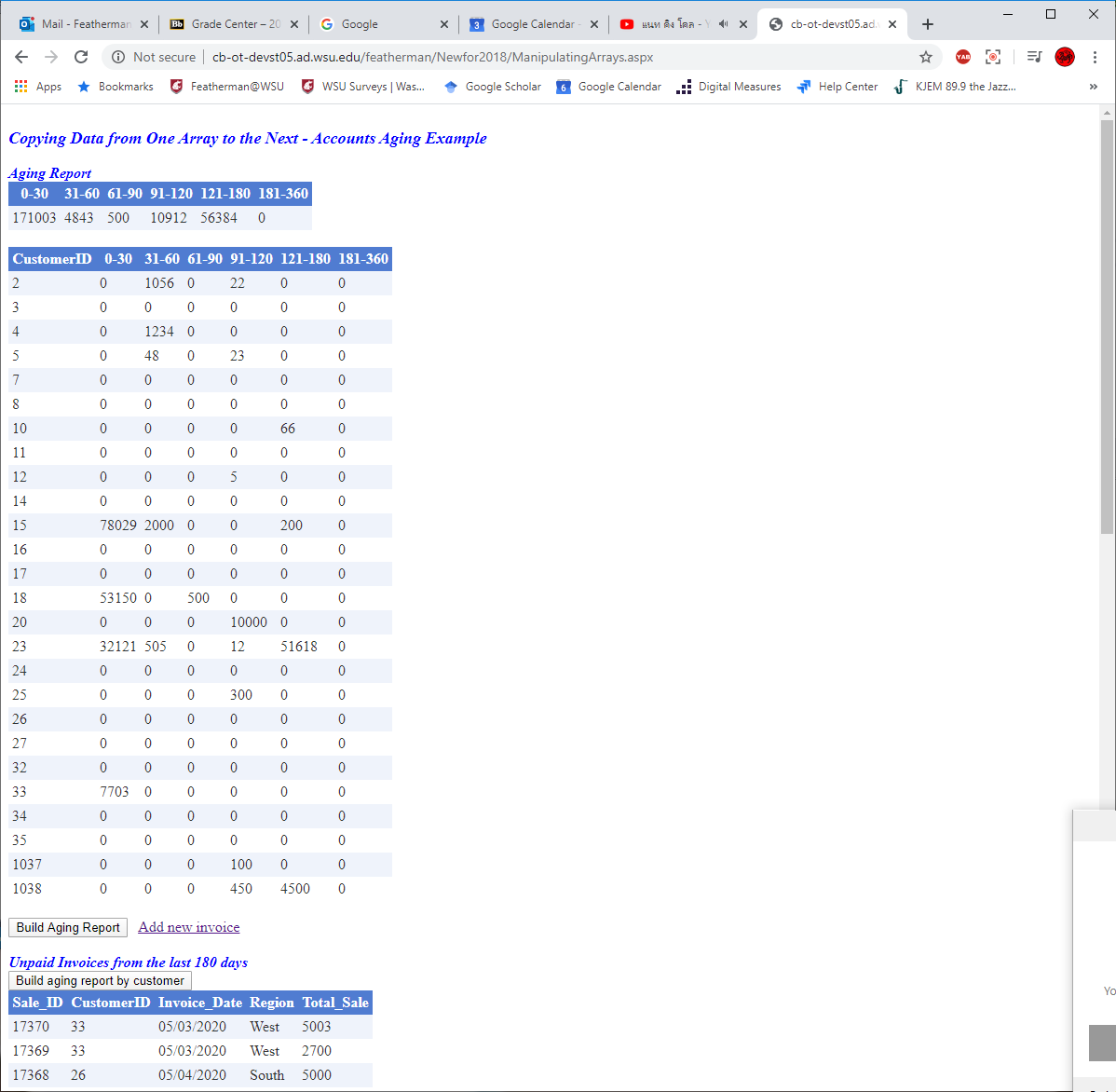
More on Sub-Queries– Building new interesting datasets  
*Sub-query example for the accountants: The Aging Report*

If a company allows credit sales, then they must monitor and control accounts receivable from customers. The table of sales transactions such as shown on the right needs to be turned into a summary Aging report such as here.

