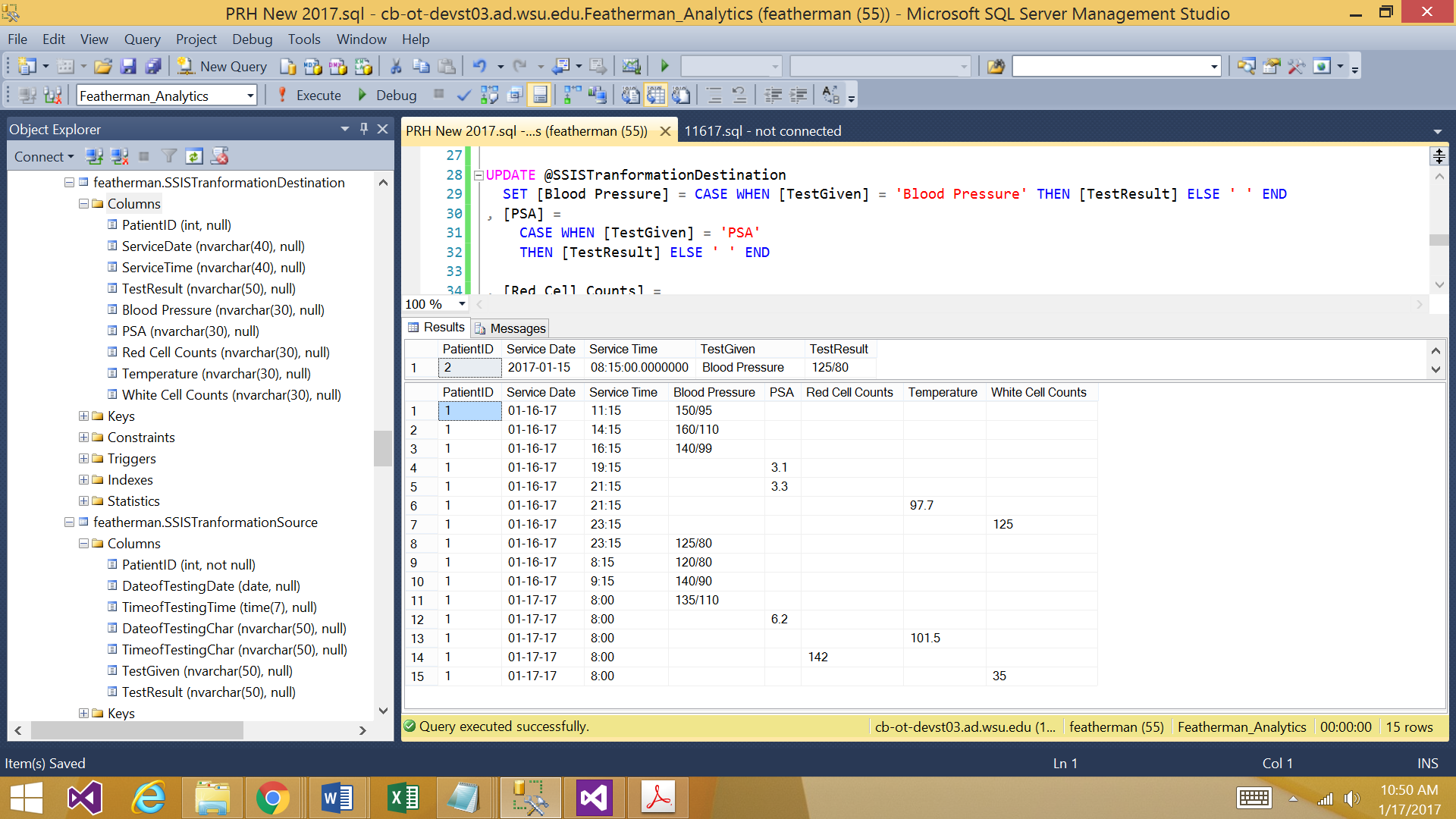
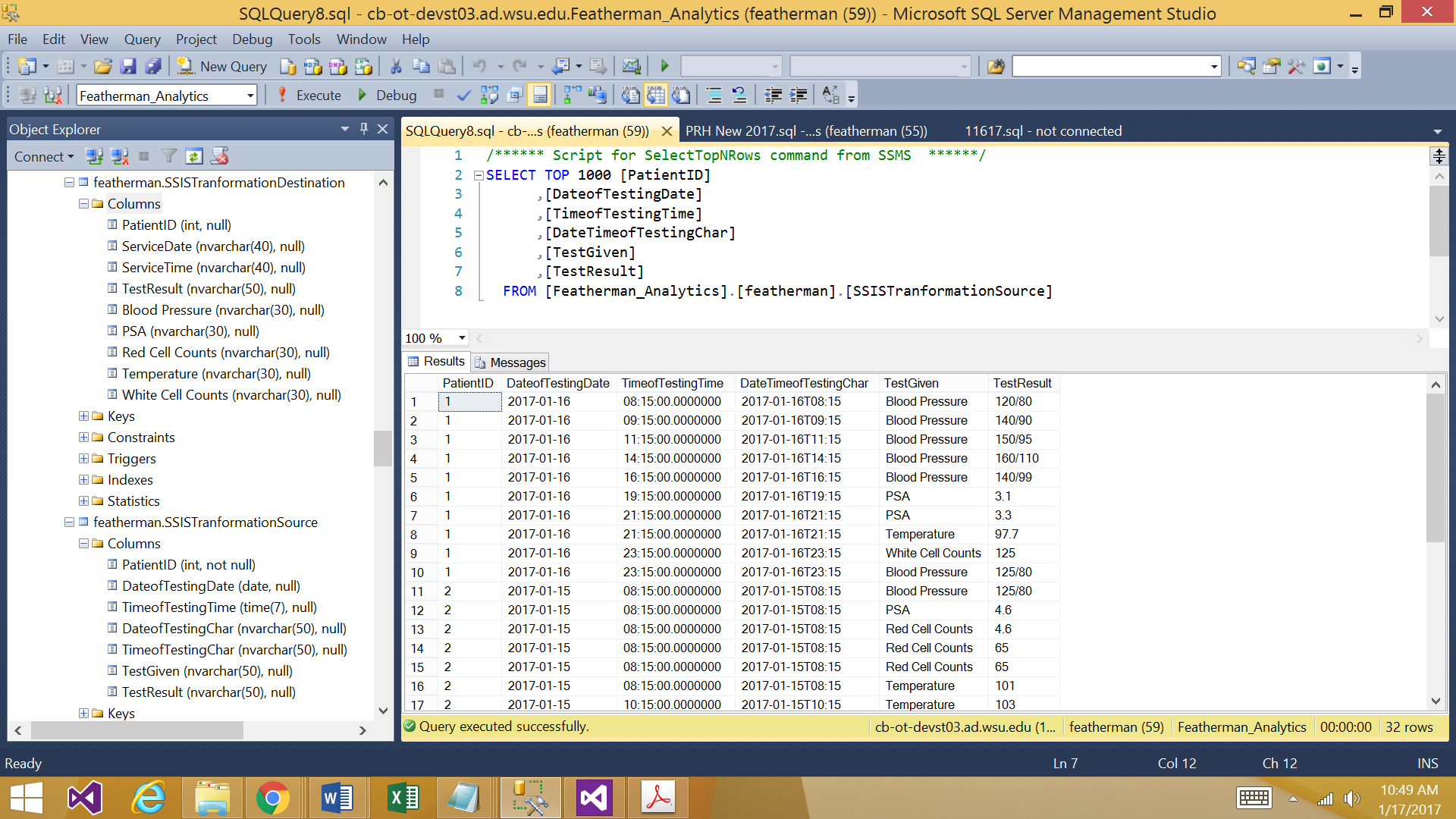
**Featherman’s Adventures in T-SQL© – Reformatting Data from Tall Tabular to Cross-Tabular With No Aggregating**

