Power Query #8

Connecting to an SQL Server Database with Power Query

Table of Contents

Project 1 & 2 Goals for Video 08 2

Efficient Query Performance Guidelines 3

Some things that usually lead to more efficient queries: 3

SQL Databases & Query Folding 3

Steps in Projects: 5

Download files 5

1) Steps for the first query in Excel 5

Group By Command 10

2) The second Query in Excel 15

3) The third query in Excel 16

4) The fourth query in Power BI Desktop 19

Notes for Direct Query in Power BI 20

Columnar Database 21

Project 1 & 2 Goals for Video 08





7 Million Rows

Transformation Requirements:

1) Pull Data From Both fTransactions and dProduct Tables
2) Include only transactions where the Quantity Sold > = 100
3) Calculate Net Revenue
4) Group By Product to get Total Net Revenue
5) Include only Group By Aggregations >=100,000
6) Sort Product Column

Power Query User Interface:

Product	Total Net Revenue
Alpine	11776864.07
Aspen	13958950.15
Bellen	28574273.3
Bower Aussie Round	25786269.36
Carlota	25235273.38
Crested Beaut	20983406.77
Darnell Tri Fly	4524925.493
Eagle	9274236.026
Fire Aspen	6589276.39
Fun Fly	36948046.09
GelFast	7416772.194
Manu LD	298465.776
Mejestic Beaut	16691525.19
Phoenix	9807279.089
Quad	92418009.38
Sunset	12248978.96
Sunshine	12666365.66
Sunspot	4141542.476
Yanaki	15145727.81

SQL written in Power Query:

Product	NetRevenue
Alpine	11776864.07
Aspen	13958950.15
Bellen	28574273.3
Bower Aussie Round	25786269.36
Carlota	25235273.38
Crested Beaut	20983406.77
Darnell Tri Fly	4524925.493
Eagle	9274236.026
Fire Aspen	6589276.39
Fun Fly	36948046.09
GelFast	7416772.194
Manu LD	298465.776
Mejestic Beaut	16691525.19
Phoenix	9807279.089
Quad	92418009.38
Sunset	12248978.96
Sunshine	12666365.66
Sunspot	4141542.476
Yanaki	15145727.81

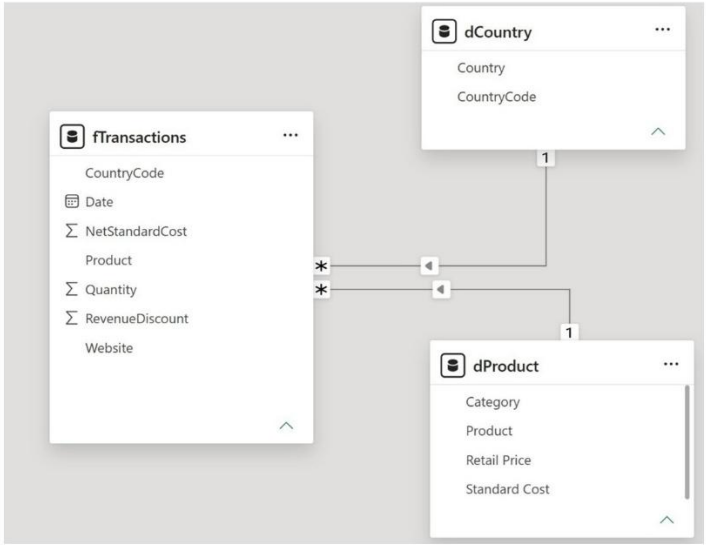
Using BoomData database:

Product	Total Net Revenue
Alpine	11775836.02
Aspen	13958950.15
Bellen	28573034.75
Bower Aussie Round	25778339.37
Carlota	25236426.35
Crested Beaut	20982050.03
Darnell Tri Fly	4525724.184
Eagle	9274236.026
Fire Aspen	6589276.39
Fun Fly	36949406.02
GelFast	7416772.194
Manu LD	298465.776
Mejestic Beaut	16691525.19
Phoenix	9807279.089
Quad	92417003.71
Sunset	12249347.43
Sunshine	12667541.08
Sunspot	4141776.651
Yanaki	15147232.08





7 Million Rows



Efficient Query Performance Guidelines

Efficient Query Performance means that the query can evaluate and refresh as quickly as possible. For example, if you have seven million rows of records in a fact table and you create a formula in a column, the formula must iterate over seven million rows. But if all the rows in the fact table are not needed for the final data model or reporting tasks, if you filter out the unnecessary rows, the column formula has to iterate over fewer rows and therefore the query is likely to evaluate, calculate and refresh more quickly. In the queries we built in Excel, when you used the condition "Quantity >= 100" to filter the fTransactions table, the fact table went from 7,000,000 rows down to 293,543 rows and so when the column formula evaluated, it had many fewer rows to iterate and therefore the query performed more efficiently.

Some things that usually lead to more efficient queries:

1) Remove unnecessary columns. This can help to reduce column scanning in the query, can make file size in the final load location smaller and it can make the reporting area easier to navigate. This this video when removed all columns except for Product, Quantity, RevenueDiscount, and dProduct.
2) Remove unnecessary rows. This can help lower the number of query table iterations and can make file size in the final load location smaller. In this video we filtered the fast table and the fact table went from 7,000,000 rows down to 293,543 rows and so when the column formula evaluated, it had many fewer rows to iterate and therefore the query performed more efficiently.
3) If you are using a reference to a query (like a reference to table or list) in a column formula, you can use the Table.Buffer or List.Buffer functions to reduce calls to the query. Example coming in the Power Class video about Custom Functions.
4) Don't use the aggregate feature in the Group By feature: it is VERY slow. Example coming in the Power Class video about Group By and the Table.Group function.
5) Remove repetitive query steps and consolidate into one step. We saw an example of this in videos #2 and #3 in this class when we consolidated the Changed Types query step.
6) When connecting to an SQL database, organize query steps so that the maximum number of steps can be sent back to the SQL database. We saw an example of this in this video.
7) Avoid high-cardinality merges.
8) Avoid merging without index columns.
9) Avoid From the Folder feature with too many files.
10) Unless you need a primary key in a fact table, remove the primary key.
11) Filter rows and remove columns after a merge rather than before.
12) Much more…

SQL Databases & Query Folding

1. **SQL**
    - An SQL database is a relational database that uses SQL Code (Structured Query Language) to create the database and to query or communicate with the database.
    - Our class will use Power Query to generate the M Code that allows us to connect to an SQL Database & extract data from an SQL Database.
    - Our class is not about learning how to write SQL Code.

2. **Credentials for server database that we use in the video examples:**

Server:	pond.highline.edu
Database 1:	boomerang
Database 2:	boomdata
User:	excelisfun
Password:	ExcelIsFun!

