

Dynamic Pivot – Making Rows out of Columns

Erland Sommarskog
Data Platform MVP



Erland Sommarskog

Independent consultant based in Stockholm

SQL Server MVP since 2001

esquel@sommarskog.se

<http://www.sommarskog.se>

**Slides and scripts are available on
<http://www.sommarskog.se/present>**

What is Dynamic Pivot?

- Say that we have data like this, a typical result set from a query.
- But users want to see a matrix with countries as columns.
- If a new country appears in the data, a new column should appear.

Product	Country	Sales
Salmon	Germany	134 456
Salmon	France	156 789
Salmon	Brazil	87 964
Salmon	USA	55 781
Trout	Germany	654 112
Trout	France	520 111
Trout	Brazil	251 666

Product	Brazil	France	Germany	India	USA
Salmon	87 964	156 789	134 456		55 781
Trout	251 666	520 111	654 112	98 126	

Some General Considerations

- Dynamic pivot is a presentational device.
- Therefore, it may be best performed client-side.
 - Excel has had pivot for ages.
 - SSRS has the Tablix report.
 - Etc.
- However, there can still be sound reasons why we want to do this in SQL.
- Absolutely an asset to have in your SQL toolbox.

Dynamic Pivot is Not Relational

- Dynamic pivot breaks several of the general principles that a relational database is built on.
- Since we are breaking the rules, we need to work harder.
- We can write a query that transforms rows to columns for a fixed set of static values.
- To make it dynamic we have to construct such a query at run-time with dynamic SQL.

Overall Workflow

- First develop a normal tabular query that produces the data set you want, and where the rows are still rows.
 - Observation: In many cases this is a GROUP BY query with one or more aggregate functions.
- Then craft a static pivot query for a selected set of hard-coded values.
 - So that you know what you will build in the next step.
- Build a dynamic pivot with dynamic SQL.

Agenda for the Rest of the Session

- How to write a static pivot.
- Introduction to dynamic SQL.
- Constructing the dynamic pivot.

Note: We will not use the PIVOT operator in this session!

Static Pivot

[01 StaticPivot.sql](#)

- A pivot query always includes a GROUP BY and a number of aggregate functions with *CASE filters*.
- CASE filters take the form CASE-WHEN-THEN-END.
 - That is, CASE with a single WHEN-THEN and with no ELSE.
- Remember: aggregates always ignore NULL values.
- Your tabular query may have been without GROUP BY, but your pivot query still needs one.
- In this case, use MIN or MAX as the aggregate. Which, doesn't matter, as they only see one value.

Benefits of the CASE filters

- Compared to the PIVOT operator, queries with CASE filters in aggregates are longer and more repetitive.
- But the pattern is very flexible.
- Want a grand total or subtotals? Just add it!
- Want to pivot on multiple columns? Easy-peasy!
- With the PIVOT operator, you are in a straitjacket.

Prelude to Dynamic SQL

- Dynamic SQL is a very powerful feature – when used correctly.
- Dynamic SQL easily lends itself to abuse.
- Incorrectly used, dynamic SQL often leads to:
 - Security holes.
 - Performance issues.
 - Code that is about impossible to read and maintain.
- To successfully use dynamic SQL, you need a *structured* and *disciplined* approach.

sp_executesql

- A system stored procedure to execute dynamic SQL.
- Creates a nameless stored procedure and saves it in the cache and runs it, all in one go.
- If the exactly same query string is submitted again, the cached plan is reused.
- If any part of the string is changed, be that a space or whatever, it is a different query and a different nameless stored procedure.

sp_executesql, Parameters

- First parameter (@stmt): The SQL string to execute. Should always be nvarchar(MAX).
- Second parameter (@params): Parameter list for the SQL string. Same syntax as the parameter list of a normal stored procedure. Must be nvarchar. (But not MAX).
- Remaining parameters: actual values for the parameters defined by @params.

