Working with dates and times in SQL Server

SQL Server gives you plenty of power for working with dates, times and datetimes.

Tamar E. Granor, Ph.D.

One of the things that makes VFP easy to work with is the inclusion of date math and a robust set of functions for taking dates and datetimes apart and putting them back together. SQL Server also supports date math and has its own set of functions for manipulating dates and datetimes. In this article, I'll look at the date and time types SQL Server supports and the language elements for working with them. In my next article, I'll show how to solve common date and time related problems in SQL Server.

Databases typically have lots of dates and times in them. They represent birthdates, dates people were hired, invoice dates, manufacture dates, appointment times, arrival times, departure times, and so much more. It's not unusual to need to do calculations based on those dates, such as computing a renewal date so many days after a given date or calculating hours worked based on arrival and departure time. Taking dates apart to, for example, extract just the year, or building dates out of a day, month and year are also common tasks.

As VFP developers, we're used to being able to use FoxPro's built-in date math and functions like GoMonth, YEAR(), DATE() and so on to perform such tasks. While the way you perform some of them is different in SQL Server, it, too, has a robust date and datetime story.

Date Types

SQL Server supports multiple date and time related data types; they're shown in Table 1. The story was considerably improved in SQL Server 2008, with the addition of separate Date and Time types, as well as a more precise DateTime type and a type that tracks time zone offset along with the date and time, so that it's independent of location.

Table 1. SQL Server supports date, datetime and time data types.

Type	Format	Notes
Datetime	YYYY-MM-DD hh:mm:ss[.nnn]	Standard date and time with precision up to milliseconds
Datetime2	YYYY-MM-DD hh:mm:ss[.nnnnnnn]	Date and time with precision to millionths of a second, added in SQL Server 2008
Smalldatetime	YYYY-MM-DD hh:mm:ss	Date and time with precision to seconds
Date	YYYY-MM-DD	Date only, added in SQL Server 2008
Time	hh:mm:ss[.nnnnnnn]	Time only with precision to millionths of a second, added in SQL Server 2008
Datetimeoffset	YYYY-MM-DD hh:mm:ss[.nnnnnnn] [+|-] hh:mm	Date and time with precision to millionths of a second and information about timezone, added in SQL Server 2008

While I normally like to present examples using the AdventureWorks example database, it includes only three of the six data types, so at least the examples in this article will use the temporary table created by the code in Listing 1 (included in this month's downloads as CreateDTTest.SQL); obviously, if you run this code yourself, you'll have different data in the first record. Note that, in this case, SQL Server implicitly converts the value being inserted into the right data type, so it doesn't matter that the value inserted into all fields of the first record is datetimeoffset and the values inserted into the second record are strings.

Listing 1. This code creates and populates a temporary table that includes all six date and time types.

```
CREATE TABLE #dttesting (
  tDateTime datetime,
  tDateTime2 datetime2,
  tSmallDateTime smalldatetime,
  dDate date,
  tTime time,
  tDateTimeOffset datetimeoffset);

DECLARE @RightNow DateTimeOffset
  = SYSDATETIMEOFFSET();

INSERT INTO #dttesting VALUES
  (@RightNow, @RightNow, @RightNow,
   @RightNow, @RightNow, @RightNow),
  ('1958-09-28', '1958-09-28', '1958-09-28',
   '1958-09-28', '09:37:52', '1958-09-28');
```

In the rest of this article, I'll use "date/time" to mean "date, time or datetime."

Getting the current date and time

VFP offers the DATE() and DATETIME() functions to return the current date and datetime, respectively. SQL Server has six functions that provide the current datetime; they're shown in Table 2.

Table 2. SQL Server offers a variety of ways to get the current date and time.

Function	Notes
CURRENT_TIMESTAMP	ANSI SQL standard way to retrieve the current date and time. Returns a datetime value.
GETDATE()	Retrieves the current date and time. Returns a datetime value.
GETUTCDATE()	Retrieves the current date and time in UTC (coordinated universal time). Returns a datetime value.
SYSDATETIME()	Retrieves the current date and time. Returns a datetime2 value.
SYSUTCDATETIME()	Retrieves the current date and time in UTC (coordinated universal time). Returns a datetime2 value.
SYSDATETIMEOFFSET	Retrieves the current date and time with time zone offset. Returns a datetimeoffset value.

