

Steven Tidwell

DYNAMIC SQL: Tables/Pivots/Parameters



Who Am I?

Steven Tidwell



- Technical Lead, Custom Client Database Development - Ctuit Software
- 20 + years in the IT World.
 - Started with SQL 7
 - I have designed and programmed two Point-Of-Sale systems.
 - PIMA POS™ - School Lunch Room Software
 - SALON POS™ - Beauty Salon scheduling POS System
 - Inventory Tracking Software for mobile service based companies



@SQLCODEMONKEY



Steven Tidwell



www.sqlcodemonkey.com



mybloggerplace.com

Thank You Sponsors!

Visit the Sponsor tables to enter their end of day raffles.

Turn in your completed Event Evaluation form at the end of the day in the Registration area to be entered in additional drawings.

Want more free training? Check out the **Houston Area SQL Server User Group** which meets on the 2nd Tuesday of each month. Details at <http://houston.sqlpass.org>



GOALS FOR TODAY

- Create a Dynamic Table at Runtime
- Create Dynamic Headers
- Create a Dynamic SQL Statement to present our data



WHY USE DYNAMIC SQL?

- Being successful in the world of database development and administration requires flexibility and the ability to adapt to ever changing technology. You will face many unknown challenges or situations where you might not know exactly what kind of data you will be working with at runtime.
- Dynamic SQL is one of the best tools for solving problems in the face of the unknown.
- Once you are acquainted with Dynamic SQL, the applications you can use it for are staggering. Dynamic SQL allows you to solve scenarios where you are presented with many objects, but you really do not know all the details of those objects.



PROS / CONS OF DYNAMIC SQL

PROS

- Reuse code for different tables or other objects
- Dynamically build tables
- Use variable names in statements that require constants
- Avoid statements that would be either impossible or very hard to code because of the high number of possibilities involved
- Return row sets with a variable number of columns and/or variable column names
- Allow parameterized filtering with the IN clause
- Sorting by any column from a table

CONS

- Performance loss: The execution plan for a dynamic query cannot be cached
- The error management becomes more unreliable. There is no easy way to validate the dynamic code or control its effects
- Temporary tables from the main statement cannot be used, unless they are global
- If the algorithm of the main statement has many loops, calculations or slow queries, that time will add to the time of executing the dynamic code
- Maintenance is difficult because the schema is hard coded in the dynamic code. The main statement is harder to understand than regular code because it is necessary to consider how it affects the dynamic code, without seeing it.



BUILDING A DYNAMIC QUERY

There are 3 main parts to building a DYNAMIC SQL Statement

- Declare a Variable to hold your Dynamic SQL Statement
DECLARE @MySQLVariable NVARCHAR(MAX)

WHY NVARCHAR?

You could use VARCHAR, however, you could potentially lose data if any extended UNICODE characters were in any of the objects you work with.

FOR CONSISTANCY USE NVARCHAR

- Build the Dynamic SQL command string and store it in the variable you just created:
SET @MySQLVariable = **N**'SELECT e.MyName FROM dbo.MyTable e'
- EXEC sp_ExecuteSQL @MySQLVariable

What does the N mean? *National Language Character Set*



BUILDING A DYNAMIC TABLE

We have all built a normal SQL Temp Table

```
CREATE TABLE #MySQLDemo  
(  
    ID INT IDENTITY(1,1) NOT NULL, Column1 VARCHAR(20), Column2 VARCHAR(20),  
    Column3 VARCHAR(20)  
)
```

BORING TABLE

But why do I want to create a Dynamic Table?

What happens if you want Dates to be your column names, and you don't know the dates that are going to be chosen when you build the table?

A Dynamic Table will allow you to pass in the name of the columns that you want in your table at runtime!

!! EXCITING !!



BUILDING A DYNAMIC TABLE

**STOP
DEMO TIME**



DEMO: Building Date Table

```
-- SECTION 0 - BUILDING A DATE TABLE --
IF OBJECT_ID('tempdb.dbo.#MyDates') IS NOT NULL
BEGIN
    DROP TABLE #MyDates
END
CREATE TABLE #MyDates
(
    DateID INT IDENTITY(1,1) NOT NULL, tDate SMALLDATETIME, DOW VARCHAR(13), ColumnHeader VARCHAR(40)
)

-- FILL THE DATE TABLE USING THE START AND END PARAMETERS --
INSERT INTO #MyDates
(tDate, DOW, ColumnHeader)
SELECT
TOP (DATEDIFF(DAY,@Start,@End) + 1)
"tDate" = DATEADD(DAY, ROW_NUMBER() OVER(ORDER BY a.OBJECT_ID) -1, @Start),
"DOW" = DATENAME(WEEKDAY, DATEADD(DAY, ROW_NUMBER() OVER(ORDER BY a.OBJECT_ID) -1, @Start)),
-- // THE COLUMN HEADER BUILT WILL BE THE SAME USED IN THE PIVOT TABLE \ \ --
"ColumnHeader" =
    CONCAT(DATENAME(WEEKDAY, DATEADD(DAY, ROW_NUMBER() OVER(ORDER BY a.OBJECT_ID) -1, @Start)), SPACE(5),
    FORMAT(DATEADD(DAY, ROW_NUMBER() OVER(ORDER BY a.OBJECT_ID) -1, @Start), 'MM/dd/yyyy'))
FROM
    sys.ALL_OBJECTS a
    CROSS JOIN sys.ALL_OBJECTS b;
```



