Comparing Dates in SQL

by Gary B. Haley, and Dr. James R. Smith (v20100221)

A lot of developers have to write SQL now and then, and while most developers can write better-than-adequate SQL, one error I've seen time and time again has to do with the way dates are compared.

If your SQL is looking for records that fall between a beginning and end date, like many do, you have your choice of several different ways to accomplish this, most of which have disadvantages in differing situations. What seems to be the most natural way to find records between specified dates is to use the "BETWEEN" SQL command. *However, this **rarely** produces results that are 100% accurate.*

For example, let's say you have SQL that looks in a table for records that have a date that falls between two dates, which are passed as parameters, like a weekly report. The first parameter, **@BegDate**, is passed **'01/01/2010'**. The second parameter, **@EndDate**, is passed **'01/07/2010'** to search the following records in a temp table:

```
ID    Description DateBeingSearched
--    ----------- ----------------------
30    Whatever1   01/01/2010 12:20:09.124
45    Whatever2   01/05/2010 16:05:54.855
88    Whatever3   01/07/2010 09:34:16.435
92    Whatever4   01/08/2010 00:00:00.000
```

Case 1:

```
SELECT t.ID                    or:   SELECT t.ID
FROM dbo.temp AS t                   FROM dbo.temp AS t
WHERE t.DateBeingSearched            WHERE ( (t.DateBeingSearched >= @BegDate)
  BETWEEN @BegDate                     AND (t.DateBeingSearched <= @EndDate) );
  AND @EndDate;
```

The queries will return ID # 45, for sure, but 88 is not returned when @EndDate is '01-07-2010 00:00:00.000' but COULD BE returned when Now() or getDate() is used to assign the date to @EndDate. ID # 30 COULD possible be returned if Now() or getNow() is used for @BegDate.

Case 2 (adding a day to end date):

```
SELECT t.ID                    or:   SELECT t.ID
FROM dbo.temp AS t                   FROM dbo.temp AS t
WHERE t.DateBeingSearched            WHERE ( (t.DateBeingSearched >= @BegDate)
  BETWEEN @BegDate                     AND (t.DateBeingSearched < @EndDate + 1) );
  AND @EndDate + 1;
```

The queries will return ID #'s 45, 88, and possibly 30 and 92 if @BegDate and @EndDate include times, however, ID # 30 is within the intent of this query and ID # 92 is outside of the intent of this query.

Case 3 (converting to string):

```
SELECT t.ID
FROM dbo.temp AS t
WHERE convert(char (10), t.DateBeingSearched, 101)
  BETWEEN convert (char (10), @BegDate, 101)
  AND convert (char(10), @EndDate, 101);
```

This query will return expected results, **but it can take up to 3 times as long**, depending on the rest of the SQL. (Tests have shown the average to be about 30% longer, which can push the time out beyond IIS timeouts.)

A quick **TEMPORARY** fix is to put the following SET statements, or something similar, near the top of the query, which will temporarily fix it to return more accurate data, but still might leave off records that were saved within a few milliseconds before midnight. (This temporary fix might be acceptable as a permanent fix in cases where you know there will be no one saving records at or around midnight.)

```
--Zero out the time portion of the date (below is easily the fastest way to accomplish that per SQL Server 2005 tests):
SET @BegDate = convert(char(8), @BegDate, 1);

--Put the maximum time possible in the end date:
SET @EndDate = dateAdd(hh,  23 - datePart(hh, @EndDate), @EndDate);
SET @EndDate = dateAdd(n,   59 - datePart(n,  @EndDate), @EndDate);
SET @EndDate = dateAdd(s,   59 - datePart(s,  @EndDate), @EndDate);
SET @EndDate = dateAdd(ms, 985 - datePart(ms, @EndDate), @EndDate);
```

This alters the beginning date to include ALL records on that date and alters the end dates to include ALL **or MOST** records on that date.

Did you notice the ".985" milliseconds instead of ".999"? This is by design, to account for the PC's internal rounding. No PC is accurate to the millisecond, and might round to the nearest .025 of a second, or .001. If it rounds up from .999, this will not only go to the next second, but to the next day as well. If you use ".999" instead of ".985" the example above will also return the record with the date: 01/08//2010 00:00:00.000. The chances of a record having a time between 23.59.59.986 and .999 are so remote it is probably completely insignificant because in *most* cases (keep in mind your business requirements might include international business) it is after normal business hours **and** most computers would have likely rounded it up the next day when the record was saved.

