TSQL - Working with dates

|  |  |
| --- | --- |
| **Sample TSQl Date Functions to support Dates walkthrough. See Blackboard for supporting file.** |  |

Contents

[TSQL - Working with dates Introduction 1](#_bookmark0)

1. [Introduction 2](#_bookmark1)
2. [Getting the date 2](#_bookmark2)
3. [Creating the test data 2](#_bookmark3)
4. [Locale issues 3](#_bookmark4)
5. [Displaying dates 3](#_bookmark5)
6. [Converting our broken table 4](#_bookmark6)
7. [Working with parts of dates 5](#_bookmark7)
8. [More issues with using strings 6](#_bookmark8)
9. [Converting/comparing dates 7](#_bookmark9)
10. [Comparing dates and times 8](#_bookmark10)
11. [More date functions 10](#_bookmark11)
12. [Last day of month 11](#_bookmark12)
13. [Setting the first day of the week 11](#_bookmark13)
14. [Query to find upcoming birthdays for the current week 11](#_bookmark14)
15. [Conclusion 13](#_bookmark15)

# Introduction

One of the reasons I write these SQL Server Quick Guides, is so that when I see commonly asked questions in our lab sessions that require an in depth answer I can pass them onto a support guide.

This year we have several asking about how SQL Server works with Dates and/or Time. 'when storing dates in SQL Server, should we use the datetime datatype, or varchar'?

Certainly there's often questions about changing the format of dates that indicate that the user has stored the date as a string.

So this SQl Server Quick Guide focuses on Dates and provided are some walk through guide for to illustrate how best to work with dates and times in SQL Server, and along the way, establish why there are things we should NEVER do.

# Getting the date

How to determine todays date in SQl Server?

select CURRENT\_TIMESTAMP, getdate(), GETUTCDATE()

Will get the current date in your timezone twice, and then GETUTCDATE gets the UTC date, as you'd expect. If you are old, like me, UTC is the same as GMT. If you're not, then, UTC is the time localised to have no time zone off set. According to the documentation CURRENT\_TIMESTAMP and getdate() differ in this respect ( the docs say getdate() uses the timezone, and does not say that CURRENT\_TIMESTAMP does ). This is wrong. They are both the same thing. The difference is that CURRENT\_TIMESTAMP is standard SQL, and so that's the one you should use ( because it should work anywhere ). getdate() is a SQL Server function, it is not guaranteed to work on other SQL platforms.

# Creating the test data

First, we're going to create some data. Our table stores names and birthdays. It's pretty simple, but none of the concepts I am going to show you, would change if the table had a lot of columns besides the ones we are using.

create table tblMember (

id int identity, name varchar(100), DOB varchar(50)

)

insert into tblMember values('Fred', '1969/02/17'), ('Bill', '1971/07/04'), ('Bob', '1973/09/15'), ('Sarah', 'The first Monday in 1982')

insert into tblMember values('Robert', '05/14/1995'), ('Hank', '13/05/2001'), ('Laura', 'Feb 17 1969'), ('Emma', 'Not telling')

You can probably already see some issues with this data. If you ever have to work with data like this, and you've got a big table with a few bad values, there are ways to get a list of the rows with broken values in them. Here's how you can do it in SS2012

select id, name, dob from tblMember where try\_parse(DOB as date) is null

But this version, which works in older versions, is just as good.

select id, name, dob from tblMember where isdate(DOB) = 0

# Locale issues

One thing you may have noted with the test data is that, between all the other entries, there are a number of entries that are clearly yyyy/mm/dd and some that are clearly yyyy/dd/mm. It's more common actually for people to store dates as mm/dd/yyyy or dd/mm/yyyy, and just assume the locale of their system will never change. The only date format that is locale proof, is **yyyymmdd**, with no slashes. So, if you must store a date as a string, that is the only reasonable way to do it, unless you want to mess with longer strings that use text for the month ( which I'll admit is more human readable, but it's still a two step process to convert it in order to change the string format

). However, again, no string format is more clearly a date, than the date format. As you can see from the other entries I've inserted, if you store a date as a string, and especially if you collect it as a string, then there are all sorts of possible entries that are human understandable, and useless in an automated environment ( for example, if we wanted to send all our users birthday cards on their birthday ).