DEMO: Building Base Table

```
-- SECTION 1 -- BUILDING BASE TABLE FOR DYNAMIC TABLE --  
IF OBJECT_ID('tempdb.dbo.#ItemSalesByDate') IS NOT NULL  
    BEGIN  
        DROP TABLE #ItemSalesByDate  
    END  
CREATE TABLE #ItemSalesByDate  
(  
    ItemID INT, ItemName VARCHAR(100)  
)
```



DEMO: Building Dynamic Table

```
}-- SECTION 1 -- BUILDING THE COLUMN STRUCTURE FOR THE DYNAMIC TABLE --  
-- // STEP 1 \\ --  
}DECLARE  
    @ColumnNameToAdd NVARCHAR(MAX), -- THIS HOLDS THE COLUMN NAMES CREATED IN THE DATE TABLE  
}    @DynamicSQL NVARCHAR(MAX) -- THIS HOLDS THE DYNAMIC SQL CODE
```

```
-- SECTION 1 -- SET THE PARAMETERS FOR THE TABLE  
-- // STEP 2 \\ --  
SET @DynamicSQL = N'ALTER TABLE #ItemSalesByDate ADD [?] MONEY;'  
SET @ColumnNameToAdd = ''
```

```
}SELECT  
    @ColumnNameToAdd = @ColumnNameToAdd + REPLACE(@DynamicSQL, '?', ColumnHeader)  
FROM  
    #MyDates  
-- // NOW EXECUTE THE DYNAMIC SQL \\ --  
-- // STEP 4 \\ --  
EXEC sp_ExecuteSQL @ColumnNameToAdd
```



A PLACE TO PUT YOUR STUFF

"That's all I want, that's all you need in life, is a little place for your stuff, ya know? "

- George Carlin

- **STUFF**: The **STUFF** function inserts a string into another string. It deletes a specified length of characters in the first string at the start position and then inserts the second string into the first string at that start position.
- When using Dynamic SQL, the **STUFF** command allows you to build a string of column headers. Then we can match those headers, with the headers in our table and **PIVOT** to **SUM** the values we need.

BUT HOW DOES STUFF WORK?



STUFF & XML: Living in Perfect Harmony

GET XML element string with FOR XML

- Adding FOR XML PATH to the end of a query allows us to output the results of the query as XML elements, with the element name contained in the PATH argument. For example, if we were to run the follow statement:

```
SELECT ', ' + ColumnName FROM #Temp1 FOR XML PATH('')
```

- By passing in a blank string (FOR XML PATH('')), we get the following results:
,aaa,bbb,ccc,ddd,eee

STOP: We have a leading , in our string!



STUFF & XML: Dropping the ,

Remove the leading comma with STUFF

- The STUFF statement literally “stuffs” one string into another, replacing characters within the first string. We, however, are using it simply to remove the first character of the resultant list of values.

```
SELECT abc = STUFF(( SELECT ', ' + ColumnName FROM #Temp1 FOR XML PATH('')),1,1,'') FROM #Temp1
```

The Parts of STUFF

- The string to be “stuffed” (in our case the full list of names with a leading comma)
- The location to start deleting and inserting characters (1, we’re stuffing into a blank string)
- The number of characters to delete (1, being the leading comma)

Now we end of with: **aaa,bbb,ccc,ddd,eee**



BUILDING HEADERS WITH STUFF & XML

**STOP
DEMO TIME**



BUILDING HEADERS WITH STUFF & XML

```
-- DEMO SECTION 2 - USING STUFF TO BUILD THE COLUMNS FOR THE PIVOT TABLE --  
DECLARE @PivotColumnName NVARCHAR(MAX)  
SET @PivotColumnName =  
    STUFF(  
        SELECT ',' + QUOTENAME(ColumnHeader)  
        FROM #MyDates md  
        GROUP BY ColumnHeader, DateID  
        ORDER BY DateID  
        FOR XML PATH(''), TYPE  
    ).value('.', 'NVARCHAR(MAX)')  
    ,1,1, '');
```



What have we learned so far?

