Featherman’s T-SQL Adventures - Categorizing Data with CASE(), NTILE(), RANK()  
  
Analysts are often putting data into categories to better analyze what the data is telling them. It may be counter intuitive, but you often have to put records into groups to better analyze them, this was the sentiment of the GROUP BY() query module. Here we use three more advanced SQL functions to put records into groups. CASE() statements allow yo uto put records into groups based on your rules. NTILE() breaks records into a set number of equally sized groups, and RANK() creates a column that ranks the records by your condition.

This document demonstrates these industry-standard techniques to categorize data. Some common examples of putting records into categories include identifying best/mid-level/worst customers, products, stores, employees, etc. We use the three SQL functions listed above to create new columns that have values that can be used to categorize records. The payback is that the new columns can be used as slicers in reports. A textual or numeric value is calculated and put into each column (for every row in the dataset) depending on what category the value falls within

So it’s a 2-step process. 1) create the new column of data and 2) write a query or report that uses that new column as the slicer. CASE() statements require more typing but give the developer complete control over the formation of groups. The processing of the other two functions covered in this document, NTILE() and RANK() is performed by SQL for you based on your specifications. We categorize rows of data to more easily filter and group them, prioritize them, and in-general facilitate further examination. The text or numeric columns created have less cardinality (such as 6 regions of the USA not 50 states) and thus can be used as dimensions on the X-axis in column charts or slicers, etc.

We will also use CASE processing to perform different functionality for different groups, and to protect our code from the dreaded divide by 0 fatal error. At first review, many students assess the functionality here as nice, but not earth-shattering. This potential initial reaction is based perhaps on lack of experience. CASE() statements allow you to create groups when you do not have a column in your dataset that can be used for the grouping (unlike a

Here is our first example. The CASE statement on the next page creates a new column named [Cost Category] (on the right) and places one of several values into it ($, $$, $$$, $$$$). Compare this [Cost Category] and the last line of the CASE statement *AS [new column name]*. In this instance the AS statement gives the column name. The purpose of this type of CASE statement is to categorize data. Later it is shown that it is possible to put records into groups such as quartiles, but here we write the logic to do the categorization. So we are just adding a new column with a word in it that indicates the category the record belongs to. This new column can then be used as a slicer. Seems simple but is very useful.

|  |  |
| --- | --- |
| USE [Featherman\_Analytics];  SELECT [Category], [Sub Category], [Model], [Product], [Cost]  , ([Cost] \* 1.25) as [25% Markup]  , ([Cost] \* 1.5) as [50% Markup], [Dealer Price]  , CASE  WHEN [COST] IS NULL THEN ' '  WHEN [Cost] BETWEEN 0 AND 99.99 THEN '$'  WHEN [Cost] BETWEEN 100 AND 299.99 THEN '$$'  WHEN [Cost] BETWEEN 300 AND 499.99 THEN '$$$'  WHEN [Cost] >= 500 THEN '$$$$'  END  AS [Cost Category]  FROM [dbo].[AW\_Products\_Flattened]  --WHERE [Dealer Price] < ([Cost] \* 1.25)  *Updates from WSU alumni indicate that analysts write a lot of CASE statements to add columns of descriptors (to put rows into categories), to create slicers that facilitate examining a sub-group of data.* | So here we put records into textual categories -- i.e., such as low, medium, high levels of a dimension. Another example is examining sales records and segmenting consumers into categories (i.e., infrequent customer, regular customer, and VIP customer). The example here puts a value into each row (either the symbol $, $$, $$$, or $$$$) for a new column based on ranges of values of the cost of a product (<100, >=100 & <300, >=300 & <500, and >=500.  The CASE syntax starts with a comma which lets you know that we are creating a new column, next we use the term CASE, set what to do for each condition, and the CASE statement ends with the term END. The AS term is used to supply a column name.  While there are many uses for a CASE statement such as to apply different discounting pattern, based on different levels of another variable (ie # the more days a product has been in stock on the shelf, the higher the level of discounting); the most common use of CASE statements is to create categories as shown here.  We can also perform this functionality in DAX using a SWITCH statement. The approach is similar, create a new column and place the textual group name in that new column based a categorization scheme such as ranges. |

