Working with Time Spans and Durations in SQL Server

Written by **Jeff Smith** on **15 October 2007**

What is the best way to return the "duration" of an event in SQL, given the start and end datetime values? How can we add up these durations to return grand totals? What data types should be used to return this data to our clients? How do we handle overflows, such as when hours go over 23 or minutes total up to over 59? Are there any T-SQL functions or other techniques that are useful in these scenarios?

Introduction

The concept of durations and TimeSpans in SQL causes lots of confusion for beginners and experienced programmers alike. Let's take some time to examine some options we have when storing, working with and returning this kind of data.

I will use the following table in my examples, if you'd like to play along at home:

```
create table Events
(
        EventID int identity primary key,
        StartDate datetime not null,
        EndDate datetime not null
)

go

insert into Events (StartDate, EndDate)
select '2007-01-01 6:34:12 AM', '2007-01-01 12:45:34 PM' union all
select '2007-01-02 9:23:08 AM', '2007-01-02 5:05:37 PM' union all
select '2007-01-03 4:34:12 PM', '2007-01-03 4:55:18 PM' union all
select '2007-01-04 11:02:00 AM', '2007-01-04 2:53:21 PM' union all
select '2007-01-05 7:52:55 AM', '2007-01-05 9:08:48 AM' union all
select '2007-01-06 7:59:11 PM', '2007-01-07 1:23:11 AM' union all
select '2007-01-07 3:12:23 AM', '2007-01-07 8:02:25 PM'

go

select * from Events

TranID      StartDate                 EndDate
----------- ------------------------- -----------------------
1           2007-01-01 06:34:12.000   2007-01-01 12:45:34.000
2           2007-01-02 09:23:08.000   2007-01-02 17:05:37.000
3           2007-01-03 16:34:12.000   2007-01-03 16:55:18.000
4           2007-01-04 11:02:00.000   2007-01-04 14:53:21.000
5           2007-01-05 07:52:55.000   2007-01-05 09:08:48.000
6           2007-01-06 19:59:11.000   2007-01-07 01:23:11.000
7           2007-01-07 03:12:23.000   2007-01-07 20:02:25.000

(7 row(s) affected)
```

This is just a simple table of Events that have both a start and an end DateTime. Let us suppose that we need to calculate, for each Event, the duration of that event. We'd also like to ultimately return the total duration of all of the events combined.

Defining "Duration"

As always, the first step has nothing to do with writing code -- it is clarifying the specifications. What do we mean by "duration"? What type of data, specifically, do we need to return? A integer representing the total number of minutes, or a DateTime offset from a certain value, or several integers indicating Hours/Minutes/Seconds, or a string in some specific format (hopefully not!), or something else? The term "duration" doesn't clearly indicate what is needed, it is too general. Also, how accurate do we need to be -- can we round to the nearest hour, or minute, or second? And what happens if our total durations spans more than 24 hours -- do we break it down into days?

Typically, for durations that spans hours, seconds are as accurate as you need to be. (Plus, the DateTime data type is only accurate to a 1/300th of a second anyway, not a true millisecond.) A handy function, DateDiff(), can return for any two the dates the difference in any specific unit that we wish -- hours, seconds, minutes, and so on. (For more information about how DateDiff() works see DATEDIFF Function Demystified.) So, we can call DateDiff(second, Date1,Date2) and easily get the total number of seconds between two dates, which we can use to get the duration of each event in seconds:

```
select *, DateDiff(second, StartDate, EndDate) as TotalSeconds
from Events

EventID     StartDate                 EndDate
TotalSeconds
----------- ------------------------- ------------------------- ------
-----------
1           2007-01-01 06:34:12.000   2007-01-01 12:45:34.000   22282
2           2007-01-02 09:23:08.000   2007-01-02 17:05:37.000   27749
3           2007-01-03 16:34:12.000   2007-01-03 16:55:18.000   1266
4           2007-01-04 11:02:00.000   2007-01-04 14:53:21.000   13881
5           2007-01-05 07:52:55.000   2007-01-05 09:08:48.000   4553
6           2007-01-06 19:59:11.000   2007-01-07 01:23:11.000   19440
7           2007-01-07 03:12:23.000   2007-01-07 20:02:25.000   60602

(7 row(s) affected)
```

Now, technically, we have returned the durations -- but it may not be very useful in that format. How can we take the total number of seconds and turn it into something more easily analyzed? The answer is easy: Use some basic math!