# Displaying dates

Of course, it's common to want to pull out dates in a specific format, and this is one of the strengths of the date time type, it is pure data, but you can turn it in to any display string format you like. SQL Server can also implicitly try to convert strings to dates, if they are used in place of dates, but this is both slow, and if any error occurs, the whole request will fail. Try this to see:

select CONVERT(nvarchar(30), DOB, 109) from tblMember

You can try this:

select DOB, CONVERT(nvarchar(30), DOB, 109) from tblMember where isdate(DOB) = 1

And it will return the original date strings, it won't format them. Convert works for all possible types, so it's not at all clear, even with the format identifier, that you want to work with a date time. This is what you need to do:

select DOB, CONVERT(nvarchar(30), convert(datetime,DOB), 109) from tblMember where isdate(DOB) = 1

I am not sure why SQL Server fails to do the convert for invalid dates, yet ends up not realising it needs to work with dates, but either way, it's pretty obvious to me that turning a string in to a date, so you can turn it in to a formatted string, is insane.

For your reference, [this](http://msdn.microsoft.com/en-us/library/ms187928.aspx) is a link to the Microsoft documentation on the possible numeric values to pass in to define the string format you want a date converted to.

# Converting our broken table

So, if you have to work on a system like this, what you should you do ? I'd start by adding a new column, and making any insert and update code maintain both values ( triggers can be used in this regard ). Then you can convert all your select statements at your leisure, testing each one as you go, until you're sure the old column is no longer used, and delete it.

Let's add that column now, and insert our values in to it:

Alter Table tblMember Add Birthday date

Now we can try to insert our values.

update tblMember set Birthday = DOB

If a statement like this works, then you had no bad rows in your database. Otherwise you'll need to do this:

update tblMember set Birthday = try\_parse(DOB as date)

or for before SS2012:

update tblMember set Birthday = convert(date,DOB) where isdate(dob)

Now run this SQL:

select \* from tblMember

You will see something like this:

id

1

2

3

4

5

6

7

8

name

Fred Bill Bob Sarah Robert Hank Laura Emma

DOB

Birthday

1969/02/17

1971/07/04

1973/09/15

1969-02-17

1971-07-04

1973-09-15

The first Monday in 1982

NULL

05/14/1995

13/05/2001

Feb 17 1969

Not telling

1995-05-14

NULL

1969-02-17

NULL

Notice that our birthdays are all in the same format now, making them much easier to read. Notice also that nonsensica entires are gone. If the field is not NULL, we know what it's format is, and we know how to work with it.

# Working with parts of dates

Often, a date or datetime is just a lump of data that you consider in one go. However, sometimes you want to work with parts of it, for example when you select orders from a database and want to group by the month of the order. SQL Server has several functions to help with this. There are functions called day, month and year, which are very convenient for when you want to group by these things, but there is also a DatePart function, which can also return all of the values listed above.

From SQL Server 2008, there are types for dates without times and times without dates, as well as datetime2, which is more standards compliant than the old datetime format. As such, these types should be preferred, but while I will touch on them, and I used date and not datetime in our table, a lot of my examples use datetime because I am sure it's the thing you'll run in to most in the real world. Datetime2 can express a larger range of dates that datetime, as well as having a larger default level of precision ( which can also be changed by the user )

