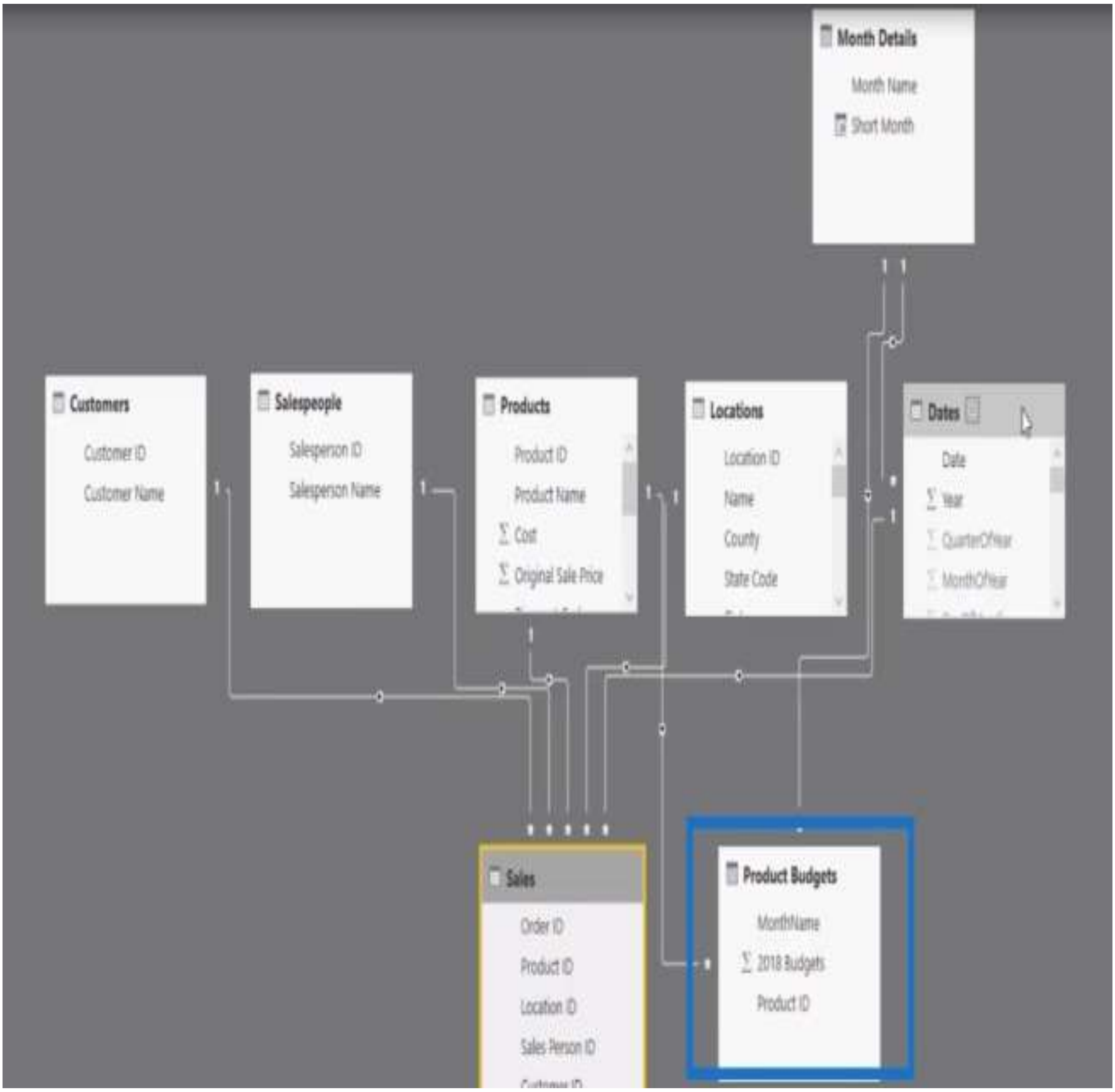
a. Explain Table manipulation functions – CROSSJOIN, EXCEPT, INTERSECT with examples.

CROSSJOIN	This function returns a table that contains the cartesian product of all rows from all tables in the argument	CROSSJOIN (<table>,<table>[,<table>]….)

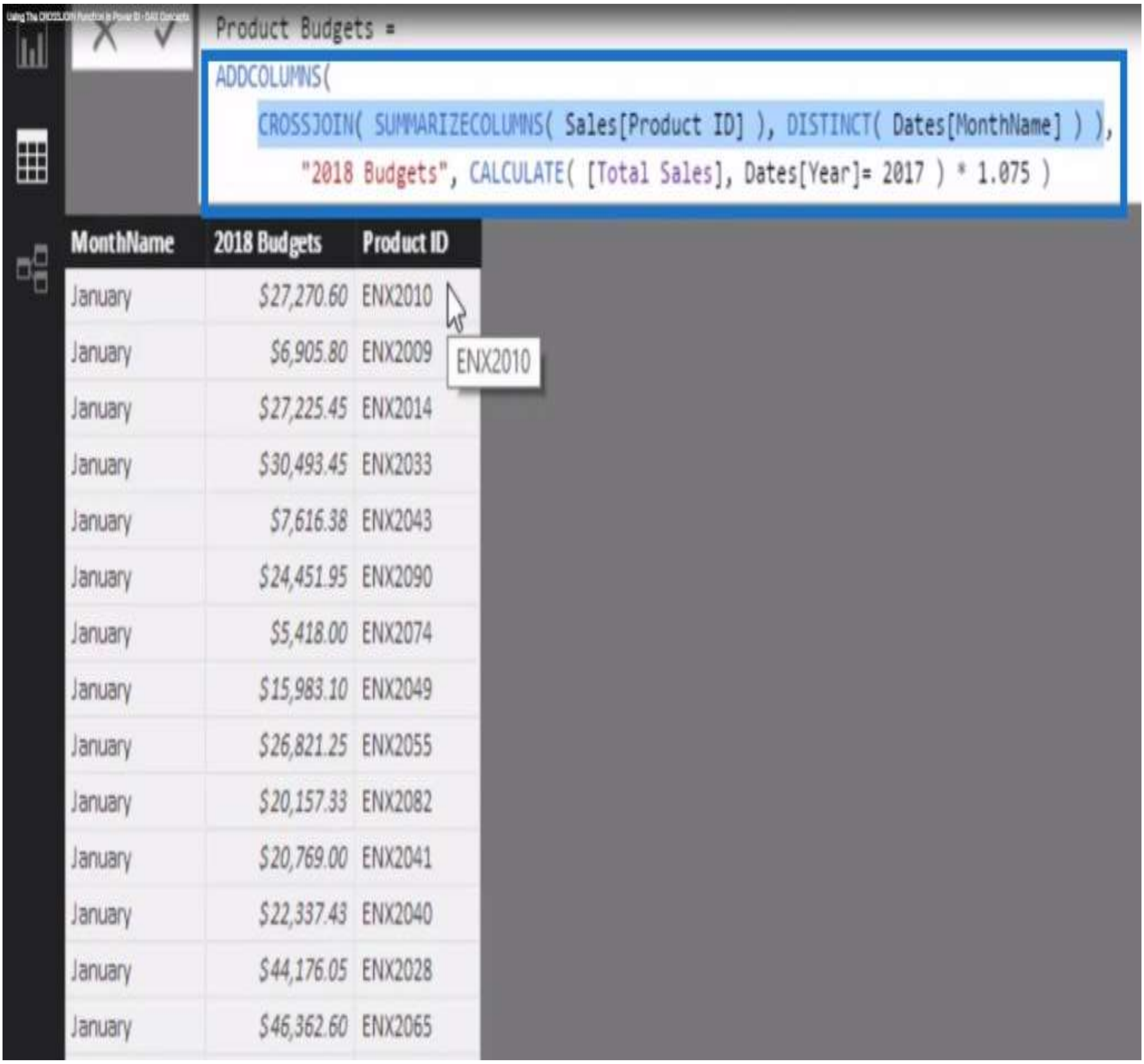
This is where CROSSJOIN comes in. It allows you to mesh together two columns or a range of columns from different tables and make them as one. Using CROSSJOIN, I managed to create the Product Budgets table.



There are probably other ways to do this, but this is the most effective way to recreate a table out of nothing.

To achieve this table, I used other table functions as well, which I'll briefly run through but I will focus on CROSSJOIN for now.