Converting Time Units With Math

I'm sure that (hopefully?) we all know that:

- To convert seconds to hours, simply divide by 3600 (since each hour has 60 seconds * 60 minutes). The remainder is the remaining seconds.
- To convert seconds to minutes, simply divide by 60. The remainder is the remaining seconds.

Nothing too shocking there, right? So, let's do some math. If we have a TotalSeconds, we can get:

- Hours = (TotalSeconds / 3600)
- Remaining Minutes = (TotalSeconds % 3600) / 60
- Remaining Seconds = (TotalSeconds % 60)

(The % is the modulo operator in T-SQL, which returns the remainder when dividing two integers.)

Thus, we can write our SQL like this to return 3 integer columns (Hours, Minutes, Seconds) for each event:

```
select
  EventID,
  TotalSeconds / 3600 as Hours,
  (TotalSeconds % 3600) / 60 as Minutes,
  TotalSeconds % 60 as Seconds
from
(
        select EventID, DateDiff(second, StartDate, EndDate) as
TotalSeconds
        from Events
) x
```

```
EventID     Hours       Minutes     Seconds
----------- ----------- ----------- -----------
1           6           11          22
2           7           42          29
3           0           21          6
4           3           51          21
5           1           15          53
6           5           24          0
7           16          50          2

(7 row(s) affected)
```

Now our results like a lot nicer! We can easily see how long each event is in units that we can quickly identify and work with. Our clients can also easily format the Hours/Minutes/Seconds values into whatever format they need since we are returning clean integer values; no string parsing or converting is required.

Calculating Duration Totals

Now that we have the duration for each event, how do we get the grand totals?

At first glance, it may appear that we could simply sum up our Hours/Minutes/Seconds calculations by wrapping them in SUM() expressions:

```
select
    sum(TotalSeconds / 3600) as Hours,
    sum((TotalSeconds % 3600) / 60) as Minutes,
    sum(TotalSeconds % 60) as Seconds
from
(
    select EventID, DateDiff(second, StartDate, EndDate) as
TotalSeconds
    from Events
) x

Hours       Minutes     Seconds
----------- ----------- -----------
38          214         133

(1 row(s) affected)
```

However, look at those results -- they do not make much sense, we have Seconds and Minutes greater than 59 returned. We'd need to do further math to carry all seconds over 59 to the Minutes column, and then carry Minutes over 59 to the Hours column, and we may wish to return the total number of days as well for hours over 23. It sure seems like this just got very complicated!

The answer, of course, is to keep it simple -- just add up the total seconds for all events first, and then calculate the resulting Hours/Minutes/Seconds from that total:

```
select
  sum(TotalSeconds) / 3600 as Hours,
  (sum(TotalSeconds) % 3600) / 60 as Minutes,
  sum(TotalSeconds) % 60 as Seconds
from
(
    select EventID, DateDiff(second, StartDate, EndDate) as
TotalSeconds
    from Events
) x

Hours       Minutes     Seconds
----------- ----------- -----------
41          36          13

(1 row(s) affected)
```

Now that looks a lot better. The last SELECT may look the same as the previous one, but examine it closely: You will see that we are now taking the SUM(TotalSeconds) first, and then using our formulas on those values. Previously, we did the math *first* and *then* added up the results. This causes a very big difference in the results!

If we'd like to return the total days as well for hours over 24, again we just use some basic algebra to get what we need. Each day has 60 seconds * 60 minutes * 24 hours = 86,400 seconds in it, so we just write:

```
select
  sum(TotalSeconds) / 86400 as Days,
  (sum(TotalSeconds) % 86400) / 3600 as Hours,
  (sum(TotalSeconds) % 3600) / 60 as Minutes,
  sum(TotalSeconds) % 60 as Seconds
from
(
    select EventID, DateDiff(second, StartDate, EndDate) as
TotalSeconds
    from Events
) x

Days        Hours       Minutes     Seconds
----------- ----------- ----------- -----------
1           17          36          13

(1 row(s) affected)
```

Once again, the client can easily format these 4 values any way necessary since we are returning this nice raw data.