3. **Query Folding: connecting to SQL Server Database and using Power Query user interface:**
    - **Steps we create are sent back to SQL Database (called Query Folding).**
    - **SQL Database is often more efficient (performance) than Power Query.**
    - Some Power Query Features, like Custom M Code, Table.UnPivot function, Adding Data Types cannot be understood by SQL Database and will performed natively in Power Query.
    - Table.Buffer Function can prevent Query Folding. (Structured Query Language)
4. **The main coding clauses for queries with SQL (Structured Query Language) code are:**
    - S = SELECT = Select Columns (Fields, Attributes) and/or make Aggregate Calculations
    - F = FROM = What Tables (Relations) & What Relationships
    - W = WHERE = Filters Rows in a Table based on Conditions or Criteria
    - G = GROUP BY = Group By Calculation to create unique list of items ad the make an aggregate calculation for each item
    - H = HAVING = Filter Rows Based on Criteria for Group By Calculation
    - O = ORDER BY = Sorting
5. The SQL Code we will use to copy and paste into the SQL Database dialog box is here:

```
SELECT dProduct.Product, SUM(Quantity*(1-RevenueDiscount)*RetailPrice) AS NetRevenue
FROM fTransactions
JOIN dProduct
ON dProduct.Product = fTransactions.Product
WHERE Quantity >= 100
GROUP BY dProduct.Product
HAVING SUM(Quantity*(1-RevenueDiscount)*RetailPrice) >= 100000
ORDER BY dProduct.Product;
```

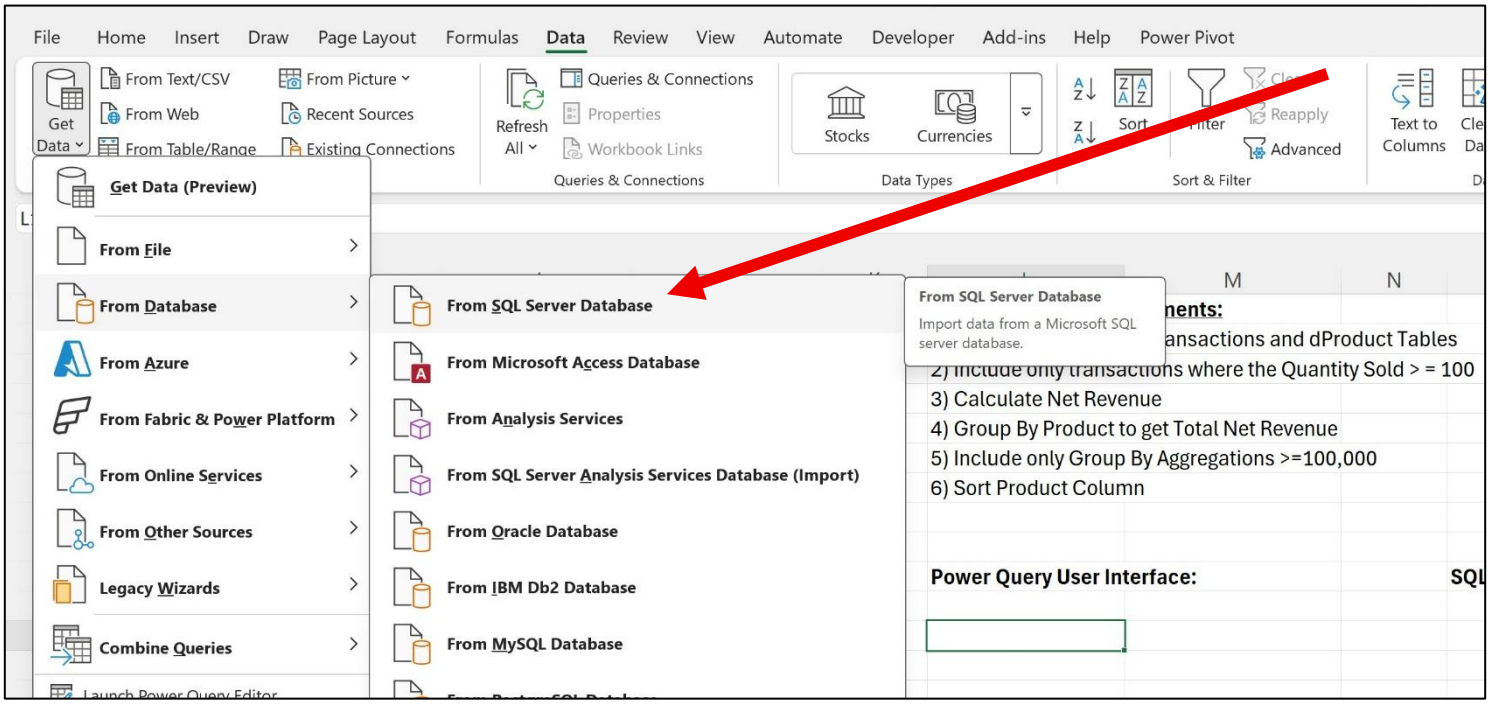
Steps in Projects:

Download files

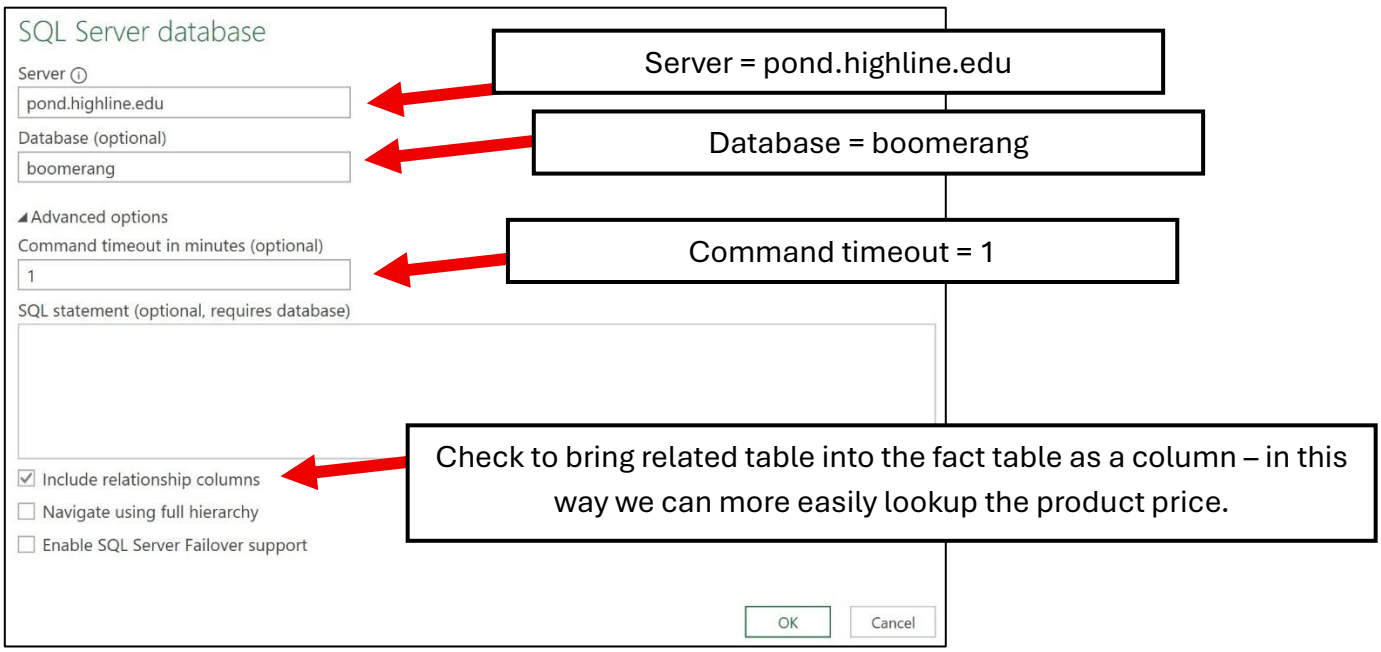
As we learned in the last video, download the zipped folder below Video #4, then unzip it in the same location you used for the previous video.

