-----  
9 n n 110

(1 row(s) affected)

-----  
10 NULL NULL

(1 row(s) affected)

-----  
11 NULL NULL

(1 row(s) affected)

-----  
12 NULL NULL

(1 row(s) affected)

-----  
13 NULL NULL

(1 row(s) affected)

-----  
14 NULL NULL

(1 row(s) affected)

-----  
15 NULL NULL

(1 row(s) affected)

-----  
16 NULL NULL

(1 row(s) affected)

-----  
17 NULL NULL

```

(1 row(s) affected)

-----
18      NULL          NULL

(1 row(s) affected)

```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Using NCHAR and UNICODE

The following example uses the `UNICODE` and `NCHAR` functions to print the `UNICODE` value and the `NCHAR` (Unicode character) of the second character of the `København` character string, and to print the actual second character, `ø`.

```

DECLARE @nstring nchar(8);
SET @nstring = N'København';
SELECT UNICODE(SUBSTRING(@nstring, 2, 1)),
       NCHAR(UNICODE(SUBSTRING(@nstring, 2, 1)));
GO

```

Here is the result set.

```

-----
248      ø
(1 row(s) affected)

```

### D. Using SUBSTRING, UNICODE, CONVERT, and NCHAR

The following example uses the `SUBSTRING`, `UNICODE`, `CONVERT`, and `NCHAR` functions to print the character number, the Unicode character, and the `UNICODE` value of each character in the string `København`.

```

-- The @position variable holds the position of the character currently
-- being processed. The @nstring variable is the Unicode character
-- string to process.
DECLARE @position int, @nstring nchar(9);
-- Initialize the current position variable to the first character in
-- the string.
SET @position = 1;
-- Initialize the character string variable to the string to process.
-- Notice that there is an N before the start of the string. This
-- indicates that the data following the N is Unicode data.
SET @nstring = N'København';
-- Print the character number of the position of the string you are at,
-- the actual Unicode character you are processing, and the UNICODE
-- value for this particular character.
PRINT 'Character #' + ' ' + 'Unicode Character' + ' ' + 'UNICODE Value';
WHILE @position <= DATALENGTH(@nstring)
BEGIN
    SELECT @position,
           NCHAR(UNICODE(SUBSTRING(@nstring, @position, 1))),
           CONVERT(NCHAR(17), SUBSTRING(@nstring, @position, 1)),
           UNICODE(SUBSTRING(@nstring, @position, 1))
    SELECT @position = @position + 1
END;
GO

```

Here is the result set.

Character #	Unicode Character	UNICODE Value

1	K	K	75
(1 row(s) affected)			
2	∅	∅	248
(1 row(s) affected)			
3	b	b	98
(1 row(s) affected)			
4	e	e	101
(1 row(s) affected)			
5	n	n	110
(1 row(s) affected)			
6	h	h	104
(1 row(s) affected)			
7	a	a	97
(1 row(s) affected)			
8	v	v	118
(1 row(s) affected)			
9	n	n	110
(1 row(s) affected)			
10	NULL	NULL	NULL
(1 row(s) affected)			
11	NULL	NULL	NULL
(1 row(s) affected)			
12	NULL	NULL	NULL
(1 row(s) affected)			
13	NULL	NULL	NULL
(1 row(s) affected)			
14	NULL	NULL	NULL

```
(1 row(s) affected)

-----
15      NULL          NULL

(1 row(s) affected)

-----
16      NULL          NULL

(1 row(s) affected)

-----
17      NULL          NULL

(1 row(s) affected)

-----
18      NULL          NULL

(1 row(s) affected)
```

## See Also

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

[UNICODE \(Transact-SQL\)](#)

# PATINDEX (Transact-SQL)

7/19/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the starting position of the first occurrence of a pattern in a specified expression, or zeros if the pattern is not found, on all valid text and character data types.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
PATINDEX ( '%pattern%' , expression )
```

## Arguments

### *pattern*

Is a character expression that contains the sequence to be found. Wildcard characters can be used; however, the % character must come before and follow *pattern* (except when you search for first or last characters). *pattern* is an expression of the character string data type category. *pattern* is limited to 8000 characters.

### *expression*

Is an [expression](#), typically a column that is searched for the specified pattern. *expression* is of the character string data type category.

## Return Types

**bigint** if *expression* is of the **varchar(max)** or **nvarchar(max)** data types; otherwise **int**.

## Remarks

If either *pattern* or *expression* is NULL, PATINDEX returns NULL.

PATINDEX performs comparisons based on the collation of the input. To perform a comparison in a specified collation, you can use COLLATE to apply an explicit collation to the input.

## Supplementary Characters (Surrogate Pairs)

When using SC collations, the return value will count any UTF-16 surrogate pairs in the *expression* parameter as a single character. For more information, see [Collation and Unicode Support](#).

0x0000 (**char(0)**) is an undefined character in Windows collations and cannot be included in PATINDEX.

## Examples

### A. Simple PATINDEX example

The following example checks a short character string (`interesting data`) for the starting location of the characters `ter`.

```
SELECT PATINDEX('%ter%', 'interesting data');
```

Here is the result set.

3

## B. Using a pattern with PATINDEX

The following example finds the position at which the pattern `ensure` starts in a specific row of the `DocumentSummary` column in the `Document` table in the AdventureWorks2012 database.

```
SELECT PATINDEX('%ensure%', DocumentSummary)
FROM Production.Document
WHERE DocumentNode = 0x7B40;
GO
```

Here is the result set.

```
-----
64
(1 row(s) affected)
```

If you do not restrict the rows to be searched by using a `WHERE` clause, the query returns all rows in the table and reports nonzero values for those rows in which the pattern was found, and zero for all rows in which the pattern was not found.

## C. Using wildcard characters with PATINDEX

The following example uses % and \_ wildcards to find the position at which the pattern `'en'`, followed by any one character and `'ure'` starts in the specified string (index starts at 1):

```
SELECT PATINDEX('%en_ure%', 'please ensure the door is locked');
```

Here is the result set.

```
-----
8
```

`PATINDEX` works just like `LIKE`, so you can use any of the wildcards. You do not have to enclose the pattern between percents. `PATINDEX('a%', 'abc')` returns 1 and `PATINDEX('%a', 'cba')` returns 3.

Unlike `LIKE`, `PATINDEX` returns a position, similar to what `CHARINDEX` does.

## D. Using COLLATE with PATINDEX

The following example uses the `COLLATE` function to explicitly specify the collation of the expression that is searched.

```
USE tempdb;
GO
SELECT PATINDEX ( '%ein%', 'Das ist ein Test' COLLATE Latin1_General_BIN) ;
GO
```

## E. Using a variable to specify the pattern

The following example uses a variable to pass a value to the `pattern` parameter. This example uses the

AdventureWorks2012 database.

```
DECLARE @MyValue varchar(10) = 'safety';
SELECT PATINDEX('%' + @MyValue + '%', DocumentSummary)
FROM Production.Document
WHERE DocumentNode = 0x7B40;
```

Here is the result set.

```
-----
```

```
22
```

## See Also

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

[\(Wildcard - Character\(s\) to Match\) \(Transact-SQL\)](#)

[\(Wildcard - Character\(s\) Not to Match\) \(Transact-SQL\)](#)

[\\_ \(Wildcard - Match One Character\) \(Transact-SQL\)](#)

[Percent character \(Wildcard - Character\(s\) to Match\) \(Transact-SQL\)](#)

# QUOTENAME (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns a Unicode string with the delimiters added to make the input string a valid SQL Server delimited identifier.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
QUOTENAME ( 'character_string' [ , 'quote_character' ] )
```

## Arguments

*'character\_string'*

Is a string of Unicode character data. *character\_string* is **sysname** and is limited to 128 characters. Inputs greater than 128 characters return NULL.

*'quote\_character'*

Is a one-character string to use as the delimiter. Can be a single quotation mark ( ' ), a left or right bracket ( [ ] ), or a double quotation mark ( " ). If *quote\_character* is not specified, brackets are used.

## Return Types

**nvarchar(258)**

## Examples

The following example takes the character string `abc[]def` and uses the `[` and `]` characters to create a valid SQL Server delimited identifier.

```
SELECT QUOTENAME('abc[]def');
```

Here is the result set.

```
[abc[]]def  
(1 row(s) affected)
```

Notice that the right bracket in the string `abc[]def` is doubled to indicate an escape character.

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example takes the character string `abc def` and uses the `[` and `]` characters to create a valid SQL Server delimited identifier.

```
SELECT QUOTENAME('abc def');
```

Here is the result set.

```
[abc def]  
(1 row(s) affected)
```

## See Also

[String Functions \(Transact-SQL\)](#)

# REPLACE (Transact-SQL)

8/23/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Replaces all occurrences of a specified string value with another string value.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
REPLACE ( string_expression , string_pattern , string_replacement )
```

## Arguments

*string\_expression*

Is the string [expression](#) to be searched. *string\_expression* can be of a character or binary data type.

*string\_pattern*

Is the substring to be found. *string\_pattern* can be of a character or binary data type. *string\_pattern* cannot be an empty string (''), and must not exceed the maximum number of bytes that fits on a page.

*string\_replacement*

Is the replacement string. *string\_replacement* can be of a character or binary data type.

## Return Types

Returns **nvarchar** if one of the input arguments is of the **nvarchar** data type; otherwise, REPLACE returns **varchar**.

Returns NULL if any one of the arguments is NULL.

If *string\_expression* is not of type **varchar(max)** or **nvarchar(max)**, REPLACE truncates the return value at 8,000 bytes. To return values greater than 8,000 bytes, *string\_expression* must be explicitly cast to a large-value data type.

## Remarks

REPLACE performs comparisons based on the collation of the input. To perform a comparison in a specified collation, you can use [COLLATE](#) to apply an explicit collation to the input.

0x0000 (**char(0)**) is an undefined character in Windows collations and cannot be included in REPLACE.

## Examples

The following example replaces the string `cde` in `abcdefghijklm` with `xxx`.

```
SELECT REPLACE('abcdefghijklm','cde','xxx');
GO
```

Here is the result set.

```
-----  
abxxxfgħixxx  
(1 row(s) affected)
```

The following example uses the `COLLATE` function.

```
SELECT REPLACE('This is a Test' COLLATE Latin1_General_BIN,  
'Test', 'desk' );  
GO
```

Here is the result set.

```
-----  
This is a desk  
(1 row(s) affected)
```

## See Also

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

# REPLICATE (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Repeats a string value a specified number of times.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
REPLICATE ( string_expression ,integer_expression )
```

## Arguments

*string\_expression*

Is an expression of a character string or binary data type. *string\_expression* can be either character or binary data.

### NOTE

If *string\_expression* is not of type **varchar(max)** or **nvarchar(max)**, REPLICATE truncates the return value at 8,000 bytes.

To return values greater than 8,000 bytes, *string\_expression* must be explicitly cast to the appropriate large-value data type.

*integer\_expression*

Is an expression of any integer type, including **bigint**. If *integer\_expression* is negative, NULL is returned.

## Return Types

Returns the same type as *string\_expression*.

## Examples

### A. Using REPLICATE

The following example replicates a `'0'` character four times in front of a production line code in the AdventureWorks2012 database.

```
SELECT [Name]
, REPLICATE('0', 4) + [ProductLine] AS 'Line Code'
FROM [Production].[Product]
WHERE [ProductLine] = 'T'
ORDER BY [Name];
GO
```

Here is the result set.

Name	Line Code
HL Touring Frame - Blue, 46	0000T
HL Touring Frame - Blue, 50	0000T
HL Touring Frame - Blue, 54	0000T
HL Touring Frame - Blue, 60	0000T
HL Touring Frame - Yellow, 46	0000T
HL Touring Frame - Yellow, 50	0000T
...	

## B. Using REPLICATE and DATALENGTH

The following example left pads numbers to a specified length as they are converted from a numeric data type to character or Unicode.

```
IF EXISTS(SELECT name FROM sys.tables
      WHERE name = 't1')
    DROP TABLE t1;
GO
CREATE TABLE t1
(
  c1 varchar(3),
  c2 char(3)
);
GO
INSERT INTO t1 VALUES ('2', '2'), ('37', '37'), ('597', '597');
GO
SELECT REPLICATE('0', 3 - DATALENGTH(c1)) + c1 AS 'Varchar Column',
       REPLICATE('0', 3 - DATALENGTH(c2)) + c2 AS 'Char Column'
FROM t1;
GO
```

Here is the result set.

Varchar Column	Char Column
002	2
037	37
597	597

(3 row(s) affected)

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C: Using REPLICATE

The following example replicates a `\0` character four times in front of an `ItemCode` value.

```
-- Uses AdventureWorks

SELECT EnglishProductName AS Name,
       ProductAlternateKey AS ItemCode,
       REPLICATE('0', 4) + ProductAlternateKey AS FullItemCode
FROM dbo.DimProduct
ORDER BY Name;
```

Here are the first rows in the result set.

Name	ItemCode	FullItemCode
Adventure Works	1	00001

-----

Adjustable Race AR-5381 0000AR-5381

All-Purpose Bike Stand ST-1401 0000ST-1401

AWC Logo Cap CA-1098 0000CA-1098

AWC Logo Cap CA-1098 0000CA-1098

AWC Logo Cap CA-1098 0000CA-1098

BB Ball Bearing BE-2349 0000BE-2349

## See Also

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

# REVERSE (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns the reverse order of a string value.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
REVERSE ( string_expression )
```

## Arguments

*string\_expression*

*string\_expression* is an [expression](#) of a string or binary data type. *string\_expression* can be a constant, variable, or column of either character or binary data.

## Return Types

**varchar** or **nvarchar**

## Remarks

*string\_expression* must be of a data type that is implicitly convertible to **varchar**. Otherwise, use [CAST](#) to explicitly convert *string\_expression*.

## Supplementary Characters (Surrogate Pairs)

When using SC collations, the REVERSE function will not reverse the order of two halves of a surrogate pair.

## Examples

The following example returns all contact first names with the characters reversed. This example uses the **AdventureWorks2012** database.

```
SELECT FirstName, REVERSE(FirstName) AS Reverse
FROM Person.Person
WHERE BusinessEntityID < 5
ORDER BY FirstName;
GO
```

Here is the result set.

FirstName	Reverse
-----------	---------

-----	-----
-------	-------

Ken neK
---------

```
Rob boR
```

```
Roberto otreboR
```

```
Terri irreT
```

```
(4 row(s) affected)
```

The following example reverses the characters in a variable.

```
DECLARE @myvar varchar(10);
SET @myvar = 'sdrawkcaB';
SELECT REVERSE(@myvar) AS Reversed ;
GO
```

The following example makes an implicit conversion from an **int** data type into **varchar** data type and then reverses the result.

```
SELECT REVERSE(1234) AS Reversed ;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example returns names of all databases, and the names with the characters reversed.

```
SELECT name, REVERSE(name) FROM sys.databases;
GO
```

The following example reverses the characters in a variable.

```
DECLARE @myvar varchar(10);
SET @myvar = 'sdrawkcaB';
SELECT REVERSE(@myvar) AS Reversed ;
GO
```

The following example makes an implicit conversion from an **int** data type into **varchar** data type and then reverses the result.

```
SELECT REVERSE(1234) AS Reversed ;
GO
```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

# RIGHT (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the right part of a character string with the specified number of characters.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
RIGHT ( character_expression , integer_expression )
```

## Arguments

*character\_expression*

Is an [expression](#) of character or binary data. *character\_expression* can be a constant, variable, or column. *character\_expression* can be of any data type, except **text** or **ntext**, that can be implicitly converted to **varchar** or **nvarchar**. Otherwise, use the [CAST](#) function to explicitly convert *character\_expression*.

*integer\_expression*

Is a positive integer that specifies how many characters of *character\_expression* will be returned. If *integer\_expression* is negative, an error is returned. If *integer\_expression* is type **bigint** and contains a large value, *character\_expression* must be of a large data type such as **varchar(max)**.

## Return Types

Returns **varchar** when *character\_expression* is a non-Unicode character data type.

Returns **nvarchar** when *character\_expression* is a Unicode character data type.

## Supplementary Characters (Surrogate Pairs)

When using SC collations, the RIGHT function counts a UTF-16 surrogate pair as a single character. For more information, see [Collation and Unicode Support](#).

## Examples

### A: Using RIGHT with a column

The following example returns the five rightmost characters of the first name for each person in the AdventureWorks2012 database.

```
SELECT RIGHT(FirstName, 5) AS 'First Name'  
FROM Person.Person  
WHERE BusinessEntityID < 5  
ORDER BY FirstName;  
GO
```

Here is the result set.

```
First Name
-----
Ken
Terri
berto
Rob

(4 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### B. Using RIGHT with a column

The following example returns the five rightmost characters of each last name in the `DimEmployee` table.

```
-- Uses AdventureWorks

SELECT RIGHT(LastName, 5) AS Name
FROM dbo.DimEmployee
ORDER BY EmployeeKey;
```

Here is a partial result set.

Name
-----
lbert
Brown
rello
lters

### C. Using RIGHT with a character string

The following example uses `RIGHT` to return the two rightmost characters of the character string `abcdefg`.

```
-- Uses AdventureWorks

SELECT TOP(1) RIGHT('abcdefg',2) FROM dbo.DimProduct;
```

Here is the result set.

-----
fg

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

# RTRIM (Transact-SQL)

7/5/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a character string after truncating all trailing spaces.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
RTRIM ( character_expression )
```

## Arguments

*character\_expression*

Is an [expression](#) of character data. *character\_expression* can be a constant, variable, or column of either character or binary data.

*character\_expression* must be of a data type that is implicitly convertible to **varchar**. Otherwise, use [CAST](#) to explicitly convert *character\_expression*.

## Return Types

**varchar** or **nvarchar**

## Examples

### A. Simple Example

The following example takes a string of characters that has spaces at the end of the sentence, and returns the text without the spaces at the end of the sentence.

```
SELECT RTRIM('Removes trailing spaces.      ');
```

Here is the result set.

```
Removes trailing spaces.
```

### B: Simple Example

The following example demonstrates how to use `RTRIM` to remove trailing spaces. This time there is another string concatenated to the first string to show that the spaces are gone.

```
SELECT RTRIM('Four spaces are after the period in this sentence.      ') + 'Next string.';
```

Here is the result set.

```
Four spaces are after the period in this sentence.Next string.
```

### C. Using RTRIM with a variable

The following example demonstrates how to use `RTRIM` to remove trailing spaces from a character variable.

```
DECLARE @string_to_trim varchar(60);
SET @string_to_trim = 'Four spaces are after the period in this sentence.      ';
SELECT @string_to_trim + ' Next string.';
SELECT RTRIM(@string_to_trim) + ' Next string.';
GO
```

Here is the result set.

```
-----
Four spaces are after the period in this sentence.      Next string.

(1 row(s) affected)

-----
Four spaces are after the period in this sentence. Next string.

(1 row(s) affected)
```

## See Also

[CAST and CONVERT \(Transact-SQL\)](#)

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

[TRIM \(Transact-SQL\)](#)

[LTRIM \(Transact-SQL\)](#)

# SOUNDEX (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a four-character (SOUNDEX) code to evaluate the similarity of two strings.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SOUNDEX ( character_expression )
```

## Arguments

*character\_expression*

Is an alphanumeric [expression](#) of character data. *character\_expression* can be a constant, variable, or column.

## Return Types

**varchar**

## Remarks

SOUNDEX converts an alphanumeric string to a four-character code that is based on how the string sounds when spoken. The first character of the code is the first character of *character\_expression*, converted to upper case. The second through fourth characters of the code are numbers that represent the letters in the expression. The letters A, E, I, O, U, H, W, and Y are ignored unless they are the first letter of the string. Zeroes are added at the end if necessary to produce a four-character code. For more information about the SOUNDEX code, see [The Soundex Indexing System](#).

SOUNDEX codes from different strings can be compared to see how similar the strings sound when spoken. The DIFFERENCE function performs a SOUNDEX on two strings, and returns an integer that represents how similar the SOUNDEX codes are for those strings.

SOUNDEX is collation sensitive. String functions can be nested.

## SOUNDEX Compatibility

In previous versions of SQL Server, the SOUNDEX function applied a subset of the SOUNDEX rules. Under database compatibility level 110 or higher, SQL Server applies a more complete set of the rules.

After upgrading to compatibility level 110 or higher, you may need to rebuild the indexes, heaps, or CHECK constraints that use the SOUNDEX function.

- A heap that contains a persisted computed column defined with SOUNDEX cannot be queried until the heap is rebuilt by running the statement `ALTER TABLE <table> REBUILD`.
- CHECK constraints defined with SOUNDEX are disabled upon upgrade. To enable the constraint, run the statement `ALTER TABLE <table> WITH CHECK CHECK CONSTRAINT ALL`.

- Indexes (including indexed views) that contain a persisted computed column defined with SOUNDEX cannot be queried until the index is rebuilt by running the statement `ALTER INDEX ALL ON <object> REBUILD`.

## Examples

The following example shows the SOUNDEX function and the related DIFFERENCE function. In the first example, the standard SOUNDEX values are returned for all consonants. Returning the SOUNDEX for `Smith` and `Smythe` returns the same SOUNDEX result because all vowels, the letter `y`, doubled letters, and the letter `h`, are not included.

```
-- Using SOUNDEX
SELECT SOUNDEX ('Smith'), SOUNDEX ('Smythe');
```

Here is the result set. Valid for a Latin1\_General collation.

```
-----  
S530  S530  
(1 row(s) affected)
```

The DIFFERENCE function compares the difference of the SOUNDEX pattern results. The following example shows two strings that differ only in vowels. The difference returned is `4`, the lowest possible difference.

```
-- Using DIFFERENCE
SELECT DIFFERENCE('Smithers', 'Smythers');
GO
```

Here is the result set. Valid for a Latin1\_General collation.

```
-----  
4  
(1 row(s) affected)
```

In the following example, the strings differ in consonants; therefore, the difference returned is `2`, the greater difference.

```
SELECT DIFFERENCE('Anothers', 'Brothers');
GO
```

Here is the result set. Valid for a Latin1\_General collation.

```
-----  
2  
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example shows the SOUNDEX function and the related DIFFERENCE function. In the first example, the standard SOUNDEX values are returned for all consonants. Returning the SOUNDEX for `Smith` and `Smythe`

returns the same SOUNDEX result because all vowels, the letter `y`, doubled letters, and the letter `h`, are not included.

```
-- Using SOUNDEX
SELECT SOUNDEX ('Smith'), SOUNDEX ('Smythe');
```

Here is the result set. Valid for a Latin1\_General collation.

```
-----
S530  S530
(1 row(s) affected)
```

The `DIFFERENCE` function compares the difference of the `SOUNDEX` pattern results. The following example shows two strings that differ only in vowels. The difference returned is `4`, the lowest possible difference.