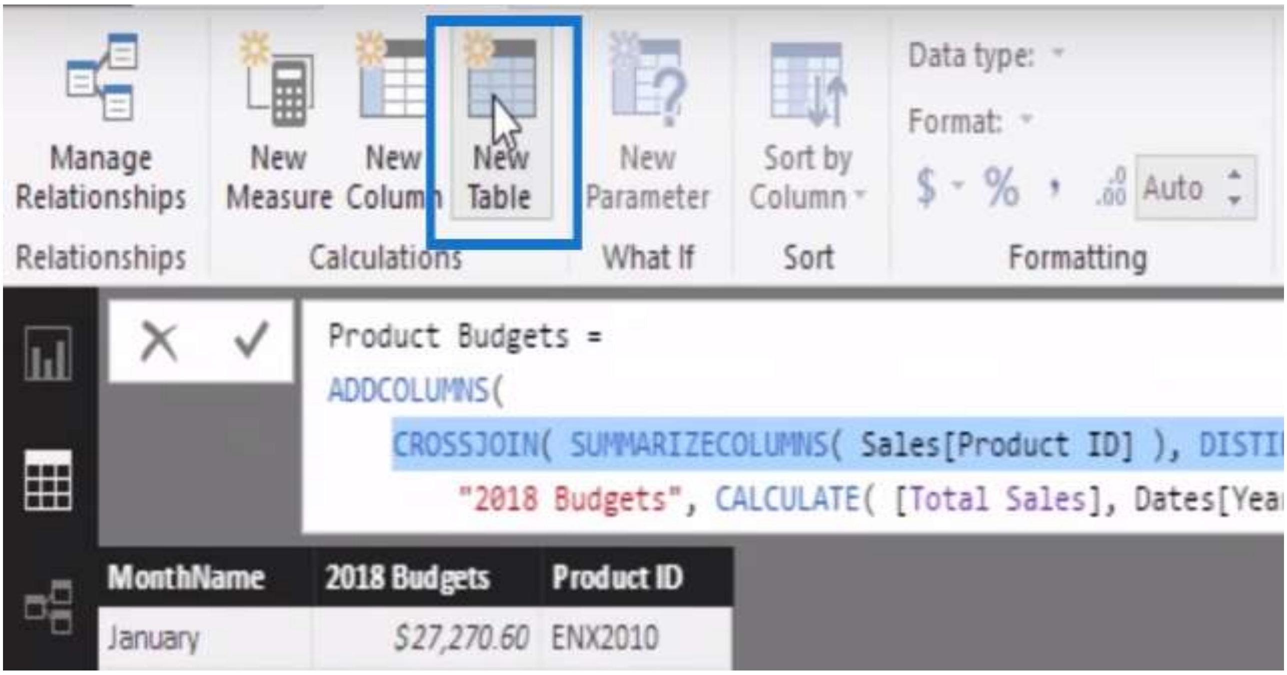
This function enabled me to create the MonthName column which came from the Date table, and the Product ID column which came from the Product table. So that's two totally different tables that I merged to create another table.



```
Product Budgets =
ADDCOLUMNS(
    CROSSJOIN( SUMMARIZECOLUMNS( Sales[Product ID] ), DISTINCT( Dates[MonthName] ) ),
        "2018 Budgets", CALCULATE( [Total Sales], Dates[Year]= 2017 ) * 1.075 )
```

MonthName	2018 Budgets	Product ID
January	$27,270.60	ENX2010
January	$6,905.80	ENX2009
January	$27,225.45	ENX2014
January	$30,493.45	ENX2033
January	$7,616.38	ENX2043
January	$24,451.95	ENX2090
January	$5,418.00	ENX2074
January	$15,983.10	ENX2049
January	$26,821.25	ENX2055
January	$20,157.33	ENX2082
January	$20,769.00	ENX2041
January	$22,337.43	ENX2040
January	$44,176.05	ENX2028
January	$46,362.60	ENX2065

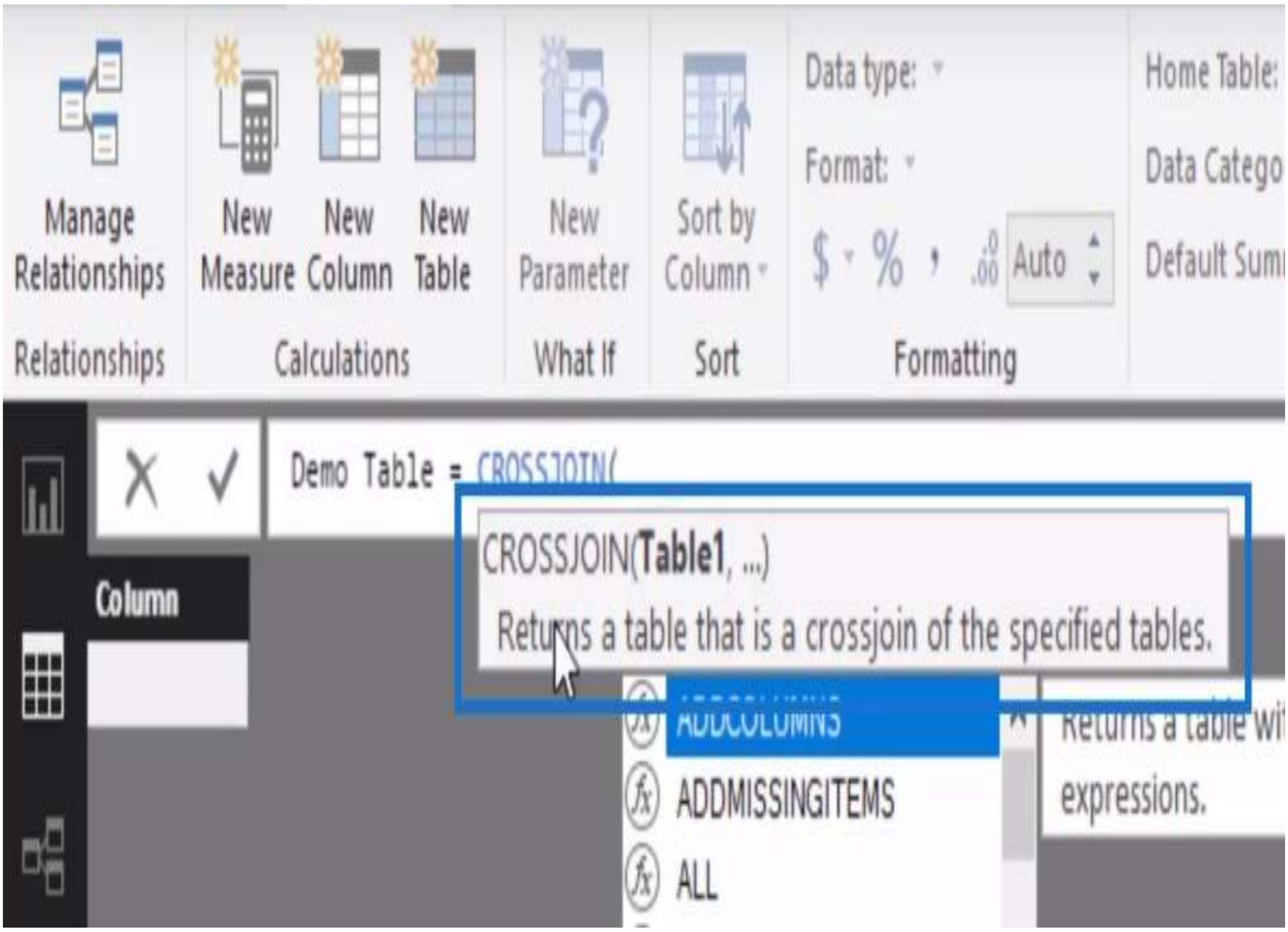
How did I do it? Let's create a brand new table by clicking this icon and using DAX formula.



```
Product Budgets =
ADDCOLUMNS(
    CROSSJOIN( SUMMARIZECOLUMNS( Sales[Product ID] ), DISTI
        "2018 Budgets", CALCULATE( [Total Sales], Dates[Yea
```

MonthName	2018 Budgets	Product ID
January	$27,270.60	ENX2010

I’m going to create a demo table here and use the CROSSJOIN function. As you can see, IntelliSense is pretty good at code completion; it tells us what we need to know about the function we just typed.



```
Demo Table = CROSSJOIN(
```

CROSSJOIN(**Table1**, ...)

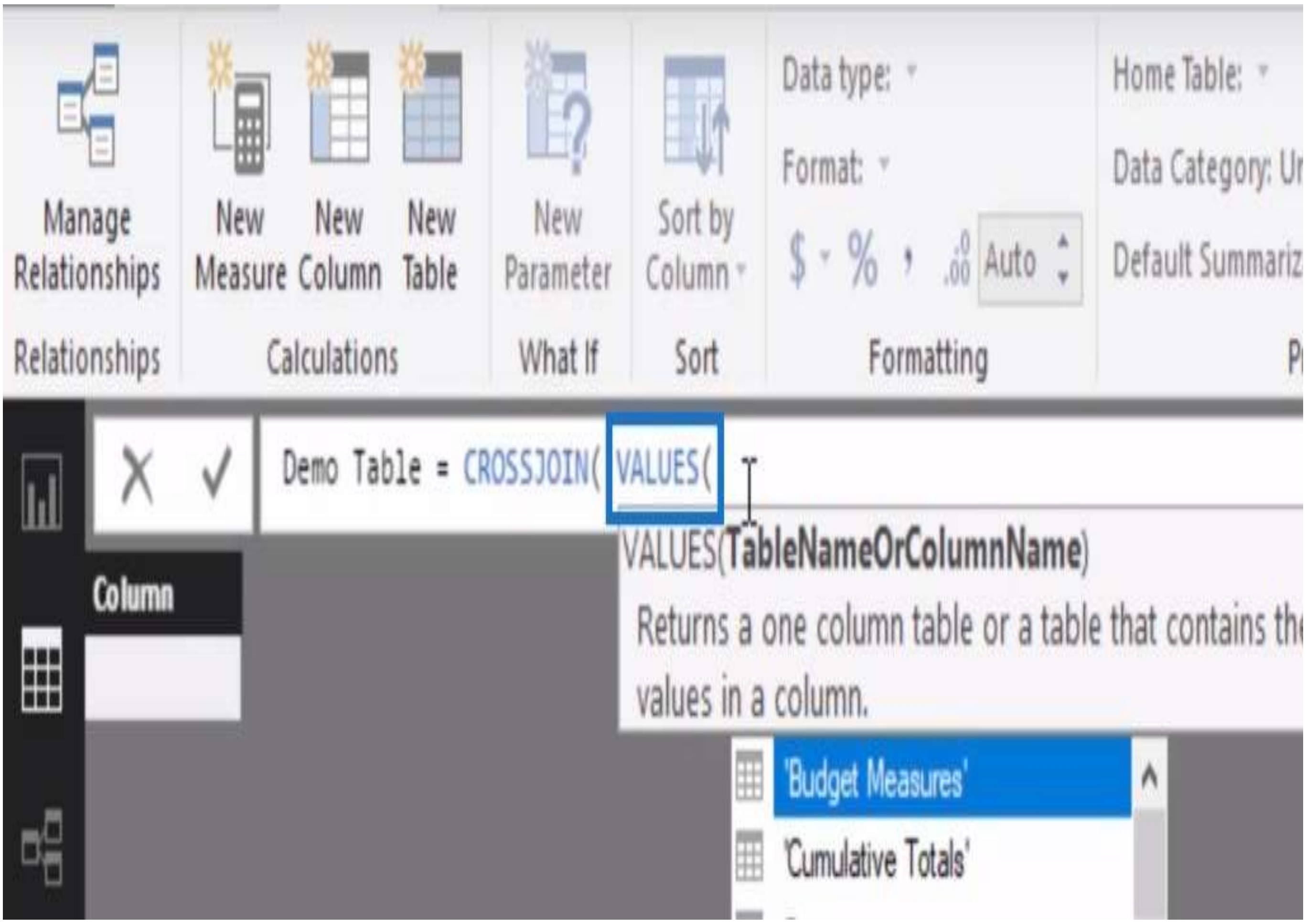
Returns a table that is a crossjoin of the specified tables.