Representing TimeSpans and Durations with DateTime

Finally, there is another option instead of breaking the values down into integer units; we could just return regular DateTime data, offset from the "base date" of 1900-01-01 at 12:00:00 AM. That 1900-01-01 date is the mathematical equivalent of a "0", in that it is always the result of subtracting any date from itself, and adding it to any other date will have no affect:

```
declare @d datetime
declare @BaseDate datetime

set @d= '2005-02-05 6:23:51 PM'

set @BaseDate =  @d - @d

select @BaseDate as BaseDate, cast(0 as datetime) as BaseDateAsZero

select @d as Date, @d + @BaseDate as DatePlusBaseDate, @d + 0
DatePlusZero

BaseDate                    BaseDateAsZero
--------------------------- --------------------------
1900-01-01 00:00:00.000     1900-01-01 00:00:00.000

(1 row(s) affected)

Date                        DatePlusBaseDate          DatePlusZero
--------------------------- ------------------------- -----------------
--------
```

```
2005-02-05 18:23:51.000    2005-02-05 18:23:51.000   2005-02-05
18:23:51.000

(1 row(s) affected)
```

Take a few minutes to really examine the above code, and play with it yourself as well if necessary to get a feel for what happens when you add an subtract DateTimes. Thus, we could calculate our Duration column using standard DateTime data like this:

```
select *, EndDate-StartDate as Duration
from Events

EventID     StartDate                 EndDate
Duration
----------- ------------------------- ------------------------- ------
-----------------------
1           2007-01-01 06:34:12.000   2007-01-01 12:45:34.000   1900-
01-01 06:11:22.000
2           2007-01-02 09:23:08.000   2007-01-02 17:05:37.000   1900-
01-01 07:42:29.000
3           2007-01-03 16:34:12.000   2007-01-03 16:55:18.000   1900-
01-01 00:21:06.000
4           2007-01-04 11:02:00.000   2007-01-04 14:53:21.000   1900-
01-01 03:51:21.000
5           2007-01-05 07:52:55.000   2007-01-05 09:08:48.000   1900-
01-01 01:15:53.000
6           2007-01-06 19:59:11.000   2007-01-07 01:23:11.000   1900-
01-01 05:24:00.000
7           2007-01-07 03:12:23.000   2007-01-07 20:02:25.000   1900-
01-01 16:50:02.000

(7 row(s) affected)
```

Compare those results with when we returned Hours/Minutes/Seconds as integers -- if you ignore the date portion, you will see they are the same values, just formatted differently. Thus, our client applications could just omit the Date part of these DateTime durations, and we'd be good to go. There are two issues with this, however:

1. We cannot use SUM() on DateTime data to get totals
2. We cannot completely ignore the Date portion -- what if it overflows to 1900-01-02 ? (i.e., the duration is greater than 24 hours)

Calculating the SUM() of DateTime Values

Let's start with the first point. If we try to write:

```
select sum(EndDate-StartDate) as Duration
from Events
```

we get:

```
Server: Msg 409, Level 16, State 2, Line 1
The sum or average aggregate operation cannot take a datetime data
type as an argument.
```

To work around this, we could convert our DateTime values to float or decimal or some other data type that can be used with SUM(), add up that converted data, and then convert back to a DateTime:

```
select cast(sum(cast(EndDate - StartDate as float)) as DateTime) as
TotalDuration
from Events

TotalDuration
-----------------------------------------------------
1900-01-02 17:36:13.000

(1 row(s) affected)
```

Or, we could simply add up the number of seconds for each duration as we did before, and then add the TotalSeconds returned to the "BaseDate":

```
select dateadd(second, sum(datediff(second,startDate,EndDate)),0) as
TotalDuration
from Events

TotalDuration
-----------------------------------------------------
1900-01-02 17:36:13.000

(1 row(s) affected)
```

That works equally well. Break down the two formulas if necessary to see how they are working; or if you like, re-write them using Derived tables to make it more clear. As always, I encourage readers to stop and experiment with simple little scripts like these to test out new concepts to see exactly how things work -- it is much easier and quicker than doing it on your live data, right?

