Solve date problems

SQL Server's date and time functions make it easy to solve various date-related problems, like finding the start and end dates of a period, finding anniversaries in a specified period, and identifying weekdays in a period.

Tamar E. Granor, Ph.D.

In my last article, I explored the various date and time types in SQL Server and the collection of functions and operations you can use with them. In this article, I'll look at some common date problems and show how to solve them.

First, a quick review. SQL Server supports six data types related to dates and times. Four of them are variations on a combined date and time construct. Three of those (SmallDateTime, DateTime and DateTime2) vary only in precision. The fourth, DateTimeOffset, is the same as DateTime2, but also includes a timezone offset component. There are also separate Date and Time types. For this article, I'll refer to these in the aggregate as date/time.

There are functions for retrieving the current date/time, computing the difference between date/time values, calculating new date/time values from existing ones, taking date/time values apart, and constructing date/time values from their components. I'll use a number of them in solving the problems in this article.

The examples use the Adventureworks 2014 sample database.

Find the first and last days of the period

Given a date, it's not unusual to want to find the first and last day of the containing week, month, quarter, or year. For some of these periods, it's easy to build the start and end dates by pulling the given date apart and creating a new one. For other periods, it's trickier, but the DateAdd() and DateDiff() functions provide a generic approach.

Let's start with the easy way, which works for months and years. First, as noted in my previous article, the function EOMonth()gives you the last day of the month for a specified date. To find the first day of the month, you can extract the month and year from the specified date and combine them with a day of 1. A similar approach works for the first and last day of the year. Listing 1 shows the code; it's included in this month's downloads as FirstLastSimple.SQL. Figure 1 shows the results. If you want them as one of the other date/time types, use the appropriate DateTime*x*FromParts() function for the first of the month and the first and last day of the year, and wrap EOMonth() with Cast() or Convert() for the last day of the month.

Listing 1. There are simple ways to find the first and last day of the month or the year containing a specified date.

```
DECLARE @Date Date;

SET @Date = '1/4/2017';

SELECT DATEFROMPARTS(DATEPART(Year, @Date),
         DATEPART(Month, @Date), 1) AS BoM,
       EOMONTH(@Date) AS EoM,
       DATEFROMPARTS(DATEPART(Year, @Date),
         1, 1) AS BoY,
       DATEFROMPARTS(DATEPART(Year, @Date),
         12, 31) AS EoY;
```

BoM	EoM	BoY	EoY
2017-01-01	2017-01-31	2017-01-01	2017-12-31

Figure 1. Finding the first and last day of the month or the year is straightforward.

But this approach doesn't work for the first and last days of the week or the quarter, because we don't know what the appropriate days or months are. Instead, we need to calculate those, and we can do months and years the same way, if we wish.

The DateAdd() function lets us move a date/time forward or backward by a number of years, months, days, hours, etc. The DateDiff() function allows us to compute the difference between two date/time values in whichever units we want (though see my last article to understand exactly how that works). We can combine them to figure out the first or last day of the period containing a particular date, as in Listing 2, where we find the first and last day of the week. The code is included in this month's downloads as FirstLastOfWeek.SQL.

Listing 2. You can combine DateAdd() and DateDiff() to compute dates such as the first and last of the month.

```
DECLARE @Date Date;

SET @Date = '1/4/2017';

SELECT DATEADD(Week,
```

```
      DATEDIFF(Week, 0, @Date), 0) AS BoW,
    DATEADD(DAY, -1, DATEADD(Week,
      DATEDIFF(Week, 0, @Date) + 1, 0))
      AS EoW;
```

Let's work out the first of week code first because it's simpler. Working from the inside out, DateDiff(Week, 0, @Date) computes the number of weeks from SQL Server's zero date (January 1, 1900) to the specified date. Then, the surrounding DateAdd() adds that many weeks to the zero date, resulting in the first day of the week containing @Date.

The end of week calculation is similar, but a little more complex. It computes the first day of the week following the one containing the specified date by doing the same DateDiff() calculation, adding 1 and then applying DateAdd(). Then, it uses DateAdd() a second time to subtract one day from that date. (For the last day of the week, you could also compute the first day of the week and then add 6.)

You can find the first and last day of the quarter containing a particular date by changing every occurrence of "Week" in that code to "Quarter," as in Listing 3 (included in this month's downloads as FirstLastofQuarter.SQL).