Mansha Nawaz SQL Guides Page 5 of 14

As always, trying to work with our string field, breaks

select datepart(dw, dob) from tblMember

Now that our function knows we require a conversion to datetime, this will work:

select datepart(dw, dob) from tblMember where isdate(dob) = 1

If we had SS2012 we could use try\_parse and get nulls ( which would mean our select couldn't skip valid rows altogether because of a bad value ), but it's much cleaner with a datetime type:

select datepart(dw, birthday) from tblMember

Of course, from here we can do more interesting things, such as working out how many days someone has been alive:

select datediff(dd, birthday, getdate()) from tblMember

Or their age in years ( this is probably more useful )

select datediff(yy, birthday, getdate()) from tblMember

We can even define a custom string that contains elements of the date

select convert(char(4), year(birthday)) + '->' + case when month(birthday) < 10 then '0' else '' end + convert(char(2), month(birthday)) + '->' +

case when month(birthday) < 10 then '0' else '' end + convert(char(2), day(birthday)) from tblMember

That last one is definitely easier in SS2012

select format(birthday, 'YYYY->MM->dd') from tblMember

# More issues with using strings

Let's change gear and talk about other things that will break if you store dates as strings. It's very common to write SQL that specifies a date as a string. If your date IS a string, then this means you need to match the format. Try running this:

select \* from tblMember where dob = 'Feb 17 1969'

Compare to this result:

select \* from tblMember where birthday = 'Feb 17 1969'

Because our date time field correctly cares about the underlying data, and not the format, this will work properly.

The same is true if we want to order our result. Try running this:

select \* from tblMember order by dob desc

As the field is a string, it's been sorted in text order. However, this is probably the order we wanted:

select \* from tblMember order by birthday desc

Of course, we could cast this to datetime, but, if we know a datetime is what we want, why not store it that way in the first place ?

# Converting/comparing dates

One thing that happens commonly is that older databases use datetime to store a date, and then want to compare to get rows where the date is today, or another fixed day. There's a lot of code on the web for stripping the time component of a date time, most of it complex and hard to remember. The most efficient way of doing this is actually super easy to remember:

declare @now datetime = getdate()

declare @nowDate date = convert(date, @now) declare @nowTime time = convert(time, @now) select @now, @nowDate, @nowTime

Casting a date time to a date simply strips the time. Casting a date or time to datetime, will populate the extra info with the default, which is a good way to establish what the default date is ( the default time is midnight, not surprisingly )

declare @now datetime = getdate()

declare @nowTime time = convert(time, @now)

set @now = convert(datetime, @nowTime) select @now

The default date is 1 Jan, 1901. This will become important, in the next section.

# Comparing dates and times

In order to do some comparisons, we need first to create some values, which we do as follows:

declare @now datetime = getdate()

declare @then datetime = dateadd(dd, 45, @now) set @then = dateadd(minute, 164, @then) select @now, @then

declare @nowDate date = convert(date, @now) declare @nowTime time = convert(time, @now) declare @thenDate date = convert(date, @then) declare @thenTime time = convert(time, @then)

This creates a date and a time, for today, and 45 days and 164 minutes from now.

select @then + @now select @then - @now

This returns the following:

2128-06-26 03:34:53.367

1900-02-15 02:44:00.000

Jan 1, 1900 is the baseline, it is 'zero' and so adding or subtracting values like this works, but what it gives you isn't especially useful.

select convert(time, (@then - @now))

returns

02:44:00.0000000

which is to say, the time difference between the two values, removing any difference in days.

select @thenDate + @nowDate

select @thenTime + @nowTime select @thenDate - @nowDate select @thenTime - @nowTime

All of these generate errors. I suspect that allowing + and - on datetimes is something that was done without too much thought being put in to it, and because it does nothing terribly useful, it's been removed for the newer types of DATE or TIME. The correct way to compare dates is the DATEDIFF function. It's fully documented here: [datediff.](http://www.w3schools.com/sql/func_datediff.asp)

In short, the datediff function takes two dates and the part to compare. Here is the list of possible values. Note: the full names AND the abbreviations are both valid.

