Featherman’s T-SQL Adventures © - Arrays (aka SQL Table Variables)  
*Why learn arrays? Loading data into arrays makes it waaaaaay easier to manipulate the data, and build analytics*

***Conceptual:*** Arrays are used in data management, data manipulation, data compiling, and report generation processes. Arrays are created in the RAM of the database server and deployed temporarily to perform these data management tasks. Transaction data from different sources stored on disk can be copied into one or more RAM based arrays so that the data can be comingled, compiled, often condensed, and then analyzed. After the data is ‘[wrangled’](https://favtutor.com/blogs/data-wrangling) into shape it is often saved into database tables at different levels of condensing and detail. The database tables often serve as the data source for dashboards and reports. Other times the DBA builds the dataset for a report builder or analyst.

It is typical to have to bring together many different data sources to *analyze business performance* (i.e., it seems like we are playing with data, but actually we are practicing building datasets, upon which further analysis (perhaps data science) and million-dollar business decisions can be made. Marketers for example often purchase datasets such as DMV data to merge with their internal customer data to locate where the affluent consumers love by zip code (if car model is a sign of affluence -- which is an assumption).  
  
Once the data is merged, analytics can be run, dashboards updated, and the newly configured data can be written back to disk storage for long-term storage. The process of data retrieval and manipulation can be automated with SQL stored procedure, for example so that a dashboard can be updated every hour. The major benefit of using arrays is that they can take in data from many different sources and merge, the data into new formations, and then saved back to disk. If you connect reports to this process, then the report generation and distribution process can be automated (which is the goal).

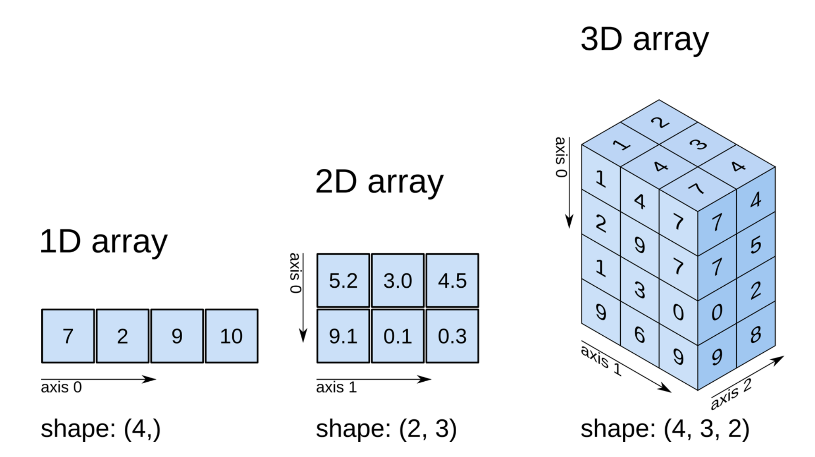
To perform analytics and generate insight into business problems the analyst often needs to merge many different datasets. The insight and value is generated when new columns of metrics can be built that help to make sense of a business scenario. An example is to merge in weather data or credit score data to investigate seasonal changes to automobile sales in different regions. While analysts and DBA’s can pull together data into arrays for data processing for a special research project, DBA’s also spend considerable time automating these processes to perform regular ETL (Extract, transform, Load) processes to build the data pipeline that supports business analysis.

Figure courtesy https://softwarerecs.stackexchange.com

In the pic to the right you can see that arrays can be of different shapes. We use a 2D array that resembles a table of data. If we were to expand one column of a 2D array to cut that by all the values of another dimension, then you would have a 3D cube. Original data warehouses were built on the data cube model. 3D arrays are not discussed further.  
  
*Consider the scenario* of a graduating student that needs to choose amongst three job offers in three different cities. They are needing to base their career decision on multiple criteria (e.g., weather data, crime data, housing price data, entertainment opportunity ratings, etc.). A thoughtful graduate would pull all the data for each decision criteria into a spreadsheet to facilitate comparison. An array is similarly a spreadsheet in the memory of the computer, you can manipulate the data in each cell (intersection of row and column), and you can then aggregate the data in new ways. In this example you would want to create a new column such as a weighted average so you can compare your options numerically.

Once you pull the data together into an array of your specification, you can make new calculations and comparisons to facilitate decision-making. Using data from different data sources you can build new metrics – typically using an UPDATE SET, however you can also build columns when bringing in the data with the INSERT INTO SELECT FROM WHERE methodology. Often by pulling data together you can see the patterns and can have an insight epiphany. Building new columns of metrics is the alchemy of business analysts, this is the value added. Successful business analysts create new insights and understanding.

The point is worth repeating you don’t get promoted because you can do SQL, manage data, create reports & dashboards, and automate processes, these are expected. The analyst or DBA gets promoted when they provide insight and analysis to top management that makes problem-solving easier. It’s a merging of business understanding, intuition, technical skills and persistence. Analyst is the fast-track to Director.  
  
But we are getting ahead of ourselves. Back to the nuts and bolts of SQL Arrays, which by now I hope you can see are a central building block of data management and reporting. Because you can add columns of data together, they can be at different levels of granularity. This is another innovation that has not been presented in our class together.

Table variables (arrays) have been used by SQL programmers for over 50 years to transfer data from storage to working memory to perform data management, data integrations, and data aggregation. Shuttling data from disk storage to RAM storage for processing is a standard operating procedure for data centers because RAM has always worked a lot faster than secondary storage (RAM processing is measured in nanoseconds compared to milliseconds for disk read/write speed). For our needs, arrays tables are used to hold data and are useful to pull data together when other SQL query methodologies do not work. Why arrays historically? Before data warehouses the reports had to be run at night when business operations stopped. With arrays, data can be copied into working memory and the database remains free for inserts/updates/and deletes.

A recurring reporting problem is that data is needed at different levels of granularity, for example it is common to add a final column with totals or % of total metric. In SSRS, PowerBI or Tableau reports you can add column and row totals easily. If your dataset from a SQL query needs a mixture of detailed and summary data however (as a function of ETL processing) then you can use one or more arrays to mingle data that was calculated at different levels of summarization. Often several layers of aggregation and data shaping are needed in ETL processes.