Proof –run this in your query analyzer (the results below show my desktop rounding at about 3 or 4 milliseconds):

```
DECLARE @dt dateTime
SET @dt = '2010-01-20 23:59:59.990' SELECT @dt AS '990'; -- returns: 2010-01-20 23:59:59.990
SET @dt = '2010-01-20 23:59:59.991' SELECT @dt AS '991'; -- returns: 2010-01-20 23:59:59.990
SET @dt = '2010-01-20 23:59:59.992' SELECT @dt AS '992'; -- returns: 2010-01-20 23:59:59.993
SET @dt = '2010-01-20 23:59:59.993' SELECT @dt AS '993'; -- returns: 2010-01-20 23:59:59.993
SET @dt = '2010-01-20 23:59:59.994' SELECT @dt AS '994'; -- returns: 2010-01-20 23:59:59.993
SET @dt = '2010-01-20 23:59:59.995' SELECT @dt AS '995'; -- returns: 2010-01-20 23:59:59.997
SET @dt = '2010-01-20 23:59:59.996' SELECT @dt AS '996'; -- returns: 2010-01-20 23:59:59.997
SET @dt = '2010-01-20 23:59:59.997' SELECT @dt AS '997'; -- returns: 2010-01-21 00:00:00.000
SET @dt = '2010-01-20 23:59:59.998' SELECT @dt AS '998'; -- returns: 2010-01-21 00:00:00.000
SET @dt = '2010-01-20 23:59:59.999' SELECT @dt AS '999'; -- returns: 2010-01-21 00:00:00.000
```

By the way, you have no way of knowing which has the most accurate rounding - the PC / server running the report or the PC / server saving the record, which adds to the mix of uncertainty.

Comparing Dates in SQL

by Gary B. Haley, and Dr. James R. Smith (v20100221)

There are at least two good solutions to the problem of finding accurate **daily** report data with SQL that searches for records that fall between two dates, which is, by far, the most common (and, hopefully, your SQL is in a stored procedure and *not* in your code):

Solution 1 (zero-out the time, add a day to the end date, and use less-than):

```
--Tip:  For easier maintenance and readability, limit naming SQL objects to the 26 letters of the alphabet and 0-9.
CREATE PROCEDURE dbo.UseAThruZAnd0Thru9OnlyInSPNames
      (
        @BegDate dateTime
      , @EndDate dateTime
      )
AS
BEGIN
      --Tip: Set NOCOUNT to "ON" which turns off count returns - saves time, broadband and network traffic:
      SET NOCOUNT ON;

      --Zero-out the times in both the beginning and end search dates, in case they include times other than 00:00:00.000:
      SET @BegDate = convert(char(10), @BegDate, 101);
      SET @EndDate = convert(char(10), @EndDate, 101);
      --End of zero-out-times statements^

      SELECT st.ID
      FROM dbo.SomeTable AS st
      WHERE ( (st.ADate >= @BegDate)
        AND (st.ADate < @EndDate + 1) );
        --Note:  The WHERE clause above only works when used with the statements above that zero out times.
END
GO
```

Solution 2 (use integers instead of dates):

```
--Tip:  Never start a stored procedure name with "sp_".
--       (Many system SPs begin with "sp_" and have to be searched through with every use of that stored procedure.)
CREATE PROCEDURE dbo.NeverUseUnderscoresInSPNames
      (
        @BegDate dateTime
      , @EndDate dateTime
      )
AS
BEGIN
      SET NOCOUNT ON;

      DECLARE
              @Beg int --Do not use smallInt or tinyInt
            , @End int --Do not use smallInt or tinyInt
            , @RefDate dateTime; --Practically any date - used in dateDiff() to compare

      -- Zero out dates and add a day:
      SET @RefDate = convert(char(10), @RefDate + 1, 101);

      SET @Beg = dateDiff(d, @RefDate, @BegDate);
      SET @End = dateDiff(d, @RefDate, @EndDate);

      SELECT st.ID
      FROM dbo.SomeTable AS st
      WHERE dateDiff(d, @RefDate, st.SomeDateInTheDB) BETWEEN @Beg AND @End;
END
GO
```

Lessons Learned: **Solution 1 might be best used to correct the output of existing SQL and seems to be easier for most developer-type minds to follow, while Solution 2 (Dr. James R. Smith's input) is probably best used during new SQL development and / or very large data sets, as tests have shown it to be slightly faster more often.**

The most important lesson to be learned here though, and the point of the efforts put forth in this article, is to never, EVER, use dates in a BETWEEN SQL statement. (Feedback welcomed!)