```
-- Using DIFFERENCE
SELECT DIFFERENCE('Smithers', 'Smythers');
GO
```

Here is the result set. Valid for a Latin1\_General collation.

```
-----
4
(1 row(s) affected)
```

In the following example, the strings differ in consonants; therefore, the difference returned is `2`, the greater difference.

```
SELECT DIFFERENCE('Anothers', 'Brothers');
GO
```

Here is the result set. Valid for a Latin1\_General collation.

```
-----
2
(1 row(s) affected)
```

## See Also

[DIFFERENCE \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

[ALTER DATABASE Compatibility Level \(Transact-SQL\)](#)

# SPACE (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns a string of repeated spaces.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SPACE ( integer_expression )
```

## Arguments

*integer\_expression*

Is a positive integer that indicates the number of spaces. If *integer\_expression* is negative, a null string is returned.

For more information, see [Expressions \(Transact-SQL\)](#)

## Return Types

**varchar**

## Remarks

To include spaces in Unicode data, or to return more than 8000 character spaces, use REPLICATE instead of SPACE.

## Examples

The following example trims the last names and concatenates a comma, two spaces, and the first names of people listed in the `Person` table in `AdventureWorks2012`.

```
USE AdventureWorks2012;
GO
SELECT RTRIM(LastName) + ',' + SPACE(2) + LTRIM.FirstName
FROM Person.Person
ORDER BY LastName, FirstName;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example trims the last names and concatenates a comma, two spaces, and the first names of people listed in the `DimCustomer` table in `AdventureWorksPDW2012`.

```
-- Uses AdventureWorks

SELECT RTRIM(LastName) + ',' + SPACE(2) + LTRIM(FirstName)
FROM dbo.DimCustomer
ORDER BY LastName, FirstName;
GO
```

## See Also

[REPLICATE \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

[Built-in Functions \(Transact-SQL\)](#)

# STR (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data

Warehouse ✓ Parallel Data Warehouse

Returns character data converted from numeric data.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
STR ( float_expression [ , length [ , decimal ] ] )
```

## Arguments

*float\_expression*

Is an expression of approximate numeric (**float**) data type with a decimal point.

*length*

Is the total length. This includes decimal point, sign, digits, and spaces. The default is 10.

*decimal*

Is the number of places to the right of the decimal point. *decimal* must be less than or equal to 16. If *decimal* is more than 16 then the result is truncated to sixteen places to the right of the decimal point.

## Return Types

**varchar**

## Remarks

If supplied, the values for *length* and *decimal* parameters to STR should be positive. The number is rounded to an integer by default or if the decimal parameter is 0. The specified length should be greater than or equal to the part of the number before the decimal point plus the number's sign (if any). A short *float\_expression* is right-justified in the specified length, and a long *float\_expression* is truncated to the specified number of decimal places. For example, STR(12,10) yields the result of 12. This is right-justified in the result set. However, STR(1223,2) truncates the result set to \*\*. String functions can be nested.

### NOTE

To convert to Unicode data, use STR inside a CONVERT or **CAST** conversion function.

## Examples

The following example converts an expression that is made up of five digits and a decimal point to a six-position character string. The fractional part of the number is rounded to one decimal place.

```
SELECT STR(123.45, 6, 1);
GO
```

Here is the result set.

```
-----
123.5
(1 row(s) affected)
```

When the expression exceeds the specified length, the string returns `**` for the specified length.

```
SELECT STR(123.45, 2, 2);
GO
```

Here is the result set.

```
-- 
**
(1 row(s) affected)
```

Even when numeric data is nested within `STR`, the result is character data with the specified format.

```
SELECT STR(FLOOR(123.45), 8, 3);
GO
```

Here is the result set.

```
-----
123.000
(1 row(s) affected)
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example converts an expression that is made up of five digits and a decimal point to a six-position character string. The fractional part of the number is rounded to one decimal place.

```
SELECT STR(123.45, 6, 1);
GO
```

Here is the result set.

```
-----
123.5
(1 row(s) affected)
```

When the expression exceeds the specified length, the string returns `**` for the specified length.

```
SELECT STR(123.45, 2, 2);
GO
```

Here is the result set.

```
--  
**  
  
(1 row(s) affected)
```

Even when numeric data is nested within `STR`, the result is character data with the specified format.

```
SELECT STR(FLOOR(123.45), 8, 3);
GO
```

Here is the result set.

```
-----  
123.000  
  
(1 row(s) affected)
```

## See Also

[String Functions \(Transact-SQL\)](#)

# STRING\_AGG (Transact-SQL)

7/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2017) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Concatenates the values of string expressions and places separator values between them. The separator is not added at the end of string.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
STRING_AGG ( expression, separator ) [ <order_clause> ]  
  
<order_clause> ::=  
    WITHIN GROUP ( ORDER BY <order_by_expression_list> [ ASC | DESC ] )
```

## Arguments

### *separator*

Is an [expression](#) of `NVARCHAR` or `VARCHAR` type that is used as separator for concatenated strings. It can be literal or variable.

### *expression*

Is an [expression](#) of any type. Expressions are converted to `NVARCHAR` or `VARCHAR` types during concatenation. Non-string types are converted to `NVARCHAR` type.

Optionally specify order of concatenated results using `WITHIN GROUP` clause:

```
WITHIN GROUP ( ORDER BY <order_by_expression_list> [ ASC | DESC ] )
```

A list of non-constant [expressions](#) that can be used for sorting results. Only one `order_by_expression` is allowed per query. The default sort order is ascending.

## Return Types

Return type is depends on first argument (expression). If input argument is string type (`NVARCHAR`, `VARCHAR`), result type will be same as input type. The following table lists automatic conversions:

INPUT EXPRESSION TYPE	RESULT
<code>NVARCHAR(MAX)</code>	<code>NVARCHAR(MAX)</code>
<code>VARCHAR(MAX)</code>	<code>VARCHAR(MAX)</code>
<code>NVARCHAR(1...4000)</code>	<code>NVARCHAR(4000)</code>
<code>VARCHAR(1...8000)</code>	<code>VARCHAR(8000)</code>

INPUT EXPRESSION TYPE	RESULT
int, bigint, smallint, tinyint, numeric, float, real, bit, decimal, smallmoney, money, datetime, datetime2,	NVARCHAR(4000)

## Remarks

`STRING_AGG` aggregate takes all expressions from rows and concatenates them into a single string. Expression values are implicitly converted to string types and then concatenated. The implicit conversion to strings follows the existing rules for data type conversions. For more information about data type conversions, see [CAST and CONVERT \(Transact-SQL\)](#).

If the input expression is type `VARCHAR`, the separator cannot be type `NVARCHAR`.

Null values are ignored and the corresponding separator is not added. To return a place holder for null values, use the `ISNULL` function as demonstrated in example B.

`STRING_AGG` is available in any compatibility level.

## Examples

### A. Generate list of names separated in new lines

The following example produces a list of names in a single result cell, separated with carriage returns.

```
SELECT STRING_AGG (FirstName, CHAR(13)) AS csv
FROM Person.Person;
```

Here is the result set.

**CSV**

Syed  
Catherine  
Kim  
Kim  
Kim  
Hazem  
...

`NULL` values found in `name` cells are not returned in result.

#### NOTE

If using the Management Studio Query Editor, the **Results to Grid** option cannot implement the carriage return. Switch to **Results to Text** to see the result set properly.

### B. Generate list of names separated with comma without NULL values

The following example replaces null values with 'N/A' and returns the names separated by commas in a single result cell.

```
SELECT STRING_AGG ( ISNULL(FirstName,'N/A'), ',') AS csv
FROM Person.Person;
```

Here is the result set.

**CSV**

John,N/A,Mike,Peter,N/A,N/A,Alice,Bob

### C. Generate comma-separated values

```
SELECT
STRING_AGG(CONCAT(FirstName, ' ', LastName, ' (' , ModifiedDate, ')'), CHAR(13))
AS names
FROM Person.Person;
```

Here is the result set.

**NAMES**

Ken Sánchez (Feb 8 2003 12:00AM)  
Terri Duffy (Feb 24 2002 12:00AM)  
Roberto Tamburello (Dec 5 2001 12:00AM)  
Rob Walters (Dec 29 2001 12:00AM)  
...

#### NOTE

If using the Management Studio Query Editor, the **Results to Grid** option cannot implement the carriage return. Switch to **Results to Text** to see the result set properly.

### D. Return news articles with related tags

Article and their tags are separated into different tables. Developer wants to return one row per each article with all associated tags. Using following query:

```
SELECT a.articleId, title, STRING_AGG (tag, ',') as tags
FROM dbo.Article AS a
LEFT JOIN dbo.ArticleTag AS t
    ON a.ArticleId = t.ArticleId
GROUP BY a.articleId, title;
```

Here is the result set.

ARTICLEID	TITLE	TAGS
172	Polls indicate close election results	politics,polls,city council
176	New highway expected to reduce congestion	NULL
177	Dogs continue to be more popular than cats	polls,animals

### E. Generate list of emails per towns

The following query finds the email addresses of employees and groups them by towns:

```
SELECT town, STRING_AGG (email, ',') AS emails
FROM dbo.Employee
GROUP BY town;
```

Here is the result set.

TOWN	EMAILS
Seattle	syed0@adventure-works.com;catherine0@adventure-works.com;kim2@adventure-works.com
LA	sam1@adventure-works.com;hazem0@adventure-works.com

Emails returned in the emails column can be directly used to send emails to group of people working in some particular towns.

#### F. Generate a sorted list of emails per towns

Similar to previous example, the following query finds the email addresses of employees, groups them by town, and sorts the emails alphabetically:

```
SELECT town,
       STRING_AGG (email, ',') WITHIN GROUP (ORDER BY email ASC) AS emails
  FROM dbo.Employee
 GROUP BY town;
```

Here is the result set.

TOWN	EMAILS
Seattle	catherine0@adventure-works.com;kim2@adventure-works.com;syed0@adventure-works.com
LA	hazem0@adventure-works.com;sam1@adventure-works.com

## See Also

[String Functions \(Transact-SQL\)](#)

# STRING\_ESCAPE (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Escapes special characters in texts and returns text with escaped characters. **STRING\_ESCAPE** is a deterministic function.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
STRING_ESCAPE( text , type )
```

## Arguments

*text*

Is a **nvarchar** expression representing the object that should be escaped.

*type*

Escaping rules that will be applied. Currently the value supported is `'json'`.

## Return Types

**nvarchar(max)** text with escaped special and control characters. Currently **STRING\_ESCAPE** can only escape JSON special characters shown in the following tables.

SPECIAL CHARACTER	ENCODED SEQUENCE
Quotation mark ("")	\"
Reverse solidus (\)	\\\
Solidus (/)	\/
Backspace	\b
Form feed	\f
New line	\n
Carriage return	\r
Horizontal tab	\t

CONTROL CHARACTER	ENCODED SEQUENCE
CHAR(0)	\u0000

CONTROL CHARACTER	ENCODED SEQUENCE
CHAR(1)	\u0001
...	...
CHAR(31)	\u001f

## Remarks

## Examples

### A. Escape text according to the JSON formatting rules

The following query escapes special characters using JSON rules and returns escaped text.

```
SELECT STRING_ESCAPE('\' /  
\\ " ', 'json') AS escapedText;
```

Here is the result set.

escapedText
-------------

-----
-------

\\\t\\/\n\\\\\\t\\\"\\t
-------------------------

### B. Format JSON object

The following query creates JSON text from number and string variables, and escapes any special JSON character in variables.

```
SET @json = FORMATMESSAGE('{"id": %d,"name": "%s", "surname": "%s" }',  
    17, STRING_ESCAPE(@name,'json'), STRING_ESCAPE(@surname,'json') );
```

## See Also

[String Functions \(Transact-SQL\)](#)

[SUBSTRING \(Transact-SQL\)](#)

## STRING\_SPLIT (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:**  SQL Server (starting with 2016)  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

Splits the character expression using specified separator.

## NOTE

The **STRING\_SPLIT** function is available only under compatibility level 130. If your database compatibility level is lower than 130, SQL Server will not be able to find and execute **STRING\_SPLIT** function. You can change a compatibility level of database using the following command:

```
ALTER DATABASE DatabaseName SET COMPATIBILITY LEVEL = 130
```

Note that compatibility level 120 might be default even in new Azure SQL Databases.



## Syntax

**STRING\_SPLIT ( string , separator )**

## Arguments

*string*

Is an [expression](#) of any character type (i.e. **nvarchar**, **varchar**, **nchar** or **char**).

### *separator*

Is a single character [expression](#) of any character type (e.g. **nvarchar(1)**, **varchar(1)**, **nchar(1)** or **char(1)**) that is used as separator for concatenated strings.

## Return Types

Returns a single-column table with fragments. The name of the column is **value**. Returns **nvarchar** if any of the input arguments are either **nvarchar** or **nchar**. Otherwise returns **varchar**. The length of the return type is the same as the length of the string argument.

## Remarks

**STRING\_SPLIT** takes a string that should be divided and the separator that will be used to divide string. It returns a single-column table with substrings. For example, the following statement

`SELECT value FROM STRING_SPLIT('Lorem ipsum dolor sit amet.', ' ');` using the space character as the separator, returns following result table:

VALUE
ipsum
dolor
sit
amet.

If the input string is **NULL**, the **STRING\_SPLIT** table-valued function returns an empty table.

**STRING\_SPLIT** requires at least compatibility mode 130.

## Examples

### A. Split comma separated value string

Parse a comma separated list of values and return all non-empty tokens:

```
DECLARE @tags NVARCHAR(400) = 'clothing,road,,touring,bike'

SELECT value
FROM STRING_SPLIT(@tags, ',')
WHERE RTRIM(value) <> '';
```

STRING\_SPLIT will return empty string if there is nothing between separator. Condition RTRIM(value) <> "" will remove empty tokens.

### B. Split comma separated value string in a column

Product table has a column with comma-separate list of tags shown in the following example:

PRODUCTID	NAME	TAGS
1	Full-Finger Gloves	clothing,road,touring,bike
2	LL Headset	bike
3	HL Mountain Frame	bike,mountain

Following query transforms each list of tags and joins them with the original row:

```
SELECT ProductId, Name, value
FROM Product
CROSS APPLY STRING_SPLIT(Tags, ',');
```

Here is the result set.

PRODUCTID	NAME	VALUE
1	Full-Finger Gloves	clothing
1	Full-Finger Gloves	road

PRODUCTID	NAME	VALUE
1	Full-Finger Gloves	touring
1	Full-Finger Gloves	bike
2	LL Headset	bike
3	HL Mountain Frame	bike
3	HL Mountain Frame	mountain

### C. Aggregation by values

Users must create a report that shows the number of products per each tag, ordered by number of products, and to filter only the tags with more than 2 products.

```
SELECT value as tag, COUNT(*) AS [Number of articles]
FROM Product
    CROSS APPLY STRING_SPLIT(Tags, ',')
GROUP BY value
HAVING COUNT(*) > 2
ORDER BY COUNT(*) DESC;
```

### D. Search by tag value

Developers must create queries that find articles by keywords. They can use following queries:

To find products with a single tag (clothing):

```
SELECT ProductId, Name, Tags
FROM Product
WHERE 'clothing' IN (SELECT value FROM STRING_SPLIT(Tags, ','));
```

Find products with two specified tags (clothing and road):

```
SELECT ProductId, Name, Tags
FROM Product
WHERE EXISTS (SELECT *
    FROM STRING_SPLIT(Tags, ',')
    WHERE value IN ('clothing', 'road'));
```

### E. Find rows by list of values

Developers must create a query that finds articles by a list of ids. They can use following query:

```
SELECT ProductId, Name, Tags
FROM Product
JOIN STRING_SPLIT('1,2,3', ',')
ON value = ProductId;
```

This is replacement for common anti-pattern such as creating a dynamic SQL string in application layer or Transact-SQL, or by using LIKE operator:

```
SELECT ProductId, Name, Tags
FROM Product
WHERE ',1,2,3,' LIKE '%' + CAST(ProductId AS VARCHAR(20)) + ',';
```

## See Also

[String Functions \(Transact-SQL\)](#)

[SUBSTRING \(Transact-SQL\)](#)

# STUFF (Transact-SQL)

9/7/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

The STUFF function inserts a string into another string. It deletes a specified length of characters in the first string at the start position and then inserts the second string into the first string at the start position.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
STUFF ( character_expression , start , length , replaceWith_expression )
```

## Arguments

### *character\_expression*

Is an [expression](#) of character data. *character\_expression* can be a constant, variable, or column of either character or binary data.

### *start*

Is an integer value that specifies the location to start deletion and insertion. If *start* or *length* is negative, a null string is returned. If *start* is longer than the first *character\_expression*, a null string is returned. *start* can be of type **bigint**.

### *length*

Is an integer that specifies the number of characters to delete. If *length* is longer than the first *character\_expression*, deletion occurs up to the last character in the last *character\_expression*. *length* can be of type **bigint**.

### *replaceWith\_expression*

Is an [expression](#) of character data. *character\_expression* can be a constant, variable, or column of either character or binary data. This expression replaces *length* characters of *character\_expression* beginning at *start*. Providing `NULL` as the *replaceWith\_expression*, removes characters without inserting anything.

## Return Types

Returns character data if *character\_expression* is one of the supported character data types. Returns binary data if *character\_expression* is one of the supported binary data types.

## Remarks

If the start position or the length is negative, or if the starting position is larger than length of the first string, a null string is returned. If the start position is 0, a null value is returned. If the length to delete is longer than the first string, it is deleted to the first character in the first string.

An error is raised if the resulting value is larger than the maximum supported by the return type.

## Supplementary Characters (Surrogate Pairs)

When using SC collations, both *character\_expression* and *replaceWith\_expression* can include surrogate pairs. The

length parameter counts each surrogate in *character\_expression* as a single character.

## Examples

The following example returns a character string created by deleting three characters from the first string, `abcdef`, starting at position `2`, at `b`, and inserting the second string at the deletion point.

```
SELECT STUFF('abcdef', 2, 3, 'ijklmn');
GO
```

Here is the result set.

```
-----
aijklmnef
(1 row(s) affected)
```

## See Also

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

# SUBSTRING (Transact-SQL)

9/27/2017 • 5 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns part of a character, binary, text, or image expression in SQL Server.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
SUBSTRING ( expression ,start , length )
```

## Arguments

*expression*

Is a **character**, **binary**, **text**, **ntext**, or **image** *expression*.

*start*

Is an integer or **bigint** expression that specifies where the returned characters start. (The numbering is 1 based, meaning that the first character in the expression is 1). If *start* is less than 1, the returned expression will begin at the first character that is specified in *expression*. In this case, the number of characters that are returned is the largest value of either the sum of *start* + *length* - 1 or 0. If *start* is greater than the number of characters in the *value expression*, a zero-length expression is returned.

*length*

Is a positive integer or **bigint** expression that specifies how many characters of the *expression* will be returned. If *length* is negative, an error is generated and the statement is terminated. If the sum of *start* and *length* is greater than the number of characters in *expression*, the whole value expression beginning at *start* is returned.

## Return Types

Returns character data if *expression* is one of the supported character data types. Returns binary data if *expression* is one of the supported **binary** data types. The returned string is the same type as the specified expression with the exceptions shown in the table.

SPECIFIED EXPRESSION	RETURN TYPE
<b>char</b> / <b>varchar</b> / <b>text</b>	<b>varchar</b>
<b>nchar</b> / <b>nvarchar</b> / <b>ntext</b>	<b>nvarchar</b>
<b>binary</b> / <b>varbinary</b> / <b>image</b>	<b>varbinary</b>

## Remarks

The values for *start* and *length* must be specified in number of characters for **ntext**, **char**, or **varchar** data types and bytes for **text**, **image**, **binary**, or **varbinary** data types.

The *expression* must be **varchar(max)** or **varbinary(max)** when the *start* or *length* contains a value larger than 2147483647.

## Supplementary Characters (Surrogate Pairs)

When using supplementary character (SC) collations, both *start* and *length* count each surrogate pair in *expression* as a single character. For more information, see [Collation and Unicode Support](#).

## Examples

### A. Using SUBSTRING with a character string

The following example shows how to return only a part of a character string. From the `sys.databases` table, this query returns the system database names in the first column, the first letter of the database in the second column, and the third and fourth characters in the final column.

```
SELECT name, SUBSTRING(name, 1, 1) AS Initial ,
       SUBSTRING(name, 3, 2) AS ThirdAndFourthCharacters
  FROM sys.databases
 WHERE database_id < 5;
```

Here is the result set.

NAME	INITIAL	THIRDANDFOURTHCHARACTERS
master	m	st
tempdb	t	mp
model	m	de
msdb	m	db

Here is how to display the second, third, and fourth characters of the string constant `abcdef`.

```
SELECT x = SUBSTRING('abcdef', 2, 3);
```

Here is the result set.

```
x
-----
bcd
(1 row(s) affected)
```

### B. Using SUBSTRING with text, ntext, and image data

#### NOTE

To run the following examples, you must install the `pubs` database.

The following example shows how to return the first 10 characters from each of a **text** and **image** data column in the `pub_info` table of the `pubs` database. **text** data is returned as **varchar**, and **image** data is returned as

## varbinary.

```
USE pubs;
SELECT pub_id, SUBSTRING(logo, 1, 10) AS logo,
       SUBSTRING(pr_info, 1, 10) AS pr_info
  FROM pub_info
 WHERE pub_id = '1756';
```

Here is the result set.

pub_id	logo	pr_info
1756	0x474946383961E3002500	This is sa

(1 row(s) affected)

The following example shows the effect of SUBSTRING on both **text** and **ntext** data. First, this example creates a new table in the `pubs` database named `npub_info`. Second, the example creates the `pr_info` column in the `npub_info` table from the first 80 characters of the `pub_info.pr_info` column and adds an ü as the first character. Lastly, an `INNER JOIN` retrieves all publisher identification numbers and the `SUBSTRING` of both the **text** and **ntext** publisher information columns.