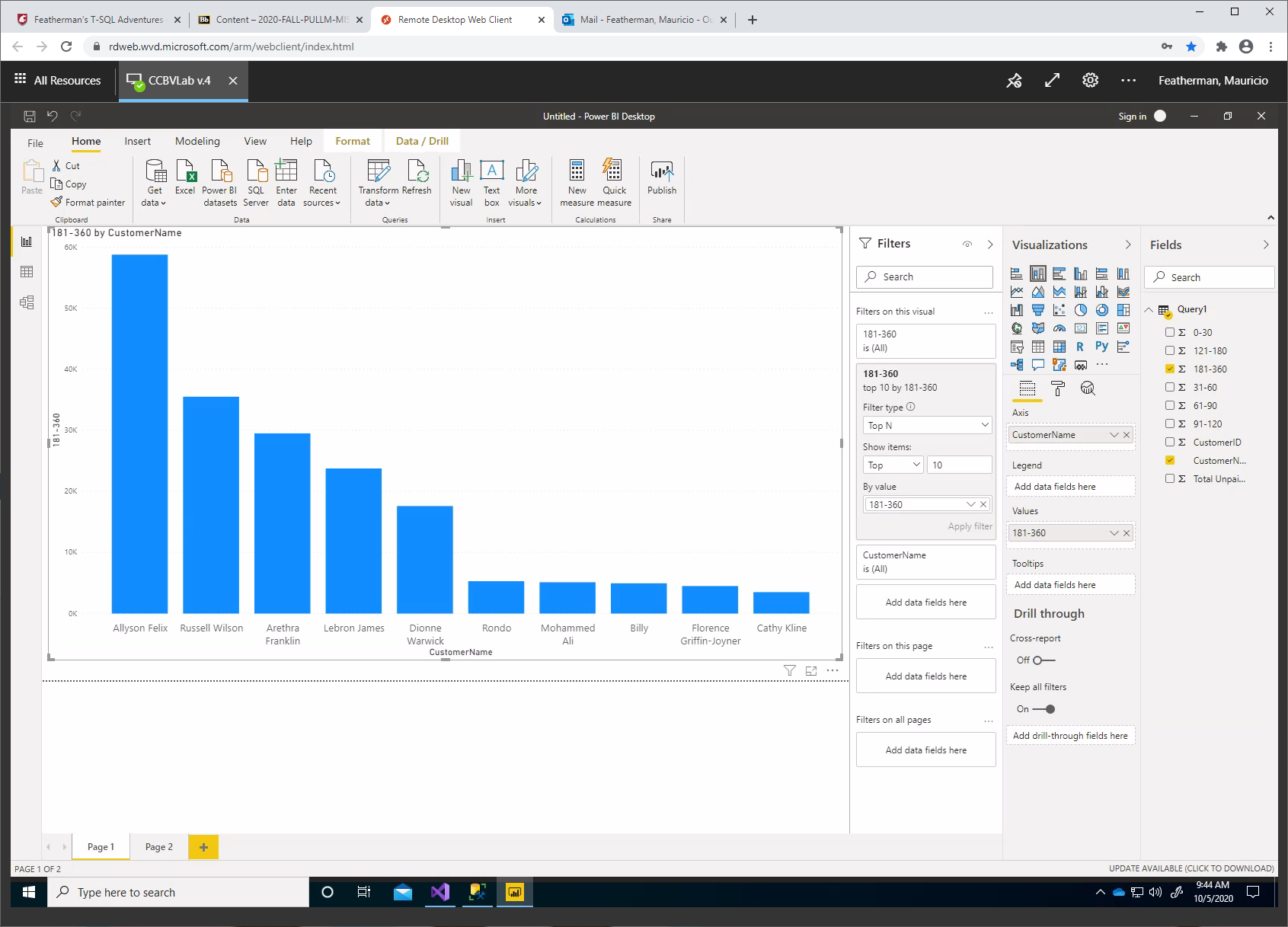
This example demonstrates an even more useful detailed aging report, with one row for each customer. A similar sub-query is used to build each calculated column.

