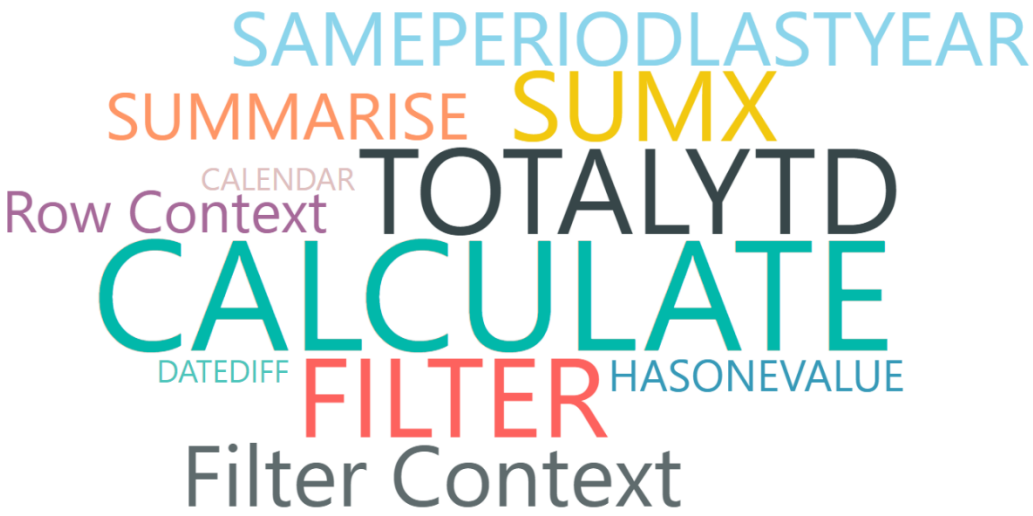
DAX Boot Camp Class Labs



Configuration Instructions

Requirements:

To run these labs, you must have Microsoft Power BI Desktop installed. You can download this from Microsoft here, http://Tinyurl.com/PowerBIDesktop

Your computer should meet the following minimum requirements:

- Microsoft Windows XP Professional, Microsoft Windows 7, Microsoft Windows 8, Windows 10
- 2 GHz or faster CPU
- 4 GB of available RAM
- 20 GB of free hard drive space
- Internet Explorer 10 or greater.
- A computer with Internet access

Setup and Preparation:

To prepare the student's machine:

1. Unzip the DAX Boot Camp folder on your C drive. The result will be C:\Dax Boot Camp
2. Inside the DAX Boot Camp folder, you will find the Class Labs and Completed Labs folders

Revision History:

Version	Date	Comments
1.0	04/4/19	Initial draft
1.1	05/10/19	Updates
1.2	08/13/2020	Lab Updates

Course Outline

Relationships Between Tables
- Understanding Relationships
- Defining Relationships

Dax Fundamentals
- Data Loading Basics

Data Modeling – Structuring the Data
- Data Import
- Data Architecture
- Understanding the Modeling Engine

Creating Calculated Columns
- Introduction to DAX
- Conditional and Logical Functions

Creating Calculated Measures
- Creating Aggregates
- Incorporating Time Intelligence

CALCULATE Function
- Changing Filter Context
- Learning Use-Case
- CALCULATE Utilization

Time Intelligence w/ Conditional Logic
- Incorporating Logic with Time Intelligence

Table Functions
- Table Function behavior
- FILTER function highlights

ALLEXCEPT
- Further modifying Filter Context

Working with Totals
- Understanding the Total Row
- Working with Variables
- HASONEVALUE Function

Evaluation Context
- Row Context
- Filter Context
- Context Transition

Semi-Additive Measures
- Account Balances
- Inventory levels
- Forecasting

Dynamic Security
- Row-Level Security
- Dynamic Row-Level Security

Role-Playing Tables
- Understanding Role-Playing Tables
- Navigation via Measures
- Multiple Table Imports

Course Goals

Specific topics I would like to learn about this week:

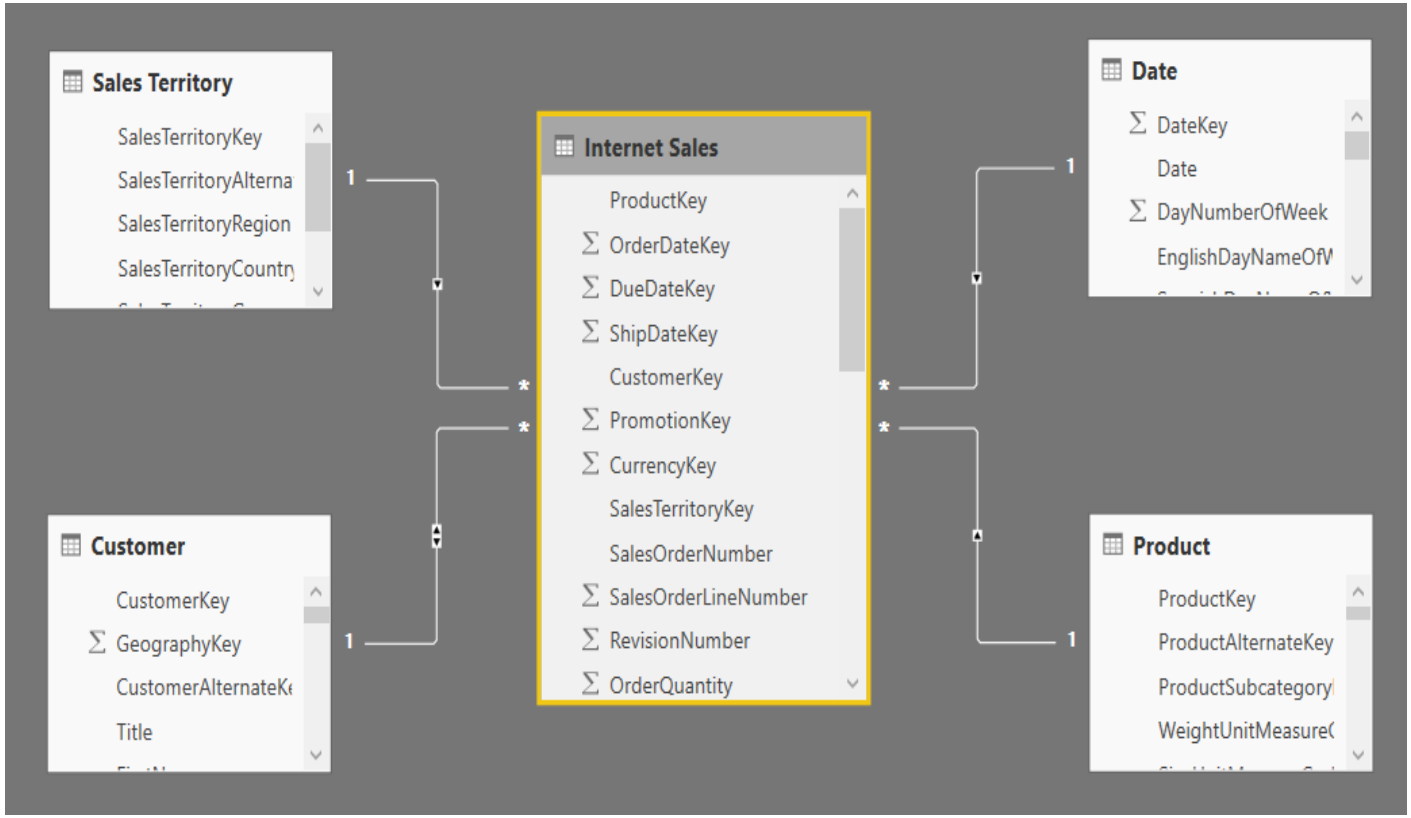
- _____
- _____
- _____
- _____
- _____

Contents

Module 1: Understanding and Defining Relationships

Module 1 Demo: Creating our first relationships

This first exercise is designed to help you understand the components of a data model involving relationships. Relationships between tables are a key fundamental to how Power BI and DAX work together.



Notes	

Module 2: DAX Fundamentals

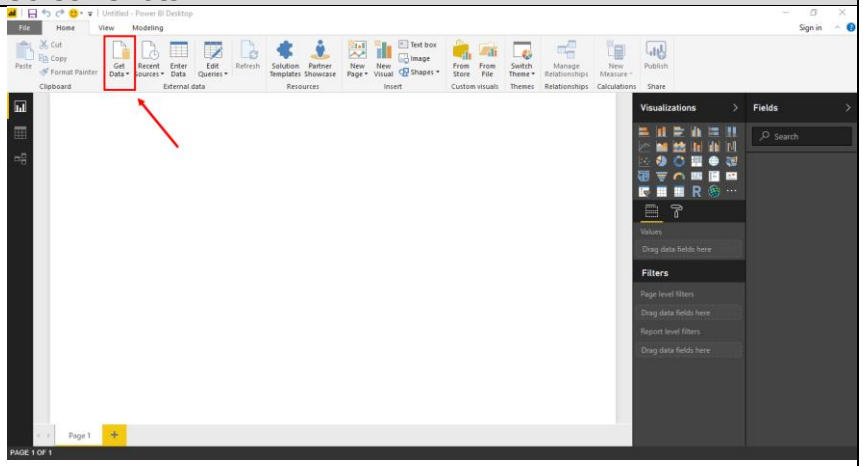
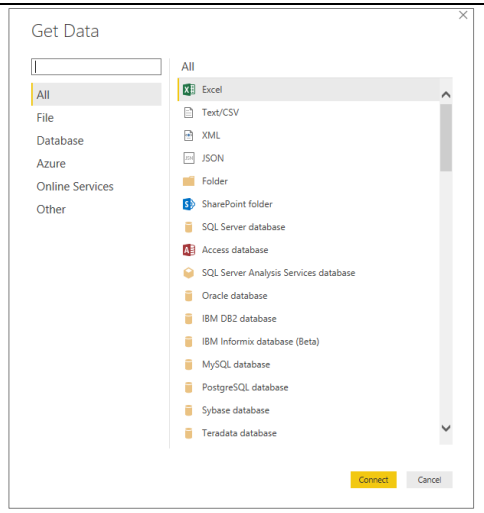
Module 2A Demo: Data Loading Basics

You are a buyer for a major bike retailer. As part of your job you are given Excel files that you need to analyze, but often these files include too much information. You would like to use Power BI to parse one of the files you are given and return just the data required.

Module Requirements

1. Import results from the Products table in the Excel spreadsheet found here: C:\Dax Boot Camp\Data & Module Resources\Module 02\Source Data.xlsx
2. Remove columns that are not necessary. You should only have the columns ProductAlternateKey, EnglishProductName, Color, and ListPrice.
3. Rename the remaining columns to more user-friendly names like Product Number, Name, Color and List Price.
4. Load the results into Data Model.

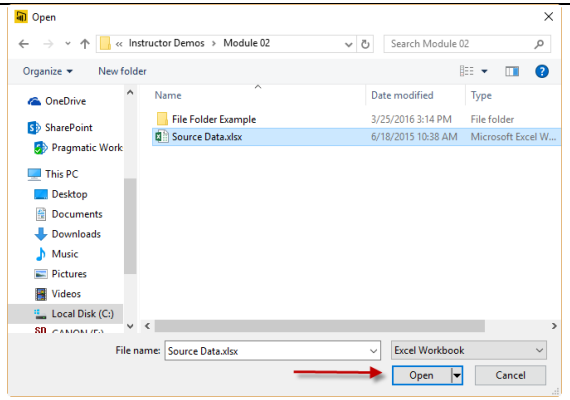
If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Launch the **Power BI Desktop** application. 2. When the application opens, close the startup screen, so you see just the design Report design surface. 3. To begin, bring data into Power BI Desktop. You must use the **Get Data** button on the **Home** ribbon. If you hit the down arrow on the **Get Data** button, it gives you a small list of some of the most common connectors. But, if you click the button above the arrow, it gives you the full list of all connectors. For this example, click the button, so that it returns a full list of connectors.	
4. In the **Get Data** window, you can either search for the type of connector you need by typing in the search bar or browse the list to find what you need. Choose the **Excel** connector and then click **Connect**.	

5. You will then be prompted to select the Excel file you wish to use as a data source. Navigate to and select the following file:

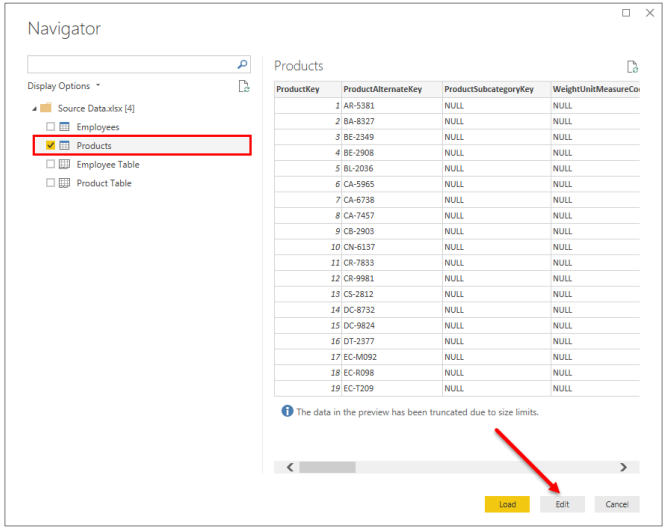
   C:\Dax Boot Camp\Data\Module 02\Source Data.xlsx

   Once you have selected this file, click **Open.**



6. This will launch the **Navigator** window. This shows all the tables and spreadsheets that are part of this workbook. The icon next to the object indicates whether it is a table or spreadsheet. Notice that this workbook has both spreadsheets and tables you can select from.

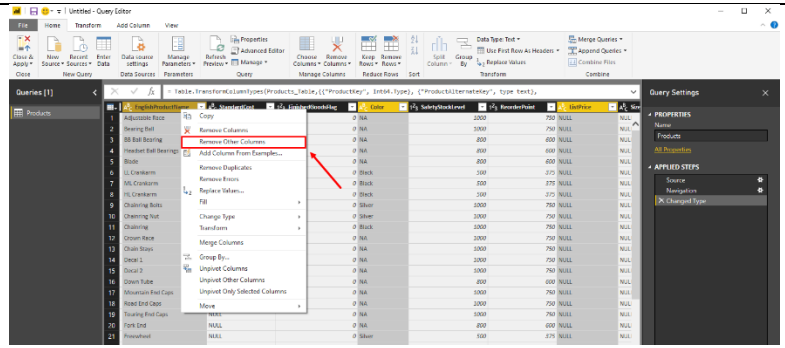
   Select the table called **Products** and then select **Edit**. This will launch the **Query Editor** where we can start to apply transformations to our dataset.



7. This dataset has a few extra columns that are not necessary.

   To remove the columns you do not need, multi-select(ctrl + click) the following columns:
   - **ProductAlternateKey**
   - **EnglishProductName**
   - **Color**
   - **ListPrice**

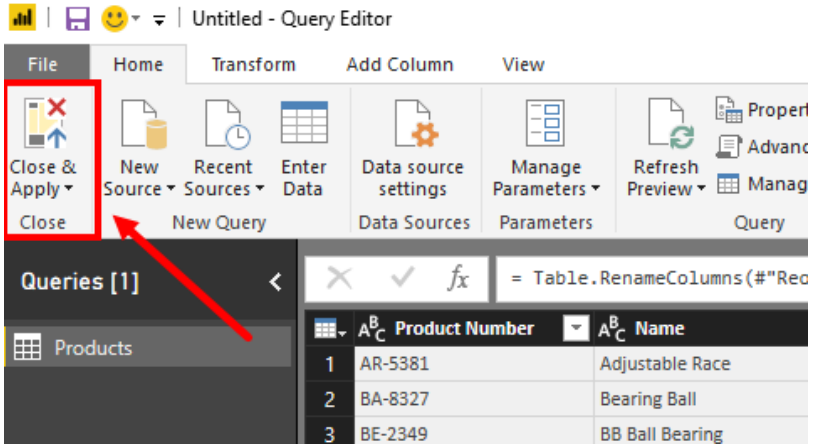
   Then right-click on one of the selected column headers and select **Remove Other Columns**, which should leave only the four that you selected.



8. Next, rename the remaining columns with more appropriate friendly names. To rename a column, you can either right-click on the column header and select **Rename** or simply double-click on the column header and type a new name over the old name.

   Rename the columns to have the following names:
   - **Product Number**
   - **Name**
   - **Color**
   - **List Price**

   Note there is no change to the **Color column.**



9. This completes the first basic data import example. To load the results of the query, click **Close & Apply** in the **Home** ribbon.
   Selecting this will load the results into the Data Model, which we will talk more about in a later module.

   Click **Save** and name the .pbix file **Module 02A**.

Notes	

Module 3: Data Modeling- Data Structure

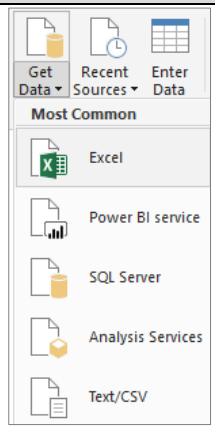
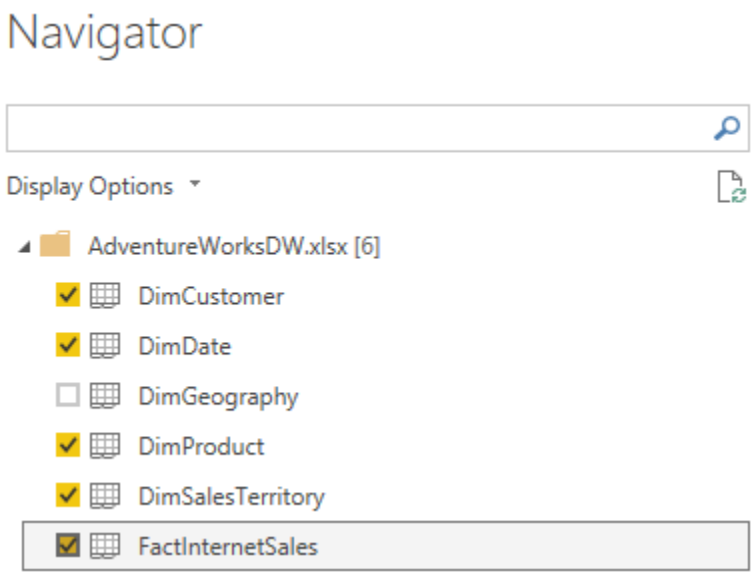
Module 3A Demo: Importing Data into the Data Model

You work for a retail bike store and are creating your first data model based on your company's sales data. The first step is to import the data from the appropriate data source.

Module Requirements

1. Import the Adventure Works sales data into Power BI Desktop. The data can be located here C:\Dax Boot Camp\Data & Module Resources\Databases\AdventureWorksDW.xlsx.

If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Launch the **Power BI Desktop** application. 2. When the application opens, close the startup screen, so you see just the Report design surface. 3. To begin bringing data into Power BI Desktop, you must use the **Get Data** button on the **Home** ribbon. 4. Click the down arrow on the **Get Data** button and select **Excel**.	
5. This will launch a File Explorer window where you will need to navigate to and select the **AdventureWorksDW.xlsx** file in the following location: **C:\Dax Boot Camp\Data\Databases\AdventureWorksDW.xlsx** Once you have selected this file, click **Open**. 6. Select the following tables and then select **Load**: **DimCustomer** **DimDate** **DimProduct** **DimSalesTerritory** **FactInternetSales**	

7. You also need to import data from another Excel file.

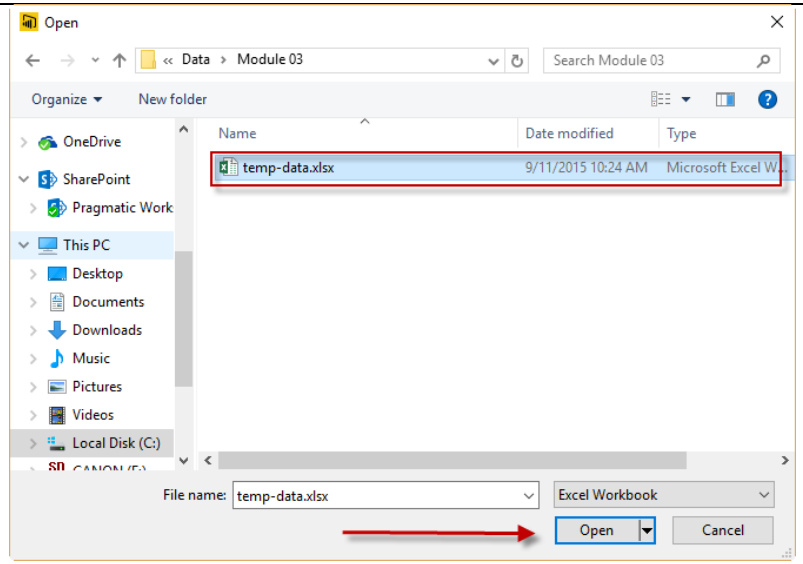
   Click the **Get Data** button on the **Home** ribbon.

8. From the **Get Data** window, select **Excel** and then click **Connect**.

9. This will launch a File Explorer window where you will need to navigate to and select the **temp-data.xlsx** file in the following location:

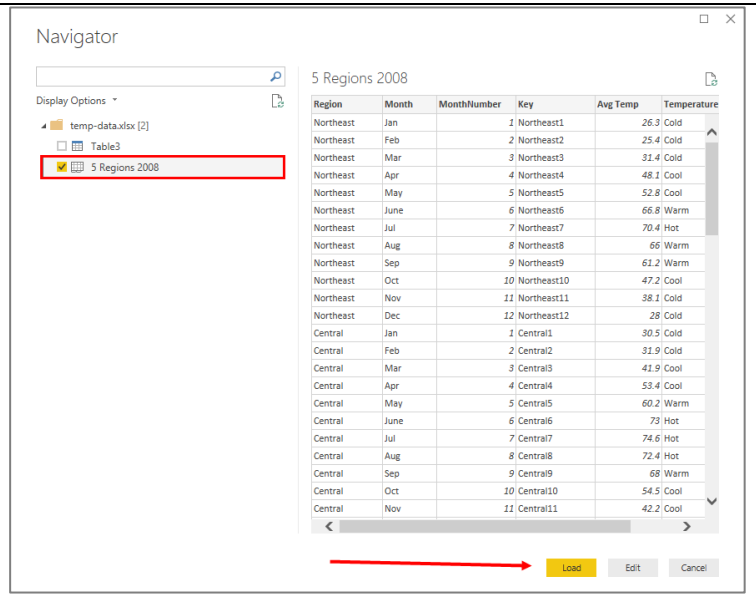
   C:\Dax Boot Camp\Data & Module Resources\Module 03\temp-data.xlsx

   Once you have selected this file, click **Open**.



10. Select **5 Regions 2008** and then click **Load**.

11. Save your file as **Module 03A.pbix**. You have finished this lab exercise.



Notes	

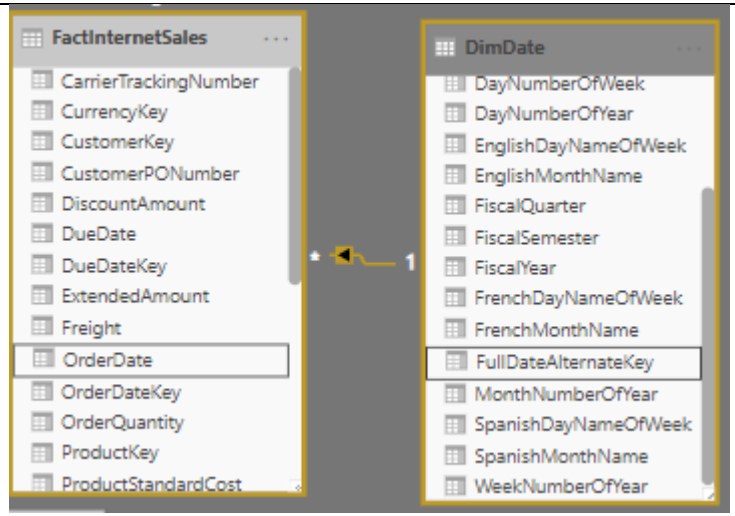
Module 3B Demo: Architecting a Data Model with the Power BI Desktop

With the data now imported into the model, it is time to start adding enhancements like relationships and hierarchies. Take the model started in the previous model and add new features that make it easier to manage and use in the reporting layer.

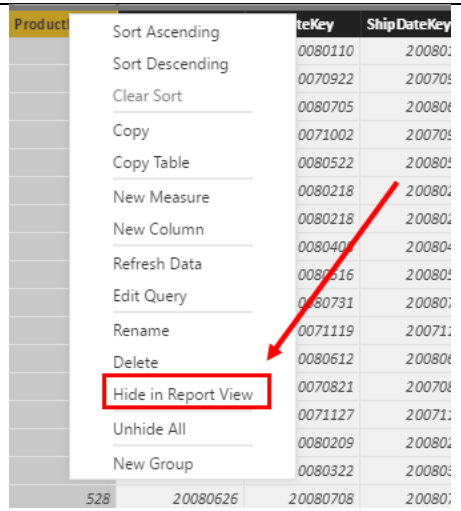
Module Requirements

1. Create the relationships between the FactInternetSales and the corresponding tables for performance.
2. Rename any tables and columns to add spaces where appropriate.
3. Hide columns and tables that aren't necessary.
4. Create a hierarchy in the Date table
5. Ensure all fields in the date table are set not to summarize the values. This is needed on all numeric columns.

If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the **Module 3A** file from your last module. The completed **Module 3A** file can be found in the folder **C:\Dax Boot Camp\Completed Labs\Module 03\Module 03A.pbix**. 2. Click on the **Model** view on the left-hand side of the screen.	
3. Draw relationships between each key column. This may require you to move the tables around on the relationship view to make this easier. Start by creating a relationship between **FactInternetSales** and the **DimDate** table. Click on **OrderDate** in the **FactInternetSales** table and drag and drop it on top of the **FullDateAlternateKey** column in the **DimDate** table.	

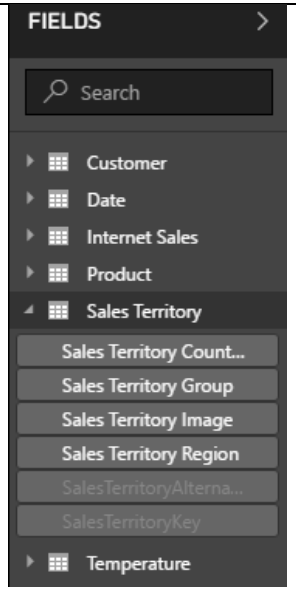
4. Switch over to the **Data** view on the left-hand side. Select the **FactInternetSales** table and hide all the **key** columns (Columns that have the word key in the field name) by right-clicking each column and selecting **Hide in Report View**.



5. Right-click on each field and select **Rename**. Make sure to do the following when renaming the fields:
   - Add spaces where appropriate
   - Remove either **Dim** or **Fact** prefix
   - Rename **5 Regions 2008** to **Temperature**

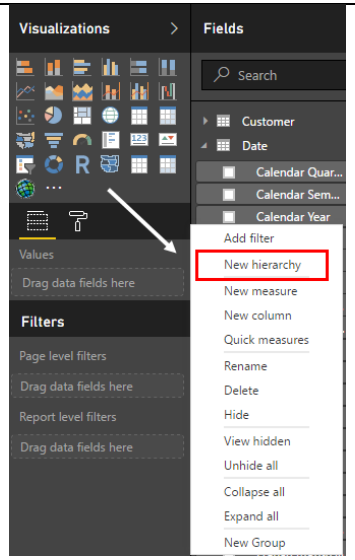
6. Hide each of the key columns in the rest of the tables by right-clicking on them and selecting **Hide in Report View**. Keys columns usually are not something you would place on a report, so it makes sense to hide them here. (Except the **Full Date Alternate Key** in the Date table)

7. Go back through each table and add spaces where appropriate to the column names to give a more user-friendly experience. This may take a few moments, but later labs depend on you renaming each field appropriately. Do not worry about renaming the hidden columns.



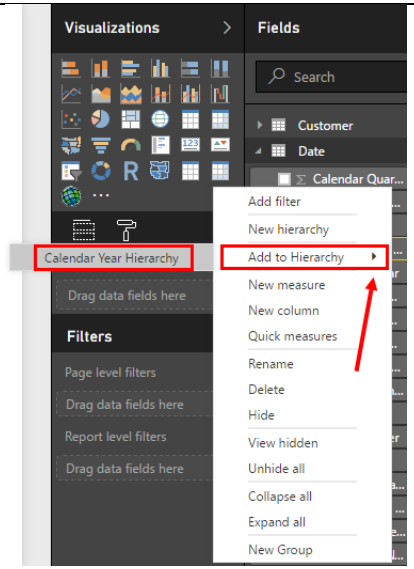
8. Next, go to the **Report** view to build a hierarchy on the **Date** table. Go to the **Fields** list on the right side of your screen and expand the **Date** table.

