**[4 Ways To Define Lookup Values In A Query](http://www.sqlservercentral.com/blogs/bert-wagner/2018/06/05/4-ways-to-define-lookup-values-in-a-query/)**

[Bert Wagner](http://www.sqlservercentral.com/blogs/bert-wagner/)

Posted on 5 June 2018

FROM: <http://www.sqlservercentral.com/blogs/bert-wagner/2018/06/05/4-ways-to-define-lookup-values-in-a-query/>

Occasionally you may need to create a lookup table of values for a single query.  Building a permanent table of these values is a good option if you need to reuse the values frequently, but for one-off, ad-hoc queries you may want to hard-code the lookup table of values directly in your query.

We can hard-code these values using several different techniques.  Below are the techniques I’ve seen used before, going in order from my least to most favorite.

**Table Variables**

DECLARE @T TABLE

(

ColorName varchar(20),

HexCode char(7)

);

INSERT INTO @T VALUES

('FireBrick','#B22222'),

('HotPink','#FF69B4'),

('Tomato','#FF6347'),

('PapayaWhip','#FFEFD5'),

('RebeccaPurple','#663399'),

('LawnGreen','#7CFC00'),

('MidnightBlue','#191970');

SELECT \* FROM @T;

Table variables get a bad rap in SQL Server because they don’t provide accurate row estimates to the query optimizer.  This can cause SQL Server to come up with some really terrible execution plans that will kill your query’s performance.

However, if your use case is a single small lookup table of less than 100 records, table variables might actually be a viable option.  They are quick and easy to setup and can be added to the top of your query.

With that said, I don’t think I’ve ever used a table variable in this type of scenario (or any scenario really).  I know some people love using them and I think that’s fine as long as you are keeping track of your query performance.  For me though, there are so many better options available…

**Temporary Tables**

CREATE TABLE #T

(

ColorName varchar(20),

HexCode char(7)

);

INSERT INTO #T VALUES

('FireBrick','#B22222'),

('HotPink','#FF69B4'),

('Tomato','#FF6347'),

('PapayaWhip','#FFEFD5'),

('RebeccaPurple','#663399'),

('LawnGreen','#7CFC00'),

('MidnightBlue','#191970');

SELECT \* FROM #T;

Temp tables are the answer to many of the table variable’s shortcomings.

Temp tables can perform well with larger amounts of data because they can be indexed and can have statistics generated on them.  Both of these features typically help SQL Server generate better execution plans.

There is some overhead in coding a temp table though: just like a table variable, a temp table needs to be created and inserted into before being able to use it in your query.  While normally not a huge deal, this is not something I want to have to do in those instances where I want to define some lookup values quickly…

**SELECT with UNION ALL**

SELECT

\*

FROM

(

SELECT 'FireBrick','#B22222' UNION ALL

SELECT 'HotPink','#FF69B4' UNION ALL

SELECT 'Tomato','#FF6347' UNION ALL

SELECT 'PapayaWhip','#FFEFD5' UNION ALL

SELECT 'RebeccaPurple','#663399' UNION ALL

SELECT 'LawnGreen','#7CFC00' UNION ALL

SELECT 'MidnightBlue','#191970'

) T(ColorName,HexCode);

The next option is hard-coding values in SELECT statements and then joining them together with UNION ALLs.

This is probably the most common technique I see, and for good reason: the syntax is straight forward and doesn’t require any special setup; perfect for the one-time use ad-hoc scenario.

Its format also makes it easy to use [the ALT + highlight shortcut](https://bertwagner.com/2017/11/28/multiline-edit-block-selection-alt-highlight-trick/) to quickly create a derived table lookup from the results of another query or values copied from elsewhere.

I do like this method a lot, but there is one method that I like slightly more…

**The VALUES() Constructor**

SELECT

\*

FROM

(VALUES

('FireBrick','#B22222'),

('HotPink','#FF69B4'),

('Tomato','#FF6347'),

('PapayaWhip','#FFEFD5'),

('RebeccaPurple','#663399'),

('LawnGreen','#7CFC00'),

('MidnightBlue','#191970')

) T(ColorName,HexCode);

You’ve probably use the [VALUES constructor](https://docs.microsoft.com/en-us/sql/t-sql/queries/table-value-constructor-transact-sql?view=sql-server-2017) in an INSERT INTO statement, but did you know you can also use it in a FROM clause?

This syntax is similar to our SELECT + UNION ALL technique above, except we are swapping in single quotes and parentheses for SELECTs and UNION ALLs.  I find this slightly easier to write, if only because it requires typing out fewer characters and feels more programmatic.

**One Last Tip: CTE**

WITH HtmlColors AS (

SELECT 'FireBrick' AS ColorName,'#B22222' AS HexCode UNION ALL

SELECT 'HotPink','#FF69B4' UNION ALL

SELECT 'Tomato','#FF6347' UNION ALL

SELECT 'PapayaWhip','#FFEFD5' UNION ALL

SELECT 'RebeccaPurple','#663399' UNION ALL

SELECT 'LawnGreen','#7CFC00' UNION ALL

SELECT 'MidnightBlue','#191970'

)

SELECT \* FROM HtmlColors

This isn’t really an additional technique, but something related that I use often and feels appropriate to mention.

If using either the SELECT + UNION ALL or VALUES techniques, you can put those lookup queries into a common table expression for easier referencing.

This doesn’t give any performance advantage, but it does help keep your code clean by putting your lookup logic right at the top of your file.  This becomes particularly useful when using those hard-coded lookup values as parameters, allowing all changes to your query to be made right at the top of your file during subsequent runs.

*Thanks for reading. You might also enjoy* [*following me on Twitter.*](https://twitter.com/bertwagner)