Listing 3. To find the first and last day of the year, just change every occurrence of "Week" in the prior query to "Quarter".

```
SELECT DATEADD(Quarter,
         DATEDIFF(Quarter, 0, @Date), 0)
         AS BoQ,
       DATEADD(Day, -1, DATEADD(Quarter,
         DATEDIFF(Quarter, 0, @Date) + 1, 0))
         AS EoQ;
```

You can do use the same formulation for months and years, too. Just specify the appropriate datepart.

Birthdays and anniversaries

We often need to find those having an anniversary during a given period. It might be people with a birthday this week, those celebrating employment anniversaries next month, and so on. The key point here is that we're looking for dates where the month and day match, but the year is irrelevant.

For simplicity, we'll use employee birthdays to demonstrate the solutions here, but the same ideas apply to any kind of anniversary dates. Let's start with finding every employee who has a birthday on a given date. There are actually two good solutions here. The first is to use DatePart() to extract the month and day and compare them, as in Listing 4; this query is included as SameBirthDateDatePart.SQL in this month's downloads. Figure 2 shows those employees whose birthday is January 4.

Listing 4. When you want to match day and month exactly, you can just extract them with DatePart().

```
DECLARE @Date Date;

SET @Date = '1/4/2017';

SELECT FirstName, LastName,
       Employee.BusinessEntityID, Birthdate
   FROM [HumanResources].[Employee]
     JOIN [Person].[Person]
       ON Employee.BusinessEntityID =
          Person.BusinessEntityID
   WHERE DATEPART(Day, BirthDate) =
         DATEPART(Day, @Date)
     AND DATEPART(Month, BirthDate) =
         DATEPART(Month, @Date)
```

FirstName	LastName	BusinessEntityID	BirthDate
Michael	Rothkugel	133	1991-01-04
Barbara	Moreland	245	1976-01-04
Erin	Hagens	258	1971-01-04

Figure 2. It's easy to find out who shares a birthday.

The alternative approach to this problem uses the technique from earlier in this article to convert the birthdate to the birthday this year and compares that to the specified date. Listing 5, included in this month's downloads as SameBirthDateDateAdd.SQL, produces the same results as Listing 4.

Listing 5. An alternative way to find people with a given birthday uses DateAdd() and DateDiff().

```
DECLARE @Date Date;

SET @Date = '1/4/2017';

SELECT FirstName, LastName,
       Employee.BusinessEntityID, BirthDate
   FROM [HumanResources].[Employee]
     JOIN [Person].[Person]
       ON Employee.BusinessEntityID =
          Person.BusinessEntityID
   WHERE DATEADD(Year, DATEDIFF(Year,
     BirthDate, @Date), BirthDate) = @Date
```

We can extend this process to, for example, find everyone with a birthday this week. The tricky part of finding everyone with a birthday in a given week is that the week can cross the boundaries of a year.

Let's start with a version of the query that doesn't account for crossing the end of the year. Listing 6 (included in this month's downloads as SameWeekNoYearEnd.SQL) shows how to find everyone with a birthday in the week beginning on a specified date, as long as the week ends in the same year it starts in. It uses the same expression as the previous example to move the birthday into the same year as the specified date. Figure 3 shows the results for the date specified in the example, January 4, 2017.

Listing 6. If you don't have to worry about crossing the end of a year, finding all birthdays in the week beginning on a given date is fairly simple.

```
DECLARE @Date Date;

SET @Date = '1/4/2017';

SELECT FirstName, LastName,
       Employee.BusinessEntityID, BirthDate
```

```
FROM [HumanResources].[Employee]
  JOIN [Person].[Person]
    ON Employee.BusinessEntityID =
       Person.BusinessEntityID
WHERE DATEADD(YEAR, DATEDIFF(Year,
        BirthDate, @Date), BirthDate)
  BETWEEN @Date AND DATEADD(Day, 6, @Date)
```

FirstName	LastName	BusinessEntityID	BirthDate
James	Hamilton	25	1983-01-07
Brandon	Heidepriem	35	1977-01-10
Thomas	Michaels	45	1986-01-10
Michael	Rothkugel	133	1991-01-04
Benjamin	Martin	174	1986-01-05
Reed	Koch	175	1989-01-08
Laura	Norman	234	1976-01-06
Barbara	Moreland	245	1976-01-04
Erin	Hagens	258	1971-01-04
Dan	Wilson	271	1976-01-06
Tete	Mensa-Annan	284	1978-01-05

Figure 3. Moving the date into the current year lets you find all birthdays in a specified week.