[02_sp_executesql.sql](#)

The @debug Parameter

- When you work with dynamic SQL, your code should *always* include these two lines:

```
IF @debug = 1  
    PRINT @sql;
```

- When your dynamic SQL does not behave as intended, you need to see what you have produced.
 - It is easier to read, if you have included line breaks.
- PRINT is limited to 4000 characters, so longer SQL strings will be truncated.
 - See [here for tips](#) on how to deal with this.

Bad Practice – Parameter Inlining

[02_sp_executesql.sql](#)

- Inlining parameter values rather than using proper parameterisation is bad for a number of reasons.
- Cache entries cannot be reused, which degrades overall server performance with more compilation.
- More difficult to get right:
 - Single quotes in data can cause errors.
 - Date/time and numbers requires explicit conversion to string.
 - Interpretation of date/time may depend on user settings.
- Biggest issue: opens for **SQL injection**.

Inlining and Dynamic Pivot

- You should always use parameters where the syntax permits it, never inline!
- ...but with dynamic pivot, avoiding inlining entirely is difficult.
- We will learn to how do inlining properly next.

Introducing QUOTENAME

[03_quotename.sql](#)

- QUOTENAME wraps a string in the given delimiter(s).
- Default delimiters are square brackets, [].
- Any closing delimiter in the string is doubled.
- You can also specify the single quote as delimiter.
- If input exceeds 128 characters, you get NULL back.

The Virtue of QUOTENAME

- When you have dynamic identifiers (tables, columns etc) always wrap them in QUOTENAME, to avoid:
 - Syntax errors when names have spaces etc.
 - And for that matter SQL injection.
- When inlining shorter strings, use QUOTENAME with single quote as delimiter, to avoid:
 - Syntax errors with single quotes in the string value.
 - SQL injection holes.
- For longer strings, you can use [quotestring](#).

A Dynamic Pivot Example

- Our task:

```
CREATE PROCEDURE ProductCountrySales_sp @fromdate date,  
                                          @todate   date,  
                                          @debug    bit = 0 AS
```

- Return sales per product and country for a period.
- Products as rows.
 - Only products with sales in the period.
- Countries as columns.
 - Only countries with sales in the period.
 - Sorted by continent, and countries alphabetically within continent.

The Six Steps of a Dynamic Pivot

1. Identify the values that define the dynamic columns and save to a temp table.
2. Create the initial part of the dynamic query.
3. Generate CASE-filtered aggregate calls with help of the temp table.
4. Optional: add any final columns.
5. Add the rest of the query: FROM, JOIN, WHERE etc.
6. Run the query.

Initial Skeleton

```
BEGIN TRY
    DECLARE @lineend nchar(3) = N', ' + NCHAR(13) + NCHAR(10),
            @sql      nvarchar(MAX);
    -- More code will be added as we go on.
    IF @debug = 1
        PRINT @sql;
    EXEC sp_executesql @sql, N'@fromdate date, @todate date',
                      @fromdate, @todate;
END TRY
BEGIN CATCH
    IF @@trancount > 0 ROLLBACK TRANSACTION;
    ; THROW;
END CATCH;
```

Variable
declarations

Running
the query
(Step 6)

Standard
error
handling.

Adding Step 5

- That is, FROM – JOIN – WHERE – GROUP BY – ORDER BY that defines the data set you are returning.
- You take this from the tabular query and static pivot you wrote initially.
- It is here you define table aliases to use in earlier steps.

Step 5

Note: Always an opening blank line to make debug output readable

```
SELECT @sql += N'  
    FROM    [Order Details] OD  
    JOIN    Products P      ON P.ProductID = OD.ProductID  
    JOIN    Orders O        ON OD.OrderID   = O.OrderID  
    JOIN    Customers C     ON C.CustomerID = O.CustomerID  
    WHERE   O.OrderDate BETWEEN @fromdate AND @todate  
    GROUP   BY P.ProductName  
    ORDER   BY P.ProductName';  
  
IF @debug = 1  
    PRINT @sql;
```


Step 1 – The #pivotcols Table

- This table typically has three columns:

```
CREATE TABLE #pivotcols
    (colno      int      NOT NULL PRIMARY KEY,
     filterval  char(2)  NOT NULL UNIQUE,
     colname    sysname  NOT NULL UNIQUE);
```

- All three are unique.
- **colno** – Defines the order of the pivot columns.
- **filterval** – The values to use in the CASE filters.
 - Data type depends on what you are pivoting by.
- **colname** – The names of the pivot columns.

