Designing a Data Model in Power BI Desktop

Lab Time: 60 minutes

Lab Folder: C:\Student\Modules\03_DataModeling\Lab

Lab Overview: In this set of lab exercises, you will continue to work on the Power BI Desktop project you started in the previous lab. At this point, you have already imported sales data from a SQL Azure database into the dataflow named **Wingtip Sales Data** and transformed it using the Power Query features in the browser. You have also imported the **Wingtip Sales Data** dataflow entities into a Power BI Desktop project named **Wingtip Sales Model.pbix** which will be your starting point for this lab. The goal of this lab is for you to use the data modeling tools of Power BI Desktop and DAX to write expressions for calculated columns and measures. Along the way, you will build a few simple reports and add visuals so you can see the effects of your data modeling efforts.

Lab Dependencies: This lab assumes you completed the previous lab titled **Designing Dataflows to Extract and Transform Data** in which you created a Power BI Desktop project named **Wingtip Sales Model.pbix**. In the previous lab you should have imported data into the data model using data from the **Wingtip Sales Data** dataflow which has been transformed into a star schema that is better suited for data modeling and analysis. If you would like to begin work on this lab without first completing the previous lab, use the Windows Explorer to copy the lab solution file named **Wingtip Sales Model.pbix** which is located in the student folder at **C:\Student\Modules\02_Dataflows\Lab\Solution** into the folder at **C:\Student\Projects**.

Exercise 1: Configure Table Relationships

In this exercise, you will ensure the table relationships between the tables in the data model have been created correctly.

1. Open the **Wingtip Sales Model** project.

    a) Launch Power BI Desktop.

    b) Using the Power BI Desktop **File > Open** command, open **Wingtip Sales Model.pbix** located at the following path.

    ```
    C:\Student\Projects\Wingtip Sales Model.pbix
    ```

   c) When the project opens, click the table icon in the left navigation to enter **Data** view.
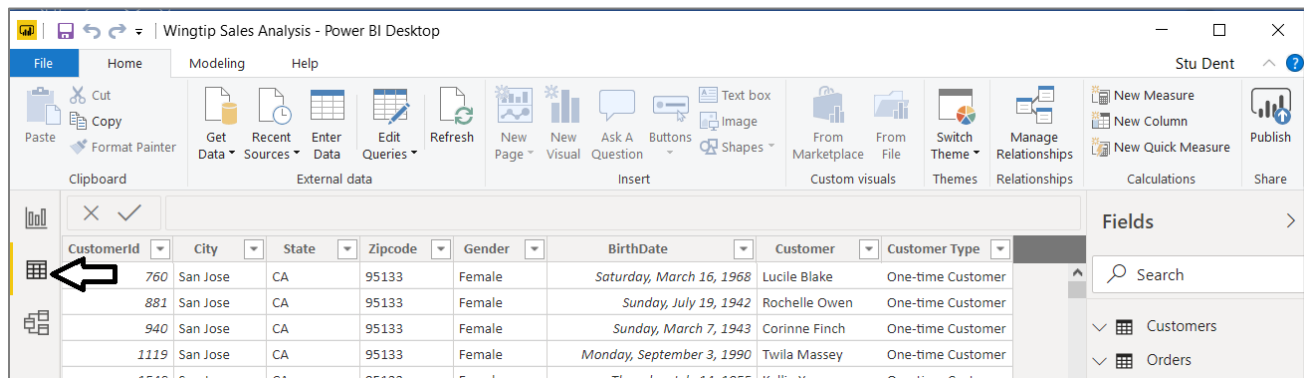
2. Look at the data in each of the four tables in the data model

    a) Click on each of the tables in the **Fields** list one by one to see what the data in each table looks like.
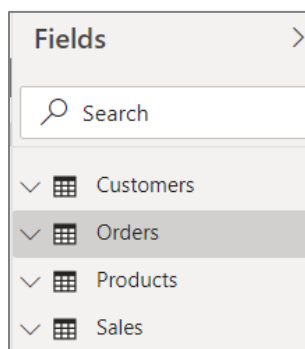
3. Inspect the tables relationships that have been created in the project's data model.

    a) Click the bottom button in the sidebar to navigate to Model view.
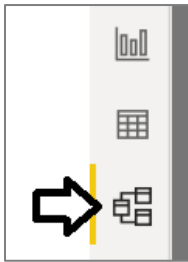
    b) You should see that the four tables in a star schema where **Sales** has a relationship with each of the three other tables.
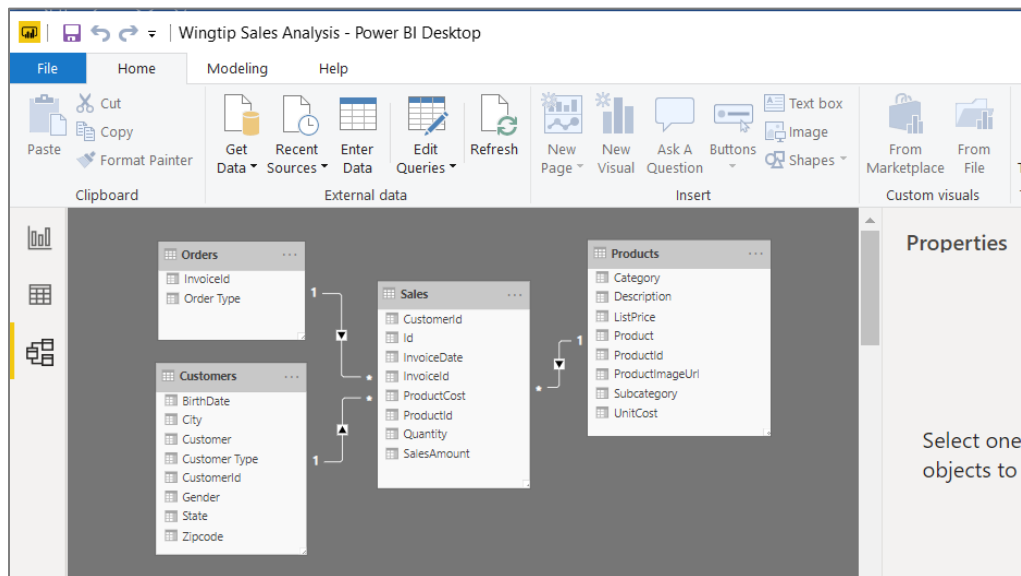
    c) Currently, all three table relationships are shown with a single arrow indicating their **Cross filter direction** is set to **Single**.
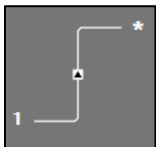

In certain scenarios it might make sense to change the **Cross filter direction** of a table relationship from **Single** to **Both**. Remember that the filter on the one-side table of a one-to-many relationship is automatically propagated to the many-side table. For example, a filter on the **State** column of the **Customers** will be automatically applied to **Sales.** However, the opposite is not true.

The filtering on a table on the many-side of a table relationship will not be automatically propagated to the table on the one-side if the **Cross filter direction** is set to **Single**. For example, you can change the **Cross filter direction** of the relationship between **Sales** and **Products** to **Both** if you want filtering on the **Sales** table to be automatically propagated to the **Products** table. This would allow filtering on the **Customers** table to propagate all the way over to the **Products** table which, in turn, would allow you to filter on customer **State** to see which products have sold in that state.

You should be cautious about setting the **Cross filter direction** of a table relationship to **Both**. Table relationships set to **Both** increase the size of your data model and can also degrade performance in larger data models. The disadvantages of using **Both** really only apply to larger data models such as those with tens of millions or hundreds of millions of rows. You will not really see any differences in performance with smaller data models such as the Wingtip Sales data model you are working with in this lab exercise. This lab will only involve table relationships with the **Cross filter direction** is set to **Single** to encourage the best practice for building larger data models.

4. Inspect the data model from the perspective of a report builder.

   a) Click the report icon in the left navigation to navigate to **Report** view.
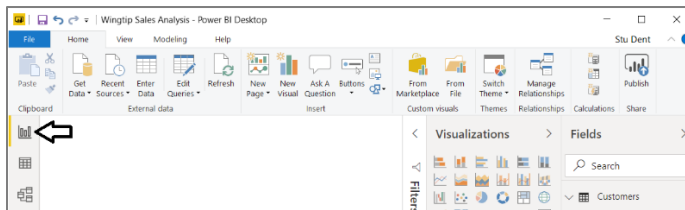
   b) Inside the **Fields** list, use the mouse to expand the fields inside the **Sales** table.