GetDate() is the SQL Server implementation of the ANSI Standard CURRENT_TIMESTAMP. You can use them interchangeably.

There are no functions to return just the current date or just the current time. To get those, you can use one of the current datetime functions and then apply CAST() or CONVERT() to convert to the desired type. Listing 2 shows two ways each of getting the current date and the current time.

Listing 2. There are no built-in functions to provide the current date or the current time individually. Instead, convert the current datetime.

```
SELECT CAST(GetDate() as Date),
       CONVERT(date, GETDATE()),
       CAST(GetDate() AS time),
       CONVERT(time, GetDate())
```

Date Math

For the older datetime types (datetime and smalldatetime), SQL Server supports the same kind of date math as VFP. You can add or subtract numbers to or from those to get new datetimes. So, for example, the query in Listing 3 produces the results shown in Figure 1.

Listing 3. You can do date math with datetime and smalldatetime.

```
SELECT tDateTime + 1 AS tNextDay,
       tSmallDateTime - 1 AS tPriorDay
  FROM #dttesting
```

tNextDay	tPriorDay
2016-11-29 15:04:57.873	2016-11-27 15:05:00
1958-09-29 00:00:00.000	1958-09-27 00:00:00

Figure 1. You can add to and subtract from datetime and smalldatetime values.

Note that even though the values are datetimes, adding an integer changes the value by days, not seconds (as it would in VFP). You can add and subtract fractional values and the new values differ by the appropriate fraction of a day, so adding .5 to a datetime or smalldatetime value gives you the datetime 12 hours later.

The newer datetime types don't support direct date math like this. When you attempt it, you get an error like "Operand type clash: datetime2 is incompatible with numeric."

The same restrictions apply to subtracting one date or time value from another. The two older types allow such subtraction, though the result is of the same type and has to be converted with CAST() to tell you the number of days between the two dates. For example, the query in Listing 4 produces the results in Figure 2.

Listing 4. You can subtract a datetime from another, but the result is a datetime. Use CAST() to turn it into a number.

```
SELECT GetDate() - tDateTime AS tDiff,
       CAST(GetDate() - tDateTime AS Int)
         AS nDays
  FROM #dttesting
```

tDiff	nDays
1900-01-01 00:02:52.107	0
1958-03-04 15:07:49.980	21247

Figure 2. With CAST(), you can find out how many days between two datetimes or smalldatetimes.

Calculating with dates

If you can't do date math with the newer date and time types, how do you calculate the difference between two dates, or the date 30 days from today? With a pair of functions, DateDiff() and DateAdd(). DateAdd() is similar to VFP's GoMonth() function, but handles a much broader range of calculations.

Computing date and time differences

DateDiff() computes the difference between two dates, times, or datetimes. You specify what units (called *dateparts*) you want the difference in. The syntax for DateDiff() is shown in Listing 5.

Listing 5. The DateDiff() function calculates the difference between two dates, times or datetimes, and returns the value as a number of the specified datepart.

```
Result = DateDiff( DatePart, Start, End)
```

So to find the number of days between two datetimes, you use code like Listing 6. The order of the dates here is the reverse of that used when subtracting. That is, the function subtracts the first date provided (Start) from the second (End). If End is earlier than Start, the result is negative.

Listing 6. To find the number of days between two dates, use DateDiff() and pass day as the first parameter.

```
SELECT DateDiff(day, tDateTime2, GetDate())
         AS nDays
  FROM #dttesting
```

What's really powerful about DateDiff(), though, is that it can handle a wide range of dateparts, not just days. Table 3 shows the options; though the document shows lower case for all of them, my tests indicate that upper case and mixed case work as well. Note also that dateparts are used by multiple functions; this table addresses all uses, including some dateparts that aren't supported by DateDiff().

Table 3. SQL Server supports a wide variety of dateparts.