|  |  |
| --- | --- |
| --This query examines the product catalog, comparing it to the prices that have been charged at the bike shops  USE [Featherman\_Analytics];  SELECT [Category], [Sub Category], [Model], [Product], FORMAT([Cost], 'N2') as [Cost], [Dealer Price]  , FORMAT(([Cost] \* 1.25), 'N2') as [25% Markup]  , FORMAT(([Cost] \* 1.4), 'N2') as [40% Markup]  , FORMAT(([Dealer Price] - [Cost]), 'N2') as [ProfitPerUnit]  , FORMAT(([Dealer Price] - [Cost])/[Cost], 'P2') as [Profit%]  , CASE  WHEN ([Dealer Price] - [Cost]) <0 THEN 'FUBAR'  WHEN ([Dealer Price] - [Cost]) BETWEEN .01 and 5 THEN 'pennies'  WHEN ([Dealer Price] - [Cost]) BETWEEN 5.01 and 9.99 THEN 'nickels'  WHEN ([Dealer Price] - [Cost]) BETWEEN 10.01 and 25 THEN 'dimes'  WHEN ([Dealer Price] - [Cost]) BETWEEN 25.01 and 100 THEN ‘quarters’  WHEN ([Dealer Price] - [Cost]) >100 THEN 'good profit'  END  AS [Cost Category]  FROM [dbo].[AW\_Products\_Flattened]  ORDER BY ([Dealer Price] - [Cost]) DESC  You are encouraged to copy the query into PowerBI and create some tables. after you paste the query into PowerBI  *BE SURE TO REMOVE THE FORMAT() statements and the COMMENTS*  The FORMAT statements turn the columns into text, that is useless for chart-making (well you can count the rows in text columns). The columns need to be numeric so that you can add or average the data in the columns | Here we categorize the dealer prices into groups based on to what extent the price charged to dealers covers the cost or the product.  The CASE statement is used to identify the products selling at lower than a 25% markup (which is probably just break-even). If you copy this SQL query, and then add a WHERE clause then you could retrieve a list of the products that are probably being sold to bike shops at too low a price.  This example should drive home the point about the usefulness of the technique. Each line in the WHEN statement can provide a different probe or test of the data and provide a different result in the column. Here we are still using categorical labels, we could also perform different calculations based on a test condition.  How does the functionality work? The values in each row are evaluated. The first When condition that equates to true will stop the evaluation of that row and the functionality after the THEN statement is performed (here different words are written into the column). The terms are used later as filters in a slicer or on the X-axis of a column chart.  Let’s not forget that *the SQL engine is examining every row of the dataset and providing the cost category.* |
|  | |
| Here we use a matrix report that leverages the profitability metric we created – which uses a CASE() statement to categorize the product models. We can see that the Road bikes have the most products that are categorized as FUBAR – which are for some reason sold to Bike shops below cost. You can create many different types of reports using the new dimension we created.   If an analyst creates a new category structure to better understand the data, then they can often drive insight, and management attention to problem areas. One of the insights provided by this query is that many products are sold to bike shops for just 1 or 3% profit. Sortof a waste of time. Either sell a better quality product or raise prices to at least cover a 20% profit (to better cover expenses and ensure business sustainability). | |

|  |  |
| --- | --- |
| USE [AdventureWorksDW2012];  SELECT [EnglishCountryRegionName], [City], [ResellerName]  , sum([OrderQuantity]) as [Total]  , CASE  WHEN sum([OrderQuantity]) IS NULL THEN ' '  WHEN sum([OrderQuantity]) BETWEEN 0 AND 99 THEN 'A: New Store'  WHEN sum([OrderQuantity]) BETWEEN 100 AND 299 THEN 'B: Emerging Store'  WHEN sum([OrderQuantity]) BETWEEN 300 AND 799 THEN 'C: Good Store'  WHEN sum([OrderQuantity]) BETWEEN 800 AND 2499 THEN 'D: Frequent Flyer'  WHEN sum([OrderQuantity]) >= 2500 THEN 'E: Elite Partner'  END  AS [Store Category]  ,RANK() OVER (ORDER BY sum([OrderQuantity]) DESC) AS Ranking  FROM [dbo].[DimReseller] as r  INNER JOIN [dbo].[FactResellerSales] as s ON s.[ResellerKey] = r.[ResellerKey]  INNER JOIN [dbo].[DimGeography] as g ON g.[GeographyKey] = r.[GeographyKey]  GROUP BY [EnglishCountryRegionName], [StateProvinceName],[City],[ResellerName] | Here we calculate the # units sold to each reseller and then use this total units metric to create a new category to differentiate within each WHEN category, and the THEN statement evaluates the calculated sum. Note the BETWEEN syntax is very useful - but is hard-coded and cannot vary with time and new definitions of what a good store is. A future example will allow the groups to be formed using a formula such as average +25% or +50% to create groups. Use relational operators > or < only at the top end and bottom end.  The powerful RANK() function is also used and assigns a rank to each record. The ranking is provided and the data is sorted on the ranking column. No need to belabor the syntax but the ranking is being done OVER the entire dataset, based on adding up the order quantities. |
|  | |