```
IF EXISTS (SELECT table_name FROM INFORMATION_SCHEMA.TABLES
           WHERE table_name = 'npub_info')
    DROP TABLE npub_info;
GO
-- Create npub_info table in pubs database. Borrowed from instpubs.sql.
USE pubs;
GO
CREATE TABLE npub_info
(
    pub_id char(4) NOT NULL
        REFERENCES publishers(pub_id)
        CONSTRAINT UPKCL_npubinfo PRIMARY KEY CLUSTERED,
    pr_info ntext NULL
);
GO
-- Fill the pr_info column in npub_info with international data.
RAISERROR('Now at the inserts to pub_info...',0,1);

GO
INSERT npub_info VALUES('0736', N'üThis is sample text data for New Moon Books, publisher 0736 in the pubs
database')
,('0877', N'üThis is sample text data for Binnet & Hardley, publisher 0877 in the pubs database')
,('1389', N'üThis is sample text data for Algodata Infosystems, publisher 1389 in the pubs database')
,('9952', N'üThis is sample text data for Scootney Books, publisher 9952 in the pubs database')
,('1622', N'üThis is sample text data for Five Lakes Publishing, publisher 1622 in the pubs database')
,('1756', N'üThis is sample text data for Ramona Publishers, publisher 1756 in the pubs database')
,('9901', N'üThis is sample text data for GGG&G, publisher 9901 in the pubs database. GGG&G i')
,('9999', N'üThis is sample text data for Lucerne Publishing, publisher 9999 in the pubs database');
GO
-- Join between npub_info and pub_info on pub_id.
SELECT pr.pub_id, SUBSTRING(pr.pr_info, 1, 35) AS pr_info,
       SUBSTRING(npr.pr_info, 1, 35) AS npr_info
  FROM pub_info pr INNER JOIN npub_info npr
    ON pr.pub_id = npr.pub_id
 ORDER BY pr.pub_id ASC;
```

# Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

## C. Using SUBSTRING with a character string

The following example shows how to return only a part of a character string. From the `dbo.DimEmployee` table, this query returns the last name in one column with only the first initial in the second column.

```
-- Uses AdventureWorks

SELECT LastName, SUBSTRING(FirstName, 1, 1) AS Initial
FROM dbo.DimEmployee
WHERE LastName LIKE 'Bar%'
ORDER BY LastName;
```

Here is the result set.

LastName	Initial
-----	-----
Barbariol	A
Barber	D
Barreto de Mattos	P

The following example shows how to return the second, third, and fourth characters of the string constant `abcdef`.

```
USE ssawPDW;

SELECT TOP 1 SUBSTRING('abcdef', 2, 3) AS x FROM dbo.DimCustomer;
```

Here is the result set.

x
-----
bcd

## See Also

[String Functions \(Transact-SQL\)](#)

# TRANSLATE (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2017) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the string provided as a first argument after some characters specified in the second argument are translated into a destination set of characters.

## Syntax

```
TRANSLATE ( inputString, characters, translations)
```

## Arguments

inputString

Is an [expression](#) of any character type (nvarchar, varchar, nchar, char).

characters

Is a [expression](#) of any character type containing characters that should be replaced.

translations

Is a character [expression](#) that matches second argument by type and length.

## Return Types

Returns a character expression of the same type as `inputString` where characters from the second argument are replaced with the matching characters from third argument.

## Remarks

`TRANSLATE` function will return an error if characters and translations have different lengths. `TRANSLATE` function should return unchanged input if null values are provided as characters or replacement arguments. The behavior of the `TRANSLATE` function should be identical to the [REPLACE](#) function.

The behavior of the `TRANSLATE` function is equivalent to using multiple `REPLACE` functions.

`TRANSLATE` is always SC collation aware.

## Examples

### A. Replace square and curly braces with regular braces

The following query replaces square and curly braces in the input string with parentheses:

```
SELECT TRANSLATE('2*[3+4]/\{7-2}', '[]{}', '()());
```

Here is the result set.

```
2*(3+4)/(7-2)
```

#### NOTE

The `TRANSLATE` function in this example is equivalent to but much simpler than the following statement using `REPLACE`:

```
SELECT REPLACE(REPLACE(REPLACE(REPLACE('2*[3+4]/{7-2}', '[','('), ']',')'), ')','{','('), '}',''));
```

## B. Convert GeoJSON points into WKT

GeoJSON is a format for encoding a variety of geographic data structures. With the `TRANSLATE` function, developers can easily convert GeoJSON points to WKT format and vice versa. The following query replaces square and curly braces in input with regular braces:

```
SELECT TRANSLATE('[137.4, 72.3]' , '[,]', '( )') AS Point,  
       TRANSLATE('(137.4 72.3)' , '( )', '[,]') AS Coordinates;
```

Here is the result set.

POINT	COORDINATES
(137.4 72.3)	[137.4,72.3]

## See Also

[String Functions \(Transact-SQL\)](#)

[REPLACE \(Transact-SQL\)](#)

# TRIM (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2017) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Removes the space character `char(32)` or other specified characters from the start or end of a string.

## Syntax

```
TRIM ( [ characters FROM ] string )
```

## Arguments

### characters

Is a literal, variable, or function call of any non-LOB character type (`nvarchar`, `varchar`, `nchar`, or `char`) containing characters that should be removed. `nvarchar(max)` and `varchar(max)` types are not allowed.

### string

Is an expression of any character type (`nvarchar`, `varchar`, `nchar`, or `char`) where characters should be removed.

## Return Types

Returns a character expression with a type of string argument where the space character `char(32)` or other specified characters are removed from both sides. Returns `NULL` if input string is `NULL`.

## Remarks

By default `TRIM` function removes the space character `char(32)` from both sides. This is equivalent to `LTRIM(RTRIM(@string))`. Behavior of `TRIM` function with specified characters is identical to behavior of `REPLACE` function where characters from start or end are replaced with empty strings.

## Examples

### A. Removes the space character from both sides of string

The following example removes spaces from before and after the word `test`.

```
SELECT TRIM( '      test      ') AS Result;
```

Here is the result set.

```
test
```

### B. Removes specified characters from both sides of string

The following example removes a trailing period and trailing spaces.

```
SELECT TRIM( '.,! ' FROM '#      test      .') AS Result;
```

Here is the result set.

```
# test
```

## See Also

[String Functions \(Transact-SQL\)](#)

[LTRIM \(Transact-SQL\)](#)

[RTRIM \(Transact-SQL\)](#)

[REPLACE \(Transact-SQL\)](#)

# UNICODE (Transact-SQL)

9/27/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the integer value, as defined by the Unicode standard, for the first character of the input expression.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
UNICODE ( 'ncharacter_expression' )
```

## Arguments

'*ncharacter\_expression*'

Is an **nchar** or **nvarchar** expression.

## Return Types

**int**

## Remarks

In versions of SQL Server earlier than SQL Server 2012 and in Azure SQL Database, the **UNICODE** function returns a UCS-2 codepoint in the range 0 through 0xFFFF. In SQL Server 2012 and later editions, when using SC collations, **UNICODE** returns a UTF-16 codepoint in the range 0 through 0x10FFFF.

## Examples

### A. Using **UNICODE** and the **NCHAR** function

The following example uses the **UNICODE** and **NCHAR** functions to print the **UNICODE** value of the first character of the `Åkergatan` 24-character string, and to print the actual first character, `Å`.

```
DECLARE @nstring nchar(12);
SET @nstring = N'Åkergatan 24';
SELECT UNICODE(@nstring), NCHAR(UNICODE(@nstring));
```

Here is the result set.

```
----- -
197      Å
```

### B. Using **SUBSTRING**, **UNICODE**, and **CONVERT**

The following example uses the **SUBSTRING**, **UNICODE**, and **CONVERT** functions to print the character number, the Unicode character, and the **UNICODE** value of each of the characters in the string `Åkergatan 24`.

```

-- The @position variable holds the position of the character currently
-- being processed. The @nstring variable is the Unicode character
-- string to process.
DECLARE @position int, @nstring nchar(12);
-- Initialize the current position variable to the first character in
-- the string.
SET @position = 1;
-- Initialize the character string variable to the string to process.
-- Notice that there is an N before the start of the string, which
-- indicates that the data following the N is Unicode data.
SET @nstring = N'Åkergatan 24';
-- Print the character number of the position of the string you are at,
-- the actual Unicode character you are processing, and the UNICODE
-- value for this particular character.
PRINT 'Character #' + ' ' + 'Unicode Character' + ' ' + 'UNICODE Value';
WHILE @position <= DATALENGTH(@nstring)
-- While these are still characters in the character string,
BEGIN;
SELECT @position,
      CONVERT(char(17), SUBSTRING(@nstring, @position, 1)),
      UNICODE(SUBSTRING(@nstring, @position, 1));
SELECT @position = @position + 1;
END;

```

Here is the result set.

Character #	Unicode Character	UNICODE Value
1	Å	197
2	k	107
3	e	101
4	r	114
5	g	103
6	a	97
7	t	116
8	a	97
9	n	110
10		32
11	2	50
12	4	52

Character #	Unicode Character	UNICODE Value
1	Å	197
2	k	107
3	e	101
4	r	114
5	g	103
6	a	97
7	t	116
8	a	97
9	n	110
10		32
11	2	50
12	4	52

# Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

## C. Using UNICODE and the NCHAR function

The following example uses the `UNICODE` and `NCHAR` functions to print the UNICODE value of the first character of the `Åkergatan` 24-character string, and to print the actual first character, `Å`.

```
DECLARE @nstring nchar(12);
SET @nstring = N'Åkergatan 24';
SELECT UNICODE(@nstring), NCHAR(UNICODE(@nstring));
```

Here is the result set.

```
----- -
197      Å
```

## D. Using SUBSTRING, UNICODE, and CONVERT

The following example uses the `SUBSTRING`, `UNICODE`, and `CONVERT` functions to print the character number, the Unicode character, and the UNICODE value of each of the characters in the string `Åkergatan 24`.

```
-- The @position variable holds the position of the character currently
-- being processed. The @nstring variable is the Unicode character
-- string to process.
DECLARE @position int, @nstring nchar(12);
-- Initialize the current position variable to the first character in
-- the string.
SET @position = 1;
-- Initialize the character string variable to the string to process.
-- Notice that there is an N before the start of the string, which
-- indicates that the data following the N is Unicode data.
SET @nstring = N'Åkergatan 24';
-- Print the character number of the position of the string you are at,
-- the actual Unicode character you are processing, and the UNICODE
-- value for this particular character.
PRINT 'Character #' + ' ' + 'Unicode Character' + ' ' + 'UNICODE Value';
WHILE @position <= DATALENGTH(@nstring)
-- While these are still characters in the character string,
BEGIN;
    SELECT @position,
        CONVERT(char(17), SUBSTRING(@nstring, @position, 1)),
        UNICODE(SUBSTRING(@nstring, @position, 1));
    SELECT @position = @position + 1;
END;
```

Here is the result set.

Character # Unicode Character UNICODE Value

Character #	Unicode Character	UNICODE Value
1	Å	197
2	k	107
3	e	101
4	r	114
5	g	103
6	a	97
7	t	116
8	a	97
9	n	110
10		32
11	2	50
12	4	52

## See Also

[NCHAR \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

[Collation and Unicode Support](#)

# UPPER (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns a character expression with lowercase character data converted to uppercase.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
UPPER ( character_expression )
```

## Arguments

*character\_expression*

Is an [expression](#) of character data. *character\_expression* can be a constant, variable, or column of either character or binary data.

*character\_expression* must be of a data type that is implicitly convertible to **varchar**. Otherwise, use [CAST](#) to explicitly convert *character\_expression*.

## Return Types

**varchar** or **nvarchar**

## Examples

The following example uses the `UPPER` and `RTRIM` functions to return the last name of people in the `Person` table in the AdventureWorks2012 database so that it is uppercase, trimmed, and concatenated with the first name.

```
SELECT UPPER(RTRIM(LastName)) + ', ' + FirstName AS Name
FROM Person.Person
ORDER BY LastName;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example uses the `UPPER` and `RTRIM` functions to return the last name of people in the `dbo.DimEmployee` table so that it is in uppercase, trimmed, and concatenated with the first name.

```
-- Uses AdventureWorks

SELECT UPPER(RTRIM(LastName)) + ', ' + FirstName AS Name
FROM dbo.DimEmployee
ORDER BY LastName;
```

Here is a partial result set.

Name

-----

ABbas, Syed

ABERCROMBIE, Kim

ABOLROUS, Hazem

## See Also

[Data Types \(Transact-SQL\)](#)

[String Functions \(Transact-SQL\)](#)

# System Functions (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following system functions perform operations on and return information about values, objects, and settings in SQL Server.

\$PARTITION	ERROR_PROCEDURE
@@ERROR	ERROR_SEVERITY
@@IDENTITY	ERROR_STATE
@@PACK_RECEIVED	FORMATMESSAGE
@@ROWCOUNT	GET_FILESTREAM_TRANSACTION_CONTEXT
@@TRANCOUNT	GETANSINULL
BINARY_CHECKSUM	HOST_ID
CHECKSUM	HOST_NAME
COMPRESS	ISNULL
CONNECTIONPROPERTY	ISNUMERIC
CONTEXT_INFO	MIN_ACTIVE_ROWVERSION
CURRENT_REQUEST_ID	NEWID
CURRENT_TRANSACTION_ID	NEWSEQUENTIALID
DECOMPRESS	ROWCOUNT_BIG
ERROR_LINE	SESSION_CONTEXT
ERROR_MESSAGE	SESSION_ID
ERROR_NUMBER	XACT_STATE

## See Also

[Built-in Functions \(Transact-SQL\)](#)

# \$PARTITION (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the partition number into which a set of partitioning column values would be mapped for any specified partition function in SQL Server 2017.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
[ database_name. ] $PARTITION.partition_function_name(expression)
```

## Arguments

*database\_name*

Is the name of the database that contains the partition function.

*partition\_function\_name*

Is the name of any existing partition function against which a set of partitioning column values are being applied.

*expression*

Is an [expression](#) whose data type must either match or be implicitly convertible to the data type of its corresponding partitioning column. *expression* can also be the name of a partitioning column that currently participates in *partition\_function\_name*.

## Return Types

**int**

## Remarks

\$PARTITION returns an **int** value between 1 and the number of partitions of the partition function.

\$PARTITION returns the partition number for any valid value, regardless of whether the value currently exists in a partitioned table or index that uses the partition function.

## Examples

### A. Getting the partition number for a set of partitioning column values

The following example creates a partition function `RangePF1` that will partition a table or index into four partitions. \$PARTITION is used to determine that the value `10`, representing the partitioning column of `RangePF1`, would be put in partition 1 of the table.

```
USE AdventureWorks2012;
GO
CREATE PARTITION FUNCTION RangePF1 ( int )
AS RANGE FOR VALUES (10, 100, 1000) ;
GO
SELECT $PARTITION.RangePF1 (10) ;
GO
```

## B. Getting the number of rows in each nonempty partition of a partitioned table or index

The following example returns the number of rows in each partition of table `TransactionHistory` that contains data.

The `TransactionHistory` table uses partition function `TransactionRangePF1` and is partitioned on the `TransactionDate` column.

To execute this example, you must first run the PartitionAW.sql script against the **AdventureWorks2012** sample database. For more information, see [PartitioningScript](#).

```
USE AdventureWorks2012;
GO
SELECT $PARTITION.TransactionRangePF1(TransactionDate) AS Partition,
COUNT(*) AS [COUNT] FROM Production.TransactionHistory
GROUP BY $PARTITION.TransactionRangePF1(TransactionDate)
ORDER BY Partition ;
GO
```

## C. Returning all rows from one partition of a partitioned table or index

The following example returns all rows that are in partition 5 of the table `TransactionHistory`.

### NOTE

To execute this example, you must first run the PartitionAW.sql script against the **AdventureWorks2012** sample database. For more information, see [PartitioningScript](#).

```
SELECT * FROM Production.TransactionHistory
WHERE $PARTITION.TransactionRangePF1(TransactionDate) = 5 ;
```

## See Also

[CREATE PARTITION FUNCTION \(Transact-SQL\)](#)

# @@ERROR (Transact-SQL)

8/29/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the error number for the last Transact-SQL statement executed.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
@@ERROR
```

## Return Types

integer

## Remarks

Returns 0 if the previous Transact-SQL statement encountered no errors.

Returns an error number if the previous statement encountered an error. If the error was one of the errors in the sys.messages catalog view, then @@ERROR contains the value from the sys.messages.message\_id column for that error. You can view the text associated with an @@ERROR error number in sys.messages.

Because @@ERROR is cleared and reset on each statement executed, check it immediately following the statement being verified, or save it to a local variable that can be checked later.

Use the TRY...CATCH construct to handle errors. The TRY...CATCH construct also supports additional system functions (ERROR\_LINE, ERROR\_MESSAGE, ERROR\_PROCEDURE, ERROR\_SEVERITY, and ERROR\_STATE) that return more error information than @@ERROR. TRY...CATCH also supports an ERROR\_NUMBER function that is not limited to returning the error number in the statement immediately after the statement that generated an error. For more information, see [TRY...CATCH \(Transact-SQL\)](#).

## Examples

### A. Using @@ERROR to detect a specific error

The following example uses `@@ERROR` to check for a check constraint violation (error #547) in an `UPDATE` statement.

```
USE AdventureWorks2012;
GO
UPDATE HumanResources.EmployeePayHistory
    SET PayFrequency = 4
    WHERE BusinessEntityID = 1;
IF @@ERROR = 547
    PRINT N'A check constraint violation occurred.';
GO
```

### B. Using @@ERROR to conditionally exit a procedure

The following example uses `IF...ELSE` statements to test `@@ERROR` after an `INSERT` statement in a stored procedure. The value of the `@@ERROR` variable determines the return code sent to the calling program, indicating success or failure of the procedure.

```
USE AdventureWorks2012;
GO
-- Drop the procedure if it already exists.
IF OBJECT_ID(N'HumanResources.usp_DeleteCandidate', N'P') IS NOT NULL
    DROP PROCEDURE HumanResources.usp_DeleteCandidate;
GO
-- Create the procedure.
CREATE PROCEDURE HumanResources.usp_DeleteCandidate
(
    @CandidateID INT
)
AS
-- Execute the DELETE statement.
DELETE FROM HumanResources.JobCandidate
    WHERE JobCandidateID = @CandidateID;
-- Test the error value.
IF @@ERROR <> 0
    BEGIN
        -- Return 99 to the calling program to indicate failure.
        PRINT N'An error occurred deleting the candidate information.';
        RETURN 99;
    END
ELSE
    BEGIN
        -- Return 0 to the calling program to indicate success.
        PRINT N'The job candidate has been deleted.';
        RETURN 0;
    END;
GO
```

### C. Using `@@ERROR` with `@@ROWCOUNT`

The following example uses `@@ERROR` with `@@ROWCOUNT` to validate the operation of an `UPDATE` statement. The value of `@@ERROR` is checked for any indication of an error, and `@@ROWCOUNT` is used to ensure that the update was successfully applied to a row in the table.

```

USE AdventureWorks2012;
GO
IF OBJECT_ID(N'Purchasing.usp_ChangePurchaseOrderHeader',N'P')IS NOT NULL
    DROP PROCEDURE Purchasing.usp_ChangePurchaseOrderHeader;
GO
CREATE PROCEDURE Purchasing.usp_ChangePurchaseOrderHeader
(
    @PurchaseOrderID INT
    ,@BusinessEntityID INT
)
AS
-- Declare variables used in error checking.
DECLARE @ErrorVar INT;
DECLARE @RowCountVar INT;

-- Execute the UPDATE statement.
UPDATE PurchaseOrderHeader
    SET BusinessEntityID = @BusinessEntityID
    WHERE PurchaseOrderID = @PurchaseOrderID;

-- Save the @@ERROR and @@ROWCOUNT values in local
-- variables before they are cleared.
SELECT @ErrorVar = @@ERROR
    ,@RowCountVar = @@ROWCOUNT;

-- Check for errors. If an invalid @BusinessEntityID was specified,
-- the UPDATE statement returns a foreign key violation error #547.
IF @ErrorVar < > 0
    BEGIN
        IF @ErrorVar = 547
            BEGIN
                PRINT N'ERROR: Invalid ID specified for new employee.';
                RETURN 1;
            END
        ELSE
            BEGIN
                PRINT N'ERROR: error '
                    + RTRIM(CAST(@ErrorVar AS NVARCHAR(10)))
                    + N' occurred.';
                RETURN 2;
            END
    END
-- Check the row count. @RowCountVar is set to 0
-- if an invalid @PurchaseOrderID was specified.
IF @RowCountVar = 0
    BEGIN
        PRINT 'Warning: The BusinessEntityID specified is not valid';
        RETURN 1;
    END
ELSE
    BEGIN
        PRINT 'Purchase order updated with the new employee';
        RETURN 0;
    END;
GO

```

## See Also

- [TRY...CATCH \(Transact-SQL\)](#)
- [ERROR\\_LINE \(Transact-SQL\)](#)
- [ERROR\\_MESSAGE \(Transact-SQL\)](#)
- [ERROR\\_NUMBER \(Transact-SQL\)](#)
- [ERROR\\_PROCEDURE \(Transact-SQL\)](#)

[ERROR\\_SEVERITY](#) (Transact-SQL)

[ERROR\\_STATE](#) (Transact-SQL)

[@@ROWCOUNT](#) (Transact-SQL)

[sys.messages](#) (Transact-SQL)

# @@IDENTITY (Transact-SQL)

8/29/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Is a system function that returns the last-inserted identity value.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@IDENTITY
```

## Return Types

**numeric(38,0)**

## Remarks

After an INSERT, SELECT INTO, or bulk copy statement is completed, @@IDENTITY contains the last identity value that is generated by the statement. If the statement did not affect any tables with identity columns, @@IDENTITY returns NULL. If multiple rows are inserted, generating multiple identity values, @@IDENTITY returns the last identity value generated. If the statement fires one or more triggers that perform inserts that generate identity values, calling @@IDENTITY immediately after the statement returns the last identity value generated by the triggers. If a trigger is fired after an insert action on a table that has an identity column, and the trigger inserts into another table that does not have an identity column, @@IDENTITY returns the identity value of the first insert. The @@IDENTITY value does not revert to a previous setting if the INSERT or SELECT INTO statement or bulk copy fails, or if the transaction is rolled back.

Failed statements and transactions can change the current identity for a table and create gaps in the identity column values. The identity value is never rolled back even though the transaction that tried to insert the value into the table is not committed. For example, if an INSERT statement fails because of an IGNORE\_DUP\_KEY violation, the current identity value for the table is still incremented.

@@IDENTITY, SCOPE\_IDENTITY, and IDENT\_CURRENT are similar functions because they all return the last value inserted into the IDENTITY column of a table.

@@IDENTITY and SCOPE\_IDENTITY return the last identity value generated in any table in the current session. However, SCOPE\_IDENTITY returns the value only within the current scope; @@IDENTITY is not limited to a specific scope.

IDENT\_CURRENT is not limited by scope and session; it is limited to a specified table. IDENT\_CURRENT returns the identity value generated for a specific table in any session and any scope. For more information, see [IDENT\\_CURRENT \(Transact-SQL\)](#).

The scope of the @@IDENTITY function is current session on the local server on which it is executed. This function cannot be applied to remote or linked servers. To obtain an identity value on a different server, execute a stored procedure on that remote or linked server and have that stored procedure (which is executing in the context of the remote or linked server) gather the identity value and return it to the calling connection on the local server.

Replication may affect the @@IDENTITY value, since it is used within the replication triggers and stored

procedures. @@IDENTITY is not a reliable indicator of the most recent user-created identity if the column is part of a replication article. You can use the SCOPE\_IDENTITY() function syntax instead of @@IDENTITY. For more information, see [SCOPE\\_IDENTITY \(Transact-SQL\)](#)

#### NOTE

The calling stored procedure or Transact-SQL statement must be rewritten to use the `SCOPE_IDENTITY()` function, which returns the latest identity used within the scope of that user statement, and not the identity within the scope of the nested trigger used by replication.

## Examples

The following example inserts a row into a table with an identity column (`LocationID`) and uses `@@IDENTITY` to display the identity value used in the new row.