If you run this sub-query in your SSMS, you will learn how it works. This sub-query takes massive number of transaction rows of data and compiles it quickly into the time buckets that are used in United States accounting systems. The table to the right is as of this writing 2427 lines of data. It all needs to be examined and categorized without error.

|  |  |
| --- | --- |
| USE Featherman\_Analytics;  SELECT *[CustomerID], [CustomerName]*  , ISNULL(  (  SELECT SUM([Total\_Sale])  FROM [featherman].[Sales] as s  WHERE [Paid] = 0 AND (GETDATE() - [Invoice\_Date])  BETWEEN 0 AND 30  AND c.[CustomerID] = s.CustomerID  )  , 0) as [0-30]  , ISNULL(  (SELECT SUM([Total\_Sale])  FROM [featherman].[Sales] as s  WHERE [Paid] = 0 AND (GETDATE() - [Invoice\_Date])  BETWEEN 31 AND 60  AND c.[CustomerID] = s.CustomerID  )  , 0) as [31-60]  , ISNULL((SELECT SUM([Total\_Sale])  FROM [featherman].[Sales] as s  WHERE [Paid] = 0 AND (GETDATE() - [Invoice\_Date])  BETWEEN 61 AND 90  AND c.[CustomerID] = s.CustomerID), 0) as [61-90]  , ISNULL((SELECT SUM([Total\_Sale])  FROM [featherman].[Sales] as s  WHERE [Paid] = 0 AND (GETDATE() - [Invoice\_Date])  BETWEEN 91 AND 120  AND c.[CustomerID] = s.CustomerID), 0) as [91-120]  , ISNULL((SELECT SUM([Total\_Sale])  FROM [featherman].[Sales] as s  WHERE [Paid] = 0 AND (GETDATE() - [Invoice\_Date])  BETWEEN 121 AND 180  AND c.[CustomerID] = s.CustomerID), 0) as [121-180]  , ISNULL((SELECT SUM([Total\_Sale])  FROM [featherman].[Sales] as s  WHERE [Paid] = 0 AND (GETDATE() - [Invoice\_Date])  BETWEEN 181 AND 360  AND c.[CustomerID] = s.CustomerID), 0) as [181-360]  , ISNULL((SELECT FORMAT(SUM([Total\_Sale]), 'N0')  FROM [featherman].[Sales] as s  WHERE [Paid] = 0 AND (GETDATE() - [Invoice\_Date])  BETWEEN 0 AND 360  AND c.[CustomerID] = s.CustomerID), 0) as [Total Unpaid Last Year]  FROM [featherman].[Customers] as c    *The indenting is optional but is helpful to organize the code and reduce the complexity of the code.*  *You can develop your own style of using indentation. Experiment with indentation until you develop your own style.* | No you can’t use a case statement to provide this functionality, a case statement returns only 1 column  This query is simple and powerful at the same time. It is simple in that *only two dimensions are included, CustomerID and CustomerName*, then six columns of accounting data is compiled and the formula is identical for all six columns.  The power is that the [featherman].[Sales] fact table is queried over and over using the same three criteria:  a) is the invoice unpaid?  b) is the invoice date within a date range in the past?   c) is the CustomerID in the sales table match a value of CustomerID in the [featherman].[Customers]dimension table which we are using to generate the report?  These subqueries all use a JOIN functionality in the WHERE statement c.[CustomerID] = s.CustomerID  Another interesting aspect of this query is the calculation of the number days ago the invoice was.  (GETDATE() - [Invoice\_Date])  Better than that we can use the results of that date calculation in the BETWEEN statement such as  GETDATE() - [Invoice\_Date]) BETWEEN 181 AND 360  *This query produces the dataset that you see in the picture above. Once you have this data compiled, it can be saved into its own database table.*  *Once you have data into a database then you can use free chart making software to visualize the data as shown below.* |

*A likely scenario is to run the aging report once per week, as part of the process of refreshing the database table and attached dashboards the process would be:*  
  
a) **create a database table** that will store the calculated data (this only needs to be done once). The reports and dashboards are connected to the database table that gets refreshed. It is helpful if this table has the same schema as the query. A future module will show how to create a database table based on a query.  
b) to refresh the database table, **delete the rows**  
c) take the query above and convert it into an **INSERT INTO SELECT \* FROM query** to load the query results into the database table.

***Challenge***: what if you want the aging report to group customers by city or state?

A problem with the accounting dataset is that it is hard to turn into charts because the data is segmented into so many columns. The chart on the right identifies which customers are responsible for long overdue invoices (between 180 and 360 days) but this report is cumbersome.