Returns a table wi
expressions.

It informs us that CROSSJOIN “returns a table that is a crossjoin of the specified tables.”

So basically what this means is that the function will merge two tables.

I will start with VALUES, which allows us to easily extract or create a table of a particular column.

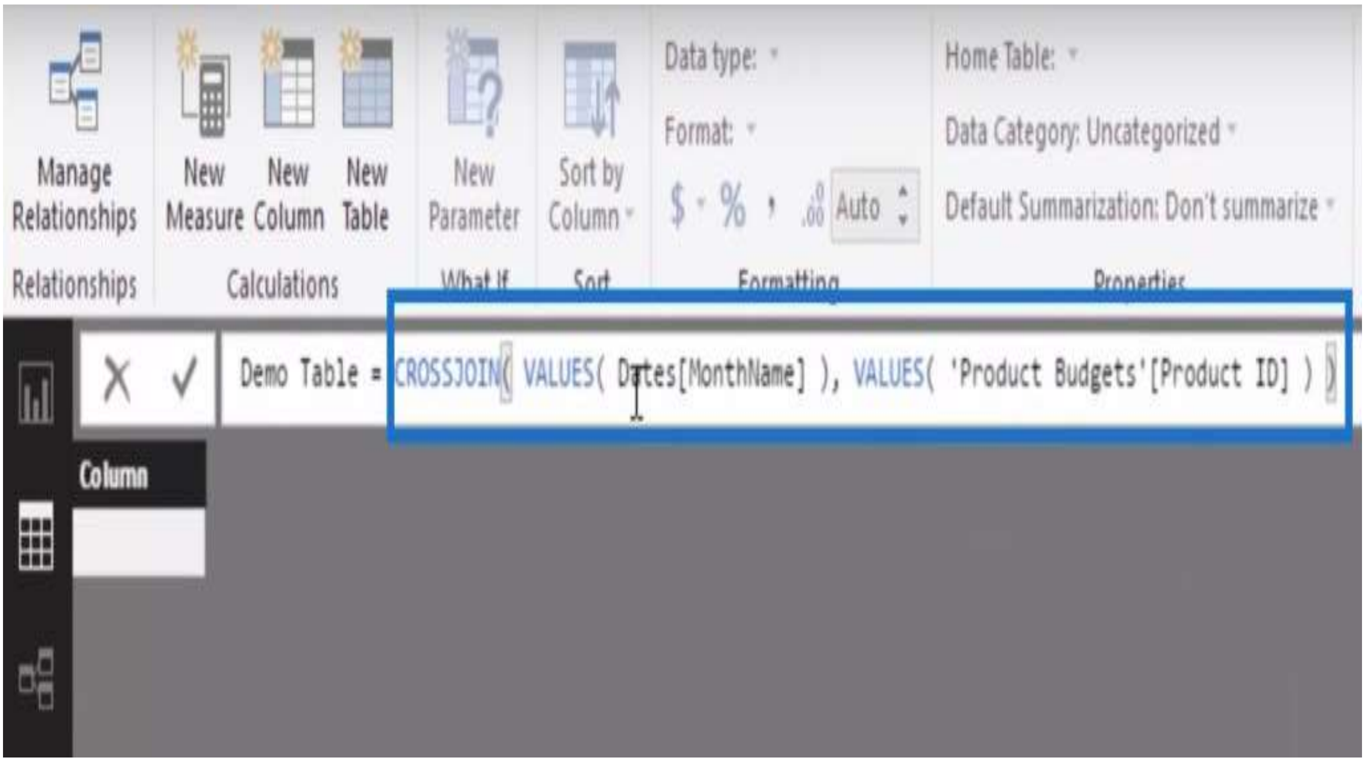


```
Demo Table = CROSSJOIN( VALUES(
```

VALUES(TableNameOrColumnName)

Returns a one column table or a table that contains th values in a column.

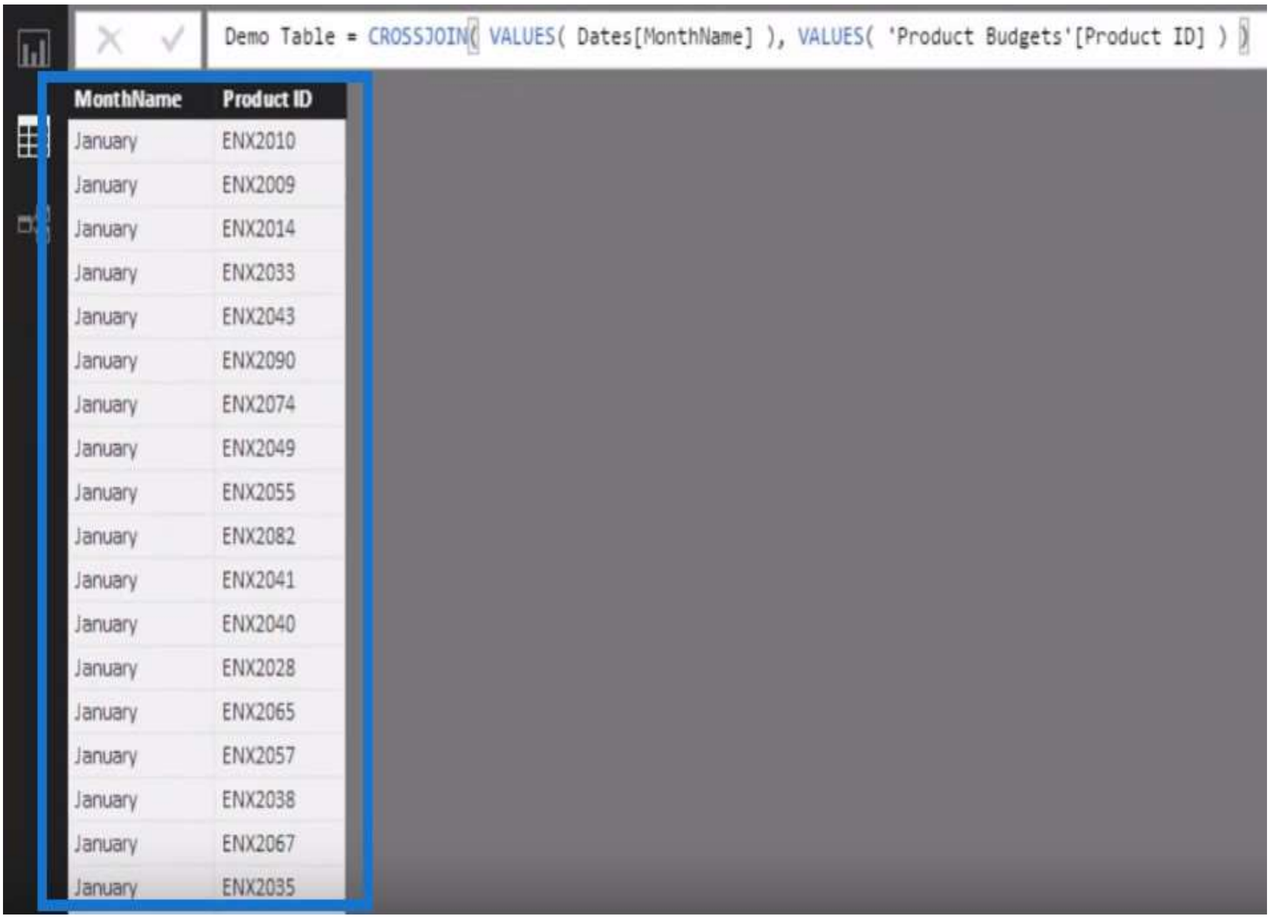
Then I'll add MonthName. Then I'll add VALUES again, then my Product ID.



```
Demo Table = CROSSJOIN( VALUES( Dates[MonthName] ), VALUES( 'Product Budgets'[Product ID] ) )
```

Remember that this doesn’t have to be a single date column. It can be multiple columns in a table. A table can be created virtually using a formula and a range of different table functions. So these are all a derivative of all these DAX formulas, while VALUES will just bring one column of information.

Using the CROSSJOIN will give me every single product and month.



```
Demo Table = CROSSJOIN( VALUES( Dates[MonthName] ), VALUES( 'Product Budgets'[Product ID] ) )
```

MonthName	Product ID
January	ENX2010
January	ENX2009
January	ENX2014
January	ENX2033
January	ENX2043
January	ENX2090
January	ENX2074
January	ENX2049
January	ENX2055
January	ENX2082
January	ENX2041
January	ENX2040
January	ENX2028
January	ENX2065
January	ENX2057
January	ENX2038
January	ENX2067
January	ENX2035

This is how you utilize CROSSJOIN – to create these brand new tables that didn’t exist before and effectively incorporate them into your models. Remember that these are from two totally different tables – the Dates and Product Budgets tables.