```
USE AdventureWorks2012;
GO
--Display the value of LocationID in the last row in the table.
SELECT MAX(LocationID) FROM Production.Location;
GO
INSERT INTO Production.Location (Name, CostRate, Availability, ModifiedDate)
VALUES ('Damaged Goods', 5, 2.5, GETDATE());
GO
SELECT @@IDENTITY AS 'Identity';
GO
--Display the value of LocationID of the newly inserted row.
SELECT MAX(LocationID) FROM Production.Location;
GO
```

## See Also

[System Functions \(Transact-SQL\)](#)

[CREATE TABLE \(Transact-SQL\)](#)

[IDENT\\_CURRENT \(Transact-SQL\)](#)

[INSERT \(Transact-SQL\)](#)

[SCOPE\\_IDENTITY \(Transact-SQL\)](#)

[SELECT \(Transact-SQL\)](#)

# @@PACK\_RECEIVED (Transact-SQL)

8/29/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the number of input packets read from the network by SQL Server since it was last started.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@PACK_RECEIVED
```

## Return Types

**integer**

## Remarks

To display a report containing several SQL Server statistics, including packets sent and received, run **sp\_monitor**.

## Examples

The following example shows the usage of `@@PACK_RECEIVED`.

```
SELECT @@PACK_RECEIVED AS 'Packets Received';
```

Here is a sample result set.

Packets Received
-----
128

## See Also

[@@PACK\\_SENT](#)  
[sp\\_monitor](#)  
[System Statistical Functions](#)

# @@ROWCOUNT (Transact-SQL)

8/30/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the number of rows affected by the last statement. If the number of rows is more than 2 billion, use [ROWCOUNT\\_BIG](#).

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@ROWCOUNT
```

## Return Types

**int**

## Remarks

Transact-SQL statements can set the value in @@ROWCOUNT in the following ways:

- Set @@ROWCOUNT to the number of rows affected or read. Rows may or may not be sent to the client.
- Preserve @@ROWCOUNT from the previous statement execution.
- Reset @@ROWCOUNT to 0 but do not return the value to the client.

Statements that make a simple assignment always set the @@ROWCOUNT value to 1. No rows are sent to the client. Examples of these statements are: SET @*local\_variable*, RETURN, READTEXT, and select without query statements such as SELECT GETDATE() or SELECT 'Generic Text'.

Statements that make an assignment in a query or use RETURN in a query set the @@ROWCOUNT value to the number of rows affected or read by the query, for example: SELECT @*local\_variable* = c1 FROM t1.

Data manipulation language (DML) statements set the @@ROWCOUNT value to the number of rows affected by the query and return that value to the client. The DML statements may not send any rows to the client.

DECLARE CURSOR and FETCH set the @@ROWCOUNT value to 1.

EXECUTE statements preserve the previous @@ROWCOUNT.

Statements such as USE, SET <option>, DEALLOCATE CURSOR, CLOSE CURSOR, BEGIN TRANSACTION, or COMMIT TRANSACTION reset the ROWCOUNT value to 0.

Natively compiled stored procedures preserve the previous @@ROWCOUNT. Transact-SQL statements inside natively compiled stored procedures do not set @@ROWCOUNT. For more information, see [Natively Compiled Stored Procedures](#).

## Examples

The following example executes an `UPDATE` statement and uses `@@ROWCOUNT` to detect if any rows were changed.

```
USE AdventureWorks2012;
GO
UPDATE HumanResources.Employee
SET JobTitle = N'Executive'
WHERE NationalIDNumber = 123456789
IF @@ROWCOUNT = 0
PRINT 'Warning: No rows were updated';
GO
```

## See Also

[System Functions \(Transact-SQL\)](#)

[SET ROWCOUNT \(Transact-SQL\)](#)

# @@TRANCOUNT (Transact-SQL)

8/29/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the number of BEGIN TRANSACTION statements that have occurred on the current connection.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
@@TRANCOUNT
```

## Return Types

**integer**

## Remarks

The BEGIN TRANSACTION statement increments @@TRANCOUNT by 1. ROLLBACK TRANSACTION decrements @@TRANCOUNT to 0, except for ROLLBACK TRANSACTION *savepoint\_name*, which does not affect @@TRANCOUNT. COMMIT TRANSACTION or COMMIT WORK decrement @@TRANCOUNT by 1.

## Examples

### A. Showing the effects of the BEGIN and COMMIT statements

The following example shows the effect that nested `BEGIN` and `COMMIT` statements have on the `@@TRANCOUNT` variable.

```
PRINT @@TRANCOUNT
-- The BEGIN TRAN statement will increment the
-- transaction count by 1.
BEGIN TRAN
    PRINT @@TRANCOUNT
    BEGIN TRAN
        PRINT @@TRANCOUNT
    -- The COMMIT statement will decrement the transaction count by 1.
    COMMIT
    PRINT @@TRANCOUNT
COMMIT
PRINT @@TRANCOUNT
--Results
--0
--1
--2
--1
--0
```

### B. Showing the effects of the BEGIN and ROLLBACK statements

The following example shows the effect that nested `BEGIN TRAN` and `ROLLBACK` statements have on the `@@TRANCOUNT` variable.

```
PRINT @@TRANCOUNT
-- The BEGIN TRAN statement will increment the
-- transaction count by 1.
BEGIN TRAN
    PRINT @@TRANCOUNT
    BEGIN TRAN
        PRINT @@TRANCOUNT
-- The ROLLBACK statement will clear the @@TRANCOUNT variable
-- to 0 because all active transactions will be rolled back.
ROLLBACK
PRINT @@TRANCOUNT
--Results
--0
--1
--2
--0
```

## See Also

[BEGIN TRANSACTION \(Transact-SQL\)](#)  
[COMMIT TRANSACTION \(Transact-SQL\)](#)  
[ROLLBACK TRANSACTION \(Transact-SQL\)](#)  
[System Functions \(Transact-SQL\)](#)

# BINARY\_CHECKSUM (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the binary checksum value computed over a row of a table or over a list of expressions.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
BINARY_CHECKSUM ( * | expression [ ,...n ] )
```

## Arguments

`*`

Specifies that the computation is over all the columns of the table. BINARY\_CHECKSUM ignores columns of noncomparable data types in its computation. Noncomparable data types include **text**, **ntext**, **image**, **cursor**, **xml**, and noncomparable common language runtime (CLR) user-defined types.

*expression*

Is an [expression](#) of any type. BINARY\_CHECKSUM ignores expressions of noncomparable data types in its computation.

## Remarks

BINARY\_CHECKSUM(\*), computed on any row of a table, returns the same value as long the row is not subsequently modified. BINARY\_CHECKSUM satisfies the properties of a hash function: BINARY\_CHECKSUM applied over any two lists of expressions returns the same value if the corresponding elements of the two lists have the same type and are equal when compared using the equals (=) operator. For this definition, null values of a specified type are considered to compare as equal. If one of the values in the expression list changes, the checksum of the list also generally changes. However, there is a small chance that the checksum will not change. For this reason, we do not recommend using BINARY\_CHECKSUM to detect whether values have changed, unless your application can tolerate occasionally missing a change. Consider using HashBytes instead. When an MD5 hash algorithm is specified, the probability of HashBytes returning the same result for two different inputs is much lower than that of BINARY\_CHECKSUM.

BINARY\_CHECKSUM can be applied over a list of expressions, and returns the same value for a specified list. BINARY\_CHECKSUM applied over any two lists of expressions returns the same value if the corresponding elements of the two lists have the same type and byte representation. For this definition, null values of a specified type are considered to have the same byte representation.

BINARY\_CHECKSUM and CHECKSUM are similar functions: They can be used to compute a checksum value on a list of expressions, and the order of expressions affects the resultant value. The order of columns used for BINARY\_CHECKSUM(\*) is the order of columns specified in the table or view definition. These include computed columns.

CHECKSUM and BINARY\_CHECKSUM return different values for the string data types, where locale can cause strings with different representation to compare equal. The string data types are **char**, **varchar**, **nchar**, **nvarchar**, or **sql\_variant** (if the base type of **sql\_variant** is a string data type). For example, the BINARY\_CHECKSUM values

for the strings "McCavity" and "Mccavity" are different. In contrast, in a case-insensitive server, CHECKSUM returns the same checksum values for those strings. CHECKSUM values should not be compared with BINARY\_CHECKSUM values.

BINARY\_CHECKSUM supports up to 8,000 characters of type **varbinary(max)** and up to 255 characters of type **nvarchar(max)**.

## Examples

The following example uses `BINARY_CHECKSUM` to detect changes in a row of a table.

```
USE AdventureWorks2012;
GO
CREATE TABLE myTable (column1 int, column2 varchar(256));
GO
INSERT INTO myTable VALUES (1, 'test');
GO
SELECT BINARY_CHECKSUM(*) from myTable;
GO
UPDATE myTable set column2 = 'TEST';
GO
SELECT BINARY_CHECKSUM(*) from myTable;
GO
```

## See also

[Aggregate Functions \(Transact-SQL\)](#)

[CHECKSUM \(Transact-SQL\)](#)

[CHECKSUM\\_AGG \(Transact-SQL\)](#)

# CHECKSUM (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the checksum value computed over a row of a table, or over a list of expressions. CHECKSUM is intended for use in building hash indexes.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CHECKSUM ( * | expression [ ,...n ] )
```

## Arguments

\*

Specifies that computation is over all the columns of the table. CHECKSUM returns an error if any column is of noncomparable data type. Noncomparable data types are **text**, **ntext**, **image**, XML, and **cursor**, and also **sql\_variant** with any one of the preceding types as its base type.

*expression*

Is an [expression](#) of any type except a noncomparable data type.

## Return types

**int**

## Remarks

CHECKSUM computes a hash value, called the checksum, over its list of arguments. The hash value is intended for use in building hash indexes. If the arguments to CHECKSUM are columns, and an index is built over the computed CHECKSUM value, the result is a hash index. This can be used for equality searches over the columns.

CHECKSUM satisfies the properties of a hash function: CHECKSUM applied over any two lists of expressions returns the same value if the corresponding elements of the two lists have the same type and are equal when compared using the equals (=) operator. For this definition, null values of a specified type are considered to compare as equal. If one of the values in the expression list changes, the checksum of the list also generally changes. However, there is a small chance that the checksum will not change. For this reason, we do not recommend using CHECKSUM to detect whether values have changed, unless your application can tolerate occasionally missing a change. Consider using [HashBytes](#) instead. When an MD5 hash algorithm is specified, the probability of HashBytes returning the same result for two different inputs is much lower than that of CHECKSUM.

The order of expressions affects the resultant value of CHECKSUM. The order of columns used with CHECKSUM(\*) is the order of columns specified in the table or view definition. This includes computed columns.

The CHECKSUM value is dependent upon the collation. The same value stored with a different collation will return a different CHECKSUM value.

## Examples

The following examples show using `CHECKSUM` to build hash indexes. The hash index is built by adding a computed checksum column to the table being indexed, and then building an index on the checksum column.

```
-- Create a checksum index.  
SET ARITHABORT ON;  
USE AdventureWorks2012;  
GO  
ALTER TABLE Production.Product  
ADD cs_Pname AS CHECKSUM(Name);  
GO  
CREATE INDEX Pname_index ON Production.Product (cs_Pname);  
GO
```

The checksum index can be used as a hash index, particularly to improve indexing speed when the column to be indexed is a long character column. The checksum index can be used for equality searches.

```
/*Use the index in a SELECT query. Add a second search  
condition to catch stray cases where checksums match,  
but the values are not the same.*/  
SELECT *  
FROM Production.Product  
WHERE CHECKSUM(N'Bearing Ball') = cs_Pname  
AND Name = N'Bearing Ball';  
GO
```

Creating the index on the computed column materializes the checksum column, and any changes to the `ProductName` value will be propagated to the checksum column. Alternatively, an index could be built directly on the column indexed. However, if the key values are long, a regular index is not likely to perform as well as a checksum index.

## See also

[CHECKSUM\\_AGG \(Transact-SQL\)](#)  
[HASHBYTES \(Transact-SQL\)](#)  
[BINARY\\_CHECKSUM \(Transact-SQL\)](#)

# COMPRESS (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Compresses the input expression using the GZIP algorithm. The result of the compression is byte array of type **varbinary(max)**.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
COMPRESS ( expression )
```

## Arguments

*expression*

Is a **nvarchar(n)**, **nvarchar(max)**, **varchar(n)**, **varchar(max)**, **varbinary(n)**, **varbinary(max)**, **char(n)**, **nchar(n)**, or **binary(n)** expression. For more information, see [Expressions \(Transact-SQL\)](#).

## Return types

Returns the data type of **varbinary(max)** that represents the compressed content of input.

## Remarks

Compressed data cannot be indexed.

The COMPRESS function compresses the data provided as the input expression and must be invoked for each section of data to be compressed. For automatic compression at the row or page level during storage, see [Data Compression](#).

## Examples

### A. Compress Data During the Table Insert

The following example shows how to compress data inserted into table:

```
INSERT INTO player (name, surname, info )
VALUES (N'Ovidiu', N'Craciun',
        COMPRESS(N'{"sport":"Tennis","age": 28,"rank":1,"points":15258, "turn":17}'));
INSERT INTO player (name, surname, info )
VALUES (N'Michael', N'Raheem', compress(@info));
```

### B. Archive compressed version of deleted rows

The following statement deletes old player records from the `player` table and stores the records in the `inactivePlayer` table in a compressed format to save space.

```
DELETE player
WHERE datemodified < @startOfYear
OUTPUT id, name, surname datemodifier, COMPRESS(info)
INTO dbo.inactivePlayers ;
```

## See also

[String Functions \(Transact-SQL\)](#)

[DECOMPRESS \(Transact-SQL\)](#)

# CONNECTIONPROPERTY (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns information about the connection properties for the unique connection that a request came in on.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CONNECTIONPROPERTY ( property )
```

## Arguments

### *property*

Is the property of the connection. *property* can be one of the following values.

VALUE	DATA TYPE	DESCRIPTION
net_transport	<b>nvarchar(40)</b>	Returns the physical transport protocol that is used by this connection. Is not nullable.  Return values are: <b>HTTP</b> , <b>Named pipe</b> , <b>Session</b> , <b>Shared memory</b> , <b>SSL</b> , <b>TCP</b> , and <b>VIA</b> .  Note: Always returns <b>Session</b> when a connection has multiple active result sets (MARS) enabled, and connection pooling is enabled.
protocol_type	<b>nvarchar(40)</b>	Returns the protocol type of the payload. It currently distinguishes between TDS (TSQL) and SOAP. Is nullable.
auth_scheme	<b>nvarchar(40)</b>	Returns the SQL Server Authentication scheme for a connection. The authentication scheme is either Windows Authentication (NTLM, KERBEROS, DIGEST, BASIC, NEGOTIATE) or SQL Server Authentication. Is not nullable.
local_net_address	<b>varchar(48)</b>	Returns the IP address on the server that this connection targeted. Available only for connections that are using the TCP transport provider. Is nullable.

VALUE	DATA TYPE	DESCRIPTION
local_tcp_port	<b>int</b>	Returns the server TCP port that this connection targeted if the connection were a connection that is using the TCP transport. Is nullable.
client_net_address	<b>varchar(48)</b>	Asks for the address of the client that is connecting to this server. Is nullable.
physical_net_transport	<b>nvarchar(40)</b>	Returns the physical transport protocol that is used by this connection. Accurate when a connection has multiple active result sets (MARS) enabled.
<Any other string>		Returns NULL if the input is not valid.

## Remarks

**local\_net\_address** and **local\_tcp\_port** return NULL in SQL Database.

The values that are returned are the same as the options shown for the corresponding columns in the [sys.dm\\_exec\\_connections](#) dynamic management view. For example:

```
SELECT
ConnectionProperty('net_transport') AS 'Net transport',
ConnectionProperty('protocol_type') AS 'Protocol type';
```

## See also

[sys.dm\\_exec\\_sessions \(Transact-SQL\)](#)  
[sys.dm\\_exec\\_requests \(Transact-SQL\)](#)

# CONTEXT\_INFO (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:**  SQL Server (starting with 2008)  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

Returns the **context\_info** value that was set for the current session or batch by using the [SET CONTEXT\\_INFO](#) statement.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
CONTEXT_INFO()
```

## Return value

The value of **context\_info**.

If **context\_info** was not set:

- In SQL Server returns NULL.
- In SQL Database returns a unique session-specific GUID.

## Remarks

Multiple active result sets (MARS) enables applications to run multiple batches, or requests, at the same time on the same connection. When one of the batches on a MARS connection runs SET CONTEXT\_INFO, the new context value is returned by the CONTEXT\_INFO function when it is run in the same batch as the SET statement. The new value is not returned by the CONTEXT\_INFO function run in one or more of the other batches on the connection, unless they started after the batch that ran the SET statement completed.

## Permissions

Requires no special permissions. The context information is also stored in the **sys.dm\_exec\_requests**, **sys.dm\_exec\_sessions**, and **sys.sysprocesses** system views, but querying the views directly requires SELECT and VIEW SERVER STATE permissions.

## Examples

The following simple example sets the **context\_info** value to `0x1256698456`, and then uses the `CONTEXT_INFO` function to retrieve the value.

```
SET CONTEXT_INFO 0x1256698456;
GO
SELECT CONTEXT_INFO();
GO
```

## See also

[SET CONTEXT\\_INFO \(Transact-SQL\)](#)

# CURRENT\_REQUEST\_ID (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the ID of the current request within the current session.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CURRENT_REQUEST_ID()
```

## Return types

**smallint**

## Remarks

To find exact information about the current session and current request, use @@SPID and CURRENT\_REQUEST\_ID(), respectively.

## See also

[@@SPID \(Transact-SQL\)](#)

# CURRENT\_TRANSACTION\_ID (Transact-SQL)

7/31/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the transaction ID of the current transaction in the current session.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
CURRENT_TRANSACTION_ID( )
```

## Return types

**bigint**

## Return Value

Transaction ID of the current transaction in the current session, taken from [sys.dm\\_tran\\_current\\_transaction \(Transact-SQL\)](#).

## Permissions

Any user can return the transaction ID of the current session.

## Examples

The following example returns the transaction ID of the current session:

```
SELECT CURRENT_TRANSACTION_ID();
```

## See also

[sp\\_set\\_session\\_context \(Transact-SQL\)](#)

[SESSION\\_CONTEXT \(Transact-SQL\)](#)

[Row-Level Security](#)

[CONTEXT\\_INFO \(Transact-SQL\)](#)

[SET CONTEXT\\_INFO \(Transact-SQL\)](#)

# DECOMPRESS (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2016) Azure SQL Database Azure SQL Data

Warehouse Parallel Data Warehouse

Decompress input expression using GZIP algorithm. Result of the compression is byte array (VARBINARY(MAX) type).

[Transact-SQL Syntax Conventions](#)

## Syntax

```
DECOMPRESS ( expression )
```

## Arguments

*expression*

Is a **varbinary(n)**, **varbinary(max)**, or **binary(n)**. For more information, see [Expressions \(Transact-SQL\)](#).

## Return Types

Returns the data type of **varbinary(max)** type. The input argument is decompressed using the ZIP algorithm. The user should explicitly cast result to a target type if needed.

## Remarks

## Examples

### A. Decompress Data at Query Time

The following example shows how to show compress data from a table:

```
SELECT _id, name, surname, datemodified,
       CAST(DECOMPRESS(info) AS NVARCHAR(MAX)) AS info
  FROM player;
```

### B. Display Compressed Data Using Computed Column

The following example shows how to create a table to store decompressed data:

```
CREATE TABLE (
    _id int primary key identity,
    name nvarchar(max),
    surname nvarchar(max),
    info varbinary(max),
    info_json as CAST(decompress(info) as nvarchar(max))
);
```

## See Also

[String Functions \(Transact-SQL\)](#)

[COMPRESS \(Transact-SQL\)](#)

# ERROR\_LINE (Transact-SQL)

3/24/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✖ Azure SQL Data Warehouse ✖ Parallel Data Warehouse

Returns the line number at which an error occurred that caused the CATCH block of a TRY...CATCH construct to be run.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ERROR_LINE ( )
```

## Return Type

**int**

## Return Value

When called in a CATCH block:

- Returns the line number at which the error occurred.
- Returns the line number in a routine if the error occurred within a stored procedure or trigger.

Returns NULL if called outside the scope of a CATCH block.

## Remarks

This function may be called anywhere within the scope of a CATCH block.

ERROR\_LINE returns the line number at which the error occurred regardless of the number of times it is called or where it is called within the scope of the CATCH block. This contrasts with functions, such as @@ERROR, which return an error number in the statement immediately following the one that causes an error or in the first statement of a CATCH block.

In nested CATCH blocks, ERROR\_LINE returns the error line number specific to the scope of the CATCH block in which it is referenced. For example, the CATCH block of a TRY...CATCH construct could contain a nested TRY...CATCH construct. Within the nested CATCH block, ERROR\_LINE returns the line number for the error that invoked the nested CATCH block. If ERROR\_LINE is run in the outer CATCH block, it returns the line number for the error that invoked that CATCH block.

## Examples

### A. Using ERROR\_LINE in a CATCH block

The following code example shows a `SELECT` statement that generates a divide-by-zero error. The line number at which the error occurred is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT ERROR_LINE() AS ErrorLine;
END CATCH;
GO
```

## B. Using ERROR\_LINE in a CATCH block with a stored procedure

The following code example shows a stored procedure that will generate a divide-by-zero error. `ERROR_LINE` returns the line number in the stored procedure in which the error occurred.

