**SQL Server v.Next (Denali) : CTP3 T-SQL Enhancements : FORMAT()**

<http://sqlblog.com/blogs/aaron_bertrand/archive/2011/08/09/sql-server-v-next-denali-ctp3-t-sql-enhancements-format.aspx>

As its name implies, the [FORMAT](http://technet.microsoft.com/en-us/library/hh213505%28SQL.110%29.aspx)() function was added to Denali in CTP3 to make formatting easier. For the purists, all formatting should be done at the presentation layer. For the rest of us, we know that our users and developers consuming our stored procedures expect us to be able to return data exactly how they want it presented. For years, we've been doing this with convoluted tricks using CAST, CONVERT or STR, but these often leave a lot to be desired. (And let's leave the obvious "let their browser handle it" arguments aside, as not all data consumers are using browsers, nor are their browser settings always tuned to the language or culture they want to use right now.)

I thought I would show a couple of examples where a simpler solution, based on formatting functionality we already have in .NET, could really streamline the formatting of strings - particularly dates and currency. For those familiar with C#, this new additional to T-SQL should be a friendly face indeed.

Standard Date Formatting

As a simple example, in order to give one user a weekday name in English and another user a weekday name in French, today we might solve this by applying a SET LANGUAGE command to each session that is not using the native language:

|  |
| --- |
| DECLARE  @d DATE = '2011-11-13',  @old\_lang VARCHAR(32) = @@LANGUAGE,  @new\_lang VARCHAR(32) = 'Français';  SET LANGUAGE @new\_lang;  SELECT DATENAME(WEEKDAY, @d);  SET LANGUAGE @old\_lang; |

Results:

|  |
| --- |
| -------- dimanche |

We may also solve it in a more complex way; for example, some people have a day of week table, a month name table, and then have a set of data for each language they support. Just dealing with weekdays:

|  |
| --- |
| CREATE TABLE dbo.Weekdays (  WeekdayNumber TINYINT,  [Language] VARCHAR(32),  Display VARCHAR(32) );  INSERT dbo.Weekdays(WeekdayNumber, [Language], Display) VALUES (1, 'English', 'Sunday'),  (2, 'English', 'Monday'),  -- ... etc etc  (1, 'Français', 'dimanche'),  (2, 'Français', 'lundi');  -- ... etc etc |

Then they do a lookup on every query (and often a join to make those nice pretty display names show up in a report). You laugh, but I have seen it:

|  |
| --- |
| DECLARE  @d DATE = '2011-11-13',  @new\_lang VARCHAR(32) = 'Français';  SELECT Display  FROM dbo.Weekdays  WHERE WeekdayNumber = DATEPART(WEEKDAY, @d)  AND [Language] = @new\_lang; |

Results:

|  |
| --- |
| --------  dimanche |

This can get old pretty quick. And while it works, it can be problematic because if you are supporting several languages, you are probably also supporting folks with different DATEFIRST settings (e.g. in some countries a week goes from Monday -> Sunday), so the code will also need to accommodate for that shift depending on where the WeekdayNumber variable comes from.

Now with FORMAT(), we can do all of this in a much easier way. There is a lot more flexibility here, without any hard-coding, where clauses or case statements to display the right element for the right person; all you need is the culture you're after:\*

|  |
| --- |
| DECLARE @d DATE = '2011-11-13';  -- quick sample to get 'dimanche'  -- like the convoluted methods above:  SELECT FORMAT(@d, N'dddd', N'fr-fr');  -- some other examples:  SELECT  FORMAT(@d, N'D'),  FORMAT(@d, N'D', N'en-gb'),  FORMAT(@d, N'D', N'de-de') UNION SELECT  FORMAT(@d, N'D', N'lt-lt'),  FORMAT(@d, N'D', N'fr-fr'),  FORMAT(@d, N'D', N'ja-jp'); |

Results:

|  |
| --- |
| --------  dimanche  ------------------------- ------------------------- -------------------------- Sunday, November 13, 2011 13 November 2011 Sonntag, 13. November 2011  2011 m. lapkričio 13 d. dimanche 13 novembre 2011 2011年11月13日 |

Notice that in no instance is the date (November 13th) misinterpreted as an invalid date - which often happens when using string literals and CONVERT(), depending on culture, language or regional settings.

