**Stored Procedures Assignment**

As an analyst, it is important to be able to retrieve and manipulate the data you need to support the decisions that need to be made. the dataThis assignment follows, the prior task of creating PIVOT() queries. Your task here is to turn each of the four PIVOT() queries from a prior assignment into parameterized stored procedures. You can consume the stored procedures in different reporting/analytical software:

a) [in Excel using PowerQuery](http://reevessmith.wordpress.com/2014/08/19/power-query-and-stored-procedures-with-parameters/)
b) [in an SSRS report](http://www.dotnetfunda.com/articles/show/2808/ssrs-demo-using-stored-procedure-with-parameters)
c) [in a Tableau report](http://kb.tableausoftware.com/articles/knowledgebase/using-stored-procedures-functions)
d) in an EXCEL using a Microsoft Query to an external data source (don’t waste any time with this methodology – power query is the update to this older Excel query methodology.

BE SURE TO PRINT AND READ THE “Turning your Query into a Stored Procedure” word processing document before you start this assignment. It will help you complete the assignment if you refer to the document as you work. The steps are tricky and the instructions have to be carefully followed.

A few problems due to system setup

1. We cannot create stored procedures in the group of AdventureWorks databases. You can however create a stored procedure in your own database (or use a database that was created for you) that can retrieve or manipulate data in AdventureWorks databases.
2. If you compare the two queries below – on the left is a nice hard-coded PIVOT() query and on the right that query is made dynamic by adding a parameter (a variable for country)
3. The variables are named using naming conventions. This document uses the standard str prefix for a string (text variable), an interger variable would have the prefix int (i.e. intCustomerID), a decimal variable would use dec (i.e. decTotal) and a date variable would have dt prefix as in (dtOrderDate). The naming convention for variables is personal and coding teams follow their own conventions.
4. You can start a query with a USE statement, but you must start a stored procedure in its own query batch. You can select your database in the database name selection box on the left hand side of the query editor, or you can get around this restriction to creating stored procedures as show in the right hand side (employ the USE statement but add a GO statement to execute that line and start a new query.
5. Since you are creating the query and resultant stored procedure in your database but you are querying another database, you have to refer to each table using the fully qualified name – [databsename].[dbo].[tablename].

As a review, it is best to only have three fields in the SELECT statement when performing a PIVOT query, the column to column for the rows, the column to define the columns and the column that will be aggregated. Using the current skillset you can only use a variable to filter the rows not the columns (you need to learn dynamic SQL to filter the columns using a variable-for now you can do this in SSRS dataset using a multi-select control). In the example below we could have added a parameter for the StateProvinceName column. Rather we filtered on Country which is interesting as it is not one of the three fields in the SELECT statement.

Compare the query on the left with the stored procedure on the right. In this assignment you are asked to take your four PIVOT queries from a previous assignment and turn each into a stored procedure. After you create the stored procedure, look into the Programmability folder of your database (in the object explorer) and into the stored procedures folder. Press F5 to refresh the database if you do not see the stored procedure.

*Notice on the right the USE statement refers to another database. Be sure to change this USE statement to refer to YOUR database.*