As you can see the SQL done well, provides the data in the right format for data visualization. Here maps in Power BI are used to visualize the data. The map can also drill down from state to city. The slicer on Country is especially helpful to clean up the presentation (show only certain parts of the globe at one time). Run the query and slice on store category (the result of the CASE field) Which stores are the elite partners?

|  |
| --- |
|  |

IF you modify the prior SQL query to add a second column of numbers then you can sort the textual column by the number column and the columns will be sorted correctly in the charts and the categories do not have to be prefaced A, B, etc.

, CASE   
WHEN sum([OrderQuantity]) IS NULL THEN 0  
WHEN sum([OrderQuantity]) BETWEEN 0 AND 99 THEN 1  
WHEN sum([OrderQuantity]) BETWEEN 100 AND 299 THEN 2  
WHEN sum([OrderQuantity]) BETWEEN 300 AND 799 THEN 3  
WHEN sum([OrderQuantity]) BETWEEN 800 AND 2499 THEN 4  
WHEN sum([OrderQuantity]) >= 2500 THEN 5

END AS [Store Category#]

, CASE   
WHEN sum([OrderQuantity]) IS NULL THEN ' '  
WHEN sum([OrderQuantity]) BETWEEN 0 AND 99 THEN 'New Store'  
WHEN sum([OrderQuantity]) BETWEEN 100 AND 299 THEN 'Emerging Store'  
WHEN sum([OrderQuantity]) BETWEEN 300 AND 799 THEN 'Good Store'  
WHEN sum([OrderQuantity]) BETWEEN 800 AND 2499 THEN 'Frequent Flyer'  
WHEN sum([OrderQuantity]) >= 2500 THEN 'Elite Partner'

END

AS [Store Category]

Actually CASE() statements are very powerful and can be used in 100 different ways. Did you notice however that you have to specify the ranges (ie BETWEEN 200 and 299.99. Specifying the numeric ranges gives you 100% control, but sometimes you just want to put records into groups such as terciles or quartiles based on some value.

*You can use NTILE() to quickly break your records into groups of roughly equal size. For example an NTILE(5) would create five ‘bins’ each of which would hold roughly 20% of the records. By default a new column is made with a number in it to signify which group (ie 1 = lowest, 5 = highest). . It is very easy for your reporting software to read the number in the cell and render it as an indicant (green, yellow, red stop light graphic for example).You can also place categorizing words in each row as shown in the following example.* NTILE() is used next to assign resellers into quartile groups.