Array tables excel at allowing you to pull data together, co-mingling data from different sources, into a compiled dataset that is ready for analysis, by specialists in different business disciplines (ie finance, marketing, HR, etc.) Bringing disparate data into one organized dataset allows dashboards and reports to be easily generated and updated. A pre-requisite is that you need to have a dimension attribute (column) in common between the different source data tables. The field in common will usually be a primary key such as employeeID, productID, storeID, but can also be textual city, state, or country (for example when compiling econometrics data). When you can’t figure a way to pull data together from different sources (i.e., the joins don’t work, and neither does a sub-query), then pull the data into one or more an array table. If you need more than one array then you can join the array tables inside your query.

A future module will show the use of array during an Extract, Transform and Load process. The newly compiled array table can then be merged with other array data in an iterative ETL process of data culling, formatting and combining. The data organization can be automated by running stored procedures and the custom-built array data dataset can be saved to SQL database tables for permanent storage.

Array tables then are your canvas and you the emerging DBA can paint your masterpiece dataset onto them, column by column.

Arrays are either wide (many columns), tall (many rows) or both. You specify the columns, their names and data types.

To improve data management and simplify organization of data assets, a DBA may want to build fewer more complex arrays, each one serving as the datasource for many reporting needs (rather than build lots of queries, build a few that are multi-purpose!!). Arrays should be documented.

*Let’s get started!*

***Examples:***

|  |  |
| --- | --- |
| USE [Featherman\_Analytics];  DECLARE @SubCategory nvarchar(30) = 'Mountain Bikes'  SELECT [Model], Sum(OrderQuantity) as [Total]  FROM FactResellersales as s  Inner JOIN [dbo].[AW\_Products\_Flattened] as p  ON p.ProductKey = s.ProductKey  WHERE [Sub Category] = @SubCategory  GROUP BY [Model]  In a later module when we automate the report creation, we turn the query into a stored procedure. The value for the @subCategory variable would be passed from the reporting interface that is ‘calling’ the stored procedure  For example if the query is saved into a stored procedure called spUnitsForSubCategory then you would run the query and retrieve the data by typing  EXEC spUnitsForSubCategory 'Mountain Bikes' | *Before we jump into array which are a grid of variables where each column can be a different datatype, let’s take another look at declaring and using a single variable in SQL. What is a variable? A variable is a named piece of system memory (in RAM) that is created by, accessible and manipulated by SQL programs and the users of the SQL programs.*  *The DECLARE @variablename datatype is the way to create a variable. In this line you can set the value of the variable immediately (such as run a sub-query) or use a SET command later on. The program shown to the left allows one textual sub-category value. Here we show the query in test mode (by giving a value to the variable). Future usage of the query will receive values (from a dropdown list in an ASP.NET webpage, Excel, SSRS, or Tableau project) passed in and used in the WHERE statement to filter the results retrieved from the database. It works really well. Variables are used to pass values from one system to the next.*  *A simple nvarchar variable that can accept a word or phrase. Most variables that programmers use can store one value, a number, a date or a word or set of words. In T-SQL use the DECLARE phrase to create a variable in the memory (RAM) of the development machine. The DECLARE line shown to the left creates a variable of datatype nvarchar that can accept up to 30 characters (letters or numbers).*  *The idea is that this query would be ‘called’ or ‘invoked’ from Excel or a reporting software which passes in the value for the variable. This is called a parameterized query and can provide different results depending on value passed into the query, similar to a slicer. We can see that the variable is used in the WHERE clause to filter the results. So remember you can always pull a summary value into your dataset by using local variable and sub-queries. If you bring in a variable at a different level of summarization, then you can make a whole new set of metrics, such as percent of total for different categories.*  *Next in this document we focus on a special type of variable called a table variable because the shape is a 2D grid table. You specify the number of columns, then later make a plan to fill the rows.* |

***Some Data Management Tips***

*Its time to do some DBA work copying tables of data from one database to the next. This is much easier than exporting and emailing an excel, zip, or csv file. Another technique that DBA’s use is to create a .bak backup file and copy that to a new machine and restore the data onto a different machine that also has SQL Server Database software installed.*  
Procedure 1: copy database table from one DB into a second DB **creating a** **new database table**   
To copy data from one database table into a new database table in the same database or into a different database, use the following procedure. When selecting the data, you could change the columns, filter the columns, filter the rows, change granularity, etc. When you copy the data, you create a new table. You can however only create the table once, so future data migration into the same table would use procedure #2, as you already have the table created.  
  
SELECT \*   
INTO newtable   
FROM oldtable  
---------------------------so if you log into SSMS using *your* database you could then copy data into your   
database CREATING A NEW TABLE in your database, here is an example

***What is an array anyway?*** *An array is a spreadsheet like partition of RAM, that can store data into rows and columns.*

*An array is like a tabular container that you put different data types and data sources into and make soup (i.e., new columns of analytics).*

USE [MF##YourDatabase]

SELECT \* INTO [MF##YourDatabase] .[dbo].[Australia]

FROM [Featherman\_Analytics].[featherman].[BikeSales\_Australia]

--------------------------------------------------------------------------------------------------------------------------  
Procedure 2: save SQL query results **into a** **new database table**

USE [MF##YourDatabase]

SELECT \* INTO [MF##YourDatabase].[dbo].[NewTableName]

FROM

( SELECT SQL Statement goes here )

as ArrayTableName -- here just put some random name of a temp table. You never use this.

