# Performance Test of New Date Manipulation Functions (SQL Spackle)

By Wayne Sheffield

***Introduction***

SQL Server 2012 introduces several new date/time functions that allow you to build a date/time from the individual parts of a date/time value. As I was experimenting with them, I was thinking that this is pretty neat, and bound to be very useful. And then I started wondering how well these functions performed.

Before I had time to perform a test, I was working on something else where I needed to truncate a date/time value to the first day of the month. I immediately whipped out my tool belt – it has the routine that I learned from Lynn Pettis a few years back on the forums here from [his blog post on Common Date Routines](https://www.sqlservercentral.com/blogs/lynnpettis/archive/2009/03/25/some-common-date-routines.aspx). This routine does the DateAdd(MONTH, DateDiff(MONTH, '1900-01-01', @Date), '1900-01-01') trick to see how many month boundaries exist from a known date to the tested date, and then to add this number of month boundaries back to the known date. With the month datepart being used, the result ends up being the first of the month of the tested date. Using other dateparts gives us other dates with remainders being truncated. See Lynn’s blog for more uses.

As I was looking at that wonderfully efficient code, I realized that I could get the same results with the [DateTimeFromParts](http://msdn.microsoft.com/en-us/library/hh213233%28v=sql.110%29.aspx) function in SQL Server 2012. So I coded an example, and it worked beautifully.

And now, I had a burning question. Performance-wise, just how do these different methods of truncating dates compare against each other? So, stop everything that I’m doing, and code a test.

***The Test Data***

First things first. A performance test requires a bunch of test data. And I mean a bunch. So, I grabbed some code from an [article](http://www.sqlservercentral.com/articles/formatting/72066/) that Jeff Moden wrote at the beginning of 2011. Sure enough, there’s a million row dataset of random dates… which is exactly what I’m looking for. So, by borrowing some code from Jeff, I have my test data:

SELECT TOP 1000000

RowNum = IDENTITY(INT,1,1),

SomeDate = CAST(RAND(CHECKSUM(NEWID()))\*3653.0+36524.0 AS DATETIME)

INTO dbo.DateTest

FROM sys.All\_Columns t1,

sys.All\_Columns t2;

Next up is to generate a test and to produce timings. In the following code, I truncate each date to the first day of the year, month and quarter, and then I truncate the time down to the hour, minute, and second. I perform this conversion four times: the first time using Lynn’s DateAdd/DateDiff code; the second time by stripping out the various parts of the date, and putting it back together with the DateTimeFromParts function; the third time is using pre-extracted date/time information in this table; and the fourth time is using the new SQL Server 2012 [Format](http://msdn.microsoft.com/en-us/library/hh213505%28v=sql.110%29.aspx) function. All queries dump the results into temp tables. I run this block of code with SET STATISTICS TIME ON just before each query to capture the time to run each query, and that batch is performed 25 times.

***The Test***

First up, the code:

SET STATISTICS TIME OFF;

USE master;

GO

IF NOT EXISTS (SELECT 1 FROM sys.databases WHERE name = 'SandBox')

CREATE DATABASE SandBox;

GO

USE SandBox;

GO

-- Build a million-row test table with random dates

IF OBJECT\_ID('dbo.DateTest','U') IS NULL

SELECT TOP 1000000

RowNum = IDENTITY(INT,1,1),

SomeDate = CAST(RAND(CHECKSUM(NEWID()))\*3653.0+36524.0 AS DATETIME)

INTO dbo.DateTest

FROM sys.All\_Columns t1,

sys.All\_Columns t2;

GO

-- Add columns to have pre-defined the parts of this date

IF NOT EXISTS (SELECT 1 FROM sys.columns WHERE name = 'Yr' AND object\_id = OBJECT\_ID('dbo.DateTest','U'))

ALTER TABLE dbo.DateTest

ADD Yr SMALLINT,

Mth TINYINT,

Dy TINYINT,

Hr TINYINT,

Mn TINYINT,

Sec TINYINT,

Qtr TINYINT;

GO

-- Populate the columns with the appropriate data from the date

UPDATE dbo.DateTest

SET Yr = DATEPART(YEAR, SomeDate),

Mth = DATEPART(MONTH, SomeDate),

Dy = DATEPART(DAY, SomeDate),

Hr = DATEPART(HOUR, SomeDate),

Mn = DATEPART(MINUTE, SomeDate),

Sec = DATEPART(SECOND, SomeDate),

Qtr = ((CEILING(MONTH(SomeDate)/3.0)\*3)-2);

-- Declare and populate variables for the date math.