```
-- Verify that the stored procedure does not already exist.
IF OBJECT_ID ( 'usp_ExampleProc', 'P' ) IS NOT NULL
    DROP PROCEDURE usp_ExampleProc;
GO

-- Create a stored procedure that
-- generates a divide-by-zero error.
CREATE PROCEDURE usp_ExampleProc
AS
    SELECT 1/0;
GO

BEGIN TRY
    -- Execute the stored procedure inside the TRY block.
    EXECUTE usp_ExampleProc;
END TRY
BEGIN CATCH
    SELECT ERROR_LINE() AS ErrorLine;
END CATCH;
GO
```

## C. Using ERROR\_LINE in a CATCH block with other error-handling tools

The following code example shows a `SELECT` statement that generates a divide-by-zero error. Along with the line number at which the error occurred, information that relates to the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

## See Also

[TRY...CATCH \(Transact-SQL\)](#)

[sys.messages \(Transact-SQL\)](#)

[ERROR\\_NUMBER \(Transact-SQL\)](#)

[ERROR\\_MESSAGE \(Transact-SQL\)](#)

[ERROR\\_PROCEDURE \(Transact-SQL\)](#)

[ERROR\\_SEVERITY \(Transact-SQL\)](#)

[ERROR\\_STATE \(Transact-SQL\)](#)

[RAISERROR \(Transact-SQL\)](#)

[@@ERROR \(Transact-SQL\)](#)

# ERROR\_MESSAGE (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the message text of the error that caused the CATCH block of a TRY...CATCH construct to be run.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ERROR_MESSAGE ( )
```

## Return Types

**nvarchar(4000)**

## Return Value

When called in a CATCH block, returns the complete text of the error message that caused the CATCH block to be run. The text includes the values supplied for any substitutable parameters, such as lengths, object names, or times.

Returns NULL if called outside the scope of a CATCH block.

## Remarks

ERROR\_MESSAGE may be called anywhere within the scope of a CATCH block.

ERROR\_MESSAGE returns the error message regardless of how many times it is run, or where it is run within the scope of the CATCH block. This is in contrast to functions like @@ERROR, which only returns an error number in the statement immediately after the one that causes an error, or the first statement of a CATCH block.

In nested CATCH blocks, ERROR\_MESSAGE returns the error message specific to the scope of the CATCH block in which it is referenced. For example, the CATCH block of an outer TRY...CATCH construct could have a nested TRY...CATCH construct. Within the nested CATCH block, ERROR\_MESSAGE returns the message from the error that invoked the nested CATCH block. If ERROR\_MESSAGE is run in the outer CATCH block, it returns the message from the error that invoked that CATCH block.

## Examples

### A. Using ERROR\_MESSAGE in a CATCH block

The following code example shows a `SELECT` statement that generates a divide-by-zero error. The message of the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

## B. Using ERROR\_MESSAGE in a CATCH block with other error-handling tools

The following code example shows a `SELECT` statement that generates a divide-by-zero error. Along with the error message, information that relates to the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber
        ,ERROR_SEVERITY() AS ErrorSeverity
        ,ERROR_STATE() AS ErrorState
        ,ERROR_PROCEDURE() AS ErrorProcedure
        ,ERROR_LINE() AS ErrorLine
        ,ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Using ERROR\_MESSAGE in a CATCH block

The following code example shows a `SELECT` statement that generates a divide-by-zero error. The message of the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

### D. Using ERROR\_MESSAGE in a CATCH block with other error-handling tools

The following code example shows a `SELECT` statement that generates a divide-by-zero error. Along with the error message, information that relates to the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber
        ,ERROR_SEVERITY() AS ErrorSeverity
        ,ERROR_STATE() AS ErrorState
        ,ERROR_PROCEDURE() AS ErrorProcedure
        ,ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

## See Also

[sys.messages \(Transact-SQL\)](#)  
[TRY...CATCH \(Transact-SQL\)](#)  
[ERROR\\_LINE \(Transact-SQL\)](#)  
[ERROR\\_NUMBER \(Transact-SQL\)](#)  
[ERROR\\_PROCEDURE \(Transact-SQL\)](#)  
[ERROR\\_SEVERITY \(Transact-SQL\)](#)  
[ERROR\\_STATE \(Transact-SQL\)](#)  
[RAISERROR \(Transact-SQL\)](#)  
[@@ERROR \(Transact-SQL\)](#)

# ERROR\_NUMBER (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the error number of the error that caused the CATCH block of a TRY...CATCH construct to be run.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ERROR_NUMBER ( )
```

## Return Types

**int**

## Return Value

When called in a CATCH block, returns the error number of the error message that caused the CATCH block to be run.

Returns NULL if called outside the scope of a CATCH block.

## Remarks

This function may be called anywhere within the scope of a CATCH block.

ERROR\_NUMBER returns the error number regardless of how many times it is run, or where it is run within the scope of the CATCH block. This is in contrast to @@ERROR, which only returns the error number in the statement immediately after the one that causes an error, or the first statement of a CATCH block.

In nested CATCH blocks, ERROR\_NUMBER returns the error number specific to the scope of the CATCH block in which it is referenced. For example, the CATCH block of an outer TRY...CATCH construct could have a nested TRY...CATCH construct. Within the nested CATCH block, ERROR\_NUMBER returns the number from the error that invoked the nested CATCH block. If ERROR\_NUMBER is run in the outer CATCH block, it returns the number from the error that invoked that CATCH block.

## Examples

### A. Using ERROR\_NUMBER in a CATCH block

The following code example shows a `SELECT` statement that generates a divide-by-zero error. The number of the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrorNumber;
END CATCH;
GO
```

## B. Using `ERROR_NUMBER` in a CATCH block with other error-handling tools

The following code example shows a `SELECT` statement that generates a divide-by-zero error. Along with the error number, information that relates to the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Using `ERROR_NUMBER` in a CATCH block

The following code example shows a `SELECT` statement that generates a divide-by-zero error. The number of the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrorNumber;
END CATCH;
GO
```

### D. Using `ERROR_NUMBER` in a CATCH block with other error-handling tools

The following code example shows a `SELECT` statement that generates a divide-by-zero error. Along with the error number, information that relates to the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

## See Also

[sys.messages \(Transact-SQL\)](#)  
[TRY...CATCH \(Transact-SQL\)](#)  
[ERROR\\_LINE \(Transact-SQL\)](#)  
[ERROR\\_MESSAGE \(Transact-SQL\)](#)  
[ERROR\\_PROCEDURE \(Transact-SQL\)](#)  
[ERROR\\_SEVERITY \(Transact-SQL\)](#)  
[ERROR\\_STATE \(Transact-SQL\)](#)  
[RAISERROR \(Transact-SQL\)](#)  
[@@ERROR \(Transact-SQL\)](#)

# ERROR\_PROCEDURE (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the name of the stored procedure or trigger where an error occurred that caused the CATCH block of a TRY...CATCH construct to be run.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ERROR_PROCEDURE ( )
```

## Return Types

**nvarchar(128)**

## Return Value

When called in a CATCH block, returns the stored procedure name where the error occurred.

Returns NULL if the error did not occur within a stored procedure or trigger.

Returns NULL if called outside the scope of a CATCH block.

## Remarks

ERROR\_PROCEDURE may be called anywhere within the scope of a CATCH block.

ERROR\_PROCEDURE returns the name of the stored procedure or trigger where the error occurred, regardless of the number of times it is called or where it is called within the scope of the CATCH block. This contrasts with functions, such as @@ERROR, which return the error number in the statement immediately following the one that caused the error or in the first statement of the CATCH block.

In nested CATCH blocks, ERROR\_PROCEDURE returns the name of the stored procedure or trigger specific to the scope of the CATCH block in which it is referenced. For example, the CATCH block of a TRY...CATCH construct could have a nested TRY...CATCH. Within the nested CATCH block, ERROR\_PROCEDURE returns the name of the stored procedure or trigger where the error occurred that invoked the nested CATCH block. If

ERROR\_PROCEDURE is run in the outer CATCH block, it returns the name of the stored procedure or trigger where the error occurred that invoked that CATCH block.

## Examples

### A. Using ERROR\_PROCEDURE in a CATCH block

The following code example shows a stored procedure that generates a divide-by-zero error. `ERROR_PROCEDURE` returns the name of the stored procedure in which the error occurred.

```

-- Verify that the stored procedure does not already exist.
IF OBJECT_ID ( 'usp_ExampleProc', 'P' ) IS NOT NULL
    DROP PROCEDURE usp_ExampleProc;
GO

-- Create a stored procedure that
-- generates a divide-by-zero error.
CREATE PROCEDURE usp_ExampleProc
AS
    SELECT 1/0;
GO

BEGIN TRY
    -- Execute the stored procedure inside the TRY block.
    EXECUTE usp_ExampleProc;
END TRY
BEGIN CATCH
    SELECT ERROR_PROCEDURE() AS ErrorProcedure;
END CATCH;
GO

```

## B. Using `ERROR_PROCEDURE` in a `CATCH` block with other error-handling tools

The following code example shows a stored procedure that generates a divide-by-zero error. Along with the name of the stored procedure in which the error occurred, information that relates to the error is returned.

```

-- Verify that the stored procedure does not already exist.
IF OBJECT_ID ( 'usp_ExampleProc', 'P' ) IS NOT NULL
    DROP PROCEDURE usp_ExampleProc;
GO

-- Create a stored procedure that
-- generates a divide-by-zero error.
CREATE PROCEDURE usp_ExampleProc
AS
    SELECT 1/0;
GO

BEGIN TRY
    -- Execute the stored procedure inside the TRY block.
    EXECUTE usp_ExampleProc;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_MESSAGE() AS ErrorMessage,
        ERROR_LINE() AS ErrorLine;
    END CATCH;
GO

```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Using `ERROR_PROCEDURE` in a `CATCH` block

The following code example shows a stored procedure that generates a divide-by-zero error. `ERROR_PROCEDURE` returns the name of the stored procedure in which the error occurred.

```

-- Verify that the stored procedure does not already exist.
IF OBJECT_ID ( 'usp_ExampleProc', 'P' ) IS NOT NULL
    DROP PROCEDURE usp_ExampleProc;
GO

-- Create a stored procedure that
-- generates a divide-by-zero error.
CREATE PROCEDURE usp_ExampleProc
AS
    SELECT 1/0;
GO

BEGIN TRY
    -- Execute the stored procedure inside the TRY block.
    EXECUTE usp_ExampleProc;
END TRY
BEGIN CATCH
    SELECT ERROR_PROCEDURE() AS ErrorProcedure;
END CATCH;
GO

```

#### D. Using `ERROR_PROCEDURE` in a `CATCH` block with other error-handling tools

The following code example shows a stored procedure that generates a divide-by-zero error. Along with the name of the stored procedure in which the error occurred, information that relates to the error is returned.

```

-- Verify that the stored procedure does not already exist.
IF OBJECT_ID ( 'usp_ExampleProc', 'P' ) IS NOT NULL
    DROP PROCEDURE usp_ExampleProc;
GO

-- Create a stored procedure that
-- generates a divide-by-zero error.
CREATE PROCEDURE usp_ExampleProc
AS
    SELECT 1/0;
GO

BEGIN TRY
    -- Execute the stored procedure inside the TRY block.
    EXECUTE usp_ExampleProc;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_MESSAGE() AS ErrorMessage;
    END CATCH;
GO

```

## See Also

[sys.messages \(Transact-SQL\)](#)  
[TRY...CATCH \(Transact-SQL\)](#)  
[ERROR\\_LINE \(Transact-SQL\)](#)  
[ERROR\\_MESSAGE \(Transact-SQL\)](#)  
[ERROR\\_NUMBER \(Transact-SQL\)](#)  
[ERROR\\_SEVERITY \(Transact-SQL\)](#)  
[ERROR\\_STATE \(Transact-SQL\)](#)

RAISERROR (Transact-SQL)

@@ERROR (Transact-SQL)

# ERROR\_SEVERITY (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the severity of the error that caused the CATCH block of a TRY...CATCH construct to be run.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
ERROR_SEVERITY ( )
```

## Return Types

**int**

## Return Value

When called in a CATCH block, returns the severity of the error message that caused the CATCH block to be run.

Returns NULL if called outside the scope of a CATCH block.

## Remarks

ERROR\_SEVERITY may be called anywhere within the scope of a CATCH block.

ERROR\_SEVERITY returns the error severity regardless of how many times it is run, or where it is run within the scope of the CATCH block. This is in contrast to functions like @@ERROR, which only returns the error number in the statement immediately after the one that causes an error, or in the first statement of a CATCH block.

In nested CATCH blocks, ERROR\_SEVERITY returns the error severity specific to the scope of the CATCH block in which it is referenced. For example, the CATCH block of an outer TRY...CATCH construct could have a nested TRY...CATCH construct. Within the nested CATCH block, ERROR\_SEVERITY returns the severity from the error that invoked the nested CATCH block. If ERROR\_SEVERITY is run in the outer CATCH block, it returns the severity from the error that invoked that CATCH block.

## Examples

### A. Using ERROR\_SEVERITY in a CATCH block

The following example shows a `SELECT` statement that generates a divide-by-zero error. The severity of the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT ERROR_SEVERITY() AS ErrorSeverity;
END CATCH;
GO
```

## B. Using ERROR\_SEVERITY in a CATCH block with other error-handling tools

The following example shows a `SELECT` statement that generates a divide by zero error. Along with the severity, information that relates to the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

# Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

## C. Using ERROR\_SEVERITY in a CATCH block

The following example shows a `SELECT` statement that generates a divide-by-zero error. The severity of the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT ERROR_SEVERITY() AS ErrorSeverity;
END CATCH;
GO
```

## D. Using ERROR\_SEVERITY in a CATCH block with other error-handling tools

The following example shows a `SELECT` statement that generates a divide by zero error. Along with the severity, information that relates to the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

## See Also

[sys.messages \(Transact-SQL\)](#)  
[TRY...CATCH \(Transact-SQL\)](#)  
[ERROR\\_LINE \(Transact-SQL\)](#)  
[ERROR\\_MESSAGE \(Transact-SQL\)](#)  
[ERROR\\_NUMBER \(Transact-SQL\)](#)  
[ERROR\\_PROCEDURE \(Transact-SQL\)](#)  
[ERROR\\_STATE \(Transact-SQL\)](#)  
[RAISERROR \(Transact-SQL\)](#)  
[@@ERROR \(Transact-SQL\)](#)

# ERROR\_STATE (Transact-SQL)

3/24/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✖ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Returns the state number of the error that caused the CATCH block of a TRY...CATCH construct to be run.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
-- Syntax for SQL Server, Azure SQL Data Warehouse, Parallel Data Warehouse  
ERROR_STATE ( )
```

## Return Types

**int**

## Return Value

When called in a CATCH block, returns the state number of the error message that caused the CATCH block to be run.

Returns NULL if called outside the scope of a CATCH block.

## Remarks

Some error messages can be raised at multiple points in the code for the Microsoft SQL Server Database Engine. For example, an "1105" error can be raised for several different conditions. Each specific condition that raises the error assigns a unique state code.

When viewing databases of known issues, such as the Microsoft Knowledge Base, you can use the state number to determine if the recorded issue might be the same as the error you have encountered. For example, if a Knowledge Base article discusses an 1105 error message with a state of 2, and the 1105 error message you received had a state of 3, your error probably had a different cause than the one reported in the article.

A SQL Server support engineer can also use the state code from an error to find the location in the source code where that error is being raised, which may provide additional ideas on how to diagnose the problem.

ERROR\_STATE may be called anywhere within the scope of a CATCH block.

ERROR\_STATE returns the error state regardless of how many times it is run, or where it is run within the scope of the CATCH block. This is in contrast to functions like @@ERROR, which only returns the error number in the statement immediately after the one that causes an error, or in the first statement of a CATCH block.

In nested CATCH blocks, ERROR\_STATE returns the error state specific to the scope of the CATCH block in which it is referenced. For example, the CATCH block of an outer TRY...CATCH construct could have a nested TRY...CATCH construct. Within the nested CATCH block, ERROR\_STATE returns the state from the error that invoked the nested CATCH block. If ERROR\_STATE is run in the outer CATCH block, it returns the state from the error that invoked that CATCH block.

## Examples

### A. Using ERROR\_STATE in a CATCH block

The following example shows a `SELECT` statement that generates a divide-by-zero error. The state of the error is returned.

```
BEGIN TRY
    -- Generate a divide by zero error
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT ERROR_STATE() AS ErrorState;
END CATCH;
GO
```

### B. Using ERROR\_STATE in a CATCH block with other error-handling tools

The following example shows a `SELECT` statement that generates a divide-by-zero error. Along with the error state, information that relates to the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### C. Using ERROR\_STATE in a CATCH block

The following example shows a `SELECT` statement that generates a divide-by-zero error. The state of the error is returned.

```
BEGIN TRY
    -- Generate a divide by zero error
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT ERROR_STATE() AS ErrorState;
END CATCH;
GO
```

### D. Using ERROR\_STATE in a CATCH block with other error-handling tools

The following example shows a `SELECT` statement that generates a divide-by-zero error. Along with the error state, information that relates to the error is returned.

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

## See Also

[sys.messages \(Transact-SQL\)](#)  
[TRY...CATCH \(Transact-SQL\)](#)  
[ERROR\\_LINE \(Transact-SQL\)](#)  
[ERROR\\_MESSAGE \(Transact-SQL\)](#)  
[ERROR\\_NUMBER \(Transact-SQL\)](#)  
[ERROR\\_PROCEDURE \(Transact-SQL\)](#)  
[ERROR\\_SEVERITY \(Transact-SQL\)](#)  
[RAISERROR \(Transact-SQL\)](#)  
[@@ERROR \(Transact-SQL\)](#)

# FORMATMESSAGE (Transact-SQL)

7/13/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Constructs a message from an existing message in sys.messages or from a provided string. The functionality of FORMATMESSAGE resembles that of the RAISERROR statement. However, RAISERROR prints the message immediately, while FORMATMESSAGE returns the formatted message for further processing.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
FORMATMESSAGE ( { msg_number | 'msg_string' } , [ param_value [ ,...n ] ] )
```

## Arguments

*msg\_number*

Is the ID of the message stored in sys.messages. If *msg\_number* is <= 13000, or if the message does not exist in sys.messages, NULL is returned.

*msg\_string*

**Applies to:** SQL Server ( SQL Server 2016 through [current version](#)).

Is a string enclosed in single quotes and containing parameter value placeholders. The error message can have a maximum of 2,047 characters. If the message contains 2,048 or more characters, only the first 2,044 are displayed and an ellipsis is added to indicate that the message has been truncated. Note that substitution parameters consume more characters than the output shows because of internal storage behavior. For information about the structure of a message string and the use of parameters in the string, see the description of the *msg\_str* argument in [RAISERROR \(Transact-SQL\)](#).

*param\_value*

Is a parameter value for use in the message. Can be more than one parameter value. The values must be specified in the order in which the placeholder variables appear in the message. The maximum number of values is 20.

## Return Types

**nvarchar**

## Remarks

Like the RAISERROR statement, FORMATMESSAGE edits the message by substituting the supplied parameter values for placeholder variables in the message. For more information about the placeholders allowed in error messages and the editing process, see [RAISERROR \(Transact-SQL\)](#).

FORMATMESSAGE looks up the message in the current language of the user. If there is no localized version of the message, the U.S. English version is used.

For localized messages, the supplied parameter values must correspond to the parameter placeholders in the U.S. English version. That is, parameter 1 in the localized version must correspond to parameter 1 in the U.S. English

version, parameter 2 must correspond to parameter 2, and so on.

## Examples

### A. Example with a message number

The following example uses a replication message 20009 stored in sys.messages as, "The article '%s' could not be added to the publication '%s'." FORMATMESSAGE substitutes the values First Variable and Second Variable for the parameter placeholders. The resulting string, "The article 'First Variable' could not be added to the publication 'Second Variable'.", is stored in the local variable @var1 .

```
SELECT text FROM sys.messages WHERE message_id = 20009 AND language_id = 1033;
DECLARE @var1 VARCHAR(200);
SELECT @var1 = FORMATMESSAGE(20009, 'First Variable', 'Second Variable');
SELECT @var1;
```

### B. Example with a message string

**Applies to:** SQL Server ( SQL Server 2016 through [current version](#)).

The following example takes a string as an input.

```
SELECT FORMATMESSAGE('This is the %s and this is the %s.', 'first variable', 'second variable') AS Result;
```

Returns: This is the first variable and this is the second variable.

### C. Additional message string formatting examples

The following examples show a variety of formatting options.

```
SELECT FORMATMESSAGE('Signed int %i, %d %i, %d, %+i, %+d, %i, %+d', 5, -5, 50, -50, -11, -11, 11, 11);
SELECT FORMATMESSAGE('Signed int with leading zero %020i', 5);
SELECT FORMATMESSAGE('Signed int with leading zero 0 %020i', -55);
SELECT FORMATMESSAGE('Unsigned int %u, %u', 50, -50);
SELECT FORMATMESSAGE('Unsigned octal %o, %o', 50, -50);
SELECT FORMATMESSAGE('Unsigned hexadecimal %x, %X, %x, %X, %x', 11, 11, -11, 50, -50);
SELECT FORMATMESSAGE('Unsigned octal with prefix: %#o, %#o', 50, -50);
SELECT FORMATMESSAGE('Unsigned hexadecimal with prefix: %#x, %#X, %#X, %x, %x', 11, 11, -11, 50, -50);
SELECT FORMATMESSAGE('Hello %s!', 'TEST');
SELECT FORMATMESSAGE('Hello %20s!', 'TEST');
SELECT FORMATMESSAGE('Hello %-20s!', 'TEST');
SELECT FORMATMESSAGE('Hello %20s!', 'TEST');
```

## See Also

[THROW \(Transact-SQL\)](#)

[sp\\_addmessage \(Transact-SQL\)](#)

[sys.messages \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

[RAISERROR \(Transact-SQL\)](#)

# GET\_FILESTREAM\_TRANSACTION\_CONTEXT (Transact-SQL)

3/24/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a token that represents the current transaction context of a session. The token is used by an application to bind FILESTREAM file-system streaming operations to the transaction. For a list of FILESTREAM topics, see [Binary Large Object \(Blob\) Data \(SQL Server\)](#).

[Transact-SQL Syntax Conventions](#)

## Syntax

```
GET_FILESTREAM_TRANSACTION_CONTEXT ()
```

## Return Type

**varbinary(max)**

## Return Value

NULL is returned if the transaction has not been started, or has been canceled or committed.

## Remarks

The transaction must be explicit. Use BEGIN TRANSACTION followed by COMMIT TRANSACTION or ROLLBACK TRANSACTION.

When you call GET\_FILESTREAM\_TRANSACTION\_CONTEXT, the caller is granted file system access to the transaction for the duration of the transaction. To allow another user access to the transaction through the file system, use EXECUTE AS to run GET\_FILESTREAM\_TRANSACTION\_CONTEXT as the other user.

## Examples

The following example uses `GET_FILESTREAM_TRANSACTION_CONTEXT` in a Transact-SQL transaction to obtain the transaction context.