--------------------------------------------------------------------------------------------------------------------------  
Procedure 3: save an @rray of data **into a** **new database table**

*After you get the data compiled correctly in an array, you can copy the data from the array into a new DB table (perhaps to serve as the datasource for a report)*

SELECT \* INTO [MF##YourDatabase].[dbo].[NewTableName]

FROM @SalesPerformance

-------------------------------------------------------------------------------------------------------------------------------------

Procedure 4: copy data **into an** **existing** **database** table

*You can copy rows of data from one database table into a second database table as follows (for example when copying data to archives).*

INSERT INTO database table   
SELECT \* FROM [database].[table]

*If you are getting the data from a SELECT query and you are wanting to save that resultset directly into an existing database table, you first have to make sure the table does NOT have an identity (autonumber) column. Next you have to specify the columns that you are inserting into (which suggests that you do not have to supply a value for all the columns (can have null columns). An example is*

INSERT INTO [MF##YourDatabase].[dbo].[AcctgAgingReport]

([CustomerID], [CustomerName], [0-30],[31-60],[61-90], [91-120], [121-180], [181-360], [Total Unpaid Last Year])

SELECT …… mod 5 Aging report example

-------------------------------------------------------------------------------------------------------------------------------------

Procedure 5: load data from one database table **into an** **existing** **database** table

*You can copy rows of data from one database table into a second database table as follows (for example when copying data to archives).  
Sometimes you clear out the rows in the existing table before you load new data in, other times you append new data to an existing table.*

INSERT INTO database table   
SELECT \* FROM [database].[table]

-------------------------------------------------------------------------------------------------------------------------------------

Procedure 6: load data from existing database table **into an @rray**

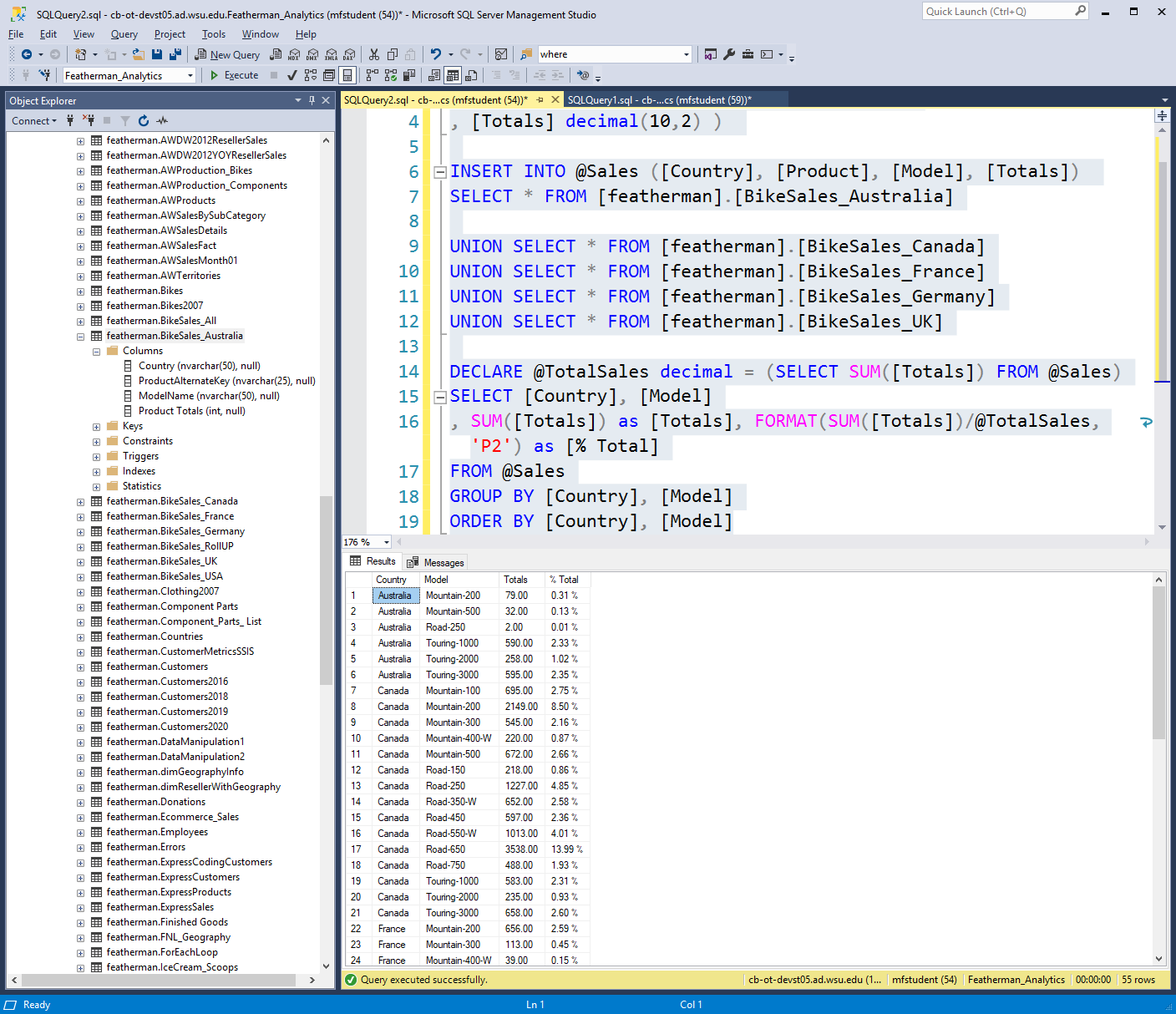
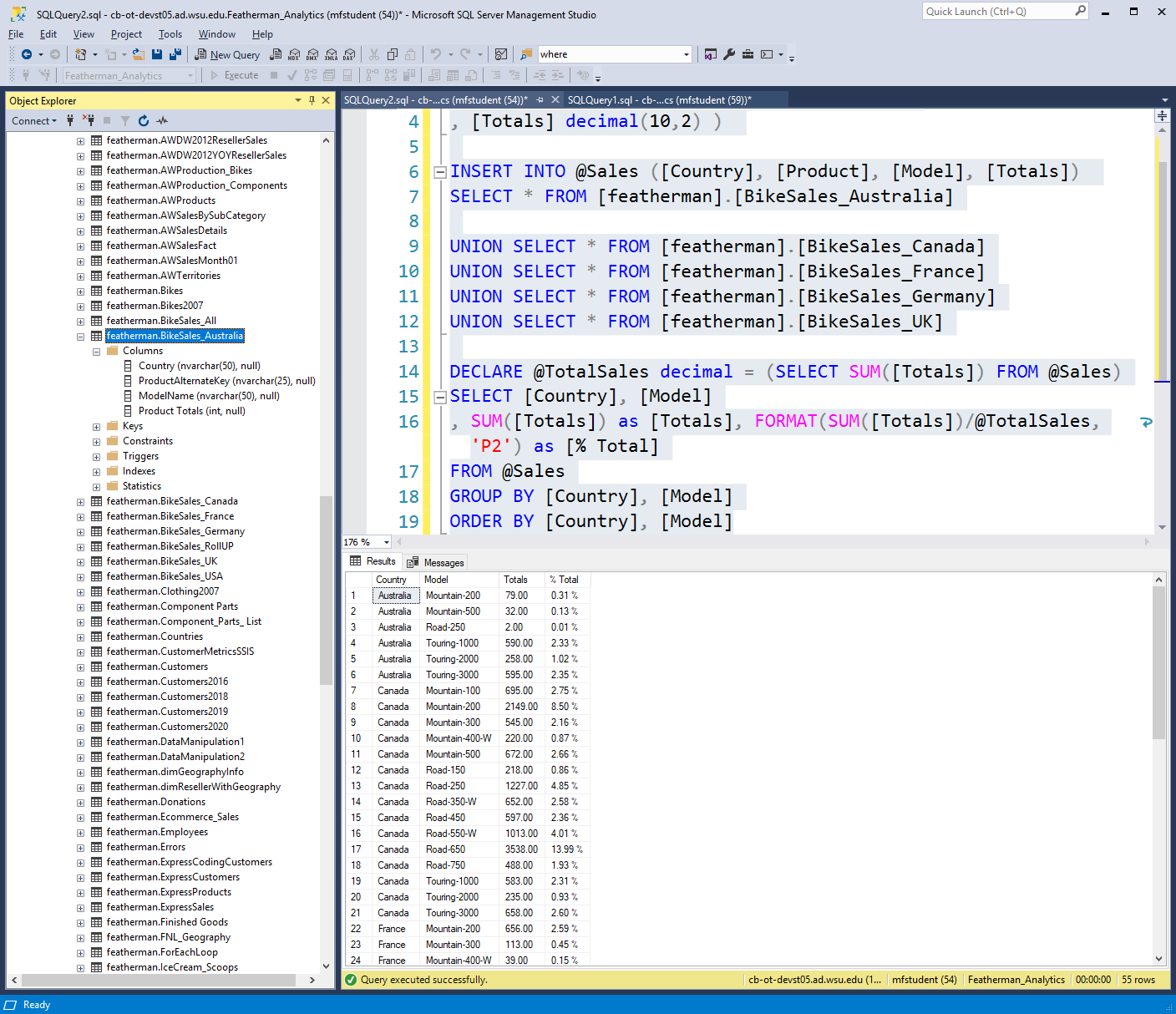
*You can also copy rows of data from a database into an array for further processing. You must already have the array created and columns specified before you can load some or all the columns. Often you load data into an array retrieved from different database tables, then processed, compiled and condensed. Again the data can be written back to different database tables based on the dimension of interest.*