1) Steps for the first query in Excel

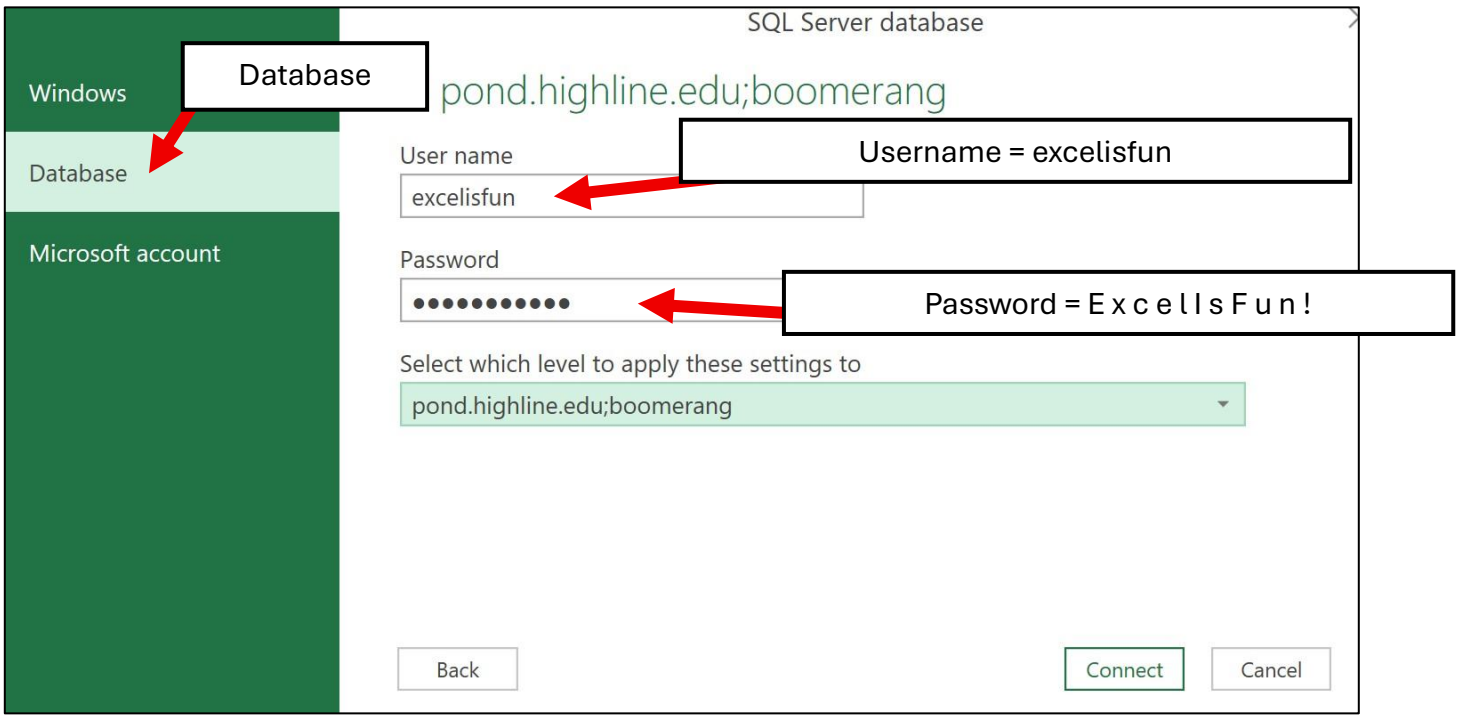
1) As shown below, to connect to an SQL Server Database in Excel Power Query, in the Data tab, Get & Transform group, click the Get Data dropdown, then click From Data Base and then click From SQL Server Database.



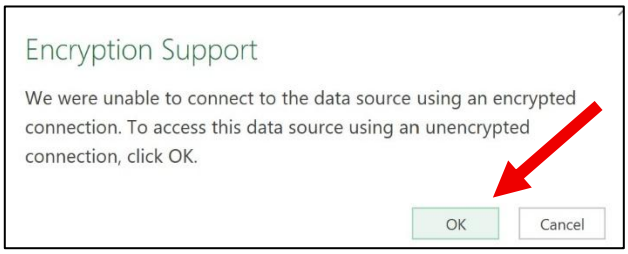
2) As shown below, complete the SQL Server Database dialog box:



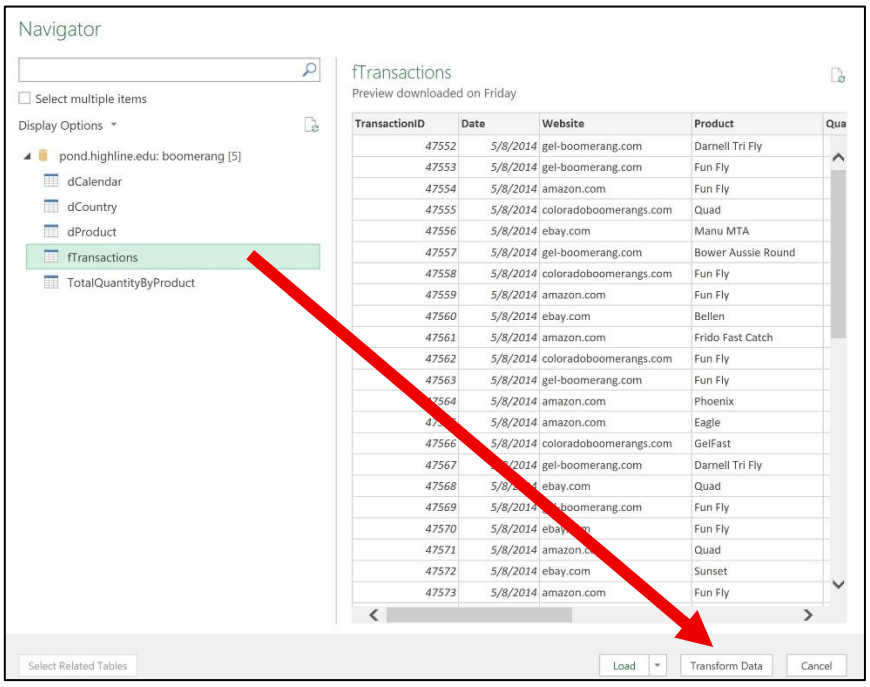
3) As shown below, select Database and then enter credentials:



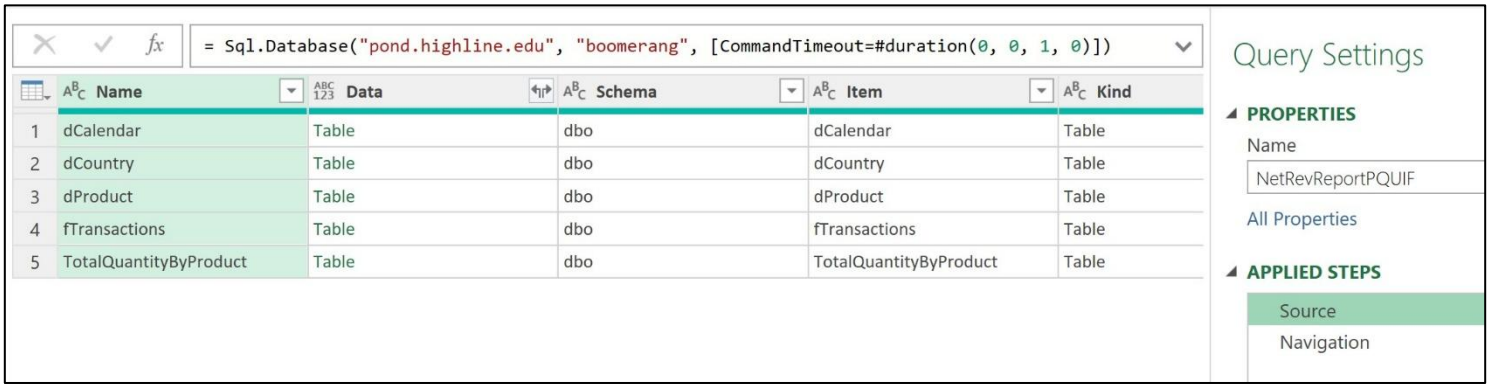
4) Click OK in the Encryption dialog box:



