Featherman’s T-SQL Adventures: Taking a peek ahead to the further use of variables

This document presents a few transitionary topics, the first is the use of variables, the second introduces the concept of passing in a parameter for a query (such as product subcategory) using a variable. Also you are shown how to deal with percentages. For example, what if we wanted to see the % of total sales each model accounted for. This first query could be your first attempt.

|  |  |
| --- | --- |
| USE [AdventureWorksDW2014];  SELECT [ModelName]  , SUM([OrderQuantity]) as [Total Units]  , (SELECT SUM([OrderQuantity]) FROM [dbo].[FactResellerSales] as s INNER JOIN [dbo].[DimProduct] as p ON s.ProductKey = p.ProductKey AND [ProductSubcategoryKey] IN (1,2,3)) as [Total]  FROM [dbo].[DimProduct] as p  INNER JOIN [dbo].[FactResellerSales] as s on s.ProductKey = p.ProductKey  WHERE [ProductSubcategoryKey] IN (1,2,3)  GROUP BY [ModelName] | This query is a good first step, we have data at two levels of granularity, the total for each model within three sub-categories. The second is the total for all three categories. |
| USE [AdventureWorksDW2014];  SELECT [ModelName]  , SUM([OrderQuantity]) as [Total Units]  , FORMAT(SUM([OrderQuantity]) /  --the denominator is a sub-query--------------  (SELECT SUM([OrderQuantity]) FROM [dbo].[FactResellerSales] as r  INNER JOIN [dbo].[DimProduct] as p  ON p.ProductKey = r.ProductKey  WHERE [ProductSubcategoryKey] IN (1,2,3)), 'P2' ) as [%]  --------------------------  FROM [dbo].[DimProduct] as p  INNER JOIN [dbo].[FactResellerSales] as s  ON s.ProductKey = p.ProductKey  WHERE [ProductSubcategoryKey] IN (1,2,3)  GROUP BY [ModelName] | Now we decide to change our sub-query to produce a % of total formula. We even wrap the formula (sum for model/Sum for selkected product categories) in a Format() function to display percentages.  But we are not getting the results wwe want. Why, SQL Why? It has to do with the formatting of the output.  Thank you William A. Chilton class of 2019! Apparently we need to make sure the number is defined as a decimal datatype when performing division, and the result is less than 1 (like a ratio or percent). |
| USE [AdventureWorksDW2014];    SELECT [ModelName]  , SUM([OrderQuantity]) as [Total Units]  , Format(CAST(SUM([OrderQuantity]) AS DECIMAL (12,2)) /  (  SELECT CAST(SUM([OrderQuantity]) AS DECIMAL(12,2)) FROM [dbo].[FactResellerSales] as r  INNER JOIN [dbo].[DimProduct] as p ON p.ProductKey = r.ProductKey  WHERE [ProductSubcategoryKey] IN (1,2,3)  ), 'P2') as [% of Total]  FROM [dbo].[DimProduct] as p  INNER JOIN [dbo].[FactResellerSales] as s  ON s.ProductKey = p.ProductKey  WHERE [ProductSubcategoryKey] IN (1,2,3)  GROUP BY [ModelName] |  |
| USE [AdventureWorksDW2014];  DECLARE @TotalUnits decimal(10,2) = 10  SET @TotalUnits = 15  SET @TotalUnits = (@TotalUnits \* 10) + 50  PRINT @TotalUnits | We are going to introduce the use of variables in SQL. Whenever you see the @ sign, the item is a variable. Let’s see how they work.  1) first you declare them (not DIM)  2) you do not need the term AS when creating a variable 3) you can declare the variable and give it a value in the same line 4) you can change the value of a variable using the SET term 5) when you see an = sign the values are assigned from right to left 6) the PRINT function displays the value of the variable |
| USE [AdventureWorksDW2014];  DECLARE @TotalUnits decimal(10,2)  SET @TotalUnits = (  SELECT SUM([OrderQuantity]) FROM [dbo].[FactResellerSales] as r  INNER JOIN [dbo].[DimProduct] as p ON p.ProductKey = r.ProductKey  WHERE [ProductSubcategoryKey] IN (1,2,3)  )  SELECT [ModelName]  , SUM([OrderQuantity]) as [Total Units]  , FORMAT(SUM([OrderQuantity]) / @TotalUnits, 'P2') as [%]  FROM [dbo].[DimProduct] as p  INNER JOIN [dbo].[FactResellerSales] as s  ON s.ProductKey = p.ProductKey  WHERE [ProductSubcategoryKey] IN (1,2,3)  GROUP BY [ModelName] | So now we use a variable to store the value from the sub-query. The first benefit is that the denominator of the formula is a lot easier to read.  When code gets too verbose (as our SQL queries have) it’s a good idea to segment them into parcels. So looking at the query on the left you can see that the formula to produce the % of total column is easier to read.  For some reason the output is formatted correctly now. If it wasn’t then we would use the CAST() as DECIMAL (12,2) |