More on #pivotcols

- **colno** typically set with the ROW_NUMBER function.
- You don't always need all three columns; sometimes you can coalesce two or all three into one.
- For more advanced scenarios, you may need help columns or multiple **filterval** or **colname** columns.
- But the archetype for #pivotcols are the these three columns: **colno**, **filterval** and **colname**.
- And now to the simple step 2:

```
SELECT @sql = N'SELECT P.ProductName ' + @lineend;
```

Step 3 – The Centrepiece

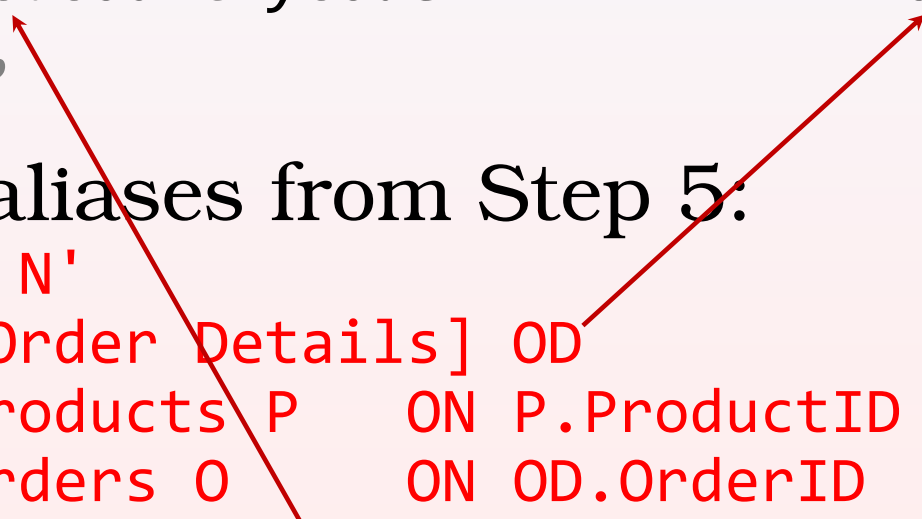
- We should generate several lines of this pattern:

```
SUM(CASE WHEN C.CountryCode = 'FR' THEN OD.Amount END)  
AS France,  
SUM(CASE WHEN C.CountryCode = 'DE' THEN OD.Amount END)  
AS Germany,
```

.....

- We have the aliases from Step 5:

```
SELECT @sql += N'  
FROM [Order Details] OD  
JOIN Products P ON P.ProductID = OD.ProductID  
JOIN Orders O ON OD.OrderID = O.OrderID  
JOIN Customers C ON C.CustomerID = O.CustomerID
```



Step 3 – Generate The Lines

- To generate the lines:

```
SELECT CONCAT(SPACE(7),
```

Indentation to
make debug
prettier.

First fixed
part.

```
N' SUM(CASE WHEN C.CountryCode = '
```

```
QUOTENAME(filterval, '''),
```

```
N' THEN OD.Amount
```

```
QUOTENAME(colname),
```

```
@lineend)
```

Always enclose
non-numeric
filter values in
quotes with
QUOTENAME!

Second
fixed part

```
FROM #pivotcols;
```

Comma and
line break

Always enclose
colname with
QUOTENAME

Concatenate Lines into One String

- For our dynamic pivot we need the rows in #pivotcols combined into a single string.
- To this end, use the STRING_AGG function.
- This is an aggregate function like SUM, COUNT etc.
- Takes two parameters:
 1. What expression to aggregate.
 2. Separator to have between the values.
- Use WITHIN GROUP to specify order of the values.

[05_String_agg.sql](#)

More on **STRING_AGG**

- Return type depends on input type.
- For dynamic pivot, you should always cast input to `nvarchar(MAX)` to avoid truncation errors.
- Available in SQL 2017 and later, as well as Azure SQL Database and Azure Managed Instance.
- We will look at alternative for older versions later.