|  |  |
| --- | --- |
| datepart | Abbreviation |
| year | yy, yyyy |
| quarter | qq, q |
| month | mm, m |
| dayofyear | dy, y |
| day | dd, d |
| week | wk, ww |
| weekday | dw, w |
| hour | hh |
| minute | mi, n |
| second | ss, s |
| millisecond | ms |
| microsecond | mcs |
| nanosecond | ns |

Note that DATEDIFF always uses Sunday as the first day of the week ( although later on I show you how to change what the first day of the week is for other operations ).

select datediff(dd, @nowdate, @thendate) select datediff(mm, @nowdate, @thendate)

*-- this also works with datetimes* select datediff(day, @now, @then) select datediff(mm, @now, @then)

*-- A combination of dates and datetimes is fine*

select datediff(dd, @now, @thenDate) select datediff(mm, @nowDate, @then)

select datediff(minute, @nowTime, @thenTime)

These all return the same values. 45 for the dd check, mm for the month check. The minute check return 164, so it's a total minutes, not minutes disregarding other larger values.

select datediff(minute, @now, @thenTime)

This returns -59988960. This is because when we pass in a time, a date of 1/1/1900 is assumed and taken in to account. Of course, as we saw above, passing in two times works fine ( because the assumed date is the same for both )

select datediff(minute, @nowDate, @thenDate) select datediff(minute, @now, @thenDate)

Just using dates and asking for a difference that is time based, creates it's own issues, because it assumes a time of midnight. The values here are 64800 and 64037, for me right now. The difference between the two, is the difference between the actual time ( lunchtime ) and midnight, which is the time in @nowDate.

It's worth noting that I'm talking about a number of different types in SQL Server, both DATE and TIME were new in SS 2008, as were DATETIME2 and DATETIMEOFFSET. So if you are in SS2005, your options will be limited.

# More date functions

There are two clusters of new methods for dates. The second cluster is a single function, but all the others are basically the same thing. They take the form of XXXFromParts, and can create each of the possible date or time types, from arguments representing the possible parts. They are as follows:

DATEFROMPARTS(2012, 12, 1)

DATETIMEFROMPARTS(2012, 12, 1, 10, 15, 30, 000)

-- the last argument is how many significant chars are in the millisecond part DATETIME2FROMPARTS(100, 12, 1, 10, 15, 30, 279, 5)

SMALLDATETIMEFROMPARTS(1900, 12, 1, 10, 15)

-- Againt, the milliseconds have a variable # of chars TIMEFROMPARTS(12,30,15,1, 2)

-- The last three arguments are an hour and minute offset, and a precision SELECT DATETIMEOFFSETFROMPARTS ( 2010, 12, 31, 14, 23, 23, 0, 12, 0, 7 )

# Last day of month

The other new function is EOMONTH, which calculates the last day of a month. EOMONTH(date) returns the last day of that month. EOMONTH(date, offset) returns the last day of the month that is 'offset' months forward or backwards from the month passed in.

# Setting the first day of the week

The datepart 'dw' returns the day of the week. The default for English is Sunday. If you write any code that uses the day of the week for any purpose, apart from DATEDIFF ( which ignores the setting ), then you should capture the current value, set it to what you expect, and reset it when you are done. You can get the current first day of the week using @@DATEFIRST, as in

SELECT @@DATEFIRST

or

declare @firstDay int

set @firstDay = @@DATEFIRST

You can set the first day of the week like this:

*-- Make Monday the first day of the week.*

SET DATEFIRST 1;

# Query to find upcoming birthdays for the current week

A common query for Human Resource databases or different social sites is to find out employee/subscribers name whose birthday comes in the near future (in the current week, or in next few days). To find out whose birthday is coming in a given number of days is very simple.

This example uses date data type instead of datetime to store the date of birth (DOB). This makes it a bit easier to handle the dates and compare them to current date.