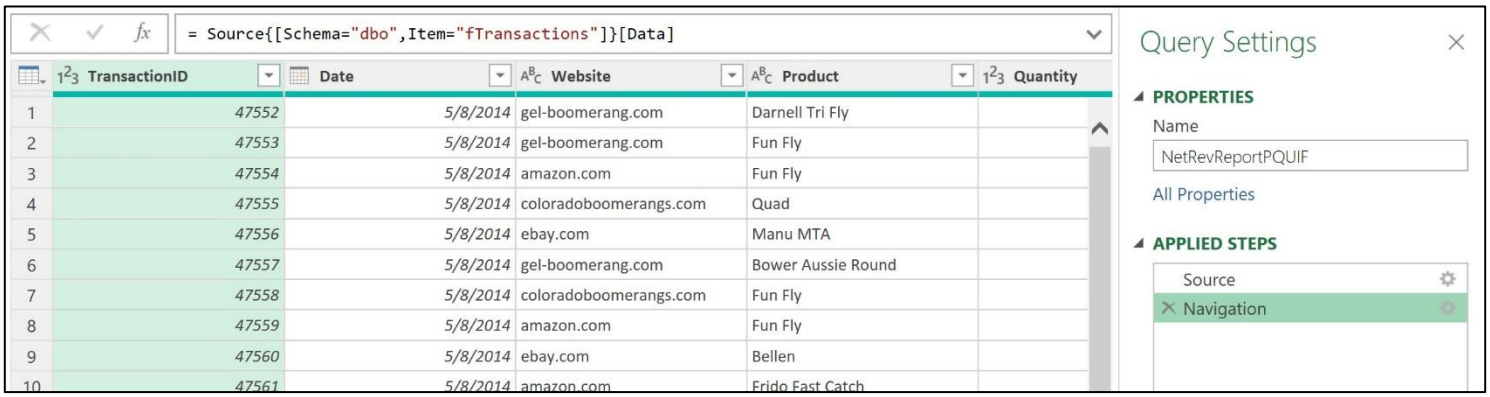
5) In the Navigation dialog box, select the fTransactions table and then click the Transform Data button:



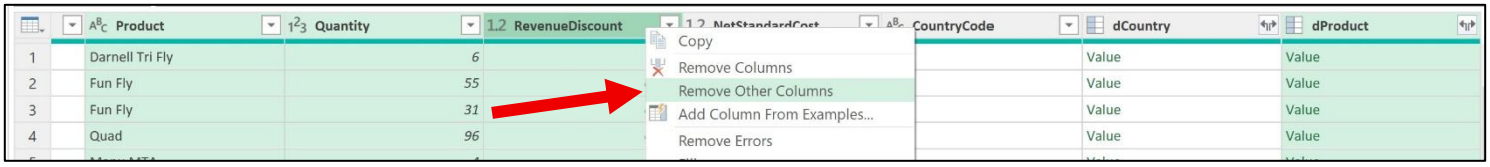
6) When you connect to an SQL database, Power Query uses the Sql.Database function, as shown below:



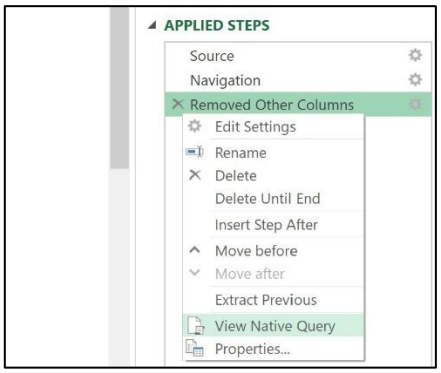
7) The navigation step imports the fTransactions table:



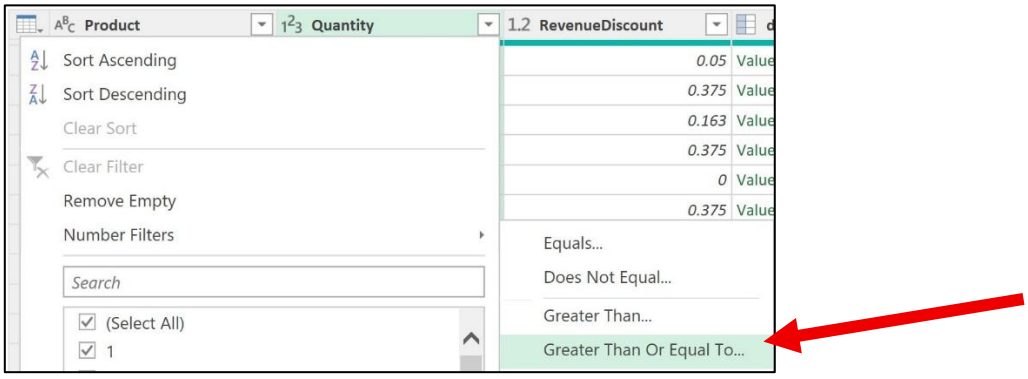
8) As shown below, the first step that we create is to select the Product, Quantity, RevenueDiscount, and dProducts columns and then right-click any one of the selected columns and click Remove Other Columns:



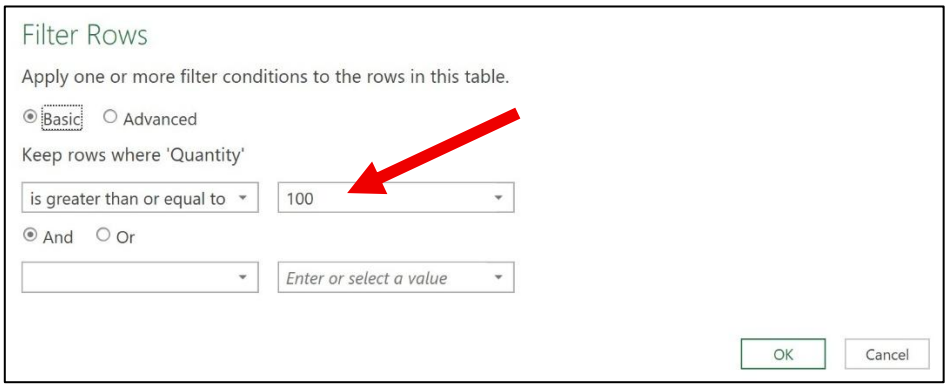
9) As shown below, when you right-click the step, the View Native Query option is visible and therefore the step is being sent back to the SQL Server Database for evaluation.



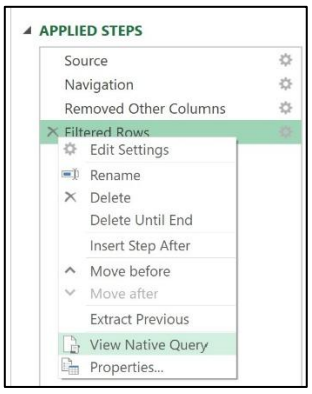
10) As shown below, for the next query step, filter the Quantity column with Greater Than Or Equal To...