A client clinic has a weekly need to reformat data from a tall and narrow tabular report (usually about 10,000 rows per week) to a parameterized report that changes the formatting of the data. The work is currently being done in Excel with about eight hours of cut and paste operations, followed by chart-making. While this developer was initially loathe to use SQL to automate the fliping of data (make SQL build a dataset to produce the output like Excel can), the problem was interesting enough and actually gives an insight into solving the drudgery that some workers endure. . Making a database function like a spreadsheet (allow the freedom of copying and pasting data can bend your mind, however this module was created because the skills transferred actually turn out to be useful, and used in many different situations.  
  
Further the client has been using the Excel formatted report for a decade so it may be easier to automate the report formatting, rather than introduce new technology. The report format is shown to the left, this is an example of the desired result. The clinical health care worker uses an EMR (Electronic Medical Reporting system) to record patient vital signs at different time intervals. The requirement is to have this data formatted for easy consumption as shown on the left.

The initial reaction of this developer was to test and implement some reporting software to solve the problem, where the reporting software formats the data and produces the charts. You can see that the report is easy to vertically scan (eyeball), with the relevant data organized by columns. The client just wants to see the data and is not asking for any metrics to compare values for example percentage change in blood pressure or PSA (FYI men at some point should have their PSA levels monitored during routine physicals). Because there is no analysis of the data rather just reformattting, the output of data makes heavy use of nvarchar (text) fields.   
The original format of the data is to the left. Imagine 10,000 rows of this type of data. The first column is the patient ID#, then the columns show the date, time and results of a medical test being administered (such as blood work).  
  
Before continuing with this module, the reader is cautioned to review introductory SQL content including SELECT INTO queries, table variables, and date functions.

So this module has an ETL focus, first data is pulled from a database table and loaded into an array table. Next the data is poured into a second array table, more columns are added, then the data is poured into the new array table. Here the final results are not saved back to the database. As a side note - the first iteration of this module used a loop to reformat the data, and this module uses an UPDATE SET query which is more efficient.

|  |  |
| --- | --- |
|  | The hospital data could not be shown here so dummy data and table were created to demonstrate the ETL.  This is an example of the data that needs to be transformed and reported. The data is pulled from a hospital ERP system and needs to be categorized by the different values in the TestGiven column.  Notice that the date field flips the formatting to year-month-day (there is a culture setting for this) and the time field add a lot of extra values for miliseconds. The data is coming out of an ASP.NET webpage, and it would take some work to format it as it is going into the database. Here we format as it comes out of the database. |
|  | This is the data as it is stored in the database table. Luckily the TestGiven field has low cardinality (low # of potential values) that do not change, so a destination table shown to the left was created to catch the values in a reformatted manner.  This would be considerably harder if the number of query response values was different each month, and the columns needed to be created on the fly. |
|  | Here is the table that could hold the data after it is transformed for ‘reporting.’ Looking at the sample data above, you can see that it is nice to be able to see all of the transactions for each account#, and that the transactions are in the same column (for easier eye-ball analysis and for subsequent line charts). Notice we will make an arraytable with the same schema.  In the report the data can be quickly scanned. Since the destination table was created, our job is to transform the data to put it into this table schema. |

|  |  |
| --- | --- |
| USE [Featherman\_Analytics];  SELECT [PatientID] , CONVERT(varchar, [DateofTestingDate], 10) as [Service Date]  , FORMAT([TimeofTestingTime], 'h\:mm', 'en-US' ) AS [Service Time]  , [TestGiven], [TestResult]  FROM [featherman].[SSISTranformationSource]  WHERE [PatientID] = 1  ORDER BY [PatientID], [DateofTestingDate], [TimeofTestingTime] | Before we start we can see that the CONVERT line reformatts the date nicely, however turns it into a VARCHAR text field. Try out the different formatting types by changing the 10 to 11, etc.   The same is true for the FORMAT function. |