There are several fields in the **Sales** table that will never be used when designing reports such as the ID columns. The data model will be easier for consumers such as report builders to understand if you hide fields which will not be used and add unnecessary clutter.

5. Use **Model** view to update the data model by hiding fields in the **Sales** table that are unnecessary to display in report view.

   a) Navigate to **Model** view in the Power BI Desktop window.

   b) Using the mouse, click on the **CustomerId** column in the **Sales** table to select it.
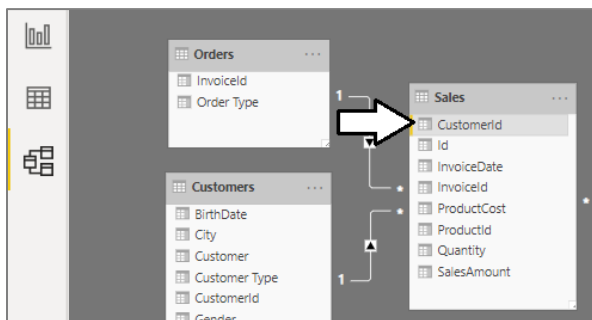
   c) Hold down the **Ctrl** key and use the mouse to additionally select the columns **Id**, **InvoiceId** and **ProductId**.
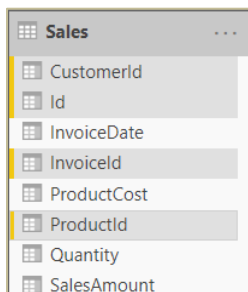
d) Right-click on one of the selected columns and select the **Hide in report view** command.
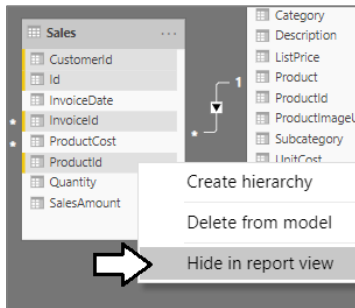
e) Now, navigate back to **Report** view and examine the set of fields displayed in the **Fields** list for the **Sales** table.
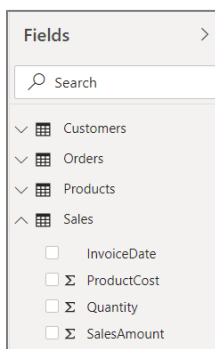

You should be able to see that hiding unnecessary columns from **Report** view makes your data model easier to use. This is especially true in the scenario where you are creating a data model that other less-technical people will be using to build reports & dashboards.

6. Modify the formatting of the **BirthDate** column in the **Customers** table.

   a) In the Power BI Desktop windows, navigate back to **Data** view.

   b) In the **Fields** list on the right, select the **Customers** table to display its rows and columns.

   c) Select the **BirthDate** column.

   d) Modify the **BirthDate** column formatting using the **Format** menu to select a format of **Date Time > 3/14/2001 (M/d/yyyy)**.
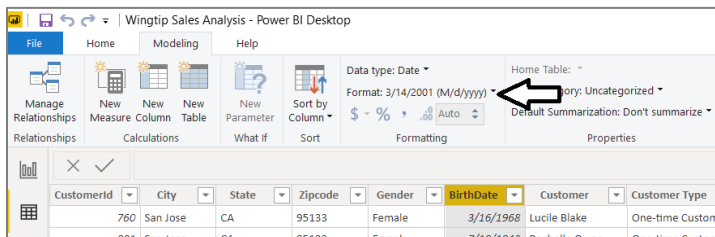
e) The **BirthDate** column should now reflect the change in formatting.
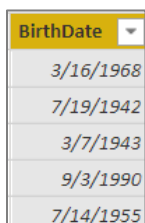
7. Modify the formatting of columns in the **Products** table.

   a) In the **Fields** list on the right, select the **Products** table to display its rows and columns.

   b) Select the **UnitCost** column by clicking on its column header.

   c) Use the **Format** menu button in the ribbon to update the format setting to **Currency > $ English (United States)** and set the number of decimal places to **2** in the spin control located underneath the **Format** dropdown menu.


   d) Changing the format setting of the **ListPrice** column to **$ English (United States)** and set the number of decimal places to **2** so it matches the **UnitCost** column.

8. Modify the formatting of columns in the **Sales** table.

   a) In the **Fields** list on the right, select the **Sales** table to display its rows and columns.

   b) Select the **Quantity** column by clicking on its column header.

   c) Modify the **Quantity** column by clicking to select the comma button on the ribbon to add a comma separator.

   d) Select the **SalesAmount** column by clicking on its column header.

   e) Modify the formatting of the **SalesAmount** column to **Currency > $ English (United States)** and set the number of decimal places to **2** in the spin control located underneath the **Format** dropdown menu
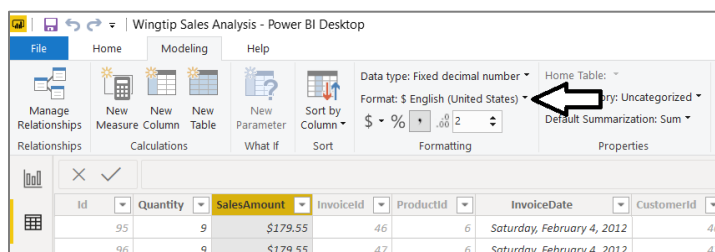
f) Select the **InvoiceDate** column by clicking on its column header.

g) Modify the formatting of the **InvoiceDate** to of **Date Time > 3/14/2001 (M/d/yyyy)**.
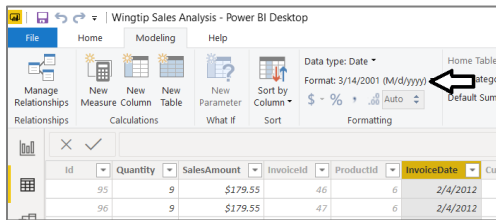
h) Select the **ProductCost** column by clicking on its column header.

i) Modify the formatting of the **ProductCost** column to **Currency> $ English (United States)** and set the number of decimal places to **2** in the spin control located underneath the **Format** dropdown menu.
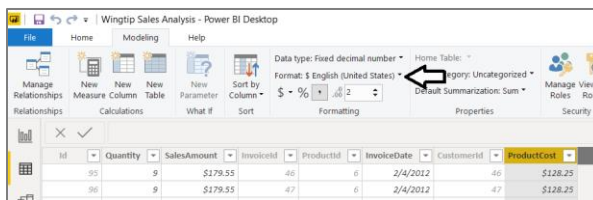
9. Save the work you have done by clicking the **Save** button in the upper left corner of the Power BI Desktop window.

Exercise 2: Create Calculated Columns using DAX

In this exercise you will create several calculated columns which will require you to write and test DAX expressions. After creating calculated columns, you will use them to enhance the report in the current project.

1. Create a calculated column in the **Customers** table named **Age** to indicate the age of the customer.

   a) Navigate to **Data** view.

   b) Select the **Customers** table in the **Fields** list.

   c) Create a new calculated column by clicking the **New Column** button in the ribbon.

   d) Enter the following DAX expression into the formula bar to create the calculated column named **Age**.