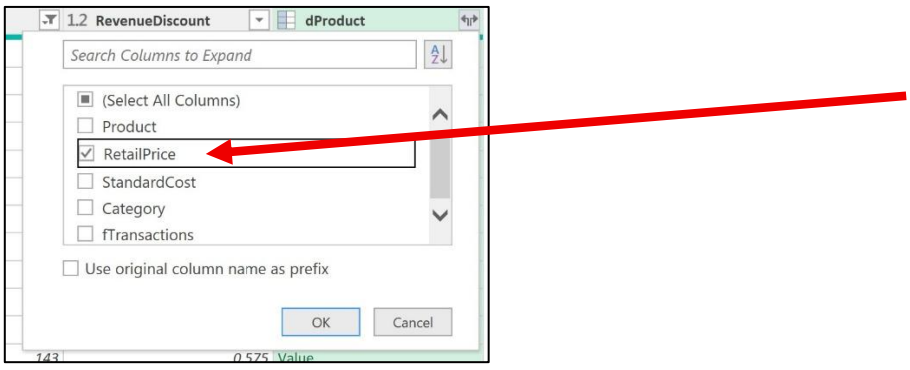
11) As shown below, enter the lower limit of 100:



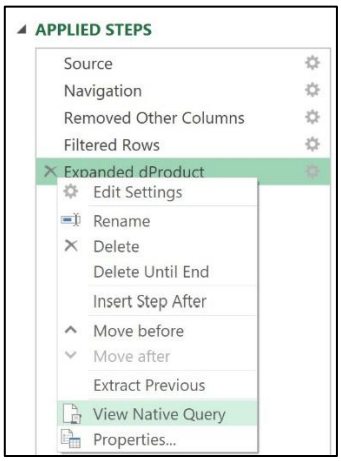
12) As shown below, when you right-click the step, the View Native Query option is visible and therefore the step is being sent back to the SQL Server Database for evaluation.



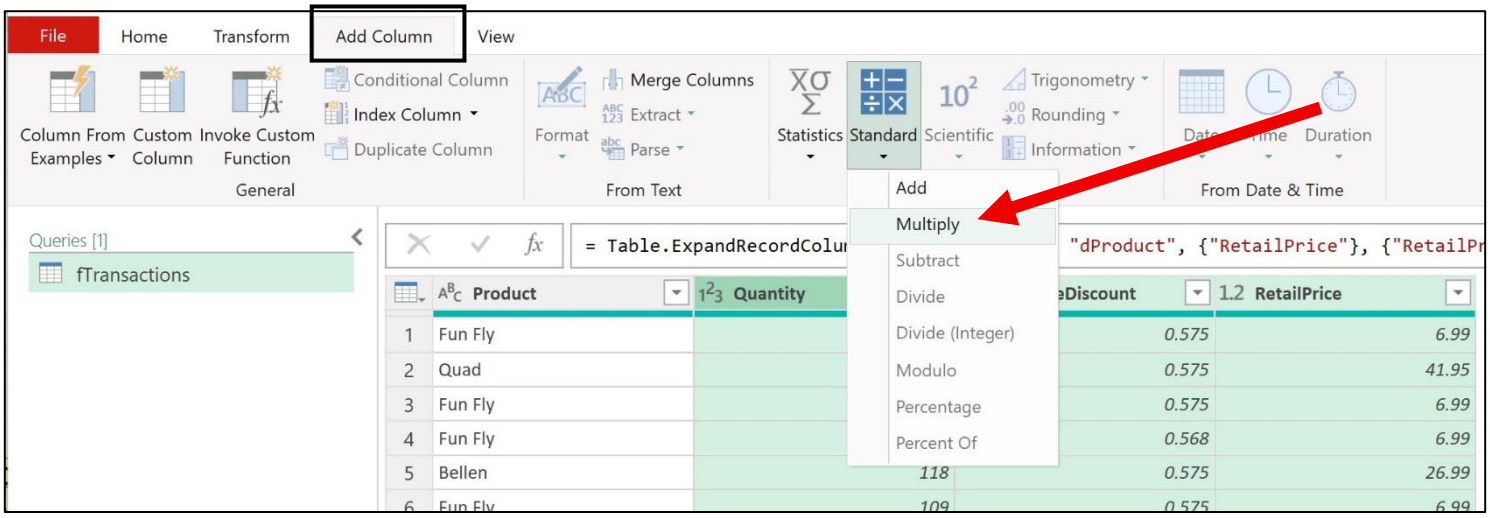
13) As shown below, for the next query step, expand the dProduct column to get RetailPrice:



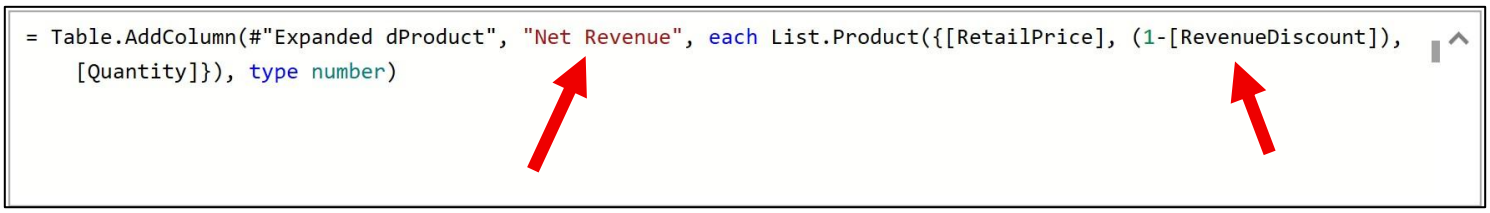
14) As shown below, when you right-click the step, the View Native Query option is visible and therefore the step is being sent back to the SQL Server Database for evaluation.



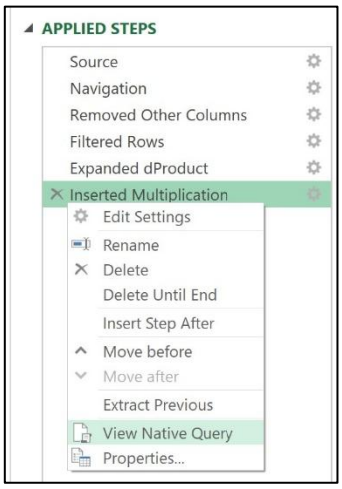
15) As shown below, for the next query step, select the three columns: Quantity, RevenueDiscount, and RetailPrice, and then in the Add Column tab, in the From Number group, click the Standard dropdown arrow and then select Multiply:



16) As shown below, in the formula bar edit the arguments in the Table.AddColumn function so that the second argument contains “Net Revenue” and the third argument contains the element (1-[RevenueDiscount]) rather than just [Revenue]:



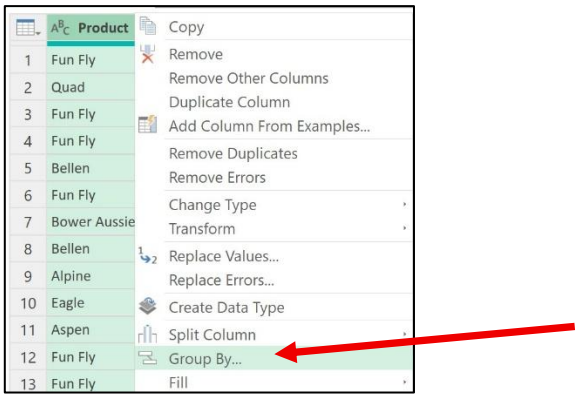
17) As shown below, when you right-click the step, the View Native Query option is visible and therefore the step is being sent back to the SQL Server Database for evaluation, even though we edited the formula!