DECLARE @Base DATETIME = '1900-01-01T00:00:00',

@Base2 DATETIME = '2000-01-01T00:00:00';

-- Ensure temp tables don't exist yet (for subsequent runs)

IF OBJECT\_ID('tempdb..#test1','U') IS NOT NULL DROP TABLE #test1;

IF OBJECT\_ID('tempdb..#test2','U') IS NOT NULL DROP TABLE #test2;

IF OBJECT\_ID('tempdb..#test3','U') IS NOT NULL DROP TABLE #test3;

IF OBJECT\_ID('tempdb..#test4','U') IS NOT NULL DROP TABLE #test4;

-- Test #1: Use the DateAdd/DateDiff method.

RAISERROR ('DateAdd/DateDiff', 10, 1) WITH NOWAIT;

SET STATISTICS TIME ON;

SELECT SomeDate,

DATEADD(YEAR, DATEDIFF(YEAR, @Base, SomeDate), @Base) AS [FirstDayOfYear],

DATEADD(MONTH, DATEDIFF(MONTH, @Base, SomeDate), @Base) AS [FirstDayOfMonth],

DATEADD(QUARTER, DATEDIFF(QUARTER, @Base, SomeDate), @Base) AS [FirstDayOfQuarter],

DATEADD(HOUR, DATEDIFF(HOUR, @Base, SomeDate), @Base) AS [StartOfHour],

DATEADD(MINUTE, DATEDIFF(MINUTE, @Base, SomeDate), @Base) AS [StartOfMinute],

DATEADD(SECOND, DATEDIFF(SECOND, @Base2, SomeDate), @Base2) AS [StartOfSecond]

INTO #test1

FROM dbo.DateTest;

SET STATISTICS TIME OFF;

-- Test #2: Extract out the parts of the date, and put them back together in the DateTimeFromParts function.

RAISERROR ('DateTimeFromParts1', 10, 1) WITH NOWAIT;

SET STATISTICS TIME ON;

SELECT SomeDate,

DATETIMEFROMPARTS(ca.Yr, 1, 1, 0, 0, 0, 0) AS [FirstDayOfYear],

DATETIMEFROMPARTS(ca.Yr, ca.Mth, 1, 0, 0, 0, 0) AS [FirstDayOfMonth],

DATETIMEFROMPARTS(ca.Yr, ca.Qtr, 1, 0, 0, 0, 0) AS [FirstDayOfQuarter],

DATETIMEFROMPARTS(ca.Yr, ca.Mth, ca.Dy, ca.Hr, 0, 0, 0) AS StartOfHour,

DATETIMEFROMPARTS(ca.Yr, ca.Mth, ca.Dy, ca.Hr, ca.Mn, 0, 0) AS StartOfMinute,

DATETIMEFROMPARTS(ca.Yr, ca.Mth, ca.Dy, ca.Hr, ca.Mn, ca.Sec, 0) AS StartOfSecond

INTO #test2

FROM dbo.DateTest dt

CROSS APPLY (SELECT DATEPART(YEAR, dt.SomeDate) AS Yr,

DATEPART(MONTH, dt.SomeDate) AS Mth,

((CEILING(MONTH(dt.SomeDate)/3.0)\*3)-2) AS Qtr,

DATEPART(DAY, dt.SomeDate) AS Dy,

DATEPART(HOUR, dt.SomeDate) AS Hr,

DATEPART(MINUTE, dt.SomeDate) AS Mn,

DATEPART(SECOND, dt.SomeDate) AS Sec ) ca;

SET STATISTICS TIME OFF;

-- Test #3: Build the date from the pre-extracted appropriate parts with the DateTimeFromParts function.