9. Right-click on the **Calendar Year** field and select **New hierarchy**.

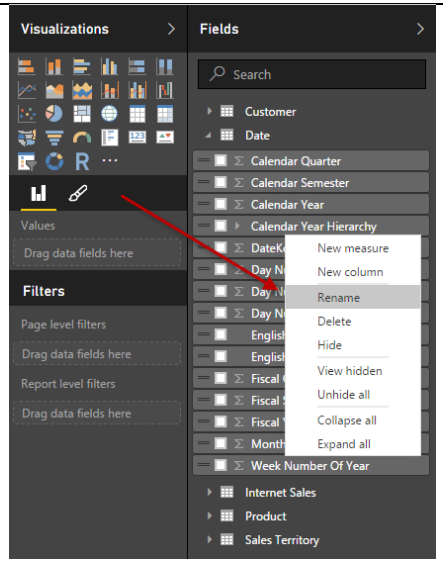


10. Then right-click on **Calendar Quarter** and select **Add to Hierarchy**> **Calendar Year Hierarchy**

11. Repeat this step for both **English Month Name** and **Full Date Alternate Key**.

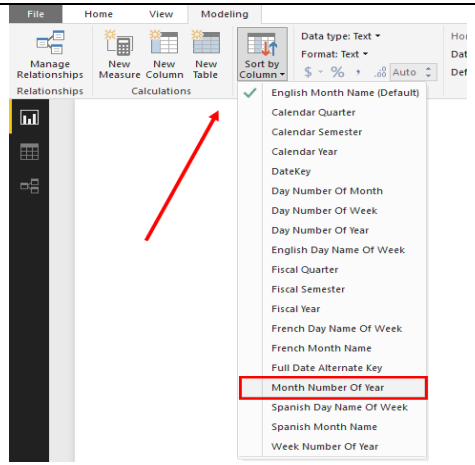


12. Right-click on the **Calendar Year Hierarchy** and select **Rename**. Name the hierarchy **Date Drilldown**.



13. An issue that often needs to be corrected in your data model is the sort order of the fields you store. For example, by default, the **English Month Name** column will sort alphabetically instead of chronologically. To fix this issue, you must change the **Sort by Column** on the necessary fields.

    For this example, go the **Fields** list and select the **English Month Name** field from the **Date** table. With this field selected, go to the **Modeling** Ribbon on the top of your screen and click the **Sort by Column** property. Choose the **Month Number Of Year**, which will sort the month by the numbers 1, 2, 3, etc… instead of April, August, December, etc…

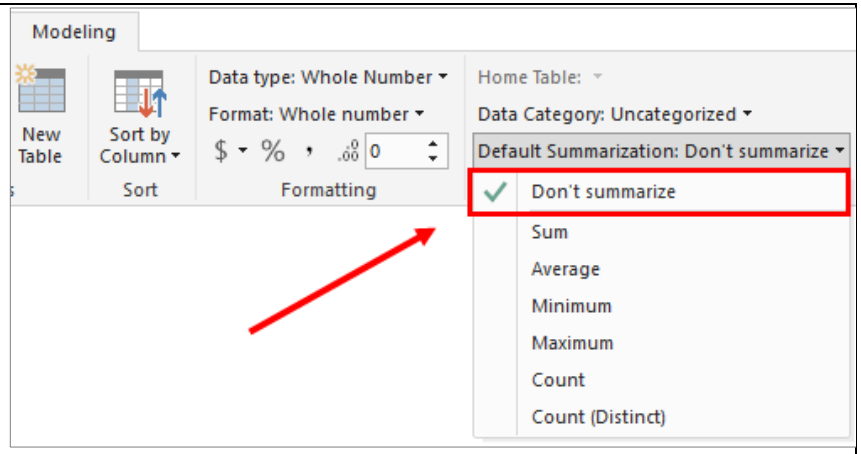


14. Another issue to be aware of is Power BI will automatically aggregate any field that is that has a numeric data type. Often these fields are aggregated when they should not be. For example, would you ever SUM the numeric field **Calendar Year**? The answer is obviously no. To fix this, change the **Default Summarization** property.

    Expand the **Date** table in the field list. First, select the **Calendar Quarter** field. Next, go to the **Modeling** Ribbon and change the property called **Default Summarization** to **Don't Summarize**.

15. Repeat these steps for the **Calendar Quarter**, **Calendar Semester**, **Calendar Year**, **Day Number of Month**, **Day Number of Week**, **Day Number of Year**, **Fiscal Quarter**, **Fiscal Semester**, **Fiscal Year**, **Month Number Of Year**, and **Week Number Of Year** fields.

16. Save your file as **Module 03B.pbix**. You have finished this lab exercise.



Notes

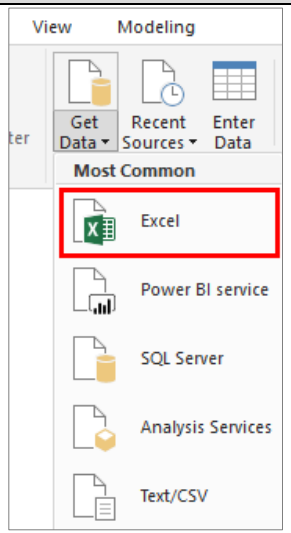
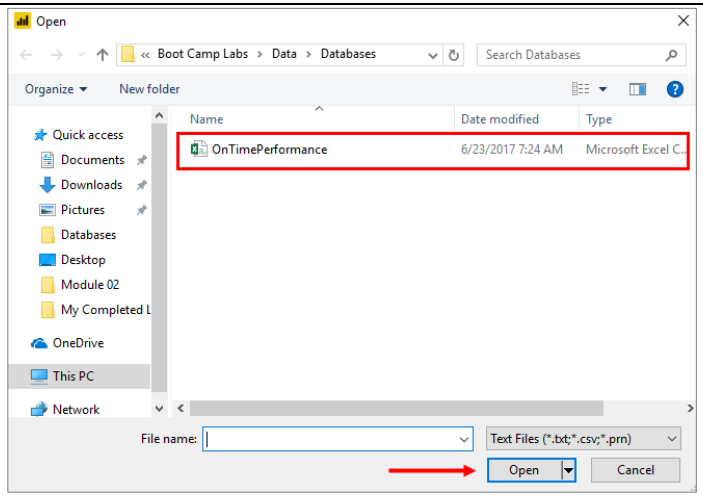
Module 3C Lab: Importing Data into the Data Model

All modules that are labeled "Lab" are labs that will not be done with the guidance of an instructor.

You are a data analyst for a major airline company. The business intelligence department in the company you work for has just completed a data mart and would like for you to begin ensuring that it will answer the proper questions.

Module Requirements

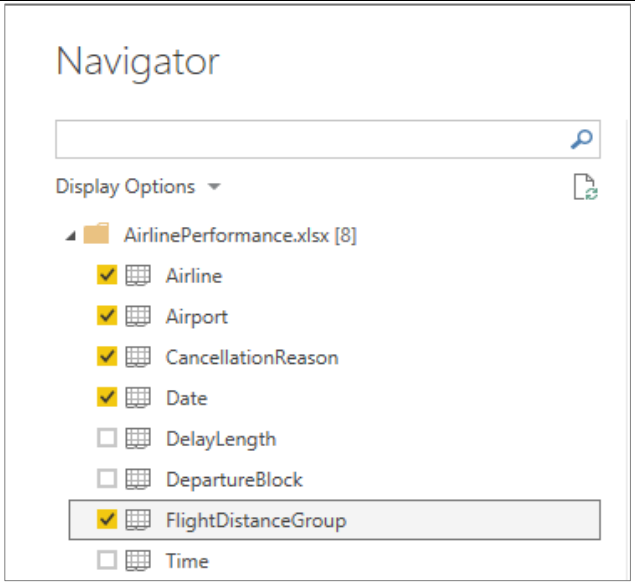
1. Import the Airline Performance data into Power BI Desktop. The data is located in two files. The first file is found here C:\Dax Boot Camp\Data & Module Resources\Databases\AirlinePerformance.xlsx. The second file is found here C: \Dax Boot Camp\Data & Module Resources\Databases\OnTimePerformance.csv
2. For the AirlinePerformance.xlsx we only need the following tables:
    a. Airline
    b. Airport
    c. CancellationReason
    d. Date
    e. FlightDistanceGroup

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Start by opening the **Module 03C (Start)** file found in the folder **C:\Dax Boot Camp\Completed Labs\Module 03\Module 03C (Start).pbix**. 2. Select the down arrow on the **Get Data** button from the **Home** tab in the ribbon at the top of the screen and then select **Excel.**	
3. This will launch a File Explorer window where you will need to navigate to and select the **AirlinePerformance.xlsx** file in the following location: **C:\Dax Boot Camp\Data\Databases\AirlinePerformance.xlsx** Once you have selected this file click **Open**	

4. Select the following tables from the **Navigator** window that has launched.

   Airline, **Airport**, **CancellationReason**, **Date**, and **FlightDistanceGroup**

5. Click **Load** to begin the import. This may take a few minutes to import depending on the resources of your machine.



6. Next, we need to import data from the file **OnTimePerformance.CSV**.

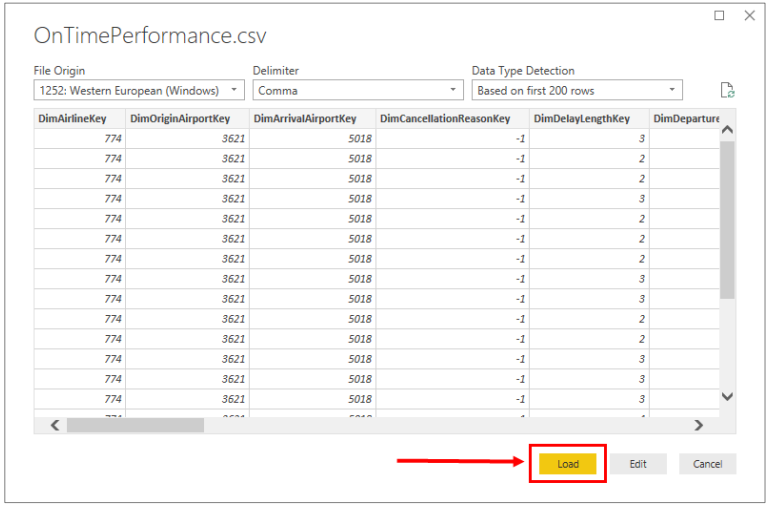
7. From the **Get Data** drop-down select **Text/CSV**.

   This will launch a File Explorer window where you will need to navigate to and select the file **OnTimePerformance.csv** from the following location:

   C:\Dax Boot Camp\Data\Databases\OnTimePerformance.csv

   Once you have selected this file, click **Open** and then **Load**.

8. Save your file as **Module 03C.pbix**. You have finished this lab exercise.



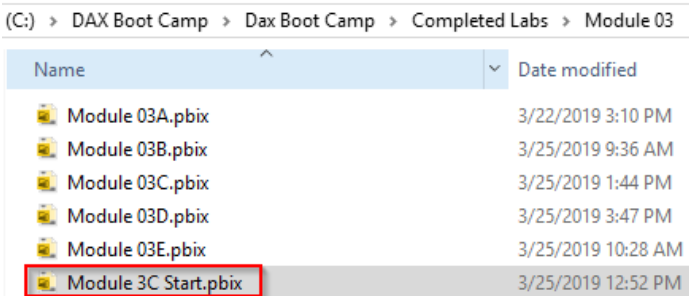
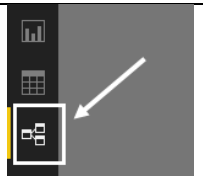
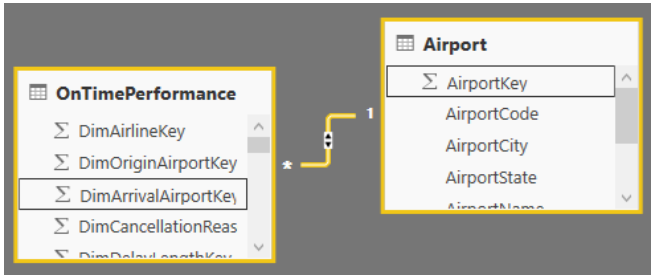
Module 3D Lab: Architecting a Data Model with the Power BI Desktop

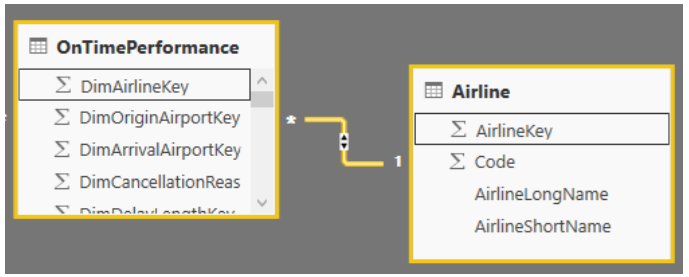
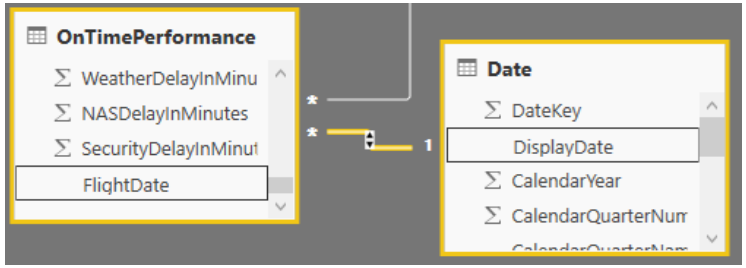
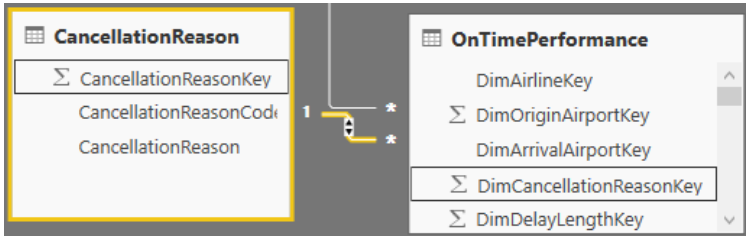
All modules that are labeled "Lab" are labs that will not be done with the guidance of an instructor.

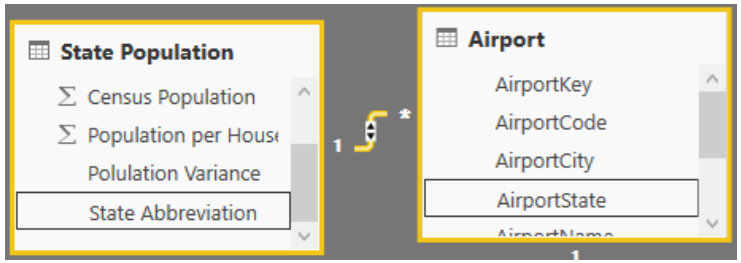
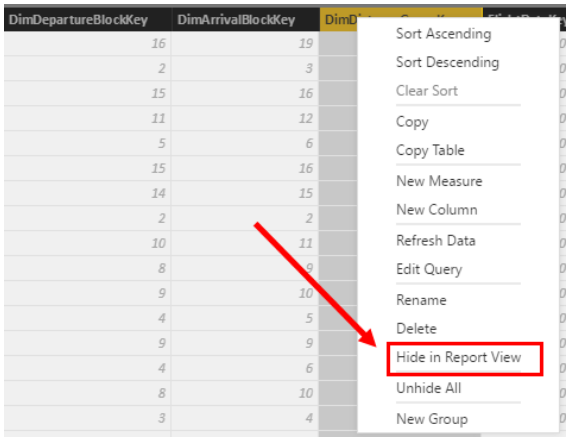
Once data is imported into a Data Model there are several usability enhancements that can be made so when it comes time to build reports you have everything you need and nothing you do not. Take the model started in the previous model and add new features that make it easier to manage and use in the reporting layer.

Module Requirements

1. Create the relationships between the OnTimePerformance and the corresponding tables for performance.
2. Rename any tables and columns to add spaces where appropriate.
3. Hide columns and tables that aren't necessary.
4. Create hierarchies on the Date and Airport tables
5. Ensure all fields in the date table are set not to summarize the values. This is needed on all numeric columns.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the **Module 3C** file from your last module. The completed **Module 3C** file can be found in the folder **C:\Dax Boot Camp\Completed Labs\Module 03\Module 03C Start.pbix**.	
2. Click on the **Relationships** view on the left-hand side of the screen.	
3. Draw relationships between each key column. This may require you to move the tables around on the relationship view to make this easier. Start by creating a relationship between **OnTimePerformance** and the **Airpor**t table. Click on **DimArrivalAirportKey** in the **OnTimePerformance** table and drag and drop it on top of the **AirportKey** column in the **Airport** table.	

4. Draw relationships between the **OnTimePerformance** and **Airline** table. Click on **DimAirlineKey** in the **OnTimePerformance** table and drag and drop it on top of the **AirlineKey** column in the **Airline** table.	
5. Draw relationships between the **OnTimePerformance** and **Date** table. Click on **FlightDate** in the **OnTimePerformance** table and drag and drop it on top of the **DisplayDate** column in the **Date** table.	
6. Draw relationships between the **OnTimePerformance** and **CancellationReason** table. Click on **DimCancellationReasonKey** in the **OnTimePerformance** table and drag and drop it on top of the **CancellationReasonKey** column in the **CancellationReason** table.	
7. Draw relationships between the **OnTimePerformance** and **FlightDistanceGroup** table. Click on **DimDistanceGroupKey** in the **OnTimePerformance** table and drag and drop it on top of the **DistanceGroupKey** column in the **FlightDistanceGroup** table.	

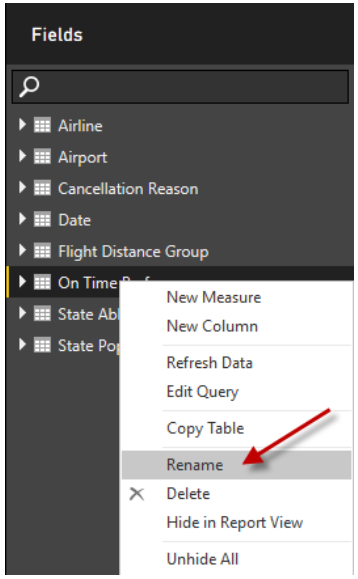
8. Last, create a relationship based off what was done in previous exercises. Draw relationships between the **Airport** and **State Population** table. Click on **AirportState** in the **Airport** table and drag and drop it on top of the **State Abbreviation** column in the **State Population** table.	
9. Switch over to the **Data** view on the left-hand side. Select the **OnTimePerformance** table and hide all the key columns (Columns that have the word key in the field name) by right-clicking each column and select **Hide in Report View**.	

10. Right-click on each table and select **Rename**. Add spaces where appropriate to the table names to give a more user-friendly name. You can do this on the right-hand side of the screen in the fields list.

11. Hide each of the key columns in the rest of the tables by right-clicking on them and selecting **Hide in Report View**. Keys columns usually are not something you would put in a report.

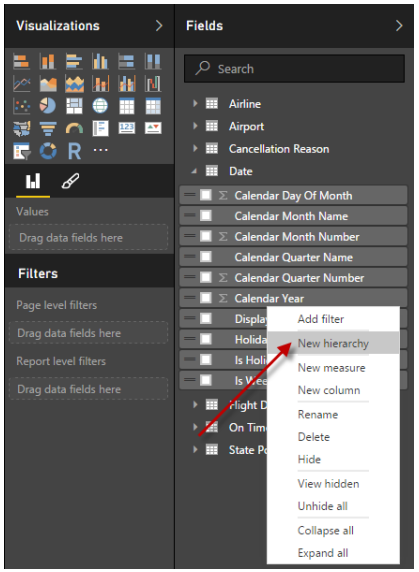
12. Go back through each table and add spaces where appropriate to the column names to give a more user-friendly experience. This may take a few moments, but later labs depend on you renaming each field appropriately. Do not worry about renaming the hidden columns.

13. Also, make sure that you hide the entire **State Abbreviations** table by right-clicking on the table name and selecting **Hide in Report View.**



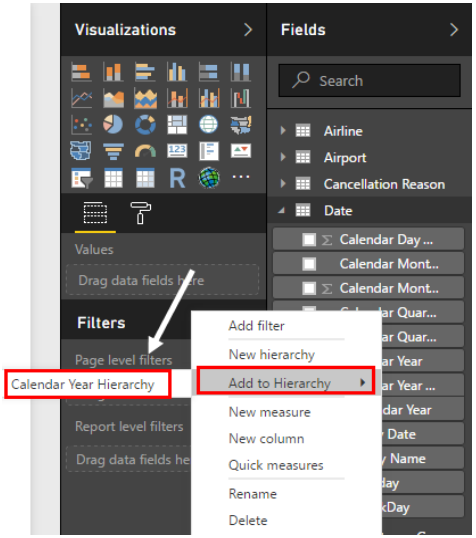
14. Next, go to the **Report** view to build a few hierarchies on this model. Go to the **Fields** list on the right side of your screen and expand the **Date** table.

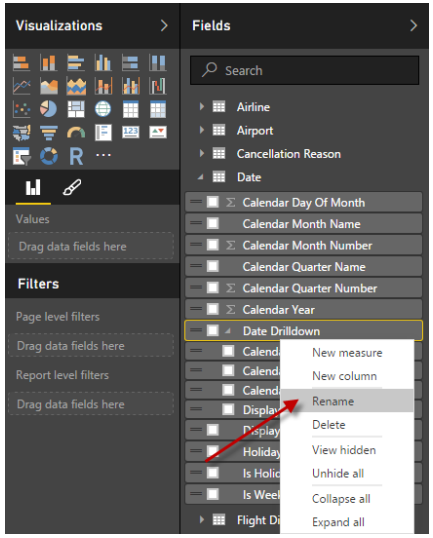
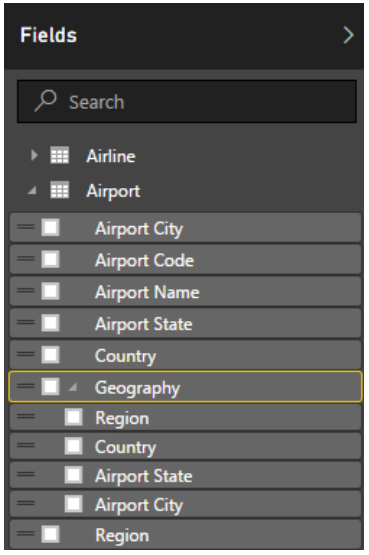
15. Right-click on the **Calendar Year** field and select **New hierarchy**.



16. Then right-click on **Calendar Quarter Name** and select **Add to Hierarchy**> **Calendar Year Hierarchy**

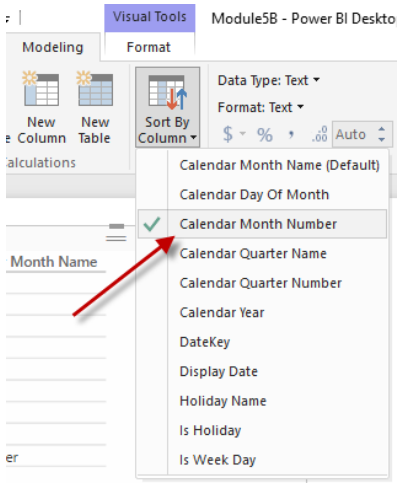
17. Repeat this step for both **Calendar Month Name** and **Display Date**.



18. Right-click on the **Calendar Year Hierarchy** and select **Rename**. Name the hierarchy **Date Drilldown**.	
19. Repeat these steps on the **Airport** table use the fields **Region >**, select **Country >**, select **Airport State >**, and then select **Airport City**. Rename the hierarchy **Geography**.	

20. An issue that often needs to be corrected in your data model is the sort order of the fields you store. For example, by default, the **Calendar Month Name** column will sort alphabetically instead of chronologically. To fix this issue, you must change the **Sort by Column** in the necessary fields.

    For this example, go the **Fields** list and select the **Calendar Month Name** field from the **Date** table. With this field selected, go to the **Modeling** Ribbon on the top of your screen and click the **Sort by Column** property. Choose the **Calendar Month Number**, which will sort the month by the numbers 1, 2, 3, etc… instead of April, August, December, etc…

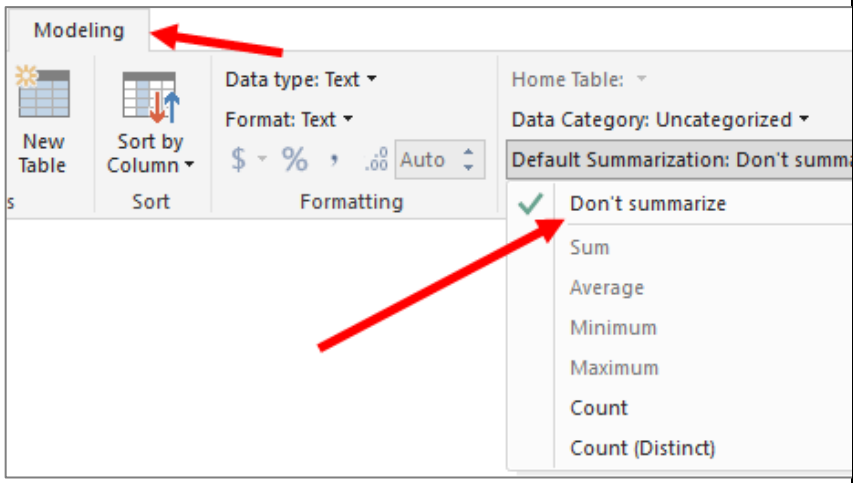


21. Another issue to be aware of is Power BI will automatically aggregate any field that is that has a numeric data type. Often these fields are aggregated when they should not be. For example, would you ever SUM the numeric field Calendar Year? The answer is obviously no. The way to fix this is by changing the **Default Summarization** property.

    This example will also require that you expand the **Date** table in the field list. Start by selecting the **Calendar Day Of Month** field. Next, go to the **Modeling** Ribbon on the top of your screen and change the property called **Default Summarization** to **Don't Summarize**.

22. Repeat these steps for the **Calendar Day of Month**, **Calendar Month Number**, **Calendar Quarter Number**, and **Calendar Year** fields.

23. Save your file as **Module 03D.pbix**. You have finished this lab exercise.



Module 4: Data Modeling- Calculated Columns

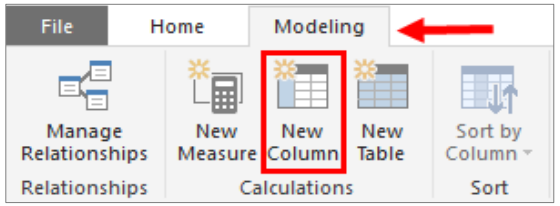
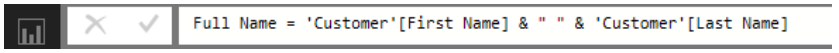
Module 4A Demo: Creating Calculated Columns

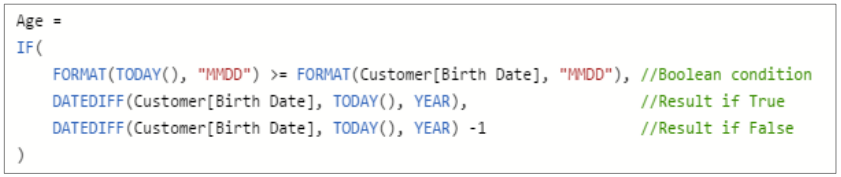
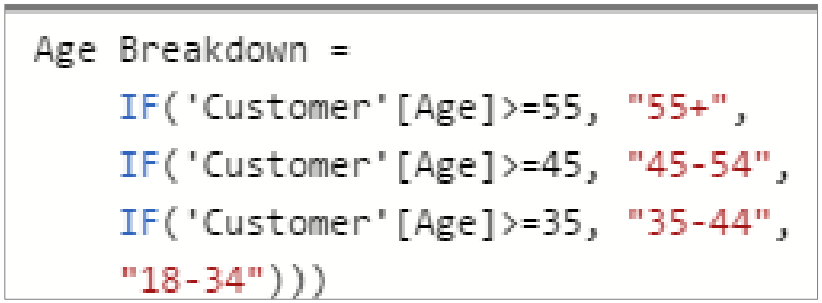
With the structure of the data model finalized it is now time to add in several calculated columns to provide new fields that can be used in the report. This module will walk you through creating several new fields based off DAX formulas you write.

Module Requirements

1. Create Calculated Columns that do the following:
    a. Full Name: a combination of the First and Last Name columns
    b. Month Year: a month number and year concatenated together and separated by a hyphen
    c. Age: the difference between the customer's birth date and the current date
    d. Age Breakdown: place customers in age buckets of 55+, 45-54, 35-44, and 18-34
2. Create navigation columns to help join in the temperature data.