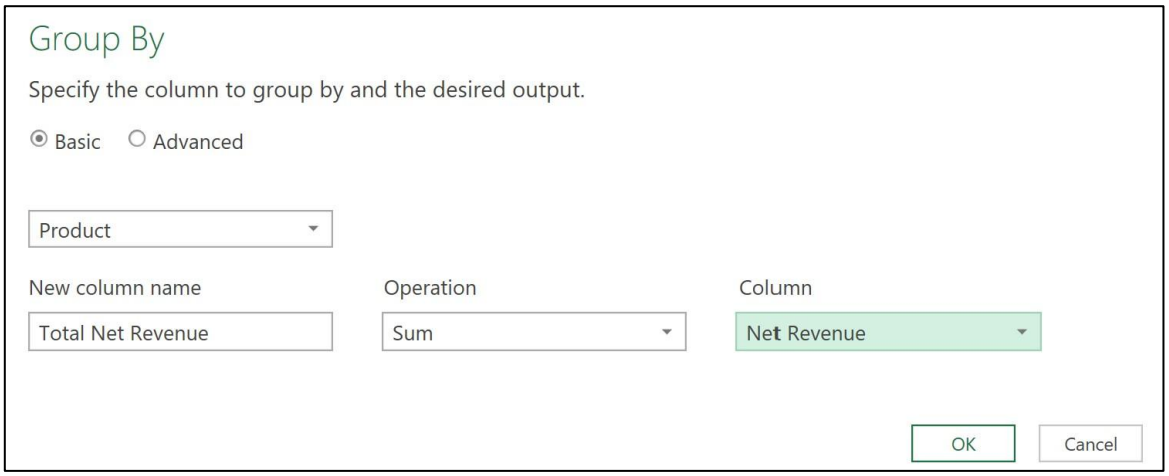
Group By Command

The Group By command in SQL coding and in M Code, performs an action just like a PivotTable does, because it can create a unique list of items from a column and then create aggregate calculations for each unique item in the unique list.

18) As shown below, for the next query step, right-click the Product column and from the menu select Group By:



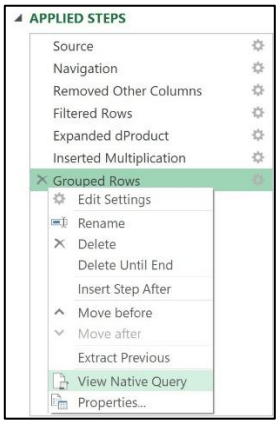
19) Complete the Group By dialog box as shown below:



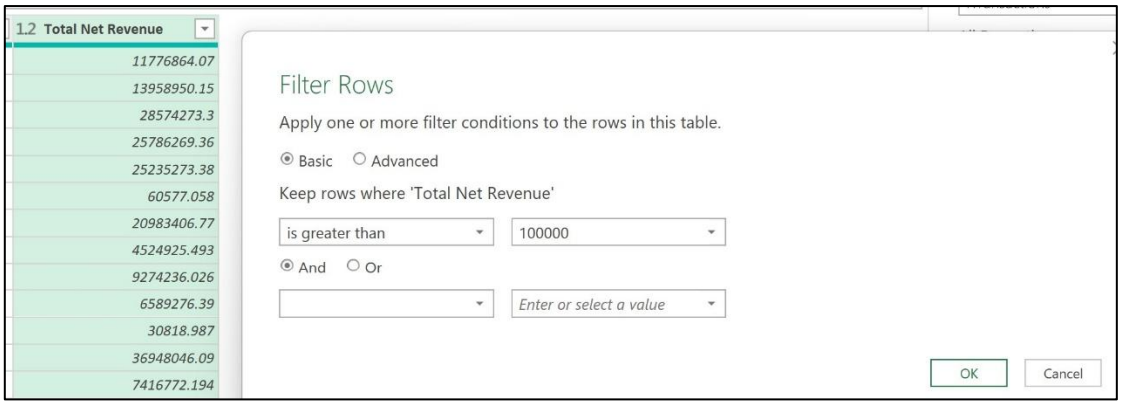
20) As shown below, the Table.Group function is used to complete a Group By action in M Code:

```
= Table.Group(#"Inserted Multiplication", {"Product"}, {{"Total Net Revenue", each List.Sum([Net Revenue]),
    type number}})
```

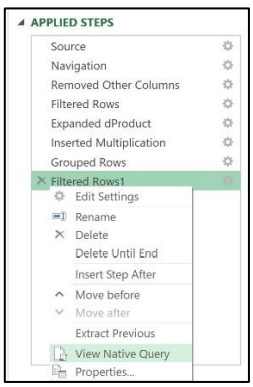
21) As shown below, when you right-click the step, the View Native Query option is visible and therefore the step is being sent back to the SQL Server Database for evaluation.



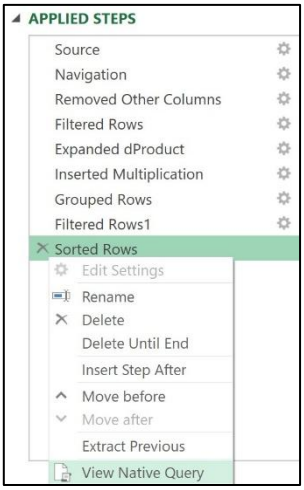
22) As shown below, for the next query step, filter the Total Net Revenue column so that only records where the aggregated amount is great than 100,000:



23) As shown below, when you right-click the step, the View Native Query option is visible and therefore the step is being sent back to the SQL Server Database for evaluation.



24) As shown below, after sorting the Product column from A to Z for the last query step, when you right-click the step, the View Native Query option is visible and therefore the step is being sent back to the SQL Server Database for evaluation.

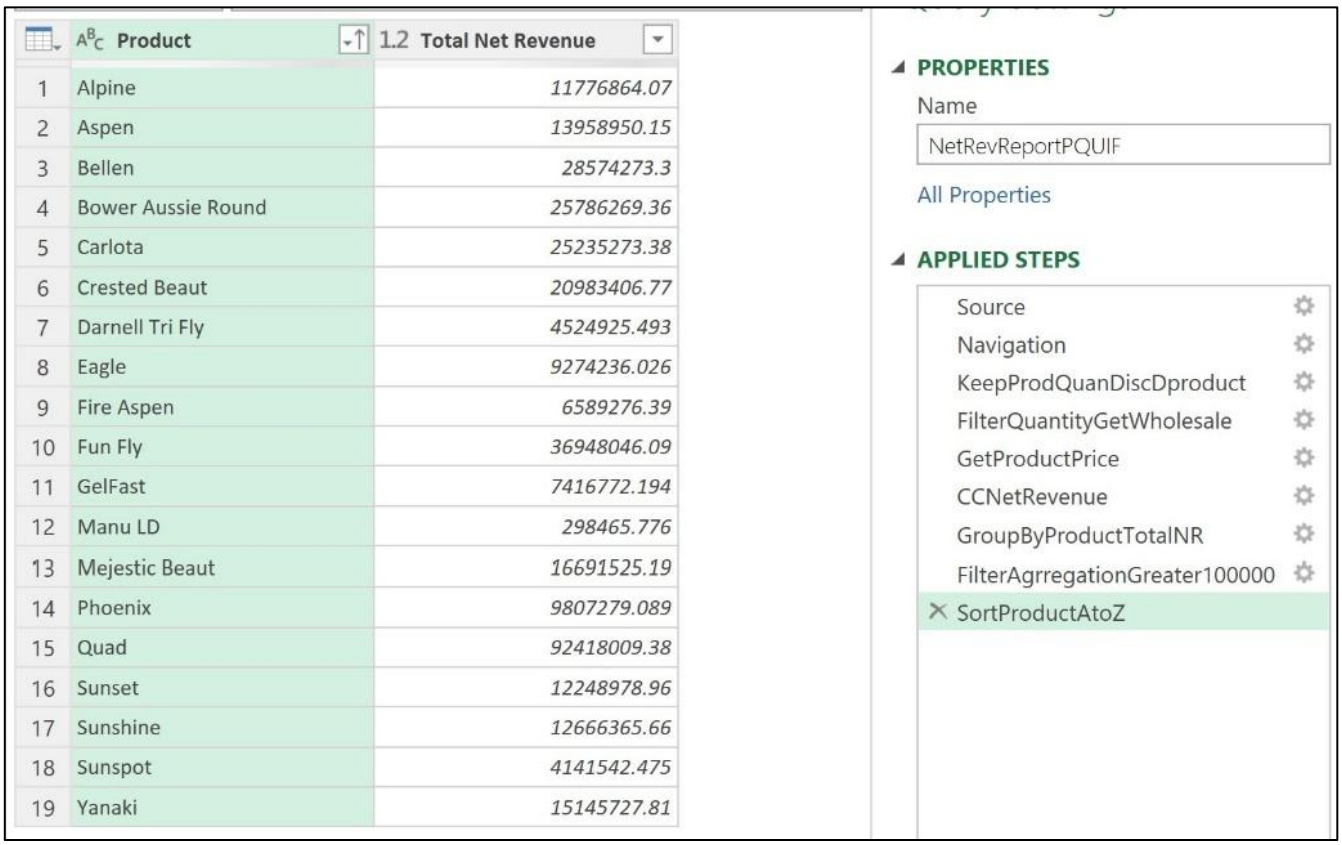


25) As shown below, from the last query step, if you right-click that step and click the View Native Query option, you can see the SQL code that is sent back to the SQL Server Database.