INSERT INTO @array   
SELECT \* FROM [database].[table]

You can also add a GROUP BY to this INSERT INTO query to change the granularity of the data (summarize it) before loading the data into a table or array.

**Copying Data: DBA Examples**

|  |  |
| --- | --- |
| USE [Featherman\_Analytics]  SELECT \*  INTO Featherman\_Analytics.dbo.ManagersList  FROM [AdventureWorksDW2012].[dbo].[DimEmployee]  WHERE [Title] LIKE '%manager%'  *If you are logged into YOUR database then you can use this technique to move some data into your database account. Log into YOUR database, then change the code above to YOUR database. Your USE and INTO statement should be YOUR database.*  *You are making a new table so give it a nice name.* | *First we demonstrate code that demonstrates code that copies a table of data from AdventureWorksDW2012 into a new DB table. In this case both tables are in the same database but you can use this code to copy from one database to the next such as copying data from featherman\_analytics into your MF01### database (or similar). Change this query and use YOUR database similarly.*  *This code creates a new table in the database specified by the INTO statement. The code creates the ManagersList table in the* Featherman\_Analytics *database and inserts the following data INTO it. The data copied in comes FROM the DimEmployee table of AdventureWorksDW2012.* |
| INSERT INTO SalesArchives  (SELECT \*  FROM SalesTransactions WHERE CustomerID = @p1 AND OrderDate < ‘1/1/2016’)  DELETE FROM SalesTransactions WHERE CustomerID = @p1 AND OrderDate < ‘1/1/2016’ | In this example you ***copy rows of data to an archives database table, then delete the rows from the transaction table***  Here data is first copied from the SalesTransactions table to the SalesArchives table (to keep the transactions fact table from getting too big, slowing down report queries).  After copying the data to archives you can delete it from the SalesTransactions table. |
| INSERT INTO @RSSales ([ProductKey],[Reseller Units])  (  SELECT [ProductKey], SUM([OrderQuantity])  FROM [dbo].[FactResellerSales]  GROUP BY [ProductKey] ) | *Pertinent to this module on arrays, this query demonstrates how to load rows of data from database tables into an array named* @RSSales*. We copy data from database tables INTO an array, to merge and integrate it with other data. Why?*  *We will use the INSERT INTO arrayname SELECT \* FROM SQL table command* |
| USE [Featherman\_Analytics];  INSERT INTO [featherman].[BikeSales\_All]  SELECT \*  FROM [featherman].[BikeSales\_Australia]  UNION SELECT \*  FROM [featherman].[BikeSales\_Canada]  UNION SELECT \*  FROM [featherman].[BikeSales\_France]  UNION SELECT \*  FROM [featherman].[BikeSales\_Germany]  UNION SELECT \*  FROM [featherman].[BikeSales\_UK]  *--this last query displays the comingled data*  SELECT \* FROM [featherman].[BikeSales\_All] | The next example will use an INSERT INTO and a UNION SELECT command, so they are introduced here. *Interestingly, this INSERT INTO command did not need parentheses around it.*  The INSERT INTO command copies rows of data from one table into the next and creates the table the data is copied INTO. (The FROM statement could include a WHERE statement to copy in a filtered set of data.  We could write 4 more INSERT INTO commands to copy the rest of the data from Canada, France, Germany and UK,  but IF the schema for the tables are all the same, then you can write the shorter UNION SELECT command which is an append query that copies and adds more rows to the destination table.  BTW you don’t have to copy all the rows of data, or all the columns of data from the source table (the FROM tables). So why do this query? To merge data from different data sources into one large table for further analysis. |

***Array Examples:***

*Use the prior information to move and copy data for each country, as needed. Now let’s see how arrays can make an analyst’s work patterns easier.*

*First array example: Merging tables of data into one large table. Here we load an array with data from different tables, pulling all the data needed for a report into one array. In this special case each of the database tables have the same schema. This example loads an array with data, it could also be used to load a database table with data from several other database tables*

It is common to give regional managers their own datatables on the corporate server to enable simple uploads of business performance data using a simple script. This approach can speed up data management and reporting at the regional level. Notice on the first image that there are six tables of data, one for each country where sales are made (The Rollup table stores the combined data – the opposite of drill-down is rollup).