```
using System;
using System.Data.SqlClient;
using System.Data;

namespace ConsoleApplication
{
    /// <summary>
    /// This class is a wrapper that contains methods for:
    ///
    ///     GetTransactionContext() - Returns the current transaction context.
    ///     BeginTransaction() - Begins a transaction.
    ///     CommitTransaction() - Commits the current transaction.
}
```

```
///</summary>

class SqlAccessWrapper
{
    /// <summary>
    /// Returns a byte array that contains the current transaction
    /// context.
    /// </summary>
    /// <param name="sqlConnection">
    /// SqlConnection object that represents the instance of SQL Server
    /// from which to obtain the transaction context.
    /// </param>
    /// <returns>
    /// If there is a current transaction context, the return
    /// value is an Object that represents the context.
    /// If there is not a current transaction context, the
    /// value returned is DBNull.Value.
    /// </returns>

    public Object GetTransactionContext(SqlConnection sqlConnection)
    {
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "SELECT GET_FILESTREAM_TRANSACTION_CONTEXT();";
        cmd.CommandType = CommandType.Text;
        cmd.Connection = sqlConnection;

        return cmd.ExecuteScalar();
    }

    /// <summary>
    /// Begins the transaction.
    /// </summary>
    /// <param name="sqlConnection">
    /// SqlConnection object that represents the server
    /// on which to run the BEGIN TRANSACTION statement.
    /// </param>

    public void BeginTransaction(SqlConnection sqlConnection)
    {
        SqlCommand cmd = new SqlCommand();

        cmd.CommandText = "BEGIN TRANSACTION";
        cmd.CommandType = CommandType.Text;
        cmd.Connection = sqlConnection;

        cmd.ExecuteNonQuery();
    }

    /// <summary>
    /// Commits the transaction.
    /// </summary>
    /// <param name="sqlConnection">
    /// SqlConnection object that represents the instance of SQL Server
    /// on which to run the COMMIT statement.
    /// </param>

    public void CommitTransaction(SqlConnection sqlConnection)
    {
        SqlCommand cmd = new SqlCommand();

        cmd.CommandText = "COMMIT TRANSACTION";
        cmd.CommandType = CommandType.Text;
        cmd.Connection = sqlConnection;

        cmd.ExecuteNonQuery();
    }
}
```

```

class Program
{
    static void Main(string[] args)
    {
        //Open a connection to the local instance of SQL Server.

        SqlConnection sqlConnection = new SqlConnection("Integrated Security=true;server=(local)");
        sqlConnection.Open();

        SqlAccessWrapper sql = new SqlAccessWrapper();

        //Create a transaction so that sql.GetTransactionContext() will succeed.
        sql.BeginTransaction(sqlConnection);

        //The transaction context will be stored in this array.
        Byte[] transactionToken;

        Object txObj = sql.GetTransactionContext(sqlConnection);
        if (DBNull.Value != txObj)
        {
            transactionToken = (byte[])txObj;
            Console.WriteLine("Transaction context obtained.\n");
        }

        sql.CommitTransaction(sqlConnection);
    }
}

```

```

Imports System
Imports System.Data.SqlClient
Imports System.Data

Namespace ConsoleApplication
    ''' <summary>
    ''' This class is a wrapper that contains methods for:
    '''

    ''' GetTransactionContext() - Returns the current transaction context.
    ''' BeginTransaction() - Begins a transaction.
    ''' CommitTransaction() - Commits the current transaction.
    '''

    ''' </summary>

    Class SqlAccessWrapper
        ''' <summary>
        ''' Returns a byte array that contains the current transaction
        ''' context.
        ''' </summary>
        ''' <param name="sqlConnection">
        ''' SqlConnection object that represents the instance of SQL Server
        ''' from which to obtain the transaction context.
        ''' </param>
        ''' <returns>
        ''' If there is a current transaction context, the return
        ''' value is an Object that represents the context.
        ''' If there is not a current transaction context, the
        ''' value returned is DBNull.Value.
        ''' </returns>

        Public Function GetTransactionContext(ByVal sqlConnection As SqlConnection) As Object
            Dim cmd As New SqlCommand()
            cmd.CommandText = "SELECT GET_FILESTREAM_TRANSACTION_CONTEXT()"
            cmd.CommandType = CommandType.Text
            cmd.Connection = sqlConnection

            Return cmd.ExecuteScalar()
        End Function
    End Class
End Namespace

```

```

End Function

''' <summary>
''' Begins the transaction.
''' </summary>
''' <param name="sqlConnection">
''' SqlConnection object that represents the server
''' on which to run the BEGIN TRANSACTION statement.
''' </param>

Public Sub BeginTransaction(ByVal sqlConnection As SqlConnection)
    Dim cmd As New SqlCommand()

    cmd.CommandText = "BEGIN TRANSACTION"
    cmd.CommandType = CommandType.Text
    cmd.Connection = sqlConnection

    cmd.ExecuteNonQuery()
End Sub

''' <summary>
''' Commits the transaction.
''' </summary>
''' <param name="sqlConnection">
''' SqlConnection object that represents the instance of SQL Server
''' on which to run the COMMIT statement.
''' </param>

Public Sub CommitTransaction(ByVal sqlConnection As SqlConnection)
    Dim cmd As New SqlCommand()

    cmd.CommandText = "COMMIT TRANSACTION"
    cmd.CommandType = CommandType.Text
    cmd.Connection = sqlConnection

    cmd.ExecuteNonQuery()
End Sub
End Class

Class Program
    Shared Sub Main()
        '''Open a connection to the local instance of SQL Server.

        Dim sqlConnection As New SqlConnection("Integrated Security=true;server=(local)")
        sqlConnection.Open()

        Dim sql As New SqlAccessWrapper()

        '''Create a transaction so that sql.GetTransactionContext() will succeed.
        sql.BeginTransaction(sqlConnection)

        '''The transaction context will be stored in this array.
        Dim transactionToken As Byte()

        Dim txObj As Object = sql.GetTransactionContext(sqlConnection)

        '''If the returned object is not NULL, there is a valid transaction
        '''token, and it must be converted into a format that is usable within
        '''the application.

        If Not txObj.Equals(DBNull.Value) Then
            transactionToken = DirectCast(txObj, Byte())
            Console.WriteLine("Transaction context obtained." & Chr(10) & "")
        End If

        sql.CommitTransaction(sqlConnection)
    End Sub
End Class

```

```
End Namespace
```

## See Also

[PathName \(Transact-SQL\)](#)

[Binary Large Object \(Blob\) Data \(SQL Server\)](#)

# GETANSINULL (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the default nullability for the database for this session.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
GETANSINULL ( [ 'database' ] )
```

## Arguments

'*database*'

Is the name of the database for which to return nullability information. *database* is either **char** or **nchar**. If **char**, *database* is implicitly converted to **nchar**.

## Return Types

**int**

## Remarks

When the nullability of the specified database allows for null values and the column or data type nullability is not explicitly defined, GETANSINULL returns 1. This is the ANSI NULL default.

To enable the ANSI NULL default behavior, one of these conditions must be set:

- ALTER DATABASE *database\_name* SET ANSI\_NULL\_DEFAULT ON
- SET ANSI\_NULL\_DFLT\_ON ON
- SET ANSI\_NULL\_DFLT\_OFF OFF

## Examples

The following example returns the default nullability for the `AdventureWorks2012` database.

```
USE AdventureWorks2012;
GO
SELECT GETANSINULL('AdventureWorks2012')
GO
```

Here is the result set.

-----

1

(1 row(s) affected)

## See Also

[System Functions \(Transact-SQL\)](#)

# HOST\_ID (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the workstation identification number. The workstation identification number is the process ID (PID) of the application on the client computer that is connecting to SQL Server.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
HOST_ID ()
```

## Return Types

**char(10)**

## Remarks

When the parameter to a system function is optional, the current database, host computer, server user, or database user is assumed. Built-in functions must always be followed by parentheses.

System functions can be used in the select list, in the WHERE clause, and anywhere an expression is allowed.

## Examples

The following example creates a table that uses `HOST_ID()` in a `DEFAULT` definition to record the terminal ID of computers that insert rows into a table recording orders.

```
CREATE TABLE Orders
(OrderID      int      PRIMARY KEY,
 CustomerID  nchar(5) REFERENCES Customers(CustomerID),
 TerminalID   char(8)  NOT NULL DEFAULT HOST_ID(),
 OrderDate    datetime NOT NULL,
 ShipDate    datetime NULL,
 ShipperID    int      NULL REFERENCES Shippers(ShipperID));
GO
```

## See Also

[Expressions \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

# HOST\_NAME (Transact-SQL)

9/21/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2016) ✓ Azure SQL Database ✗ Azure SQL Data

Warehouse ✗ Parallel Data Warehouse

Returns the workstation name.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
HOST_NAME ()
```

## Return Types

**nvarchar(128)**

## Remarks

When the parameter to a system function is optional, the current database, host computer, server user, or database user is assumed. Built-in functions must always be followed by parentheses.

System functions can be used in the select list, in the WHERE clause, and anywhere an expression is allowed.

### IMPORTANT

The client application provides the workstation name and can provide inaccurate data. Do not rely upon HOST\_NAME as a security feature.

## Examples

The following example creates a table that uses `HOST_NAME()` in a `DEFAULT` definition to record the workstation name of computers that insert rows into a table recording orders.

```
CREATE TABLE Orders
(OrderID      int          PRIMARY KEY,
 CustomerID   nchar(5)    REFERENCES Customers(CustomerID),
 Workstation   nchar(30)   NOT NULL DEFAULT HOST_NAME(),
 OrderDate    datetime    NOT NULL,
 ShipDate     datetime    NULL,
 ShipperID    int         NULL REFERENCES Shippers(ShipperID));
GO
```

## See Also

[Expressions \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

# ISNULL (Transact-SQL)

9/27/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Replaces NULL with the specified replacement value.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
ISNULL ( check_expression , replacement_value )
```

## Arguments

*check\_expression*

Is the [expression](#) to be checked for NULL. *check\_expression* can be of any type.

*replacement\_value*

Is the expression to be returned if *check\_expression* is NULL. *replacement\_value* must be of a type that is implicitly convertible to the type of *check\_expression*.

## Return Types

Returns the same type as *check\_expression*. If a literal NULL is provided as *check\_expression*, returns the datatype of the *replacement\_value*. If a literal NULL is provided as *check\_expression* and no *replacement\_value* is provided, returns an [int](#).

## Remarks

The value of *check\_expression* is returned if it is not NULL; otherwise, *replacement\_value* is returned after it is implicitly converted to the type of *check\_expression*, if the types are different. *replacement\_value* can be truncated if *replacement\_value* is longer than *check\_expression*.

### NOTE

Use [COALESCE \(Transact-SQL\)](#) to return the first non-null value.

## Examples

### A. Using ISNULL with AVG

The following example finds the average of the weight of all products. It substitutes the value `50` for all NULL entries in the `Weight` column of the `Product` table.

```
USE AdventureWorks2012;
GO
SELECT AVG(ISNULL(Weight, 50))
FROM Production.Product;
GO
```

Here is the result set.

```
-----
```

```
59.79
```

```
(1 row(s) affected)
```

## B. Using ISNULL

The following example selects the description, discount percentage, minimum quantity, and maximum quantity for all special offers in `AdventureWorks2012`. If the maximum quantity for a particular special offer is `NULL`, the `MaxQty` shown in the result set is `0.00`.

```
USE AdventureWorks2012;
GO
SELECT Description, DiscountPct, MinQty, ISNULL(MaxQty, 0.00) AS 'Max Quantity'
FROM Sales.SpecialOffer;
GO
```

Here is the result set.

DESCRIPTION	DISCOUNTPCT	MINQTY	MAX QUANTITY
No Discount	0.00	0	0
Volume Discount	0.02	11	14
Volume Discount	0.05	15	4
Volume Discount	0.10	25	0
Volume Discount	0.15	41	0
Volume Discount	0.20	61	0
Mountain-100 Cl	0.35	0	0
Sport Helmet Di	0.10	0	0
Road-650 Overst	0.30	0	0
Mountain Tire S	0.50	0	0
Sport Helmet Di	0.15	0	0
LL Road Frame S	0.35	0	0
Touring-3000 Pr	0.15	0	0

DESCRIPTION	DISCOUNTPCT	MINQTY	MAX QUANTITY
Touring-1000 Pr	0.20	0	0
Half-Price Peda	0.50	0	0
Mountain-500 Si	0.40	0	0

(16 row(s) affected)

### C. Testing for NULL in a WHERE clause

Do not use ISNULL to find NULL values. Use IS NULL instead. The following example finds all products that have `NULL` in the weight column. Note the space between `IS` and `NULL`.

```
USE AdventureWorks2012;
GO
SELECT Name, Weight
FROM Production.Product
WHERE Weight IS NULL;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

### D. Using ISNULL with AVG

The following example finds the average of the weight of all products in a sample table. It substitutes the value `50` for all NULL entries in the `Weight` column of the `Product` table.

```
-- Uses AdventureWorks

SELECT AVG(ISNULL(Weight, 50))
FROM dbo.DimProduct;
```

Here is the result set.

```
-----
52.88
```

### E. Using ISNULL

The following example uses ISNULL to test for NULL values in the column `MinPaymentAmount` and display the value `0.00` for those rows.

```
-- Uses AdventureWorks

SELECT ResellerName,
       ISNULL(MinPaymentAmount, 0) AS MinimumPayment
  FROM dbo.DimReseller
 ORDER BY ResellerName;
```

Here is a partial result set.

RESELLERNAME	MINIMUMPAYMENT
A Bicycle Association	0.0000
A Bike Store	0.0000
A Cycle Shop	0.0000
A Great Bicycle Company	0.0000
A Typical Bike Shop	200.0000
Acceptable Sales & Service	0.0000

## F. Using IS NULL to test for NULL in a WHERE clause

The following example finds all products that have `NULL` in the `Weight` column. Note the space between `IS` and `NULL`.

```
-- Uses AdventureWorks

SELECT EnglishProductName, Weight
FROM dbo.DimProduct
WHERE Weight IS NULL;
```

## See Also

[Expressions \(Transact-SQL\)](#)

[IS NULL \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

[WHERE \(Transact-SQL\)](#)

[COALESCE \(Transact-SQL\)](#)

# ISNUMERIC (Transact-SQL)

9/27/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✓ Parallel Data Warehouse

Determines whether an expression is a valid numeric type.

 [Transact-SQL Syntax Conventions](#)

## Syntax

```
ISNUMERIC ( expression )
```

## Arguments

*expression*

Is the [expression](#) to be evaluated.

## Return Types

**int**

## Remarks

ISNUMERIC returns 1 when the input expression evaluates to a valid numeric data type; otherwise it returns 0. Valid numeric data types include the following:

<b>int</b>	<b>numeric</b>
<b>bigint</b>	<b>money</b>
<b>smallint</b>	<b>smallmoney</b>
<b>tinyint</b>	<b>float</b>
<b>decimal</b>	<b>real</b>

### NOTE

ISNUMERIC returns 1 for some characters that are not numbers, such as plus (+), minus (-), and valid currency symbols such as the dollar sign (\$). For a complete list of currency symbols, see [money and smallmoney \(Transact-SQL\)](#).

## Examples

The following example uses `ISNUMERIC` to return all the postal codes that are not numeric values.

```
USE AdventureWorks2012;
GO
SELECT City, PostalCode
FROM Person.Address
WHERE ISNUMERIC(PostalCode)<> 1;
GO
```

## Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

The following example uses `ISNUMERIC` to return all the postal codes that are not numeric values.

```
USE master;
GO
SELECT name, isnumeric(name) AS IsNameANumber, database_id, isnumeric(database_id) AS IsIdANumber
FROM sys.databases;
GO
```

## See Also

[Expressions \(Transact-SQL\)](#)

[System Functions \(Transact-SQL\)](#)

[Data Types \(Transact-SQL\)](#)

# MIN\_ACTIVE\_ROWVERSION (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the lowest active **rowversion** value in the current database. A **rowversion** value is active if it is used in a transaction that has not yet been committed. For more information, see [rowversion \(Transact-SQL\)](#).

## NOTE

The **rowversion** data type is also known as **timestamp**.



## Syntax

```
MIN_ACTIVE_ROWVERSION
```

## Return Types

Returns a **binary(8)** value.

## Remarks

MIN\_ACTIVE\_ROWVERSION is a non-deterministic function that returns the lowest active **rowversion** value in the current database. A new **rowversion** value is typically generated when an insert or update is performed on a table that contains a column of type **rowversion**. If there are no active values in the database, MIN\_ACTIVE\_ROWVERSION returns the same value as @@DBTS + 1.

MIN\_ACTIVE\_ROWVERSION is useful for scenarios such as data synchronization that use **rowversion** values to group sets of changes together. If an application uses @@DBTS rather than MIN\_ACTIVE\_ROWVERSION, it is possible to miss changes that are active when synchronization occurs.

The MIN\_ACTIVE\_ROWVERSION function is not affected by changes in the transaction isolation levels.

## Examples

The following example returns **rowversion** values by using `MIN_ACTIVE_ROWVERSION` and `@@DBTS`. Notice that the values differ when there are no active transactions in the database.

```
-- Create a table that has a ROWVERSION column in it.  
CREATE TABLE RowVersionTestTable (rv ROWVERSION)  
GO  
  
-- Print the current values for the database.  
PRINT ''  
PRINT 'DBTS'  
PRINT @@DBTS  
PRINT 'MIN_ACTIVE_ROWVERSION'  
PRINT MIN_ACTIVE_ROWVERSION()
```

```

GO
----- Results -----
--DBTS
--0x000000000000007E2
--MIN_ACTIVE_ROWVERSION
--0x000000000000007E3

-- Insert a row.
INSERT INTO RowVersionTestTable VALUES (DEFAULT)
SELECT * FROM RowVersionTestTable
GO
----- Results -----
--rv
--0x000000000000007E3

-- Print the current values for the database.
PRINT ''
PRINT 'DBTS'
PRINT @@DBTS
PRINT 'MIN_ACTIVE_ROWVERSION'
PRINT MIN_ACTIVE_ROWVERSION()
GO
----- Results -----
--DBTS
--0x000000000000007E3
--MIN_ACTIVE_ROWVERSION
--0x000000000000007E4

-- Insert a new row inside a transaction but do not commit.
BEGIN TRAN
INSERT INTO RowVersionTestTable VALUES (DEFAULT)
SELECT * FROM RowVersionTestTable
GO
----- Results -----
--rv
--0x000000000000007E3
--0x000000000000007E4

-- Print the current values for the database.
PRINT ''
PRINT 'DBTS'
PRINT @@DBTS
PRINT 'MIN_ACTIVE_ROWVERSION'
PRINT MIN_ACTIVE_ROWVERSION()
GO
----- Results -----
--DBTS
--0x000000000000007E4
--MIN_ACTIVE_ROWVERSION
--0x000000000000007E4

-- Commit the transaction.
COMMIT
GO

-- Print the current values for the database.
PRINT ''
PRINT 'DBTS'
PRINT @@DBTS
PRINT 'MIN_ACTIVE_ROWVERSION'
PRINT MIN_ACTIVE_ROWVERSION()
GO
----- Results -----
--DBTS
--0x000000000000007E4
--MIN_ACTIVE_ROWVERSION
--0x000000000000007E5

```

## See Also

[@@DBTS \(Transact-SQL\)](#)

[rowversion \(Transact-SQL\)](#)

# NEWID (Transact-SQL)

7/31/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Creates a unique value of type **uniqueidentifier**.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
NEWID ( )
```

## Return Types

**uniqueidentifier**

## Remarks

`NEWID()` is compliant with RFC4122.

## Examples

### A. Using the NEWID function with a variable

The following example uses `NEWID()` to assign a value to a variable declared as the **uniqueidentifier** data type. The value of the **uniqueidentifier** data type variable is printed before the value is tested.

```
-- Creating a local variable with DECLARE/SET syntax.  
DECLARE @myid uniqueidentifier  
SET @myid = NEWID()  
PRINT 'Value of @myid is: '+ CONVERT(varchar(255), @myid)
```

Here is the result set.

```
Value of @myid is: 6F9619FF-8B86-D011-B42D-00C04FC964FF
```

### NOTE

The value returned by NEWID is different for each computer. This number is shown only for illustration.

### B. Using NEWID in a CREATE TABLE statement

**Applies to:** SQL Server

The following example creates the `cust` table with a **uniqueidentifier** data type, and uses NEWID to fill the table with a default value. In assigning the default value of `NEWID()`, each new and existing row has a unique value for the `CustomerID` column.

```

-- Creating a table using NEWID for uniqueidentifier data type.
CREATE TABLE cust
(
    CustomerID uniqueidentifier NOT NULL
        DEFAULT newid(),
    Company varchar(30) NOT NULL,
    ContactName varchar(60) NOT NULL,
    Address varchar(30) NOT NULL,
    City varchar(30) NOT NULL,
    StateProvince varchar(10) NULL,
    PostalCode varchar(10) NOT NULL,
    CountryRegion varchar(20) NOT NULL,
    Telephone varchar(15) NOT NULL,
    Fax varchar(15) NULL
);
GO
-- Inserting 5 rows into cust table.
INSERT cust
(CustomerID, Company, ContactName, Address, City, StateProvince,
PostalCode, CountryRegion, Telephone, Fax)
VALUES
(NEWID(), 'Wartian Herkku', 'Pirkko Koskitalo', 'Torikatu 38', 'Oulu', NULL,
'90110', 'Finland', '981-443655', '981-443655')
,(NEWID(), 'Wellington Importadora', 'Paula Parente', 'Rua do Mercado, 12', 'Resende', 'SP',
'08737-363', 'Brasil', '(14) 555-8122', '')
,(NEWID(), 'Cactus Comidas para Ilevar', 'Patricia Simpson', 'Cerrito 333', 'Buenos Aires', NULL,
'1010', 'Argentina', '(1) 135-5555', '(1) 135-4892')
,(NEWID(), 'Ernst Handel', 'Roland Mendel', 'Kirchgasse 6', 'Graz', NULL,
'8010', 'Austria', '7675-3425', '7675-3426')
,(NEWID(), 'Maison Dewey', 'Catherine Dewey', 'Rue Joseph-Bens 532', 'Bruxelles', NULL,
'B-1180', 'Belgium', '(02) 201 24 67', '(02) 201 24 68');
GO

```

## C. Using uniqueidentifier and variable assignment

The following example declares a local variable called `@myid` as a variable of **uniqueidentifier** data type. Then, the variable is assigned a value by using the `SET` statement.