- We have learned how to build a table and add columns to it at runtime
- Build custom column headers using STUFF and XML

**Now we will put it all together and finish
our DYNAMIC Query**



Goals: and fancy disclaimer

OUR GOALS:

- We will build a Dynamic Query to show the results of items sold by date.
- When we are finished, you can use this example to aggregate any data across a range of dates.

NOTE: The data we will be working with is taken directly from the PIMA POS™ system that I wrote. You can get a copy of the database used by visiting www.sqlcodemonkey.com

Disclaimer:

The tables have been altered to only have the relevant data needed for this demo. All names have been randomly generated using "Behind the Name" database plugin. The sales figures and data are actual items purchased during the time frame that this sample was taken.



Manipulating the Data

- We will be using 4 tables for our demo:
 - **Patron** – The Name and ID of the person purchasing the items
 - **Transaction Tables**
 - **Header** holds the top level information about the purchase
 - **Detail** holds the actual items that were purchased
 - **Item** – The Name and Item ID
- In order to get the data into a usable format, we must first manipulate the data to be in a layout that we can PIVOT on.
- We will build a “Pivot Column” that will be created in the **SAME** format as the Dynamic Column Headers that we created for our table.

NOTE:

IF THE COLUMNS **DO NOT MATCH**, YOU WILL NOT BE ABLE TO PIVOT THE DATA

GUESS WHAT?

IT IS DEMO TIME AGAIN!



DEMO: The data we are going to use

	PatronID	FirstName	LastName	isValid
1	1	Stephanie	T	1

	TransHeadID	PatronID	TotalSale	AmountTendered	SaleDate	SaleType	TransactionLane	Cashier	IsValid
1	1168	3	0.00	0.00	2010-08-10 10:03:00	NULL	99	3316	1

	TransactionDetailID	TransHeadID	ItemID	ItemPrice	DOB	IsValid
1	3799	1164	46	6.00	2010-08-09 10:01:00	1

	ItemID	ItemName	ItemPrice	isValid
1	64	.	0.00	1



Building the Output

Using what we learned earlier

- Declare our variable to hold the data
- Create the Select Statement
- Pivot the Data to get into Grid Form
- Do a final select on the data
 - *DO SELECT INSIDE THE DYNAMIC SQL FOR A TEMP TABLE*
 - *DO SELECT OUTSIDE THE DYNAMIC SQL FOR A GLOBAL TABLE*

**LET'S PUT IT ALL TOGETHER AND
BUILD THE CODE
DEMO TIME!**



DEMO: Building a Subset of All Data

```
-- INSERT THE SMALL SET OF DATA INTO TEMP TABLE --
INSERT INTO #SmallSet
    (ColumnHeader, ItemID, ItemName, Amount)
SELECT
    -- // NOTE: THE HEADER IS BUILT THE EXACT SAME WAS AS THE DATE TABLE COLUMN NAME \ \ --
    CONCAT(DATENAME(WEEKDAY, td.DOB), SPACE(5), FORMAT(td.DOB, 'MM/dd/yyyy')),
    i.ItemID, i.ItemName, td.ItemPrice
FROM
    dbo.Transaction_Header th
        INNER JOIN dbo.Transaction_Detail td ON td.TransHeadID = th.TransHeadID
        INNER JOIN dbo.Item i ON i.ItemID = td.ItemID
WHERE
    td.DOB BETWEEN @Start AND @End
    AND th.TotalSale IS NOT NULL
    AND th.IsValid <> 99
```



DEMO: The Dynamic Query

```
DECLARE @DynamicOutput NVARCHAR(MAX)

SET @DynamicOutput = N'
INSERT INTO #ItemSalesByDate
SELECT
    ItemId, ItemName, ' + @PivotColumnName + '
FROM
    #SmallSet s
PIVOT
    (SUM(AMOUNT) FOR ColumnHeader IN (' + @PivotColumnName + ')) AS PVT
ORDER BY
    ItemName

-- SINCE WE ARE INSERTING INTO A TEMP TABLE - AND WE NEED COLUMN HEADERS FOR THE OUTPUT --
-- WE MUST DO OUR SELECT STATEMENT IN THE DYNAMIC SQL --

SELECT
    ItemId, ItemName, ' + @PivotColumnName + '
FROM
    #ItemSalesByDate '

EXEC sp_ExecuteSQL @DynamicOutput
```



Getting rid of NULL Values

Why do we have NULL Values?

- A Null value might occur when there isn't any data for a particular item and date

How do we get rid of the NULL Values in the data?