   ```
   Age = Floor( (TODAY()-Customers[BirthDate])/365, 1)
   ```

   e) Press the **ENTER** key to add the calculated column. You should be able to see a whole number for the age of each customer.
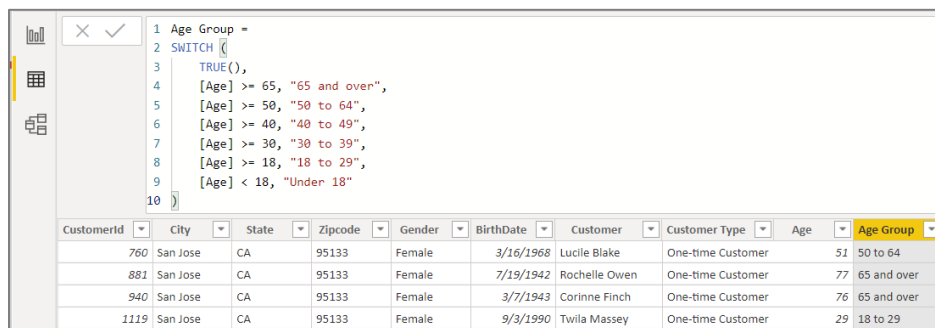

   f) Use the **Default Summarization** dropdown menu to change the **Default Summarization** setting for **Age** to **Average**.
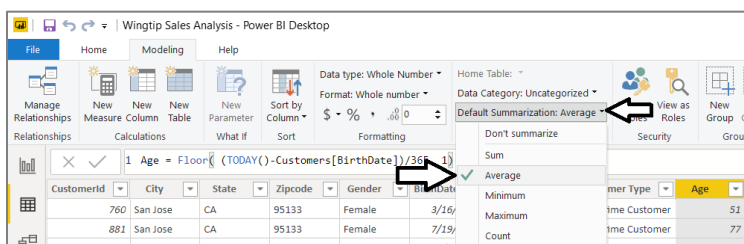
2. Add a calculated column to the **Customers** table named **Age Group** to break customers up into age-based sets.

    a) Create a new calculated column in the **Customers** table by clicking the **New Column** button in the ribbon.

    b) Enter to following DAX expression into the formula bar to create the calculated column named **Age Group**.

```
Age Group =
SWITCH (
    TRUE(),
    [Age] >= 65, "65 and over",
    [Age] >= 50, "50 to 64",
    [Age] >= 40, "40 to 49",
    [Age] >= 30, "30 to 39",
    [Age] >= 18, "18 to 29",
    [Age] < 18, "Under 18"
)
```

    c) After creating the calculated column, you should be able to verify that the **Age Group** column calculates a value for each customer row which places that customer in a bucket for a particular age group.

It's a best practice to format DAX expressions with line breaks and indenting to make them easier to read and maintain. Unfortunately, Power BI Desktop offers no built-in features to format DAX expression as you write them. However, there is a free DAX formatting tool for free on the Internet at https://www.daxformatter.com/. This tool allows you to paste your DAX expressions into a web page and it generates the properly-formatted output which you can copy-and-paste back into the DAX expression editor in Power BI Desktop.

    d) Save the work you have done by clicking the **Save** button in the upper left corner of the Power BI Desktop window.

## Exercise 3: Create a Dynamic Lookup Table using DAX

In this exercise, you will extend the data model by adding a new lookup table named **SalesRegions** which assigns each state to a geographic sales region. This will involve creating a new table using a DAX expression which calculates the start date and the end date dynamically. Once you have created the **SalesRegions** table, you must then create a table relationship between the **SalesRegions** table and the **Customers** table to integrate the new table into the data model. At the end of this exercise, you will also create two new calculated columns in the **Customers** table to pull in related data from the **SalesRegions** table.

1. Create the **SalesRegions** table using a DAX expression.

    a) Navigate to Data view in the **Wingtip Sales Model.pbix** project.

    b) Click the **New Table** button to create a new table using DAX.

c) You should now be able to add the DAX expression to create the new table.
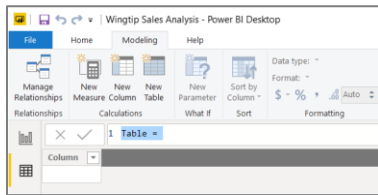
Instead of having you write one heck-of-a-long DAX expression, the lab exercise will have you copy and paste the required DAX expression from a text file in the **Students** folder.

d) Locate the file named **CreateSalesRegionsTable.txt** at the following path and open it with Windows Notepad.

```
C:\Student\Modules\03_DataModeling\Lab\DAX\CreateSalesRegionsTable.txt
```

e) Take a moment to inspect the DAX expression inside **CreateSalesRegionsTable.txt**.

f) Select the entire contents of **CreateSalesRegionsTable.txt** and then copy it into the Windows clipboard.

g) Return to Power BI Desktop and paste in the DAX expression for the new table.

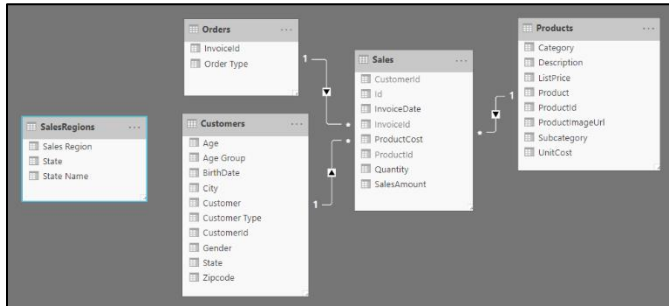h) Press the **ENTER** key to create the new table named **SalesRegions**.

2. Create a table relationship between the **SalesRegions** table and the **Customers** table.

a) Navigate to **Relationship** view.

b) You should be able to see the new **SalesRegions** table on the right.

c) Rearrange layout of tables in **Model** view by moving **SalesRegions** to the left and all the other tables to the right.

d) Create a new table relationship between **SalesRegions** and **Customers** by clicking and dragging the **State** column from the **SalesRegions** table and dropping it on top of the **State** column in the **Customers** table.

e) You should be able to confirm that a new table relationship has been created between **SalesRegions** and **Customers**.
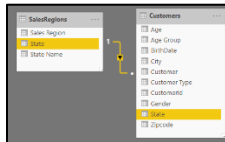
3. Add a calculated column to the **Customers** table named **Sales Region** to display the sales region for each state.

a) Navigate to data view.

b) Select the **Customers** table in the **Fields** list.

c) Create a new calculated column by clicking the **New Column** button in the ribbon.

Since a relationship exists between **SalesRegions** and **Customers**, you can use the **RELATED** function provided by DAX to create calculated columns in the **Customers** table that pull in data from the **SalesRegions** table.

d) Enter to following DAX expression into the formula bar to create the calculated column named **Sales Region**.

```
Sales Region = RELATED(SalesRegions[Sales Region])
```

e) Press the **ENTER** key to add the calculated column to the table. You should be able to see a value in the **Sales Region** column for each row in the **Customers** table.

4. Add a calculated column to the **Customers** table named **State Name** to display the full state name.

a) Create a new calculated column in the **Customers** table by clicking the **New Column** button in the ribbon.

b) Enter to following DAX expression into the formula bar to create the calculated column named **State Name**.

```
State Name = RELATED(SalesRegions[State Name])
```

c) Press the **ENTER** key to add the calculated column to the table. You should be able to see a value in the **Sales Region** column for each row in the **Customers** table.

> Now that you have pulled all the important data from the **SalesRegions** table into the **Customers** table, there is no need to display the **SalesRegions** table in report view. In the next step, you will hide the **SaleRegions** table to simplify view of the data model that is shown in report view.

5. Hide the **SalesRegions** table from report view.

    a) Navigate to relationship view.

    b) Right-click on the the **SalesRegions** table
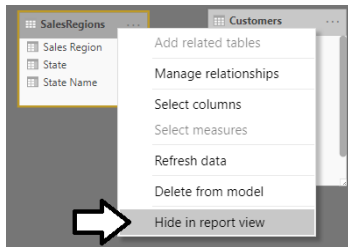
    c) Navigate to **Report** view and verify that the **SalesRegions** table is not displayed as one of the tables in the **Fields** list.
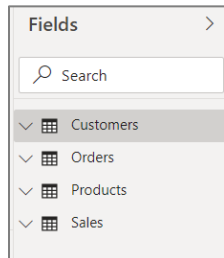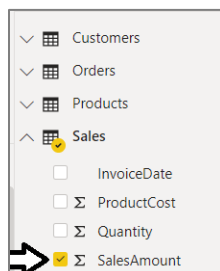

6. See the effect of your formatting by adding a visual to a report.

   a) Navigate to report view. There should be an empty report for the project with a single page named **Page 1**.
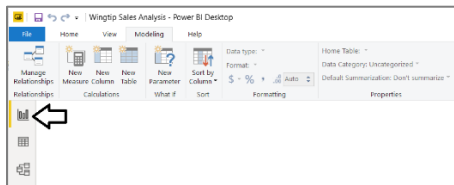
   b) Change the name of the page in the report from **Page 1** to **Sales by State**.