A common requirement is to need to combine data from the regional BikeSales tables (rollup the data for higher level management) for some further analysis. How would you do that? Copy rows into Excel? No need (and that process is not automated so every time you want to refresh the report, ugh, back to Excel copy/paste purgatory. Behold the UNION query – which is used to add more rows of data to an array (or database) when the schema is the same across all the tables. You can use UNION queries to copy more rows of data into an existing table that is storing similar data.

What we want is the table on the right where you can see the data is merged from the different tables (Australia, Canada, France shown).

This example demonstrates how to use the INSERT INTO, SELECT \* FROM, and UNION SELECT commands to pull rows of data from different database tables into an array, then write that array to a new table in the database. In this following example all the tables are in the same database (at corporate HQ) but the tables can also be in different databases. This SQL code should give you some ideas, you can pull different data together, transform it, add metrics then save the data into a database table, finishing by connecting a report to that table. We will of course automate the entire process. Analyst/DBA’s that can automate report delivery for the rest of the company, are heroes, because they produce the compiled information managers need to make informed decisions. Better decisions based on data can be a competitive advantage.

|  |  |  |
| --- | --- | --- |
| USE [Featherman\_Analytics];  DECLARE @Sales TABLE ([Country] varchar(50)  , [Product] varchar(50), [Model] varchar(50)  , [Totals] decimal(10,2) )  INSERT INTO @Sales ([Country], [Product], [Model], [Totals])  SELECT \* FROM [featherman].[BikeSales\_Australia]  UNION SELECT \* FROM [featherman].[BikeSales\_Canada]  UNION SELECT \* FROM [featherman].[BikeSales\_France]  UNION SELECT \* FROM [featherman].[BikeSales\_Germany]  UNION SELECT \* FROM [featherman].[BikeSales\_UK]  DECLARE @TotalSales decimal  = (SELECT SUM([Totals]) FROM @Sales)  SELECT [Country], [Model]  , SUM([Totals]) as [Totals] , FORMAT(SUM([Totals])/@TotalSales, 'P2') as [% Total]  FROM @Sales  GROUP BY [Country], [Model]  ORDER BY [Country], [Model]  */\*It is common to compile data in arrays then save the compiled dataset into a new DB table to facilitate data management and reporting. Change the table names to your DB if you copied the tables, and then run this query.*  SELECT \* INTO [Featherman\_Analytics]  .[dbo].[BikeSales\_ByModelRollUP]  FROM  (SELECT [Country], [Model]  , SUM([Totals]) as [Totals]  , FORMAT(SUM([Totals])/@TotalSales, 'P2') as [% Total]  FROM @Sales GROUP BY [Country], [Model]  ) as [t]  */\* -------------------------------------- if you have already created the databasetable then you could use the code below to a) delete the prior data and b) load the compiled data into the existing table \*/*  DELETE [featherman].[BikeSales\_ByModelRollUP]  INSERT INTO [featherman].[BikeSales\_ByModelRollUP]  SELECT [Country], [Model]  , SUM([Totals]) as [Totals]  , FORMAT(SUM([Totals])/@TotalSales, 'P2') as [% Total]  FROM @Sales  GROUP BY [Country], [Model]  *In this example the % Total column is formatted so nicely, however using percent signs changes (and requires) the field to be nvarchar text.*  *This is not advised, because changing the datatype to text stop any further metrics (You would first have to change the column back to numeric in the next query or reporting software The next example displays the percent columns using 2 decimal places.* | Example #1 – creating an array to hold merged data then changing the granularity of the data and adding a new metric. Could you perform this query without an array? Yes but we are just getting started.  1) First you create a table array, here we have 4 columns of data (Country, Product, Model, Totals).  2) Next we insert all the data from the BikeSales\_Australia into that array.  3) Next you can append copy in 4 more tables of data using the aforementioned shorter UNION SELECT command.   4) Next we use a sub-query to total a column of the array, storing that value into a local variable.  5) Now that we have all the data in the array and have totaled one column, we use a SELECT statement to build and display the output. Here we add a new % of total metric column. We use a GROUP BY statement to change the level of granularity. *The* *product column was omitted to change the granularity from product to model*. *The result is a reduction in the # rows.*  What’s complex is that the array still had the 352 rows of product level data, and we could save the data at that raw, unprocessed level. The SELECT statement therefore adds a new column, changes the granularity and displays the results (fewer rows) but that data while on screen in a resultset is not in an array.  6) But we now that the data is shaped and formatted, we can save it to a new table in your database. When you need to create a new table name then use the SELECT \* INTO FROM (SELECT…)  The yellow code to the left created the table schema in the pic below.    7) when you want to refresh the data in the table to update the dashboards and reports, then you could use the code shaded on the left.  When you want to add records to an existing table you use an INSERT INTO SQL statement. If the table allows NULLS in some columns then you do not have to fill all the columns at the same time. The next query can continue the data transformations. It optional to delete the current data in the table before adding more. This type of query could be run once per hour or day to refresh the dashboards and reports (which is the point).  This query can be rolled into a stored procedure and run on a schedule. | |
| USE [Featherman\_Analytics];  DECLARE @Sales TABLE ([Country] varchar(50), [Product] varchar(50), [Model] varchar(50), [Totals] decimal(10,2), [%Total] decimal(10,2))  DECLARE @TotalSalesA decimal  = (SELECT SUM([Product Totals]) FROM [featherman].[BikeSales\_Australia])  DECLARE @TotalSalesC decimal  = (SELECT SUM([Product Totals]) FROM [featherman].[BikeSales\_Canada])  DECLARE @TotalSalesF decimal  = (SELECT SUM([Product Totals]) FROM [featherman].[BikeSales\_France])  DECLARE @TotalSalesG decimal  = (SELECT SUM([Product Totals]) FROM [featherman].[BikeSales\_Germany])  DECLARE @TotalSalesUK decimal  = (SELECT SUM([Product Totals]) FROM [featherman].[BikeSales\_UK])  INSERT INTO @Sales ([Country], [Product], [Model], [Totals], [%Total])  SELECT [Country], [ProductAlternateKey], [ModelName], [Product Totals] ,[Product Totals]/@TotalSalesA FROM [featherman].[BikeSales\_Australia]  UNION SELECT [Country], [ProductAlternateKey], [ModelName], [Product Totals] ,[Product Totals]/@TotalSalesC FROM [featherman].[BikeSales\_Canada]  UNION SELECT [Country], [ProductAlternateKey], [ModelName], [Product Totals] ,[Product Totals]/@TotalSalesF FROM [featherman].[BikeSales\_France]  UNION SELECT [Country], [ProductAlternateKey], [ModelName], [Product Totals] ,[Product Totals]/@TotalSalesG FROM [featherman].[BikeSales\_Germany]  UNION SELECT [Country], [ProductAlternateKey], [ModelName], [Product Totals] ,[Product Totals]/@TotalSalesUK FROM [featherman].[BikeSales\_UK]  SELECT [Country], [Model] , SUM([Totals]) as [Totals] , SUM([%Total]) as [% Total]  *This example does not copy the array data to a database, but that functionality can be added.*  FROM @Sales  GROUP BY [Country], [Model]  ORDER BY [Country], [Model] | | Example #2 – Changing the % of Total  What if the manager wants to drill down to see % of totals *by model* *within* each country? We would need several local variables.  How do we provide that data grouping? First we add a new column to the array that will hold a new metric. Next we pull the totals from the 5 tables into 5 variables. Next we run our INSERT INTO SELECT FROM UNION SELECT statements. The insert into loads the first set of rows, the UNION SELECT queries just add more rows. The [%Total] column values are calculated using a SELECT command in the red box to the left, whih only works as we have the local variables. *Cumbersome but effective.*  Each UNION SELECT statement also calculates the % of total for rows of data from that country, for each product in the dataset and populates the values for the new column [%Total].  The final SELECT statement is how you display the compiled data. The GROUP BY statement is used to *change the granularity from product to Model* (by removing product from the GROUP BY) | |