If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the **Module 3B** file from your last module. The completed **Module 3B** file can be found in the folder **C:\Dax Boot Camp\Completed Labs\Module 03\Module 03B.pbix**. 2. Click on the **Data** view on the left-hand side of the screen.	
3. Select the **Customer** table from the **Fields** list. 4. Next, go up to the **Modeling** Ribbon on the top of your screen and select **New Column**.	
5. Use the formula bar and type the following code: **Full Name = 'Customer'[First Name] & " " & 'Customer'[Last Name]** This will create a column with the first and last name combined.	
6. Create another new Calculated Column on the Customer table by selecting the **New Column** button on the **Modeling** ribbon at the top of the screen. Type the following for the new formula:	

Customer Age = DATEDIFF('Customer'[Birth Date],TODAY(),YEAR)	
7. The previous calculation created to determine age is not always correct. This is because it simply subtracts the birth year from the current year, not taking into account the months and days. We are going to create a new calculation to fix this. Create another new Calculated Column on the Customer table by selecting the **New Column** button on the **Modeling** ribbon at the top of the screen. Type the following for the new formula: **Age = IF(FORMAT(TODAY(), "MMDD") >= FORMAT('Customer'[Birth Date], "MMDD"), DATEDIFF(Customer[Birth Date], TODAY(), YEAR), DATEDIFF(Customer[Birth Date], TODAY(), YEAR) -1)**	
8. Create another new Calculated Column on the **Customer** table by selecting the **New Column** button on the **Modeling** ribbon. Type the following for the new formula: **Age Breakdown = IF('Customer'[Age]>=55, "55+", IF('Customer'[Age]>=45, "45-54", IF('Customer'[Age]>=35, "35-44", "18-34")))** This creates a banding effect for our marketing department to be able to group customers.	

9. Navigate to the **Date** table next in the **Fields** list.

   Create another new Calculated Column on the **Date** table by selecting the **New Column** button on the **Modeling** ribbon. Type the following for the new formula:

   Month Year = RIGHT("0" & 'Date'[Month Number Of Year],2)&"-"&'Date'[Calendar Year]

10. By Default, this will not sort correctly. To correct it, select the new **Month Year** column and change the **Sort By Column** property in the **Modeling** Ribbon to **Calendar Year**.

```
Month Year =
RIGHT("0" & 'Date'[Month Number Of Year],2)
      "-" &
      'Date'[Calendar Year]
```

11. Next, to tie in the temperature data, which is currently not related in any other tables in the model, we must create a composite key value.

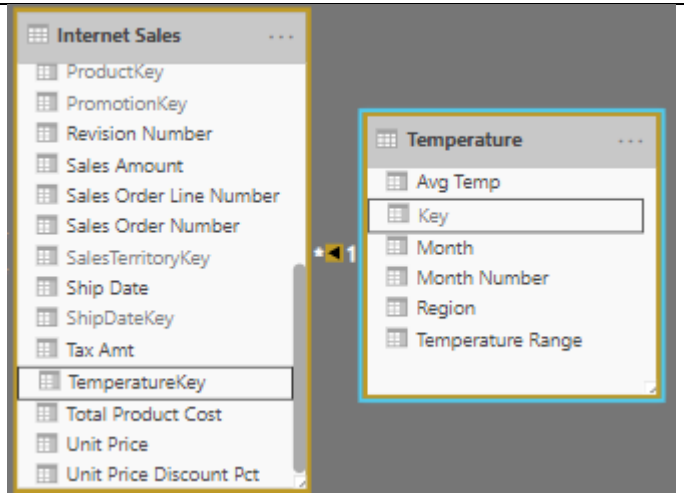
    Select the **Internet Sales** table in the **Fields** list and then click **New Column** in the **Modeling** Ribbon. Type the following for the new formula:

    TemperatureKey = RELATED('Sales Territory'[Sales Territory Region]) & RELATED('Date'[Month Number Of Year])

```
TemperatureKey =
RELATED('Sales Territory'[Sales Territory Region]
RELATED('Date'[Month Number Of Year])
```

12. With this new key column created, a relationship can now be created between **Internet Sales** and **Temperature**.

    Go to the **Relationship** view and draw the relationship between **Temperature** using the **Key** column and **Internet Sales** and **TemperatureKey**.



13. Next go back to the **Data** view and select the **Sales Territory** table in the **Fields** list.

14. Create a new calculated column in the **Sales Territory** table by selecting the **New Column** button on the **Modeling** ribbon. In the formula bar type:

    Sales By Region = SUMX(RELATEDTABLE('Internet Sales'), [Sales Amount])

    This gives you the sales by region, which we will use to determine which regions are considered High, Medium, or Low volume regions.

```
Sales By Region = SUMX(RELATEDTABLE('Internet Sales'),[Sales Amount])
```

15. Now we want to determine the most recent purchase date for each of our customers. We can do this using the functions MAXX and RelatedTable.

    Create a new Calculated Column on the **Customer** table. Type the following for the new formula:

Date of Most Recent Purchase = MAXX(RELATEDTABLE('Internet Sales'),'Internet Sales'[Order Date])	`Date of Most Recent Purchase = MAXX( RELATEDTABLE('Internet Sales'), 'Internet Sales'[Order Date] )`
16. Create another new Calculated Column on the **Sales Territory** table by selecting the **New Column** button on the **Modeling** ribbon. Type the following for the new formula: **Region Volume = SWITCH(TRUE(), 'Sales Territory'[Sales By Region] >= 6000000, "High", 'Sales Territory'[Sales By Region] >= 2000000, "Medium", 'Sales Territory'[Sales By Region] >= 0, "Low")** ****TIP** You can use Shift + Enter to put a return in your DAX formula.** 17. Save your file as **Module 04A.pbix**. You have finished this lab exercise.	`Region Volume = SWITCH( TRUE(), 'Sales Territory'[Sales By Region] >= 6000000, "High", 'Sales Territory'[Sales By Region] >= 2000000, "Medium", 'Sales Territory'[Sales By Region] >= 0, "Low")`

Notes	

Module 4B Lab: Creating Calculated Columns

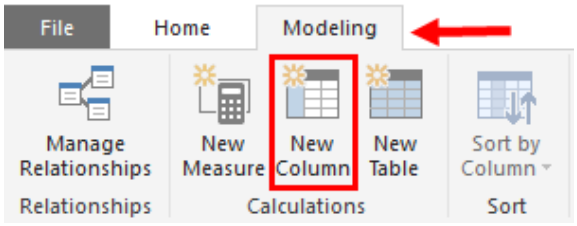
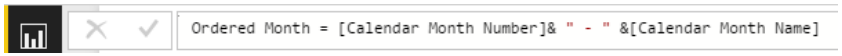
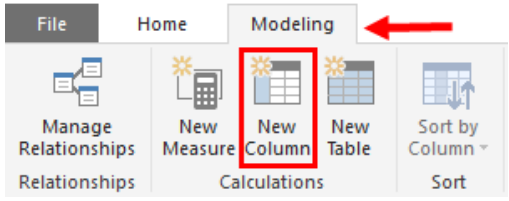
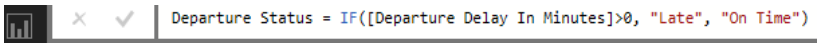
All modules that are labeled "Lab" are labs that will not be done with the guidance of an instructor.

Next, you would like to add several new fields in the On Time Performance and Airport tables. These new calculated fields will allow you to do things like analyzing whether a flight left on time or was late on departure or arrival. This will give you more functionality when it comes to building your reports later.

Module Requirements

1. Create Calculated Columns that do the following:
    a. Ordered Month: a combination of the month number and the month name together
    b. Departure Status: determines if a flight is on time or late
    c. Airport Flights: tells the number of flights per airport
    d. Airport Volume: categories each airport as high, medium or low volume based on the number of flights

If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the **Module 3D** file from your last module. The completed **Module 3D** file can be found in the folder **C:\Dax Boot Camp\Completed Labs\Module 03\Module 03D.pbix.** 2. Navigate to the **Data** view and select the **Date** table from the **Fields** list. Then click the **New Column** option under the **Modeling** ribbon at the top of the screen.	
3. Use the formula bar and type the following code: **Ordered Month= [Calendar Month Number] & " - " & [Calendar Month Name]** This will create a column with the month number and the month name separated by a hyphen.	
4. Select the **On Time Performance** table. Hide the remaining columns in the table except for **Departure Delay in Minutes**. 5. Next, create a new Calculated Column on the **On Time Performance** table by selecting the **New Column** button on the **Modeling** ribbon	
6. Name your column Departure Status, and type in the following formula: **Departure Status = IF([Departure Delay In Minutes]>0, "Late", "On Time")**	

7. Next, determine which airports are the busiest by creating a calculated column that returns High, Medium and Low volume air traffic. Select the **Airport** table and click **New Column** on the **Modeling** ribbon. Use the following formula to return the number of flights for each airport: **Airport Flights = COUNTX(RELATEDTABLE('On Time Performance'), 'On Time Performance'[Flight Number])**	``` Airport Flights = COUNTX(RELATEDTABLE('On Time Performance'), 'On Time Performance'[Flight Number]) ```
8. Now you can build calculations on top of other calculations. Click **New Column** again on the **Modeling** ribbon and use the following formula to create a grouping of the busiest airports: **Airport Volume = SWITCH(TRUE(), [Airport Flights] < 2000, "Low Volume", [Airport Flights] < 5000, "Medium Volume", "High Volume")** Save your file as **Module 04B.pbix**. You have finished this lab exercise.	``` Airport Volume = SWITCH(TRUE(), [Airport Flights] < 2000, "Low Volume", [Airport Flights] < 5000, "Medium Volume", "High Volume") ```

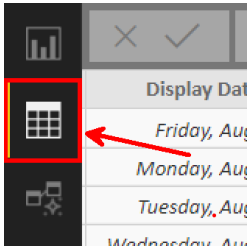
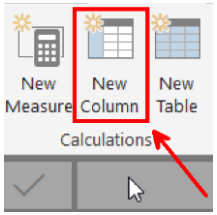
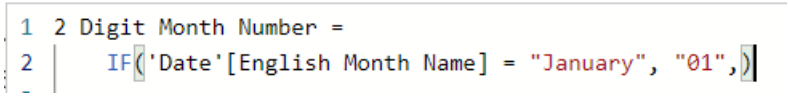
Module 4C Demo: Conditional Logic Functions

Inside of our current Date table, you want to add a new column of 2-Digit Month Numbers based on your English Month Name column. This can be done using IF or SWITCH statements.

Module Requirements

1. Create a new calculated column of 2-digit month numbers using nested IF statements
2. Recreate your column of 2-digit month numbers using a SWITCH statement.

If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the **Module 4A** file. The completed **Module 4A** file can be found in the folder **C:\DAX Boot Camp\Completed Labs\Module 4\Module 4A.pbix**. 2. Navigate to the **Data** view, by selecting it on the left side of the screen.	
3. Create a **New Calculated Column** in the **Date** table by clicking the **New Column** button in the **Modeling** ribbon.	
4. Write the beginning of the column as a single **IF** statement: **2 Digit Month Number =** **IF('Date'[English Month Name]="January", "01")**	

5. Complete the remaining column values by continuing with **nested IF** statements as follows:

2 Digit Month Number =
IF([English Month Name]="January", "01",
IF([English Month Name]="February", "02",
IF([English Month Name]="March", "03",
IF([English Month Name]="April", "04",
IF([English Month Name]="May", "05",
IF([English Month Name]="June", "06",
IF([English Month Name]="July", "07",
IF([English Month Name]="August", "08",
IF([English Month Name]="September", "09",
IF([English Month Name]="October", "10",
IF([English Month Name]="November", "11",
IF([English Month Name]="December", "12"
))))))))))))

Note: The resulting value is string (“01”) because using a whole number (01) would strip any leading zeros.

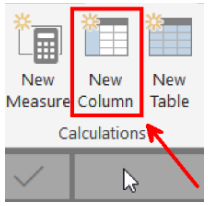
```
1  2 Digit Month Number =
2      IF('Date'[English Month Name] = "January", "01",
3      IF('Date'[English Month Name] = "February", "02",
4      IF('Date'[English Month Name] = "March", "03",
5      IF('Date'[English Month Name] = "April", "04",
6      IF('Date'[English Month Name] = "May", "05",
7      IF('Date'[English Month Name] = "June", "06",
8      IF('Date'[English Month Name] = "July", "07",
9      IF('Date'[English Month Name] = "August", "08",
10     IF('Date'[English Month Name] = "September", "09",
11     IF('Date'[English Month Name] = "October", "10",
12     IF('Date'[English Month Name] = "November", "11",
13     IF('Date'[English Month Name] = "December", "12"
14 ))))))))))))
```

6. Verify the new column in the **Date** table.

7. You have now created a new calculated column with nested IF statements.

8. Now let's try recreating this column to use a single **SWITCH** statement.

9. Create a new **calculated column** in the **Date** table.



10. Write your statement using SWITCH as follows:

```
2 Digit Month Number SW =
SWITCH(
  [English Month Name],
    "January", "01",
    "February", "02",
    "March", "03",
    "April", "04",
    "May", "05",
    "June", "06",
    "July", "07",
    "August", "08",
    "September", "09",
    "October", "10",
    "November", "11",
    "December", "12"
  )
```

```
1  2 Digit Month Number SW =
2  SWITCH('Date'[English Month Name],
3      "January", "01",
4      "Feburary", "02",
5      "March", "03",
6      "April", "04",
7      "May", "05",
8      "June", "06",
9      "July", "07",
10     "August", "08",
11     "September", "09",
12     "October", "10",
13     "November", "11",
14     "December", "12"
15 )
```

11. Verify your new column is correct.

2 Digit Month Number SW
07
07
07
07
07

12. You have now created calculated columns using the IF and SWITCH conditional logic functions 13. Save your file as **Module 04C.pbix**. You have finished this lab exercise.	

Notes	

Module 5: Data Modeling – Calculated Measures

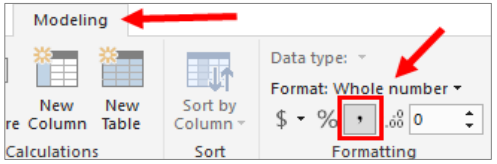
Module 5A Demo: Creating Calculated Measures

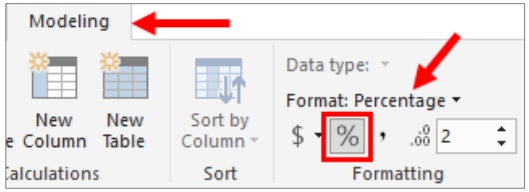
The next step is to create several Calculated Measures to provide additional metrics that can be used when building reports. In this module you will add several metrics to the model you have designed so far.

Module Requirements

1. Create Calculated Measures that do the following:
    a. Total Quantity
    b. Total Sales
    c. Total Cost
    d. Profit
    e. Profit Margin
    f. US Profit
    g. Total Transactions
    h. Count of Days

If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the **Module 4C** file from your last module. The completed **Module 4C** file can be found in the folder **C:\Dax Boot Camp\Completed Labs\Module 04\Module 04C.pbix**. 2. In this module, you will be creating several calculated measures. Measures can be created in either the **Report** or **Data** view. For this example, you will need to go to the **Data** view and then select the **Internet Sales** table.	
3. Go to the **Modeling** Ribbon and select **New Measure**. 4. In the formula bar, type: **Total Quantity = SUM('Internet Sales'[Order Quantity])**	`Total Quantity = SUM('Internet Sales'[Order Quantity])`
5. Apply formatting to this measure by selecting the comma icon in the **Modeling** Ribbon.	
6. Go to the **Modeling** Ribbon and select **New Measures**. 7. In the formula bar, type: **Total Sales = SUM('Internet Sales'[Sales Amount])** 8. You should also apply formatting to this measure by selecting the Format value of **$ English (United States)** in the **Modeling** Ribbon.	`Total Sales = SUM('Internet Sales'[Sales Amount])` 

9. Go to the **Modeling** Ribbon and select **New Measures**. 10. In the formula bar, type the following formula: **Total Cost = SUM([Total Product Cost])** 11. You should also apply formatting to this measure by selecting the Format value of **$ English (United States)** in the **Modeling** Ribbon.	`Total Cost = SUM([Total Product Cost])` 
12. Go to the **Modeling** Ribbon and select **New Measures**. 13. In the formula bar, type the following formula: **Profit = [Total Sales] - [Total Cost]** 14. You should also apply formatting to this measure by selecting the Format value of **$ English (United States)** in the **Modeling** Ribbon.	`Profit = [Total Sales] - [Total Cost]` 
15. Go to the **Modeling** Ribbon and select **New Measures**. 16. In the formula bar, type the following formula: **Profit Margin = DIVIDE([Profit],[Total Sales])** 17. You should also apply formatting to this measure by selecting the Format value of **Percentage** in the **Modeling** Ribbon.	`Profit Margin = DIVIDE([Profit],[Total Sales])` 
18. Go to the **Modeling** Ribbon and select **New Measures**. 19. In the formula bar, write the following formula:	`US Profit = CALCULATE([Profit], 'Sales Territory'[Sales Territory Country] = "United States")`

US Profit = CALCULATE([Profit],'Sales Territory'[Sales Territory Country] = "United States")

20. You should also apply formatting to this measure by selecting the Format value of **$ English (United States)** in the **Modeling** Ribbon.

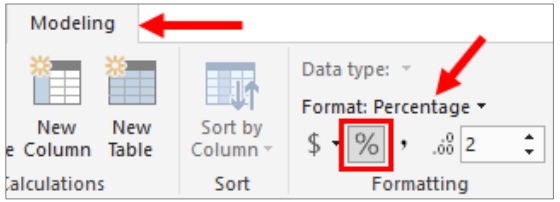
21. Go to the **Modeling** Ribbon and select **New Measure**.

22. In the formula bar, type the following formula:

Percent US Profit = DIVIDE([US Profit],[Profit])

```
Percent US Profit = DIVIDE([US Profit],[Profit])
```

23. You should also apply formatting to this measure by selecting the Format value of **Percentage** in the **Modeling** Ribbon.



24. Save your file as **Module 05A.pbix**. You have finished this lab exercise.

25. Go to the **Modeling** Ribbon and select **New Measure**.

26. In the formula bar,

Total Transactions = COUNT('Internet Sales'[Order Date])

```
1 Total Transactions =
2 COUNTX('Internet Sales', 'Internet Sales'[Order Date])
```

27. Next, rewrite the calculation COUNTX('Internet Sales'[Order Date])

```
1 Total Transactions =
2 COUNTROWS('Internet Sales')
```

28. Next, rewrite the calculation COUNTROWS('Internet Sales')

29. Now create a measure for distinct date count.

30. Create a new measure, in the formula bar, type:

Count of Days = DISTINCTCOUNT('Internet Sales'[Order Date])

```
1 Count of Days = DISTINCTCOUNT('Internet Sales'[Order Date])
```

Notes	

Module 6: CALCULATE

Module 6A Demo: CALCULATE – Percent of Total

Sometimes we don't want the automatic filtering in the data model to take place. In these instances, we can use the CALCULATE functions to override or ignore the default behavior.

Module Requirements

1. Open the pbix file Module 05B found here C:\Dax Bootcamp\Completed Labs\Module 05\Module 05A.pbix.
2. Create a calculation that calculates the percent of Total Sales per country

Sales Territory Country	Total Sales	Total Sales all Countries	% of Total
United States	$9,389,789.51	$29,358,677.22	31.98%
Australia	$9,061,000.58	$29,358,677.22	30.86%
United Kingdom	$3,391,712.21	$29,358,677.22	11.55%
Germany	$2,894,312.34	$29,358,677.22	9.86%
France	$2,644,017.71	$29,358,677.22	9.01%
Canada	$1,977,844.86	$29,358,677.22	6.74%
Total	**$29,358,677.22**	**$29,358,677.22**	**100.00%**

For step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the **Module 5A** file from your last module. The completed **Module 5A** file can be found in the folder **C:\Dax Boot Camp\Completed Labs\Module 05\Module 05A.pbix**. 2. If you are not already looking at the Report view, then select it on the left side of the screen.	
3. In order to calculate a percent of total, we need to divide the current value by the total. This is not quite as easy as it should be and it's due to filter context.	VALUE / TOTAL VALUE = PERCENT OF TOTAL
4. The first thing we need to accomplish is calculating the total value, which in this context is total sales for all the available countries. **Total Sales all Countries =** **CALCULATE(** **[Total Sales],** **ALL('Sales Territory'[Sales Territory Country]))** 5. Change the formatting to US Currency 6. When this field is added to a visual, it will always show the total sales for all the countries aggregated together.	``` Total Sales all Countries = CALCULATE([Total Sales], ALL('Sales Territory'[Sales Territory Country])) ```

7. This gives us the total sales for all our countries in our dataset. The CALCULATE function is ignoring any filters that are being applied to the **country** attribute.

   The calculation itself isn't entirely useful when added to a visual, but if we use it within another calculation is when it becomes viable.

Sales Territory Country	Total Sales	Total Sales all Countries
Australia	$9,061,000.58	$29,358,677.22
Canada	$1,977,844.86	$29,358,677.22
France	$2,644,017.71	$29,358,677.22
Germany	$2,894,312.34	$29,358,677.22
NA		$29,358,677.22
United Kingdom	$3,391,712.21	$29,358,677.22
United States	$9,389,789.51	$29,358,677.22
Total	**$29,358,677.22**	**$29,358,677.22**

8. Next, remove the blank value that now appears for the Country NA. Modify the DAX measure to look like the following:

```
Total Sales all Countries =
IF(
    [Total Sales] = BLANK(),
    BLANK(),
    CALCULATE(
        [Total Sales],
        ALL('Sales Territory'[Sales Territory Country])))
```

```
1 Total Sales all Countries =
2 IF(
3     [Total Sales] = BLANK(),
4     BLANK(),
5     CALCULATE(
6         [Total Sales],
7         ALL('Sales Territory'[Sales Territory Country])))
8
```

9. Create a new calculated measure:

10. **% of Total =**
    DIVIDE(
    [Total Sales],
    [Total Sales all Countries])

11. Change the formatting to percentage

```
% of Total =
DIVIDE(
    [Total Sales],
    [Total Sales all Countries])
```

12. Add **[% of Total]** to the table on the existing report.

    The measure [Total Sales all Countries] is only in the table for validation purposes, it can be removed, and the calculation will still work as expected.

Sales Territory Country	Total Sales	Total Sales all Countries	% of Total
United States	$9,389,789.51	$29,358,677.22	31.98%
Australia	$9,061,000.58	$29,358,677.22	30.86%
United Kingdom	$3,391,712.21	$29,358,677.22	11.55%
Germany	$2,894,312.34	$29,358,677.22	9.86%
France	$2,644,017.71	$29,358,677.22	9.01%
Canada	$1,977,844.86	$29,358,677.22	6.74%
Total	**$29,358,677.22**	**$29,358,677.22**	**100.00%**

Notes	

Module 6B Lab: CALCULATE – All Time Sales

This lab will not have step-by-step instructions

Your team has asked that you create a new calculated measure that returns "Total Sales" for all time, this calculated measure should ignore any filters placed on the date.

Module Requirements

1. Open the pbix file Module 06B found here C:\Dax Boot Camp\Class Labs\Module 06\Module 06B.pbix
2. Create a new calculated measure called [Total Sales(All Time)].
3. The new measure should ignore any filters applied to the date table.
4. **BONUS – Modify the measure to remove the years 2009 and 2010.

Calendar Year	Total Sales	Total Sales (All Time)
2005	$3,266,373.66	$29,358,677.22
2006	$6,530,343.53	$29,358,677.22
2007	$9,791,060.30	$29,358,677.22
2008	$9,770,899.74	$29,358,677.22
2009		$29,358,677.22
2010		$29,358,677.22
Total	**$29,358,677.22**	**$29,358,677.22**

Hints

1. Remember that CALCULATE allows you to override or ignore an existing filter context.
2. Remember to reuse calculated measures that have already been created.
3. With the ALL function you can apply a filter to a specific column, or an entire table.
4. The completed lab can be found in the completed labs folder: C:\Dax Boot Camp\Completed Labs\Module 06\Module 06B.pbix
5. The completed DAX code can be found in the completed labs folder: C:\Dax Boot Camp\Data & Module Resources\Module 06\DAX Hint 06B.docx.

Module 6C Demo: CALCULATE - US Sales

Previously we used CALCULATE to ignore a filter context, in this example we are going to show how CALCULATE can override a filter context.

Module Requirements

1. Open the pbix file Module 06A found here C:\Dax Boot Camp\Completed Labs\Module 06\Module 6A.pbix

Turn to next page for step-by-step instructions.

Sales Territory Country	Total Sales (US)	Total Sales
Australia	$9,389,789.51	$9,061,000.58
Canada	$9,389,789.51	$1,977,844.86
France	$9,389,789.51	$2,644,017.71
Germany	$9,389,789.51	$2,894,312.34
United Kingdom	$9,389,789.51	$3,391,712.21
United States	$9,389,789.51	$9,389,789.51
Total	**$9,389,789.51**	**$29,358,677.22**

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 06A.pbix,*** this file can be found at the following location: *C:\Dax Boot Camp\Completed Labs\Module 06\Module 06A.pbix* 2. We always want to return sales for the United States, ignoring any filters that may be applied to the country. 3. Create a new calculated measure: **Total Sales (US) =** **CALCULATE(** **[Total Sales],** **'Sales Territory'[Sales Territory Country] = "United States")**	``` Total Sales (US) = CALCULATE([Total Sales], 'Sales Territory'[Sales Territory Country] = "United States") ```

4. Add this new measure to the table, now we can see the Sales of United States repeated for all the rows.

5. The new measure [Total Sales (US)] will respect any other filters that are applied. Feel free to add filters to the report or slicers and watch how the value of Total Sales changes accordingly.

Student Challenge! Attempt to remove the blank value that now appears in the chart!

Sales Territory Country	Total Sales	Total Sales (US)
Australia	$9,061,000.58	$9,389,789.51
Canada	$1,977,844.86	$9,389,789.51
France	$2,644,017.71	$9,389,789.51
Germany	$2,894,312.34	$9,389,789.51
NA		$9,389,789.51
United Kingdom	$3,391,712.21	$9,389,789.51
United States	$9,389,789.51	$9,389,789.51
Total	**$29,358,677.22**	**$9,389,789.51**

6. Next, create a new measure called Total Sales (US & Canada). There are multiple ways to create this measure. Two common methods may be to be use the functions OR IN.

7. Write the following DAX expression:

```
Total Sales (US) =
CALCULATE(
    [Total Sales],
    OR(
        'Sales Territory'[Sales Territory Country] = "United States",
        'Sales Territory'[Sales Territory Country] = "Canada"))
```

8. Optionally, the expression can also be written this way:

```
Total Sales (US) =
CALCULATE(
    [Total Sales],
    'Sales Territory'[Sales Territory Country]
              IN{"United States", "Canada"})
```

Country	Total Sales	Total Sales (US & Canada)
Australia	$9,061,000.58	$11,367,634.37
Canada	$1,977,844.86	$11,367,634.37
France	$2,644,017.71	$11,367,634.37
Germany	$2,894,312.34	$11,367,634.37
United Kingdom	$3,391,712.21	$11,367,634.37
United States	$9,389,789.51	$11,367,634.37
Total	**$29,358,677.22**	**$11,367,634.37**

9. Next, let's create a new measure that returns United State sales for the year 2008.

10. The DAX expression in the screenshot to the right will return sales for the United States for the year 2008.

In this example, multiple filters have been added to Calculate.

11. Save your file as **Module 06C.pbix**. You have finished this lab exercise.

```
1 Total Sales (US & 2008) =
2 CALCULATE(
3     [Total Sales],
4     'Sales Territory'[Sales Territory Country] = "United States",
5     'Date'[Calendar Year] = 2008)
```

Notes	

Module 6D Lab: CALCULATE – US Sales

This lab will not have step-by-step instructions

Your manager has requested that you fix the measure [Total Sales (US)]. Currently the measure changes anytime an end user changes the year, your manager has asked that the measure ignore any filters applied to the date table.

Module Requirements

1. Open the pbix file Module 06D found here C:\Dax Boot Camp\Class Labs\Module 06\Module 06D.pbix
2. Modify the existing calculated measure **Total Sales(US)**.
3. In addition to the existing filters also ignore any filters applied to the date table.

Country	Total Sales	Total Sales (US)
United States	$3,324,031	$9,389,790
Australia	$2,563,884	$9,389,790
United Kingdom	$1,210,286	$9,389,790
Germany	$1,076,891	$9,389,790
France	$922,179	$9,389,790
Canada	$673,628	$9,389,790
Total	**$9,770,900**	**$9,389,790**

Hints

1. Remember that the CALCULATE function is not limited to one filter.
2. Remember the calculation used in Module 06B.
3. With the ALL function you can apply a filter to a specific column, or an entire table.
4. The completed lab can be found in the completed labs folder: C:\Dax Boot Camp\Completed Labs\Module 06\Module 06D.pbix
5. The completed DAX code can be found in the completed labs folder: C:\ Dax Boot Camp\Data & Module Resources\Module 06\DAX Hint 06D.docx.