Datepart	Alternative notations	Notes
year	yy, yyyy	Year. For DateDiff(), based on only the year portion of the dates, not the number of days between the specified dates.
quarter	qq, q	Quarter. For DateDiff(), based on strict definition of quarters, not the number of three month periods between the dates.
month	mm, m	Month of the year. For DateDiff() and DateAdd(), number of months. For DateDiff(), based on only the year and month portion of the dates, not the number of days.
dayofyear	dy, y	Day from the first of the year. For DateDiff() and DateAdd(), number of days, same as day.
day	dd, d	Day of the month. For DateDiff() and DateAdd(), number of days, same as dayofyear.
week	wk, ww	Week of the year, based on the SET DATEFIRST setting. For DateDiff() and DateAdd(), number of weeks.
weekday	dw, w	Day of the week, based on the SET DATEFIRST setting. For DateDiff() and DateAdd(), number of days, same as day.
hour	hh	Hours.
minute	mi, n	Minutes.
second	ss, s	Seconds.
millisecond	ms	Thousands of a second.
microsecond	mcs	Millionths of a second.
nanosecond	ns	Billionths of a second.
TZoffset	tz	Timezone offset (in minutes), applies only to datetime2 and datetimeoffset types. Not supported for DateDiff() and DateAdd().
ISO_WEEK	Isowk, isoww	ISO standard 8601 week. Not supported for DateDiff() and DateAdd().

DateDiff() returns an Int value. That means that the maximum difference it supports between the start and end values varies based on which datepart you specify. Given other limitations on the accuracy of datetime values, you're unlikely to run into this issue until you get down to seconds or less. DateDiff() can handle differences of up to 68 years in seconds; for milliseconds, the limit is less than 25 days; for nanoseconds, it's a little more than 2 seconds. (That makes sense when you realize that the range of Int is from -2,147,483,648 to +2,147,483,647.)

The table hints at an important point about how DateDiff() works. The calculation goes down only as far as the specified datepart. So, for example, if you specify a start date of 12/31/2016 and an end date of 1/1/2017, with a datepart of year, DateDiff() gives you 1, but a start date of 1/1/2017 and an end date of 1/2/2017 gives you 0. That may or may not be surprising, but consider the example in Listing 7; the result is shown in Figure 3. Here, we declare a start time and two end times. Each of the end times differs from the start time by 90 minutes, but when we check the difference in hours, we get 2 in the first case and -1 in the second. (DateDiff() returns a negative value when the End time is earlier than the Start time.) That's because DateDiff() doesn't subtract the first date from the second (in, say, nanoseconds) and then convert to hours. Instead, it looks only at the parts larger than the one you specify and the one you specify. Here, the dates are the same, so they contribute 0 and then, then hours portion of the start time is subtracted from the hours portion of the end time.

Listing 7. DateDiff() drills down only to the datepart you specify.

```
DECLARE @start datetime2 =
  '2016-10-27 11:45:00';
DECLARE @end1 datetime2 =
  '2016-10-27 13:15:00';
DECLARE @end2 datetime2 =
  '2016-10-27 10:15:00';

SELECT DATEDIFF(hh, @start, @end1),
       DATEDIFF(hh, @start, @end2);
```

(No column na...	(No column na...
2	-1

Figure 3. The difference in hours between datetimes that are the same distance apart can be different because of the way DateDiff() works.

Finding new dates and times from old

The DateAdd() function lets you find the date/time so many days, weeks, months, hours, or whatever before or after the date/time you already have. The syntax is shown in **Listing 8**. The function uses the same list of dateparts as DateDiff().

Listing 8. Use DateAdd() to add or subtract dateparts from a date/time.

```
Result = DATEADD(DatePart, Number, Start)
```

Since DateAdd() is like a supercharged version of VFP's GoMonth() function, let's start with an example that shows the similarity. To find the date one month after a specified date, you use code like Listing 9. Figure 4 shows results.

Listing 9. DateAdd() is like VFP's GoMonth() function, except that it can handle much more than months.

```
SELECT tDateTime2, DATEADD(mm, 1, tDateTime2)
    FROM #dtTesting
```

tDateTime2	(No column name)
2016-11-28 15:04:57.8723961	2016-12-28 15:04:57.8723961
1958-09-28 00:00:00.0000000	1958-10-28 00:00:00.0000000

Figure 4. When you use month or mm as the datepart, DateAdd() works just like VFP's GoMonth().

Of course, what makes DateAdd() powerful is its ability to calculate based on any datepart. So, for example, to find the date six quarters before a specified date, you use code like Listing 10.

Listing 10. To find an earlier date, pass a negative number as the second parameter to DateAdd().

```
SELECT tDateTime2, DATEADD(q, -6, tDateTime2)
    FROM #dtTesting
```

DateAdd() returns a value of the same type as Start, unless you pass a string in for the start date, in which case, the function returns a datetime.