The rest of this document demonstrates the common techniques used in this module to load and manage the data in arrays. The above query is re-written five times to show there are different ways to produce the same output.  
  
The query above uses five database queries each of which calculates the total unit sales for one country, and assigns the values to local variables. Next the query uses five more database queries to merge and import data from five database tables into one array. While the data is being imported, the values for a column of metrics are calculated. Finally the data is displayed after changing the granularity of the data from product level to model level.

New techniques

1. On page 14 some important changes are made. First a second array is used and the granularity is changed from product to country and model. Next rather than use five database queries to calculate and load five local variables, the five queries are calculated using the data from the array.   
  
When the data is imported from one array to the second array the values for the [%Total] metric are calculated using a CASE statement, within the INSERT INTO SELECT FROM WHERE statement. While at first the code looks complex, the final effect is that the code is easier to write and read. The granularity of the data is also subtly changed from country/product to country/model.  
  
2. On page 16, further important changes are made to demonstrate a different way to run the query. First only one array is used and we are back to the products/country level of granularity. The change is that the values for the [%Total] column are not calculated as part of an INSERT INTO SELECT FROM WHERE statement (which is doing a lot and therefore needs planning). Rather the [%Total] values are calculated in a standalone UPDATE SET statement. The UPDATE SET functionality is therefore easier to envision and implement. The data is still grouped at the end to change the granularity to country/model

3. On page 18, the realization is made that we need to reduce the number of database calls for the values for the local variables (total units by country). Because the data is all the same metric, and only differs by value of the dimensional attribute (ie. France vs. UK, etc.) the realization was made that a GROUP BY query could be used, but how and where to store the results of that GROUP BY query?  
  
This version of the query uses a second array to store the country totals so five database calls are replaced by one. The classic INSERT INTO SELECT FROM query is used to load the data into the second array. Next a great leap forward is made (on lines 20-22 on page 18) where the UPDATE JOIN command is used. The UPDATE SET command can calculate and load the values into rows of the [%Total] column. The beauty is that since the two arrays can be JOINED that columns from either array can be used to create new columns of metrics. This is very helpful

4. On page 20 two small changes are made to show other options. First the array is changed back to a granularity of Country/Model. This means that the final display of the array does not need a GROUP BY transformation. Next the prior examples loaded the data from five tables into one array (for further processing). It is also possible to load the data from five database tables into one database table. This work has been done prior to the displayed query (using INSERT INTO SELECT FROM WHERE statements). Notice the two arrays are loaded using one INSERT INTO SELECT FROM WHERE statement retrieving the data from the [BikeSales\_All] database table. The fact that the data from one table could be loaded into two different arrays each with their own analytical focus should give you ideas that you can form data using arrays, and the same data can be presented using different dimensions (ie country, model, time period etc.). If you want to join the tables though as on lone 19, the arrays both need a dimension field in common (even if they have different levels of granularity as here country/model (array 1) and country (array 2).

It is subtle but notice on lines 8 – 11, when the data is imported using the INSERT INTO SELECT FROM WHERE statement the granularity of the data is changed from product (in the underlying database table) to country/model.

A picture containing text, electronics

Description automatically generated5. The last demonstration on line 22 – the decision was made to directly update a column of a database table using the contents of an array. The raw data that has been imported or copied into the [BikeSales\_All] table is at product level, and had to be condensed with a GROUP BY to save it at country/model level.  
  
