



# Creating a date dimension or calendar table in SQL Server

By: [Aaron Bertrand \(/sqlserverauthor/49/aaron-bertrand/\)](/sqlserverauthor/49/aaron-bertrand/) | Updated: 2020-04-06 | [Comments \(61\)](#) | Related: [1 \(/sqlservertip/4054/creating-a-date-dimension-or-calendar-table-in-sql-server/\)](/sqlservertip/4054/creating-a-date-dimension-or-calendar-table-in-sql-server/) | [2 \(/sqlservertip/4176/the-sql-server-numbers-table-explained--part-1/\)](/sqlservertip/4176/the-sql-server-numbers-table-explained--part-1/) | [3 \(/sqlservertip/4177/the-sql-server-numbers-table-explained--part-2/\)](/sqlservertip/4177/the-sql-server-numbers-table-explained--part-2/) | [More \(/sql-server-developer-resources/\)](/sql-server-developer-resources/) > [Dates \(/sql-server-tip-category/121/dates/\)](/sql-server-tip-category/121/dates/)

## Problem

A calendar table can be immensely useful, particularly for reporting purposes, and for determining things like business days between two dates. I often see people struggling with manually populating a calendar or date dimension table; usually there are lots of loops and iterative code constructs being used. In this tip I will show you how to build and use a calendar table using a set-based solution that is powerful and easily customizable.

## Solution

I build calendar tables all the time, for a variety of business applications, and have come up with a few ways to handle certain details. Sharing them here will hopefully prevent you from re-inventing any wheels when populating your own tables.

One of the biggest objections I hear to calendar tables is that people don't want to create a table. I can't stress enough how cheap a table can be in terms of size and memory usage, especially as underlying storage continues to be larger and faster, compared to using all kinds of functions to determine date-related information in every single query. Twenty or thirty years of dates stored in a table takes a few MBs at most, even less with compression, and if you use them often enough, they'll always be in memory.

I also always explicitly set things like `DATEFORMAT`, `DATEFIRST`, and `LANGUAGE` to avoid ambiguity, default to U.S. English for week starts and for month and day names, and assume that quarters for the fiscal year align with the calendar year. You may need to change some of these specifics depending on your display language, your fiscal year, and other factors.

This is a one-time population, so I'm not worried about speed, even though this specific CTE approach is no slouch. I like to materialize all of the columns to disk, rather than rely on computed columns, since the table becomes read-only after initial population. So I'm going to do a lot of those calculations during the initial series of CTEs (<https://www.sentryone.com/blog/aaronbertrand/backtobasics-ctes>). To start, I'll show the output of each CTE one at a time.

You can change some of these details to experiment on your own. In this example, I'm going to populate the date dimension table with data spanning 30 years, starting from 2010-01-01.

First, we have a recursive CTE that returns a sequence representing the number of days between our start date (2010-01-01) and 30 years later less a day (2039-12-31):



set or regional settings from interfering with  
(1)  
interpretation of dates / literals

[MENU](#)

```
SET DATEFIRST 7, -- 1 = Monday, 7 = Sunday
DATEFORMAT mdy,
LANGUAGE US_ENGLISH;
-- assume the above is here in all subsequent code blocks.

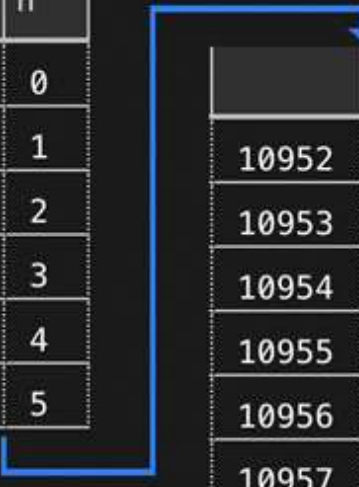
DECLARE @StartDate date = '20100101';

DECLARE @CutoffDate date = DATEADD(DAY, -1, DATEADD(YEAR, 30, @StartDate));

;WITH seq(n) AS
(
    SELECT 0 UNION ALL SELECT n + 1 FROM seq
    WHERE n < DATEDIFF(DAY, @StartDate, @CutoffDate)
)
SELECT n FROM seq
ORDER BY n
OPTION (MAXRECURSION 0);
```

This returns the following list of numbers:

	n
1	0
2	1
3	2
4	3
5	4
6	5



	n
10952	10951
10953	10952
10954	10953
10955	10954
10956	10955
10957	10956

Next, we can add a second CTE that translates those numbers into all the dates in our range:



```
DECLARE @StartDate date = '20100101';
```