But what happens if we specify a date late in December? We know from the previous example that there are people with birthdays on January 4, but if we set @Date to '12/29/2017', we get the results shown in Figure 4.

FirstName	LastName	BusinessEntityID	BirthDate
David	Ortiz	70	1984-12-29

Figure 4. At the end of the year, the simple test in Listing 6 doesn't work.

To see what's causing the problem, we can add the translated birthday to the field list and look for only those records with January 4 birthdays, as in the query in Listing 7; the results are shown in Figure 5.

Listing 7. By including the computed birthday this year in the results, we can see why this version doesn't work across the end of a year.

```
SELECT FirstName, LastName,
       Employee.BusinessEntityID, BirthDate,
       DATEADD(YEAR, DATEDIFF(Year,
         BirthDate, @Date), BirthDate)
  FROM [HumanResources].[Employee]
    JOIN [Person].[Person]
      ON Employee.BusinessEntityID =
         Person.BusinessEntityID
  WHERE Month(BirthDate) = 1
    AND Day(BirthDate) = 4
```

FirstName	LastName	BusinessEntit...	BirthDate	TransBDay
Michael	Rothkugel	133	1991-01-04	2017-01-04
Barbara	Moreland	245	1976-01-04	2017-01-04
Erin	Hagens	258	1971-01-04	2017-01-04

Figure 5. Looking at the translated birthday helps to show why the expression using DateAdd() and DateDiff() isn't sufficient.

Using the DateAdd(…DateDiff()…) expression moves the birthdate into 2017, the same year as the specified date. But if we're looking for birthdays in the week beginning December 29, we need the January birthdays to be translated into 2018. Doing that requires a more complex expression using CASE; it's shown in Listing 8 (included in this month's downloads as SameWeek.SQL). The CASE expression divides into two cases, based on whether the birthday comes earlier or later in the year than the specified date. If it comes later in the year, we just translate to the same year. If it comes earlier in the year, though, we translate to the next year.

Listing 8. To work across year-end, we need to figure out whether the birthday is earlier or later in the year than the specified date.

```
DECLARE @Date Date;

SET @Date = '1/4/2017';

SELECT FirstName, LastName,
       Employee.BusinessEntityID, BirthDate
  FROM [HumanResources].[Employee]
    JOIN [Person].[Person]
      ON Employee.BusinessEntityID =
         Person.BusinessEntityID
  WHERE CASE
    WHEN DATEPART(Month, Birthdate)
           < DATEPART(Month, @Date)
      OR (DATEPART(Month, Birthdate) =
          DATEPART(Month, @Date) AND
          DATEPART(Day, Birthdate) <
          DATEPART(Day, @Date))
    THEN DATEADD(Year, DATEDIFF(Year,
           BirthDate, @Date) + 1, BirthDate)
    ELSE DATEADD(Year, DATEDIFF(Year,
           BirthDate, @Date), BirthDate) END
    BETWEEN @Date AND DATEADD(Day, 6, @Date)
```

If we run the query as shown, with a specified date of January 4, 2017, we get the same results as in Figure 3. Moving the date into the current year lets you find all birthdays in a specified week.. But if we change the specified date to December 29, 2017, we get the results shown in Figure 6.

FirstName	LastName	BusinessEntit...	BirthDate
David	Ortiz	70	1984-12-29
Michael	Rothkugel	133	1991-01-04
Ivo	Salmre	135	1982-01-03
Barbara	Moreland	245	1976-01-04
Erin	Hagens	258	1971-01-04

Figure 6. With the more complicated date translation, we can find all the birthdays in a specified week, even across the end of the year.

The final thing we'd want here is to order the results by birthday. To do that, we add an ORDER BY clause using the same complex expression, as in Listing 9; a version with this code is included in the month's downloads as SameWeekOrdered.SQL. The results for a start date of December 29, 2017 are shown in Figure 7.

Listing 9. You can use the same date translation expression to put the results in order.

```
ORDER BY CASE
  WHEN DATEPART(Month, Birthdate)
       < DATEPART(Month, @Date)
       OR (DATEPART(Month, Birthdate) =
            DATEPART(Month, @Date) AND
            DATEPART(Day, Birthdate)
            < DATEPART(Day, @Date))
  THEN DATEADD(YEAR, DATEDIFF(YEAR, BirthDate,
         @Date) + 1, BirthDate)
  ELSE DATEADD(YEAR, DATEDIFF(YEAR, BirthDate,
         @Date), BirthDate) END
```

FirstName	LastName	BusinessEntit...	BirthDate
David	Ortiz	70	1984-12-29
Ivo	Salmre	135	1982-01-03
Barbara	Moreland	245	1976-01-04
Erin	Hagens	258	1971-01-04
Michael	Rothkugel	133	1991-01-04

Figure 7. We can sort the birthdays by using the same expression that translates them.

