DATEADD and DATEDIFF are better than CONVERTing to varchar. Both queries have the same execution plan, but execution plans are primarly about **data** access strategies and do not always reveal implicit costs involved in the CPU time taken to perform all the pieces. If both queries are run against a table with millions of rows, the CPU time using DateDiff can be close to 1/3rd of the Convert CPU time!

To see execution plans for queries:

set showplan\_text on   
GO

Both DATEADD and DATEDIFF will execute a CONVERT\_IMPLICIT.

Although the CONVERT solution is simpler and easier to read for some, it *is* slower. There is no need to cast back to datetime (this is implicitly done by the server). There is also no real need in the DateDiff method for DateAdd afterward as the integer result will also be implicitly converted back to datetime.

**SELECT CONVERT(varchar, MyDate, 101) FROM DatesTable**

  |--Compute Scalar(DEFINE:([Expr1004]=CONVERT(varchar(30),[TEST].[dbo].[DatesTable].[MyDate],101)))   
       |--Table Scan(OBJECT:([TEST].[dbo].[DatesTable]))

**SELECT DATEADD(dd, 0, DATEDIFF(dd, 0, MyDate)) FROM DatesTable**

  |--Compute Scalar(DEFINE:([Expr1004]=dateadd(day,(0),CONVERT\_IMPLICIT(datetime,datediff(day,'1900-01-01 00:00:00.000',CONVERT\_IMPLICIT(datetime,[TEST].[dbo].[DatesTable].[MyDate],0)),0))))   
       |--Table Scan(OBJECT:([TEST].[dbo].[DatesTable]))

Using FLOOR() as @digi suggested has performance closer to DateDiff, but is not recommended as casting the datetime data type to float and back does not always yield the original value.

Remember guys: Don't believe anyone. Look at the performance statistics, and test it yourself!

Be careful when you're testing your results. Selecting many rows to the client will hide the performance difference becauses it takes longer to send the rows over the network than it does to perform the calculations. So make sure that the work for all the rows is done by the server but there is no rowset sent to the client.

There seems to be confusion for some people about when cache optimization affects queries. Running two queries in the same batch or in separate batches has no effect on caching. So you can either expire the cache manually or simply run the queries back and forth multiple times. Any optimization for query #2 would also affect any subsequent queries, so throw out execution #1 if you like.

Here is [full test script and performance results](http://stackoverflow.com/questions/2775/whats-the-best-way-to-remove-the-time-portion-of-a-datetime-value-sql-server/3696991#3696991) that prove DateDiff is substantially faster than converting to varchar.

**What Is Really Best?**

I've seen inconsistent claims about what's fastest for truncating the time from a date in SQL Server, and some people even said they did testing, but my experience has been different. So let's do some more stringent testing and let everyone have the script so if I make any mistakes people can correct me.

**Float Conversions Are Not Accurate**

First, I would stay away from converting datetime to float, because it does not convert correctly. You may get away with doing the time-removal thing accurately, but I think it's a bad idea to use it because it implicitly communicates to developers that this is a safe operation and **it is not**. Take a look:

declare @d datetime;   
set @d = '2010-09-12 00:00:00.003';   
select Convert(datetime, Convert(float, @d));   
-- result: 2010-09-12 00:00:00.000 -- oops

This is not something we should be teaching people in our code or in our examples online.

Also, it is not even the fastest way!

**Proof – Performance Testing**

If you want to perform some tests yourself to see how the different methods really do stack up, then you'll need this setup script to run the tests farther down:

create table AllDay (Tm datetime NOT NULL CONSTRAINT PK\_AllDay PRIMARY KEY CLUSTERED);   
declare @d datetime;   
set @d = DateDiff(Day, 0, GetDate());   
insert AllDay select @d;   
while @@ROWCOUNT != 0   
   insert AllDay   
   select \* from (   
      select Tm =   
         DateAdd(ms, (select Max(DateDiff(ms, @d, Tm)) from AllDay) + 3, Tm)   
      from AllDay   
   ) X   
   where Tm < DateAdd(Day, 1, @d);   
exec sp\_spaceused AllDay;  -- 25,920,000 rows

Please note that this creates a 427.57 MB table in your database and will take something like 15-30 minutes to run. If your database is small and set to 10% growth it will take longer than if you size big enough first.

Now for the actual performance testing script. Please note that it's purposeful to not return rows back to the client as this is crazy expensive on 26 million rows and would hide the performance differences between the methods.

**Performance Results**

set statistics time on;   
-- (All queries are the same on io: logical reads 54712)   
GO   
declare   
    @dd date,   
    @d datetime,   
    @di int,   
    @df float,   
    @dv varchar(10);   
   
-- Round trip back to datetime   
select @d = CONVERT(date, Tm) from AllDay; -- CPU time = 21234 ms,  elapsed time = 22301 ms.   
select @d = CAST(Tm - 0.50000004 AS int) from AllDay; -- CPU = 23031 ms, elapsed = 24091 ms.   
select @d = DATEDIFF(DAY, 0, Tm) from AllDay; -- CPU = 23782 ms, elapsed = 24818 ms.   
select @d = FLOOR(CAST(Tm as float)) from AllDay; -- CPU = 36891 ms, elapsed = 38414 ms.   
select @d = CONVERT(VARCHAR(8), Tm, 112) from AllDay; -- CPU = 102984 ms, elapsed = 109897 ms.   
select @d = CONVERT(CHAR(8), Tm, 112) from AllDay; -- CPU = 103390 ms,  elapsed = 108236 ms.   
select @d = CONVERT(VARCHAR(10), Tm, 101) from AllDay; -- CPU = 123375 ms, elapsed = 135179 ms.   
   