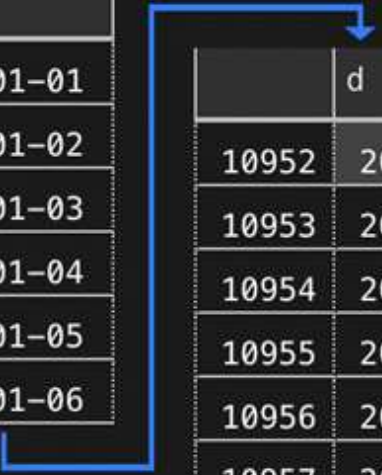
[MENU](#)

```
DECLARE @CutoffDate date = DATEADD(DAY, -1, DATEADD(YEAR, 30, @StartDate));

;WITH seq(n) AS
(
    SELECT 0 UNION ALL SELECT n + 1 FROM seq
    WHERE n < DATEDIFF(DAY, @StartDate, @CutoffDate)
),
d(d) AS
(
    SELECT DATEADD(DAY, n, @StartDate) FROM seq
)
SELECT d FROM d
ORDER BY d
OPTION (MAXRECURSION 0);
```

Which returns the following range of dates:

	d
1	2010-01-01
2	2010-01-02
3	2010-01-03
4	2010-01-04
5	2010-01-05
6	2010-01-06



	d
10952	2039-12-26
10953	2039-12-27
10954	2039-12-28
10955	2039-12-29
10956	2039-12-30
10957	2039-12-31

Now, we can start extending those dates with information commonly vital to calendar tables / date dimensions. Many are bits of information you can extract from the date, but it's more convenient to have them readily available in a view or table than it is to have every query calculate them inline. I'm working a little backward here, but I'm going to create an intermediate CTE to extract exactly once some computations I'll later have to make multiple times. This query:



```

--StartDate date = '20100101';
--(2)

```

[MENU](#)

```

DECLARE @CutoffDate date = DATEADD(DAY, -1, DATEADD(YEAR, 30, @StartDate));

WITH seq(n) AS
(
    SELECT 0 UNION ALL SELECT n + 1 FROM seq
    WHERE n < DATEDIFF(DAY, @StartDate, @CutoffDate)
),
d(d) AS
(
    SELECT DATEADD(DAY, n, @StartDate) FROM seq
),
src AS
(
    SELECT
        TheDate      = CONVERT(date, d),
        TheDay       = DATEPART(DAY,      d),
        TheDayName   = DATENAME(WEEKDAY,  d),
        TheWeek      = DATEPART(WEEK,     d),
        TheISOWeek   = DATEPART(ISO_WEEK, d),
        TheDayOfWeek = DATEPART(WEEKDAY,  d),
        TheMonth     = DATEPART(MONTH,    d),
        TheMonthName = DATENAME(MONTH,    d),
        TheQuarter   = DATEPART(Quarter,  d),
        TheYear      = DATEPART(YEAR,     d),
        TheFirstOfMonth = DATEFROMPARTS(YEAR(d), MONTH(d), 1),
        TheLastOfYear = DATEFROMPARTS(YEAR(d), 12, 31),
        TheDayOfYear = DATEPART(DAYOFYEAR, d)
    FROM d
)
SELECT * FROM src
ORDER BY TheDate
OPTION (MAXRECURSION 0);

```

Yields this data:

	TheDate	TheDay	TheDayName	TheWeek	TheISOWeek	TheDayOfWeek	TheMonth	TheMonthName	TheQuarter	TheYear	TheFirstOfMonth	TheLastOfYear	TheDayOfYear
1	2010-01-01	1	Friday	1	53	6	1	January	1	2010	2010-01-01	2010-12-31	1
2	2010-01-02	2	Saturday	1	53	7	1	January	1	2010	2010-01-01	2010-12-31	2
3	2010-01-03	3	Sunday	2	53	1	1	January	1	2010	2010-01-01	2010-12-31	3
4	2010-01-04	4	Monday	2	1	2	1	January	1	2010	2010-01-01	2010-12-31	4
5	2010-01-05	5	Tuesday	2	1	3	1	January	1	2010	2010-01-01	2010-12-31	5
6	2010-01-06	6	Wednesday	2	1	4	1	January	1	2010	2010-01-01	2010-12-31	6
...													
10952	2039-12-26	26	Monday	53	52	2	12	December	4	2039	2039-12-01	2039-12-31	360
10953	2039-12-27	27	Tuesday	53	52	3	12	December	4	2039	2039-12-01	2039-12-31	361
10954	2039-12-28	28	Wednesday	53	52	4	12	December	4	2039	2039-12-01	2039-12-31	362
10955	2039-12-29	29	Thursday	53	52	5	12	December	4	2039	2039-12-01	2039-12-31	363
10956	2039-12-30	30	Friday	53	52	6	12	December	4	2039	2039-12-01	2039-12-31	364
10957	2039-12-31	31	Saturday	53	52	7	12	December	4	2039	2039-12-01	2039-12-31	365