The first query, which retrieves the workers whose birthday is in given days, can also be formulated as the following:

*--Create table variable to hold our test records*

DECLARE @Workers TABLE (WorkerName VARCHAR(50), DOB DATE);

*--Insert test records*

INSERT INTO @Workers

SELECT 'Alex', '1972-08-24' UNION ALL SELECT 'James', '1985-09-26' UNION ALL SELECT 'Mansha', '1983-08-25' UNION ALL SELECT 'Tara', '1991-09-24' UNION ALL SELECT 'William', '1992-08-19' UNION ALL SELECT 'Judy', '1989-09-15';

*--Variable to provide required number of days*

DECLARE @InNextDays INT; SET @InNextDays = 3;

*-- Query to find workers, whose birthday is in given number of days*

SELECT \*

FROM @Workers

WHERE DATEADD( Year, DATEPART( Year, GETDATE()) - DATEPART( Year, DOB), DOB) BETWEEN CONVERT( DATE, GETDATE())

AND CONVERT( DATE, GETDATE() + @InNextDays);

The idea is, that the actual birthday is shifted to current year by adding the number of years between the year of DOB and the current year. After that, a simple date range comparison is used.

The second query which retrieves the workers, whose birthday is on current week can be written like:

*-- Query to find workers, whose birthday is in current week*

SELECT \*

FROM @Workers

WHERE DATEPART( Week, DATEADD( Year, DATEPART( Year, GETDATE()) - DATEPART( Year, DOB), DOB))

= DATEPART( Week, GETDATE());

The basic idea is the same as in the first query I explained. Only this time the week numbers are compared.

Note: In the example, the default week numbering is used so depending on the requirements; the query may need to be adjusted to use different week numbers (for example ISO week). For more information, refer to [DATEPART.](http://msdn.microsoft.com/en-us/library/ms174420.aspx)

### --Create table variable to hold our test records

### DECLARE @Workers TABLE (WorkerName VARCHAR(50), DOB DATE);

### --Insert test records INSERT INTO @Workers

### SELECT 'Alex', '1972-08-24' UNION ALL

### SELECT 'James', '1985-09-26' UNION ALL SELECT 'Mansha', '1983-08-25' UNION ALL SELECT 'Tara', '1991-09-24' UNION ALL SELECT 'William', '1992-08-19' UNION ALL SELECT 'Judy', '1989-09-15';

### --Variable to provide required number of days DECLARE @InNextDays INT;

### SET @InNextDays = 3;

### -- Query to find workers, whose birthday is in given number of days SELECT \*

### FROM @Workers

### WHERE DATEADD( Year, DATEPART( Year, GETDATE()) - DATEPART( Year, DOB), DOB) BETWEEN CONVERT( DATE, GETDATE())

### AND CONVERT( DATE, GETDATE() + @InNextDays);

##### Can you now utilise and adapt the scenario to tackle some sorting of data on date? User Stories: Outstanding invoices more than 30 day? Demographics on age? Etc

# Conclusion

SQL and TSQL have rich support for dates, times, and dates with times. If you are working with dates, you should always store them as dates. You should not store time information unless it is relevant, and sometimes storing it can cause bugs ( for example, if you check a range between two dates using <= and >= then having a time component that defaults to midnight can cause your last date to be ignored, and the BETWEEN operator should be avoided for the same reason, if your values include a time component ).

Storing dates as strings is always the worst possible option, and any database you inherit that does this, should have fixing this as a significant priority, by adding a new column that uses the date type, keeping it up to date with a trigger, and retiring the old column over time.

It is always better to use standard SQL if you can, and to be aware when you are using propietary SQL, because you never know when your boss or a client may require you to work on another SQL platform.

While your code may always work by making assumptions that are valid most of the time, those sort of obscure bugs are the hardest to track down, because they tend to be intermittent, and so your code should be as defensive as possible.

I've not really covered the ways to get SQL Server to format your dates as strings, because I believe it's better to entrust data to your data layer, and formatting to your presentation layer. Asking SQL Server to format and convert your data, is a load best passed to the presentation layer.