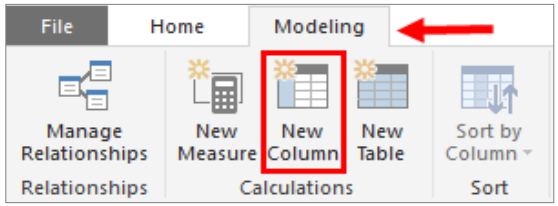
Module 7:

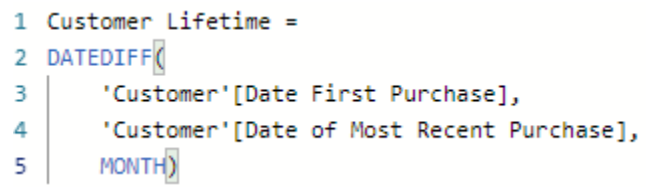
Module 7A Demo: Using Conditional Logic

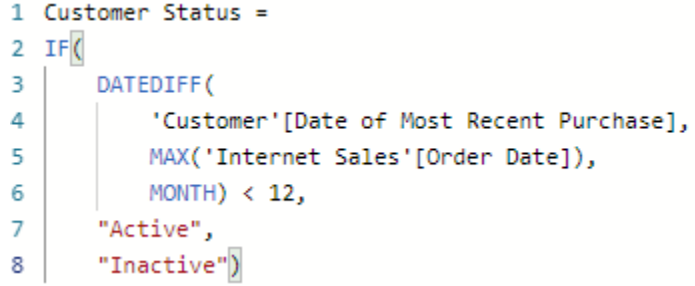
Module Requirements

1. Create measures that determine how long any person has been a customer and if that customer has made a purchase in the last year
2. Create a measure that assigns each customer a loyalty value using SWITCH(TRUE()) and the && operator
3. Add these measures to a table visual with customer names and verify they are correct

If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open **Module 7A**. The file can be found in the folder **C:\DAX Boot Camp\Class Labs\Module 07A.pbix**. 2. Let's start by creating a new column on the Customer table using the **New Column** button in the **Modeling** tab	

3. This column should determine how long someone has been our customer.
 	
`Customer Lifetime =`
`DATEDIFF(`
`'Customer'[Date First Purchase],`
`'Customer'[Date of Most Recent Purchase],`
`MONTH)`
	

4. Next let's build another column that will determine if the customer is active and has made a purchase in the last year:
 	
`Customer Status =`
`IF(`
`DATEDIFF(`
`'Customer'[Date of Most Recent Purchase],`
`MAX('Internet Sales'[Order Date]),`
`MONTH) < 12,`
`"Active",`
`"Inactive")`
	

Note: Instead of calculating the max of customer's order date, in the real world, we would use the TODAY() function.

5. Next, check the columns in the Data View to make sure they're working as intended. You can also add these fields to a table visual in the Report View to verify they're working as intended.

Full Name	Customer Lifetime	Customer Status
Jon Yang	2	Inactive
Eugene Huang	3	Inactive
Ruben Torres	2	Inactive
Christy Zhu	2	Inactive
Elizabeth Johnson	2	Inactive
Julio Ruiz	2	Inactive
Janet Alvarez	2	Inactive
Marco Mehta	2	Inactive

6. Now we can begin creating our final column to determine customer loyalty using SWITCH(TRUE()):

Customer Loyalty =
SWITCH(TRUE(),
'Customer'[Customer Lifetime] >=3 &&
'Customer'[Customer Status] = "Active",
"Longtime Customer")

```
1 Customer Loyalty =
2 SWITCH(TRUE(),
3     'Customer'[Customer Lifetime] >=3 &&
4     'Customer'[Customer Status] = "Active",
5     "Longtime Customer")
```

7. Finish creating the measure with the remainder of the loyalty classifications:

```
Customer Loyalty =

SWITCH(TRUE(),
    'Customer'[Customer Lifetime] >= 36 && 'Customer'[Customer Status] = "Active",    "Longtime Customer",
    'Customer'[Customer Lifetime] >= 36 && 'Customer'[Customer Status] = "Inactive",  "Lost Longtime Customer",
    'Customer'[Customer Lifetime] >= 12 && 'Customer'[Customer Status] = "Active",    "Customer",
    'Customer'[Customer Lifetime] >= 12 && 'Customer'[Customer Status] = "Inactive",  "Lost Customer",
    'Customer'[Customer Lifetime] <  12 && 'Customer'[Customer Status] = "Active",    "Brief Customer",
    'Customer'[Customer Lifetime] <  12 && 'Customer'[Customer Status] = "Inactive",  "Lost Brief Customer")
```

8. Now, check this column in the data view or add this column to the table to verify it is working.

Customer Lifetime	Customer Status	Customer Loyalty
0	Active	Brief Customer
0	Inactive	Lost Brief Customer
1	Active	Brief Customer
1	Inactive	Lost Brief Customer
2	Active	Brief Customer
2	Inactive	Lost Brief Customer

9. You've just created some helpful conditional logical functions using time intelligence and operators!

10. Save your file as **Module 07A.pbix**. You have finished this lab exercise.

Notes

Module 7B Demo: Creating Calculated Measures – Time Intelligence

The next step is to create several Calculated Measures to provide additional metrics that can be used when building reports. In this module you will add several metrics to the model you have designed so far.

Module Requirements

1. Create Calculated Measures that do the following:
    a. Prior Year Profit
    b. YTD Profit
    c. Last Year YTD Profit

Rolling 12 Months Profit**If you want step-by-step instructions, turn to the next page.**

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the **Module 7A** file from your last module. The completed **Module 7A** file can be found in the folder **C:\Dax Boot Camp\Completed Labs\Module 07\Module 07A.pbix**. 2. For this example, you will need to go to the **Data** view and then select the **Internet Sales** table.	
3. Next, you will need to build several time intelligence formulas. Go to the **Modeling** Ribbon and select **New Measures**. 4. In the formula bar write the following formula: **Prior Year Profit = CALCULATE('Internet Sales'[Profit],SAMEPERIODLASTYEAR('Date'[FullDateAlternateKey]))** 5. You should also apply formatting to this measure by selecting the Format value of **$ English (United States)** in the **Modeling** Ribbon.	`Prior Year Profit = CALCULATE('Internet Sales'[Profit], SAMEPERIODLASTYEAR('Date'[FullDateAlternateKey]))` 
6. Go to the **Modeling** Ribbon and select **New Measures**. 7. In the formula bar, type the following formula: **YTD Profit = TOTALYTD([Profit], 'Date'[FullDateAlternateKey])**	`YTD Profit = TOTALYTD([Profit], 'Date'[FullDateAlternateKey])` 

8. You should also apply formatting to this measure by selecting the Format value of **$ English (United States)** in the **Modeling** Ribbon.

9. Go to the **Modeling** Ribbon and select **New Measures**.

10. In the formula bar write the following formula:

    Last Year YTD Profit = TOTALYTD([Profit], DATEADD('Date'[FullDateAlternateKey], -12, MONTH))

```
1 Last Year YTD Profit =
2 CALCULATE(
3     [YTD Profit],
4     SAMEPERIODLASTYEAR('Date'[FullDateAlternateKey]))
```

11. You should also apply formatting to this measure by selecting the Format value of **$ English (United States)** in the **Modeling** Ribbon.

12. Next, you will need to build several time intelligence formulas. Go to the **Modeling** Ribbon and select **New Measure**.

13. In the formula bar, type the following formula:

```
Rolling 12 Months Profit =
CALCULATE(
    [Profit],
    DATESBETWEEN(
        'Date'[FullDateAlternateKey],
NEXTDAY(SAMEPERIODLASTYEAR(LASTDATE('Date'[FullDateAlternateKey]))),
LASTDATE('Date'[FullDateAlternateKey])))
```

```
1 Rolling 12 Months Profit =
2 CALCULATE(
3     [Profit],
4     DATESBETWEEN(
5         'Date'[FullDateAlternateKey],
6         NEXTDAY(SAMEPERIODLASTYEAR(LASTDATE('Date'[FullDateAlternateKey]))),
7         LASTDATE('Date'[FullDateAlternateKey])))
```

14. You should also apply formatting to this measure by selecting the Format value of **$ English (United States)** in the **Measure tools** Ribbon.

15. Next, let's create a rolling 3 month average. In this demo, we will look at a couple of ways to accomplish this task, this will give you more demos to play with and more examples.

16. In our first attempt we may try something like the following, however, you will notice after validation this is more of a 90 day average not a 3 month average.

```
Avg Sales, 3 Month Average (First Attempt) =
VAR EndDate    = LASTDATE('Date'[FullDateAlternateKey])

RETURN
CALCULATE(
    AVERAGE('Internet Sales'[Sales Amount]),
       DATESINPERIOD(
            'Date'[FullDateAlternateKey],
            EndDate,
            -3,
            MONTH))
```

Calendar Year	English Month Name	Total Sales	Avg Sales, 3 Month Average (First Attempt)
2005	July	$473,388.16	3,242.38
2005	August	$506,191.69	3,243.64
2005	September	$473,943.03	3,244.47
2005	October	$513,329.47	3,225.62
2005	November	$543,993.41	3,216.95

17. Now, try it again but using the Values function to return the months.

```
Avg Sales, 3 Month Average (Values) =
VAR EndDate    = LASTDATE('Date'[FullDateAlternateKey])

RETURN
CALCULATE(
    AVERAGEX(
       VALUES('Date'[Month Number Of Year]),
       [Total Sales]),
       DATESINPERIOD(
            'Date'[FullDateAlternateKey],
            EndDate,  -3, MONTH))
```

```
1  Avg Sales, 3 Month Average (Values) =
2  VAR EndDate    = LASTDATE('Date'[FullDateAlternateKey])
3
4  RETURN
5  CALCULATE(
6      AVERAGEX(
7          VALUES('Date'[Month Number Of Year]),
8          [Total Sales]),
9          DATESINPERIOD(
10             'Date'[FullDateAlternateKey],
11             EndDate,
12             -3,
13             MONTH))
```

18. Another way to write the same expression would be to use summarize. Summarize is quite popular and is relatable because it's essentially a group by type operation.

```
Avg Sales, 3 Month Average (Summarize) =
VAR EndDate   = LASTDATE('Date'[FullDateAlternateKey])

RETURN
CALCULATE(
    AVERAGEX(
        SUMMARIZE(
            'Date',
            'Date'[Calendar Year],
            'Date'[English Month Name]),
        [Total Sales]),
        DATESINPERIOD(
            'Date'[FullDateAlternateKey],
            EndDate,
            -3,
            MONTH))
```

Tip: *DATESBETWEEN could easily replace DATESINPERIOD here, but DATESINPERIOD is slightly less complex and it is easier not to make a mistake with DATESINPERIOD.*

```
1  Avg Sales, 3 Month Average (Summarize) =
2  VAR EndDate   = LASTDATE('Date'[FullDateAlternateKey])
3
4  RETURN
5  CALCULATE(
6      AVERAGEX(
7          SUMMARIZE(
8              'Date',
9              'Date'[Calendar Year],
10             'Date'[English Month Name]),
11         [Total Sales]),
12         DATESINPERIOD(
13             'Date'[FullDateAlternateKey],
14             EndDate,
15             -3,
16             MONTH))
```

19. Next, create a 7 day moving average.

```
Sales 7-Day Moving Average =
AVERAGEX(
   DATESBETWEEN(
      'Date'[FullDateAlternateKey],
      DATEADD(LASTDATE('Date'[FullDateAlternateKey]), -6, DAY),
        LASTDATE('Date'[FullDateAlternateKey])),
   [Total Sales])
```

```
1  Sales 7-Day Moving Average =
2  AVERAGEX(
3      DATESBETWEEN(
4          'Date'[FullDateAlternateKey],
5          DATEADD(LASTDATE('Date'[FullDateAlternateKey]), -6, DAY),
6          LASTDATE('Date'[FullDateAlternateKey])),
7      [Total Sales])
```

Notes	

Module 5C Lab: Creating Calculated Measures

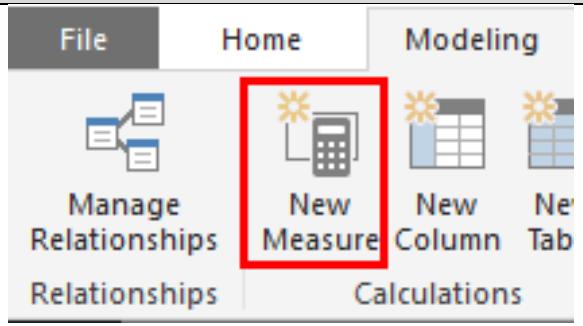
All modules that are labeled "Lab" are labs that will not be done with the guidance of an instructor.

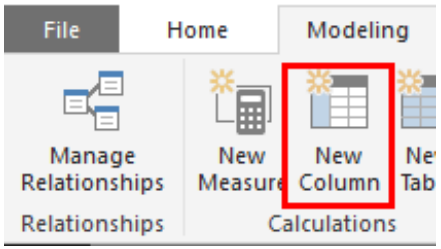
Now that you have most of the organizational pieces out of the way, you want to create some measures for your model to show the number of flights that have been flown, and how long each flight takes on average.

Module Requirements

1. Create a measure in your model to count the number of Airlines currently serving each airport.
2. Create measures in your model to calculate the average flight distance and average flight time.
3. Create 2 measures to return the number of flights and the number of flights for the same period last year.

If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the **Module 4B** file from your last module. The completed **Module 4B** file can be found in the folder **C:\Dax Boot Camp\Completed Labs\Module 04\Module 04B.pbix.** 2. Navigate to the **Data** view and find the **On Time Performance** table. Click on the option to create a **New Measure** from the **Modeling** ribbon. 3. Using the following formula to return back the average flight time: **Average Flight Time = AVERAGE([Actual Elapsed Time])**	 `Average Flight Time = AVERAGE([Actual Elapsed Time])`
4. Create another **Measure** to calculate the average flight distance in miles: **Average Flight Distance in Miles = AVERAGE([Distance In Miles])**	`Average Flight Distance In Miles = AVERAGE([Distance In Miles])`

5. Navigate to the **Airport** table and create a calculated **Column** to identify the number of airlines serving each airport. You learned how to do this in the previous module. Use the following formula:
 	
Number of Airlines = CALCULATE(DISTINCTCOUNT('On Time Performance'[DimAirlineKey]))
	
	`Number of Airlines =`
	`CALCULATE(DISTINCTCOUNT('On Time Performance'[DimAirlineKey]))`

6. In the **Airport** table, we now want to add a calculated **Measure**. If we summed this, the data model will incorrectly sum each level up in the geography hierarchy.

   We want to return the max number of ailrlines for each level in the table. So, start by hiding the **Number of Airlines** column you just created.

   Right-click on the column and select **Hide in Report View**.

7. Now create a **Measure** that returns the maximum number of airlines for each level:

   Number Airlines = MAX([Number of Airlines])

```
Number Airlines = MAX([Number of Airlines])
```

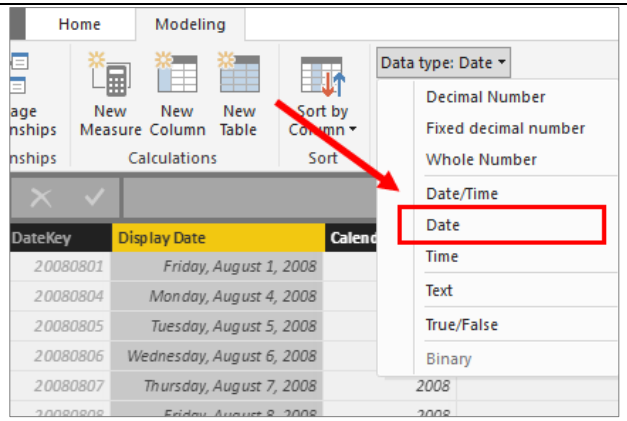
8. Navigate back to the **On Time Performance** table, and do a simple Count measure to count the number of flights:

   Number of Flights = COUNTA([DimAirlineKey])

9. Working with time intelligence functions is a very powerful part of DAX. This allows us to return things like year to date, year over year total, etc… based on our metrics.

   Before we can calculate the previous year, we need to tell the data model where the source of our dates is. Navigate to the **Date** table and select the **Display Date** column. Change the data type from **Text** to **Date**, if it is not already.

   Navigate to the **On Time Performance** table and make the same data type change on the **Flight Date** column.



10. Navigate to the **On Time Performance** table, and create a **New Measure** called Last Years Flights. The formula after will be:

    Last Years Flights = CALCULATE([Number of Flights], SAMEPERIODLASTYEAR('Date'[Display Date]))

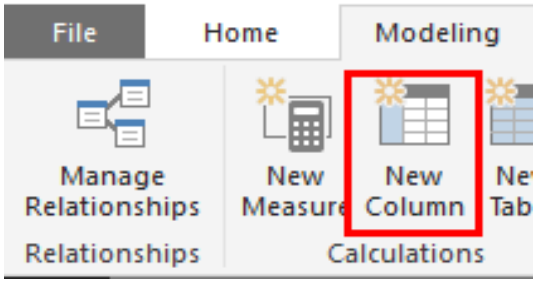
```
Last Years Flights =
CALCULATE(
        [Number of Flights],
        SAMEPERIODLASTYEAR('Date'[Display Date])
        )
```

11. Next, we want to create a **Measure** that will calculate the total transactions.

    While still on the **On Time Performance** table, select **New Measure** from the **Modeling** tab.

    Enter the following formula into the formula expression bar:

    Total Transactions = COUNTROWS('On Time Performance')



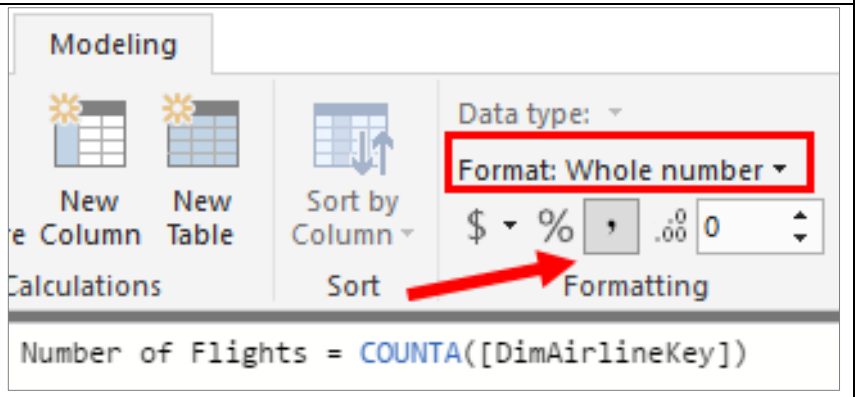

```
Total Transactions =
COUNTROWS('On Time Performance')
```

12. For all of the calculations we just created we need to go back and apply proper formatting.

    Select the calculation from the **Fields** list and from the **Modeling** ribbon you can adjust the format and add comma separators, decimals places, etc.

    Some of these measures will already have proper formatting applied.

13. Save your file as **Module 05C.pbix**. You have finished this lab exercise.



Module 8: Table Functions- Using FILTER

Module 8A Demo: FILTER – Creating a Filter table

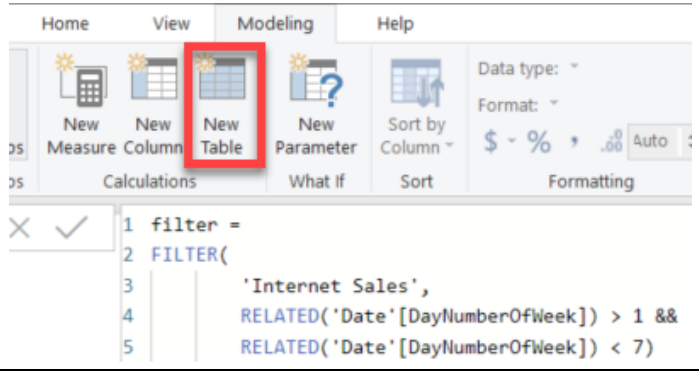
In this example we want to determine exactly how many orders were made in a given time period this is strictly on weekdays .This will help us identify if there is a correlation between our total number of orders compared to the day of week it is.. We are going to use the FILTER function to help accomplish this goal.

Module Requirements

1. Open the pbix file Module 08A found here C:\Dax Boot Camp\Class Labs\ Module 08.pbix

Turn to next page for step-by-step instructions.

Year	Total Quantity	Total Quantity (Weekday)
2005	1,013	737
2006	2,677	1,904
2007	24,443	17,537
2008	32,265	23,166
Total	**60,398**	**43,344**

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 08A.pbix,*** this file can be found at the following location: *C:\Dax Boot Camp\Class Labs\ Module 08.pbix* 2. Add the Calendar Year column and the Total Quantity measure to a table 3. You will see that the MIN function is returning the first date of the current filter context.	 Year | Total Quantity 2005 | 1,013 2006 | 2,677 2007 | 24,443 2008 | 32,265 **Total** | **60,398**
4. Let's look at the FILTER function now. 5. Create a new CALCULATED TABLE. 6. Use the following DAX formula in the new table: **FILTER =** **FILTER(** **'Internet Sales',** **RELATED('Date'[Day Number Of Week]) > 1 &&** **RELATED('Date'[Day Number Of Week]) < 7)**	 Home View Modeling Help New Measure, New Column, New Table, New Parameter, Sort by Column, Data type:, Format: Calculations What If Sort Formatting 1 filter = 2 FILTER(3 'Internet Sales', 4 RELATED('Date'[DayNumberOfWeek]) > 1 && 5 RELATED('Date'[DayNumberOfWeek]) < 7)

7. FILTER returns a table expression, usually the FILTER function is used inside of a function that returns a scalar expression that is our larger goal here.

8. For now, however, we are showing the results of the FILTER function by creating a calculated table and then using that table for validation purposes.

9. Navigate to the data view and then to the new table “Filter”. We can see here that the table has reduced to only records where the day of the week is always a weekday.

OrderDate
Friday, July 1, 2005
Friday, July 1, 2005
Friday, July 1, 2005
Friday, July 1, 2005
Friday, July 1, 2005
Monday, July 4, 2005
Monday, July 4, 2005
Tuesday, July 5, 2005
Tuesday, July 5, 2005
Tuesday, July 5, 2005

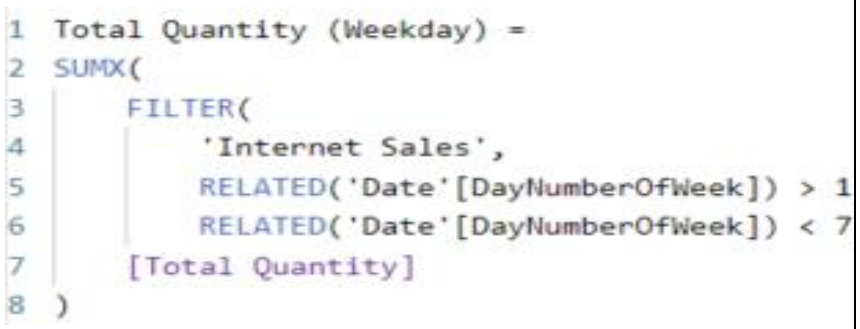
10. To determine total quantity of items sold during the week for our example, what we want to do is filter the existing sales table down to meet two criteria.

    - Order Date is not on Saturday or Sunday of any given week.

11. We can then do a SUMX operation on the filtered table to return the Total Homes on Market.

12. Create a calculated measure to use the following code:

```
Total Quantity (Weekday) =
SUMX(
    FILTER(
        'Internet Sales',
        RELATED('Date'[Day Number Of Week]) > 1 &&
        RELATED('Date'[Day Number Of Week]) < 7),
    [Total Quantity])
```



```
1  Total Quantity (Weekday) =
2  SUMX(
3      FILTER(
4          'Internet Sales',
5          RELATED('Date'[DayNumberOfWeek]) > 1
6          RELATED('Date'[DayNumberOfWeek]) < 7
7      [Total Quantity]
8  )
```

13. Here are the results:

14. Save your file as **Module 08A.pbix**. You have finished this lab exercise.

Year	Total Quantity	Total Quantity (Weekday)
2005	1,013	737
2006	2,677	1,904
2007	24,443	17,537
2008	32,265	23,166
Total	**60,398**	**43,344**

Notes	

Module 8B Lab: FILTER - Total Sales on Weekdays

This lab will not have step-by-step instructions.

One of your end users has requested a new measure called Total Sales on Weekdays. They want you to create that measure for them.

Module Requirements

1. Open the pbix file Module 08B found here C:\Dax Boot Camp\Class Labs\Module 08\Module 08B.pbix
2. Make a new calculated measure called **Total Sales on Weekdays**

Year	Total Quantity	Total Quantity (Weekday)	Total Sales	Total Sales (Weekday)
2005	1,013	737	$3,266,373.66	$2,385,567.14
2006	2,677	1,904	$6,530,343.53	$4,669,330.56
2007	24,443	17,537	$9,791,060.30	$7,056,875.82
2008	32,265	23,166	$9,770,899.74	$6,938,891.23
Total	**60,398**	**43,344**	**$29,358,677.22**	**$21,050,664.74**

Hints

1. Same as the previous measure we created but with a different metric.

Module 8C Demo: FILTER – Time Intelligence

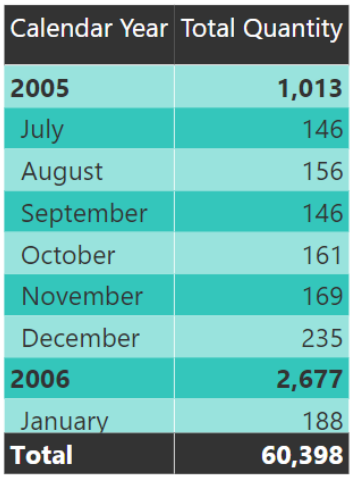
In this module we review different ways to write time intelligence, these patterns help to understand DAX and Filter context better.

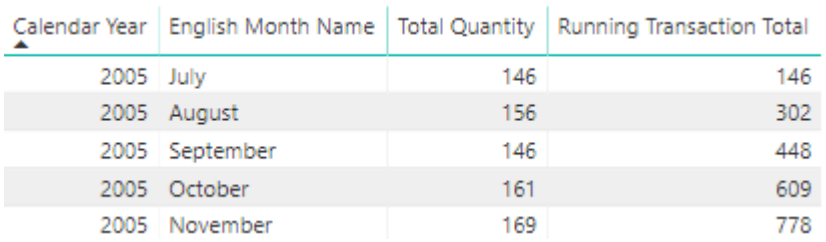
Module Requirements

1. Open the pbix file Module 08C found here C:\Dax Boot Camp\Class Labs\Module 08\Module 08C.pbix
2. Create a new measure that calculates the starting value for the current date period. **[Starting Month Rolling Total (All Time)]**

Calendar Year	Total Quantity	Starting Month Rolling Total (All Time)
2005	**1,013**	
July	146	
August	156	146
September	146	302
October	161	448
November	169	609
December	235	778
2006	**2,677**	**1,013**
Total	**60,398**	

Step-by-Step Instructions

Click Steps	Screen Shots
1. Open ***Module 08C.pbix,*** this file can be found at the following location: *C:\Dax Boot Camp\Completed Labs\Module 08\Module 08B.pbix* 2. We are starting off by looking at our Total Quantity measure in a Matrix. (It is easier to see some things in a hierarchal format when using multiple columns like Year & Month). 3. What we are looking to do is take the aggregated total for all previous dates and have that value is that *starting* value for the current date period.	