Here we set up a table in SSMS that is at the right level of granularity so that it can accept the data at the right level (as per requirements), we just need to load the data and then provide the values for the calculated column. The @CountryTotals array is again used to store the results of the GROUP BY query.

To reflect a common reporting scenario, first the [%Total] column of the database table is zero’d out using an UPDATE SET command. The change here is that the UPDATE JOIN statement   
a) directly updates the column in the database not in the array (why not?) and   
b) the JOIN is connecting a database table to an array (why not)

|  |  |
| --- | --- |
| Example #2 Rewritten - Learning new tricks - using a CASE statement and a second array | |
| So here we leverage the INSERT INTO SELECT command to calculate the % of total column WHEN THE ARRAY IS LOADED WITH DATA. The prior query knowledge is useful to load data into arrays and create new calculated columns. Here the data transformation of the GROUP BY subtly changes the level of detail of the analysis, improving the usefulness of the data. The data is condensing from product to Model. | The above query is now rewritten several ways to show additional methodologies.  First we use a **second array** which does not include the product field (thus changing the granularity to Model and Country). The array also has the additional [%Total] field.  The ease of the UNION SELECT is apparent. Local variables are created on lines 14 – 18. The purpose of doing this is to aggregate and import summary data needed to expand the analysis. Very common for % total calculation.  The second array is loaded in lines 20 – 30. The values calculated by the CASE statement on 22-26 are inserted into the [%Total] column.  The [%Total] is calculated for all the rows of data, then the data is grouped by Country and Model.  The main difference is that data is loaded into the array then totaled, also the [%Total] data is calculated when the merged data is loaded into the second array. Finally CASE statements are easier to work with. |
| USE [Featherman\_Analytics];  DECLARE @Sales TABLE ([Country] varchar(50), [Product] varchar(50), [Model] varchar(50)  , [Totals] decimal(10,2), [%Total] decimal(10,2))  DECLARE @SalesByModel TABLE ([Country] varchar(50), [Model] varchar(50)  , [Totals] decimal(10,2), [%Total] decimal(10,2))  INSERT INTO @Sales ([Country], [Product], [Model], [Totals]) SELECT \* FROM [featherman].[BikeSales\_Australia]  UNION SELECT \* FROM [featherman].[BikeSales\_Canada]  UNION SELECT \* FROM [featherman].[BikeSales\_France]  UNION SELECT \* FROM [featherman].[BikeSales\_Germany]  UNION SELECT \* FROM [featherman].[BikeSales\_UK]  DECLARE @TotalSalesA decimal = (SELECT SUM([Totals]) FROM @Sales WHERE Country = 'Australia')  DECLARE @TotalSalesC decimal = (SELECT SUM([Totals]) FROM @Sales WHERE Country = 'Canada')  DECLARE @TotalSalesF decimal = (SELECT SUM([Totals]) FROM @Sales WHERE Country = 'France')  DECLARE @TotalSalesG decimal = (SELECT SUM([Totals]) FROM @Sales WHERE Country = 'Germany')  DECLARE @TotalSalesUK decimal = (SELECT SUM([Totals]) FROM @Sales WHERE Country = 'UK')  INSERT INTO @SalesByModel([Country], [Model], [Totals], [%Total])  SELECT [Country], [Model], SUM([Totals]),  CASE  WHEN [Country] = 'Australia' THEN SUM([Totals])/@TotalSalesA  WHEN [Country] = 'Canada' THEN SUM([Totals])/@TotalSalesC  WHEN [Country] = 'France' THEN SUM([Totals])/@TotalSalesF  WHEN [Country] = 'Germany' THEN SUM([Totals])/@TotalSalesG  WHEN [Country] = 'UK' THEN SUM([Totals])/@TotalSalesUK  END as t  FROM @Sales  GROUP BY [Country], [Model]  ORDER BY [Country], [Model]  SELECT \* FROM @SalesByModel | So you can use sub-queries to pre-load local variables with summary data to speed up analytics,  or you can load the array with all the data, then total a column perhaps with a criteria Both techniques are useful.  If you have time, copy this code, changing the tables and database info to **your account**. Verify that the output is the same. |

|  |  |
| --- | --- |
| Example #2 Rewritten - Learning new tricks – Using an UPDATE SET COMMAND |  |
|  | Here we introduce the UPDATE SET command which is used to add columns of calculated values.  A common pattern is to INSERT the dimension columns INTO the array, and then use UPDATE SET commands to manipulate the data to create new columns of metrics.  Here we use an array at the product level, but we include the % of Country total which will get calculated for each product within each country. The last SELECT statement on lines 28-31 changes the granularity to country and product.  So we load the array again, calculate the local variables, then provide values for one column [%Total] on line 19. The CASE statement within the UPDATE SET is elegant and parsimonious. |

|  |  |
| --- | --- |
| USE [Featherman\_Analytics];  DECLARE @Sales TABLE ([Country] varchar(50), [Product] varchar(50)  , [Model] varchar(50), [Totals] decimal(10,2), [%Total] decimal(10,2))  INSERT INTO @Sales ([Country], [Product], [Model], [Totals])  SELECT \* FROM [featherman].[BikeSales\_Australia]  UNION SELECT \* FROM [featherman].[BikeSales\_Canada]  UNION SELECT \* FROM [featherman].[BikeSales\_France]  UNION SELECT \* FROM [featherman].[BikeSales\_Germany]  UNION SELECT \* FROM [featherman].[BikeSales\_UK]  DECLARE @TotalSalesA decimal =  (SELECT SUM([Totals]) FROM @Sales WHERE Country = 'Australia')  DECLARE @TotalSalesC decimal =  (SELECT SUM([Totals]) FROM @Sales WHERE Country = 'Canada')  DECLARE @TotalSalesF decimal =  (SELECT SUM([Totals]) FROM @Sales WHERE Country = 'France')  DECLARE @TotalSalesG decimal =  (SELECT SUM([Totals]) FROM @Sales WHERE Country = 'Germany')  DECLARE @TotalSalesUK decimal =  (SELECT SUM([Totals]) FROM @Sales WHERE Country = 'UK')  UPDATE @Sales SET [%Total] =  CASE  WHEN [Country] = 'Australia' THEN [Totals]/@TotalSalesA  WHEN [Country] = 'Canada' THEN [Totals]/@TotalSalesC  WHEN [Country] = 'France' THEN [Totals]/@TotalSalesF  WHEN [Country] = 'Germany' THEN [Totals]/@TotalSalesG  WHEN [Country] = 'UK' THEN [Totals]/@TotalSalesUK  END  SELECT [Country], [Model], SUM([Totals]) as [Totals], SUM([%Total]) as [%Total]  FROM @Sales  GROUP BY [Country], [Model]  ORDER BY [Country], [Model] | You will be writing many, many UPDATE SET commands, they are great! |