|  |  |
| --- | --- |
| USE [Featherman\_Analytics];  DECLARE @SSISTransformationSource TABLE ([PatientID] int , [Service Date] DATE, [Service Time] TIME , [TestGiven] nvarchar(50), [TestResult] nvarchar(50))  INSERT INTO @SSISTransformationSource  SELECT [PatientID] , CONVERT(varchar, [DateofTestingDate], 10) as [Service Date]  , FORMAT([TimeofTestingTime], 'h\:mm', 'en-US' ) AS [Service Time], [TestGiven], [TestResult]  FROM [featherman].[SSISTranformationSource]  ORDER BY [DateofTestingDate], [TimeofTestingTime]  SELECT \* FROM @SSISTransformationSource  ------------------------------------------  DECLARE @SSISTranformationDestination  TABLE ([PatientID] int, [Service Date] nvarchar(40), [Service Time] nvarchar(40)  , [TestGiven] nvarchar(50), [TestResult] nvarchar(50), [Blood Pressure] nvarchar(30), [PSA] nvarchar(30), [Red Cell Counts] nvarchar(30), [Temperature] nvarchar(30), [White Cell Counts] nvarchar(30))  INSERT INTO @SSISTranformationDestination  SELECT [PatientID] , CONVERT(varchar, [Service Date], 10) as [Service Date]  , FORMAT([Service Time], 'h\:mm', 'en-US' ) AS [Service Time]  , [TestGiven], [TestResult], [Blood Pressure] = ' ', [PSA] = ' '  , [Red Cell Counts] = ' ', [Temperature] = ' ', [White Cell Counts] = ' '  FROM @SSISTransformationSource  UPDATE @SSISTranformationDestination  SET [Blood Pressure] =    CASE WHEN [TestGiven] = 'Blood Pressure'   THEN [TestResult] ELSE ' ' END  , [PSA] =  CASE WHEN [TestGiven] = 'PSA'  THEN [TestResult] ELSE ' ' END  , [Red Cell Counts] =  CASE WHEN [TestGiven] = 'Red Cell Counts'  THEN [TestResult] ELSE ' ' END  , [Temperature] =  CASE WHEN [TestGiven] = 'Temperature'  THEN [TestResult] ELSE ' ' END  , [White Cell Counts] =  CASE WHEN [TestGiven] = 'White Cell Counts'  THEN [TestResult] ELSE ' ' END  SELECT [PatientID], [Service Date], [Service Time], [Blood Pressure], [PSA], [Red Cell Counts], [Temperature], [White Cell Counts]  FROM @SSISTranformationDestination  WHERE [PatientID] = 1  ORDER BY PatientID, [Service Date], [Service Time] | Detail line by line explanation on the next page.   The first part of this query creates the structure for an array table and loads data into it. Notice the column datatypes specified for the array table match the SQL Server source table. We actually drop one column that is not needed.  Notice the SELECT \* FROM @SSISTransformationSource table ignores the nice CONVERT and FORMAT functions. Oh Well. So we are retrieving data into an array table. Later a shortcut is shown to do this process.  Next a second array table is created with a bunch of new columns [Blood Pressure], [PSA] that are set to be empty. If we set the initial value to empty quotes ‘ ‘ then you place a space in the cell which is better than seeing NULLs.  Next we use the UPDATE SET statements to fill the columns with data. You can choose the formatting you wish with CASE code, but can you see that any formatting makes it much easier to read?  Finally we select the columns we wish (ditch a few) form the array table to demonstrate the awesome formatting. |

So this module provides another ETL example that creatively uses two different array tables to mold and shape the data. The ETL developer should remember that several simple data transformations can together pound the data into the format needed. There is no need to use staging tables to store intermediate versions of the data. Arrays are great in that you can keep adding columns of different granularity from different data sources. When solving problems of data merging, perhaps use paper and pencil to plan out the data merging, new columns needed, columns that can be dropped, measures added, etc. Since you are just working in the arrays, a lot of creativity is possible.