4. Create the following measure that utilizes the FILTER function:
 	
`Running Transaction Total =`
`CALCULATE(`
`  [Total Transactions],`
`  FILTER(`
`  ALL('Date'),`
`  'Date'[FullDateAlternateKey] <=`
`  MAX('Internet Sales'[Order Date])))`
	

Calendar Year	Total Quantity
2005	**1,013**
July	146
August	156
September	146
October	161
November	169
December	235
2006	**2,677**
January	188
Total	**60,398**

Calendar Year	English Month Name	Total Quantity	Running Transaction Total
2005	July	146	146
2005	August	156	302
2005	September	146	448
2005	October	161	609
2005	November	169	778

5. Let's work with Filter and now create a YTD Sales calculation:

```
YTD Sales =
CALCULATE(
    [Total Sales],
    FILTER(
        ALL('Date'),
        'Date'[Calendar Year] = MAX('Date'[Calendar Year]) &&
        'Date'[FullDateAlternateKey] <=
            MAX('Date'[FullDateAlternateKey])))
```

Calendar Year	English Month Name	Total Sales	YTD Sales
2005	July	$473,388.16	$473,388.16
2005	August	$506,191.69	$979,579.85
2005	September	$473,943.03	$1,453,522.89
2005	October	$513,329.47	$1,966,852.36
2005	November	$543,993.41	$2,510,845.77

```
YTD Sales =
CALCULATE(
    [Total Sales],
    FILTER(
        ALL('Date'),
        'Date'[Calendar Year] = MAX('Date'[Calendar Year]) &&
        'Date'[Month Number of Year]
            = MAX('Date'[Month Number of Year]) &&
        'Date'[FullDateAlternateKey]
           <= MAX('Date'[FullDateAlternateKey])))
```

Tuesday, July 26, 2005	$17,534.79	378,458.89
Wednesday, July 27, 2005	$28,041.32	406,500.21
Thursday, July 28, 2005	$19,785.36	426,285.58
Friday, July 29, 2005	$17,688.07	443,973.65
Saturday, July 30, 2005	$14,402.34	458,375.98
Sunday, July 31, 2005	$15,012.18	473,388.16
Monday, August 1, 2005	$17,891.35	17,891.35
Tuesday, August 2, 2005	$10,734.81	28,626.16
Wednesday, August 3, 2005	$11,230.63	39,856.79

Module 9: ALLEXCEPT and ALLSELECTED

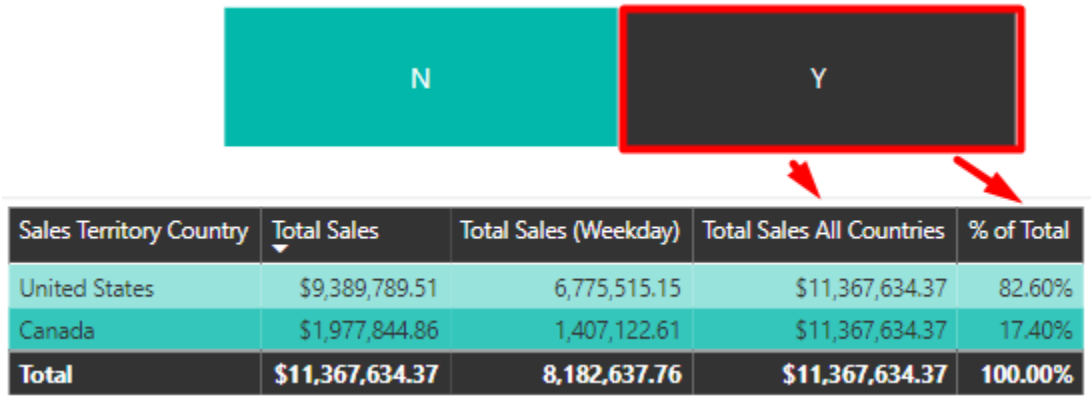
Module 9A Demo: ALLEXCEPT and ALLSELECTED

Modify the measure Total Sales All Countries, this measure is looking at Total Sales for all countries. However, you need to be able to see a breakdown by if the continent is in North America or not. The existing Total Sales All Countries measure does not allow filtering on this column.

Module Requirements

1. Open the pbix file Module 08C found here C:\Dax Boot Camp\Completed Labs\Module 08\Module 08C.pbix

Turn to next page for step-by-step instructions.



Step-by-Step Instructions	
Click Steps	**Screen Shots**

1. Open up ***Module 09A.pbix,*** this file can be found at the following location:
   C:\Dax Boot Camp\Class Labs\Module 09A.pbix

2. Create a new column in the Sales Territory table with the following code:

```
North America? (Y/N) =
IF('Sales Territory'[Sales Territory Country] = "United States",    "Y",
IF('Sales Territory'[Sales Territory Country] = "Canada",           "Y",
If('Sales Territory'[Sales Territory Country] = "Mexico",           "Y",
"N")
```

3. The existing measure currently removes ALL filters applied to the Sales Territory table. Unfortunately, this prevents us from being able to filter the measure by the column **North America? (Y/N)**.

4. We can rewrite the existing measure to only ignore filters on specific columns by adding countless filters, but this would be a lot of code and highly ineffective in most instances.

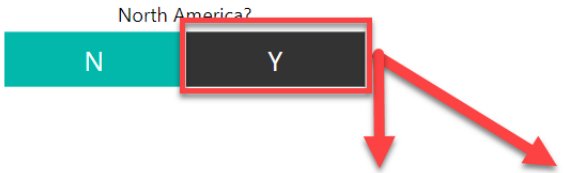
5. A more efficient way of rewriting this would be to use the function **ALLEXCEPT**.

6. Rewrite the existing calculation [Total Sales All Countries] :

```
Total Sales all Countries =
IF(
    [Total Sales] = BLANK(),
    BLANK(),
    CALCULATE(
        [Total Sales],
        ALLEXCEPT(
            'Sales Territory',
            'Sales Territory'[North America? (Y/N)])))
```

7. ALLEXCEPT ignores all the filters on the table except for those explicitly identified. In this example, all filters will be ignored except for any filters applied on the column **North America? (Y/N).**

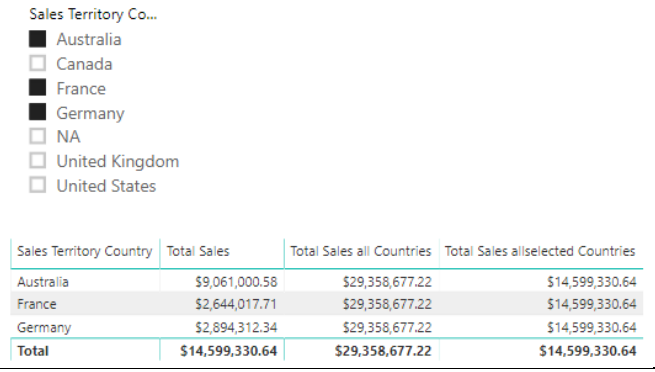
8. Interact with the report and validate that the calculation is working correctly.



Country	Total Sales	Total Sales (Weekday)	Total Sales All Countries	% of Total
Canada	$1,977,844.86	$1,407,122.60	$11,367,634.37	17.40%
United States	$9,389,789.51	$6,775,515.15	$11,367,634.37	82.60%
Total	**$11,367,634.37**	**$8,182,637.76**	**$11,367,634.37**	**100.00%**

9. ALLSELECTED is a another really cool function that is rarely used but very powerful when you need it.

10. Let's create a measure that respects slicers but ignores filters coming from the row of the table!



Sales Territory Country	Total Sales	Total Sales all Countries	Total Sales allselected Countries
Australia	$9,061,000.58	$29,358,677.22	$14,599,330.64
France	$2,644,017.71	$29,358,677.22	$14,599,330.64
Germany	$2,894,312.34	$29,358,677.22	$14,599,330.64
Total	**$14,599,330.64**	**$29,358,677.22**	**$14,599,330.64**

```
Total Sales allselected Countries =
IF(
    [Total Sales] = BLANK(), BLANK(),
CALCULATE(
    [Total Sales],
    ALLSELECTED('Sales Territory'[Sales Territory Country])))
```

Notes	

Module 9B Lab: ALLEXCEPT – Number of Days in Year

This lab will not have step-by-step instructions.

In later labs we will be using the function ALLEXCEPT in a larger capacity. Use what you learned in the last demo to accurately count the number of days in the year.

Module Requirements

1. Open the pbix file Module 09B found here C:\Dax Boot Camp\Class Labs\Module 09\Module 09B.pbix
2. Create a new calculated measure called [Days in Year], this measure should accurately count the number of days in the year regardless of any other filters.

Calendar Year	English Month Name	Total Quantity	Days in Year
2005	January		365
2005	February		365
2005	March		365
2005	April		365
2005	May		365
2005	June		365
2005	July	48	365
2005	August	52	365
Total		**28,964**	**2191**

Hints

1. Use the function COUNTROWS to determine the number of days in the current filter context.
2. Use ALLEXCEPT to get the correct count of days here.
3. The completed lab can be found in the completed labs folder: C:\Dax Boot Camp\Completed Labs\Module 09\Module 09B.pbix
4. The completed DAX code can be found in the completed labs folder: C:\ Dax Boot Camp\Data & Module Resources\Module 09\DAX Hint 09B.docx.

Module 10: Working with Totals

Module 10A Demo: Working with Variables

Variables make it easier to write complicated calculations and have some positive performance implications. We are going to be optimizing our code with variables.

Module Requirements

1. Open the pbix file Module 10A found here C:\Dax Boot Camp\Class Labs\Module 10\Module 10A.pbix
2. Create a new calculated measure called [Dynamic Measure], this measure will be used to display YTD sales for any completed months and forecast YTD sales for incomplete months.

Turn to next page for step-by-step instructions.

Year	Month	YTD Sales (filtered)	Forecast YTD Sales	Dynamic Measure	Last Date	Last Order Date
2008	January	$1,340,244.95	$877,730.34	$1,340,244.95	1/31/2008	1/31/2008
2008	February	$2,802,724.78	$1,855,911.01	$2,802,724.78	2/29/2008	2/29/2008
2008	March	$4,283,629.96	$2,827,060.60	$4,283,629.96	3/31/2008	3/31/2008
2008	April	$5,892,380.49	$3,839,859.13	$5,892,380.49	4/30/2008	4/30/2008
2008	May	$7,770,698.00	$4,965,404.26	$7,770,698.00	5/31/2008	5/31/2008
2008	June	$9,062,685.56	$6,075,002.72	$6,075,002.72	6/30/2008	6/20/2008
2008	July		$7,848,340.40	$7,848,340.40	7/31/2008	
2008	August		$9,543,167.42	$9,543,167.42	8/31/2008	
Total		**$9,062,685.56**	**$19,582,120.60**	**$19,582,120.60**	**12/31/2...**	**6/20/2008**

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 010A.pbix,*** this file can be found at the following location: *C:\Dax Boot Camp\Class Labs\Module 10A.pbix* 2. This module will be using a filtered down Internet Sales table, as the default one will not work for our given scenario. It is filtered down to end on 06/20/2008 3. We will be using several pre-created measures to help us: **Forecast YTD Sales** (which is prior year YTD sales*2), **Last Date, Last Order Date** 4. Our goal is to display YTD sales for any completed months, and forecast YTD sales for incomplete months.	

Year	Month	YTD Sales	Forecast YTD Sales
2008	January	$1,340,244.95	$877,730.34
2008	February	$2,802,724.78	$1,855,911.01
2008	March	$4,283,629.96	$2,827,060.60
2008	April	$5,892,380.49	$3,839,859.13
2008	May	$7,770,698.00	$4,965,404.26
2008	June	$9,062,685.56	$6,075,002.72
Total		**$9,062,685.56**	**$19,582,120.60**

5. Create a new measure called **Dynamic Measure** with the following code:

```
Dynamic Measure =
VAR LastSaleDate = LASTDATE('Internet Sales Filtered'[OrderDate])
VAR LastDay      = LASTDATE('Date'[Date])

RETURN
IF(
    LastSaleDate <> LastDay,
    [Forecast YTD Sales],
    [YTD Sales])
```

Year	Month	YTD Sales	Forecast YTD Sales	Dynamic Measure
2008	January	$1,340,244.95	$877,730.34	1,340,244.95
2008	February	$2,802,724.78	$1,855,911.01	2,802,724.78
2008	March	$4,283,629.96	$2,827,060.60	4,283,629.96
2008	April	$5,892,380.49	$3,839,859.13	5,892,380.49
2008	May	$7,770,698.00	$4,965,404.26	7,770,698.00
2008	June	$9,062,685.56	$6,075,002.72	6,075,002.72
Total		**$9,062,685.56**	**$19,582,120.60**	**19,582,120.60**

6. Here is another example of how to use and leverage variables:

```
Prior Year Sales =
VAR LastYear     = SAMEPERIODLASTYEAR('Date (Order)'[Date])
VAR UnitedStates =
        FILTER(ALL('Sales Territory'[SalesTerritoryCountry]), 'Sales Territory'[SalesTerritoryCountry] = "United States")
RETURN

CALCULATE(
    [Total Sales],
    LastYear,
    UnitedStates)
```

Notes	

Module 10B Demo: HASONEVALUE – Totals

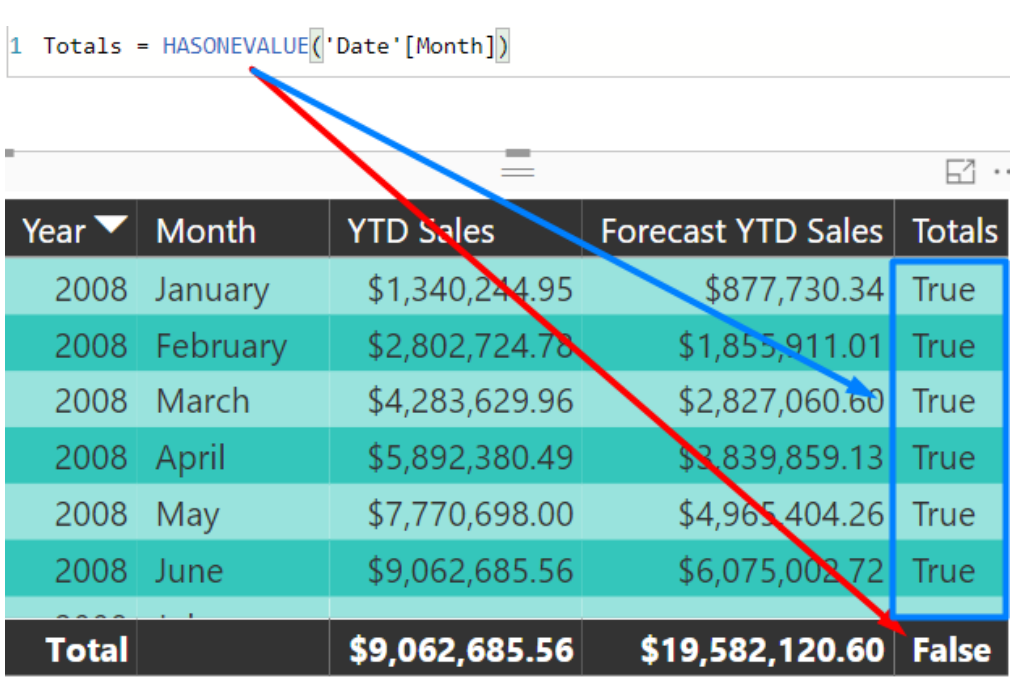
The total row can cause a lot of confusion in Power BI and this has to do with filter context. In this report the total row for any of the YTD sales appear to be very off, when in fact they are technically correct. We will use the functions HASONEVALUE and BLANK to help remove the totals row and eliminate unnecessary confusion.

Module Requirements

1. Open the pbix file Module 10A found here C:\Dax Boot Camp\Completed Labs\Module 10\Module 10A.pbix

Turn to next page for step-by-step instructions.

Year	Month	YTD Sales (filtered)	Forecast YTD Sales ...	Totals
2008	January	$1,340,244.95	$877,730.34	True
2008	February	$2,802,724.78	$1,855,911.01	True
2008	March	$4,283,629.96	$2,827,060.60	True
2008	April	$5,892,380.49	$3,839,859.13	True
2008	May	$7,770,698.00	$4,965,404.26	True
2008	June	$9,720,059.11	$6,075,002.72	True
2008	July	$9,770,899.74	$7,848,340.40	True
2008	August	$9,770,899.74	$9,543,167.42	True
Total				**False**

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 10B.pbix,*** this file can be found at the following location: *C:\Dax Boot Camp*\ Completed Labs\Module 10*Module 10A.pbix* 2. In this report the total row currently shows $9,062,685.56 for our YTD Sales total. Although this is our total YTD so far, this can be quite confusing. 3. The reason that this could be confusing is that when we look at columns, we are used to seeing the column aggregated to show the total amount added up for each column in the current display. That is not what is happening in this case. 4. We will review a more advanced technique later in this course on how to handle these situations, for now we are going to just return blank for the total row. 5. HASONEVALUE can be used to check if the current filter context is limited to only one value. The HASONEVALUE function returns either true or false and is very commonly used as the "logical test" portion of the IF function. 6. Create a new measure called Totals and then write the following expression: **Totals = HASONEVALUE('Date'[Month])**	

7. Now we are going to modify the existing measure [YTD Sales] using the functions HASONEVALUE and BLANK.

8. If the current filter context is filtered down to a single year then we want to perform our original calculation. If the current filter context is filtered down to more than one year then we want to return blank. In other words, if we are at the totals row, return blank.

9. Modify the calculated measure New Homes on Market to use the following formula:

YTD Sales =
IF(HASONEVALUE('Date'[Month]),
TOTALYTD(
[Total Sales],
'Date'[Date]),
BLANK())

```
YTD Sales =
IF(HASONEVALUE('Date'[Month]),
    TOTALYTD(
            [Total Sales],
            'Date'[Date]),
    BLANK())
```

10. Now we can see that the total value for YTD and Forecast YTD Sales are BLANK, this is because our calculation has detected that there is more than one year in the filter context.

Year	Month	YTD Sales (filtered)	Forecast YTD Sales ...	Totals
2008	January	$1,340,244.95	$877,730.34	True
2008	February	$2,802,724.78	$1,855,911.01	True
2008	March	$4,283,629.96	$2,827,060.60	True
2008	April	$5,892,380.49	$3,839,859.13	True
2008	May	$7,770,698.00	$4,965,404.26	True
2008	June	$9,720,059.11	$6,075,002.72	True
2008	July	$9,770,899.74	$7,848,340.40	True
2008	August	$9,770,899.74	$9,543,167.42	True
Total				**False**

11. What if you wanted to return a total, but you wanted a total that was more intuitive or lined up with what the end users expected? We can try that as well. This example returns the most recent month for the total row which makes logical sense when working with running totals.

```
New YTD Sales =
VAR MaxMonth = LASTNONBLANK('Date'[MonthNumberOfYear], [Total Sales])
RETURN

IF(
    HASONEVALUE('Date'[Month]),
    [Dynamic Measure],
    CALCULATE(
        [Dynamic Measure],
        'Date'[MonthNumberOfYear] = MaxMonth))
```

12. SUMX is another function I often use for calculating total values. In this case, we could assume we wanted to perform the dynamic measure for each month-year combination and then sum up the total of the results, see below:

```
Dynamic Measure Fixed =
SUMX(
    VALUES('Date'[Month]),
    [Dynamic Measure])
```

Notes	

Module 11: Evaluation Context (Row/Filter context, and Context Transition)

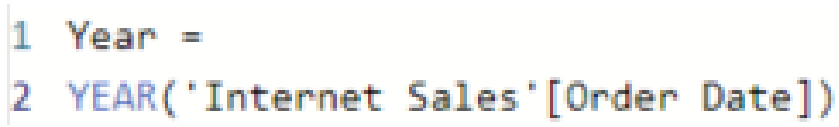
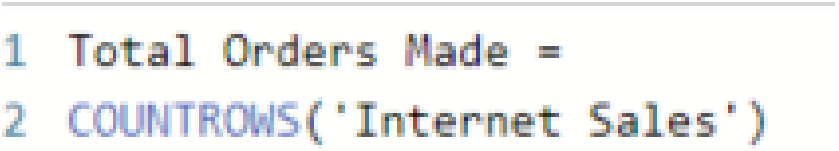
Module 11A Demo: Row Context

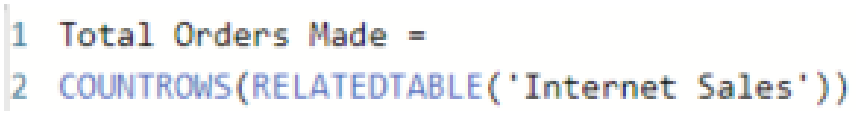
This demo is our introduction into the concept of Row Context. We will create a new calculated column, anytime a calculated column is created a row context is automatically created as well. It's important to know that row contexts do not interact with the relationships in the data model.

Module Requirements

1. Open the pbix file Module 10B found here C:\Dax Boot Camp\Completed Labs\Module 10B.pbix

Turn to next page for step-by-step instructions.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 10B.pbix,*** this file can be found at the following location: *C:\Dax Boot Camp\Completed Labs\Module 10B.pbix* 2. Create a new calculated column on the **Internet Sales** table that returns the **YEAR** from the Order Date. **Year =** **YEAR('Internet Sales'[Order Date])** 3. How does this calculated column know to return the year of the order date for that row? This occurs because a row context has been created and therefore the calculated column correctly returns the year from the Order Date for each transaction in the Internet Sales table.	
4. Navigate to the Date table, now we want to show how row context interacts with the relationships in the data model. 5. Create a new calculated column using the following DAX: **Total Orders Made =** **COUNTROWS('Internet Sales')** 6. The results are somewhat unexpected if you don't understand how row context interacts with relationships. We know that we have a relationship between the date table and the Internet Sales table, we also know that the Date table filters the Internet Sales table, therefore we should expect to see how many orders were made for each date in our date table.	

7. However, a row context does not interact with relationships in the data model and therefore our calculated column returns the total number of records from the MLS table, unfiltered.	
8. One way we can make this calculation work is by using the built-in navigation functions (related and relatedtable). These functions allow the row context to interact with a specified relationship in the data model. 9. For example, the previous calculation could be rewritten like this: **Total Orders Made =** **COUNTROWS(RELATEDTABLE('Internet Sales'))**	

Notes	

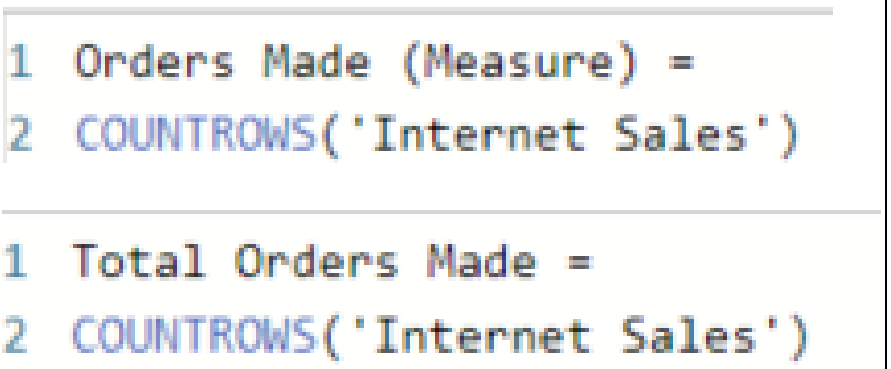
Module 11B Demo: Context Transition

In the previous demo we discussed row context and how row context interacts with the relationships in the data model. We showed you how to use the built-in navigation functions to allow the row context to interact with specified relationships in the model. In this demo we are going to discuss the concept of context transition.

Module Requirements

1. Open the pbix file Module 11A found here C:\Dax Boot Camp\Completed Labs\Module 11A.pbix

Turn to next page for step-by-step instructions.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 11A.pbix,*** this file can be found at the following location: *C:\Dax Boot Camp\Completed Labs\Module 11A.pbix* 2. Create a new calculated **measure** on the DATE table using the following DAX: **Orders Made (measure) =** **COUNTROWS('Internet Sales')** 3. Please note, this is the exact same DAX that was used in the previous module. By itself this code didn't yield the desired results and we further improved the code by using the RELATEDTABLE function. For demo purposes we have removed the RELATEDTABLE function from the calculated column **Total Orders Made**.	
4. Now create a new **calculated column** on the DATE table using the measure that we just created: **Total Orders Made CT** = [Orders Made (Measure)]	

5. Now look at the two calculated columns on the date table. You will see a huge difference in the results. These results can seem very confusing if you don't have a basic understanding of context transition.

6. The DAX code that was used for both calculated columns seems identical: COUNTROWS('Internet Sales')

7. The big difference is that obviously the second calculated column is using calculated measure that has encapsulated the code COUNTROWS('Internet Sales'). But why is this any different?

8. One fact that you are not aware of yet is that calculate measures are implicitly wrapped in a calculate.

9. For example, the calculated measure we created earlier, **Orders Made (measure),** would look like the following if you were to expand its code:

```
CALCULATE(
    COUNTROWS('Internet Sales'))
```

Total Orders Made	Total Orders Made CT
60398	
60398	
60398	
60398	
60398	
60398	
60398	
60398	4
60398	3
60398	4
60398	2
60398	8
60398	3
60398	4
60398	11
60398	9
60398	5

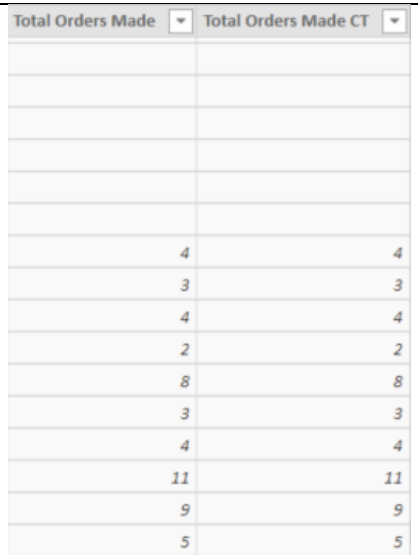
10. The optional [Filter] option has not been used so why does this code produce different results?

11. The CALCULATE function adds the row filter to the filter context. Remember that filter contexts can automatically interact with the relationships in the data model.

12. To prove the behavior of context transition, rewrite the calculated column Total Orders Made:

    Total Orders Made =
    CALCULATE(
    COUNTROWS('Internet Sales'))

13. Context transition is when row contexts are turned into filter context, this behavior can be unexpected and produce incorrect results.



Total Orders Made	Total Orders Made CT
4	4
3	3
4	4
2	2
8	8
3	3
4	4
11	11
9	9
5	5

Notes	

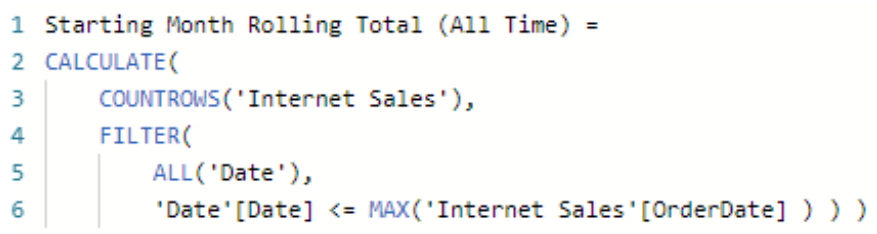
Module 11C Demo: Context Transition (Cont)

Context transition adds a row filter to a filter context. As you saw in the last demo this can help to easily produce the correct results without having to write a lot of code. However, this can also cause some very confusing results. Remember that the FILTER function and X functions (SumX, MinX, etc..) create a row context, context transition here could override the row context and produce incorrect results. Let's look at an example.

Module Requirements

1. Open the pbix file Module 11B found here C:\Dax Boot Camp\Completed Labs\Module 11B.pbix

Turn to next page for step-by-step instructions.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 11B.pbix,*** this file can be found at the following location: *C:\Dax Boot Camp\Completed Labs\Module 11B.pbix* 2. Look at the measure ***Starting Month Rolling Total (All Time)*** 3. This is a calculated measure we built previously in this course (Module 8C). 4. This calculation correctly returns a running total of all transactions. 5. Create a new measure called Total Transactions: Total Transactions = COUNTROWS('Internet Sales')	 1 Starting Month Rolling Total (All Time) = 2 CALCULATE(3 COUNTROWS('Internet Sales'), 4 FILTER(5 ALL('Date'), 6 'Date'[Date] <= MAX('Internet Sales'[OrderDate])))

6. The presence of the Filter function introduces a row context, the Filter function, like X-functions, processes one row in a table at a time.

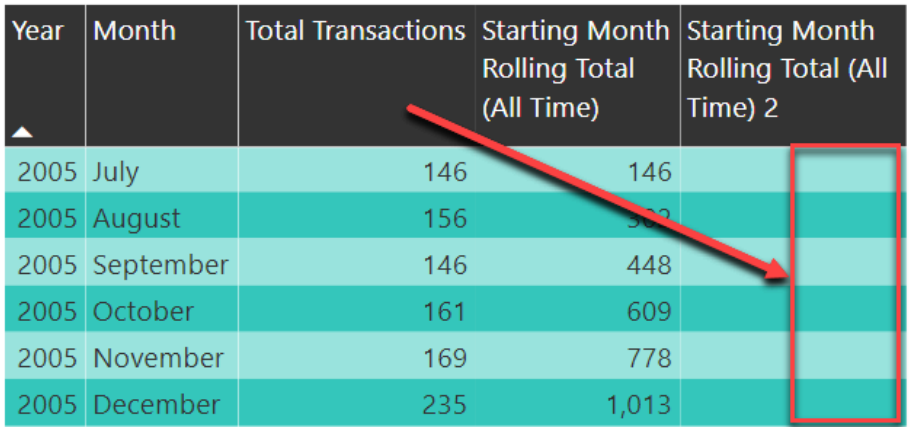
7. The row context here is necessary to return the correct results. Where this gets confusing and where newer DAX authors run into problems is when they try to use calculated measures in place of the DAX code written into the expression. Let's look at an example.

8. Create one new measure:

```
Max Date =
  MAX('Internet Sales'[Order Date] )
```

9. Create a new calculated measure using the following DAX:

```
Starting Month Rolling Total (All Time) 2 =
CALCULATE(
  [Total Transactions],
  FILTER(
    ALL('Date'),
    'Date'[Date] < [Max Date] ) )
```



Year	Month	Total Transactions	Starting Month Rolling Total (All Time)	Starting Month Rolling Total (All Time) 2
2005	July	146	146	
2005	August	156	[illegible]	
2005	September	146	448	
2005	October	161	609	
2005	November	169	778	
2005	December	235	1,013	

10. If you look at these results side by side in a table, you will see drastically different results. See the screenshot to the right.