Native Query

```
select [_].[Product2] as [Product],
    [_].[Total Net Revenue] as [Total Net Revenue]
from
(
    select [_].[Product2],
        [_].[Total Net Revenue]
    from
    (
        select [_].[Product2],
            [_].[Total Net Revenue]
        from
        (
            select [rows].[Product2] as [Product2],
                sum([rows].[Net Revenue]) as [Total Net Revenue]
            from
            (
                select [_].[Product2] as [Product2],
                    [_].[Quantity] as [Quantity],
                    [_].[RevenueDiscount] as [RevenueDiscount],
                    [_].[RetailPrice] as [RetailPrice],
                    case
                        when ([_].[RetailPrice] is null and 1 - [_].[RevenueDiscount] is null) and [_].[Quantity] is null
                        then null
                        else ((case
                            when [_].[RetailPrice] is null
                            then 1
                            else [_].[RetailPrice]
                        end) * (case
                            when 1 - [_].[RevenueDiscount] is null
                            then 1
                            else 1 - [_].[RevenueDiscount]
                        end)) * (case
                            when [_].[Quantity] is null
                            then 1
                            else [_].[Quantity]
                        end)
                    end as [Net Revenue]
                from
                (
                    select [$Outer].[Product2],
                        [$Outer].[Quantity],
                        [$Outer].[RevenueDiscount],
                        [$Inner].[RetailPrice]
                    from
                    (
                        select [_].[Product] as [Product2],
                            [_].[Quantity] as [Quantity],
                            [_].[RevenueDiscount] as [RevenueDiscount]
                        from
                        (
                            select [Product],
                                [Quantity],
                                [RevenueDiscount]
                            from [dbo].[fTransactions] as [$Table]
                        ) as [_]
                        where [_].[Quantity] >= 100
                    ) as [$Outer]
                    left outer join [dbo].[dProduct] as [$Inner] on ([$Outer].[Product2] = [$Inner].[Product])
                ) as [_]
            ) as [rows]
            group by [Product2]
        ) as [_]
        where [_].[Total Net Revenue] > 100000
    ) as [_]
) as [_]
order by [_].[Product2]
```

26) The finished Excel query #1 is shown below:



	Product	Total Net Revenue
1	Alpine	11776864.07
2	Aspen	13958950.15
3	Bellen	28574273.3
4	Bower Aussie Round	25786269.36
5	Carlota	25235273.38
6	Crested Beaut	20983406.77
7	Darnell Tri Fly	4524925.493
8	Eagle	9274236.026
9	Fire Aspen	6589276.39
10	Fun Fly	36948046.09
11	GelFast	7416772.194
12	Manu LD	298465.776
13	Mejestic Beaut	16691525.19
14	Phoenix	9807279.089
15	Quad	92418009.38
16	Sunset	12248978.96
17	Sunshine	12666365.66
18	Sunspot	4141542.475
19	Yanaki	15145727.81

2) The second Query in Excel

For second query in Excel, the goal is to copy and paste SQL code that can accomplish the same goals as in the first query. Connecting to the SQL Server Database boomerang database is the same as in the first query, except in the SQL Server Database dialog box, in the SQL statement text box, you copy and paste the provide SQL Code:

```
SELECT dProduct.Product, SUM(Quantity*(1-RevenueDiscount)*RetailPrice) AS NetRevenue

FROM fTransactions

JOIN dProduct

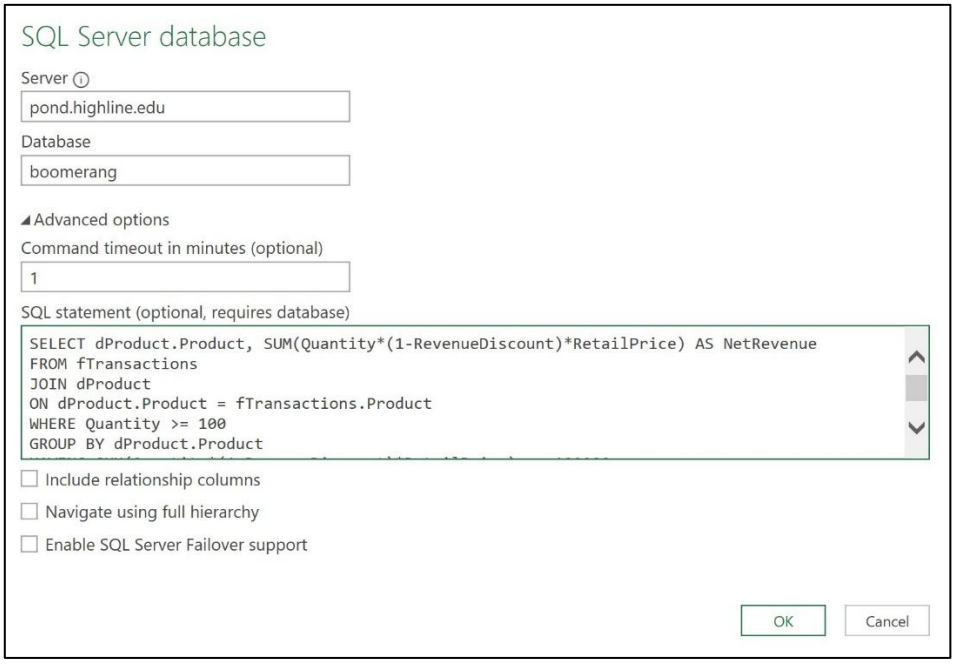
ON dProduct.Product = fTransactions.Product

WHERE Quantity >= 100

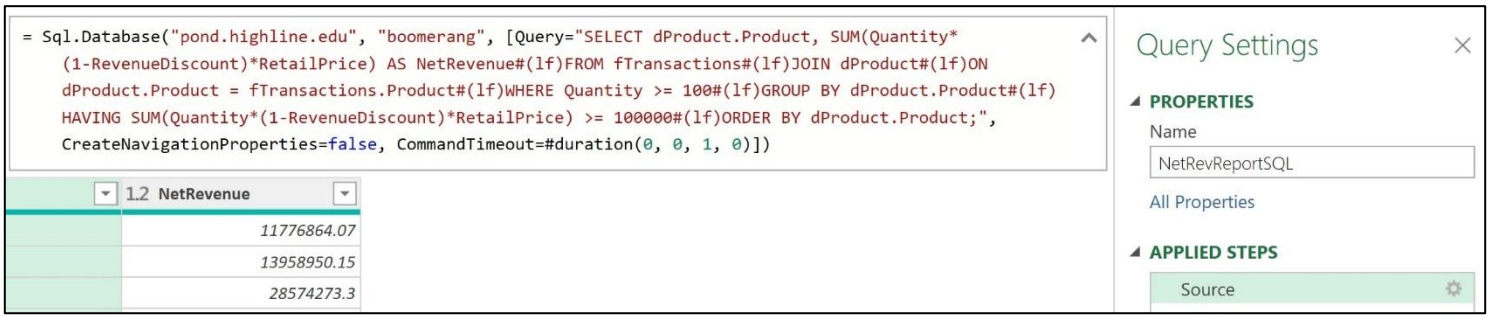
GROUP BY dProduct.Product

HAVING SUM(Quantity*(1-RevenueDiscount)*RetailPrice) >= 100000

ORDER BY dProduct.Product;
```