If you wanted your fiscal year aligned differently, you could change the year and quarter calculations, or add additional columns. Let's say your fiscal year starts October 1st, then depending on whether that's 9 months late or 3 months early, you could just substitute `d` for a `DATEADD` expression:



(SELECT d FROM  
(

[MENU](#)

```
VALUES('20200101'),  
      ('20200401'),  
      ('20200701'),  
      ('20201001')  
) AS d(d))
```

```
SELECT  
d,  
StandardQuarter      = DATEPART(QUARTER, d),  
LateFiscalQuarter    = DATEPART(QUARTER, DATEADD(MONTH, -9, d)),  
LateFiscalQuarterYear = YEAR(DATEADD(MONTH, -9, d)),  
EarlyFiscalQuarter    = DATEPART(QUARTER, DATEADD(MONTH, 3, d)),  
EarlyFiscalQuarterYear = YEAR(DATEADD(MONTH, 3, d))  
FROM q;
```

d	StandardQuarter	LateFiscalQuarter	LateFiscalQuarterYear	EarlyFiscalQuarter	EarlyFiscalQuarterYear
20200101	1	2	2019	2	2020
20200401	2	3	2019	3	2020
20200701	3	4	2019	4	2020
20201001	4	1	2020	1	2021

Whatever my source data is, I can build on those parts and get much more detail about each date:



```
StartDate date = '20100101';
```

[MENU](#)

```
DECLARE @CutoffDate date = DATEADD(DAY, -1, DATEADD(YEAR, 30, @StartDate));
```

```
;WITH seq(n) AS
```

```
(
    SELECT 0 UNION ALL SELECT n + 1 FROM seq
    WHERE n < DATEDIFF(DAY, @StartDate, @CutoffDate)
),
```

```
d(d) AS
```

```
(
    SELECT DATEADD(DAY, n, @StartDate) FROM seq
),
```

```
src AS
```

```
(
    SELECT
        TheDate          = CONVERT(date, d),
        TheDay           = DATEPART(DAY, d),
        TheDayName       = DATENAME(WEEKDAY, d),
        TheWeek          = DATEPART(WEEK, d),
        TheISOWeek       = DATEPART(ISO_WEEK, d),
        TheDayOfWeek     = DATEPART(WEEKDAY, d),
        TheMonth         = DATEPART(MONTH, d),
        TheMonthName     = DATENAME(MONTH, d),
        TheQuarter       = DATEPART(Quarter, d),
        TheYear          = DATEPART(YEAR, d),
        TheFirstOfMonth  = DATEFROMPARTS(YEAR(d), MONTH(d), 1),

```

This adds supplemental information about any given date, such as the first of period / last of period the date falls within, whether it is a leap year, a few popular string formats, and some specific ISO 8601 specifics (I'll talk more about those in another tip). You may only want some of these columns, and you may want others, too. When you're happy with the output, you can change this line:

```
SELECT * FROM dim
```

To this:

```
SELECT * INTO dbo.DateDimension FROM dim
```

Then you can add a clustered primary key (and any other indexes you want to have handy):

```
CREATE UNIQUE CLUSTERED INDEX PK_DateDimension ON dbo.DateDimension(TheDate);
```

To give an idea of how much space this table really takes, even with all those columns that you probably don't need, the max is about 2MB with a regular clustered index defined on the `TheDate` column, all the way down to 500KB for a clustered columnstore index compressed with `COLUMNSTORE_ARCHIVE` (not necessarily something you should do, depending on the workload that will work against this table, but since it is effectively read only, the DML overhead isn't really a consideration):



Structure	Reserved Size (KB)
Clustered Index	1,992
Clustered Index, row compression	1,672
Clustered Index, page compression	1,032
Clustered Columnstore	584
Clustered Columnstore_Archive	456