DateAdd() is also useful for creating date/times; I'll cover that in my next article.

There's one very specific function that answers a common question. The EOMonth() function returns the last day of the month of the date you pass it. It also accepts an optional second parameter, which lets you specify a number of months, so that you can find the last day of the month so many months before or after the date you specify. The query in Listing 11 finds the last day of the month for the tDateTime2 field, and then the last day of the month three months before and three months after the specified date. Figure 5 shows the result.

Listing 11. The EOMonth() function computes the last day of the specified date, and even lets you look forward and backward.

```
SELECT EOMONTH(tDateTime2),
       EOMONTH(tDateTime2, -3),
       EOMONTH(tDateTime2, 3)
    FROM #dttesting
```

(No column na...	(No column na...	(No column na...
2016-11-30	2016-08-31	2017-02-28
1958-09-30	1958-06-30	1958-12-31

Figure 5. Use EOMonth() to find the last day of the month.

Note that EOMonth() returns a Date, no matter which date/time type you pass in.

Taking Date/Times apart

VFP has a set of functions for parsing dates and datetimes into their components. For example, YEAR() returns the year portion of a date or datetime, while MINUTE() returns the minute portion of a datetime.

SQL Server also has DAY(), MONTH() and YEAR() functions that let you extract the specified portion of a date/time. However, it doesn't include functions to extract the hour, minute or second.

Instead, there's are two generic functions, DatePart() and DateName, that let you extract any date part from a date/time value. DatePart() returns the specified part as an integer, while DateName() returns it as nVarChar.

The syntax for the two functions is shown in Listing 12.

Listing 12. The DatePart() function lets you extract any component of a date/time.

```
iResult = DATEPART( datepart, date )
cResult = DATENAME( datepart, date )
```

For example, to get the hour, minute and second components as integers, you can use code like Listing 13. The results are shown in Figure 6.

Listing 13. Use DatePart() to take dates and times apart.

```
SELECT DATEPART(hour, tDateTime2) AS iHour,
       DATEPART(minute, tDateTime2)
         AS iMinute,
       DATEPART(second, tDateTime2) AS iSecond
    FROM #dttesting
```

iHour	iMinute	iSecond
15	4	57
0	0	0

Figure 6. SQL Server's DatePart() function takes dates and times apart.

The only dateparts where the two functions return different values (that is, values that differ other than in type) are month and weekday. Listing 14 calls each of the functions for those two dateparts; Figure 7 shows the result.

Listing 14. DATENAME() returns the name of the specified part of the date/time. For months and days, it gives it in the system language, rather than as a number.

```
SELECT DATEPART(MONTH, tDateTime2) AS iMonth,
       DATENAME(MONTH, tDateTime2) AS cMonth,
       DATEPART(WEEKDAY, tDateTime2) AS iDay,
       DATENAME(WEEKDAY, tDateTime2) AS cDay
    FROM #dttesting
```

iMonth	cMonth	iDay	cDay
11	November	2	Monday
9	September	1	Sunday

Figure 7. For months and days of the week, DateName() is equivalent to the VFP CMonth() and CDOW() functions.

The week and weekday dateparts depend on SET DATEFIRST, which specifies the first day of the week (like the VFP SET FDOW command). SET DATEFIRST accepts a value from 1 (Monday) to 7 (Sunday, which is the default for US English). Listing 15 shows how to set Wednesday as the first day of the week. The @@DATEFIRST function returns the current setting.

Listing 15. SET DATEFIRST specifies the first day of the week, which determines how weeks are counted when using the week and weekday dateparts.

```
SET DATEFIRST 3
```

The ISO_WEEK datepart is independent of the DATEFIRST setting. Each week runs from Monday to Sunday and is associated with the year in which the Thursday falls. So, for example, DATEPART(iso_week, '2016-1-2') returns 53, because January 2, 2016 fell on a Saturday and the Thursday that week was December 31, 2015, so it's counted as the last week of 2015. But DATEPART(iso_week,'2017-1-1') returns 1, because it's a Sunday and the Thursday of that week is January 5, 2017.

Assembling dates

One of the minor, but really useful, changes in VFP, was the addition of parameters to the DATE() and DATETIME() functions in VFP 6. You can pass a year, month and day to DATE(), or those three plus hour, minute and second values to DATETIME() and get back a date or datetime value. The functions provided an elegant date format independent way to specify date/times.