    c) Create a new visual in the report by selecting the checkbox for the **SalesAmount** column in the **Fields** list.

d) When you select the **SalesAmount** column, Power BI Desktop will automatically add a new visual to the report based on the visualization type of **Clustered Column Chart**.
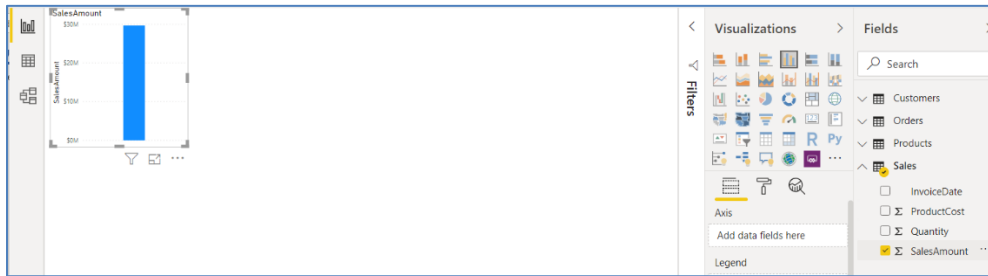

e) Drag and drop the **State** field from the **Fields** list into the **Axis** well in the **Fields** pane.
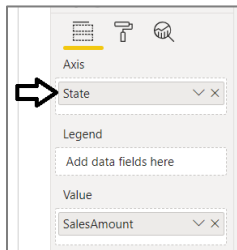
f) Use your mouse to resize the visual to be as wide as the hosting report page as shown in the following screenshot.
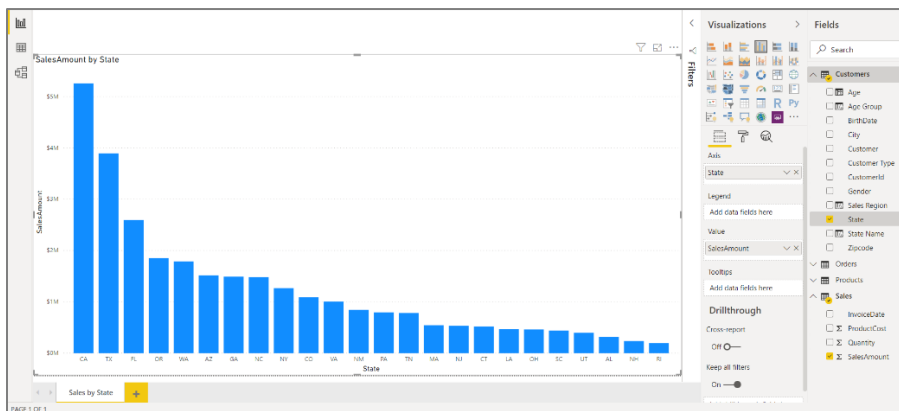
7. Add a slicer visual to the report to filter customer state by sales region.

a) Navigate to **Report** view if you are not already there.

b) Make sure that no visuals are selected on the page so that you can create a new visual.

c) Select the checkbox for the **Sales Region** column in the **Fields** list to create a new visual.

d) Click the **Slicer** button in the **Visualizations** list to change the visual type to a slicer.
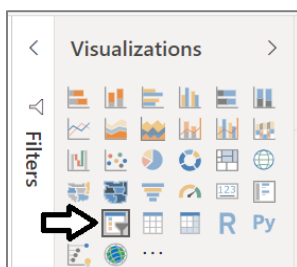
e) Using the mouse, reposition the slicer visual so it appears in the upper, left-hand corner of the page.
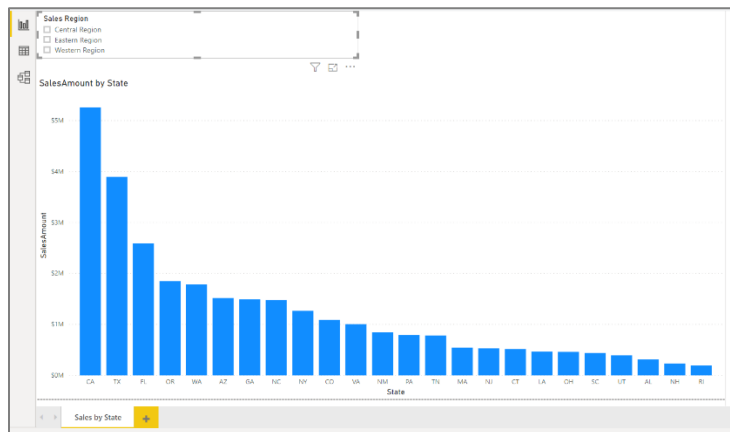
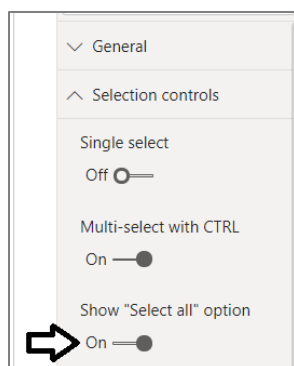You will now change the alignment of the slicer visual from a vertical layout to a horizontal layout.
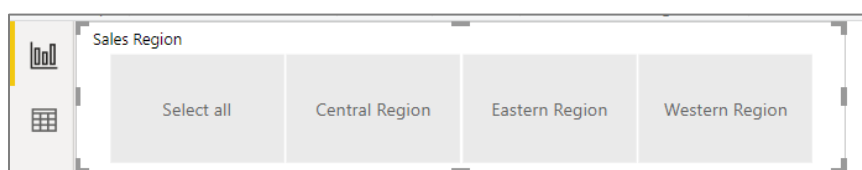
f) With the slicer visual selected, navigate to the **General** section of the **Format** pane.

g) Change the **Orientation** property to a value of **Horizontal**.
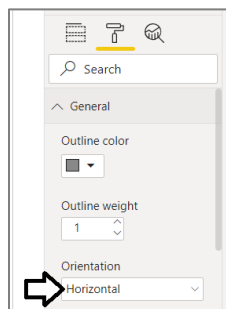
h) Enable the **Show "Select all" option** property in the **Selection controls** section by setting its value to **On**.

i) The slicer visual should now appear with a horizontal layout with an additional **Select All** node.

j) Click on the different sales regions in the slicer visual to observe its filtering effect. If you click on **Western Region**, the column chart should only show states assigned to a **Sales Region** of **Western Region**.
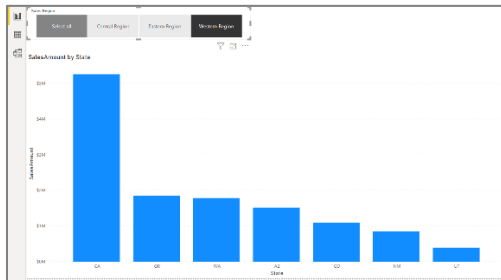
8. Save the work you have done by clicking the **Save** button in the upper left corner of the Power BI Desktop window.

> Oh no! There's a problem with the default sort order of the **Sales Region** column. Currently, column values are sorted alphabetically so **Central Region** is first, **Easter Region** is second and **Western Region** is third. However, your manager has asked you to implement a customized sort order so that **Western Region** appears first, **Central Region** is second and **Eastern Region** is third. To solve this problem, you will create a new table with DAX named **SalesRegionsSort** to implement a custom sort order.

9. Create the **SalesRegionsSort** table using a DAX expression.

a) Navigate to **Data** view in the **Wingtip Sales Model.pbix** project.

b) Click the **New Table** button to create a new table using DAX.
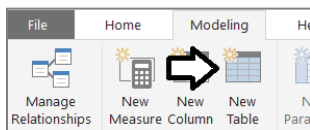
c) You should now be in editing mode where you can add the DAX expression to create the new table.

d) Locate the file named **CreateSalesRegionsSortTable.txt** at the following path and open it with Windows Notepad.

```
C:\Student\Modules\03_DataModeling\Lab\DAX\CreateSalesRegionsSortTable.txt
```

e) Take a moment to inspect the DAX expression inside **CreateSalesRegionsSortTable.txt**.

```
SalesRegionsSort =
DATATABLE (
    "Sales Region", STRING,
    "SalesRegionSort", INTEGER,
    {
        { "Western Region", 1 },
        { "Central Region", 2 },
        { "Eastern Region", 3 }
    }
)
```

f) Select the entire contents of **CreateSalesRegionsSortTable.txt** and then copy it into the Windows clipboard.

g) Press the ENTER key to submit your DAX expression and to create the new table.