-- Only to another type but not back   
select @dd = Tm from AllDay; -- CPU time = 19891 ms,  elapsed time = 20937 ms.   
select @di = CAST(Tm - 0.50000004 AS int) from AllDay; -- CPU = 21453 ms, elapsed = 23079 ms.   
select @di = DATEDIFF(DAY, 0, Tm) from AllDay; -- CPU = 23218 ms, elapsed = 24700 ms   
select @df = FLOOR(CAST(Tm as float)) from AllDay; -- CPU = 29312 ms, elapsed = 31101 ms.   
select @dv = CONVERT(VARCHAR(8), Tm, 112) from AllDay; -- CPU = 64016 ms, elapsed = 67815 ms.   
select @dv = CONVERT(CHAR(8), Tm, 112) from AllDay; -- CPU = 64297 ms,  elapsed = 67987 ms.   
select @dv = CONVERT(VARCHAR(10), Tm, 101) from AllDay; -- CPU = 65609 ms, elapsed = 68173 ms.   
GO   
set statistics time off;

**Some Rambling Analysis**

Some notes about this. First of all, if just performing a GROUP BY or a comparison, there's no need to convert back to datetime. So you can save some CPU by avoiding that, unless you need the final value for display purposes. You can even GROUP BY the unconverted value and put the conversion only in the SELECT clause:

select Convert(datetime, DateDiff(dd, 0, Tm))   
from (select '2010-09-12 00:00:00.003') X (Tm)   
group by DateDiff(dd, 0, Tm)

Also, see how the numeric conversions only take slightly more time to convert back to datetime, but the varchar conversion almost doubles? This reveals the portion of the CPU that is devoted to date calculation in the queries. There are parts of the CPU usage that don't involve date calculation, and this appears to be something close to 19875 ms in the above queries. Then the conversion takes some additional amount, so if there are two conversions that amount is used up approximately twice.

More examination reveals that compared to Convert(, 112), the Convert(, 101) query has some additional CPU expense (since it uses a longer varchar?), because the second conversion back to date doesn't cost as much as the initial conversion to varchar, but with Convert(, 112) it is closer to the same 20000 ms CPU base cost.

Here are those calculations on the CPU time that I used for the above analysis:

     method   round  single   base   
-----------  ------  ------  -----   
       date   21324   19891  18458   
        int   23031   21453  19875   
   datediff   23782   23218  22654   
      float   36891   29312  21733   
varchar-112  102984   64016  25048   
varchar-101  123375   65609   7843

* *round* is the CPU time for a round trip back to datetime.
* *single* is CPU time for a single conversion to the alternate data type (the one that has the side effect of removing the time portion).
* *base* is the calculation of subtracting from round the difference between the two invocations: single - (round - single). It's a ballpark figure that assumes the conversion to and from that data type and datetime is approximately the same in either direction. It appears this assumption is not perfect but is close because the values are all close to 20000 ms with only one exception.

One more interesting thing is that the base cost is nearly equal to the single Convert(date) method (which has to be almost 0 cost, as the server can internally extract the integer day portion right out of the first four bytes of the datetime data type).

**Conclusion**

So what it looks like is that the single-direction varchar conversion method takes about 1.8 μs and the single-direction DateDiff method takes about 0.18 μs. I'm basing this on the most conservative "base CPU" time in my testing of 18458 ms total for 25,920,000 rows, so 23218 ms / 25920000 = 0.18 μs. The apparent 10x improvement seems like a lot, but it is frankly pretty small until you are dealing with hundreds of thousands of rows (617k rows = 1 second savings).

Even given this small absolute improvement, in my opinion, the DateAdd method wins because it is the best combination of performance and clarity. The answer that requires a "magic number" of 0.50000004 is going to bite someone some day (five zeroes or six???), plus it's harder to understand.

**Additional Notes**

When I get some time I'm going to change 0.50000004 to '12:00:00.003' and see how it does. It is converted to the same datetime value and I find it much easier to remember.

For those interested, the above tests were run on a server where @@Version returns the following:

Microsoft SQL Server 2008 (RTM) - 10.0.1600.22 (Intel X86) Jul 9 2008 14:43:34 Copyright (c) 1988-2008 Microsoft Corporation Standard Edition on Windows NT 5.2 (Build 3790: Service Pack 2)

Critiques are welcome. I'm not out to prove MY way, I'm out to find the BEST way.

**Itzik Ben-Gan in** [**DATETIME Calculations, Part 1**](http://www.sqlmag.com/Article/ArticleID/94487) **(SQL Server Magazine, February 2007) shows three methods of performing such a conversion (slowest to fastest; the difference between second and third method is small):**

**SELECT CAST(CONVERT(char(8), GETDATE(), 112) AS datetime)   
   
SELECT DATEADD(day, DATEDIFF(day, 0, GETDATE()), 0)   
   
SELECT CAST(CAST(GETDATE() - 0.50000004 AS int) AS datetime)**

**Your technique (casting to float) is suggested by a reader in the April issue of the magazine. According to him, it has performance comparable to that of second technique presented above.**