**A Possible Alternative to SQL UNPIVOT**

**By** [**Paul Milligan**](http://www.sqlservercentral.com/Authors/Articles/Paul_Milligan/719251/)**, 2017/01/16**

**FROM:** [**http://www.sqlservercentral.com/articles/pivot/150905/**](http://www.sqlservercentral.com/articles/pivot/150905/)

**Introduction**

There can be situations where you wish to convert columns into rows from an SQL result set. We will use the term "flattening" for this. If the results of a query yield 3 columns of results, we want to convert this to 3 rows of 2 columns (let's call these "name/value" pairs).

**A Simple UNPIVOT**

Consider the following result set:

HoursInYear DaysInYear  MonthsInYear  
----------- ----------- ------------  
8760        365         12

We wish this result set to become:

Name         Value

------------ -----

HoursInYear  8760

DaysInYear   365

MonthsInYear 12

This is achievable using an UNPIVOT statement. So let's try give this a go first. Notice in the UNPIVOT at the bottom of the SQL I need to come up with 2 names for the columns, suitably called Name and Value. You can name these as you wish!

DECLARE @YearTable TABLE

( HoursInYear INT NOT NULL,

DaysInYear INT NOT NULL,

MonthsInYear INT NOT NULL

);

INSERT INTO @YearTable

VALUES

(

8760, 365, 12

);

SELECT

unpvt.Name,

unpvt.Value

FROM

(

SELECT

HoursInYear,

DaysInYear,

MonthsInYear

FROM @YearTable

) Src UNPIVOT(Value FOR Name IN(HoursInYear, DaysInYear, MonthsInYear)) AS unpvt;

So far so good, we now have our name value pairs.

**A More Complicated UNPIVOT**

Now consider an example where I do no necessarily know the column names in advance, particularly because a query is aggregating data. I am using simply table of products doing some aggregation and then presenting these as a list of name/value pairs.

Firstly we shall use the UNPIVOT statement to change the result set into rows. Then I will discuss an XML alternative that maybe useful in situations where you wish to flatten the result set without knowing what the column names will be in advance. Either way you will have a useful insight into how UNPIVOT works and some relatively simple XPath querying in SQL.

First, let us create and populate the Products table.

CREATE TABLE dbo.tblProducts

( ProductId INT NOT NULL IDENTITY(1001, 1),

ProductName VARCHAR(100) NOT NULL,

Price INT NOT NULL,

Suspended BIT NOT NULL

CONSTRAINT DF\_Products\_Suspended

DEFAULT (0),

CONSTRAINT PK\_Products\_ProductId

PRIMARY KEY (ProductId)

);

-- now some dummy data

INSERT INTO dbo.tblProducts(ProductName, Price)

VALUES

('Product1', 500),

('Product2', 0),

('Product3', 1000),

('Product4', 1500),

('Product5', 250),

('Product6', 0),

('Product7', 0),

('Product8', 2000),

('Product8', 2500),

('Product10', 500),

('Product11', 2100),

('Product12', 2200),

('Product13', 2300),

('Product14', 400);

Now, let us summarise the data.

SELECT

ProductsWithNoPrice = SUM( CASE

WHEN P.Price <= 0 THEN

1

ELSE

0

END

),

Products500OrLess = SUM( CASE

WHEN P.Price

BETWEEN 1 AND 500 THEN

1

ELSE

0

END

),

ProductsOver1500 = SUM( CASE

WHEN P.Price >= 1500 THEN

1

ELSE

0

END

)

FROM dbo.tblProducts P;

We now wish to flatten this data into name value pairs, so as before we can use UNPIVOT:.

Notice I've used 2 different names for the output fields; StatisticName and ProductCount.

SELECT

unpvt.StatisticName,

unpvt.ProductCount

FROM

(

SELECT

ProductsWithNoPrice = SUM( CASE

WHEN P.Price <= 0 THEN

1

ELSE

0

END

),

Products500OrLess = SUM( CASE

WHEN P.Price

BETWEEN 1 AND 500 THEN

1

ELSE

0

END

),

ProductsOver1500 = SUM( CASE

WHEN P.Price >= 1500 THEN

1

ELSE

0

END

)

FROM dbo.tblProducts P

) Src UNPIVOT(ProductCount FOR StatisticName IN(ProductsWithNoPrice, Products500OrLess, ProductsOver1500)) AS unpvt;

**An XML Alternative to the UNPIVOT**

Now suppose we want to add a new total to our statistics to simply count all products in the table. We need to add this column to our UNPIVOT. If we have several columns the UNPIVOT "IN" clause grows. Also if we were to rename columns the UNPIVOT breaks!

So I need a way to an equivalent without knowing any columns names in advance; one that will cope with breaking changes; and no dynamic SQL, read on.

A little know function exists in XPath query that we can use to extract the element name **fn:local-name(.)**. Also we can iterate through all the nodes in the XML simply using the wild card **\***.

First we need to get the data into XML, and then extract using the local-name function and wild card. I have also added a new statistic to the query called TotalCount.

DECLARE @ProductsXml XML;

SET @ProductsXml =

(

SELECT

ProductsWithNoPrice = SUM( CASE

WHEN P.Price <= 0 THEN

1

ELSE

0

END

),

Products500OrLess = SUM( CASE

WHEN P.Price

BETWEEN 1 AND 500 THEN

1

ELSE

0

END

),

ProductsOver1500 = SUM( CASE

WHEN P.Price >= 1500 THEN

1

ELSE

0

END

),

TotalCount = COUNT(\*)

FROM dbo.tblProducts P

FOR XML AUTO, ELEMENTS, TYPE

);

SELECT

StatisticName = T.c.value('fn:local-name(.)', 'varchar(100)'),

ProductCount = T.c.value('.', 'varchar(50)')

FROM @ProductsXml.nodes('//P/\*')T(c);

The result set from the above looks like:

StatisticName       ProductCount  
------------------- ------------  
ProductsWithNoPrice 3  
Products500OrLess   4  
ProductsOver1500    6  
TotalCount          14

As you can see when extracting the data I did not need to know the query names in advance. You could potentially create a function to do this and simply pass it XML either from a table or from a variable!

**Summary**

We learnt how to use UNPIVOT to flatten columns into rows of name/value pairs. Then we acheived the same result by converting the initial result set to XML and flatting this using a simple XPath query.