10. Create a relationship between the **SalesRegionsSort** table and the **Customers** table.

    a) Navigate back to **Model** view and verify you can see the new table named **SalesRegionsSort**.

    b) Using the mouse, move the **SalesRegionsSort** table underneath the **SalesRegions** table to the left of the **Customers** table.
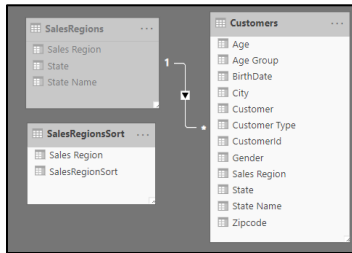
    c) Drag the **Sales Region** column from **SalesRegionsSort** and drop it on the **Sales Region** column in **Customers**.
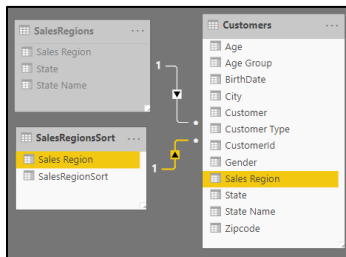
You should be able to verify that a one-to-many relationship has been created between **SalesRegionsSort** and **Customers**.

11. Add the **SalesRegionSort** column to the **Customers** table as a calculated column.

    a) Navigate to Data view and select the **Customers** table in the **Fields** list.

    b) Click the **New Column** button in the ribbon to create a new calculate column.

    c) Type in the following DAX expression to create a new calculated column named **SalesRegionSort**.

```
SalesRegionSort = RELATED(SalesRegionsSort[SalesRegionSort])
```


12. Configure a custom sort column for the **Sales Region** field.

    a) Select the **Sales Region** field by clicking its column header.

    b) Drop by the **Sort By Column** menu in the ribbon and select the **SalesRegionSort** column.

13. Simply the **Report** view of your data model by hiding the implementation details of your custom sort order.

    a) Navigate to **Model** view.

    b) Hide the **SalesRegionsSort** table by right-clicking it and selecting **Hide in Report View**.

    c) Hide the **SalesRegionSort** column in the **Customer** table by right-clicking it and selecting **Hide in Report View**.

14. Verify your custom sort order is working properly.

    a) Return to Report View and examine the slicer which shows the sales regions.

    b) Verify that the sort order now has **Western Sales** first followed by **Central Region** and then **Eastern Region**.

Exercise 4: Create Measures using DAX

In this exercise you will create four measures named **Sales Revenue, Units Sold, Product Cost** and **Profit** that will perform sum aggregations on the **Sales** table. You will also create a measure named **Customer Count** that performs a distinct count aggregation on the **Customers** table. As you will see, creating measures to use in your report visuals will give you much greater control over the column names, formatting and aggregations that are displayed in your reports.
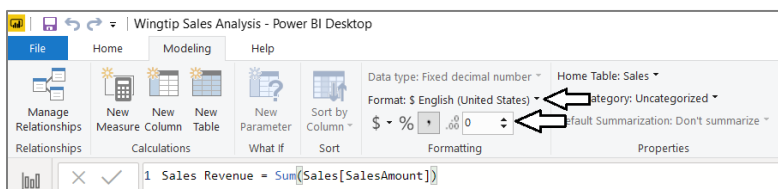
1. Create a measure in the **Sales** table named **Sales Revenue** to perform a sum aggregation on the **SalesAmount** column.

    a) Navigate to **Data** view and select the **Sales** table from the **Fields** list.

    b) Click the **New Measure** button in the **Modeling** tab of the ribbon to add a new measure to the **Sales** table.
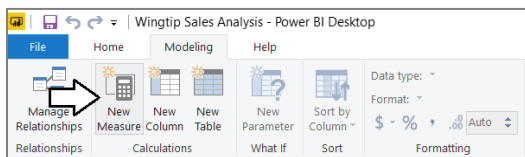
    c) Enter the following DAX expression into the formula bar to create the measure named **Sales Revenue**.
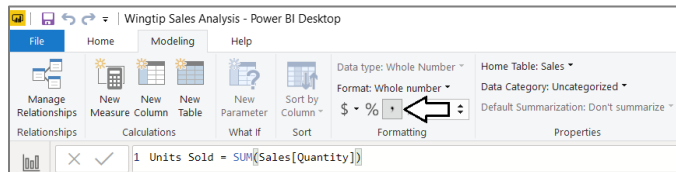
```
Sales Revenue = Sum(Sales[SalesAmount])
```

    d) Press the **ENTER** key to add the measure to data model.

    e) Modify the formatting by dropping down the **Format** menu on the ribbon and selecting **Currency > $ English (United States)**.

    f) Use the spinner control below the **Format** menu to set the number of decimal places shown to zero.

2. Create a measure in the **Sales** table named **Units Sold** to perform a sum aggregation on the **Quantity** column.

   a) Select the **Sales** table in the **Fields** list and then click the **New Measure** button in the Modeling tab of the ribbon.

   b) Enter to following DAX expression into the formula bar to create the measure named **Units Sold**.

   ```
   Units Sold = SUM(Sales[Quantity])
   ```
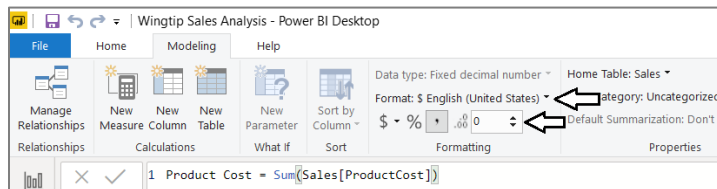
   c) Press the **ENTER** key to add the measure to data model.

   d) Modify the formatting by clicking and selecting the **Comma (,)** button on the ribbon to add a comma separator.

3. Create a measure in the **Sales** table named **Product Cost** to perform a sum aggregation on the **ProductCost** column.

   a) Create a new measure by clicking the **New Measure** button in the ribbon.

   b) Enter to following DAX expression into the formula bar to create the measure named **Product Cost**.
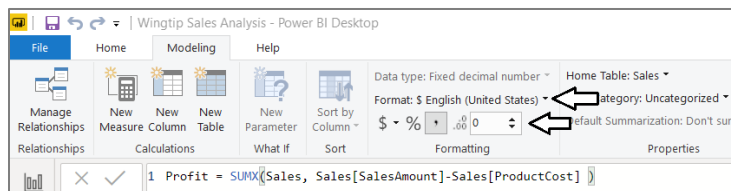
   ```
   Product Cost = Sum(Sales[ProductCost])
   ```

   c) Press the **ENTER** key to add the measure to data model.

   d) Modify the formatting by dropping down the **Format** menu on the ribbon and selecting **Currency > English (United States)**.

   e) Use the spinner control below the **Format** menu to set the number of decimal places shown to zero.

4. Create a new measure in the **Sales** table named **Profit** to sum the difference between **SalesAmount** and **ProductCost**.

   a) Select the **Sales** table in the **Fields** list and then click the **New Measure** button in the ribbon.

   b) Enter to following DAX expression into the formula bar to create a new measure named **Profit**.

   ```
   Profit = SUMX(Sales, Sales[SalesAmount]-Sales[ProductCost])
   ```

   c) Press the **ENTER** key to add the measure to data model.

   d) Modify the formatting by dropping down the **Format** menu on the ribbon and selecting **Currency > English (United States)**. Also use the spinner control below the **Format** menu to set the number of decimal places shown to zero.
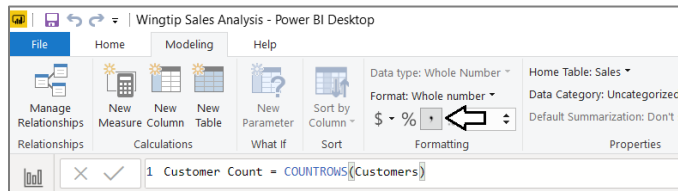

5. Create a measure in the **Sales** table named **Customer Count** to count the number of rows in the **Customers** table.

   a) Make sure the **Sales** table is selected and then click the **New Measure** button in the ribbon.

   b) Enter the following DAX expression into the formula bar to create the measure named **Customer Count**.