RAISERROR ('DateTimeFromParts2', 10, 1) WITH NOWAIT;

SET STATISTICS TIME ON;

SELECT SomeDate,

DATETIMEFROMPARTS(dt.Yr, 1, 1, 0, 0, 0, 0) AS [FirstDayOfYear],

DATETIMEFROMPARTS(dt.Yr, dt.Mth, 1, 0, 0, 0, 0) AS [FirstDayOfMonth],

DATETIMEFROMPARTS(dt.Yr, dt.Qtr, 1, 0, 0, 0, 0) AS [FirstDayOfQuarter],

DATETIMEFROMPARTS(dt.Yr, dt.Mth, dt.Dy, dt.Hr, 0, 0, 0) AS StartOfHour,

DATETIMEFROMPARTS(dt.Yr, dt.Mth, dt.Dy, dt.Hr, dt.Mn, 0, 0) AS StartOfMinute,

DATETIMEFROMPARTS(dt.Yr, dt.Mth, dt.Dy, dt.Hr, dt.Mn, dt.Sec, 0) AS StartOfSecond

INTO #test3

FROM dbo.DateTest dt;

SET STATISTICS TIME OFF;

-- Test #4: Utilize the new Format function.

-- Note: the FirstDayOfQuarter can't be done with Format, so DateAdd/DateDiff is used.

-- Just because I like it.

RAISERROR ('Format test', 10, 1) WITH NOWAIT;

SET STATISTICS TIME ON;

SELECT SomeDate,

CAST(FORMAT(SomeDate, 'yyyy-01-01') AS DATETIME) AS [FirstDayOfYear],

CAST(FORMAT(SomeDate, 'yyyy-MM-01') AS DATETIME) AS [FirstDayOfMonth],

DATEADD(QUARTER, DATEDIFF(QUARTER, @Base, SomeDate), @Base) AS [FirstDayOfQuarter],

CAST(FORMAT(SomeDate, 'yyyy-MM-dd HH:00:00') AS DATETIME) AS StartOfHour,

CAST(FORMAT(SomeDate, 'yyyy-MM-dd HH:mm:00') AS DATETIME) AS StartOfMinute,

CAST(FORMAT(SomeDate, 'yyyy-MM-d HH:mm:ss') AS DATETIME) AS StartOfSecond

INTO #test4

FROM dbo.DateTest;

SET STATISTICS TIME OFF;

GO 25

***The Results***

I then took the results from the time statistics, and used [Jeff’s DelimitedSplit function](http://www.sqlservercentral.com/articles/Tally+Table/72993/) to get the timings from each query into a result set. This data was coped into Excel, where the following line graphs tell the rest of the story:

Well, those graphs don’t tell the whole story… what these line graphs tell us is that the Format function is so inefficient, that it skews the graph and we’re not able to really compare the remaining methods. So, let’s remove the Format from the graphs, and look at them again.

***My Analysis***

To perform the same task (take an existing date, and truncate it to different levels), the DateAdd/DateDiff method is about twice as fast as the DateTimeFromParts function. However, if the individual parts of the date are already available, then the two methods are an essential tie, with the DateTimeFromParts having a very slight advantage. And, obviously, the Format function takes about 100 times longer than the fastest method here.

I think I’ll continue to use Lynn’s DateAdd/DateDiff method when I need to truncate a date.

(Test environment: Windows 7 x64; SQL Server 2012 x64 RTM; 8GB ram)

***Links***

Common Date Routines

<http://www.sqlservercentral.com/blogs/lynnpettis/2009/03/25/some-common-date-routines/>

Formatting Dates with 3 Character Months (SQL Spackle)

<http://www.sqlservercentral.com/articles/formatting/72066/>

Tally OH! An Improved SQL 8K “CSV Splitter” Function:<http://www.sqlservercentral.com/articles/Tally+Table/72993/>

DateTimeFromParts:

<http://msdn.microsoft.com/en-us/library/hh213233(v=sql.110).aspx>

Format:

<http://msdn.microsoft.com/en-us/library/hh213505(v=sql.110).aspx>