Next, we need to talk about holidays, one of the primary seasons you need to use a calendar table instead of relying on built-in date/time functions. In the original version of this tip, I added an `IsHoliday` column, but as a comment rightly pointed out, this set is probably best held in a separate table:

```
CREATE TABLE dbo.HolidayDimension
(
    TheDate date NOT NULL,
    HolidayText nvarchar(255) NOT NULL,
    CONSTRAINT FK_DateDimension FOREIGN KEY(TheDate) REFERENCES dbo.DateDimension(TheDate)
);

CREATE CLUSTERED INDEX CIX_HolidayDimension ON dbo.HolidayDimension(TheDate);
GO
```

This allows you to have more than one holiday for any given date, and in fact allows for multiple entire calendars each with their own set of holidays (imagine an additional column specifying the CalendarID).

Populating the holiday dimension table can be complex. Since I am in the United States, I'm going to deal with statutory holidays here; of course, if you live in another country, you'll need to use different logic. You'll also need to add your own company's holidays manually, but hopefully if you have things that are deterministic, like bank holidays, Boxing Day, or the third Monday of July is your annual off-site arm-wrestling tournament, you should be able to do most of that without much work by following the same sort of pattern I use below. You may also have to add some logic if your company observes weekend holidays on the previous or following weekday, which gets even more complex if those happen to collide with other company- or industry-specific non-business days. We can add most of the traditional holidays with a single pass and rather simple criteria:



(1)

MENU

```
SELECT
    TheDate,
    TheFirstOfYear,
    TheDayOfWeekInMonth,
    TheMonth,
    TheDayName,
    TheDay,
    TheLastDayOfWeekInMonth = ROW_NUMBER() OVER
    (
        PARTITION BY TheFirstOfMonth, TheDayOfWeek
        ORDER BY TheDate DESC
    )
FROM dbo.DateDimension
),
s AS
(
    SELECT TheDate, HolidayText = CASE
    WHEN (TheDate = TheFirstOfYear)
        THEN 'New Year''s Day'
    WHEN (TheDayOfWeekInMonth = 3 AND TheMonth = 1 AND TheDayName = 'Monday')
        THEN 'Martin Luther King Day'      -- (3rd Monday in January)
    WHEN (TheDayOfWeekInMonth = 3 AND TheMonth = 2 AND TheDayName = 'Monday')
        THEN 'President''s Day'             -- (3rd Monday in February)
    WHEN (TheLastDayOfWeekInMonth = 1 AND TheMonth = 5 AND TheDayName = 'Monday')
        THEN 'Memorial Day'                -- (Last Monday in May)
```

Black Friday is a little trickier, because it's the Friday after the fourth Thursday in November. Usually that makes it the fourth Friday, but several times a century it is actually the fifth Friday, so the `UNION ALL` above just grabs the day after each Thanksgiving Day.

**And then there's Easter.** This has always been a complicated problem; the rules for calculating the exact date are so convoluted (<https://en.wikipedia.org/wiki/Easter#Date>), I suspect most people can only mark those dates where they have physical calendars they can look at to confirm. If your company doesn't recognize Easter, you can skip ahead; if it does, you can use the following function, which will return the Easter holiday dates for any given year:





```
FUNCTION dbo.GetEasterHolidays(@TheYear INT)
```

[MENU](#)

```
RETURNS TABLE
```

```
WITH SCHEMABINDING
```

```
AS
```

```
RETURN
```

```
(
```

```
WITH x AS
```

```
(
```

```
SELECT TheDate = DATEFROMPARTS(@TheYear, [Month], [Day])
```

```
FROM (SELECT [Month], [Day] = DaysToSunday + 28 - (31 * ([Month] / 4))
```

```
FROM (SELECT [Month] = 3 + (DaysToSunday + 40) / 44, DaysToSunday
```

```
FROM (SELECT DaysToSunday = paschal - ((@TheYear + (@TheYear / 4) + paschal - 13) % 7)
```

```
FROM (SELECT paschal = epact - (epact / 28)
```

```
FROM (SELECT epact = (24 + 19 * (@TheYear % 19)) % 30)
```

```
AS epact) AS paschal) AS dts) AS m) AS d
```

```
)
```

```
SELECT TheDate, HolidayText = 'Easter Sunday' FROM x
```

```
UNION ALL SELECT DATEADD(DAY, -2, TheDate), 'Good Friday' FROM x
```

```
UNION ALL SELECT DATEADD(DAY, 1, TheDate), 'Easter Monday' FROM x
```

```
);
```

```
GO
```

(You can adjust the function easily, depending on whether they recognize just Easter Sunday or also Good Friday and/or Easter Monday. There is also another tip [here](https://sqlservertip.com/2015/03/tsql-function-to-determine-holidays-in-sql-server/) (<https://sqlservertip.com/2015/03/tsql-function-to-determine-holidays-in-sql-server/>) that will show you how to determine the date for Mardi Gras, given the date for Easter.)