```

DECLARE @myid uniqueidentifier ;
SET @myid = 'A972C577-DFB0-064E-1189-0154C99310DAAC12';
SELECT @myid;
GO

```

## See Also

[NEWSEQUENTIALID \(Transact-SQL\)](#)  
[ALTER TABLE \(Transact-SQL\)](#)  
[CAST and CONVERT \(Transact-SQL\)](#)  
[CREATE TABLE \(Transact-SQL\)](#)  
[Data Types \(Transact-SQL\)](#)  
[System Functions \(Transact-SQL\)](#)  
[uniqueidentifier \(Transact-SQL\)](#)  
[Sequence Numbers](#)

# NEWSEQUENTIALID (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Creates a GUID that is greater than any GUID previously generated by this function on a specified computer since Windows was started. After restarting Windows, the GUID can start again from a lower range, but is still globally unique. When a GUID column is used as a row identifier, using NEWSEQUENTIALID can be faster than using the NEWID function. This is because the NEWID function causes random activity and uses fewer cached data pages. Using NEWSEQUENTIALID also helps to completely fill the data and index pages.

## IMPORTANT

If privacy is a concern, do not use this function. It is possible to guess the value of the next generated GUID and, therefore, access data associated with that GUID.

NEWSEQUENTIALID is a wrapper over the Windows [UuidCreateSequential](#) function.

## WARNING

The UuidCreateSequential function has hardware dependencies. On SQL Server, clusters of sequential values can develop when databases (such as contained databases) are moved to other computers. When using Always On and on SQL Database, clusters of sequential values can develop if the database fails over to a different computer.



## Syntax

```
NEWSEQUENTIALID ( )
```

## Return Type

**uniqueidentifier**

## Remarks

NEWSEQUENTIALID() can only be used with DEFAULT constraints on table columns of type **uniqueidentifier**. For example:

```
CREATE TABLE myTable (ColumnA uniqueidentifier DEFAULT NEWSEQUENTIALID());
```

When NEWSEQUENTIALID() is used in DEFAULT expressions, it cannot be combined with other scalar operators. For example, you cannot execute the following:

```
CREATE TABLE myTable (ColumnA uniqueidentifier DEFAULT dbo.myfunction(NEWSEQUENTIALID()));
```

In the previous example, `myfunction()` is a scalar user-defined scalar function that accepts and returns a `uniqueidentifier` value.

`NEWSEQUENTIALID` cannot be referenced in queries.

You can use `NEWSEQUENTIALID` to generate GUIDs to reduce page splits and random IO at the leaf level of indexes.

Each GUID generated by using `NEWSEQUENTIALID` is unique on that computer. GUIDs generated by using `NEWSEQUENTIALID` are unique across multiple computers only if the source computer has a network card.

## See Also

[NEWID \(Transact-SQL\)](#)

[Comparison Operators \(Transact-SQL\)](#)

# ROWCOUNT\_BIG (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the number of rows affected by the last statement executed. This function operates like `@@ROWCOUNT`, except the return type of `ROWCOUNT_BIG` is **bigint**.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
ROWCOUNT_BIG ( )
```

## Return Types

**bigint**

## Remarks

Following a `SELECT` statement, this function returns the number of rows returned by the `SELECT` statement.

Following an `INSERT`, `UPDATE`, or `DELETE` statement, this function returns the number of rows affected by the data modification statement.

Following statements that do not return rows, such as an `IF` statement, this function returns 0.

## See Also

[COUNT\\_BIG \(Transact-SQL\)](#)

[Data Types \(Transact-SQL\)](#)

# SESSION\_CONTEXT (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2016) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the value of the specified key in the current session context. The value is set by using the [sp\\_set\\_session\\_context \(Transact-SQL\)](#) procedure.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
SESSION_CONTEXT(N'key')
```

## Arguments

'key'

The key (type sysname) of the value being retrieved.

## Return Type

**sql\_variant**

## Return Value

The value associated with the specified key in the session context, or NULL if no value has been set for that key.

## Permissions

Any user can read the session context for their session.

## Remarks

SESSION\_CONTEXT's MARS behavior is similar to that of CONTEXT\_INFO. If a MARS batch sets a key-value pair, the new value will not be returned in other MARS batches on the same connection unless they started after the batch that set the new value completed. If multiple MARS batches are active on a connection, values cannot be set as "read\_only." This prevents race conditions and non-determinism about which value "wins."

## Examples

The following simple example sets the session context value for key `user_id` to 4, and then uses the **SESSION\_CONTEXT** function to retrieve the value.

```
EXEC sp_set_session_context 'user_id', 4;
SELECT SESSION_CONTEXT(N'user_id');
```

## See Also

[sp\\_set\\_session\\_context \(Transact-SQL\)](#)

[CURRENT\\_TRANSACTION\\_ID \(Transact-SQL\)](#)

Row-Level Security

[CONTEXT\\_INFO \(Transact-SQL\)](#)

[SET CONTEXT\\_INFO \(Transact-SQL\)](#)

# SESSION\_ID (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the ID of the current SQL Data Warehouse or Parallel Data Warehouse session.

[Transact-SQL Syntax Conventions \(Transact-SQL\)](#)

## Syntax

```
-- Azure SQL Data Warehouse and Parallel Data Warehouse  
SESSION_ID()
```

## Return Value

Returns an **nvarchar(32)** value.

## General Remarks

The session ID is assigned to each user connection when the connection is made. It persists for the duration of the connection. When the connection ends, the session ID is released.

The session ID begins with the alphabetical characters 'SID'. These are case-sensitive and must be capitalized when session ID is used in SQL commands.

You can query the view [sys.dm\\_pdw\\_exec\\_sessions](#) to retrieve the same information as this function.

## Examples

The following example returns the current session ID.

```
SELECT SESSION_ID();
```

## See Also

[DB\\_NAME \(Transact-SQL\)](#)

[VERSION \(SQL Data Warehouse\)](#)

# XACT\_STATE (Transact-SQL)

9/27/2017 • 2 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Is a scalar function that reports the user transaction state of a current running request. XACT\_STATE indicates whether the request has an active user transaction, and whether the transaction is capable of being committed.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
XACT_STATE()
```

## Return Type

**smallint**

## Remarks

XACT\_STATE returns the following values.

RETURN VALUE	MEANING
1	The current request has an active user transaction. The request can perform any actions, including writing data and committing the transaction.
0	There is no active user transaction for the current request.
-1	<p>The current request has an active user transaction, but an error has occurred that has caused the transaction to be classified as an uncommittable transaction. The request cannot commit the transaction or roll back to a savepoint; it can only request a full rollback of the transaction. The request cannot perform any write operations until it rolls back the transaction. The request can only perform read operations until it rolls back the transaction. After the transaction has been rolled back, the request can perform both read and write operations and can begin a new transaction.</p> <p>When a batch finishes running, the Database Engine will automatically roll back any active uncommittable transactions. If no error message was sent when the transaction entered an uncommittable state, when the batch finishes, an error message will be sent to the client application. This message indicates that an uncommittable transaction was detected and rolled back.</p>

Both the XACT\_STATE and @@TRANCOUNT functions can be used to detect whether the current request has an active user transaction. @@TRANCOUNT cannot be used to determine whether that transaction has been classified as an uncommittable transaction. XACT\_STATE cannot be used to determine whether there are nested transactions.

## Examples

The following example uses `XACT_STATE` in the `CATCH` block of a `TRY...CATCH` construct to determine whether to commit or roll back a transaction. Because `SET XACT_ABORT` is `ON`, the constraint violation error causes the transaction to enter an uncommittable state.

```
USE AdventureWorks2012;
GO

-- SET XACT_ABORT ON will render the transaction uncommittable
-- when the constraint violation occurs.
SET XACT_ABORT ON;

BEGIN TRY
    BEGIN TRANSACTION;
        -- A FOREIGN KEY constraint exists on this table. This
        -- statement will generate a constraint violation error.
        DELETE FROM Production.Product
            WHERE ProductID = 980;

        -- If the delete operation succeeds, commit the transaction. The CATCH
        -- block will not execute.
        COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    -- Test XACT_STATE for 0, 1, or -1.
    -- If 1, the transaction is committable.
    -- If -1, the transaction is uncommittable and should
    --     be rolled back.
    -- XACT_STATE = 0 means there is no transaction and
    --     a commit or rollback operation would generate an error.

    -- Test whether the transaction is uncommittable.
    IF (XACT_STATE()) = -1
    BEGIN
        PRINT 'The transaction is in an uncommittable state.' +
            ' Rolling back transaction.'
        ROLLBACK TRANSACTION;
    END;

    -- Test whether the transaction is active and valid.
    IF (XACT_STATE()) = 1
    BEGIN
        PRINT 'The transaction is committable.' +
            ' Committing transaction.'
        COMMIT TRANSACTION;
    END;
END CATCH;
GO
```

## See Also

[@@TRANCOUNT \(Transact-SQL\)](#)  
[BEGIN TRANSACTION \(Transact-SQL\)](#)  
[COMMIT TRANSACTION \(Transact-SQL\)](#)  
[ROLLBACK TRANSACTION \(Transact-SQL\)](#)  
[SAVE TRANSACTION \(Transact-SQL\)](#)  
[TRY...CATCH \(Transact-SQL\)](#)

# System Statistical Functions (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following scalar functions return statistical information about the system:

<a href="#">@@CONNECTIONS</a>	<a href="#">@@PACK_RECEIVED</a>
<a href="#">@@CPU_BUSY</a>	<a href="#">@@PACK_SENT</a>
<a href="#">fn_virtualfilestats</a>	<a href="#">@@TIMETICKS</a>
<a href="#">@@IDLE</a>	<a href="#">@@TOTAL_ERRORS</a>
<a href="#">@@IO_BUSY</a>	<a href="#">@@TOTAL_READ</a>
<a href="#">@@PACKET_ERRORS</a>	<a href="#">@@TOTAL_WRITE</a>

All system statistical functions are nondeterministic. This means these functions do not always return the same results every time they are called, even with the same set of input values. For more information about function determinism, see [Deterministic and Nondeterministic Functions](#).

## See Also

[Built-in Functions \(Transact-SQL\)](#)

# @@CONNECTIONS (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the number of attempted connections, either successful or unsuccessful since SQL Server was last started.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@CONNECTIONS
```

## Return types

**integer**

## Remarks

Connections are different from users. Applications, for example, can open multiple connections to SQL Server without the user observing the connections.

To display a report containing several SQL Server statistics, including connection attempts, run **sp\_monitor**.

`@@MAX_CONNECTIONS` is the maximum number of connections allowed simultaneously to the server.

`@@CONNECTIONS` is incremented with each login attempt, therefore `@@CONNECTIONS` can be greater than `@@MAX_CONNECTIONS`.

## Examples

The following example shows returning the number of login attempts as of the current date and time.

```
SELECT GETDATE() AS 'Today''s Date and Time',  
@@CONNECTIONS AS 'Login Attempts';
```

Here is the result set.

Today's Date and Time	Login Attempts
-----	-----
12/5/2006 10:32:45 AM	211023

## See also

[System Statistical Functions \(Transact-SQL\)](#)

[sp\\_monitor \(Transact-SQL\)](#)

# @@CPU\_BUSY (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the time that SQL Server has spent working since it was last started. Result is in CPU time increments, or "ticks," and is cumulative for all CPUs, so it may exceed the actual elapsed time. Multiply by @@TIMETICKS to convert to microseconds.

## NOTE

If the time returned in @@CPU\_BUSY or @@IO\_BUSY exceeds approximately 49 days of cumulative CPU time, you receive an arithmetic overflow warning. In that case, the value of @@CPU\_BUSY, @@IO\_BUSY and @@IDLE variables are not accurate.



## Syntax

```
@@CPU_BUSY
```

## Return types

**integer**

## Remarks

To display a report containing several SQL Server statistics, including CPU activity, run [sp\\_monitor](#).

## Examples

The following example shows returning SQL Server CPU activity as of the current date and time. To avoid arithmetic overflow when converting the value to microseconds, the example converts one of the values to the `float` data type.

```
SELECT @@CPU_BUSY * CAST(@@TIMETICKS AS float) AS 'CPU microseconds',
       GETDATE() AS 'As of' ;
```

Here is the result set.

CPU microseconds	As of
18406250	2006-12-05 17:00:50.600

## See also

[sys.dm\\_os\\_sys\\_info \(Transact-SQL\)](#)

[@@IDLE \(Transact-SQL\)](#)

[@@IO\\_BUSY \(Transact-SQL\)](#)

[sp\\_monitor \(Transact-SQL\)](#)

[System Statistical Functions \(Transact-SQL\)](#)

# @@IDLE (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** ✓ SQL Server (starting with 2008) ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Returns the time that SQL Server has been idle since it was last started. The result is in CPU time increments, or "ticks," and is cumulative for all CPUs, so it may exceed the actual elapsed time. Multiply by @@TIMETICKS to convert to microseconds.

## NOTE

If the time returned in @@CPU\_BUSY, or @@IO\_BUSY exceeds approximately 49 days of cumulative CPU time, you receive an arithmetic overflow warning. In that case, the value of @@CPU\_BUSY, @@IO\_BUSY and @@IDLE variables are not accurate.



## Syntax

```
@@IDLE
```

## Return Types

**integer**

## Remarks

To display a report containing several SQL Server statistics, run **sp\_monitor**.

## Examples

The following example shows returning the number of milliseconds SQL Server was idle between the start time and the current time. To avoid arithmetic overflow when converting the value to microseconds, the example converts one of the values to the **float** data type.

```
SELECT @@IDLE * CAST(@@TIMETICKS AS float) AS 'Idle microseconds',
       GETDATE() AS 'as of';
```

Here is the result set.

I	
Idle microseconds	as of
-----	
8199934	12/5/2006 10:23:00 AM

## See Also

[@@CPU\\_BUSY \(Transact-SQL\)](#)  
[sp\\_monitor \(Transact-SQL\)](#)  
[@@IO\\_BUSY \(Transact-SQL\)](#)  
[System Statistical Functions \(Transact-SQL\)](#)

# @@IO\_BUSY (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the time that SQL Server has spent performing input and output operations since SQL Server was last started. The result is in CPU time increments ("ticks"), and is cumulative for all CPUs, so it may exceed the actual elapsed time. Multiply by @@TIMETICKS to convert to microseconds.

## NOTE

If the time returned in @@CPU\_BUSY, or @@IO\_BUSY exceeds approximately 49 days of cumulative CPU time, you receive an arithmetic overflow warning. In that case, the value of @@CPU\_BUSY, @@IO\_BUSY and @@IDLE variables are not accurate.



## Syntax

```
@@IO_BUSY
```

## Return Types

**integer**

## Remarks

To display a report containing several SQL Server statistics, run `sp_monitor`.

## Examples

The following example shows returning the number of milliseconds SQL Server has spent performing input/output operations between the start time and the current time. To avoid arithmetic overflow when converting the value to microseconds, the example converts one of the values to the **float** data type.

```
SELECT @@IO_BUSY*@@TIMETICKS AS 'IO microseconds',
      GETDATE() AS 'as of';
```

Here is a typical result set:

IO microseconds as of	-----
4552312500	12/5/2006 10:23:00 AM

## See Also

[sys.dm\\_os\\_sys\\_info \(Transact-SQL\)](#)  
[@@CPU\\_BUSY \(Transact-SQL\)](#)  
[sp\\_monitor \(Transact-SQL\)](#)  
[System Statistical Functions \(Transact-SQL\)](#)

# @@PACK\_SENT (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the number of output packets written to the network by SQL Server since it was last started.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@PACK_SENT
```

## Return Types

**integer**

## Remarks

To display a report containing several SQL Server statistics, including packets sent and received, run **sp\_monitor**.

## Examples

The following example shows the usage of `@@PACK_SENT`.

```
SELECT @@PACK_SENT AS 'Pack Sent';
```

Here is a sample result set.

```
Pack Sent  
-----  
291
```

## See Also

[@@PACK\\_RECEIVED \(Transact-SQL\)](#)  
[sp\\_monitor \(Transact-SQL\)](#)  
[System Statistical Functions \(Transact-SQL\)](#)

# @@PACKET\_ERRORS (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the number of network packet errors that have occurred on SQL Server connections since SQL Server was last started.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@PACKET_ERRORS
```

## Return Types

**integer**

## Remarks

To display a report containing several SQL Server statistics, including packet errors, run **sp\_monitor**.

## Examples

The following example shows using `@@PACKET_ERRORS`.

```
SELECT @@PACKET_ERRORS AS 'Packet Errors';
```

Here is a sample result set.

```
Packet Errors
-----
0
```

## See Also

[@@PACK\\_RECEIVED \(Transact-SQL\)](#)  
[@@PACK\\_SENT \(Transact-SQL\)](#)  
[sp\\_monitor \(Transact-SQL\)](#)  
[System Statistical Functions \(Transact-SQL\)](#)

# @@TIMETICKS (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the number of microseconds per tick.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@TIMETICKS
```

## Return Types

**integer**

## Remarks

The amount of time per tick is computer-dependent. Each tick on the operating system is 31.25 milliseconds, or one thirty-second of a second.

## Examples

```
SELECT @@TIMETICKS AS 'Time Ticks';
```

## See Also

[System Statistical Functions \(Transact-SQL\)](#)

# @@TOTAL\_ERRORS (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the number of disk write errors encountered by SQL Server since SQL Server last started.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@TOTAL_ERRORS
```

## Return Types

**integer**

## Remarks

Not all write errors encountered by SQL Server are accounted for by this function. Occasional non-fatal write errors are handled by the server itself and are not considered errors. To display a report containing several SQL Server statistics, including total number of errors, run **sp\_monitor**.

## Examples

This example shows the number of errors encountered by SQL Server as of the current date and time.

```
SELECT @@TOTAL_ERRORS AS 'Errors', GETDATE() AS 'As of';
```

Here is the result set.

Errors	As of
0	3/28/2003 12:32:11 PM

## See Also

[sp\\_monitor \(Transact-SQL\)](#)

[System Statistical Functions \(Transact-SQL\)](#)

# @@TOTAL\_READ (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the number of disk reads, not cache reads, by SQL Server since SQL Server was last started.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@TOTAL_READ
```

## Return Types

**integer**

## Remarks

To display a report containing several SQL Server statistics, including read and write activity, run **sp\_monitor**.

## Examples

The following example shows returning the total number of disk read and writes as of the current date and time.

```
SELECT @@TOTAL_READ AS 'Reads', @@TOTAL_WRITE AS 'Writes', GETDATE() AS 'As of';
```

Here is the result set.

Reads	Writes	As of
7760	97263	12/5/2006 10:23:00 PM

## See Also

[sp\\_monitor \(Transact-SQL\)](#)

[System Statistical Functions \(Transact-SQL\)](#)

[@@TOTAL\\_WRITE \(Transact-SQL\)](#)

# @@TOTAL\_WRITE (Transact-SQL)

9/18/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns the number of disk writes by SQL Server since SQL Server was last started.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
@@TOTAL_WRITE
```

## Return Types

**integer**

## Remarks

To display a report containing several SQL Server statistics, including read and write activity, run **sp\_monitor**.

## Examples

The following example shows returning the total number of disk reads and writes as of the current date and time.

```
SELECT @@TOTAL_READ AS 'Reads', @@TOTAL_WRITE AS 'Writes', GETDATE() AS 'As of'
```

Here is the result set.

Reads	Writes	As of
7760	97263	12/5/2006 10:23:00 PM

## See Also

[sp\\_monitor \(Transact-SQL\)](#)

[System Statistical Functions \(Transact-SQL\)](#)

[@@TOTAL\\_READ \(Transact-SQL\)](#)

# Text and Image Functions - TEXTPTR (Transact-SQL)

3/24/2017 • 4 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the text-pointer value that corresponds to a **text**, **ntext**, or **image** column in **varbinary** format. The retrieved text pointer value can be used in **READTEXT**, **WRITETEXT**, and **UPDATETEXT** statements.

## IMPORTANT

This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Alternative functionality is not available.



## Syntax

```
TEXTPTR ( column )
```

## Arguments

*column*

Is the **text**, **ntext**, or **image** column that will be used.

## Return Types

**varbinary**

## Remarks

For tables with in-row text, TEXTPTR returns a handle for the text to be processed. You can obtain a valid text pointer even if the text value is null.

You cannot use the TEXTPTR function on columns of views. You can only use it on columns of tables. To use the TEXTPTR function on a column of a view, you must set the compatibility level to 80 by using [ALTER DATABASE Compatibility Level](#). If the table does not have in-row text, and if a **text**, **ntext**, or **image** column has not been initialized by an UPDATETEXT statement, TEXTPTR returns a null pointer.

Use TEXTVALID to test whether a text pointer exists. You cannot use UPDATETEXT, WRITETEXT, or READTEXT without a valid text pointer.

These functions and statements are also useful when you work with **text**, **ntext**, and **image** data.

FUNCTION OR STATEMENT	DESCRIPTION
<code>PATINDEX('%pattern%', expression)</code>	Returns the character position of a specified character string in <b>text</b> or <b>ntext</b> columns.

FUNCTION OR STATEMENT	DESCRIPTION
DATALENGTH( <i>expression</i> )	Returns the length of data in <b>text</b> , <b>ntext</b> , and <b>image</b> columns.
SET TEXTSIZE	Returns the limit, in bytes, of the <b>text</b> , <b>ntext</b> , or <b>image</b> data to be returned with a SELECT statement.
SUBSTRING( <i>text_column</i> , <i>start</i> , <i>length</i> )	Returns a <b>varchar</b> string specified by the specified <i>start</i> offset and <i>length</i> . The length should be less than 8 KB.

## Examples

### NOTE

To run the following examples, you must install the **pubs** database.

### A. Using TEXTPTR

The following example uses the `TEXTPTR` function to locate the **image** column `logo` associated with `New Moon Books` in the `pub_info` table of the `pubs` database. The text pointer is put into a local variable `@ptrval`.

```
USE pubs;
GO
DECLARE @ptrval varbinary(16);
SELECT @ptrval = TEXTPTR(logo)
FROM pub_info pr, publishers p
WHERE p.pub_id = pr.pub_id
    AND p.pub_name = 'New Moon Books';
GO
```

### B. Using TEXTPTR with in-row text

In SQL Server, the in-row text pointer must be used inside a transaction, as shown in the following example.

```
CREATE TABLE t1 (c1 int, c2 text);
EXEC sp_tableoption 't1', 'text in row', 'on';
INSERT t1 VALUES ('1', 'This is text.');
GO
BEGIN TRAN;
DECLARE @ptrval VARBINARY(16);
SELECT @ptrval = TEXTPTR(c2)
FROM t1
WHERE c1 = 1;
READTEXT t1.c2 @ptrval 0 1;
COMMIT;
```

### C. Returning text data

The following example selects the `pub_id` column and the 16-byte text pointer of the `pr_info` column from the `pub_info` table.

```
USE pubs;
GO
SELECT pub_id, TEXTPTR(pr_info)
FROM pub_info
ORDER BY pub_id;
GO
```

Here is the result set.