|  |  |
| --- | --- |
|  | Here we create an array table with the DECLARE statement, set the variables and then INSERT data into the array. You have to go slow when declaring the columns and their datatypes. Use this step to rename the columns and datatypes as needed.  Notice there is not filtering, normally your would have some filtering, such as pulling the last week’s or month’s data. As mentioned above it is debateable whether the CONVERT and FORMAT commands are needed in this intermediate step.  The final SELECT \* FROM statement builds and displays the dataset in the SSMS output window. |
|  | Here we create the second array table, it has many more columns according to columns requested by the customer. Notice that you can add any number of columns, with datatypes. This should be a very freeing experience as you build new datasets. You could use different INSERT INTO SELECT statements to load the columns, pulling data from different tables.  Here is where the CONVERT and FORMAT statements really shine, reformatting columns of data. Please do research on CONVERT, CAST and FORMAT functions of SQL. These are central to providing the datasets requested. |
|  | This is the section that takes the one column of values and distributes them to different columns so they stack up with othe similar values. You can use the UPDATE SET command to change values according to your needs. So previously you created the empty columns, here you fill them with data.  Each of the case statements is identical in functioning. One at a time, for each of the new columns we added for the medical tests (such as PSA), all the rows of data are checked to see of the value in the TestGiven column is for example PSA. If the tested for value is found then the value is copied into the column. So we are just shuffling the values to the right column. Doing so gives a nice effect. Being able to move data around into new columns into new combinations (such as concatenation) is a very powerful tool.  So in each case the TestResult column is evaluated, and the value is copied into the right destination column. This is a powerful metaphor. The query finishes with a careful selection of the columns that are desired in the output. You can see the TestGiven column is harvested then dropped. |

|  |  |
| --- | --- |
| USE [Featherman\_Analytics];  SELECT \* INTO #SSISTransformationSource  FROM [featherman].[SSISTranformationSource]  DECLARE @SSISTranformationDestination  TABLE ([PatientID] int, [Service Date] nvarchar(40), [Service Time] nvarchar(40), [TestGiven] nvarchar(50), [TestResult] nvarchar(50) , [Blood Pressure] nvarchar(30), [PSA] nvarchar(30), [Red Cell Counts] nvarchar(30), [Temperature] nvarchar(30), [White Cell Counts] nvarchar(30))  INSERT INTO @SSISTranformationDestination  SELECT [PatientID], CONVERT(varchar, [DateofTestingDate], 10)  as [Service Date]  , FORMAT([TimeofTestingTime], 'h\:mm', 'en-US' ) AS [Service Time]  , [TestGiven], [TestResult]  , [Blood Pressure] = CASE WHEN [TestGiven] = 'Blood Pressure'  THEN [TestResult] ELSE ' ' END  , [PSA] = CASE WHEN [TestGiven] = 'PSA'   THEN [TestResult] ELSE ' ' END  , [Red Cell Counts] = CASE WHEN [TestGiven] = 'Red Cell Counts'  THEN [TestResult] ELSE ' ' END  , [Temperature] = CASE WHEN [TestGiven] = 'Temperature'  THEN [TestResult] ELSE ' ' END  , [White Cell Counts] = CASE WHEN [TestGiven] = 'White Cell Counts'  THEN [TestResult] ELSE ' ' END  FROM #SSISTransformationSource  SELECT [PatientID], [Service Date], [Service Time], [Blood Pressure], [PSA]  , [Red Cell Counts], [Temperature], [White Cell Counts]  FROM @SSISTranformationDestination  WHERE [PatientID] = 1  ORDER BY PatientID, [Service Date], [Service Time] | Newsflash!! You can copy the data and schema or just the schema into a temptable (starts with a # not a @) which lives in tempdb part of SSMS.  select top 0 \*  into #mytemptable  from myrealtable  Where @Table Variables (here called arrays) need to be constructed column by column, which does allow you to change column names and data types, if you have a database table that just happens to have the correct schema, you can either pull all the data and schema (column names and data types) or just the schema.  If you just want the schema, use SELECT TOP 0 \* INTO #temptable FROM database.tablename  Here the second array has changes in column names and datatypes from the database tables, so the schema is specified. It is possible to also set up the destination data table format such that   1. Data can be added to it and 2. You can copy the schema into a temp table   Also shown here, you can often perform all the functionality in the second INSERT INTO SELECT statement. Rather than create empty columns and use a separate UPDATE set command (useful when data is coming from different tables or databases), here the CASE statements are within the SELECT. |

This concludes this ETL adventure. Here is an example of several data transformations being needed to complete the dataset requirements. Happy coding! Featherman@wsu.edu