Now, to use that function to add the Easter holidays to the `HolidayDimension` table:

```
INSERT dbo.HolidayDimension(TheDate, HolidayText)
```

```
SELECT d.TheDate, h.HolidayText
```

```
FROM dbo.DateDimension AS d
```

```
CROSS APPLY dbo.GetEasterHolidays(d.TheYear) AS h
```

```
WHERE d.TheDate = h.TheDate;
```

Finally, you can create a view that bridges these two tables (or multiple views):

**SELECT**

```
d.TheDate,  
d.TheDay,  
d.TheDaySuffix,  
d.TheDayName,  
d.TheDayOfWeek,  
d.TheDayOfWeekInMonth,  
d.TheDayOfYear,  
d.IsWeekend,  
d.TheWeek,  
d.TheISOweek,  
d.TheFirstOfWeek,  
d.TheLastOfWeek,  
d.TheWeekOfMonth,  
d.TheMonth,  
d.TheMonthName,  
d.TheFirstOfMonth,  
d.TheLastOfMonth,  
d.TheFirstOfNextMonth,  
d.TheLastOfNextMonth,  
d.TheQuarter,  
d.TheFirstOfQuarter,  
d.TheLastOfQuarter,  
d.TheYear,  
d.TheISOYear,  
d.TheFirstOfYear,  
d.TheLastOfYear,  
d.IsLeapYear,  
d.Has53Weeks,  
d.Has53ISOWeeks,  
d.MMYYYY,  
d.Style101,  
d.Style103,  
d.Style112,  
d.Style120,  
IsHoliday = CASE WHEN h.TheDate IS NOT NULL THEN 1 ELSE 0 END,  
h.HolidayText  
FROM dbo.DateDimension AS d  
LEFT OUTER JOIN dbo.HolidayDimension AS h  
ON d.TheDate = h.TheDate;
```

And now you have a functional calendar view you can use for all of your reporting or business needs.

## Summary

Creating a dimension or calendar table for business dates and fiscal periods might seem intimidating at first, but once you have a solid methodology in line, it can be very worthwhile. There are many ways to do this; some will subscribe to the idea that many of these date-related facts can be derived at query time, or at least be non-persisted computed columns. You will have to decide if the values are calculated often enough to justify the additional space on disk and in the buffer pool.

If you are using Enterprise Edition on SQL Server 2014 or above, you could consider using In-Memory OLTP, and possibly even a non-durable table that you rebuild using a startup procedure. Or on any version or edition, you could put the calendar table into its own filegroup (or database), and mark it as read-only after initial population (this won't force the table to stay in memory all the time, but it will reduce other types of contention).

- Build a persisted calendar table to help with reporting queries, business logic, and gathering additional facts about given dates.
- See these related tips and other resources:
  - [SQL Server Function to return a range of dates \(/sqlservertip/2800/sql-server-function-to-return-a-range-of-dates/\)](/sqlservertip/2800/sql-server-function-to-return-a-range-of-dates/)
  - [TSQL Function to Determine Holidays in SQL Server \(/sqlservertip/1537/tsql-function-to-determine-holidays-in-sql-server/\)](/sqlservertip/1537/tsql-function-to-determine-holidays-in-sql-server/)
  - [SQL Server Date Time Calculation Examples \(/sqlservertip/3508/sql-server-date-time-calculation-examples/\)](/sqlservertip/3508/sql-server-date-time-calculation-examples/)
  - [Built in Time Dimension and Time Intelligence in SQL Server Analysis \(/sqlservertip/1454/built-in-time-dimension-and-time-intelligence-in-sql-server-analysis/\)](/sqlservertip/1454/built-in-time-dimension-and-time-intelligence-in-sql-server-analysis/)
  - [All Dates Tips \(/sql-server-tip-category/121/dates/\)](/sql-server-tip-category/121/dates/)

Last Updated: 2020-04-06

## About the author



(/sqlserverauthor/49/aaron-bertrand/) Aaron Bertrand (@AaronBertrand) is a passionate technologist with industry experience dating back to Classic ASP and SQL Server 6.5. He is editor-in-chief of the performance-related blog, SQLPerformance.com, and also blogs at sqlblog.org.