EXCEPT	This function returns the rows of one table which do not appear in another table.	EXCEPT (<table expression1>,<table expression2>)

Example for except function is :

```
Measure = COUNTROWS(EXCEPT('All(User)',User))
```

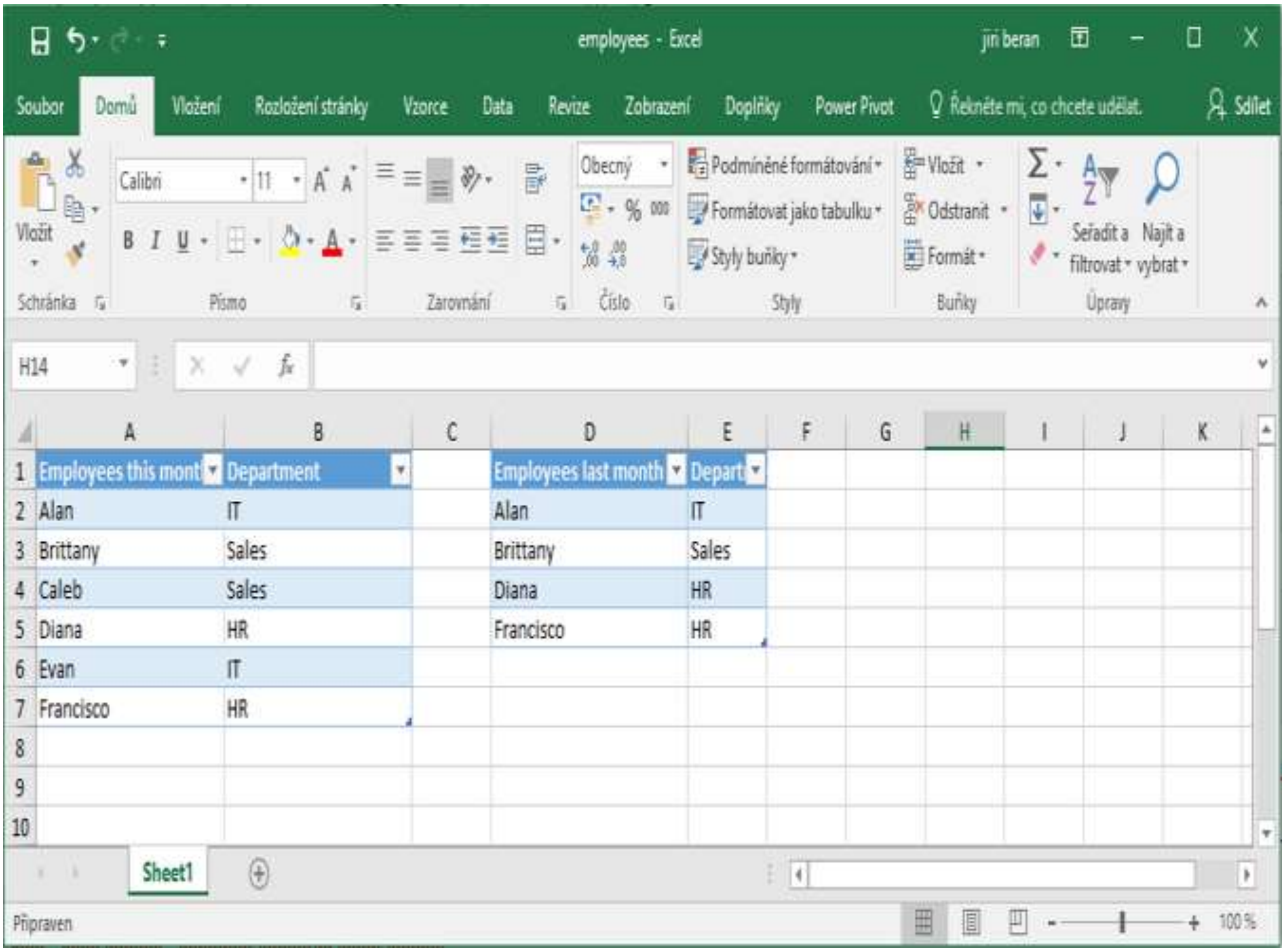
INTERSECT :

INTERSECT – finding of common rows in two tables (DAX – Power Pivot, Power BI)

INTERSECT does in DAX something similar to Inner connection in Power Query or Power BI queries.

To get the rows, that are NOT common, use EXCEPT.

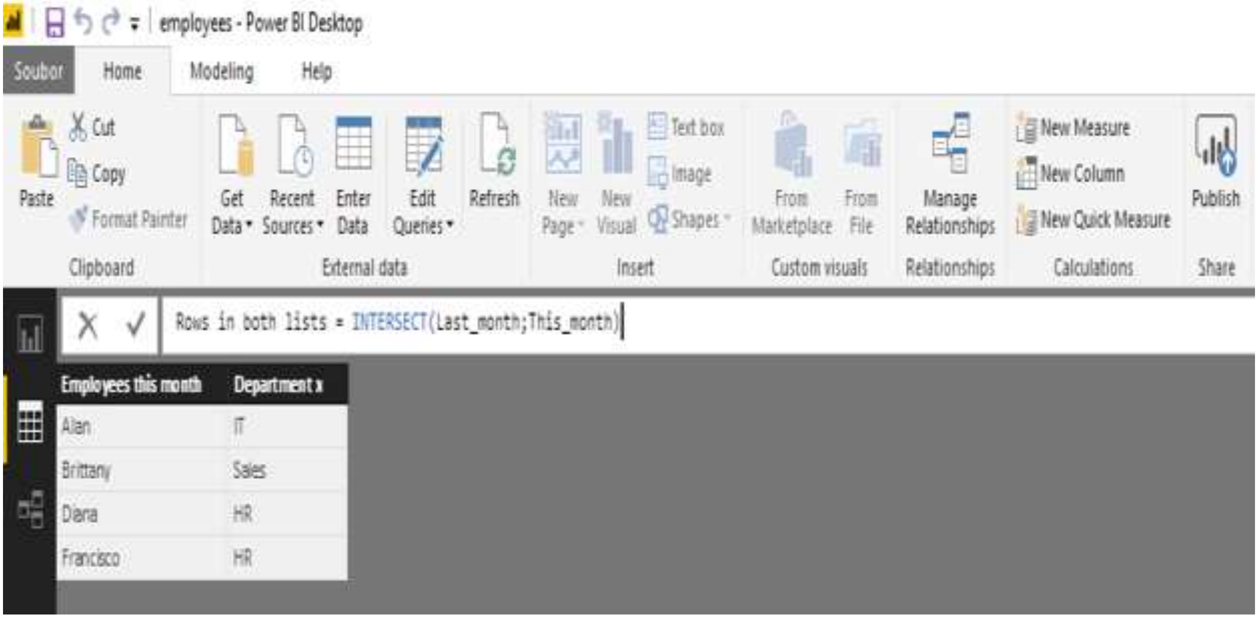
Here I mentioned two tables with examples :



Employees this mont	Department
Alan	IT
Brittany	Sales
Caleb	Sales
Diana	HR
Evan	IT
Francisco	HR

Employees last month	Depart
Alan	IT
Brittany	Sales
Diana	HR
Francisco	HR

Let´s upload both tables to Power BI. Then create new table and use INTERSECT to compare them. The syntax is totally simple - just write the names of both tables.



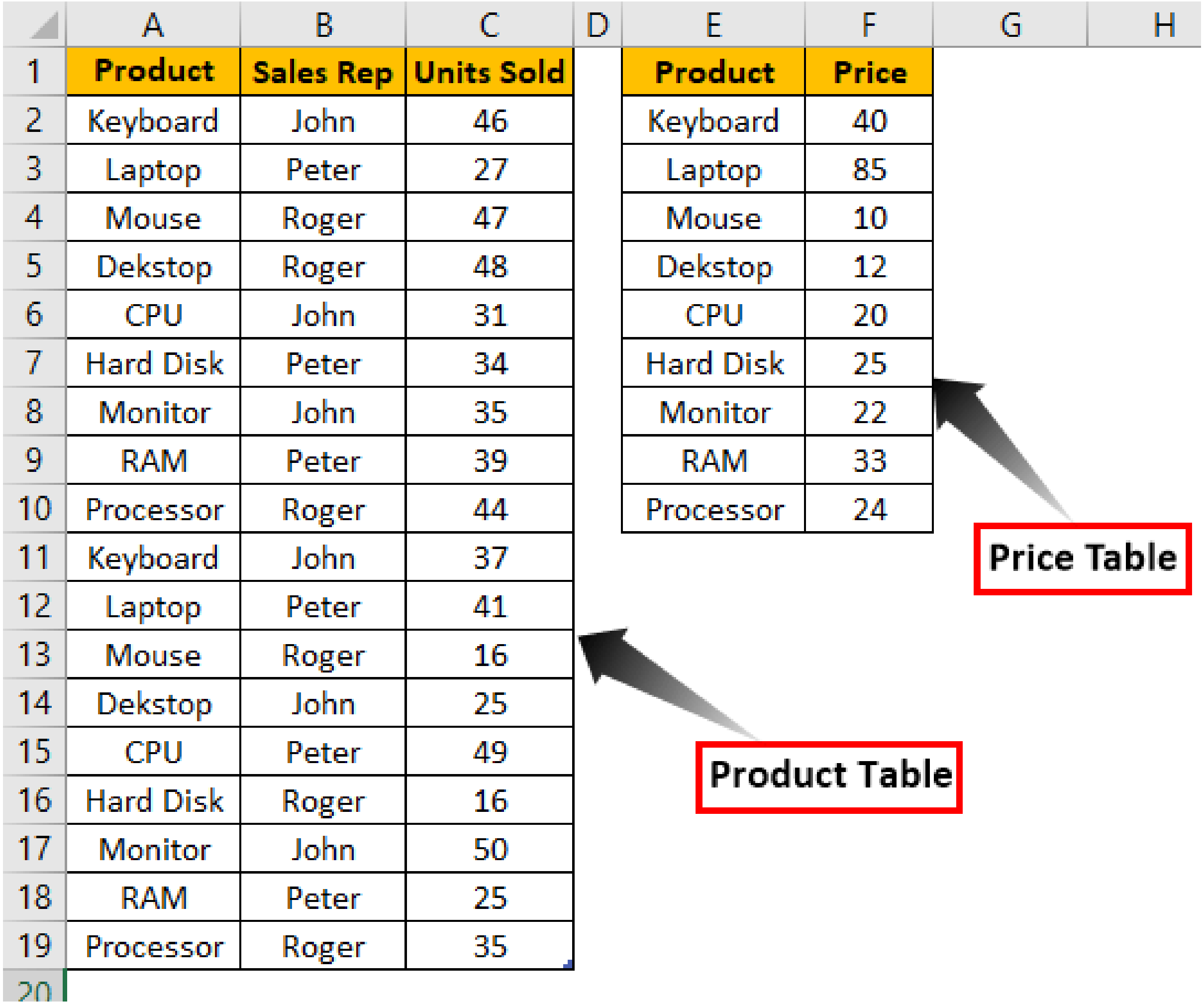
```
Rows in both lists = INTERSECT(Last_month;This_month)
```

Employees this month	Department x
Alan	IT
Brittany	Sales
Diana	HR
Francisco	HR

INTERSECT needs the same number of columns in both tables. It sees the rows identical if they have values in all columns identical.

b. Explain Relationship functions – RELATED, RELATEDTABLE and USERELATIONSHIPS with examples.