|  |  |
| --- | --- |
| USE [AdventureWorksDW2014];  DECLARE @TotalUnits decimal(10,2) = (  SELECT SUM([OrderQuantity]) FROM [dbo].[FactResellerSales] as r  INNER JOIN [dbo].[DimProduct] as p ON p.ProductKey = r.ProductKey  WHERE [ProductSubcategoryKey] IN (1,2,3) )  DECLARE @Profit decimal(10,2) = (  SELECT SUM(([UnitPrice] \* [OrderQuantity]) - [DiscountAmount] ) FROM [dbo].[FactResellerSales] as r  INNER JOIN [dbo].[DimProduct] as p ON p.ProductKey = r.ProductKey  WHERE [ProductSubcategoryKey] IN (1,2,3)  )  SELECT [ModelName]  , SUM([OrderQuantity]) as [Total Units]  , FORMAT(SUM([OrderQuantity]) /  @TotalUnits, 'P2') as [% Units]  , FORMAT(SUM(([UnitPrice] \* [OrderQuantity]) - [DiscountAmount] ) / @Profit, 'P2') as [ % Profit]  FROM [dbo].[DimProduct] as p  INNER JOIN [dbo].[FactResellerSales] as s  ON s.ProductKey = p.ProductKey  WHERE [ProductSubcategoryKey] IN (1,2,3)  GROUP BY [ModelName]  *Be sure to remove the FORMAT() code before copying code into PowerBI* | This type of query becomes more doable when variables are used to segment the code.  Two variables are created each holding one value in memory. That value is used over and over in a calculation that is run for each row specified in the GROUP BY statement.  This analysis is interesting, for example the Touring-3000 model accounts for almost 7% of the units sold but only 3% of profit (before tax and shipping). Similarly the Road-650 is not driving profit and may even be robbing revenue from other road bikes.  On the other hand, the Mountain-200 accounts for 15% of units sold but brings in almost 22% of profits. As a manager what would you do? Copy the data in Excel and see what charts you can come up with. |

|  |  |
| --- | --- |
| *Analyst Challenge*  *Add a PowerBI slicer on Product Sub-category so that you can examine each bicycle sub-category more deeply.* | This chart does not show the model names but does identify that the models ‘above the line’ provide suitable profit for the # units that are sold. The higher a dot is above the line, the more it is driving profitability.   Any model that is below the line, account for numerical unit sales, but are not providing income commensurate with the # of units sold. If these models are priced as loss leaders to capture market share or drive foot traffit into stores, an anlysis should be made to see if the peripheral products sold with the bicycles provide the profit (similar to HP making money on ink not printers).  The models that ar below the line need to be evaluated as to why they are not driving revenue. As seen elsewhere perhaps the sales price was set incorrectly or too ambitiously for production to obtain. |
|  | |
|  | This chart shows the same data in a different manner. You can see that only 5 models are making profit commensurate with the number of units sold.  Why sell products that are not driving profits? Perhaps the product line should be constricted. |

|  |  |
| --- | --- |
| USE [AdventureWorksDW2012];  DECLARE @SubCategory int = 1  SELECT [ModelName], p.[ProductKey], [EnglishProductName]  , (SELECT ISNULL(SUM([OrderQuantity]), 0) FROM [dbo].[FactResellerSales] as s  WHERE p.[ProductKey] = s.[ProductKey] ) AS [Reseller Sales]  , (SELECT ISNULL(SUM([OrderQuantity]), 0) FROM [dbo].[FactInternetSales] as i  WHERE p.[ProductKey] = i.[ProductKey] ) AS [Internet Sales]  FROM [dbo].[DimProduct] as p  WHERE [FinishedGoodsFlag] =1  AND p.ProductSubcategoryKey = @SubCategory  ORDER BY [ModelName], [ProductKey] | This query foreshadows our usage of variables that serve to accept values passed in from other programs, and are used to filter the resultset.  When we create parameterized stored procedures, we will use variables to accept values that are passed into the query. While it is not useful here to set the value of the variable in code (subcategory = 1), in the future this subcategory value will be passed infr om Excel, SSRS, and SSMS. |