The full list of cultures available are not listed in the documentation for [FORMAT](http://technet.microsoft.com/en-us/library/hh213505%28SQL.110%29.aspx)(), however they are listed in some of the other documentation. [PARSE](http://technet.microsoft.com/en-us/library/hh213316%28SQL.110%29.aspx)() lists the cultures that map between SQL Server and .NET, and [this document](http://msdn.microsoft.com/en-us/library/system.globalization.cultureinfo%28VS.71%29.aspx) lists all of the cultures available in .NET 1.0, but you can see a full list from your current operating system by running this PowerShell command:

|  |
| --- |
| [system.Globalization.CultureInfo]::GetCultures('AllCultures'); |

This results in 279 different cultures, and note that a lot of those listed are not supported natively by SQL Server (e.g. in collations or SET LANGUAGE). Were they all supported by FORMAT()? Only one way to find out! And just out of curiosity, I wanted to know what the "worst case scenario" would be - the longest weekday name, combined with the longest month name and culture format. So first I jammed all of the cultures above into a table, then I wrote a script to pull 7 days from each month against each culture, format the date as above, and take the max length:

|  |
| --- |
| CREATE TABLE dbo.Cultures (  LCID INT,   Name VARCHAR(32),   DisplayName VARCHAR(75) ); GO  -- populate this table from PowerShell, then...  ;WITH [months](m) AS  (  SELECT TOP (12) ROW\_NUMBER() OVER (ORDER BY [object\_id])  FROM sys.objects ), [days](d) AS (  SELECT TOP (7) m FROM [months] ORDER BY m ) SELECT   [date] = DATEFROMPARTS(2011, m.m, d.d),   c.DisplayName,   output = FORMAT(DATEFROMPARTS(2011, m.m, d.d), 'D', c.Name) FROM [months] AS m CROSS JOIN [days] AS d CROSS JOIN dbo.Cultures AS c; |

Partial results:

|  |
| --- |
| date DisplayName output ---------- -------------------- -------------------------- 2011-01-01 Arabic 26/محرم/1432 2011-01-01 Bulgarian 01 януари 2011 г. 2011-01-01 Catalan dissabte, 1 / gener / 2011 2011-01-01 Chinese (Simplified) 2011年1月1日 2011-01-01 Czech 1. ledna 2011 2011-01-01 Danish 1. januar 2011  ... 23,430 more rows |

To get the max length is only a slightly different query:

|  |
| --- |
| ;WITH [months](m) AS  (  SELECT TOP (12) ROW\_NUMBER() OVER (ORDER BY [object\_id])  FROM sys.objects ), [days](d) AS  (  SELECT TOP (7) m FROM [months] ORDER BY m ) SELECT MAX(LEN(FORMAT(DATEFROMPARTS(2011, m.m, d.d), 'D', c.Name))) FROM [months] AS m CROSS JOIN [days] AS d CROSS JOIN dbo.Cultures AS c; |

The answer? 47. I was kind of hoping for 42; that would have been profound.

Custom Date Formatting

In addition to using some of the standard format strings like 'D', you can also use custom formatting, which the C# folks should recognize:

|  |
| --- |
| DECLARE @d DATE = '2011-11-13';  SELECT  FORMAT(@d, N'yyyy-MM'),  FORMAT(@d, N'yyyy-MM-dd'),  FORMAT(@d, N'yyyy-MMM'),  FORMAT(@d, N'yyyy-MMM') UNION SELECT  FORMAT(@d, N'dddd, MMM dd, yyyy'),  FORMAT(@d, N'dddd, MMMM dd, yyyy'),  FORMAT(@d, N'dddd, MMMM dd, yyyy', N'fr-fr'),  FORMAT(@d, N'dddd, MMMM dd, yyyy', N'ja-jp'); |

Results:

|  |
| --- |
| ---------------------- ------------------------- --------------------- ------------------- 2011-08 2011-08-13 2011-Aug 2011-Aug Saturday, Aug 13, 2011 Saturday, August 13, 2011 samedi, août 13, 2011 土曜日, 8月 13, 2011 |

One important thing to note is that the output here is NVARCHAR(4000) - it's unfortunate that the output couldn't infer from the length of the input string that all those characters weren't necessary, but this is tough because /by definition/ the output length will vary greatly depending on the culture. So, you may end up wanting to use a surrounding CONVERT() after all, especially if you're trying to read or copy output from results to text in Management Studio.