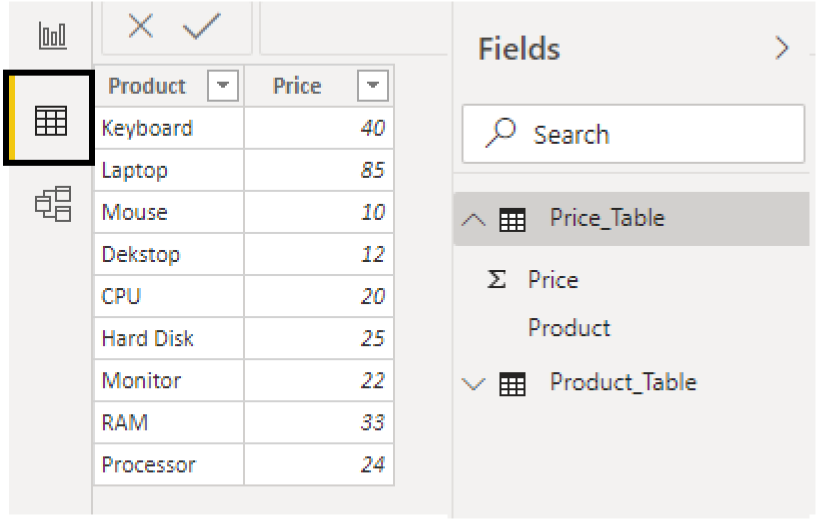
Related Fucntion :



Product	Sales Rep	Units Sold
Keyboard	John	46
Laptop	Peter	27
Mouse	Roger	47
Dekstop	Roger	48
CPU	John	31
Hard Disk	Peter	34
Monitor	John	35
RAM	Peter	39
Processor	Roger	44
Keyboard	John	37
Laptop	Peter	41
Mouse	Roger	16
Dekstop	John	25
CPU	Peter	49
Hard Disk	Roger	16
Monitor	John	50
RAM	Peter	25
Processor	Roger	35

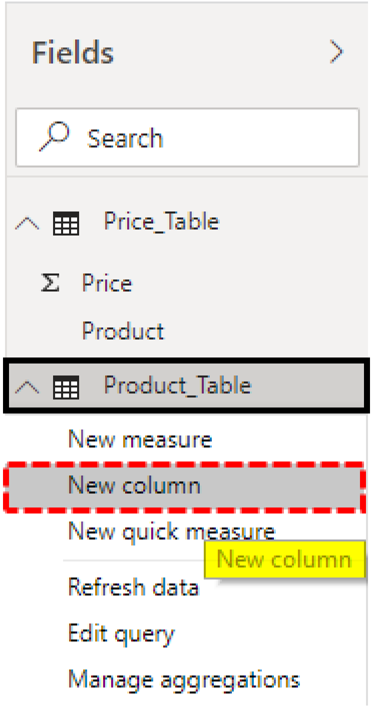
Product	Price
Keyboard	40
Laptop	85
Mouse	10
Dekstop	12
CPU	20
Hard Disk	25
Monitor	22
RAM	33
Processor	24

- Above, we have two tables, “Product Table” and “Price Table.” In “Product Table,” we have the product name and units sold details with the “Sales Rep” name for each product.
- In the “Price Table,” we have product names and their price per unit values, so we will use the RELATED function to fetch the price details to “Product Table.”Upload the above two table data to the Power BI Desktop file.

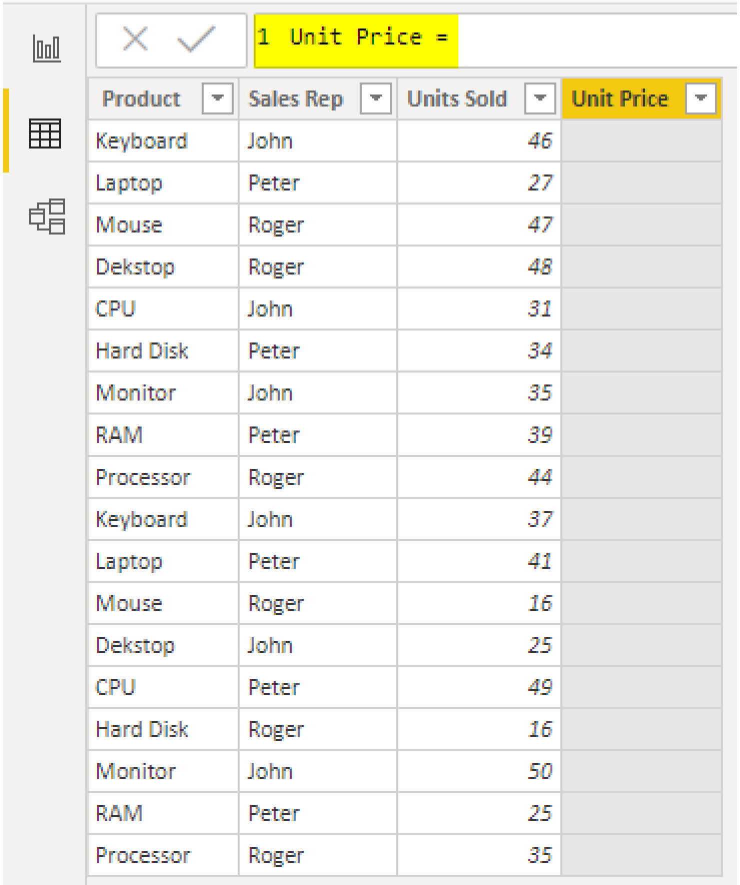


Product	Price
Keyboard	40
Laptop	85
Mouse	10
Dekstop	12
CPU	20
Hard Disk	25
Monitor	22
RAM	33
Processor	24

- Now from **"Price_Table,"** we need to fetch the cost price of each product to the "Product_Table."Right-click on the **"Product_Table"** and choose the option of "**New column**."



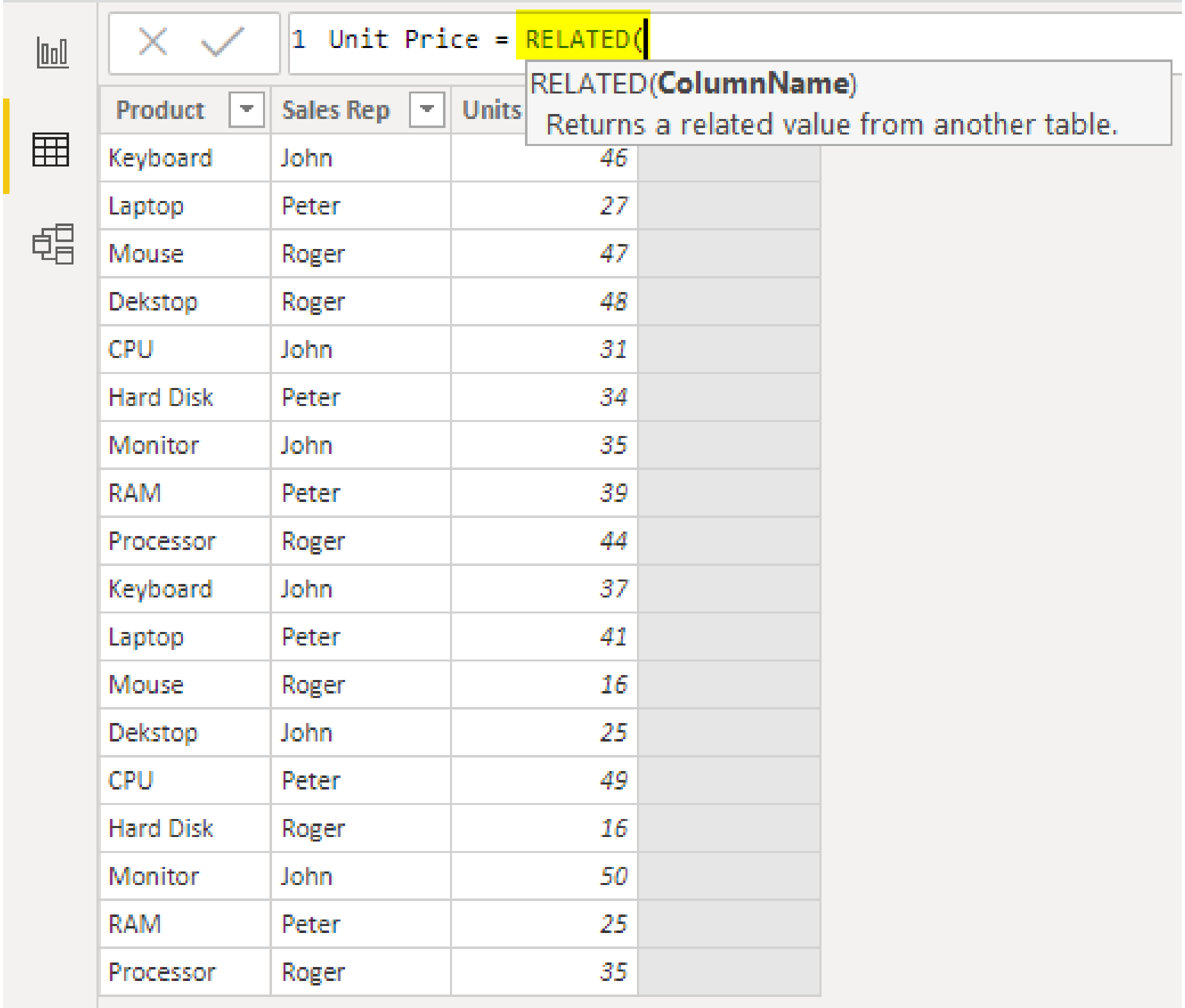
- Now, give the name for the new column as "Unit Price."



```
1 Unit Price =
```

Product	Sales Rep	Units Sold	Unit Price
Keyboard	John	46	
Laptop	Peter	27	
Mouse	Roger	47	
Dekstop	Roger	48	
CPU	John	31	
Hard Disk	Peter	34	
Monitor	John	35	
RAM	Peter	39	
Processor	Roger	44	
Keyboard	John	37	
Laptop	Peter	41	
Mouse	Roger	16	
Dekstop	John	25	
CPU	Peter	49	
Hard Disk	Roger	16	
Monitor	John	50	
RAM	Peter	25	
Processor	Roger	35	

- Open the RELATED function in Power BI.