   ```
   Customer Count = COUNTROWS(Customers)
   ```

c) Press the **ENTER** key to add the measure to data model.

d) Modify the formatting by clicking and selecting the Comma button on the ribbon to add a comma separator.



6. Create a new measure named **Order Count** to determine the number of orders.

a) Navigate to data view.

b) Select the **Sales** table from the **Fields** list.

c) Create a new measure by clicking the **New Measure** button in the ribbon.

d) Enter to following DAX expression into the formula bar to create the measure named **Order Count**.

```
Order Count = CALCULATE(
    COUNTROWS(Orders),
    CROSSFILTER(Orders[InvoiceId], Sales[InvoiceId], Both)
)
```

e) Press the **ENTER** key to add the measure to the data model.

f) Ensure the formatting for this measure is set to **Whole Number** as shown in the following screenshot. Also check the comma button to format values over 1000 with a comma separator.
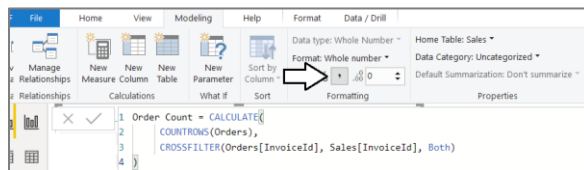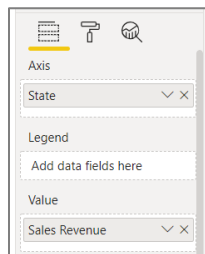


7. Update the Column chart visual to use the new measures you've just created.

a) Navigate to **Report** view.

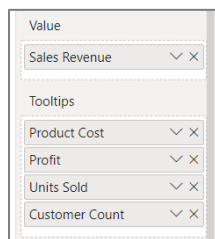b) Select the column chart visual and then select the **Fields** pane.
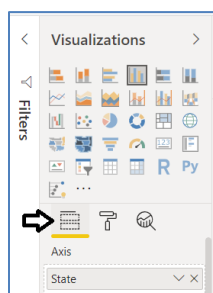


c) Currently, the **Value** well should contain the **SalesAmount** column.
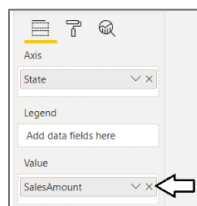
    d) Remove the **SalesAmount** column from the **Values** well and replace it with the **Sales Revenue** measure.



    e) Locate the **Tooltips** well and add the measures named **Product Cost**, **Profit**, **Units Sold** and **Customer Count**.



    f) Navigate to the Format Pane for the Column Chart visual and turn off the Title property to hide the visual title.

    g) Hover your mouse over bars in the bar chart and you should see all the measures displayed in the tool tip.
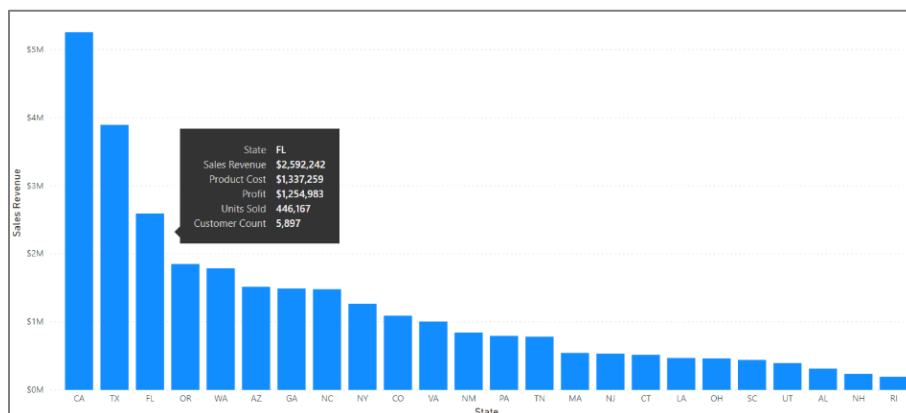


You will complete your work in this exercise by cleaning up the data model by hiding fields in report view.

8. Simplify your data model by hiding fields that will not be used when building reports.
    a) Navigate to **Model** view so you can see all the tables in the project's data model.
    b) Using the mouse, resize each table in **Model** view so that it properly displays all its fields without the need for a scrollbar.
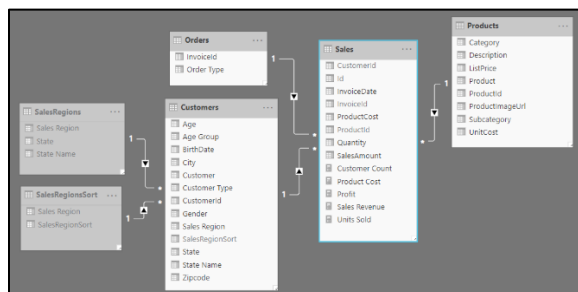
c) Use the **Hide in Report View** command to hide the following columns.

   i) The **InvoiceId** column in the **Orders** table.

   ii) The **CustomerId** column and the **BirthDate** column from the **Customers** table.

   iii) The **ProductId** column, the **UnitCost** column and the **ListPrice** column from the **Products** table.

> When you hide all the fields in a table except for its measures, Power BI Desktop treats this table as a **fact table**. The main effect this has in Power BI Desktop is that the table will be displayed at the top of the **Fields** list when in **Report** view. However, Power BI Desktop does not refresh the view until you close and reopen the **Fields** list. In the next step, you will close and reopen the **Fields** list to see the effects of creating a tables which only displays measures in Report View.

a) Configure the **Sales** table as a fact table by hiding all columns and only displaying measures.
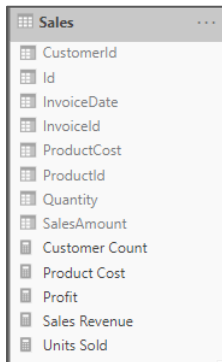


> You should be able to verify that all fields in the **Sales** table are hidden except for the 5 measures. Now it's time to see what effect this has to the **Fields** list when you are in **Report** view.

9. Close and reopen the **Fields** list to see the effects of a measure-only table.

   a) Return to **Report** view.

   b) Locate the arrow menu in the top right corner of the **Fields** list and click it twice to close and reopen the **Fields** list.
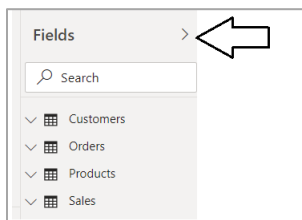


   c) Once the **Fields** list has been refreshed, you should see that the **Sales** table has been moved to the top and has been given a calculator icon to distinguish it from the other visible tables in the data model which are acting as dimension tables.
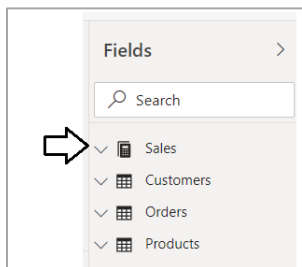


10. Save the current project by clicking the **Save** button in the ribbon.

> While hiding fields and tables is not complicated, it's a valuable technique to simplify a data model so it's easier for others to use.

## Exercise 5: Configure Geolocation Columns using Data Categories

In this exercise, you will configure geolocation metadata to fields in the **Customers** table including **State**, **City** and **Zipcode**. After that, you will create a new report page with a map visual to visualize how customer sales data is spread out across various geographic regions within the United States.