11. If you change the expression above to be <= to the [Min Date] you get entirely different results yet again. See the second screenshot to the right.

```
Starting Month Rolling Total (All Time) 2 =
CALCULATE(
   [Total Transactions],
   FILTER(
      ALL('Date'),
       'Date'[Date] <= [Max Date] ) )
```

Year	Month	Total Transactions	Starting Month Rolling Total (All Time)	Starting Month Rolling Total (All Time) 2
2005	July	146	146	60,398
2005	August	156	302	60,398
2005	September	146	448	60,398
2005	October	161	609	60,398
2005	November	169	778	60,398
2005	December	235	1,013	60,398

12. What happens if we store the min date in a variable? Does it also produce incorrect results like a measure? Let's take a look.

13. In this example, modify the measure to the following DAX:

```
Starting Month Rolling Total (All Time) 2 =
VAR MaxDate = MIN('Date'[Date])
RETURN

CALCULATE(
   COUNTROWS('Internet Sales'),
   FILTER(
      ALL('Date'),
      'Date'[Date] <= MaxDate ) )
```

14. As you can see in this example, the variable does not produce the same behavior as a calculated measure and is safe to use!

Year	Month	Total Transactions	Starting Month Rolling Total (All Time)	Starting Month Rolling Total (All Time) 2
2008	January	4,585	32,718	32,718
2008	February	4,616	37,334	37,334
2008	March	4,707	42,041	42,041
2008	April	5,088	47,129	47,129
2008	May	5,515	52,644	52,644
2008	June	5,545	58,189	58,189
2008	July	2,209	60,398	60,398

Notes	

Module 11D Demo: Context Transition (Simple)

Context Transition can be a very difficult topic to explain and even harder to grasp. In this example, we are going to take a look at a very simple data model that hopefully helps to clarify these concepts even further!

Open Module 11D from ***C:\Dax Boot Camp\Completed Labs\Module 11D\Module 11D.pbix***

Module 12: Role-Playing Tables

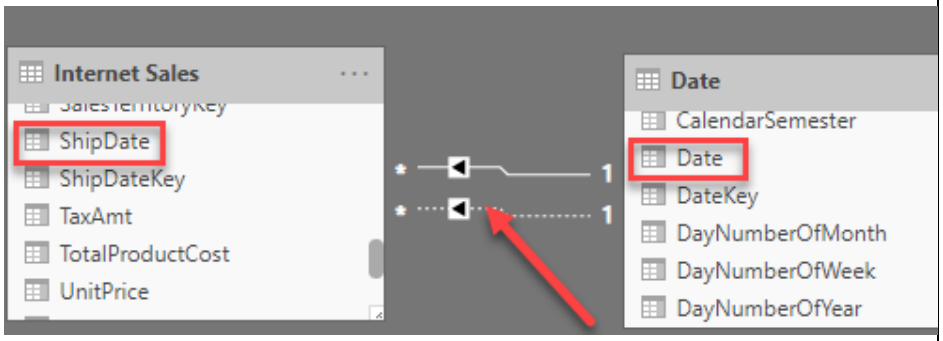
Module 12A Demo: Creating Calculated Measures – Role-Playing Tables

The next step is to create several Calculated Measures to provide additional metrics that can be used when building reports. In this module you will add several metrics to the model you have designed so far.

Module Requirements

1 Create Calculated Measures that do the following:
   a. Profit (Due Date)
   b. Profit (Ship Date)
2 Compare the differences between the above method and duplicating tables.

If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 11C.pbix,*** this file can be found at the following location: ***C:\Dax Boot Camp\Completed Labs\Module 11\Module 11C.pbix*** 2. Next, you would like to work with multiple date relationships in your model. Currently, your model only supports users looking at information based on the **Order Date**. There are multiple ways to fix this issue, but in this module on calculations, you will learn how to create a calculation that navigates through a **Due Date** relationship. Go to the **Relationship** view. 3. Next, draw a relationship between the **Ship Date** and **Date** column from the **Internet Sales** and **Date** tables.	
4. Next, you will need to build several time intelligence formulas. Go to the **Modeling** Ribbon and select **New Measure**. 5. In the formula bar write the following formula: Total Sales (Ship Date) = CALCULATE([Total Sales], USERELATIONSHIP('Date'[Date], 'Internet Sales'[ShipDate]))	1 Total Sales (Ship Date) = 2 CALCULATE(3 [Total Sales], 4 USERELATIONSHIP(5 'Date'[Date], 'Internet Sales'[ShipDate]))

Notes	

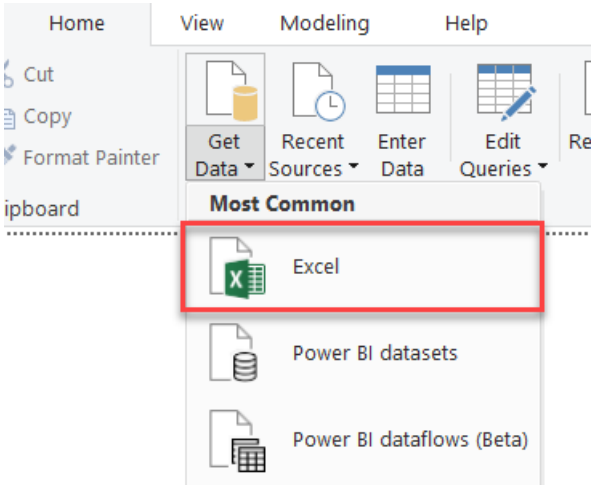
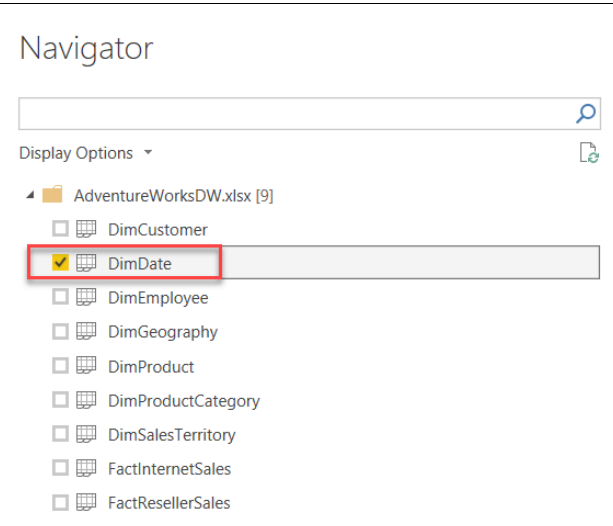
Module 12B Demo: Advanced Data Modeling - Roleplaying Tables (Multiple table imports)

Currently all the sales and measures are based off the active relationships in the data model which is on the order date. Our business requires that we also show all our metrics by the sales date. The BI manager has decided that recreating all these measures would be too much work and require too much on-going maintenance. Instead, import the date table again and create an active relationship to the Internet Sales table

Module Requirements

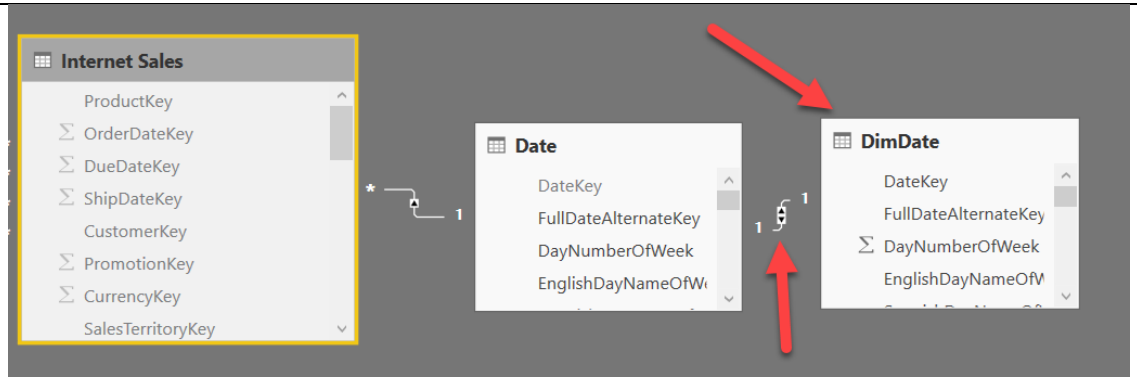
1. Import the new date tables and rename the tables and columns appropriately.
2. Recreate any work done to the original date table.

If you want step-by-step instructions, turn to the next page.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the **Module 12A**.pbix file from your last module. The completed **Module 12A** file can be found in the folder **C:\Dax Boot Camp\Completed Labs\Module 12\Module 12A.pbix.** 2. Next, we need to import the date table into the data model. 3. The data file can be found at the following location: C:\DAX Bootcamp\Data\Databases\AdventureWorksDW.xlsx 4. From the home ribbon, select the **Get Data** dropdown and select Excel.	
5. Navigate to C:\DAX Bootcamp\Data\Databases\AdventureWorksDW.xlsx and open that excel file. 6. The only table we want to import here is the DimDate table. Make sure that **Load** is selected.	

7. Switch to the **Relationship** view.

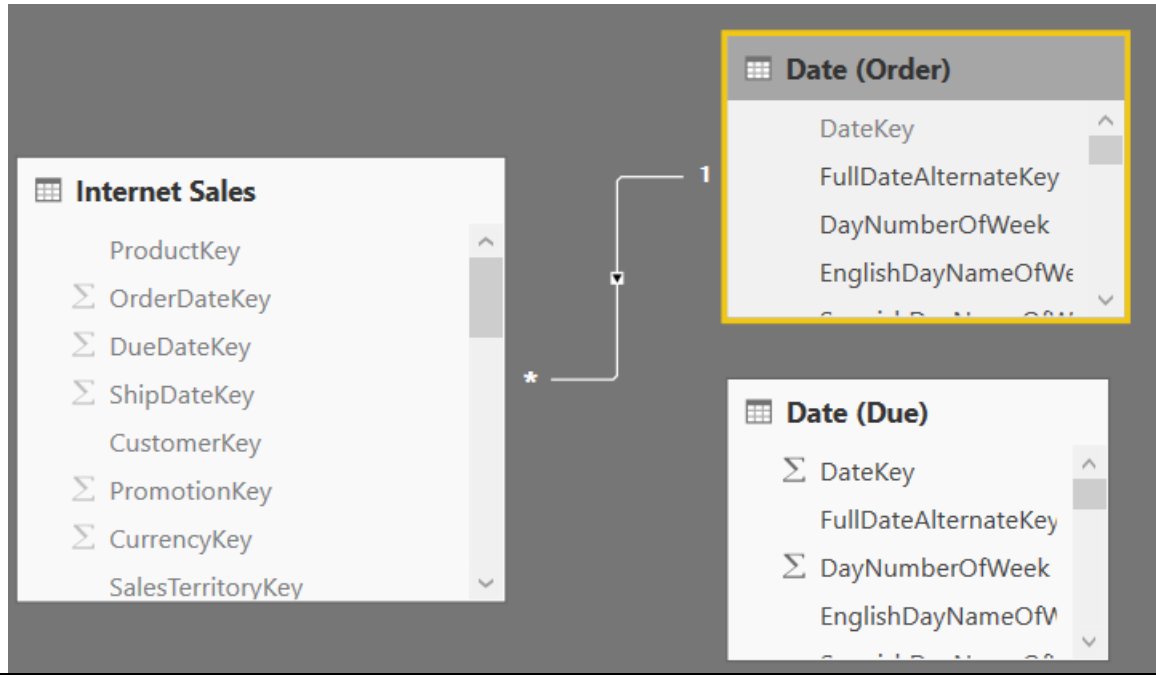
8. Notice that the new table we just loaded has an incorrect relationship. Go ahead and delete the relationship that was created, as we will be making a new one shortly.



9. Now let's change the name of the new date table that was just added from DimDate to something more useable.

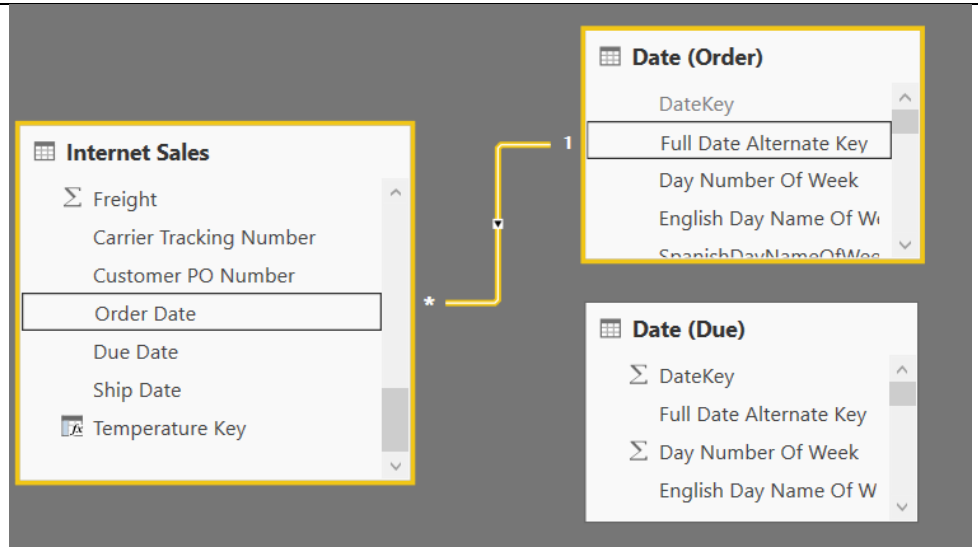
10. Since **Date** already exists, and since we are using a new method for role playing, let's call this table **Date (Ship).**

11. Now let's modify the original **Date** table and call it **Date (Order)** instead.



12. If we analyze the relationship that is currently in place with the **Date (Order)** table and the **Internet Sales** table, we will see that the relationship is currently built from **'Date (Order)'[Full Date Alternate Key]** to **'Internet Sales'[Order Date]**

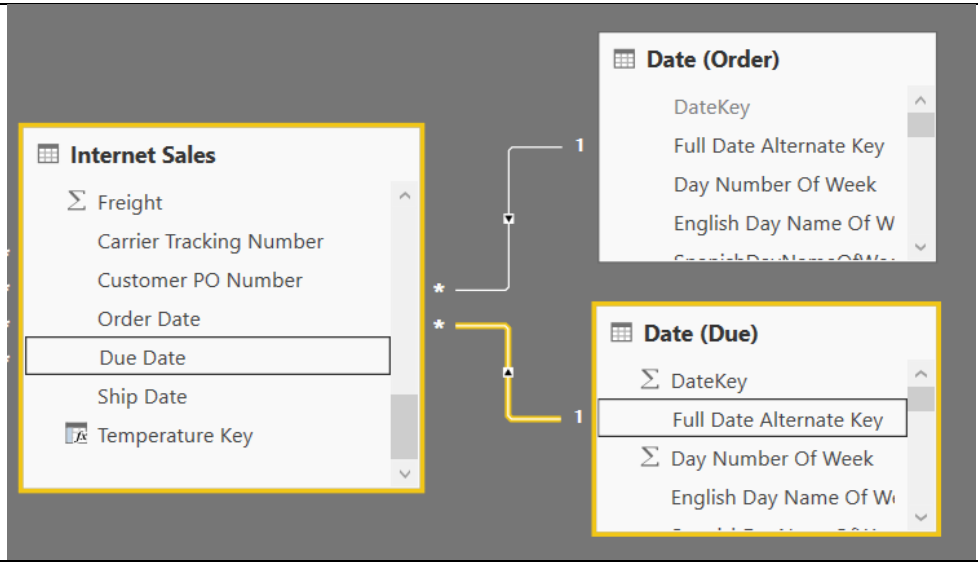
13. This is correct and it is the only relationship we will be keeping between these two tables. Contrary to the previous method, there will only be one relationship between each table.



14. Now we need to make a new relationship between our **Date (Due)** table and our **Internet Sales** table. This will be the same concept as the other Date table, except we will be using **Due Date** rather than **Order Date.**

15. Drag and drop the **Due Date** column from the **Internet Sales** table to the **Full Date Alternate Key** column on the **Date (Due)** table.

16. This creates the intended relationship that we need for our role-playing tables.



Notes	

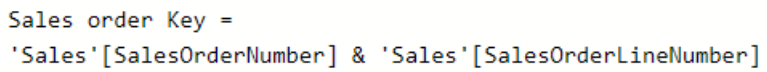
Module 13: Weighted Allocation and Mismatched Granularity

Module 13A Demo: Weighted Allocation

In this demo we want to show how double counting can occur when the data model is not built correctly and how to resolve that by using a bridge table and weighted allocation.

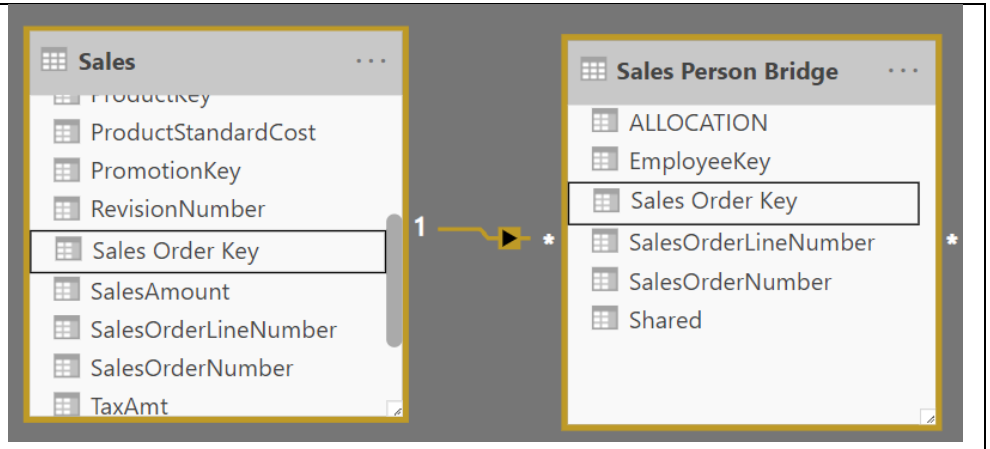
Module Requirements

1. Open Module 13A from the Class Labs folder.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the ***Module 13.pbix***. This file can be found at the following location: ***C:\DAX Boot Camp\Class Labs\Module 13A\Module 13.pbix*** 2. The first thing we want to do is build a relationship **Sales** to the **Sales Person Bridge** table. 3. Right now, this is not possible because the **Sales** table does not have a unique column that we can use to join to the **Sales Person Bridge.** 4. Create a new **Calculated Column** on the **Sales** table: **Sales Order Key =** **'Sales'[SalesOrderNumber] &** **'Sales'[SalesOrderLineNumber]**	
5. We need to repeat this process on our **Sales Person Bridge** table. 6. Create a new **Calculated Column** on the Sales Person Bridge: **Sales Order Key** = **'Sales Person Bridge'[SalesOrderNumber] &** **'Sales Person Bridge'[SalesOrderLineNumber]**	

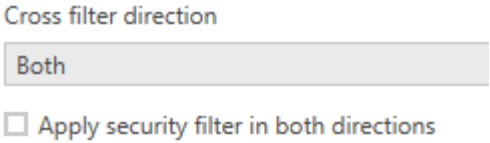
7. Next, we need to build a relationship between the **Sales** table and the **Sales Person Bridge** table. Navigate to the **Relationship View** and build a relationship between **Sales** and **Sales Person Bridge**:

'Sales'[Sales Order Key] → 'Sales Person Bridge'[Sales Order Key]



8. The relationship between these two tables also needs to be filtering in both directions. Open the **Edit Relationship** box by double clicking on the relationship between Sales and Sales Person Bridge.

9. Change the cross-filter direction from single to **both**.

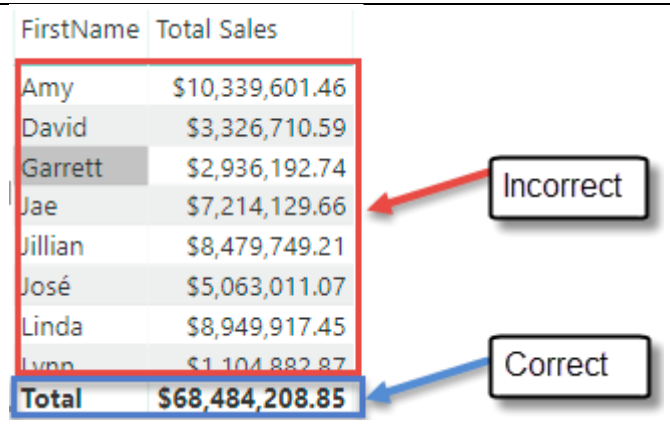


10. Back in the **Report View**, create a new **Calculated Measure** called **Total Sales**:

Total Sales =
SUM('Sales'[Sales Amount])

11. Add the **[Total Sales]** measure to the table on the report and notice that the total of all sales is $68,484,208.85, this number is correct.

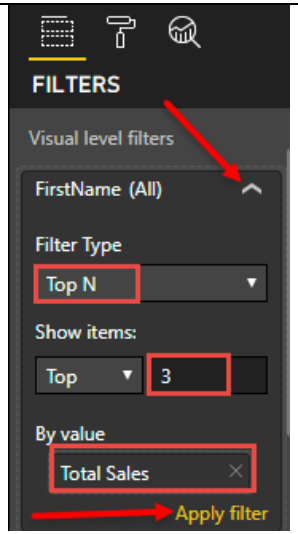
12. Although the total row is correct, the individual rows in the table are not correct. The individual rows are showing inflated numbers because it is showing that



each sales person received 100% of each sale they were a part of.

13. We can validate that the individual rows are incorrect and not adding up to the total by adding a filter to our table that filters the table down to only show the top 3 records.

14. Add a filter on the table visual to only return the top 3 sales people ordered by Total Sales. See screenshot.



15. With this filter applied we can easily determine that the individual rows within the table are not adding up to the total row. The problem here is with our Total Sales calculation.

16. Create a new **Calculated Measure** called **Total Sales w Allocation:**

```
Total Sales w Allocation =
SUMX(
   'Sales Person Bridge',
   SUM('Sales'[SalesAmount]) *
  'Sales Person Bridge'[ALLOCATION])
```

This first attempt creates really odd results…. Why? This has to do with Row Context.

```
1 Total Sales w Allocation =
2 SUMX(
3     'Sales Person Bridge',
4     SUM('Sales'[SalesAmount]) * 'Sales Person Bridge'[ALLOCATION])
```

17. We can quickly fix the previous code to work correctly by simply using the [Total Sales] measure rather than writing SUM('Sales'[SalesAmount]). Rewrite the code using the below expression:

```
Total Sales w Allocation =
SUMX(
   'Sales Person Bridge',
   [Total Sales]) *
  'Sales Person Bridge'[ALLOCATION])
```

FirstName	Total Sales	Total Sales w Allocation
Amy	$10,339,601.46	$7,416,899.29
Jillian	$8,479,749.21	$7,536,602.41
Linda	$8,949,917.45	$7,477,468.72
Total	**$24,736,995.35**	**$22,430,970.41**

18. Another way to write the above expression from step 16 above would have been to force context transition to occur with CALCULATE. For example, rewrite the expression as below:

```
Total Sales w Allocation =
SUMX(
   'Sales Person Bridge',
   CALCULATE(SUM('Sales'[SalesAmount])) *
  'Sales Person Bridge'[ALLOCATION])
```

FirstName	Total Sales	Total Sales w Allocation
Amy	$10,339,601.46	$7,416,899.29
Jillian	$8,479,749.21	$7,536,602.41
Linda	$8,949,917.45	$7,477,468.72
Total	**$24,736,995.35**	**$22,430,970.41**

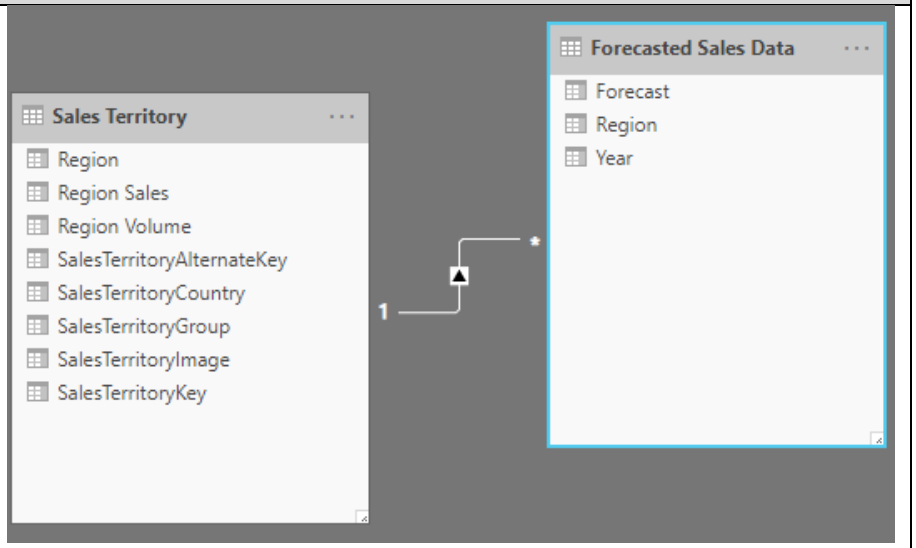
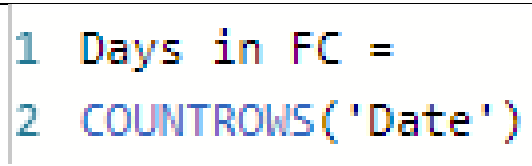
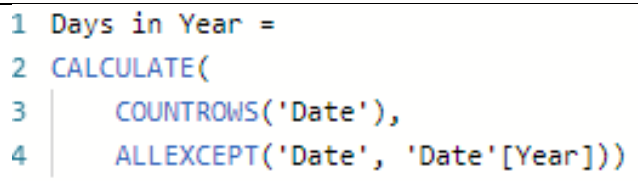
19. We can now observe the difference between the two measures, **[Total Sales w Allocation]** is returning the correct value of sales for each individual sales person!

Module 13B Demo: Mismatched Granularity

Sometimes a data model contains fact tables that store data at different levels of detail (Granularity). Depending on the data model there are a number of different ways to handle this. In this demo we take an example of a forecasted budget which is stored at the month level.

Module Requirements

1. Open Module 13B from the Class Labs folder.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the ***Module 13B.pbix***. This file can be found at the following location: ***C:\DAX Boot Camp\Class Labs\Module 13\Module 13B.pbix*** 2. The first thing we want to do is build a relationship from **Sales Territory** to **Forecasted Sales Data.** 3. Build a relationship from **Region** to **Region**.	
4. Next, create a measure called **[Days in FC]** Days in FC = COUNTROWS('Date')	
5. Create a new measure called **[Days in Year]** Days in Year = CALCULATE(COUNTROWS('Date'), ALLEXCEPT('Date', 'Date'[Year]))	

6. With these two values calculated, we can now distribute the forecasted amounts from the forecast table down to the day and month level!

7. Create a new measure called **[Forecasted Sales]**

```
Forecasted Sales =
  SUMX(
     VALUES('Date'[Year] ),
     DIVIDE(
            [Days in FC],
            [Days in Year] )
     *
     SUM('Forecasted Sales Data'[Forecast] ) )
```

This calculation is close but not quite correct. This calculation does not take into account the actual year. Therefore the Forecasted Sales amount represented is for many years.

Year	Month	Region	Total Sales	Forecasted Sales
2005	July	Northwest	$58,586	$338,027.40
2005	August	Northwest	$43,956	$338,027.40
2005	September	Northwest	$46,657	$327,123.29
2005	October	Northwest	$62,546	$338,027.40
2005	November	Northwest	$80,819	$327,123.29
2005	December	Northwest	$122,639	$338,027.40

8. Rewrite the calculation from step 7 with the following DAX:

```
Forecasted Sales =
SUMX(
   VALUES('Date'[Year]),
   DIVIDE(
      [Days in FC],
      [Days in Year] )
      *
      CALCULATE(
         SUM('Forecasted Sales Data'[Forecast] ) ,
         FILTER(
            'Forecasted Sales Data',
            'Forecasted Sales Data'[Year] = 'Date'[Year] ) ) )
```

Year	Month	Region	Total Sales	Forecasted Sales
2005	July	Northwest	$58,586	$38,643.84
2005	August	Northwest	$43,956	$38,643.84
2005	September	Northwest	$46,657	$37,397.26
2005	October	Northwest	$62,546	$38,643.84
2005	November	Northwest	$80,819	$37,397.26
2005	December	Northwest	$122,639	$38,643.84

9. It's always good to look at many different ways to write a calculation to get a better and deeper understanding of a programming language.