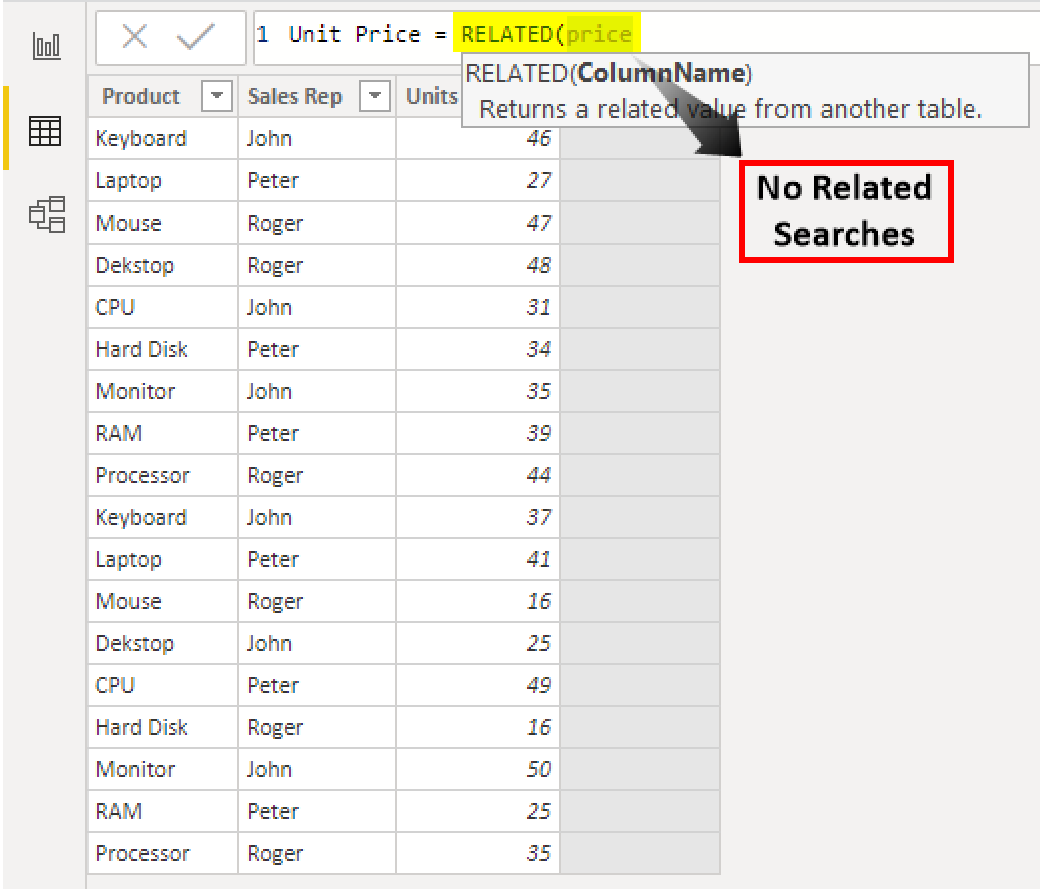


```
1 Unit Price = RELATED(
```

RELATED(**ColumnName**)
Returns a related value from another table.

Product	Sales Rep	Units	
Keyboard	John	46	
Laptop	Peter	27	
Mouse	Roger	47	
Dekstop	Roger	48	
CPU	John	31	
Hard Disk	Peter	34	
Monitor	John	35	
RAM	Peter	39	
Processor	Roger	44	
Keyboard	John	37	
Laptop	Peter	41	
Mouse	Roger	16	
Dekstop	John	25	
CPU	Peter	49	
Hard Disk	Roger	16	
Monitor	John	50	
RAM	Peter	25	
Processor	Roger	35	

- We need to choose the column from the **“Price_Table**,” but when you type the table name, we do not see any related searches.



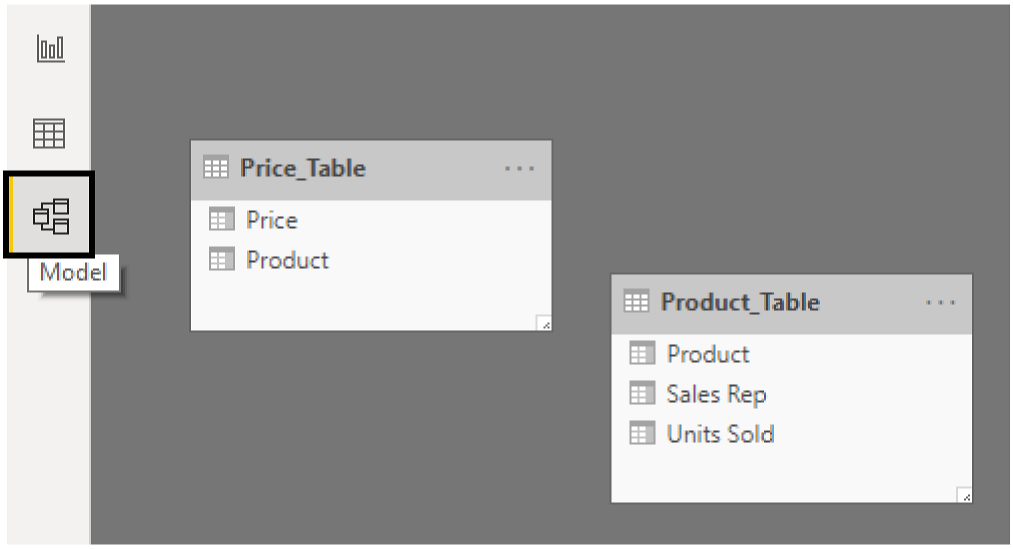
```
1  Unit Price = RELATED(price
```

RELATED(**ColumnName**)
Returns a related value from another table.

No Related Searches

Product	Sales Rep	Units
Keyboard	John	46
Laptop	Peter	27
Mouse	Roger	47
Dekstop	Roger	48
CPU	John	31
Hard Disk	Peter	34
Monitor	John	35
RAM	Peter	39
Processor	Roger	44
Keyboard	John	37
Laptop	Peter	41
Mouse	Roger	16
Dekstop	John	25
CPU	Peter	49
Hard Disk	Roger	16
Monitor	John	50
RAM	Peter	25
Processor	Roger	35

It is because before we use the RELATED function, we must create a relationship between two tables under the "**Data Modeling**" tab.



Price_Table
Price
Product

Product_Table
Product
Sales Rep
Units Sold

- As you can see above, we do not have any relationship between these two tables. However, we can create the relationship between these two tables by using the common column between these two tables. So in these two tables, the common column is "**Product**.

RELATEDTABLE Functions :

RELATEDTABLE

The RELATEDTABLE function works in the opposite direction to RELATED.
The RELATEDTABLE function starts from the table on the one-side of the relationship and gives access to the table on the many-side of the relationship. RELATEDTABLE is a table function, and returns a table of values that contains all of the rows on the many-side that are related to the current row on the one-side.

To use the RELATEDTABLE function, you specify the table name that contains the related data that you want.

RELATEDTABLE(<tableName>)

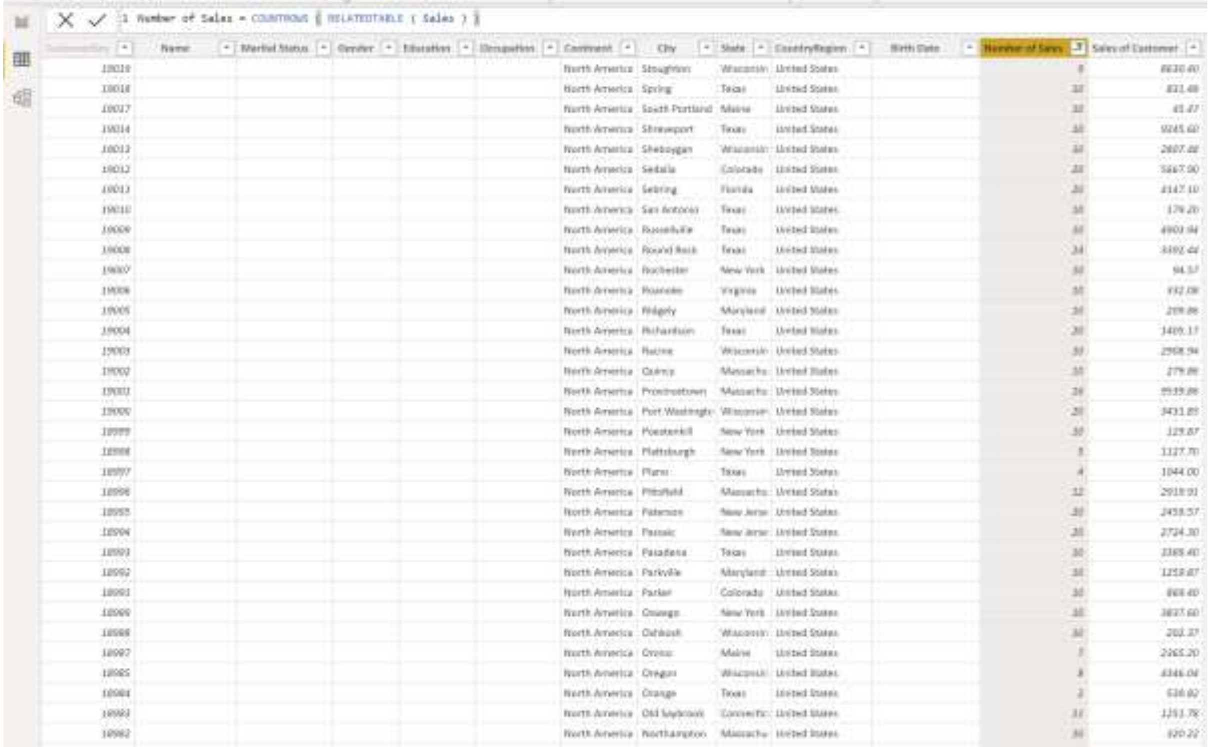
Using RELATEDTABLE

Now let's demonstrate how we can use the RELATEDTABLE function to calculate the number of sales made by each customer.