While the aging report is useful in its own right, if the categorization occurred in one column, then other reports can be made. So you need to know many ways to format data.

Click the link below to add some new records to the Featherman.Sales table. If you make a PowerBI report using the query below, you should be able to see the orders you added. However the report may not refresh as the free version of PowerBI only updates once per day.

<http://cb-ot-devst05.ad.wsu.edu/featherman/MIS325/Section2-DataModifications/InsertNewRecordAdvanced2.aspx>

|  |  |
| --- | --- |
| USE Featherman\_Analytics;  SELECT c.[CustomerID], [CustomerName], [Invoice\_Date]  *No sub-query here but this query ties this module to the prior module. Explanation of code below.*  *Here we write a query that categorizes the unpaid invoices. First, we calculate the # Days old the invoice is.*  *Then use our good friend the CASE statement to create a descriptive column that can be used as a slicer*  , CAST((GETDATE() - [Invoice\_Date]) as decimal(4)) as [Days]  , CAST(CAST((GETDATE() - [Invoice\_Date]) as decimal(4)) as nvarchar(4)) as [DaysT]  , Case  WHEN CAST((GETDATE() - [Invoice\_Date]) as decimal(4)) BETWEEN 0 and 90 THEN 'Current'  WHEN CAST((GETDATE() - [Invoice\_Date]) as decimal(4)) BETWEEN 60 and 90 THEN '60 days late'  WHEN CAST((GETDATE() - [Invoice\_Date]) as decimal(4)) BETWEEN 91 and 120 THEN '90 days late'  WHEN CAST((GETDATE() - [Invoice\_Date]) as decimal(4)) BETWEEN 121 and 180 THEN '1/3 year late'  WHEN CAST((GETDATE() - [Invoice\_Date]) as decimal(4)) BETWEEN 181 and 360 THEN '1/2 year late'  END as [AR Category]  , SUM([Total\_Sale]) as [Total Unpaid Amount]  FROM [featherman].[Sales] as s  INNER JOIN [featherman].[Customers] as c  ON c.CustomerID = s.CustomerID  WHERE [Paid] = 0 AND CAST((GETDATE() - [Invoice\_Date]) as decimal(4)) < 500  GROUP BY c.[CustomerID], [CustomerName], [Invoice\_Date]  ORDER BY c.[CustomerID]      Here we calculate the number of days between today and the invoice date. The result is in some clock system and needs to be CAST() format the data into a decimal number, number of days, to be used in charts.  To facilitate Power BI analysis this Days column is next copied and CAST() into a textual variable (nvarchar) so that it can be easily used in the X axis of a chart (which prefer text columns to display the data correctly).  Notice the two similar columns are named Days and DaysT. **In PowerBI be sure to sort DaysT by the Days column**. This pattern of sorting a textual column by a numeric column is common such as sorting the months or days of the week. Unfortunately sorting is not automatic in many data visualization software.  While we can make charts quickly in PowerBI using the [Total Unpaid Amount] field, the next stacked column chart is better.    Made from our CASE statement  Here we add up invoice amounts for each Accounts receivable category. This is helpful to see which customers are driving the accounts receivable problem. Below we count up the invoices in each category (created by the CASE statement) | |
|  |

**Answer to challenge**: a common strategy is to merge and compile data and store it in a temporary table, then query the temporary table. Often several steps of data compaction are needed to shape and summarize the data as needed. In this case the accounting data from the Customer and Sales tables is compiled saved into an AcctgAgingReport table. From this table you can group the data by state. Looks like the data entry system needs to be improved… (Note: the results compiled here by state differ than shown above as considerable time has lapsed)

Table

Description automatically generated  
SELECT [State], SUM([0-30]) as [0-30], SUM([31-60]) as [31-60]  
, SUM([61-90]) as [61-90], SUM([91-120]) as [91-120], SUM([121-180]) as [121-180]  
,SUM([181-360]) as [181-360]  
, SUM([Total Unpaid Last Year]) as [Total Unpaid Last Year]

FROM [featherman].[AcctgAgingReport] as a  
JOIN [featherman].[Customers] as c  
ON c.CustomerID = a.CustomerID  
GROUP BY [State]