|  |  |
| --- | --- |
| Example #2 Rewritten - Learning new tricks – the UPDATE JOIN command |  |
|  | Here we do away with the local variables (in the blue box above) and switch to using a second array which is show in the box above.  The data of the new array is at country level. The array is created on line 5, and loaded on lines 15-18. The same INSERT INTO SELECT command is used. You just have to know what data you need and how to group the data. THIS IS MUCH EASIER   But the real innovation shown here is that the arrays can be JOINED to each other to allow new columns of metrics to be created. This is referred as the UPDATE JOIN command. |

|  |  |
| --- | --- |
| USE [Featherman\_Analytics];  DECLARE @Sales TABLE ([Country] varchar(50), [Product] varchar(50)  , [Model] varchar(50), [Totals] decimal(10,2), [%Total] decimal(10,2))  DECLARE @CountryTotals TABLE ([Country] varchar(50), [CountryTotals] decimal(10,2))  INSERT INTO @Sales ([Country], [Product], [Model], [Totals])  SELECT \* FROM [featherman].[BikeSales\_Australia]  UNION SELECT \* FROM [featherman].[BikeSales\_Canada]  UNION SELECT \* FROM [featherman].[BikeSales\_France]  UNION SELECT \* FROM [featherman].[BikeSales\_Germany]  UNION SELECT \* FROM [featherman].[BikeSales\_UK]  INSERT INTO @CountryTotals ([Country], [CountryTotals])  SELECT DISTINCT([Country]), SUM([Totals])  FROM @Sales  GROUP BY [Country]  UPDATE @Sales SET [%Total] = [Totals]/[CountryTotals]  FROM @Sales s INNER JOIN @CountryTotals c  ON c.Country = s.Country  SELECT [Country], [Model], SUM([Totals]) as [Totals], SUM([%Total]) as [%Total]  FROM @Sales  GROUP BY [Country], [Model]  ORDER BY [Country], [Model] |  |

|  |  |
| --- | --- |
| Example #2 Rewritten - Learning new tricks – Merging Data in Tables prior to Import |  |
|  | Rather than UNION SELECT the product data by country and merge it in the array (the green highlighted code above) you really could merge the data on disk in the database.  This example assumes that all the data has already been merged into the BikeSalesAll table.  Merging data inside the database would be important when there is too much data.  The same data BikeSales\_All is loaded into two different arrays each at a different level of granularity.  We are starting to have fewer and fewer lines of code. |

|  |  |
| --- | --- |
| USE [Featherman\_Analytics];    DECLARE @SalesByModel TABLE ([Country] varchar(50), [Model] varchar(50)  , [Totals] decimal(10,2), [%Total] decimal(10,2))  DECLARE @CountryTotals TABLE ([Country] varchar(50), [CountryTotals] decimal(10,2))  INSERT INTO @SalesByModel ([Country],[Model], [Totals])  SELECT DISTINCT([Country]), [ModelName], SUM([Product Totals])  FROM [featherman].[BikeSales\_All]  GROUP BY [Country],[ModelName]  INSERT INTO @CountryTotals ([Country], [CountryTotals])  SELECT DISTINCT([Country]), SUM([Product Totals])  FROM [featherman].[BikeSales\_All]  GROUP BY [Country]  UPDATE @SalesByModel SET [%Total] = [Totals]/[CountryTotals]  FROM @SalesByModel s INNER JOIN @CountryTotals c  ON c.Country = s.Country  /\*similar code could also directly update the column in the DBtable.  --not an option here as the database table does not have the [%Total] column \*/  SELECT \* FROM @SalesByModel ORDER BY [Country],[Model]  SELECT \* FROM @CountryTotals |  |

|  |  |
| --- | --- |
| Example #2 Rewritten – Calculating and Refreshing a Database Columns from Data calculated in the array | |
|  | In the example above when the new column of analytics is calculated, you would then go update the database table next.  In this last example the database table  You can use the code in lines 3-6 to clear out the [% Total] column (so that we can see that it is in fact being populated by the UPDATE query starting on line 17. |

|  |  |
| --- | --- |
| USE [Featherman\_Analytics];  UPDATE [Featherman\_Analytics].[featherman].[BikeSales\_ByModelRollUP]  SET [% Total] = 0  SELECT \* FROM [Featherman\_Analytics].[featherman].[BikeSales\_ByModelRollUP]  ORDER BY [Country], [Model]  DECLARE @CountryTotals TABLE ([Country] varchar(50), [CountryTotals] decimal(10,2))  INSERT INTO @CountryTotals ([Country], [CountryTotals])  SELECT DISTINCT([Country]), SUM([Product Totals])  FROM [featherman].[BikeSales\_All]  GROUP BY [Country]  SELECT \* FROM @CountryTotals  UPDATE [featherman].[BikeSales\_ByModelRollUP]  SET [% Total] = [Totals]/[CountryTotals]  FROM [featherman].[BikeSales\_ByModelRollUP] s  INNER JOIN @CountryTotals c  ON c.Country = s.Country  SELECT \* FROM [Featherman\_Analytics].[featherman].[BikeSales\_ByModelRollUP]  ORDER BY [Country],[Model] |  |

Ok that’s enough! Many of the updating and data manipulation techniques were displayed using one example. Rewriting the same query many times gives the reader the ability to compare techniques and dive into their similarities and differences.

Well arrays are central to the production and distribution of analytics. I hope you enjoy using them, and that their usage solves problems for you, as they have for me.

Best wishes sports fans 😊

Mauricio