Putting it All Together

```
SELECT @sql += STRING_AGG(  
    CAST(CONCAT(SPACE(7),  
        N' SUM(CASE WHEN C.CountryCode = ',  
        QUOTENAME(filterval, '''),  
        N' THEN OD.Amount END) AS '  
        QUOTENAME(colname))  
    AS nvarchar(MAX)),  
    @lineend)  
    WITHIN GROUP (ORDER BY colno)  
FROM #pivotcols;
```

Moved
@lineend to be
separator for
STRING_AGG

Convert to
nvarchar(MAX)

Use **colno**
for ordering.

[04_DynamicPivot.sql](#)

Instead of STRING_AGG

```
SELECT @sql +=  
  (SELECT CONCAT(SPACE(7),  
    N' SUM(CASE WHEN C.CountryCode = ',  
    QUOTENAME(filterval, '''),  
    N' THEN OD.Amount END) AS ',  
    QUOTENAME(colname),  
    @lineend)  
  FROM #pivotcols  
  ORDER BY colno  
  FOR XML PATH(''), TYPE).value('.', 'nvarchar(MAX)');  
SELECT @sql = SUBSTRING(@sql, 1, LEN(@sql) - LEN(@lineend));
```

No need to cast to
nvarchar(MAX)

This line
is always
the same.

@lineend
moves into
CONCAT.
Need to strip
last lineend

[04_DynamicPivot.sql](#)

Further Examples I

SalesPerDay_sp

- Show sales of products per day in an interval, with the dates as pivot columns.
- Grand totals per product, so step 4 is non-empty.
- #pivotcols has two columns, **filterval** and **colname**.
- **filterval** is date in an YYYYMMDD string for safe interpretation in SQL and also serves for sorting.
- **colname** is date in string format desired for presentation.

Further Examples II

ProductEmployeeSales_sp

- Display sales per employee in a period.
- Employees sorted by their total sales descending in the period.
- **filterval** is numeric, thus no QUOTENAME in step 3.
- Grand total per product *and* employee.
- The latter is achieved with the [GROUPING SETS](#) feature.

Further Examples III

ProductCountrySalesLevel_sp

- Display sales level from ProductCountrySales view.
- Display alphabetically by country, no continents.
- View has country name, not country code.
- #pivotcols has a single column, CountryName.
- Uses ISNULL to replace NULL values.
- Example also shows what happens when you don't use QUOTENAME.

Further Examples IV

ProductContinentSales_sp

- An extended version of ProductCountrySales_sp with subtotals per continent, as well as grand total.
- #pivotcols has a help column, **IsContinent**, to tell if a pivot column is a base column or subtotal.
- In Step 3, we filter by either country code or continent.

Further Examples V

ProductCountrySalesUnits_sp

- Display total sales as well as average units ordered per product and country.
- Each row in #pivotcols results in two columns in the final output.
- Thus, there are two columns for the column names.

Further Examples VI

[ProductEmployeeSalesPerDay_sp.](#)

- Displays daily sales per employee in a period.
- That is, we pivot both by date and employee.
- Thus, #pivotcols has two filter columns that are unique only in combination.

Summary – Overall Process

- Start with writing a normal relational query to make sure that you produce the correct data set.
- Turn that into a static pivot with some hard-coded values, so that you know what you are aiming for with your dynamic pivot.
- Write your dynamic pivot with steps 1 to 6. (But work with them in order 6-5-1-2-3-4.)

Summary – Dynamic SQL

- Requires discipline and a structured approach.
- Always, always, always have a @debug parameter to print the generated SQL.
- Always use parameterised statements.
- Exception: the filter values for dynamic pivot.
- Use QUOTENAME for filter values (unless numeric) and dynamic identifiers.

The Final Slide

Erland Sommarskog,
esquel@sommarskog.se

Slides and scripts on
<http://www.sommarskog.se/present>.

The Curse and Blessings of Dynamic SQL:
https://www.sommarskog.se/dynamic_sql.html