To do this, we can use the RELATEDTABLE function to create a new calculated column in the 'Customer' table:

Number of Sales = COUNTROWS (RELATEDTABLE (Sales))

Here, for each customer, RELATEDTABLE will return a sub-set of the 'Sales' table that only includes rows related to that customer. COUNTROWS will then simply count the number of rows in this sub-table. As a result, the calculated column will display number of rows in the 'Sales' table that are related to each customer.



```
1 Number of Sales = COUNTROWS ( RELATEDTABLE ( Sales ) )
```

CustomerKey	Name	Marital Status	Gender	Education	Occupation	Continent	City	State	Country/Region	Birth Date	Number of Sales	Sales of Customer
19019						North America	Stoughton	Wisconsin	United States		8	8630.40
19018						North America	Spring	Texas	United States		32	831.48
19017						North America	South Portland	Maine	United States		32	45.47
19014						North America	Shreveport	Texas	United States		34	9245.60
19013						North America	Sheboygan	Wisconsin	United States		34	2807.88
19012						North America	Sedalia	Colorado	United States		28	5667.90
19011						North America	Sebring	Florida	United States		28	4147.10
19010						North America	San Antonio	Texas	United States		34	179.20
19009						North America	Russellville	Texas	United States		34	4903.94
19008						North America	Round Rock	Texas	United States		24	3392.44
19007						North America	Rochester	New York	United States		32	94.57
19006						North America	Roanoke	Virginia	United States		35	932.08
19005						North America	Ridgely	Maryland	United States		34	209.86
19004						North America	Richardson	Texas	United States		28	1409.17
19003						North America	Racine	Wisconsin	United States		33	2908.94
19002						North America	Quincy	Massachu	United States		35	279.89
19001						North America	Provincetown	Massachu	United States		28	8539.88
19000						North America	Port Washingto	Wisconsin	United States		28	3431.85
18999						North America	Poestenkill	New York	United States		30	129.87
18998						North America	Plattsburgh	New York	United States		5	1127.70
18997						North America	Plano	Texas	United States		4	1044.00
18996						North America	Pittsfield	Massachu	United States		12	2918.91
18995						North America	Paterson	New Jerse	United States		30	2458.57
18994						North America	Passaic	New Jerse	United States		28	2724.30
18993						North America	Pasadena	Texas	United States		30	3365.40
18992						North America	Parkville	Maryland	United States		34	1259.87
18991						North America	Parker	Colorado	United States		30	868.40
18989						North America	Oswego	New York	United States		30	3837.60
18988						North America	Oshkosh	Wisconsin	United States		30	202.37
18987						North America	Orono	Maine	United States		7	2265.20
18985						North America	Oregon	Wisconsin	United States		8	4346.04
18984						North America	Orange	Texas	United States		2	536.82
18983						North America	Old Saybrook	Connectic	United States		31	1253.78
18982						North America	Northampton	Massachu	United States		36	920.22

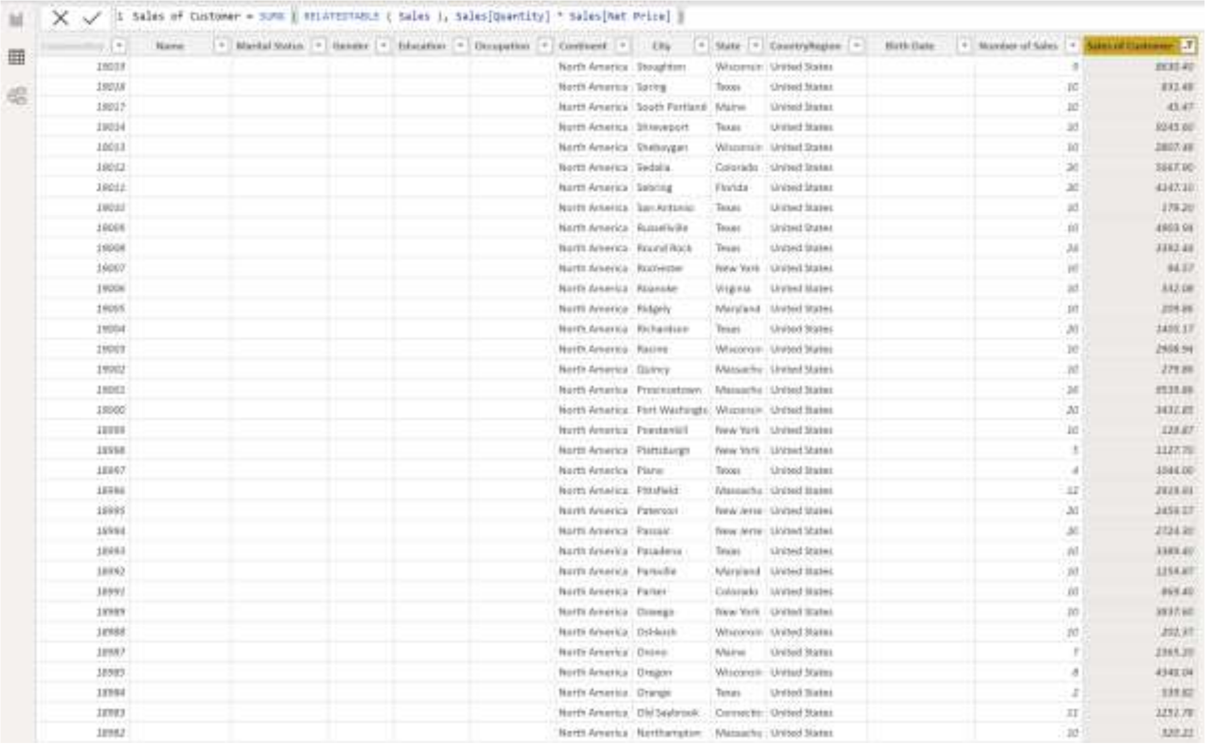
Let's go one step further. What if now, instead of counting the sales made by each customer, we want to compute the total amount sold to each customer?

To do this, we can use the RELATEDTABLE function to create a new calculated column in the 'Customer' table and show the total sales amount for each customer:

Sales of Customer = SUMX (RELATEDTABLE (Sales), Sales[Quantity] * Sales[Net Price])

In order to calculate the sales amount for each customer, we need to first find all of the transactions made by each customer. So, for each record in the 'Customer' table, RELATEDTABLE will travel through the existing relationship between the tables, and will populate a list of rows (a sub-table) from the 'Sales' table. This sub-table contains all of the sales related to that specific customer in the sales table. This table is then used as the input for the SUMX function. For each row in the sub-table, the sales amount is calculated using the SUMX iterator, and is then aggregated to give a total sales amount for that customer which is displayed in the new calculated column in the 'Customer' table.

Here, you can see the combination of: a calculated column, an expression, an iterator (SUMX) and a table function (RELATEDTABLE) together, allowing us to enrich the 'Customer' table with interesting information.



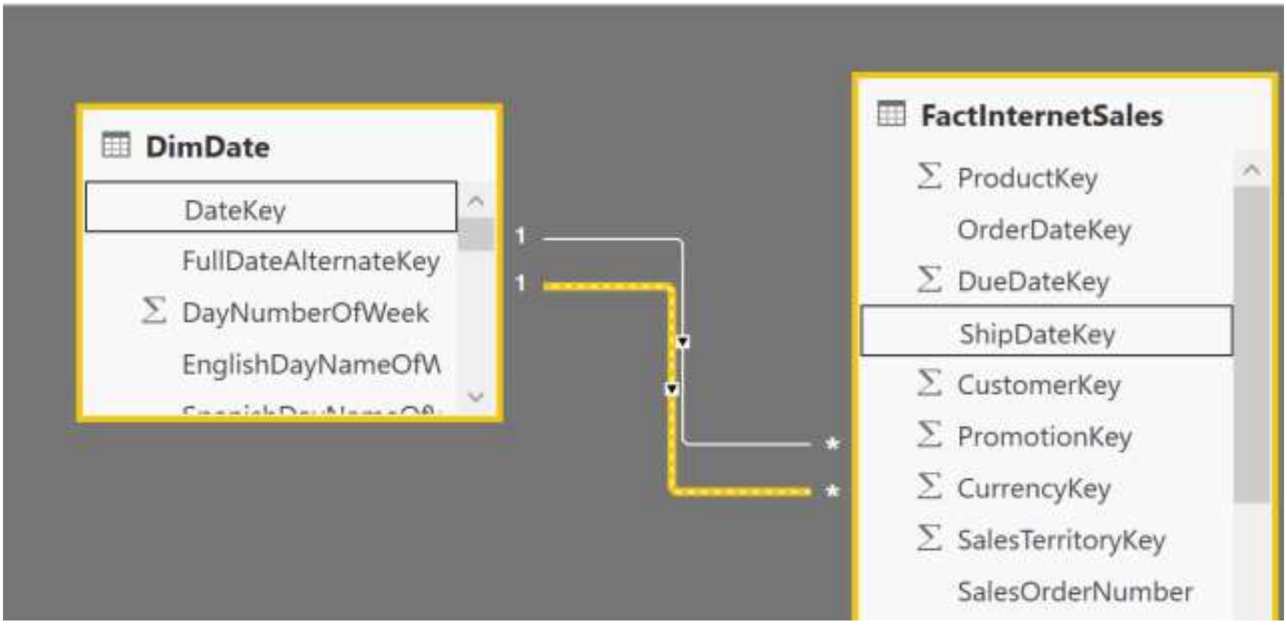
```
1 Sales of Customer = SUMX ( RELATEDTABLE ( Sales ), Sales[Quantity] * Sales[Net Price] )
```