We can use the approach to find anyone with a birthday between two specified dates. The only difference is checking whether the translated date is between the specified start and end dates, rather than between the specified date and a calculated end date; Listing 10 (included as BetweenDatesOrdered.SQL in this month's downloads) shows the code. The results are shown in Figure 8.

```
DECLARE @StartDate Date, @EndDate Date;

SET @StartDate = '12/15/2017';
SET @EndDate = '1/15/2018';

SELECT FirstName, LastName,
       Employee.BusinessEntityID, BirthDate
   FROM [HumanResources].[Employee]
     JOIN [Person].[Person]
       ON Employee.BusinessEntityID =
          Person.BusinessEntityID
   WHERE
    CASE WHEN DATEPART(Month, Birthdate) <
              DATEPART(Month, @StartDate)
           OR (DATEPART(Month, Birthdate) =
               DATEPART(Month, @StartDate) AND
               DATEPART(Day, Birthdate) <
               DATEPART(Day, @StartDate))
         THEN DATEADD(YEAR, DATEDIFF(YEAR,
           BirthDate, @StartDate) + 1,
           BirthDate)
         ELSE DATEADD(YEAR, DATEDIFF(YEAR,
           BirthDate, @StartDate), BirthDate)
     END
    BETWEEN @StartDate AND @EndDate
   ORDER BY
    CASE WHEN DATEPART(Month, Birthdate) <
              DATEPART(Month, @StartDate)
           OR (DATEPART(Month, Birthdate) =
               DATEPART(Month, @StartDate) AND
               DATEPART(Day, Birthdate) <
               DATEPART(Day, @StartDate))
         THEN DATEADD(YEAR, DATEDIFF(YEAR,
           BirthDate, @StartDate) + 1,
           BirthDate)
         ELSE DATEADD(YEAR, DATEDIFF(YEAR,
           BirthDate, @StartDate), BirthDate)
     END
```

FirstName	LastName	BusinessEntit...	BirthDate
Chris	Preston	165	1988-12-16
Betsy	Stadick	88	1966-12-17
Stuart	Macrae	230	1971-12-17
Stefen	Hesse	185	1975-12-21
Sairaj	Uddin	223	1987-12-22
Patrick	Cook	83	1973-12-23
Brian	Goldstein	85	1970-12-23
Rob	Walters	4	1974-12-23
Kevin	Liu	195	1985-12-25
Karen	Berge	220	1975-12-25
Laura	Steele	162	1980-12-25
Michael	Blythe	275	1968-12-25
Annik	Stahl	33	1976-12-26
David	Ortiz	70	1984-12-29
Ivo	Salmre	135	1982-01-03
Michael	Rothkugel	133	1991-01-04
Barbara	Moreland	245	1976-01-04
Erin	Hagens	258	1971-01-04
Tete	Mensa-Annan	284	1978-01-05
Benjamin	Martin	174	1986-01-05
Dan	Wilson	271	1976-01-06
Laura	Norman	234	1976-01-06
James	Hamilton	25	1983-01-07
Reed	Koch	175	1989-01-08
Brandon	Heidepriem	35	1977-01-10
Thomas	Michaels	45	1986-01-10
Syed	Abbas	285	1975-01-11
Jeff	Hay	160	1977-01-15

Figure 8. It's easy to get a list of everyone with birthdays (or other kinds of anniversaries) between two dates.

Identifying weekdays (and weekends)

Another common date problem is determining which days in a specified period are weekdays (or alternatively, weekend days). SQL Server's date functions make that easy.

One of the dateparts you can pass to DatePart() is weekday; the function returns a number between 1 and 7 to indicate the day of the week of the date/time you pass in. You can also use DateName() to get the name of the day in the local language. So the query in Listing 11 returns today's date, its day number, and its name; Figure 9 shows the results.

(No column name)	(No column na...	(No column na...
2017-01-05 10:44:37.177	5	Thursday

Figure 9. January 5, 2017 is a Thursday, the fifth day of the week.