[View all my tips \(/sqlserverauthor/49/aaron-bertrand/\)](/sqlserverauthor/49/aaron-bertrand/)

### Related Resources

- [Creating a date dimension or calendar table in SQL... \(/sqlservertip/4054/creating-a-date-dimension-or-calendar-table-in-sql-server/\)](/sqlservertip/4054/creating-a-date-dimension-or-calendar-table-in-sql-server/)
- [The SQL Server Numbers Table, Explained - Part 1... \(/sqlservertip/4176/the-sql-server-numbers-table-explained--part-1/\)](/sqlservertip/4176/the-sql-server-numbers-table-explained--part-1/)
- [The SQL Server Numbers Table, Explained - Part 2... \(/sqlservertip/4177/the-sql-server-numbers-table-explained--part-2/\)](/sqlservertip/4177/the-sql-server-numbers-table-explained--part-2/)
- [More Database Developer Tips... \(/sql-server-developer-resources/\)](/sql-server-developer-resources/)

## Follow

- [Get Free SQL Tips \(/get-free-sql-server-tips/?ref=GetFooterMenu\)](/get-free-sql-server-tips/?ref=GetFooterMenu)
- [Twitter \(https://twitter.com/mssqltips\)](https://twitter.com/mssqltips)
- [LinkedIn \(https://www.linkedin.com/groups/2320891/\)](https://www.linkedin.com/groups/2320891/)
- [Facebook \(https://www.facebook.com/mssqltips/\)](https://www.facebook.com/mssqltips/)
- [Pinterest \(https://www.pinterest.com/mssqltips/\)](https://www.pinterest.com/mssqltips/)
- [RSS \(https://feeds.feedburner.com/MSSQLTips-LatestSqlServerTips\)](https://feeds.feedburner.com/MSSQLTips-LatestSqlServerTips)

## Learning

- [DBAs \(/sql-server-dba-resources/\)](/sql-server-dba-resources/)
- [Developers \(/sql-server-developer-resources/\)](/sql-server-developer-resources/)
- [BI Professionals \(/sql-server-business-intelligence-resources/\)](/sql-server-business-intelligence-resources/)
- [Careers \(/sql-server-professional-development-resources/\)](/sql-server-professional-development-resources/)
- [Today's Tip \(/todays-sql-server-tip/\)](/todays-sql-server-tip/)

## Resources

- [Tutorials \(/sql-server-tutorials/\)](/sql-server-tutorials/)
- [Webcasts \(/sql-server-webcasts/\)](/sql-server-webcasts/)
- [Whitepapers \(/sql-server-whitepapers/\)](/sql-server-whitepapers/)
- [Tools \(/sql-server-tools/\)](/sql-server-tools/)

## Search

- [Tip Categories \(/sql-server-categories/\)](/sql-server-categories/)
- [Search By TipID \(/search-tip-id/\)](/search-tip-id/)

## Community

- [First Timer? \(/learn-more-about-mssqltips/\)](/learn-more-about-mssqltips/)
- [Pictures \(/mssqltips-community/1/\)](/mssqltips-community/1/)



[SQL Server mssqltips authors/](#)  
(1).

- [Contribute \(/contribute/\)](#)
- [Event Calendar \(/sql-server-event-list/\)](#)
- [User Groups \(/sql-server-user-groups/\)](#)
- [Author of the Year \(/mssqltips-author-of-year/\)](#)

**MENU**

## More Info

- [Join \(/get-free-sql-server-tips/?ref=JoinFooterMenu\)](#)
- [About \(/about/\)](#)
- [Copyright \(/copyright/\)](#)
- [Privacy \(/privacy/\)](#)
- [Disclaimer \(/disclaimer/\)](#)
- [Feedback \(/feedback/\)](#)
- [Advertise \(/advertise/\)](#)

---

Copyright (c) 2006-2020 [Edgewood Solutions, LLC \(https://www.edgewoodsolutions.com\)](https://www.edgewoodsolutions.com) All rights reserved

Some names and products listed are the registered trademarks of their respective owners.