Rewrite the previous expression using variables, this will clean up the code a bit and possibly be easier to read.

```
Forecasted Sales =
VAR CurrentYear = MAX('Date'[Year])
VAR WeightedAllocation =
    DIVIDE(
      [Days in FC],
      [Days in Year])
RETURN

    WeightedAllocation
    *
    CALCULATE(
      SUM('Forecasted Sales Data'[Forecast] ) ,
     'Forecasted Sales Data'[Year] = CurrentYear )
```

Notice the results from step 9 are the exact same as the calculation in step 8. This expression is going to be easier to debug and easier to read / understand!

Year	Month	Region	Total Sales	Forecasted Sales
2005	July	Northwest	$58,586	$38,643.84
2005	August	Northwest	$43,956	$38,643.84
2005	September	Northwest	$46,657	$37,397.26
2005	October	Northwest	$62,546	$38,643.84
2005	November	Northwest	$80,819	$37,397.26
2005	December	Northwest	$122,639	$38,643.84

Module 14: Semi-Additive Measures

Module 14A Demo: Semi-Additive Measures

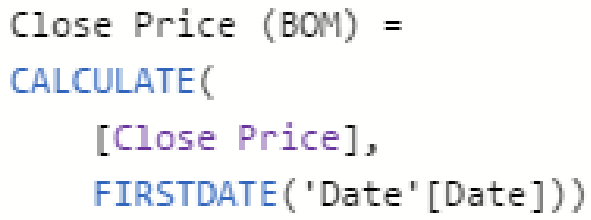
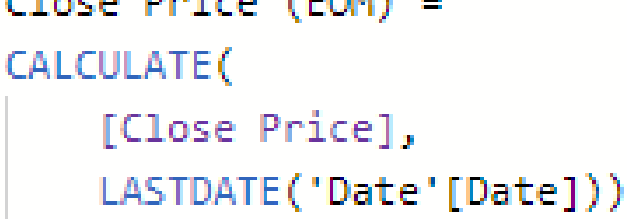
Semi-additive measures are measures that are not additive across all dimensions. Account Balance is a great example of a measure that is semi-additive because account balance can be added up across a customer dimension but not the time dimension. In this next demo we are going to review four functions in DAX that can help assist with writing semi-additive measures.

Module Requirements

1. Open the pbix file Module 14A found here C:\Dax Boot Camp\Class Labs\Module 14\Module 14A.pbix

Turn to next page for step-by-step instructions.

Year	Month	Close Price (BOM)	Close Price (EOM)	Close Price (BoM) Non Blank	Close Price (EoM) Non Blank
2012	April		$32.02	$32.42	$32.02
2012	May	$32.01	$29.19	$32.01	$29.19
2012	June	$28.45		$28.45	$30.59
2012	July		$29.47	$30.56	$29.47
2012	August	$29.41	$30.82	$29.41	$30.82
2012	September			$30.39	$29.76
2012	October	$29.49	$28.54	$29.49	$28.54
2012	November	$29.52	$26.62	$29.52	$26.62

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 14A.pbix,*** this file can be found at the following location: ***C:\Dax Boot Camp\Class Labs\Module 14\Module 14A.pbix*** 2. First, we are going to review a couple of functions that we have already seen in this course. 3. Create a new calculated measure using the following DAX formula: **Close Price (BOM) =** **CALCULATE(** **[Close Price],** **FIRSTDATE('Date'[Date]))**	
4. Now created a second calculated measure with the following DAX formula: **Close Price (EOM) =** **CALCULATE(** **[Close Price],** **LASTDATE('Date'[Date]))**	

5. Next add these two measures to the existing table on the report.

6. As you remember, the **LASTDATE** functions simply return the first or last day of the current filter context. In this example we are using stock market data and the stock market is not open every day of the week and month.

7. In a lot of scenarios these calculations would work well, but not here where we have a business with frequent closures.

Year	Month	Close Price (BOM)	Close Price (EOM)
2012	April		$32.02
2012	May	$32.01	$29.19
2012	June	$28.45	
2012	July		$29.47
2012	August	$29.41	$30.82
2012	October	$29.49	$28.54
2012	November	$29.52	$26.62
2012	December		$26.71
2013	January		$27.45
2013	February	$27.93	$27.80
2013	March	$27.95	
2013	April	$28.61	$33.10
2013	May	$32.72	$34.90
2013	July	$34.36	$31.84
Total			

8. What we would rather see here is the closing price of the first ***business*** day of the month and the last ***business*** day of the month. Fortunately, there are two functions in DAX designed for this purpose.

9. First, let's return the closing balance for the first business day of the month.

10. Create a new calculated measure using the following DAX formula:

```
Close Price (BoM) Non Blank =
CALCULATE(
    [Close Price],
    FIRSTNONBLANK('Date'[Date],
        [Close Price]))
```


```
Close Price (BoM) Non Blank =
CALCULATE(
    [Close Price],
    FIRSTNONBLANK('Date'[Date],
        [Close Price]))
```

11. Next, let's return the closing balance for the last business day of the month.

12. Create a new calculated measure using the following DAX formula:

Close Price (EoM) Non Blank =
CALCULATE(
[Close Price],
LASTNONBLANK('Date'[Date],
[Close Price]))

```
Close Price (EoM) Non Blank =
CALCULATE(
    [Close Price],
    LASTNONBLANK('Date'[Date],
        [Close Price]))
```

13. Finally add these two new calculations to the report as well for comparison. Now we can observe that the **FIRSTNONBLANK** and **LASTNONBLANK** functions return values where **FIRSTDATE** and **LASTDATE** did not.

Save this file as **Module 14A.**

Year	Month	Close Price (BOM)	Close Price (EOM)	Close Price (BoM) Non Blank	Close Price (EoM) Non Blank
2017	January		$64.65	$62.58	$64.65
2017	February	$63.58	$63.98	$63.58	$63.98
2017	March	$64.94	$65.86	$64.94	$65.86
2017	April			$65.55	$65.04
2016	January			$54.80	$55.09
2016	February	$54.71	$50.88	$54.71	$50.88
2016	March	$52.58	$55.23	$52.58	$55.23
2016	April	$55.57		$55.57	$49.87
2016	May		$53.00	$50.61	$53.00
2016	June	$52.85	$51.17	$52.85	$51.17
2016	July	$51.16		$51.16	$56.68

Notes	

Module 14B Demo: Semi-Additive Measures (Cont)

In the previous demo we reviewed a couple ways to return the closing balance, in this demo we are going to talk about the opening balance. Fortunately, DAX provides functions that we can use to obtain the opening balance.

Module Requirements

1. Open the pbix file Module 14A found here C:\Dax Boot Camp\Completed Labs\Module 14\Module 14A.pbix

Turn to next page for step-by-step instructions.

Year	Month	Close Price (OBM)	Closing Price (BOM) Calculate
2017	April	$65.86	$65.86
2017	May		$65.04
2017	February	$64.65	$64.65
2017	March	$63.98	$63.98
2017	January		$62.14
2016	December	$60.26	$60.26
2016	November	$59.92	$59.92
2016	October	$57.60	$57.60
2016	September	$57.46	$57.46
2016	August		$56.68
2016	January	$55.48	$55.48
2016	April	$55.23	$55.23
2016	February		$55.09
Total			**$65.04**

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 14A.pbix,*** this file can be found at the following location: ***C:\Dax Boot Camp\Completed Labs\Module 14A.pbix*** 2. First, create two new measures using the following DAX formulas: **Close Price (OBM) =** **OPENINGBALANCEMONTH(** **[Close Price], 'Date'[Date])** **Close Price (OBY) =** **OPENINGBALANCEYEAR(** **[Close Price], 'Date'[Date])**	``` Close Price (OBY) = OPENINGBALANCEYEAR([Close Price], 'Date'[Date]) Close Price (OBM) = OPENINGBALANCEMONTH([Close Price], 'Date'[Date]) ```

3. Add both new measures to the current table on the report.

4. From looking at the table it's obvious that we have the same problem that we had in the previous demo. The **OPENINGBALANCE** functions do not account for situations where there may be blanks in the data.

5. It's also important to mention that these **OPENINGBALANCE** functions take the last day of the previous period.

Year	Month	Close Price (OBM)	Close Price (OBY)
2012	May	$32.02	
2012	June	$29.19	
2012	August	$29.47	
2012	September	$30.82	
2012	November	$28.54	
2012	December	$26.62	
2013	January	$26.71	$26.71
2013	February	$27.45	$26.71
2013	March	$27.80	$26.71
2013	April		$26.71
2013	May	$33.10	$26.71
2013	June	$34.90	$26.71
2013	July		$26.71

6. Unfortunately, we do not have a non-blank version of these functions. In order to make these calculations work we need to write slightly more complex calculations.

7. Create a new calculated measure using the following DAX formula:

PARALLELPERIOD vs. PREVIOUSMONTH/YEAR/QUARTER

```
CALCULATE (
       [Close Price],
        LASTNONBLANK(
                 PARALLELPERIOD (
                'Date'[Date],
                    -1,
                MONTH ),
        [Close Price] ) )
```

```
Closing Price (BOM) Calculate =
CALCULATE (
        [Close Price],
        CALCULATETABLE(
                LASTNONBLANK(
                        'Date'[Date],
                        [Close Price]),
                PARALLELPERIOD (
                    'Date'[Date],
                    -1,
                    MONTH )))
```

8. The previous expression can be further simplified by replacing PARALLEPERIOD with PREVIOUSMONTH.

```
Stock Price (OBM) -nb =
CALCULATE(
    [Close Price],
    LASTNONBLANK(
        PREVIOUSMONTH('Date'[Date]),
        [Close Price] ) )
```

PREVIOUSMONTH is a derivative of PARELLPERIOD.

9. The **PARALLELPERIOD** function returns a list of dates from the previous month.

10. **LASTNONBLANK** then returns the last non blank value from the previous month.

11. Finally calculate returns the closing price for the date returned from the last non blank function.

Year	Month	Close Price (OBM)	Closing Price (BOM) Calculate
2017	April	$65.86	$65.86
2017	May		$65.04
2017	February	$64.65	$64.65
2017	March	$63.98	$63.98
2017	January		$62.14
2016	December	$60.26	$60.26
2016	November	$59.92	$59.92
2016	October	$57.60	$57.60
2016	September	$57.46	$57.46
2016	August		$56.68
2016	January	$55.48	$55.48
2016	April	$55.23	$55.23
2016	February		$55.09
Total			**$65.04**

**Student Challenge

Return the Opening Balance for the year, rewriting **Close Price (OBY)** from step 2. This new calculation should return the opening balance even when blanks are present in the dataset!

12. Congratulations, you have completed module 14B.

Notes	

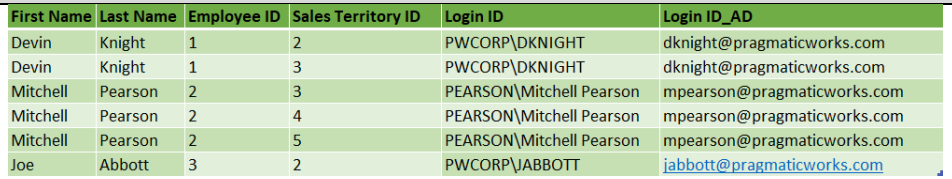
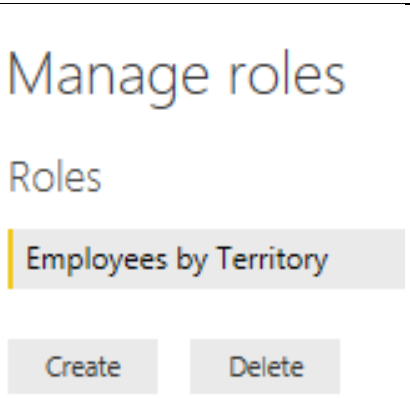
Module 15: Dynamic Security

Module 15 Demo: Dynamic Security

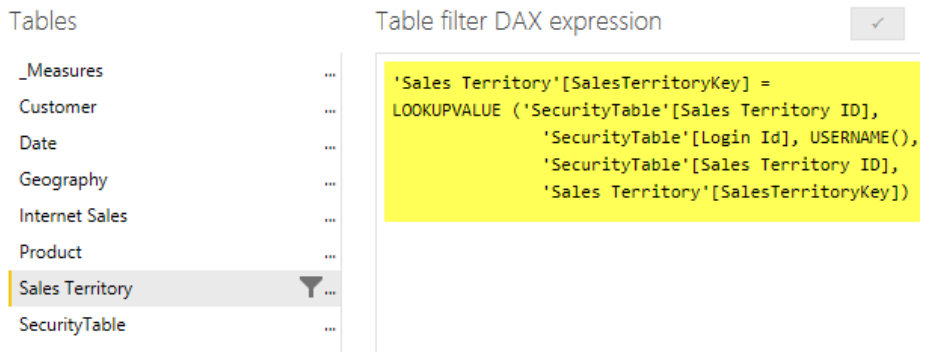
In this demo we want to take security a step further. The previous demo works great if you only need to set up and configure a few roles, but what if you need to create 20 roles, or 50, or 100? Then you need to take a more dynamic approach at Row Level Security.

Module Requirements

1. We are going to be using Module 15 for this demo. C:\Dax Boot Camp\Completed Labs\Module 15\
2. We will also be using the file **Security Table.xlsx** for this demo. C:\DAX Boot Camp\Data & Module Resources\Module 15

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. First open the file Module 15.pbix found at C:\Dax Boot Camp\Class Labs\Module 15\ 2. For dynamic security to work we need to have a table that stores all the information. For example, this table will store the name of each user logging in and what sales territory regions they have access to. 3. First, open and look at the table that stores our security information. The file is called Security Table.xlsx and is found at: *C:\DAX Boot Camp\Data & Module Resources\Module 15* 4. This table is similar to the table you will need to create, we have added some extra columns to explain the keys.	 Figure 1
5. Now let's import this table into Power BI Desktop. 6. Add the Security Table into the data model by going to the Get Data drop down and selecting Excel. 7. No relationships need to be defined in the data model, we are going to use DAX and the LOOKUPVALUE function in lieu of using relationships in the model. 8. Check the relationship view, if any relationships were created, remove them. 9. Hide the table from the report view.	

First Name	Last Name	Employee ID	Sales Territory ID	Login ID	Login ID_AD
Devin	Knight	1	2	PWCORP\DKNIGHT	dknight@pragmaticworks.com
Devin	Knight	1	3	PWCORP\DKNIGHT	dknight@pragmaticworks.com
Mitchell	Pearson	2	3	PEARSON\Mitchell Pearson	mpearson@pragmaticworks.com
Mitchell	Pearson	2	4	PEARSON\Mitchell Pearson	mpearson@pragmaticworks.com
Mitchell	Pearson	2	5	PEARSON\Mitchell Pearson	mpearson@pragmaticworks.com
Joe	Abbott	3	2	PWCORP\JABBOTT	jabbott@pragmaticworks.com

10. Now it's time to create a new role. 11. Select "Manage Roles from the Modeling ribbon. 12. Click Create to start a new role and then name it "**Employees by Territory**".	
13. Click on the Sales Territory table and then paste the following code into the DAX expression window: **'Sales Territory'[SalesTerritoryKey] =** LOOKUPVALUE ('SecurityTable'[Sales Territory ID], 'SecurityTable'[Login Id], USERNAME(), 'SecurityTable'[Sales Territory ID], 'Sales Territory'[SalesTerritoryKey]) 14. Click Save.	

Notes	

Module 16: xVelocity

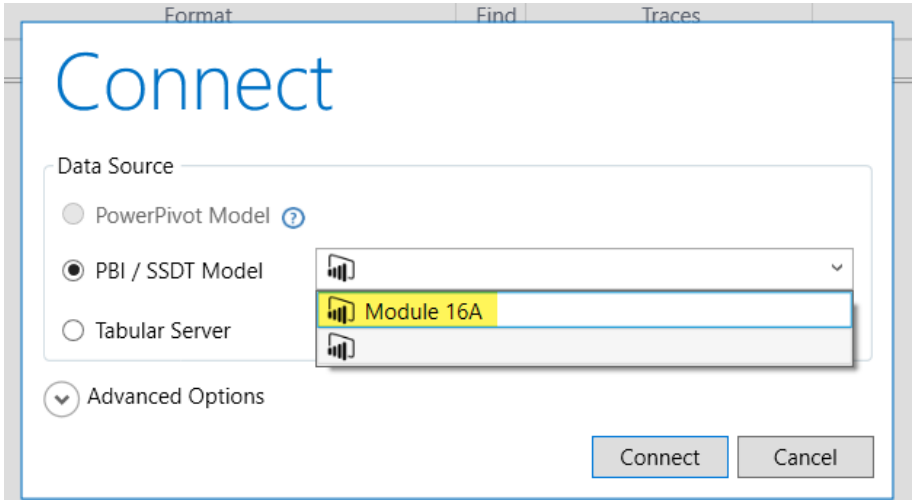
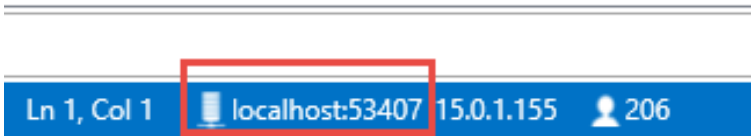
Module 16A Demo: Vertipaq Analyzer

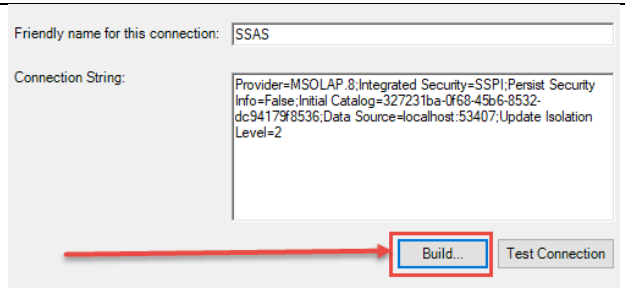
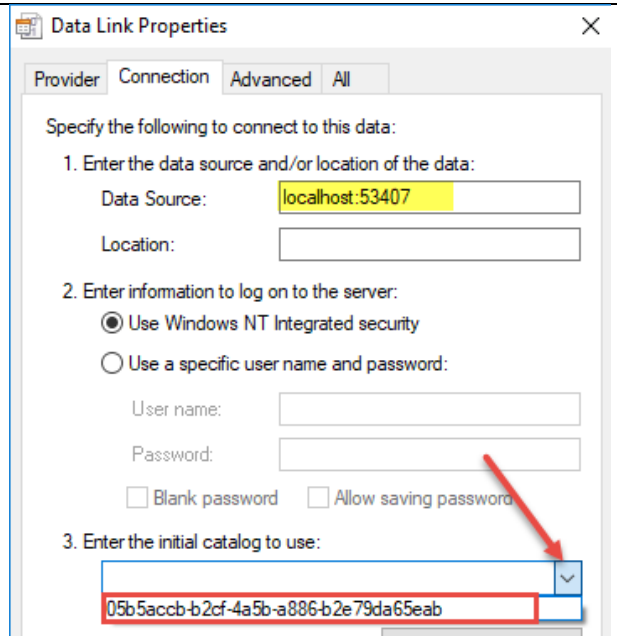
In this demo we are going to show how to model a monthly budget and compare that with daily sales values.

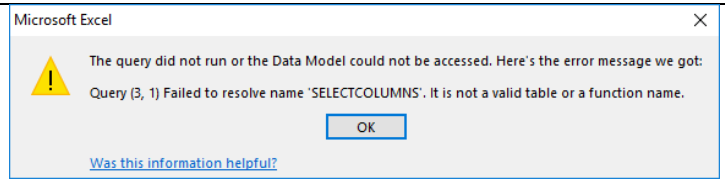
Module Requirements

1. Open the .pbix file Module 16A found here C:\DAX Boot Camp\Data & Module Resources\Module 16\Module 16A.pbix

Turn to next page for step by step instructions.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 16A.pbix,*** this file can be found at the following location: ***C:\DAX Boot Camp\Data & Module Resources\Module 16\Module 16A.pbix*** *2.* Once Module 16A has been opened then go ahead and open **DAX Studio**. DAX Studio can be found in your Additional Resources folder. Run the setup file provided. ***DaxStudio_2_7_1_setup.exe*** 3. The first thing you will see is the "Connect" screen. Select the radial button for PBI / SSDT Model and then select Module 16A from the drop-down box. 4. Click Connect.	
5. For now, the only thing we are using DAX Studio for is to get the connection information necessary so that we can connect to our workbook from Vertipaq Analyzer. 6. There is a blue bar across the bottom of the screen that contains the server name for this .pbix file. We will need this very soon, leave DAX Studio open.	
7. Next, launch the Vertipaq Analyzer. 8. The Vertipaq Analyzer can be found in your Data & Module Resources directory. Open the one titled: **<u>VertiPaq Analyzer 1.92 – 1200.xlsm</u>**	

9. The first thing we need to do is connect this workbook to the Power BI Desktop file we want to analyze. 10. Navigate to the POWERPIVOT ribbon and then select Manage. This will launch Power Pivot where we can change the connection information to point to our data model.	
11. With the Power Pivot window now open, click on Existing Connections. 12. Click on **SSAS**, located under PowerPivot Data Connections and the click on **Edit** found at the bottom. This will launch the Edit Connection dialog box. 13. Click on the Build button found below the connection string information dialog box.	
14. Update the data source to reflect the connection information provided from DAX Studio. (localhost:53407) 15. Next, you must select the initial catalog to use, each workbook generates its' own GUID. Click the drop-down box for step 3 and select the only GUID available. 16. Click OK to exit the Data Link Properties box. 17. Click Save to close out the Edit Connection box. 18. Click Close to close out of the Existing Connections box. 19. Finally, click save at the top left of the Power Pivot window.	

20. Now you are back in excel, navigate to the data ribbon and select **Refresh All.** 21. You will usually get a warning message at this point, just click OK, you may have to click ok a few times.	
22. The Vertipaq Analyzer gives us a lot of great information about our data model. We can see the how many unique values are in a column, also known as cardinality. We can also see the size of a table and the size of the columns in a table. We can also see the dictionary size and the encoding method used (compression algorithm). 23. Navigate to the columns tab, these columns are sorted in descending order so I can see at a quick glance which columns in my data model are taking up the most space. 24. We will use this model in the next couple of demos to try and optimize our data model.	

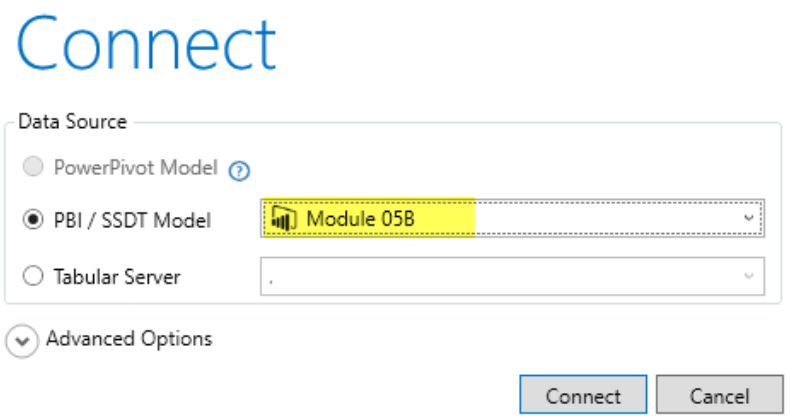
Module 16B Demo: Column Cardinality

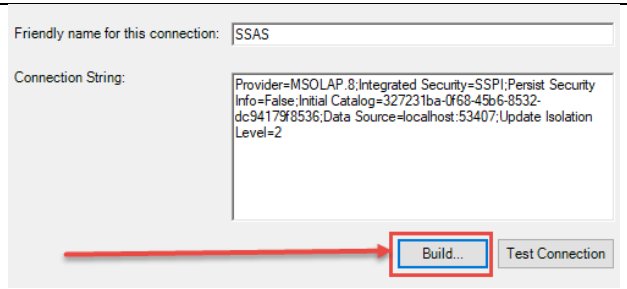
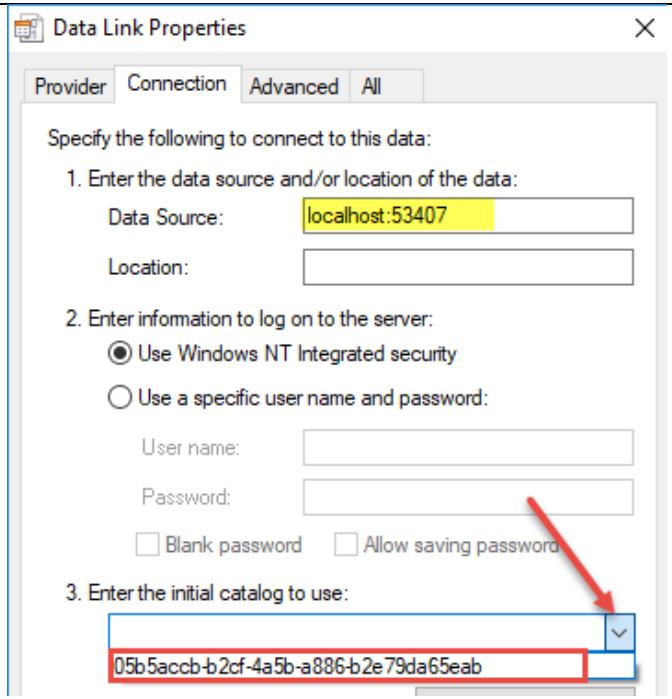
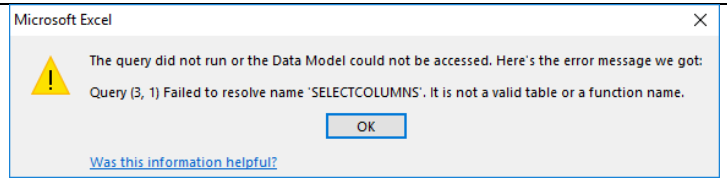
In this demo we are going to show how to model a monthly budget and compare that with daily sales values.

Module Requirements

1. Open the .pbix file Module 16B found here C:\DAX Boot Camp\Data & Module Resources\Module 16\Module 16B.pbix

Turn to next page for step by step instructions.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 16B.pbix,*** this file can be found at the following location: ***C:\DAX Boot Camp\Data & Module Resources\Module 16 \Module 16B.pbix*** 2. Once Module 16B has been opened then go ahead and open **DAX Studio**. DAX Studio can be found in your Additional Resources folder. Run the setup file provided. 3. Connect to Module 16B, similar to how we previously connected to Module 16A. 4. Module 16A and 16B are identical with one exception. Module 16B has the address column broken up into several columns, whereas module 16A has them all combined as a single column.	
5. Next, launch the Vertipaq Analyzer. 6. The Vertipaq Analyzer can be found in your Data & Module Resources directory. Open the one titled: **<u>VertiPaq Analyzer 1.92 – 1200.xlsm</u>** 7. The first thing we need to do is connect this workbook to the Power BI Desktop file we want to analyze. 8. Navigate to the POWERPIVOT ribbon and then select Manage. This will launch Power Pivot where we can change the connection information to point to our data model.	

9. With the Power Pivot window now open, click on Existing Connections. 10. Click on **SSAS**, located under PowerPivot Data Connections and the click on **Edit** found at the bottom. This will launch the Edit Connection dialog box. 11. Click on the Build button found below the connection string information dialog box.	
12. Update the data source to reflect the connection information provided from DAX Studio. This should reflect the most recent connection provided when you connected to Module 05B. 13. Next, you must select the initial catalog to use, each workbook generates its' own GUID. Click the drop-down box for step 3 and select the only GUID available. 14. Click OK to exit the Data Link Properties box. 15. Click Save to close out the Edit Connection box. 16. Click Close to close out of the Existing Connections box. 17. Finally, click save at the top left of the Power Pivot window.	
18. Now you are back in excel, navigate to the data ribbon and select **Refresh All.** 19. You will usually get a warning message at this point, just click OK, you may have to click ok a few times.	

20. Navigate over to the columns tab and you will notice that the Customer-Address column is no longer at the top of the list, this is because the column has been removed and broken up into several smaller columns with more duplicate values.

21. The only way to really know that our column is taking up less space is to find the individual columns and sum up their totals. Take a look at the second screenshot provided. Clearly, the sum of the separate columns is less than the previous total.