Another important thing to note is that composite formatting (which works like RAISERROR's printf-style substitution) is not supported. So you can't fill up a string with tokens and pass in a loosely-defined number of arguments like you might be used to doing in C#.

And finally, this methodology doesn't quite help you if you want to display yyyy-mm-dd for US and yyyy-dd-mm for France. I haven't explored FORMAT() enough just yet to see if there's a way to present this output correctly, changing nothing else except the culture parameter.

Currency Formatting

Now you have even less reason to use MONEY/SMALLMONEY. Remember all that work you had to do in order to store currency values that you could perform reliable calculations against, but also present them properly (with thousand separators, the correct currency symbol, decimals vs. commas, and all that jazz)? I see this kind of stuff all the time, when dealing with a single currency:

|  |
| --- |
| DECLARE   @m DECIMAL(12,2) = 1271627.13;  SELECT '$' + CONVERT(VARCHAR(32), CONVERT(MONEY, @m), 1); |

Results:

|  |
| --- |
| ------------- $1,271,627.13 |

What about when you are dealing with multiple cultures? Never mind conditional formatting depending on the the culture. If the amount stored is Yen, you need ¥ at the beginning of the value; if it is a Euro, you need € at the end. On top of that, some cultures swap the decimals and commas as thousand separators and decimal points; and some want spaces between digit groupings instead of printable separators. So let's just say we want to deal with Japan, USA, UK, Germany and French. That's five different outputs. How would we do this? Today, we might create a table to store all of these rules and then build some strings based on the values and the culture chosen.

|  |
| --- |
| CREATE TABLE dbo.Currencies (  Culture NCHAR(5) PRIMARY KEY,  LeadCharacter NVARCHAR(2),  TrailCharacter NVARCHAR(2),  Separator CHAR(1),  [Decimal] CHAR(1) );  INSERT dbo.Currencies VALUES(N'ja-jp', N'¥', N'', ',', '.'),  (N'en-us', N'$', N'', ',', '.'),  (N'en-gb', N'£', N'', ',', '.'),  (N'de-de', N'', N' €', '.', ','),  (N'fr-fr', N'', N' €', ' ', ',');  -- assuming SET LANGUAGE ENGLISH:  DECLARE  @m DECIMAL(12,2) = 1271627.13;  SELECT Culture, Display = LeadCharacter  + REPLACE(REPLACE(REPLACE(REPLACE(  (CONVERT(VARCHAR(32), CONVERT(MONEY, @m), 1)),  ',', '{'), '.', '}'), '}', [Decimal]), '{', Separator)  + TrailCharacter  FROM dbo.Currencies; |

(We have to do a double-nested replace here because we are often going to find cases where we replace decimals with commas and then we can't tell which commas used to be decimals, and which used to be commas.)

Results:

|  |
| --- |
| Culture Display ------- -------------- de-de 1.271.627,13 € en-gb £1,271,627.13 en-us $1,271,627.13 fr-fr 1 271 627,13 € ja-jp ¥1,271,627.13 |

Now, we've written all that messy code, and still we have an issue - the data is not correct for Yen because there are no decimals. So our code has displayed decimal places, but these would need to be trimmed off for display (ignore for a moment that this is actually a problem with the data).  
  
Now look how easy this can be with the new FORMAT function:

|  |
| --- |
| DECLARE  @m DECIMAL(12,2) = 1271627.13;  SELECT Culture, Display = FORMAT(@m, 'C', Culture)  FROM dbo.Currencies; |

Results:

|  |
| --- |
| Culture Display  ------- --------------  de-de 1.271.627,13 € en-gb £1,271,627.13 en-us $1,271,627.13 fr-fr 1 271 627,13 € ja-jp ¥1,271,627 |

The results are quite similar to the previous example, but my fingers are much happier, and I've had to do a lot less thinking - most importantly, I didn't have to store or hard-code the separators, currency symbols, or use a conditional to decide where the currency symbol belongs.

Summary

FORMAT() is clearly a much more scalable approach to formatting strings, and is one of the T-SQL enhancements in Denali that I'm most looking forward to. If you've dealt with any of these issues, I hope this has given you a taste of how much easier your life is going to be...

Don't forget to clean up the tables we created:

|  |
| --- |
| DROP TABLE dbo.Cultures, dbo.Weekdays, dbo.Currencies; |