Listing 11. DatePart() and DateName() let you find out the day of the week of any date/time.

```
SELECT GetDate(),
       DATEPART(weekday, GetDate()),
       DATENAME(weekday, GetDate())
```

To get the day of the week for all dates between specified start and end dates, first we need to generate the list of dates. We can do that with a recursive CTE. (See my November, 2015 article for an explanation of how this CTE works.) Then, we can simply use DatePart() to find the day of the week for each and keep only those we're interested in. Listing 12 shows how; the code is included in this month's downloads as WeekDays.SQL. Figure 10 shows partial results.

Listing 12. To get a list of weekdays in a specified period, generate all dates and keep only those with the right weekday datepart.

```
DECLARE @StartDate Date, @EndDate Date;

SET @StartDate = '12/15/2016';
SET @EndDate = '1/15/2017';

WITH AllDates (tDate)
AS
(SELECT @StartDate
    UNION ALL
    SELECT DATEADD(DAY, 1, tDate)
    FROM AllDates
    WHERE tDate < @EndDate
    )

SELECT tDate, DATENAME(WEEKDAY, tDate) AS DoW
   FROM AllDates
   WHERE DATEPART(WEEKDAY, tDate) NOT IN (1,7)
```

tDate	DoW
2016-12-15	Thursday
2016-12-16	Friday
2016-12-19	Monday
2016-12-20	Tuesday
2016-12-21	Wednesday
2016-12-22	Thursday
2016-12-23	Friday
2016-12-26	Monday
2016-12-27	Tuesday
2016-12-28	Wednesday
2016-12-29	Thursday
2016-12-30	Friday
2017-01-02	Monday
2017-01-03	Tuesday
2017-01-04	Wednesday

Figure 10. This list includes only weekdays in the specified period.

There's just one wrinkle. The weekday value returned by DatePart() is based on the SET DATEFIRST value. By default, in the US, it's 7, which makes Sunday the first day. Thus, the query in Listing 12 omits days 1 and 7, Sunday and Saturday. But with a different value for SET DATEFIRST, you have to omit different day numbers.

To make the query locale-independent, we need to compute the day of the week for Saturday and Sunday, and use the computed value. We can do that with the system variable @@DateFirst, which returns the current SET DATEFIRST value. Listing 13 shows how to compute the day numbers for Saturday and Sunday. Sunday is easier; just subtract the first day from 8. So, with @@DateFirst as 7 (Sunday), you get 1. When @@DateFirst is 1 (Monday), you get 7, and so forth.

For Saturday, it's a little trickier. 7-@@DateFirst does the job, except when @@DateFirst is 7 (Sunday). In that case, we want 7 for Saturday, but 7-@@DateFirst gives us 0. So the code handles that case separately.

Listing 13. We can calculate the day number for Saturday and Sunday using the @@DateFirst variable.

```
SET @Sunday = 8-@@DATEFIRST;
SET @Saturday =
  CASE WHEN @@DATEFIRST = 7
       THEN 7
       ELSE 7-@@DATEFIRST END;
```

To make the query generic, we do these calculations first and then use the variables @Saturday and @Sunday in the WHERE clause instead of constants, as in Listing 14 (included as WeekdaysGeneric.SQL in this month's downloads). To test that this really works regardless of the DATEFIRST setting, uncomment the SET DATEFIRST line in the example and try a few different values.

Listing 14. You can get a list of weekdays without changing your code, no matter how you've set DATEFIRST.

```
DECLARE @StartDate Date, @EndDate Date;

SET @StartDate = '12/15/2016';
SET @EndDate = '1/15/2017';

DECLARE @Saturday Int, @Sunday Int;

-- SET DateFirst 7;

SET @Sunday = 8-@@DATEFIRST;
SET @Saturday =
  CASE WHEN @@DATEFIRST = 7
       THEN 7
       ELSE 7-@@DATEFIRST END;


WITH AllDates (tDate)
AS
(SELECT @StartDate
    UNION ALL
    SELECT DATEADD(DAY, 1, tDate)
    FROM AllDates
    WHERE tDate < @EndDate
    )

SELECT tDate, DATENAME(WEEKDAY, tDate) AS DoW
   FROM AllDates
   WHERE DATEPART(WEEKDAY, tDate)
     NOT IN (@Saturday,@Sunday)
```

You can just put the Saturday and Sunday calculations in the query, rather than doing them first and storing them in variables, but I think this version is more readable.

Of course, to see weekend days only, just change NOT IN to IN in the main query.

Try some yourself

With the techniques from this article and the previous one in hand, you should be able to tackle lots of date/time challenges yourself. Let me know if you come across any particularly interesting ones.

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.*