1. Configure geolocation metadata for the **State** field in the **Customers** table.

   a) Return to **Data** View.

   b) From the **Fields** list on the right, select the **State** field from the **Customers** table.

   c) Drop down the **Data Category** menu from the ribbon and select **State or Province**.
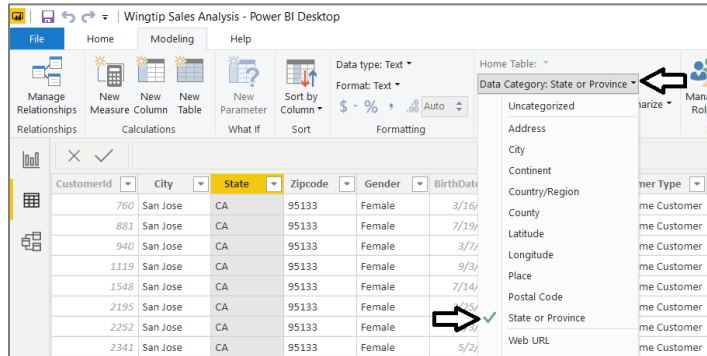


2. Modify the **City** column to contain the state as well as the city to remove ambiguity when determining geographic locations.

   a) Select the **Customers** table in the **Fields** list.

   b) Right-click on the column header for the **City** column and select the **Rename** command.
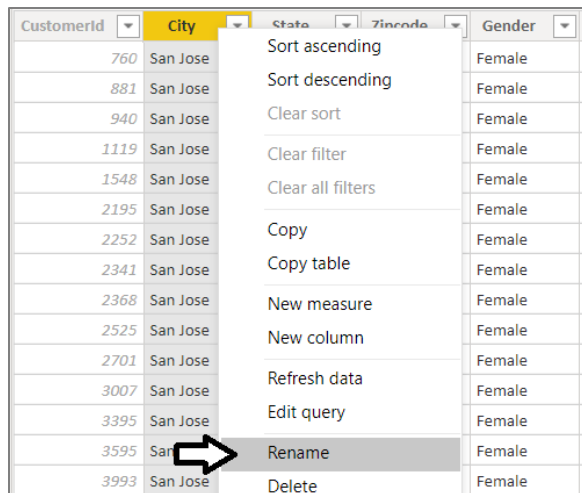


   c) Rename the column to **City Name**.

d) Navigate to the **Modeling** tab in the ribbon and make sure the **Customers** table is still selected.

e) Click the **New Column** tab to create a new calculated column.

f) Add in the following DAX expression to create a new column named **City**.

```
City = [City Name] & ", " & [State]
```

g) The **City** column should display the city and the state which will remove ambiguity when used as a geolocation field.


Certain aspects of working with Power BI are not as intuitive as they could be. For example, you need to configure the **Data Category** for the **City** column which contains values such as **Tampa, FL** and **Austin, TX**. The obvious choice of setting the **Data Category** to **City** does not work as expected. Instead, you must set the **Data Category** setting to **Place** for the visual mapping to work correctly.

3. Configure geolocation metadata for the **City** field in the **Customers** table.

a) Return to Data View.

b) From the **Fields** list on the right, select the **City** field from the **Customers** table.

c) Drop down the **Data Category** menu from the ribbon and select **Place**.
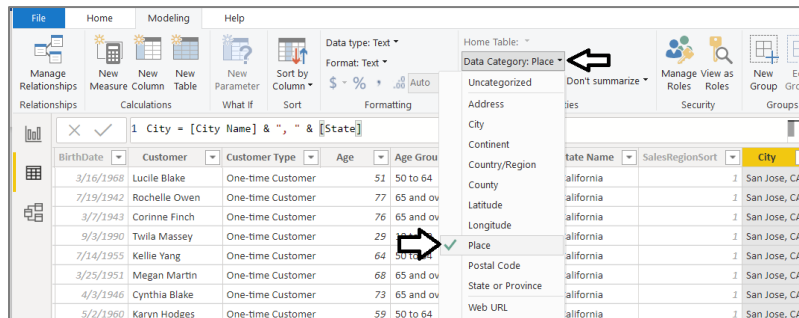
4. Configure geolocation metadata for the **Zipcode** field in the **Customers** table.

a) Return to **Data** view.

b) From the **Fields** list on the right, select the **Zipcode** field from the **Customers** table.

c) Drop down the **Data Category** menu from the ribbon and select **Postal Code**.
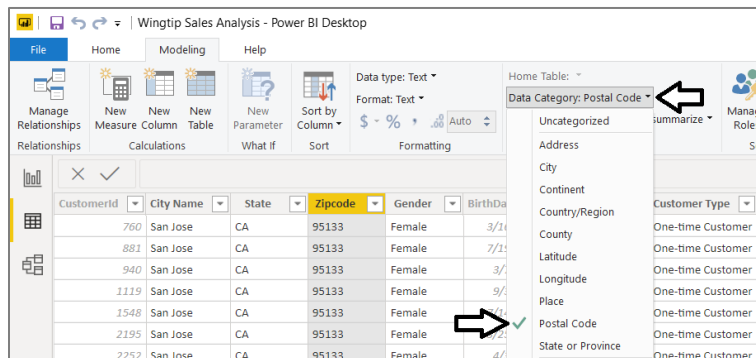
> Now that you have configured fields in the **Customers** table with geolocation metadata, it's time to put this metadata to use by creating a new report page with a map visual to visualize how sales revenue is distributed over geographic regions.

5. Change to the clustered bar chart visual into a **Map** visual.

   a) Return to **Report** view.

   b) Select he column chart visual.

   c) Click the **Map** icon in the **Visualizations** list to change the visual into a **Map** visual.
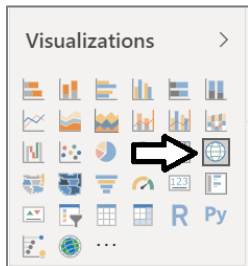
   d) The **Map** visual should display as a bubble map of the United States as shown in the following screenshot.
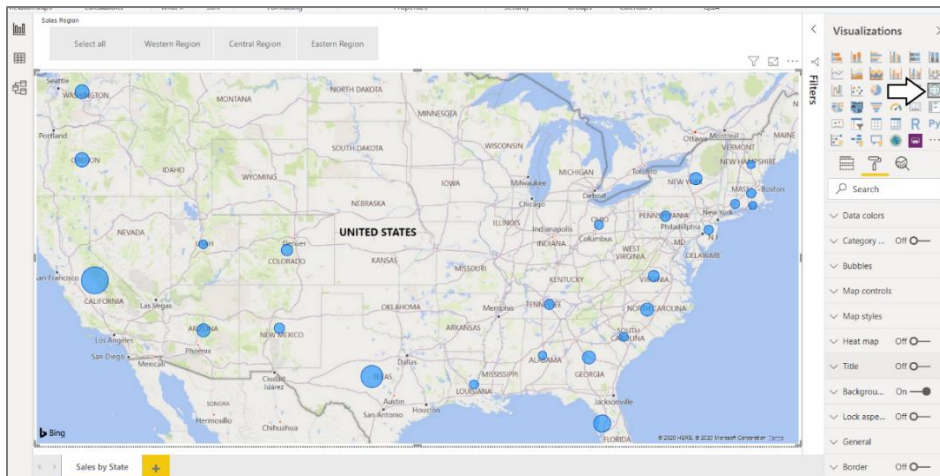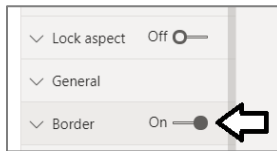

   e) With the **Map** visual selected, navigate to the **Data Colors** section in the **Format** properties pane and change the default color to a dark red so it stands out more clearly on the **Map** visual.
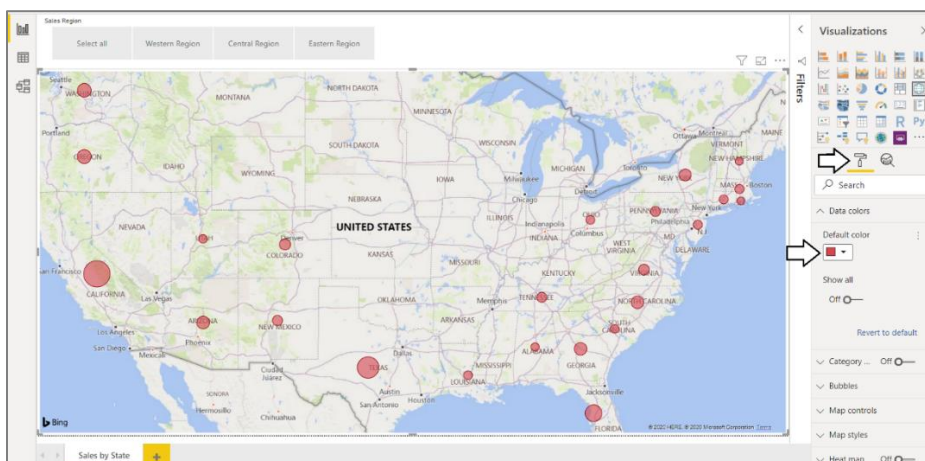
f) Move down inside the **Format** pane and enable the **Border** setting.