|  |  |
| --- | --- |
| USE AdventureWorksDW2012;SELECT \* FROM (SELECT Distinct [StateProvinceName] as State , pc.[EnglishProductCategoryName], [OrderQuantity]FROM [dbo].[DimGeography] as gINNER JOIN [dbo].[FactResellerSales] as rs ON rs.SalesTerritoryKey = g.SalesTerritoryKeyINNER JOIN [dbo].[DimProduct] as p ON p.ProductKey = rs.ProductKeyINNER JOIN [dbo].[DimProductSubcategory]as sc ON sc.ProductSubcategoryKey = p.ProductSubcategoryKeyINNER JOIN [dbo].[DimProductCategory] as pc ON pc.ProductCategoryKey = sc.ProductCategoryKeyWHERE g.[EnglishCountryRegionName] = 'United States' ) AS basetablePIVOT (SUM([OrderQuantity]) FOR [EnglishProductCategoryName]IN (Bikes, Components, Clothing, Accessories ))AS Pivottable | USE [Featherman\_dev];GOCREATE PROCEDURE [dbo].[spCityListForSpecifiedCountry3]@strCountry AS varchar(50)ASBEGINSELECT \* FROM (SELECT Distinct [StateProvinceName] as State, pc.[EnglishProductCategoryName], [OrderQuantity]FROM [AdventureWorksDW2012].[dbo].[DimGeography] as gINNER JOIN [AdventureWorksDW2012].[dbo].[FactResellerSales] as rs ON rs.SalesTerritoryKey = g.SalesTerritoryKeyINNER JOIN [AdventureWorksDW2012].[dbo].[DimProduct] as p ON p.ProductKey = rs.ProductKeyINNER JOIN [AdventureWorksDW2012].[dbo].[DimProductSubcategory]as sc ON sc.ProductSubcategoryKey = p.ProductSubcategoryKeyINNER JOIN [AdventureWorksDW2012].[dbo].[DimProductCategory] as pc ON pc.ProductCategoryKey = sc.ProductCategoryKeyWHERE g.[EnglishCountryRegionName] = @strCountry ) AS basetablePIVOT (SUM([OrderQuantity]) FOR [EnglishProductCategoryName]IN (Bikes, Components, Clothing, Accessories ))AS PivottableEND |

An accompanying document explains the method to create the stored procedure above (on the right hand side) however a quick comment is useful. Notice that the stored procedure is dynamic in that it uses a variable. You can easily use three data types discussed below: varchar, datetime and numeric.

In the sample query above the country must be passed in for the query to work, and the country must actually exist as a column value. The line above @strCountry AS varchar(50) is actually creating a variable of the datatype varchar that can store up to 50 letters or numbers. Varchar means the number of letters or numbers can be any number up to 50. The varchar datatype can store letters or numbers that don’t get calculated such as a phone number.

The @ symbol tells SQLServer that you have a variable, the name of the variable can be anything you can remember. Many programmers use the str prefix to help remember that the variable stores a string value – programmers refer to text as character strings so they use the prefix str. It’s an old convention from way back in the day, but it’s both common and optional.

Other common SQL datatypes are datetime, and numeric. Datetime can store any kind of date or time of day or date and time. A common usage might be @dtMonth AS datetime

The datatype numeric can be used to create numeric variables that store numbers. A common usage is @numRevenue AS numeric(10,2) which would allow 8 numbers to the left of the decimal point and two numbers to the right of the decimal point – for a total of 10 columns of data.

-------------------------------a few more notes, warnings and general instructions -----------------------------------

→ You can only use a variable to filter the rows. So for the IN() statement (defining the columns) below you can’t use an @variable. The columns are hard-coded. Rather the parameterization and variables used here, all are using the WHERE clause as shown below.

→ Again the following syntax formatting must be used. You need to create the stored procedure in your database. Also as shown above, use the fully qualified name of the database table in your query such as
[AdventureWorksDW2012].[dbo].[FactResellerSales]

USE [YourDatabaseName];
GO
CREATE PROCEDURE [dbo].[spTheQueryNamee]
@variable AS datatype
AS
BEGIN

→ When designing, testing and creating your stored procedure, watch which database you are connected to. You will see your stored procedures in the SSMS object explorer’s programmability |stored procedure folder. So create your stored procedures then run them. You can execute (run) your stored procedure in a new query window using the term Execute Procedure then the procedure name then the variable value such as a number or here a country name – for example
Execute Procedure [dbo].[spTheQueryName] ‘France’.