Module 05A:

TableColumn	Rows	Cardinality	Columns Total Size	Data Size	Dictionary Size
LocalDateTable_f807ea2c-971d-4009-914b-7a4d61e6c	25,933	25,933	1,629,080	51,872	1,369,704
LocalDateTable_8122d3f7-4b06-4c46-b4c5-0c5aba01d	25,933	25,933	1,629,080	51,872	1,369,704
Customer-Address	18,484	18,483	1,285,785	36,976	1,100,905

Module 05B:

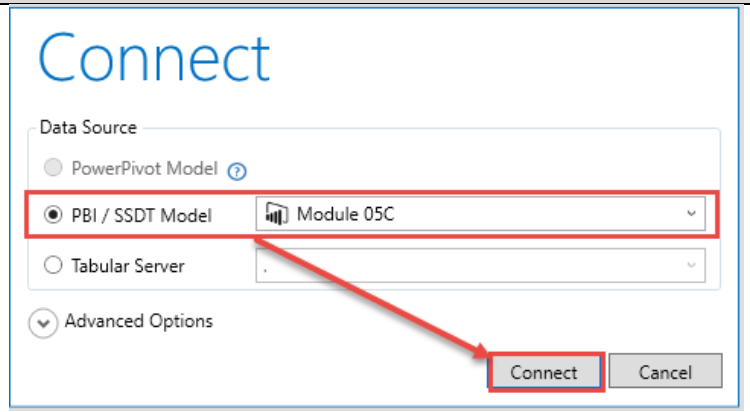
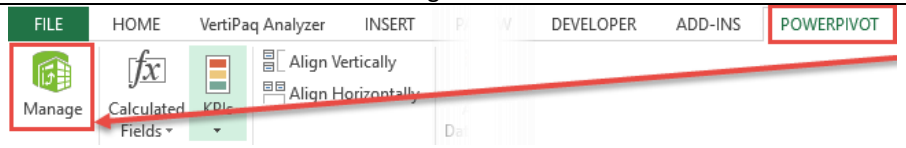
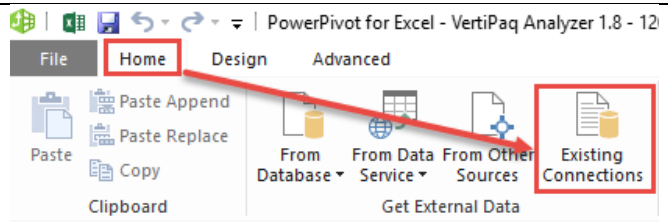
TableColumn	Rows	Cardinality	Columns Total Size	Data Size	Dictionary Size
Customer-AddressLine1	18,484	12,797	653,845	36,976	514,453
Customer-City	18,484	269	47,938	21,128	24,618
Customer-Postal Code	18,484	323	47,350	21,128	23,598
Customer-State	18,484	53	28,068	9,032	18,572
Customer-AddressLine2	18,484	167	22,180	320	20,484

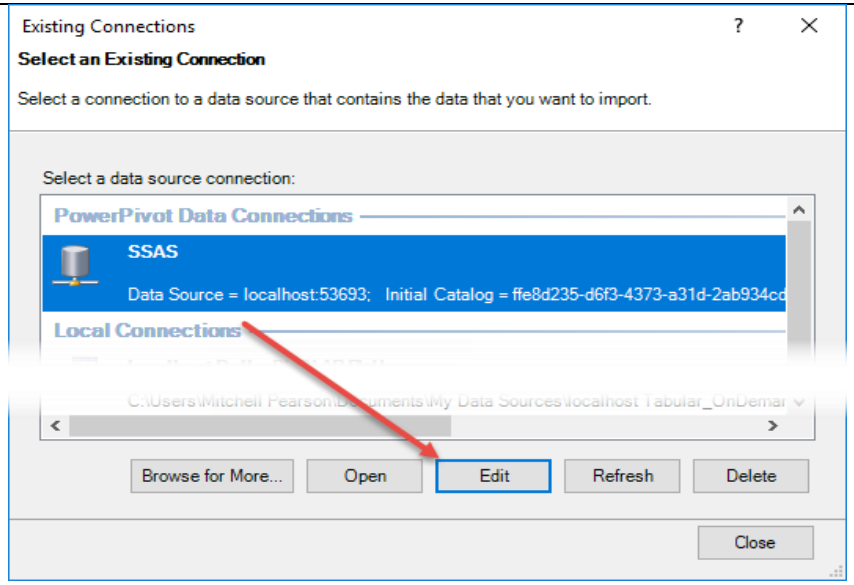
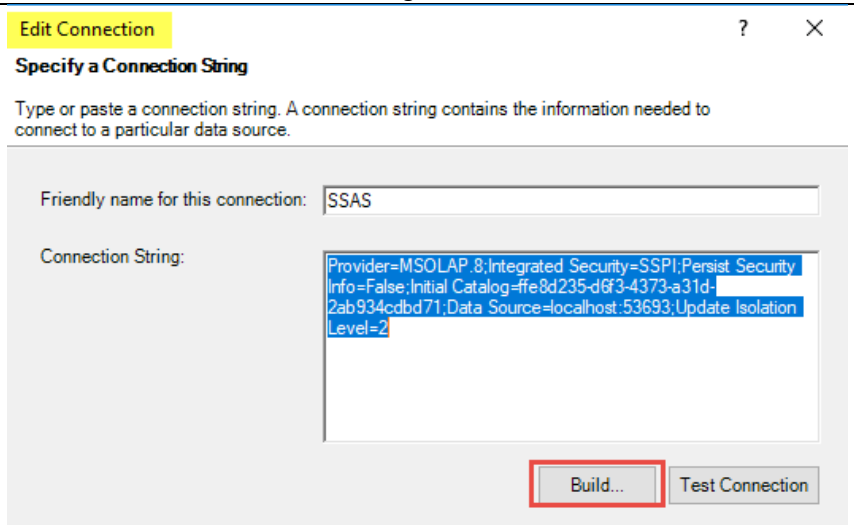
Module 16C Lab: xVelocity (Vertipaq Analyzer)

Your end users have complained about processing and query performance with the data model that they designed. Before you can research any of the performance problems you first need to connect to the .pbix file from Vertipaq Analyzer.

Module Requirements

1. Open the .pbix file Module 16C found here C:\DAX Boot Camp\Data & Module Resources \Module 16\Module 16C.pbix
2. Retrieve the connection information to the workbook using DAX Editor.
3. Connect to the workbook using Vertipaq Analyzer.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open the Module 05C file from the Class Labs folder. ***C:\DAX Boot Camp\Data & Module Resources \Module 16\Module 16C.pbix*** 2. Open DAX Studio and connect to the Power BI file.	 Figure 1
3. Get the connection information from the bottom right, this value will be different for you and is unique for each connection.	 Figure 2
4. Now that you have the connection information open Vertipaq Analyzer 1.92 -1200.xlsm, this file is found under additional resources in your class files. If you have downloaded a more recent version, then use that. 5. Once Veritpaq Analyzer opens, navigate to the Power Pivot tab and click Manage.	 Figure 3
6. Once the Power Pivot for Excel window has launched select “Existing Connections”.	 Figure 4

7. Select SSAS and then click on Edit.	 Figure 5
8. This will launch the Edit Connection dialog box. From this box select Build…	 Figure 6

9. For the last two steps we now need to give Vertipaq analyzer the connection information necessary to connect to our data model.

10. First update the Data Source property, this value should match the value you retrieved from DAX Studio (Step 3).

11. My connection information was **localhost: 51507**

12. Once the connection is established you need to choose the initial catalog (database). This will be a unique GUID that references your .pbix file. Just select the only option available from the drop-down box. This step is easy to forget and will result in connection errors.

13. Click OK. Click Save. Click Close.

14. Finally, close out the Power Pivot for Excel window, just click the X at the top right-hand corner of the screen.

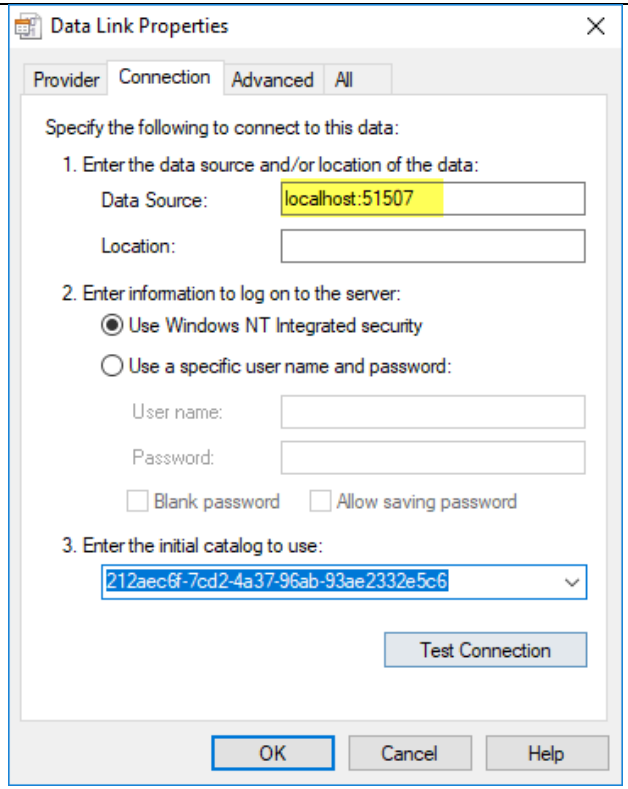


Figure 7

15. Once the connection information has been updated accordingly, we need to refresh the data. Navigate to the Data ribbon across the top and click “Refresh All”.

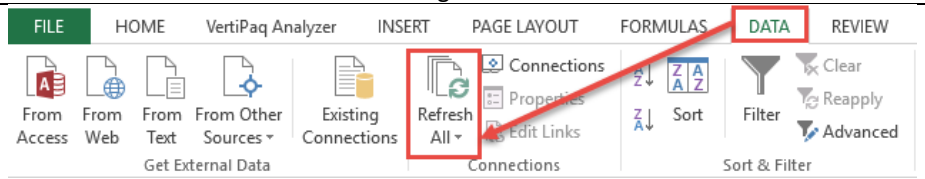


Figure 8

16. It is very common to get an error message at this point, click ok for each error message that occurs. Congratulations, you have now connected to your Power BI Workbook!

17. Keep this workbook open. This will be used for the next lab.

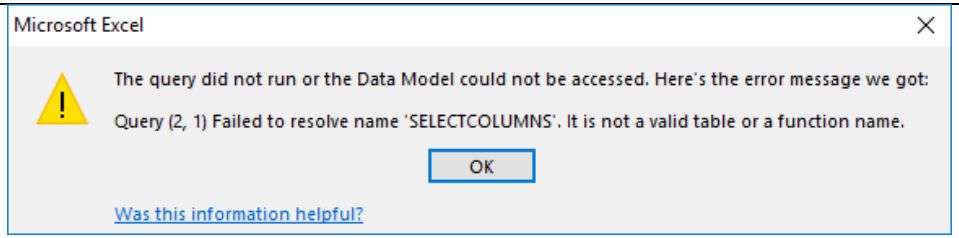


Figure 9

Module 16D Lab: xVelocity (Column Cardinality)

Your end users have complained about processing and query performance with the data model that they designed. Help your end users out by looking at their data model and improving overall compression and performance. Try to complete this example without looking at the hints provided.

Module Requirements

1. Pick back up from where you left on the last lab.
2. Reduce the overall size of the data model by using Vertipaq Analyzer to identify large columns, determine which columns can be adjusted to reduce the column size.
3. Not all columns can be changed or modified. Look for columns that have a large number of distinct values and make the necessary adjustments to reduce their size. Feel free to remove columns that are not used in the data model, typically this would be key columns that are not being used for relationships and would not be used in the visualization layer.
4. This exercise is somewhat subjective, different students will come up with different solutions based on their experiences and what they feel is or is not necessary, there is not necessarily a correct or incorrect answer. The primary purpose of this exercise is to get the student more familiar with analyzing their data model and making adjustments.

TableColumn	Rows	Cardinality	Columns Total Size	Data Size	Dictionary Size
City-City Key	133,107	133,107	4,049,756	354,960	2,629,900
City-Location	133,107	37,940	2,241,894	266,216	1,672,110
City-City, State	133,107	37,941	2,034,488	266,216	1,464,704
Sale-Sale Key	222,091	222,091	1,480,752	592,248	120
Sale-WWI Invoice ID	222,091	67,274	861,488	592,248	120
City-WWI City ID	133,107	37,941	418,120	266,216	120
City-Latest Recorded Population	133,107	9,327	412,024	374,576	120
Sale-Invoice Date Key	222,091	1,043	408,896	355,352	45,160
Sale-Delivery Date Key	222,091	1,042	408,792	355,352	45,056
Sale-City Key	222,091	1,473	406,292	355,352	39,116
Sale-Salesperson Key	222,091	108	184,664	180,936	2,816
Sale-Description	222,091	227	177,484	148,624	27,004
Sale-Customer Key	222,091	403	175,364	161,672	10,428
Sale-Profit	222,091	570	159,912	133,160	22,144
Sale-Stock Item Key	222,091	227	155,804	148,624	5,324
Sale-Tax Amount	222,091	493	137,856	112,616	21,256

Hints

1. Remember that column uniqueness can be reduced in several ways. One way of reducing column uniqueness is to simply split a column into multiple separate columns.

Step-by-Step Instructions

Click Steps	Screen Shots
1. In order to find out where to start, we are going to use Vertipaq analyzer. 2. In Vertipaq Analyzer, go to the columns tab across the bottom of the workbook. 3. The first non-key column you see is City-Location. 4. Now we must go back to our workbook and see if we need this column or if we can reduce its' size.	

TableColumn	Rows	Cardinality	Columns Total Size	Data Size	Dictionary Size
City-City Key	133,107	133,107	4,049,756	354,960	2,629,900
City-Location	133,107	37,940	2,241,894	266,216	1,672,110
City-City, State	133,107	37,941	2,034,488	266,216	1,464,704
Sale-Sale Key	222,091	222,091	1,480,752	592,248	120
Sale-WWI Invoice ID	222,091	67,274	861,488	592,248	120
City-WWI City ID	133,107	37,941	418,120	266,216	120
City-Latest Recorded Population	133,107	9,327	412,024	374,576	120
Sale-Invoice Date Key	222,091	1,043	408,896	355,352	45,160
Sale-Delivery Date Key	222,091	1,042	408,792	355,352	45,056
Sale-City Key	222,091	1,473	406,292	355,352	39,116
Sale-Salesperson Key	222,091	108	184,664	180,936	2,816
Sale-Description	222,091	227	177,484	148,624	27,004
Sale-Customer Key	222,091	403	175,364	161,672	10,428
Sale-Profit	222,091	570	159,912	133,160	22,144
Sale-Stock Item Key	222,091	227	155,804	148,624	5,324
Sale-Tax Amount	222,091	493	137,856	112,616	21,256

Figure 1

Click Steps	Screen Shots
5. There is nothing we can do to reduce the size of the Location column in the city table, but we can remove this column as it is an unnecessary column. 6. Right click on the Location column and delete it. 7. Remember, as a best practice, to perform these operations as far upstream as possible. In this example you can delete the column in Power BI Desktop or you can launch the Query Editor and delete the column there. The latter option is preferred in this situation.	

Location
0xE6100000010CEA76F69507244140CA3D1350176B55C0
0xE6100000010C4E6F35A1FF3C41401A3C026ECCD953C0
0xE6100000010C78CCE5AB89E34240E25B5837DE2053C0
0xE6100000010C250FFA884E67404051700C5C79F454C0
0xE6100000010C920DFF44C02A43402C6D2700354B54C0
0xE6100000010C3FB9B76CBE464040B463D982942F54C0
0xE6100000010C6E6C76A4FA823D40C85BAE7E6C8654C0
0xE6100000010CC9DD318683704040B0EA07D04AAA55C0
0xE6100000010CED1B4E4F0ED0404023DD2AE3BA2554C0

Figure 2

8. The next non-key column we see that is taking up a lot of space is the City, State column. This column can be split into two separate columns.

TableColumn	Rows	Cardinality	Columns Total Size	Data Size	Dictionary Size
City-City Key	133,107	133,107	4,049,756	354,960	2,629,900
City-Location	133,107	37,940	2,241,894	266,216	1,672,110
City-City, State	133,107	37,941	2,034,488	266,216	1,464,704
Sale-Sale Key	222,091	222,091	1,480,752	592,248	120
Sale-WWI Invoice ID	222,091	67,274	861,488	592,248	120
City-WWI City ID	133,107	37,941	418,120	266,216	120
City-Latest Recorded Population	133,107	9,327	412,024	374,576	120
Sale-Invoice Date Key	222,091	1,043	408,896	355,352	45,160
Sale-Delivery Date Key	222,091	1,042	408,792	355,352	45,056
Sale-City Key	222,091	1,473	406,292	355,352	39,116
Sale-Salesperson Key	222,091	108	184,664	180,936	2,816
Sale-Description	222,091	227	177,484	148,624	27,004
Sale-Customer Key	222,091	403	175,364	161,672	10,428
Sale-Profit	222,091	570	159,912	133,160	22,144
Sale-Stock Item Key	222,091	227	155,804	148,624	5,324
Sale-Tax Amount	222,091	493	137,856	112,616	21,256

Figure 3

9. Launch the Query Editor and split the column into two separate columns (City and State).

10. Right click on the City, State column and go to split column → By Delimiter…

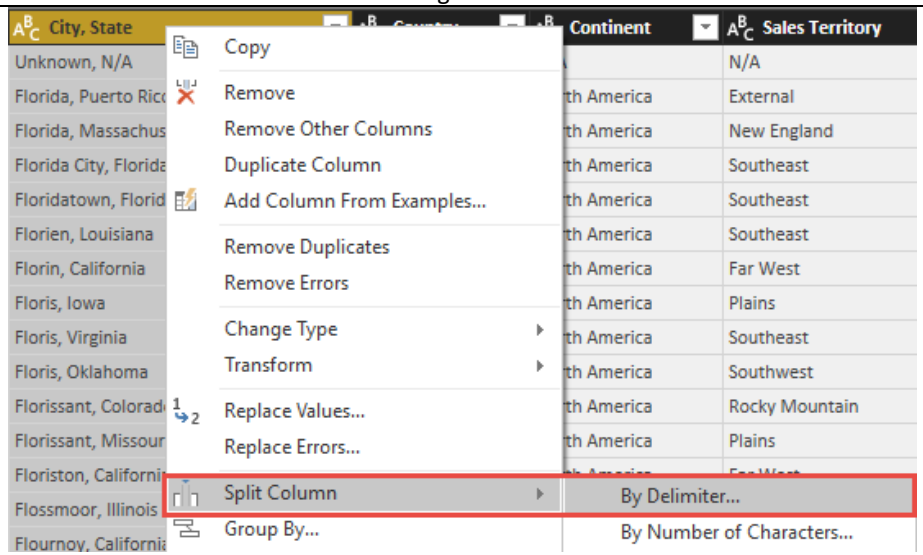


Figure 4

11. This will launch the split column by delimiter editor.

12. Select “Custom” for the type of delimiter and enter a comma followed by a space into the text box.

13. Rename the columns City and State. Close and apply all changes.

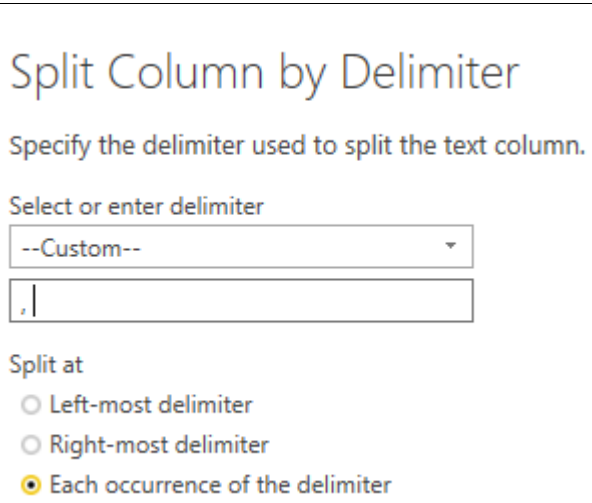


Figure 5

14. The next non-key column we find is City-Latest Recorded Population.

15. Look at the data stored in this column, it stores the last known population for each city. This produces a very high number of distinct values and it has a very large range of values making it difficult to get good compression. Subjectively we have to analyze whether this is a column we want to keep in the data model or remove from the data model.

16. Launch the Query Editor and remove this column from the data model.

17. Hit close and apply.

TableColumn	Rows	Cardinality	Columns Total Size	Data Size	Dictionary Size
City-City Key	133,107	133,107	4,049,756	354,960	2,629,900
City-Location	133,107	37,940	2,241,894	266,216	1,672,110
City-City, State	133,107	37,941	2,034,488	266,216	1,464,704
Sale-Sale Key	222,091	222,091	1,480,752	592,248	120
Sale-WWI Invoice ID	222,091	67,274	861,488	592,248	120
City-WWI City ID	133,107	37,941	418,120	266,216	120
City-Latest Recorded Population	133,107	9,327	412,024	374,576	120
Sale-Invoice Date Key	222,091	1,043	408,896	355,352	45,160
Sale-Delivery Date Key	222,091	1,042	408,792	355,352	45,056
Sale-City Key	222,091	1,473	406,292	355,352	39,116
Sale-Salesperson Key	222,091	108	184,664	180,936	2,816
Sale-Description	222,091	227	177,484	148,624	27,004
Sale-Customer Key	222,091	403	175,364	161,672	10,428
Sale-Profit	222,091	570	159,912	133,160	22,144
Sale-Stock Item Key	222,091	227	155,804	148,624	5,324
Sale-Tax Amount	222,091	493	137,856	112,616	21,256

Figure 6

18. The next non-key column we find is Sale-Description.

TableColumn	Rows	Cardinality	Columns Total Size	Data Size	Dictionary Size
City-City Key	133,107	133,107	4,049,756	354,960	2,629,900
City-Location	133,107	37,940	2,241,894	266,216	1,672,110
City-City, State	133,107	37,941	2,034,488	266,216	1,464,704
Sale-Sale Key	222,091	222,091	1,480,752	592,248	120
Sale-WWI Invoice ID	222,091	67,274	861,488	592,248	120
City-WWI City ID	133,107	37,941	418,120	266,216	120
City-Latest Recorded Population	133,107	9,327	412,024	374,576	120
Sale-Invoice Date Key	222,091	1,043	408,896	355,352	45,160
Sale-Delivery Date Key	222,091	1,042	408,792	355,352	45,056
Sale-City Key	222,091	1,473	406,292	355,352	39,116
Sale-Salesperson Key	222,091	108	184,664	180,936	2,816
Sale-Description	222,091	227	177,484	148,624	27,004
Sale-Customer Key	222,091	403	175,364	161,672	10,428
Sale-Profit	222,091	570	159,912	133,160	22,144
Sale-Stock Item Key	222,091	227	155,804	148,624	5,324
Sale-Tax Amount	222,091	493	137,856	112,616	21,256

19. Look at the column in the data model.

20. This seems like an odd column to be in a fact table, if we check our Stock Item table, we will find the same column exists there as well. This column does not need to be in two tables.

21. Delete the Description column from the Sale table. Remove this column from the Query Editor, not PBI desktop.

22. Congratulations! You have successfully cleaned up some of the larger columns in the database.

23. Please note, there is more work that can be done! Some key and ID columns may not be used in the data model, these can be removed.

24. Also, feel free to save and close this updated model. After you close the model, open it back up and reconnect to it via Dax Studio and the Vertipaq Analyzer to see your changes.

Description
Developer joke mug - understanding recursion requires understan
DBA joke mug - I will get you in order (White)
DBA joke mug - I will get you in order (Black)
Developer joke mug - that's a hardware problem (White)
Developer joke mug - there are 10 types of people in the world (Bl
Developer joke mug - that's a hardware problem (Black)
Developer joke mug - there are 10 types of people in the world (W
DBA joke mug - mind if I join you? (White)
Developer joke mug - there are 10 types of people in the world (W
DBA joke mug - two types of DBAs (Black)
Developer joke mug - inheritance is the OO way to become wealth
DBA joke mug - mind if I join you? (Black)
DBA joke mug - daaaaaa-ta (White)
Developer joke mug - old C developers never die (Black)

Figure 7

Module 16E Lab: xVelocity (Disable Load)

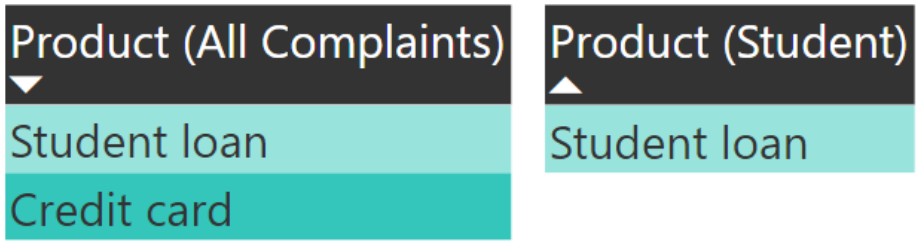
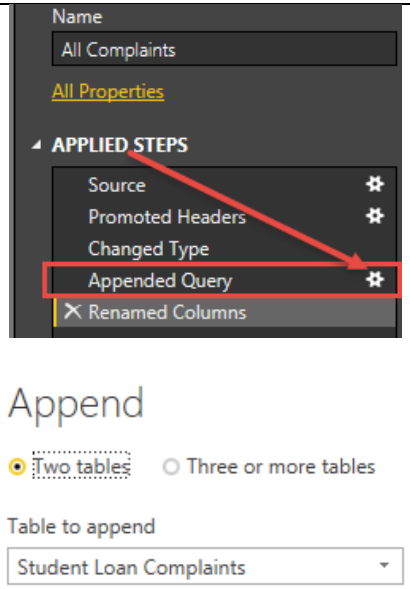
Your manager has asked you to look at a data model created by a new Power BI Developer on the team. He noticed that the loan complaints showed up in two separate tables. Investigate the data model Module 16D and fix the issue.

Module Requirements

1. Module 16E is found in the following folder: C:\DAX Boot Camp\Data & Module Resources\Module 16\Module 16E.pbix
2. Look at the data model and determine why Loan Complaints are showing up in two separate tables.
3. Loan complaints should not be appearing in two tables, fix the data model so that this does not occur.

Hints

1. Remember that sometimes tables are brought into the data model as helper tables, these tables do not necessarily need to be loaded into PBI Desktop. To prevent these tables from being loaded into PBI Desktop the load can be disabled.

Step-by-Step Instructions	
Click Steps	**Screen Shots**
1. Open ***Module 16E.pbix***, this file can be found at the following location: ***C:\DAX Boot Camp\Data & Module Resources\Module 16\Module 16E.pbix*** 2. Take a closer look at both tables in the data model. You will see the column Product in both tables stores the type of complaint. In the Student loan table you see products of the type "Student Loan". 3. In the All Complaints table you will see products of the type "Student Loan" and "Credit Card". This can be confusing.	
4. Launch the Query Editor, here we can see that the reason that student loan complaints appears in both tables is because the Student Loan table was appended to the All Complaints table. 5. Click on the settings wheel next to Appended Query to see the append operation. 6. Now that we have determined that the student complaints table was appended with the credit card complaints we can disable the load for the student complaints table.	

7. Right click on the student loan complaints table and uncheck enable load.

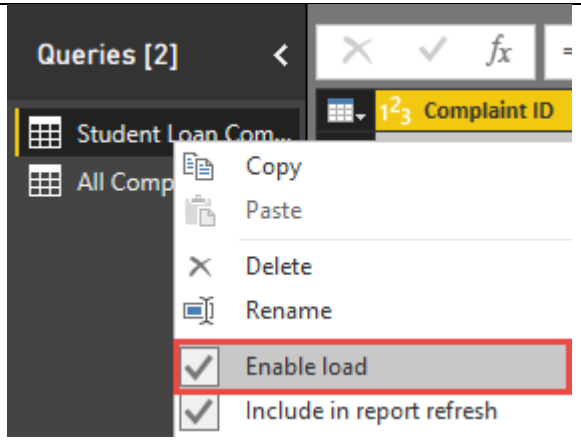


Figure 3

8. Disabling the load will give a warning prompt. This prompt is letting you know that if any visuals in your report pane are using any columns from the table those visuals will be broken.

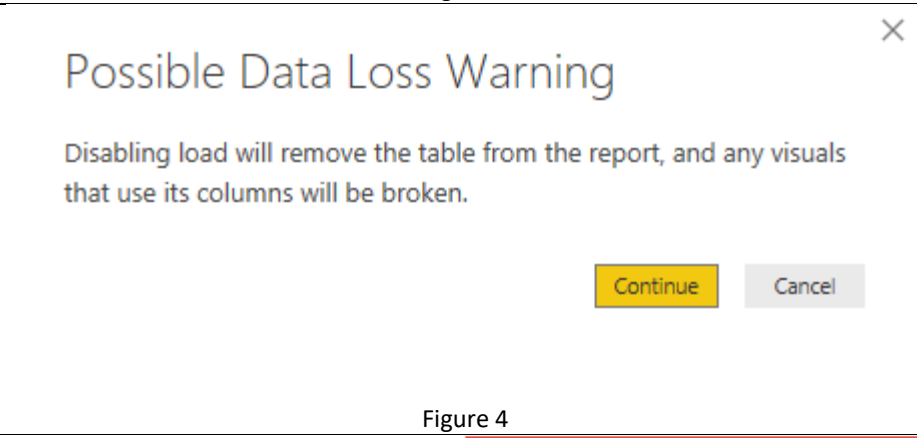


Figure 4

9. Click continue on the warning prompt and then select Close & Apply from the home ribbon.

10. The report is now complete, we do have a visual which is invalid now. Click on the visual and delete it.

11. Congratulations, you have completed this module. Feel free to save your work as Lab 16E.

Figure 5