g) Test out the **Tooltips** setting by hovering the mouse over a red bubble for a specific state. You should see that the **Tooltips** setting displays **State**, **Sales Revenue**, **Product Cost**, **Profit**, **Units Sold** and **Customer Count**.
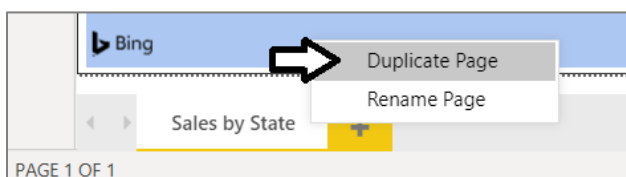

h) Test out the slicer. You should be able to select a single sales region and see the map update to reflect the new filtering.
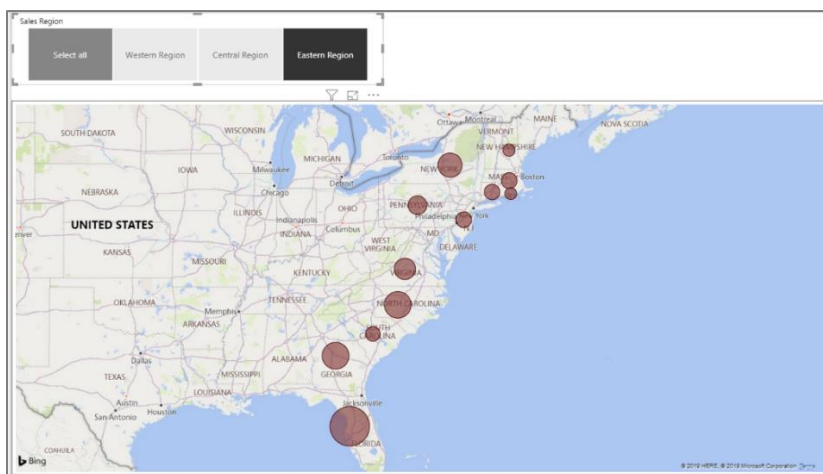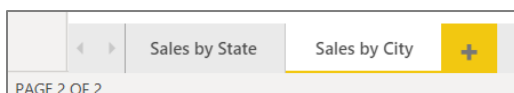

6. Create a new report page to show sales revenue by city.

   a) Create a new page by right-clicking the **Sales by State** page tab and then selecting the **Duplicate Page** command.


   b) Once the new page has been created, rename it to **Sales by City**.

7. Update the **Map** visual on the **Sales by Cities** page to display bubbles for cities instead of states.
   a) Select the **Map visual** and then locate the **Location** well in the **Fields** pane.
   b) Remove the **State** column from the **Location** well and replace it with the **City** column.
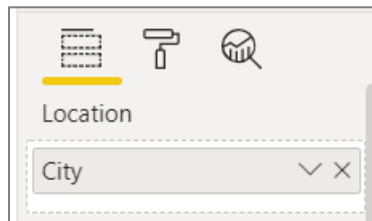
   c) The **Map** visual should now display bubbles for cities instead of states.
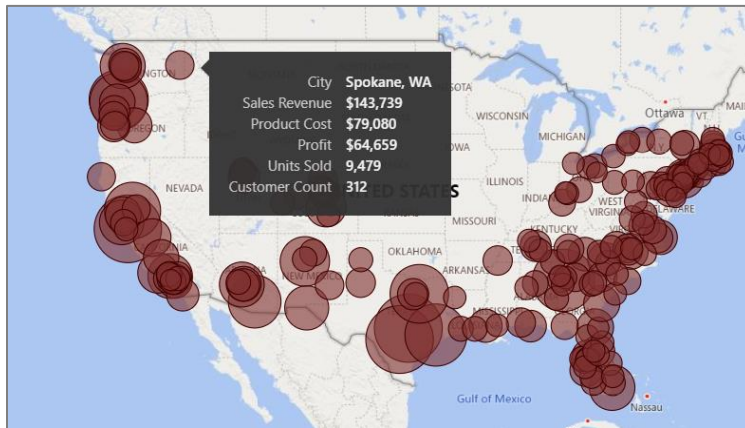
8. Update the **Slicer** visual on the **Sales by Cities** page to filter by the **State** column instead of the **Sales Region** column.
   a) Select the **Slicer** visual that is currently configured to filter by **Sales Region**.
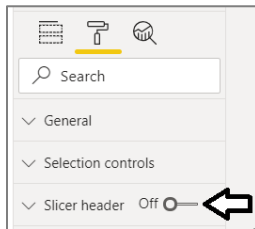   b) In the **Fields** pane. remove **Sales Region** from the **Field** well and replace it with **State**.
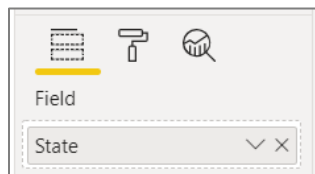
   c) In the **Selection controls** section of the **Format** pane for the slicer visual, set **Show "Select all" option** to **Off**.
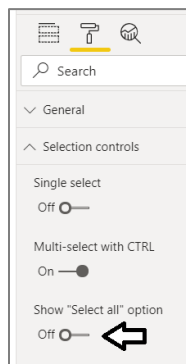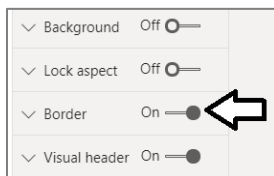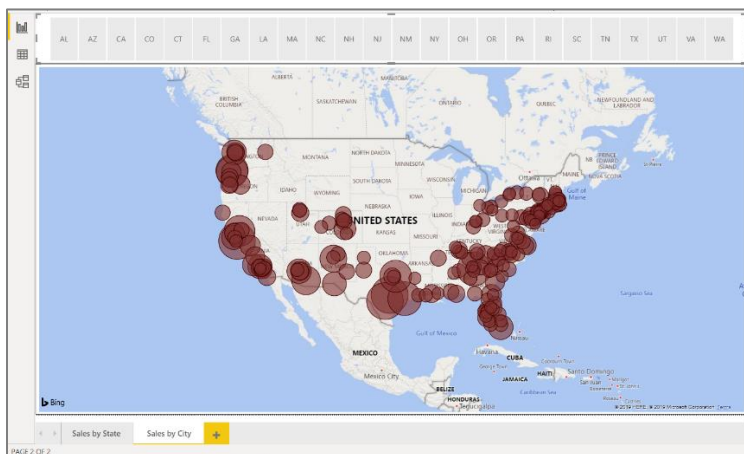
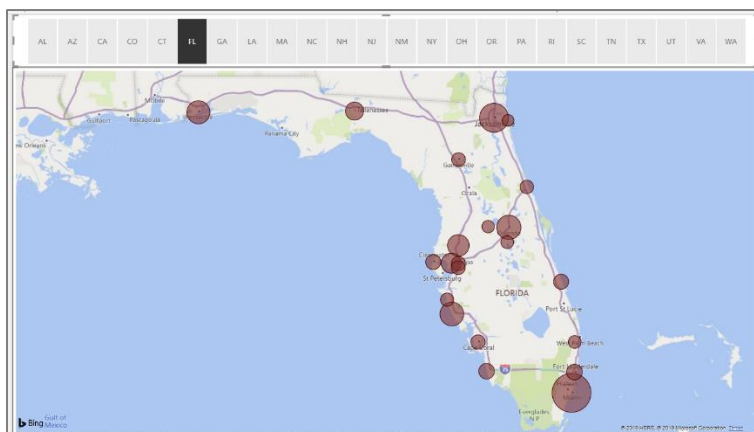d) Set the **Slicer header** property for the slicer visual to **Off**.

e) Set the **Border** property for the slicer visual to **On**.

f) Reposition the slicer visual and the map visual to match the layout shown in the following screenshot.

g) Use the slicer to select a single state at a time to test your work.

Congratulations for making great progress. You have now completed this lab. Note that you will be continuing to work on the same Power BI Desktop project named **Wingtip Sales Model.pbix** in the next lab exercise.