```
pub_id
-----
0736  0x6c0000000000feffb80100001000100
0877  0xd00000000000feffb80100001000300
1389  0xe00000000000feffb80100001000500
1622  0x700000000000feffb80100001000900
1756  0x710000000000feffb80100001000b00
9901  0x720000000000feffb80100001000d00
9952  0x6f0000000000feffb80100001000700
9999  0x730000000000feffb80100001000f00

(8 row(s) affected)
```

The following example shows how to return the first `8000` bytes of text without using TEXTPTR.

```
USE pubs;
GO
SET TEXTSIZE 8000;
SELECT pub_id, pr_info
FROM pub_info
ORDER BY pub_id;
GO
```

Here is the result set.

```
pub_id pr_info
-----
0736  New Moon Books (NMB) has just released another top ten publication. With the latest publication this
makes NMB the hottest new publisher of the year!
0877  This is sample text data for Binnet & Hardley, publisher 0877 in the pubs database. Binnet & Hardley is
located in Washington, D.C.

This is sample text data for Binnet & Hardley, publisher 0877 in the pubs database. Binnet & Hardley is located
in Washi
1389  This is sample text data for Algodata Infosystems, publisher 1389 in the pubs database. Algodata
Infosystems is located in Berkeley, California.

9999  This is sample text data for Lucerne Publishing, publisher 9999 in the pubs database. Lucerne publishing
is located in Paris, France.

This is sample text data for Lucerne Publishing, publisher 9999 in the pubs database. Lucerne publishing is
located in

(8 row(s) affected)
```

## D. Returning specific text data

The following example locates the `text` column (`pr_info`) associated with `pub_id`0736` in the `pub_info` table of the `pubs` database. It first declares the local variable `@val`. The text pointer (a long binary string) is then put into `@val` and supplied as a parameter to the `READTEXT` statement. This returns 10 bytes starting at the fifth byte (offset of 4).

```
USE pubs;
GO
DECLARE @val varbinary(16);
SELECT @val = TEXTPTR(pr_info)
FROM pub_info
WHERE pub_id = '0736';
READTEXT pub_info.pr_info @val 4 10;
GO
```

Here is the result set.

```
pr_info
-----
is sample
(1 row(s) affected)
```

## See Also

[DATALENGTH \(Transact-SQL\)](#)  
[PATINDEX \(Transact-SQL\)](#)  
[READTEXT \(Transact-SQL\)](#)  
[SET TEXTSIZE \(Transact-SQL\)](#)  
[Text and Image Functions \(Transact-SQL\)](#)  
[UPDATETEXT \(Transact-SQL\)](#)  
[WRITETEXT \(Transact-SQL\)](#)

# Text and Image Functions - TEXTVALID (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

A **text**, **ntext**, or **image** function that checks whether a specific text pointer is valid.

## IMPORTANT

This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Alternative functionality is not available.



## Syntax

```
TEXTVALID ( 'table.column' ,text_ ptr )
```

## Arguments

*table*

Is the name of the table that will be used.

*column*

Is the name of the column that will be used.

*text\_ptr*

Is the text pointer to be checked.

## Return Types

**int**

## Remarks

Returns 1 if the pointer is valid and 0 if the pointer is not valid. Note that the identifier for the **text** column must include the table name. You cannot use UPDATETEXT, WRITETEXT, or READTEXT without a valid text pointer.

The following functions and statements are also useful when you work with **text**, **ntext**, and **image** data.

FUNCTION OR STATEMENT	DESCRIPTION
<code>PATINDEX('%pattern%', expression)</code>	Returns the character position of a specified character string in <b>text</b> and <b>ntext</b> columns.
<code>DATALENGTH(expression)</code>	Returns the length of data in <b>text</b> , <b>ntext</b> , and <b>image</b> columns.

FUNCTION OR STATEMENT	DESCRIPTION
SET TEXTSIZE	Returns the limit, in bytes, of the <b>text</b> , <b>ntext</b> , or <b>image</b> data to be returned with a SELECT statement.

## Examples

The following example reports whether a valid text pointer exists for each value in the `logo` column of the `pub_info` table.

### NOTE

To run this example, you must install the **pubs** database.

```
USE pubs;
GO
SELECT pub_id, 'Valid (if 1) Text data'
    = TEXTVALID ('pub_info.logo', TEXTPTR(logo))
FROM pub_info
ORDER BY pub_id;
GO
```

Here is the result set.

```
pub_id Valid (if 1) Text data
----- -----
0736   1
0877   1
1389   1
1622   1
1756   1
9901   1
9952   1
9999   1

(8 row(s) affected)
```

## See Also

- [DATALENGTH \(Transact-SQL\)](#)
- [PATINDEX \(Transact-SQL\)](#)
- [SET TEXTSIZE \(Transact-SQL\)](#)
- [Text and Image Functions \(Transact-SQL\)](#)
- [TEXTPTR \(Transact-SQL\)](#)

# Trigger Functions (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2012) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The following scalar functions can be used in the definition of a trigger to test for changes in data values or to return other data.

## In This Section

[COLUMNS\\_UPDATED](#)

[EVENTDATA](#)

[TRIGGER\\_NESTLEVEL](#)

[UPDATE\(\)](#)

# COLUMNS\_UPDATED (Transact-SQL)

7/31/2017 • 5 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns a **varbinary** bit pattern that indicates the columns in a table or view that were inserted or updated.

COLUMNS\_UPDATED is used anywhere inside the body of a Transact-SQL INSERT or UPDATE trigger to test whether the trigger should execute certain actions.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
COLUMNS_UPDATED ( )
```

## Return types

**varbinary**

## Remarks

COLUMNS\_UPDATED tests for UPDATE or INSERT actions performed on multiple columns. To test for UPDATE or INSERT attempts on one column, use [UPDATE\(\)](#).

COLUMNS\_UPDATED returns one or more bytes that are ordered from left to right, with the least significant bit in each byte being the rightmost. The rightmost bit of the leftmost byte represents the first column in the table; the next bit to the left represents the second column, and so on. COLUMNS\_UPDATED returns multiple bytes if the table on which the trigger is created contains more than eight columns, with the least significant byte being the leftmost. COLUMNS\_UPDATED returns TRUE for all columns in INSERT actions because the columns have either explicit values or implicit (NULL) values inserted.

To test for updates or inserts to specific columns, follow the syntax with a bitwise operator and an integer bitmask of the columns being tested. For example, table **t1** contains columns **C1**, **C2**, **C3**, **C4**, and **C5**. To verify that columns **C2**, **C3**, and **C4** are all updated (with table **t1** having an UPDATE trigger), follow the syntax with **& 14**. To test whether only column **C2** is updated, specify **& 2**.

COLUMNS\_UPDATED can be used anywhere inside a Transact-SQL INSERT or UPDATE trigger.

The ORDINAL\_POSITION column of the INFORMATION\_SCHEMA.COLUMNS view is not compatible with the bit pattern of columns returned by COLUMNS\_UPDATED. To obtain a bit pattern compatible with COLUMNS\_UPDATED, reference the **ColumnID** property of the **COLUMNPROPERTY** system function when you query the **INFORMATION\_SCHEMA.COLUMNS** view, as shown in the following example.

```
SELECT TABLE_NAME, COLUMN_NAME,
COLUMNPROPERTY(OBJECT_ID(TABLE_SCHEMA + '.' + TABLE_NAME),
COLUMN_NAME, 'ColumnId') AS COLUMN_ID
FROM AdventureWorks2012.INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Person';
```

## Column sets

When a column set is defined on a table, the COLUMNS\_UPDATED function behaves in the following ways:

- When a column that is a member of the column set is explicitly updated, the corresponding bit for that column is set to 1, and the bit for the column set is set to 1.
- When a column set is explicitly updated, the bit for the column set is set to 1, and the bits for all of the sparse columns in that table are set to 1.
- For insert operations, all bits are set to 1.

Because changes to a column set cause the bits of all columns in the column set to be set to 1, columns in a column set that were not changed will appear to have been modified. For more information about columns sets, see [Use Column Sets](#).

## Examples

### A. Using COLUMNS\_UPDATED to test the first eight columns of a table

The following example creates two tables: `employeeData` and `auditEmployeeData`. The `employeeData` table holds sensitive employee payroll information and can be modified by members of the human resources department. If the social security number (SSN), yearly salary, or bank account number for an employee is changed, an audit record is generated and inserted into the `auditEmployeeData` audit table.

By using `COLUMNS_UPDATED()`, tests for any changes to the columns that contain sensitive employee information can be quickly made. Using `COLUMNS_UPDATED()` in this way works only when you are trying to detect changes to the first eight columns in the table.

```
USE AdventureWorks2012;
GO
IF EXISTS(SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
          WHERE TABLE_NAME = 'employeeData')
    DROP TABLE employeeData;
IF EXISTS(SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
          WHERE TABLE_NAME = 'auditEmployeeData')
    DROP TABLE auditEmployeeData;
GO
CREATE TABLE dbo.employeeData (
    emp_id int NOT NULL PRIMARY KEY,
    emp_bankAccountNumber char (10) NOT NULL,
    emp_salary int NOT NULL,
    emp_SSN char (11) NOT NULL,
    emp_lname nchar (32) NOT NULL,
    emp_fname nchar (32) NOT NULL,
    emp_manager int NOT NULL
);
GO
CREATE TABLE dbo.auditEmployeeData (
    audit_log_id uniqueidentifier DEFAULT NEWID() PRIMARY KEY,
    audit_log_type char (3) NOT NULL,
    audit_emp_id int NOT NULL,
    audit_emp_bankAccountNumber char (10) NULL,
    audit_emp_salary int NULL,
    audit_emp_SSN char (11) NULL,
    audit_user sysname DEFAULT SUSER_SNAME(),
    audit_changed datetime DEFAULT GETDATE()
);
GO
CREATE TRIGGER dbo.updEmployeeData
ON dbo.employeeData
AFTER UPDATE AS
/*Check whether columns 2, 3 or 4 have been updated. If any or all
columns 2, 3 or 4 have been changed, create an audit record. The
```

```

BITMASK is: power(2,(2-1))+power(2,(3-1))+power(2,(4-1)) = 14. To test
whether all columns 2, 3, and 4 are updated, use = 14 instead of >0
(below).*/

IF (COLUMNS_UPDATED() & 14) > 0
/*Use IF (COLUMNS_UPDATED() & 14) = 14 to see whether all columns 2, 3,
and 4 are updated.*/
BEGIN
-- Audit OLD record.
INSERT INTO dbo.auditEmployeeData
    (audit_log_type,
     audit_emp_id,
     audit_emp_bankAccountNumber,
     audit_emp_salary,
     audit_emp_SSN)
SELECT 'OLD',
      del.emp_id,
      del.emp_bankAccountNumber,
      del.emp_salary,
      del.emp_SSN
FROM deleted del;

-- Audit NEW record.
INSERT INTO dbo.auditEmployeeData
    (audit_log_type,
     audit_emp_id,
     audit_emp_bankAccountNumber,
     audit_emp_salary,
     audit_emp_SSN)
SELECT 'NEW',
      ins.emp_id,
      ins.emp_bankAccountNumber,
      ins.emp_salary,
      ins.emp_SSN
FROM inserted ins;
END;
GO

/*Inserting a new employee does not cause the UPDATE trigger to fire.*/
INSERT INTO employeeData
VALUES ( 101, 'USA-987-01', 23000, 'R-M53550M', N'Mendel', N'Roland', 32);
GO

/*Updating the employee record for employee number 101 to change the
salary to 51000 causes the UPDATE trigger to fire and an audit trail to
be produced.*/

UPDATE dbo.employeeData
    SET emp_salary = 51000
    WHERE emp_id = 101;
GO
SELECT * FROM auditEmployeeData;
GO

/*Updating the employee record for employee number 101 to change both
the bank account number and social security number (SSN) causes the
UPDATE trigger to fire and an audit trail to be produced.*/

UPDATE dbo.employeeData
    SET emp_bankAccountNumber = '133146A0', emp_SSN = 'R-M53550M'
    WHERE emp_id = 101;
GO
SELECT * FROM dbo.auditEmployeeData;
GO

```

## B. Using COLUMNS\_UPDATED to test more than eight columns

To test for updates that affect columns other than the first eight columns in a table, use the `SUBSTRING` function to test the correct bit returned by `COLUMNS_UPDATED`. The following example tests for updates that affect columns `3`, `5`, and `9` in the `AdventureWorks2012.Person.Person` table.

```
USE AdventureWorks2012;
GO
IF OBJECT_ID (N'Person.uContact2', N'TR') IS NOT NULL
    DROP TRIGGER Person.uContact2;
GO
CREATE TRIGGER Person.uContact2 ON Person.Person
AFTER UPDATE AS
    IF ( (SUBSTRING(COLUMNS_UPDATED(),1,1) & 20 = 20)
        AND (SUBSTRING(COLUMNS_UPDATED(),2,1) & 1 = 1) )
        PRINT 'Columns 3, 5 and 9 updated';
GO

UPDATE Person.Person
    SET NameStyle = NameStyle,
        FirstName=FirstName,
        EmailPromotion=EmailPromotion;
GO
```

## See also

[Bitwise Operators \(Transact-SQL\)](#)

[CREATE TRIGGER \(Transact-SQL\)](#)

[UPDATE\(\) \(Transact-SQL\)](#)

# EVENTDATA (Transact-SQL)

7/31/2017 • 3 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns information about server or database events. EVENTDATA is called when an event notification fires, and the results are returned to the specified service broker. EVENTDATA can also be used inside the body of a DDL or logon trigger.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
EVENTDATA( )
```

## Remarks

EVENTDATA returns data only when referenced directly inside of a DDL or logon trigger. EVENTDATA returns null if it is called by other routines, even if those routines are called by a DDL or logon trigger.

Data returned by EVENTDATA is not valid after a transaction that called EVENTDATA, either implicitly or explicitly, commits or is rolled back.

### Caution

EVENTDATA returns XML data. This data is sent to the client as Unicode that uses 2 bytes for each character. The following Unicode code points can be represented in the XML that is returned by EVENTDATA:

```
0x0009  
0x000A  
0x000D  
>= 0x0020 && <= 0xD7FF  
>= 0xE000 && <= 0xFFFF
```

Some characters that can appear in Transact-SQL identifiers and data are not expressible or permissible in XML. Characters or data that have code points not shown in the previous list are mapped to a question mark (?).

To protect the security of logins, when CREATE LOGIN or ALTER LOGIN statements are executed, passwords are not displayed.

## Schemas Returned

EVENTDATA returns a value of type **xml**. By default, the schema definition for all events is installed in the following directory: C:\Program Files\Microsoft SQL Server\nnn\Tools\Binn\schemas\sqlserver\2006\11\events\events.xsd.

Alternatively, the event schema is published at the [Microsoft SQL Server XML Schemas](#) Web page.

To extract the schema for any particular event, search the schema for the Complex Type

`EVENT_INSTANCE_\<event_type>`. For example, to extract the schema for the DROP\_TABLE event, search the schema

for `EVENT_INSTANCE_DROP_TABLE`.

## Examples

### A. Querying event data in a DDL trigger

The following example creates a DDL trigger to prevent new tables from being created in the database. The Transact-SQL statement that fires the trigger is captured by using XQuery against the XML data that is generated by EVENTDATA. For more information, see [XQuery Language Reference \(SQL Server\)](#).

#### NOTE

When you query the `\<TSQLCommand>` element by using **Results to Grid** in SQL Server Management Studio, line breaks in the command text do not appear. Use **Results to Text** instead.

```
USE AdventureWorks2012;
GO
CREATE TRIGGER safety
ON DATABASE
FOR CREATE_TABLE
AS
    PRINT 'CREATE TABLE Issued.'
    SELECT EVENTDATA().value
        ('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]', 'nvarchar(max)')
    RAISERROR ('New tables cannot be created in this database.', 16, 1)
    ROLLBACK
;
GO
--Test the trigger.
CREATE TABLE NewTable (Column1 int);
GO
--Drop the trigger.
DROP TRIGGER safety
ON DATABASE;
GO
```

#### NOTE

When you want to return event data, we recommend that you use the XQuery **value()** method instead of the **query()** method. The **query()** method returns XML and ampersand-escaped carriage return and line feed (CR/LF) instances in the output, while the **value()** method renders CR/LF instances invisible in the output.

### B. Creating a log table with event data in a DDL trigger

The following example creates a table to store information about all database level events, and populates the table with a DDL trigger. The event type and Transact-SQL statement are captured by using XQuery against the XML data generated by `EVENTDATA`.

```
USE AdventureWorks2012;
GO
CREATE TABLE ddl_log (PostTime datetime, DB_User nvarchar(100), Event nvarchar(100), TSQL nvarchar(2000));
GO
CREATE TRIGGER log
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
DECLARE @data XML
SET @data = EVENTDATA()
INSERT ddl_log
(PostTime, DB_User, Event, TSQL)
VALUES
(GETDATE(),
CONVERT(nvarchar(100), CURRENT_USER),
@data.value('/EVENT_INSTANCE/EventType[1]', 'nvarchar(100)'),
@data.value('/EVENT_INSTANCE/TSQLCommand[1]', 'nvarchar(2000)') );
GO
--Test the trigger.
CREATE TABLE TestTable (a int);
DROP TABLE TestTable ;
GO
SELECT * FROM ddl_log ;
GO
--Drop the trigger.
DROP TRIGGER log
ON DATABASE;
GO
--Drop table ddl_log.
DROP TABLE ddl_log;
GO
```

## See Also

[Use the EVENTDATA Function](#)

[DDL Triggers](#)

[Event Notifications](#)

[Logon Triggers](#)

# TRIGGER\_NESTLEVEL (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse  
 Parallel Data Warehouse

Returns the number of triggers executed for the statement that fired the trigger. TRIGGER\_NESTLEVEL is used in DML and DDL triggers to determine the current level of nesting.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
TRIGGER_NESTLEVEL ( [ object_id ] , [ 'trigger_type' ] , [ 'trigger_event_category' ] )
```

## Arguments

### *object\_id*

Is the object ID of a trigger. If *object\_id* is specified, the number of times the specified trigger has been executed for the statement is returned. If *object\_id* is not specified, the number of times all triggers have been executed for the statement is returned.

### *'trigger\_type'*

Specifies whether to apply TRIGGER\_NESTLEVEL to AFTER triggers or INSTEAD OF triggers. Specify **AFTER** for AFTER triggers. Specify **INSTEAD OF** for INSTEAD OF triggers. If *trigger\_type* is specified, *trigger\_event\_category* must also be specified.

### *'trigger\_event\_category'*

Specifies whether to apply TRIGGER\_NESTLEVEL to DML or DDL triggers. Specify **DML** for DML triggers. Specify **DDL** for DDL triggers. If *trigger\_event\_category* is specified, *trigger\_type* must also be specified. Note that only **AFTER** can be specified with **DDL**, because DDL triggers can only be AFTER triggers.

## Remarks

When no parameters are specified, TRIGGER\_NESTLEVEL returns the total number of triggers on the call stack. This includes itself. Omission of parameters can occur when a trigger executes commands causing another trigger to be fired or creates a succession of firing triggers.

To return the total number of triggers on the call stack for a particular trigger type and event category, specify *object\_id* = 0.

TRIGGER\_NESTLEVEL returns 0 if it is executed outside a trigger and any parameters are not NULL.

When any parameters are explicitly specified as NULL, a value of NULL is returned regardless of whether TRIGGER\_NESTLEVEL was used within or external to a trigger.

## Examples

### A. Testing the nesting level of a specific DML trigger

```
IF ( (SELECT TRIGGER_NESTLEVEL( OBJECT_ID('xyz') , 'AFTER' , 'DML' ) ) > 5 )
    RAISERROR('Trigger xyz nested more than 5 levels.',16,-1)
```

## B. Testing the nesting level of a specific DDL trigger

```
IF ( ( SELECT TRIGGER_NESTLEVEL ( ( SELECT object_id FROM sys.triggers
WHERE name = 'abc' ), 'AFTER' , 'DDL' ) ) > 5 )
    RAISERROR ('Trigger abc nested more than 5 levels.',16,-1)
```

## C. Testing the nesting level of all triggers executed

```
IF ( (SELECT trigger_nestlevel() ) > 5 )
    RAISERROR
        ('This statement nested over 5 levels of triggers.',16,-1)
```

## See Also

[CREATE TRIGGER \(Transact-SQL\)](#)

# UPDATE - Trigger Functions (Transact-SQL)

3/24/2017 • 1 min to read • [Edit Online](#)

**THIS TOPIC APPLIES TO:** SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Returns a Boolean value that indicates whether an INSERT or UPDATE attempt was made on a specified column of a table or view. UPDATE() is used anywhere inside the body of a Transact-SQL INSERT or UPDATE trigger to test whether the trigger should execute certain actions.

[Transact-SQL Syntax Conventions](#)

## Syntax

```
UPDATE ( column )
```

## Arguments

*column*

Is the name of the column to test for either an INSERT or UPDATE action. Because the table name is specified in the ON clause of the trigger, do not include the table name before the column name. The column can be of any [data type](#) supported by SQL Server. However, computed columns cannot be used in this context.

## Return Types

Boolean

## Remarks

UPDATE() returns TRUE regardless of whether an INSERT or UPDATE attempt is successful.

To test for an INSERT or UPDATE action for more than one column, specify a separate UPDATE(*column*) clause following the first one. Multiple columns can also be tested for INSERT or UPDATE actions by using COLUMNS\_UPDATED. This returns a bit pattern that indicates which columns were inserted or updated.

IF UPDATE returns the TRUE value in INSERT actions because the columns have either explicit values or implicit (NULL) values inserted.

### NOTE

The IF UPDATE(*column*) clause functions the same as an IF, IF...ELSE, or WHILE clause and can use the BEGIN...END block. For more information, see [Control-of-Flow Language \(Transact-SQL\)](#).

UPDATE(*column*) can be used anywhere inside the body of a Transact-SQL trigger.

## Examples

The following example creates a trigger that prints a message to the client when anyone tries to update the StateProvinceID or PostalCode columns of the Address table.

```
USE AdventureWorks2012;
GO
IF EXISTS (SELECT name FROM sys.objects
    WHERE name = 'reminder' AND type = 'TR')
    DROP TRIGGER Person.reminder;
GO
CREATE TRIGGER reminder
ON Person.Address
AFTER UPDATE
AS
IF ( UPDATE (StateProvinceID) OR UPDATE (PostalCode) )
BEGIN
RAISERROR (50009, 16, 10)
END;
GO
-- Test the trigger.
UPDATE Person.Address
SET PostalCode = 99999
WHERE PostalCode = '12345';
GO
```

## See Also

[COLUMNS\\_UPDATED \(Transact-SQL\)](#)

[CREATE TRIGGER \(Transact-SQL\)](#)