|  |  |
| --- | --- |
| /\*This is a GROUP BY query by year and month, which provides a metric that categorizes  the months into four quartiles by revenue rating, and profit rating. If you look at the  data you will see that some high revenue months were not very profitable, and even lost $$  \*/  USE [Featherman\_Analytics];  SELECT (Year([OrderDate]) \*100) + MONTH([OrderDate]) as [YearMonth]  , FORMAT(SUM([SalesAmount]), 'N0') as [Total Revenue]  --here is the quartile for revenue, calculated twice, first for the number, then for a slicer  , NTILE(4) OVER (ORDER BY SUM(SalesAmount) ) as [Sales Quartile]  , CASE NTILE(4) OVER (ORDER BY SUM(SalesAmount))  WHEN 1 THEN 'Worst month'  WHEN 2 THEN 'Sub-par month'  WHEN 3 THEN 'Good month'  WHEN 4 THEN 'Top month'  END AS [Revenue Rating]  --here is the quartile for revenue, calculated twice, first for the number, then for a slicer  , FORMAT(SUM([SalesAmount]) - SUM([TotalProductCost]), 'N0') as [Profit]  , NTILE(4) OVER (ORDER BY SUM([SalesAmount]) - SUM([TotalProductCost])) as [Profit Quartile]  , CASE NTILE(4) OVER (ORDER BY SUM([SalesAmount]) - SUM([TotalProductCost]) )  WHEN 1 THEN 'worst month'  WHEN 2 THEN 'sub-par month'  WHEN 3 THEN 'break-even month'  WHEN 4 THEN 'best month'  END AS [Profit Rating]  FROM [AdventureWorksDW2014].[dbo].[FactResellerSales] as s  INNER JOIN [Featherman\_Analytics].[dbo].[AW\_Products\_Flattened] as p  ON s.ProductKey = p.ProductKey  GROUP BY (Year([OrderDate]) \*100) + MONTH([OrderDate])  ORDER BY [YearMonth] | The NTILE() places each row into a group based on the ORDER BY statement. If you place the NTILE inside a CASE statement then you can place words inside the column rather than numbers. While numbers are useful for drawing charts and stoplight KPI’s, the words are useful in a slicer or bar chart.  The CASE() statement then amplifies the usefulness of the NTILE() command. |
|  |  |

|  |  |
| --- | --- |
| USE Featherman\_Analytics;  SELECT CustomerName As [Customer Name]  */\*this next field is a measure that does not need a SUM()*  *function because the data is already summarized \*/*  , NumberSales As [# Sales]  , FORMAT(TotalSales, 'N0') AS [Total $ales]  */\* this line of code generates the dreaded divide by xero error,*  *(TotalSales/ [NumberSales]) as [test] \*/*  , FORMAT(CASE  WHEN [NumberSales] IS NULL THEN 0  WHEN [NumberSales] = 0 THEN 0  WHEN [NumberSales] > 0 THEN (TotalSales/ [NumberSales])  END, 'N0') as [Average Order $]  , CONVERT(varchar(10),[LastPurchase], 101) AS [Last Sale Was]  , CASE *-- this next code replaces the word NULL with a 0*  WHEN [LastPurchase] IS NULL THEN 0  WHEN [LastPurchase] IS NOT NULL   THEN DATEDIFF(DAY, [LastPurchase], GETDATE())  END AS [Days Ago]  , CASE  WHEN [LastPurchase] IS NULL THEN ''  WHEN DATEDIFF(DAY, [LastPurchase], GETDATE()) BETWEEN 0 AND 30  THEN 'Great'  WHEN DATEDIFF(DAY, [LastPurchase], GETDATE()) BETWEEN 31 AND 60  THEN 'Recent'  WHEN DATEDIFF(DAY, [LastPurchase], GETDATE()) BETWEEN 61 AND 120  THEN 'Few months'  WHEN DATEDIFF(DAY, [LastPurchase], GETDATE()) BETWEEN 121 AND 360  THEN 'Most of a year'  WHEN DATEDIFF(DAY, [LastPurchase], GETDATE()) > 360  THEN 'More than a year'  END as [Rating]  FROM Featherman.Customers  ORDER BY CustomerName  For comparison purposes, in this case the **SQL IIF** isn't useful because  it only tests for one problem where we have two potential problems (NULLS and 0's)  ,IIF([NumberSales] = 0 OR [NumberSales] IS NULL  , 0  , TotalSales/ [NumberSales])  as [Average Order $]  The code in the red box is the test condition, the blue box is the value to use when the test condition equates to true  , the last part of the function is the code to run if the test condition is false.  **Recall that this evaluation is needed for each and every row in the dataset.** | This query solves a few recurring problems. The first is that NULLS will creep into your datasets, if the system that is capturing the data allows it. The green code to the left is fragile in that the [NumberSales] column in the data entry system allowed nulls. The SQL formula that performs a division is fragile in that if the the denominator is zero or NULL (empty cell) then the query crashes.  The CASE () statement is the insurance policy to catch these data errors, the useful IS NULL test is employed such that when NULLS are found in the [NumberSales] column then just place a 0 in the column as the answer. Only when the [NumberSales] column has a value then perform the division and calculate the average. Looks like two more columns could use this NULL catching.  The CONVERT(varchar(10),[LastPurchase], 101) is useful to remember as it helps in formatting data. If you are going to use the data in a reporting software then do not use this as you are changing the datatype from something useful (Datetime) to less useful but nicely formatted varchar value and applying a nice date formatting. The code for the date is 101, there are many codes for different formatting for different regions.   The DATEDIFF() is great to calculate how many days ago a value in a column was. The GETDATE() captures the current date to calculate the # days ago.  /\*notice that the formula is based on the original numeric field not the calculated and formatted column \*/  WHEN [NumberSales] > 0  THEN (TotalSales/ [NumberSales])  NOTE: Three fields allow null values in the column. NULLS can easily crash your code, so learning how to control for this potential problem is crucial. IF you can, change the data entry procedure to require a default value such as a 0 to change NULLS to 0. |