[illegible]	Name	Marital Status	Gender	Education	Occupation	Continent	City	State	CountryRegion	Birth Date	Number of Sales	Sales of Customer
19019						North America	Stoughton	Wisconsin	United States		9	[illegible]
19018						North America	Spring	Texas	United States		10	832.48
19017						North America	South Portland	Maine	United States		10	43.47
19014						North America	Shreveport	Texas	United States		20	[illegible]
19013						North America	Sheboygan	Wisconsin	United States		10	2807.48
19012						North America	Sedalia	Colorado	United States		20	3567.90
19011						North America	Sebring	Florida	United States		20	4247.10
19010						North America	San Antonio	Texas	United States		10	179.20
19009						North America	Russellville	Texas	United States		10	4903.94
19008						North America	Round Rock	Texas	United States		24	3382.48
19007						North America	Rochester	New York	United States		10	84.37
19006						North America	Roanoke	Virginia	United States		10	342.08
19005						North America	Ridgely	Maryland	United States		10	209.86
19004						North America	Richardson	Texas	United States		20	1400.17
19003						North America	Racine	Wisconsin	United States		10	2906.94
19002						North America	Quincy	Massachu	United States		10	279.88
19001						North America	Provincetown	Massachu	United States		24	[illegible]
19000						North America	Port Washingto	Wisconsin	United States		20	3432.85
18999						North America	Poestenkill	New York	United States		10	128.87
18998						North America	Plattsburgh	New York	United States		5	1127.70
18997						North America	Plano	Texas	United States		4	1044.00
18996						North America	Pittsfield	Massachu	United States		12	2828.81
18995						North America	Paterson	New Jerse	United States		20	2458.17
18994						North America	Passaic	New Jerse	United States		30	2724.30
18993						North America	Pasadena	Texas	United States		10	3389.40
18992						North America	Parkville	Maryland	United States		10	1254.87
18991						North America	Parker	Colorado	United States		10	869.40
18989						North America	Oswego	New York	United States		10	3837.60
18988						North America	Oshkosh	Wisconsin	United States		10	202.37
18987						North America	Orono	Maine	United States		7	2365.20
18985						North America	Oregon	Wisconsin	United States		8	4348.04
18984						North America	Orange	Texas	United States		2	339.82
18983						North America	Old Saybrook	Connectic	United States		11	2252.78
18982						North America	Northampton	Massachu	United States		10	320.22

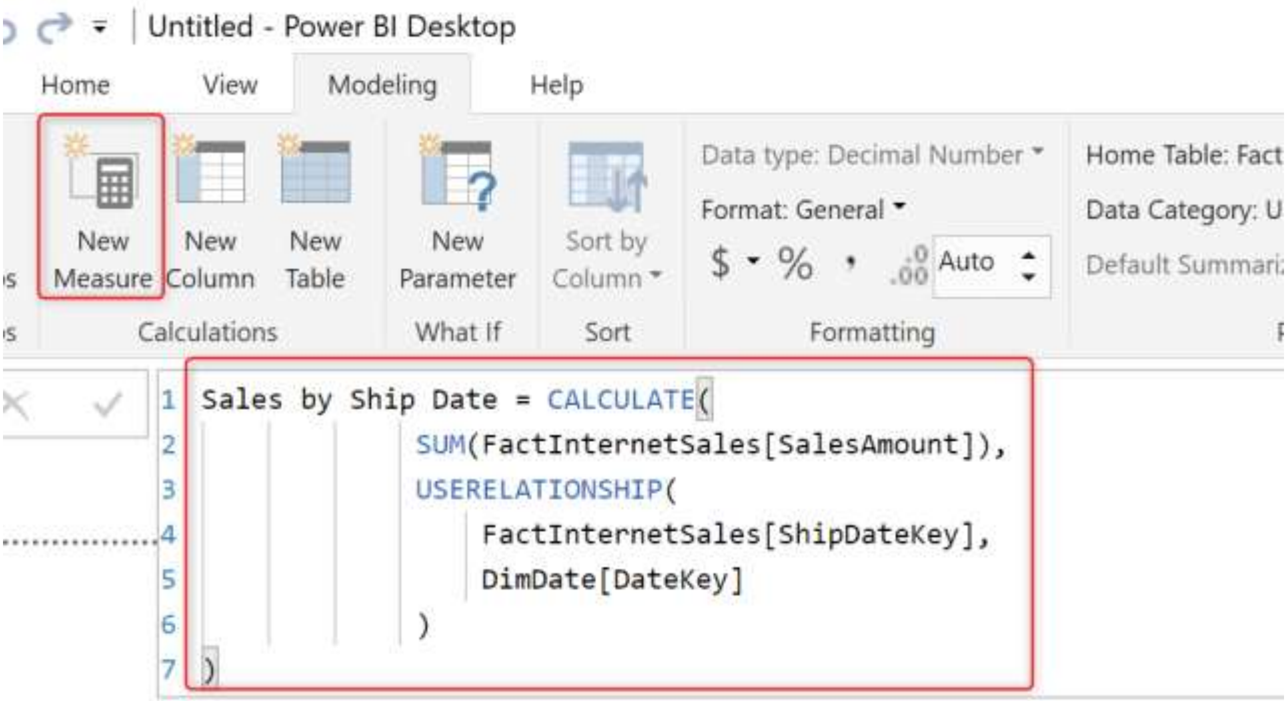
UseRelationship Function in DAX

Another method to handle inactive relationship is to use a function in DAX called UseRelationship. This DAX function is literally saying to Power BI that for this expression, use this relationship, even if it is inactive. Let’s see how this function works.

If we continue the same example of slicing and dicing by Ship Date and assume that there is no Ship Date calculated table created, then we can do it this way; Create the inactive relationship between DimDate and FactInternetSales again based on the ShipDateKey.



Now, let’s create a Measure in Power BI with below expression:

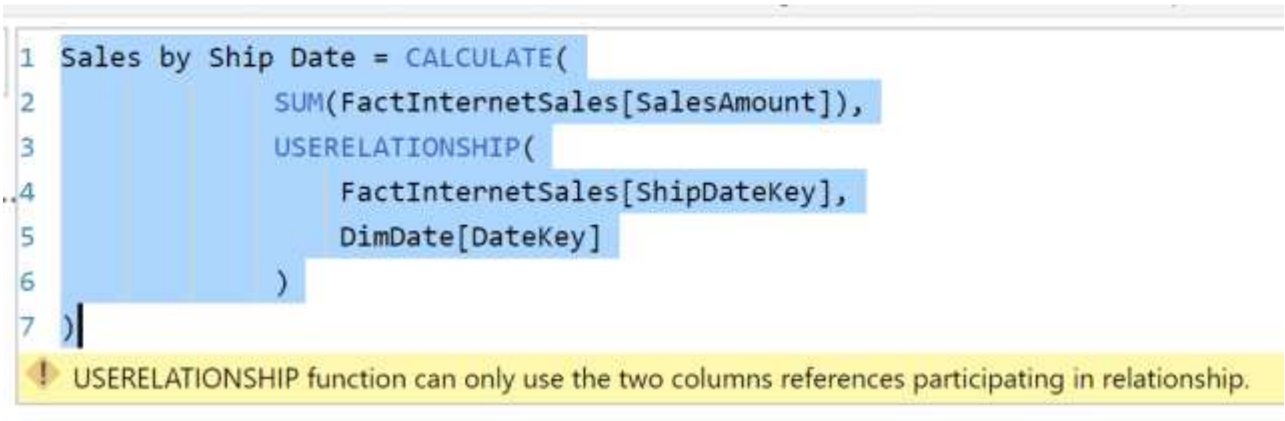


```
Sales by Ship Date = CALCULATE(
            SUM(FactInternetSales[SalesAmount]),
            USERELATIONSHIP(
                FactInternetSales[ShipDateKey],
                DimDate[DateKey]
            )
)
```

```
Sales by Ship Date = CALCULATE(

SUM(FactInternetSales[SalesAmount]),

USERELATIONSHIP(

FactInternetSales[ShipDateKey],

DimDate[DateKey]

)

)
```

This measure calculates the sum of sales by ship date. The whole secret is the usage of the UseRelationship function. This is a really simple function to use, you just need to provide two input columns to it, the two columns that are two sides of the relationship. Their order is not important.

UseRelationship (<column 1>, <column 2>)
The important tip to consider is that you HAVE to have an existing inactive relationship for this function to work, otherwise you get the error below:



```
Sales by Ship Date = CALCULATE(
            SUM(FactInternetSales[SalesAmount]),
            USERELATIONSHIP(
                FactInternetSales[ShipDateKey],
                DimDate[DateKey]
            )
)
```

! USERELATIONSHIP function can only use the two columns references participating in relationship.

Inactive relationship must exist otherwise the Use Relationship doesn’t work.

c. Create an automated calendar table using Order table and add column like Month and Year and apply date formatting.

   Files attached sir/mam..