SQL Server has the same capability, using a set of functions that vary based on the type you want to return. There's one function for each of the date/time types; Listing 16 shows the syntax for these functions.

Listing 16. SQL Server has a set of functions that can turn numbers into date/times.

```
tDateTime = DATETIMEFROMPARTS(
  iYear, iMonth, iDay,
  iHour, iMinute, iSeconds, iMilliseconds )

tDateTime2 = DATETIME2FROMPARTS(
  iYear, iMonth, iDay,
  iHour, iMinute, iSeconds,
  iFractions, iPrecision )

tDateTimeOffset = DATETIMEOFFSETFROMPARTS(
  iYear, iMonth, iDay,
  iHour, iMinute, iSeconds, iFractions,
  iHour_offset, iMinute_offset, iPrecision )

tSmallDateTime = SMALLDATETIMEFROMPARTS(
  iYear, iMonth, iDay, ihour, iMinute )

dDate = DATEFROMPARTS( iYear, iMonth, iDay )

tTime = TIMEFROMPARTS(
  iHour, iMinute, iSeconds,
  iFractions, iPrecision )
```

To start with a simple example, Listing 17 shows how to set a variable to a specified date.

Listing 17. The various xxxFromParts() functions let you create date/times from their components.

```
DECLARE @BirthDate Date;
SET @BirthDate = DateFromParts(1958, 9, 28);
```

The iFractions and iPrecision parameters of DateTime2FromParts(), TimeFromParts() and DateTimeOffsetFromParts() work together. The value you pass for iPrecision determines how iFractions is interpreted. Specifically, in assembling the result, the fractional portion of the time is set to have iPrecision digits. The value you specify for iFractions become the right-most of those digits. For example, the query in Listing 18 produces the result in Figure 8. The final parameter, 4, indicates there should be 4 decimal places; the iFractions parameter of 152 says those are the last three digits. Thus, the decimal part of the time is .0152.

Listing 18. The fractions and precision parameters of the xxxFromParts() functions interact to determine the decimal part of the time.

```
SELECT DATETIME2FROMPARTS(
  1958, 9, 28,
  9, 37, 14,
  152, 4)
```

(No column name)
1958-09-28 09:37:14.0152

Figure 8. This datetime was specified with152 for iFraction and 4 for iPrecision, resulting in .0152 for the decimal portion.

Another way to think of this is that you divide the iFraction parameter by 1 followed by iPrecision zeroes to get the fractional part, so in the previous example, you divide 152 by 10000, resulting in .0152.

DateTimeOffsetFromParts() has two parameters to specify the offset; you can indicate both a number of hours and a number of minutes. For example, to specify a datetime with an offset for US Eastern time, you pass -5 for offset hours and 0 for offset minutes, as in the first example in Listing 19. The second example shows how to specify the offset for Newfoundland, which is 3.5 hours behind UTC.

Listing 19. To build a DateTimeOffset value, you specify the offset in hours and minutes.

```
SELECT DATETIMEOFFSETFROMPARTS(
  1958, 9, 28,
  9, 37, 14, 152,
  -5, 0, 4)

SELECT DATETIMEOFFSETFROMPARTS(
  1958, 9, 28,
  9, 37, 14, 152,
  -3, -30, 4)
```

Putting date/times to work

While there are a few more date/time-related functions, the ones we've already covered let us handle most of the questions that come up around date/times. In my next article, I'll show how to use these functions to answer such questions.

Author Profile

Tamar E. Granor, Ph.D. is the owner of Tomorrow's Solutions, LLC. She has developed and enhanced numerous Visual FoxPro applications for businesses and other organizations. Tamar is author or co-author of a dozen books including the award winning Hacker's Guide to Visual FoxPro, Microsoft Office Automation with Visual FoxPro *and* Taming Visual FoxPro's SQL. *Her latest collaboration is* VFPX: Open Source Treasure for the VFP Developer, *available at www.foxrockx.com. Her other books are available from Hentzenwerke Publishing (www.hentzenwerke.com). Tamar was a Microsoft Support Most Valuable Professional from the program's inception in 1993 until 2011. She is one of the organizers of the annual Southwest Fox conference. In 2007, Tamar received the Visual FoxPro Community Lifetime Achievement Award. You can reach her at tamar@thegranors.com or through www.tomorrowssolutionsllc.com.*