DateTime Durations That Span Multiple Days

Finally, addressing point #2, if we return our duration using the DateTime data type, we need to be sure that our client does not ignore dates other than 1900-01-01. In this case, our TotalDuration value has a date of 1900-01-02, meaning that the duration is not just 17 hours but one day plus 17 hours. There is really no way of ensuring or guaranteeing that the clients will do this, especially if they are written to simply use a format that hides the datetime portion, so what you could do is either a) return the values broken out into days/hours/minutes/seconds as shown previously or b) always return the time at the base date (1900-01-01) and return an additional "Days" column.

To return each unit broken down into integers, we can just use the previous calculations shown on TotalSeconds. However, if we are starting with a DateTime offset from 1900-01-01, we can use the following simple T-SQL Date formulas to break it down:

```
declare @Duration as DateTime
set @Duration = '1900-01-02 17:36:13.000'

select
```

```
  DateDiff(day, 0, @Duration) as Days,  -- note that 0 equals 1900-01-
01, the "base date"
  DatePart(Hour, @Duration) as Hours,
  DatePart(Minute, @Duration) as Minutes,
  DatePart(Second, @Duration) as Seconds

Days        Hours       Minutes     Seconds
----------- ----------- ----------- -----------
1           17          36          13

(1 row(s) affected)
```

You will see that we just got back to our original 4 integers from the DateTime value, but this time we used the Date functions provided by T-SQL.

The other option, as mentioned, is to return our results in a DateTime format that always uses 1900-01-01 but also includes a separate integer Days column. We can simply calculate that results like this:

```
declare @Duration as DateTime
set @Duration = '1900-01-02 17:36:13.000'

select
  DateDiff(day, 0, @Duration) as DurationDays,  -- note that 0 equals
1900-01-01, the "base date"
  @Duration - DateDiff(day, 0, @Duration) as DurationTime

DurationDays   DurationTime
-------------- -------------------------
1              1900-01-01 17:36:13.000

(1 row(s) affected)
```

In the above, we are calculating the days as in our previous formula, and then simply subtracting that many days back out from our original Duration ensuring that the DateTime component always returns the time on 1900-01-01. Thus, clients can simply ignore the Date portion when outputting the results, and the additional Days column makes it easy and clear that they need to output and/or handle the Days component as well.

The advantage of returning a DateTime value over Hours/Minutes/Seconds is that clients can easily ignore the date, as mentioned, and format the time using flexible options such as leading zeroes, adding AM or PM versus military time, and so on. It is quite easy to construct a Time format at most clients using integer Hour/Minute/Second values as well, but sometimes it may take more work. Either way, the key is that we are returning properly typed data from our database and not pre-formatted strings that cause more work for both the database and the client.

What About Months and Years?

On a final note, what if we wish to break down our Durations or TimeSpans into Months or Years? The answer is that you cannot -- those units are not precise and vary from year to year and month to month, so the largest unit you can calculate with a TimeSpan is Days. After all, how many months is 29 days? It could be one, it could be zero. And you could break 35 days down into 1 month/4 days (for months with 31 days), or 1 month/5 days (30 day months), and so on. Even years vary between 365 and 366 days. You can always estimate or round TimeSpans to these units (i.e., 360 days is approximately 12 months, or 731 days is approximately 2 years) but you cannot return and calculate precise results in those units.

Summary

I hope this has given you some guidance and ideas on how to effectively work with Durations and TimeSpans in SQL Server using the DateTime data type as well as units broken down into integers. The approach to take varies depending on your needs, but as always: Keep it simple, keep it accurate, prefer math over string parsing, return clean data to your clients, and don't try to format things at the database layer.

(There's more information on TimeSpans and lots of handy User-Defined Functions that you can use to work with both DateTime and TimeSpan data on my blog post Essential SQL Server Date, Time and DateTime Functions.)