***Improving a PIVOT() query using CASE statements to Organize the data and reduce Cardinality***

When you are tasked to produce a report, you often do not have all the dimensions you want at different grouping levels. This example uses CASE() statements in a brute force methodology to enable analysis at a different level of granularity (not state, but region). In the available dataset there there was no region field, the needed attributes are often missing. You will have to use SQL CASE() to make your own categories). Remember that you have the power to make groups in yoru data.

|  |  |
| --- | --- |
| USE [AdventureWorksDW2014];  SELECT \* FROM  (SELECT  *--Here put the states into region groups this is for the rows down the page*  CASE  WHEN [StateProvinceName] IN ('Connecticut','Maine','Massachusetts','New Hampshire' , 'Maryland', 'New York','Rhode Island') THEN 'Northeast'  WHEN [StateProvinceName] IN ('Alabama','Florida','Georgia','Kentucky','Mississippi' ,'North Carolina','South Carolina', 'Tennessee', 'Virginia') THEN 'South'  WHEN [StateProvinceName] IN ('Illinois','Indiana','Michigan','Minnesota','Missouri' ,'Ohio','South Dakota', 'Texas', 'Wisconsin') THEN 'Central'  WHEN [StateProvinceName] IN ('Colorado','Idaho','Montana','Oregon','Utah' ,'Washington','Wyoming') THEN 'Northwest'  WHEN [StateProvinceName] IN ('Arizona','California','Nevada','New Mexico')  THEN 'Southwest'  ELSE 'Uncategorized' –add this line to change the NULL In the output below to this label  END AS [US Region]  *-- Here is the column for the columns across the page, the PIVOT cuts the dimension into columns*  , DATENAME(MONTH,OrderDate) AS [MonthName], [SalesAmount]  FROM [dbo].[FactResellerSales] as s  INNER JOIN [dbo].[DimReseller] as r ON r.ResellerKey = s.ResellerKey  INNER JOIN [dbo].[DimGeography] as g ON g.GeographyKey = r.GeographyKey  ) AS Array  PIVOT  (SUM([SalesAmount])  FOR [MonthName]  IN(January, February, March, April, May, June, July, August, September, October, November, December)  ) AS PivotTable | Case Statements can be used inside a PIVOT query. Case statements allow you to make new columns of data, and here because it is the first column specified in the base query, then the column createsd is used for the rows.  When making PIVOT queries, you query on three fields.  1. In the first column you specify the values to use for the rows going down the page. 2. The second column of the SELECT statement specifies the values used to build the columns, here a date that is re-formatted into months. 3. The third value is the transaction level measure that is to be aggregated. |
|  | |
|  | |
| The SQL data output is actually more limited when the data is in a table created by a PIVOT query. PIVOT() are great for building reports, but if you want a hierarchyThere is less chance for different levels of aggregation such as city/state/region. You can however make a hierarchy in  A lot of government statistical data is released in tables like shown on the left. It is a pain to receive data in that format, as you have to reduce the columns and make more rows.  PowerBI has a nice menu selection to unpivot data. Try it out! | |

Join from PC, Mac, Linux, iOS, or Android: https://wsu.zoom.us/j/5161630528