Similarly in the query below, after you turn it into a stored procedure, if you have a customer 212 and your stored procedure is called sp\_GetSalesForCustomer, then you can run the query using

EXECUTE PROCEDURE sp\_GetSalesForCustomer 212

SELECT \* FROM

( SELECT [Region], Year([orderdate]), [Total\_Sale]
FROM [dbo].[Sales] AS s
INNER JOIN [dbo].[Customers] as c ON c.customerid = s.customerid
WHERE c.customerid = @customer )

AS DataTable

PIVOT (

SUM([Total\_Sale])

FOR Year([orderdate]), IN ([2011], [2012], [2013])

) AS PivotTable

→ Before you begin again it would help if you use a SELECT DISTINCT query to build a handy list of countries, product categories, product sub-categories, and product colors (drop the NA). A recap of each PIVOT() table created and the modifications needed to turn the prior PIVOT() query into a powerful stored procedure are now presented:

Assignment Requirements – 4 stored procedures are requested

The first two queries below were made previously in a prior PIVOT() assignment you do not have to re-create them again. Just open those queries, test them with parameters and save them as stored procedures IN YOUR DATABASE using the procedure shown in the right-hand column above. The third query from the prior assignment needs a small change noted below. Only the 4th query below is new.

1. *SubCategory sales Across Countries* (filter by category) - Analysis of the units sold for each bicycle sub-category (either Internet or reseller channel) across countries. Turn this query into a stored procedure, with a variable to accept the product category (change the WHERE clause to a variable rather than the hard-coded ‘bikes’ )

|  |  |
| --- | --- |
| Change from | Change to |
| WHERE dimProductCategory.EnglishProductCategoryName = ‘bikes’ | WHERE dimProductCategory.EnglishProductCategoryName = @category |

So when the stored procedure is consumed, the program user will have to type in the product category (or select it from a list). The final output is a list of units sold by sub-category for each country ….*for whatever product category is defined.* The list of sub-categories will change depending on the category passed into the variable.

2. *Monthly Sales Across Countries* (filter by year) - Analysis of sales to each country across months. (you can more easily put the months in the rows, and the Countries in the columns). If you choose to place the months in the row headings going down the page, then the report can be referred to Analysis of monthly sales across country. The current report you have is for 2007 data. Change the query into a stored procedure that takes year as a variable that can be defined by the program user.

|  |  |
| --- | --- |
| Change from | Change to |
| WHERE YEAR(order\_date) = 2007 | WHERE YEAR(order\_date) = @year |

3. *Product Sales Across Colors* (filter by Country and ProductSubcategory) - Bicycle product analysis by each individual product (down the page) for one sub-category and color (Pivoted - across the page). For example all the different mountain bike sales can be shown within the mountain bikes product sub-category. This query is also a bit different than specified in the prior assignment. Add a variable to filter the unit sales by country. Add a second variable to filter based on product sub-category.

|  |  |
| --- | --- |
| Change from | Change to |
| WHERE Country = ‘[United States]’AND EnglishProductSubcategoryName = ‘Jerseys’ | WHERE Country = @country AND EnglishProductSubcategoryName = @SubCategory |

4. *SubCategory Sales Across Years* (filter by color) - Analysis of sub-category (down the rows) sales across years (Pivot on years). This is slightly different than the query created in the prior assignment. Here the sub-categories will be comingled, i.e. you may see bicycles mixed with clothing. Create a variable for color so that you can filter on color, which is in the products table. You will have to join in the products and product subcategories table.

|  |  |
| --- | --- |
| Change from | Change to |
| WHERE color = ‘red’ | WHERE color = @color |

**Turn-in** – After you receive the “Command Successfully completed” comment, then screen shot the query, pasting then cropping the screenshot into a MS-WORD document and add some commenting and documentation. You are building an IT asset that will be used by other people, a library of stored procedures that facilitate business operations, similar to an accounting procedure.

After you complete testing of your working stored procedure – write an explanation of how to use it, and what analysis it can be used to accomplish. (For example providing instructions how the stored procedure can be used, what variable and data type that needs to be supplied by the SSRS report, etc.).
After all four stored procedures are completed, add a cropped screenshot of your programmability folder with the stored procedures in it. Upload your MS-Word Document online.