- To get rid of NULL Values, we will create a new group of Headers with the ISNULL command in them.
- Replace the Header columns in the SELECT statement with the new ISNULL columns.
- Rerun the sample to replace NULL with 0



REMOVING NULL VALUES FROM OUTPUT

**STOP
DEMO TIME**



DEMO: Removing NULL Values with STUFF

```
DECLARE @IsNullColumns NVARCHAR(MAX)

SET @IsNullColumns =
    STUFF((
        SELECT ',ISNULL(' + QUOTENAME(ColumnHeader) + ',0)'
        FROM #MyDates
        GROUP BY ColumnHeader, DateID
        ORDER BY DateID
        FOR XML PATH(''), TYPE
    ).value('.', 'NVARCHAR(MAX)')
    ,1,1, '');
```



DEMO: Removing NULL Values with STUFF

```
DECLARE @DynamicOutput_NONULLS NVARCHAR(MAX)

SET @DynamicOutput_NONULLS = N'

INSERT INTO #ItemSalesByDate

SELECT
    ItemID, ItemName, ' + @IsNullColumns + '
FROM
    #Smallset s
PIVOT
    (SUM(AMOUNT) FOR ColumnHeader IN (' + @PivotColumnName + ')) AS PVT
ORDER BY
    ItemName

SELECT
    ItemID, ItemName, ' + @PivotColumnName + '
FROM
    #ItemSalesbyDate '

EXEC sp_ExecuteSQL @DynamicOutput_NONULLS
```



Passing Parameters to Dynamic SQL

Declaring Parameters

- With a standard SELECT statement, you can pass parameters to the query by declaring them and then assigning values to the parameter.

```
DECLARE @ItemID INT
SET @ItemID = 220

SELECT
    ItemID, ItemName, ItemPrice
FROM
    dbo.Item
WHERE
    ItemID = @ItemID
```



Passing Parameters to Dynamic SQL

Declaring Dynamic Parameters

- In Dynamic SQL, in order to use a parameter from the query, you must pass the parameter into your dynamic statement.

DEMO TIME



DEMO: Passing in Parameters

```
DECLARE @Params NVARCHAR(MAX)
```

```
SET @Params = N'@SearchString_IN VARCHAR(50)'
```



DEMO: Passing in Parameters

```
DECLARE @DynamicOutput_NONULL_SearchString NVARCHAR(MAX)

SET @DynamicOutput_NONULL_SearchString = N'

INSERT INTO #ItemSalesByDate

SELECT
    ItemID, ItemName, ' + @IsNullColumns + '
FROM
    #SmallSet s
PIVOT
    (SUM(AMOUNT) FOR ColumnHeader IN (' + @PivotColumnName + ')) AS PVT
WHERE
    ItemName = @SearchString_IN
ORDER BY
    ItemName

SELECT
    ItemID, ItemName, ' + @PivotColumnName + '
FROM
    #ItemSalesByDate '

EXEC sp_ExecutesQL @DynamicOutput_NONULL_SearchString, @Params, @SearchString_IN = @SearchString
```



DYNAMIC SQL: REAL WORLD USAGE

There are many places that you can use Dynamic SQL in your everyday career.

- Dynamic Search Strings
- Dynamic Sorting
- Running Scripts across databases
- Conditional Select Statements *



SELECT BASED ON PARAMETERS

```
USE SQLCodeMonkey
DECLARE
    @param1 INT = 1

IF @param1 = 1
BEGIN
    SELECT i.ItemID, i.ItemName
    FROM
        dbo.Item i
    END

IF @param1 = 2
BEGIN
    SELECT i.ItemID, i.ItemName
    FROM
        dbo.item i
    GROUP BY i.ItemID, i.ItemName
    END

IF @param1 = 3
BEGIN
    SELECT i.ItemID, i.ItemName
    FROM
        dbo.Item i
    ORDER BY i.ItemName DESC
    END
```



Building the SELECT Dynamically

```
DECLARE @param1 INT = 3

DECLARE @DynamicSQL NVARCHAR(MAX)

]SET @DynamicSQL = N'
    SELECT
        i.ItemID, i.ItemName
    FROM
        dbo.Item i '
-
]IF @param1 = 2
]    BEGIN
]        SET @DynamicSQL += N'
            GROUP BY i.ItemId, i.ItemName'
-
    END
-
]ELSE IF @param1 = 3
]    BEGIN
]        SET @DynamicSQL += N'
            ORDER BY i.ItemName DESC'
-
    END
-

EXEC sp_ExecuteSQL @DynamicSQL
```



Conclusion

There are hundred of other task that present themselves for Dynamic SQL

- Conditional Based Searches
- Conditional Based Select Statements

Be Creative

- When you think you have exhausted every possible solution for a problem, think how you can do it in Dynamic SQL

?? QUESTIONS ??

