The Fastest Way to Combine DATE and TIME Data Types to a DATETIME

Dwain Camps, 2015-09-08 (first published: 2015-08-28)

Recently while working with a table where there were two columns, one a DATE datatype and a second TIME datatype, I found that those needed to be combined into a single DATETIME result column. I won't go into why the DATE and TIME were separated in this case, but suffice it to say that it seemed like a good idea at the time.

Just like most date arithmetic in T-SQL, there are many ways to do this. Let's look at the first thought that came to my mind.

```
DECLARE @MyDate   DATE = '2015-08-27'
    ,@MyTime      TIME = '15:33:21.057';
SELECT MyDateTime1=CAST(@MyDate AS DATETIME) + CAST(@MyTime
AS DATETIME);
```

This works because in T-SQL it is possible to add (or subtract) two DATETIME values, and you'll find that it produces exactly the desired result, which is: **2015-08-27 15:33:21.057**.

Knowing from past experience that the first query form that jumps into your brain, isn't always the fastest one from a performance perspective, I decided to explore some other alternatives.

Here's another way that will work:

```
SELECT MyDateTime2=DATEADD(millisecond
                    ,DATEDIFF(millisecond, 0, @MyTime)
                    ,CAST(@MyDate AS DATETIME));
```

If you are unfamiliar with the DATEADD and DATEDIFF functions, you can use those links to get to the Microsoft Books-on-Line reference, or you can see some examples in one of my prior blogs: Manipulating Dates and Times in T-SQL.

Here's yet another:

```
SELECT MyDateTime3=DATEADD(day
                    ,DATEDIFF(day, 0, @MyDate)
                    ,CAST(@MyTime AS DATETIME));
```

Finally, we'll get a little creative and use a VARCHAR intermediate result to come up with this:

```
SELECT MyDateTime4=CAST(
                    CAST(@MyDate AS VARCHAR(10)) + ' ' +
CAST(@MyTime AS VARCHAR(12))
                    AS DATETIME);
```

So now we have some choices, but which do we use?

The first one is the shortest in terms of keystrokes, and arguably the easiest one to understand, although some might argue that the third one is easier to understand.

From my perspective though, and I've said this before and I'm likely to say it again, performance always counts. Even when this tiny fragment of code is part of a much larger T-SQL query, getting the fastest speed out of that complex query means getting the fastest speed out of each of its tiniest parts.

So naturally we'll turn to one of our favorite performance tools, the One Million Row Test Harness. It is so easy to create one of these for this case it is almost embarrassing, the embarrassing part of course being how few people might consider doing so. Here it is.

```
WITH Tally (n) AS
(
    SELECT TOP (1000000) ROW_NUMBER() OVER (ORDER BY (SELECT
NULL))
    FROM sys.all_columns a CROSS JOIN sys.all_columns b
)
SELECT MyDate   = DATEADD(day,
1+ABS(CHECKSUM(NEWID()))%10000, CAST('1900-01-01' AS DATE))
    ,MyTime     = DATEADD(millisecond,
1+ABS(CHECKSUM(NEWID()))%86400000, CAST('00:00' AS TIME))
INTO #Temp
FROM Tally;
DECLARE @MyDateTime DATETIME;
SET STATISTICS TIME ON;
```

```
SELECT @MyDateTime=CAST(MyDate AS DATETIME) + CAST(MyTime AS
DATETIME)
FROM #Temp;
SET STATISTICS TIME OFF;
SET STATISTICS TIME ON;
SELECT @MyDateTime=DATEADD(millisecond
                    ,DATEDIFF(millisecond, 0, MyTime)
                    ,CAST(MyDate AS DATETIME))
FROM #Temp;
SET STATISTICS TIME OFF;
SET STATISTICS TIME ON;
SELECT @MyDateTime=DATEADD(day
                    ,DATEDIFF(day, 0, MyDate)
                    ,CAST(MyTime AS DATETIME))
FROM #Temp;
SET STATISTICS TIME OFF;
SET STATISTICS TIME ON;
SELECT @MyDateTime=CAST(
                    CAST(MyDate AS VARCHAR(10)) + ' ' +
CAST(MyTime AS VARCHAR(12))
                    AS DATETIME)
FROM #Temp;
SET STATISTICS TIME OFF;
GO
DROP TABLE #Temp;
```

Short and sweet! You gotta love that in a performance test harness.

In each case we've shunted the results to a local variable to avoid inevitable delays rendering our one million rows of test data to the SSMS Results pane, which might cause some variability in the timing results.

We chose the specific in-line Tally Table we're using, because it is probably one of the fastest to type in, due to its short length in terms of characters. Needless to say, any one million row tally table would suffice.

The SELECT INTO construct also makes it pretty easy to check that we're actually creating DATE and TIME datatypes into our #Temp table, simply by commenting out the INTO part of the query and rendering the results to the screen.

By now I've done enough talking, so I'll cut to the chase. That is, which of our four methods of combining a DATE and a TIME into a DATETIME ends up being the fastest?

```
(1000000 row(s) affected)
SQL Server Execution Times:
   CPU time = 171 ms,  elapsed time = 174 ms.
 SQL Server Execution Times:
   CPU time = 234 ms,  elapsed time = 233 ms.
 SQL Server Execution Times:
   CPU time = 234 ms,  elapsed time = 233 ms.
 SQL Server Execution Times:
   CPU time = 1123 ms,  elapsed time = 1113 ms.
```

There we have it! As it turns out, my first thought seems to be outperforming the other methods quite handily, especially that nasty one that uses the VARCHAR intermediate. That won't always be the case, so it is important that for things like this you get creative and see what your SQL-brain can come up with.

If anyone has a better method of doing this combination/conversion, please post it and your results using my test harness to the comments section.

We've demonstrated a couple of things here which I think are quite important:

- How easy it is to create a one million row test harness to compare different queries to see which is the fastest.
- Make it Work, Make it Fast and Make it Pretty – OK, so I forgot to do that last part! 😉 I'm saving that for the production query.

Hope to hear from you soon!

Follow me on Twitter: @DwainCSQL