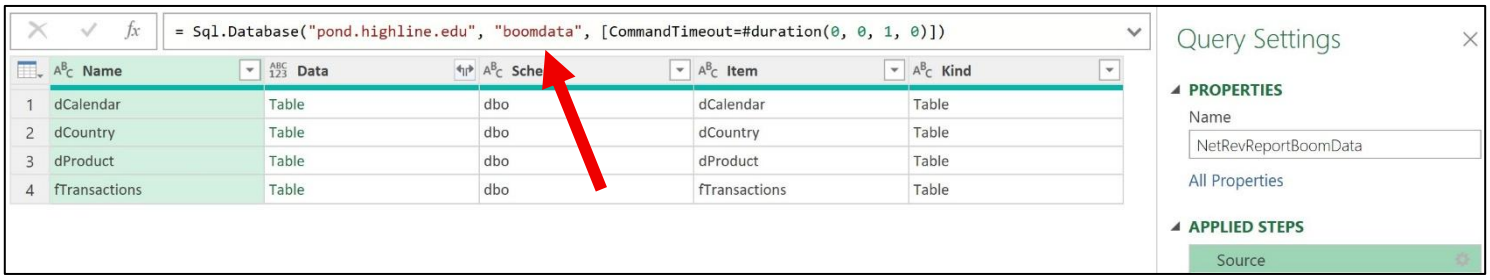


2) After clicking OK in the SQL Server Database dialog box, as shown below, there is only one Source query step and in the formula bar you can see the Sql.Database function and the SQL code:

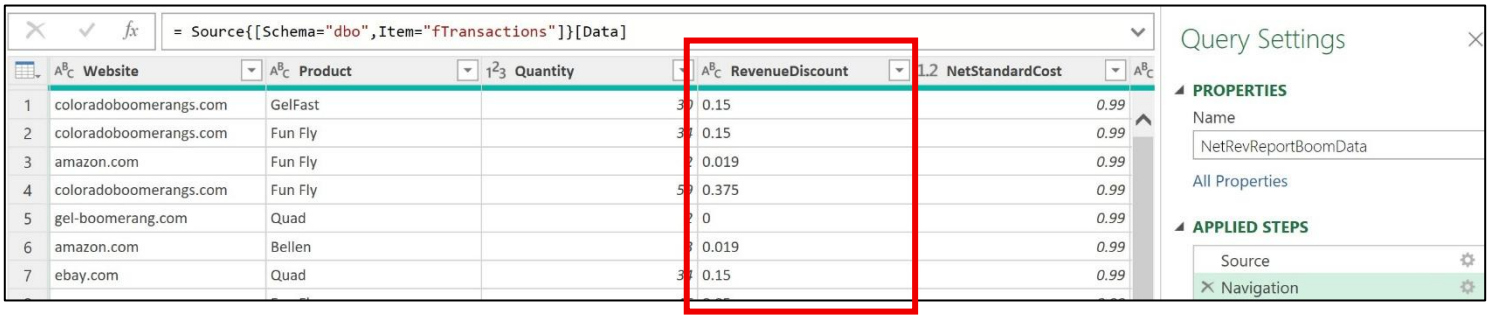


3) The third query in Excel

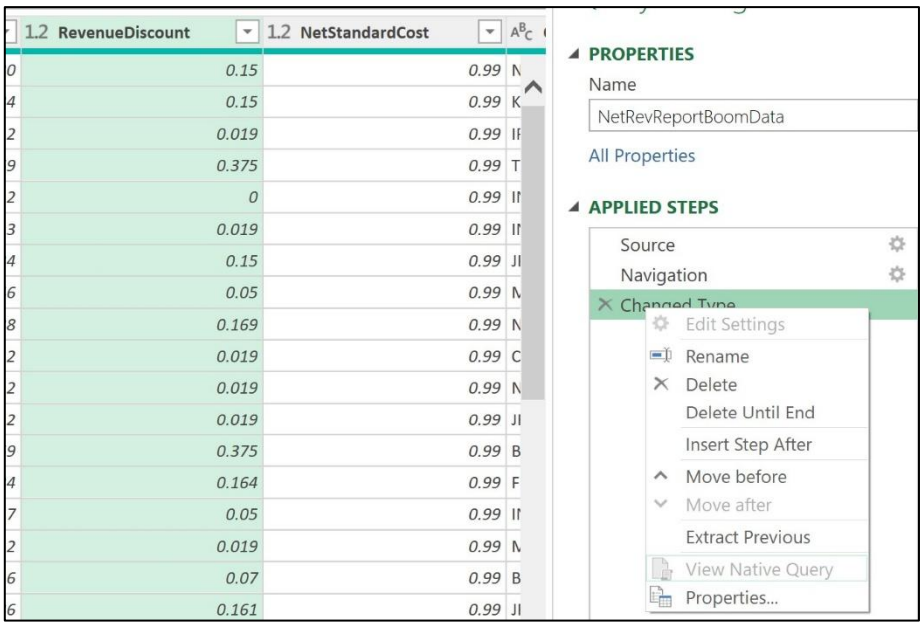
1) The only difference in the third query is that we connect to the boomdata database (as shown below) which is a database that is the same as the boomerang database except the amount are a few pennies different and the RevenueDiscount column has the wrong data type.



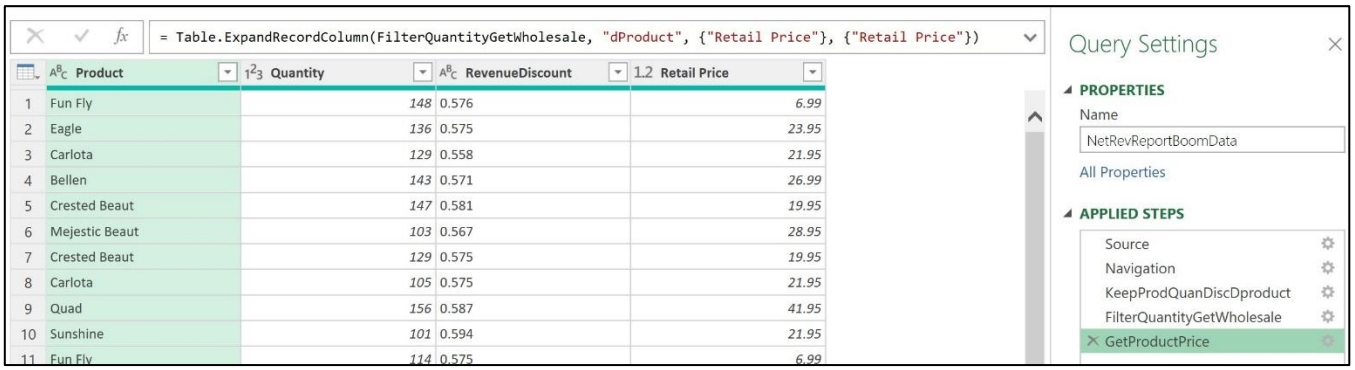
2) As shown below, the RevenueDiscount column has the wrong data type has a Text Data Type, which will not work for a number in Power Query. If you try to use a text number as a number in Power Query, like multiplying times a number, you will get an error.



3) As shown below, if you add a Decimal Data Type as the third step in the query and then right-click the step, you can see that the View Native Query option is greyed out – which means the step and all subsequent steps will not be sent back to the SQL Database through Query Folding.

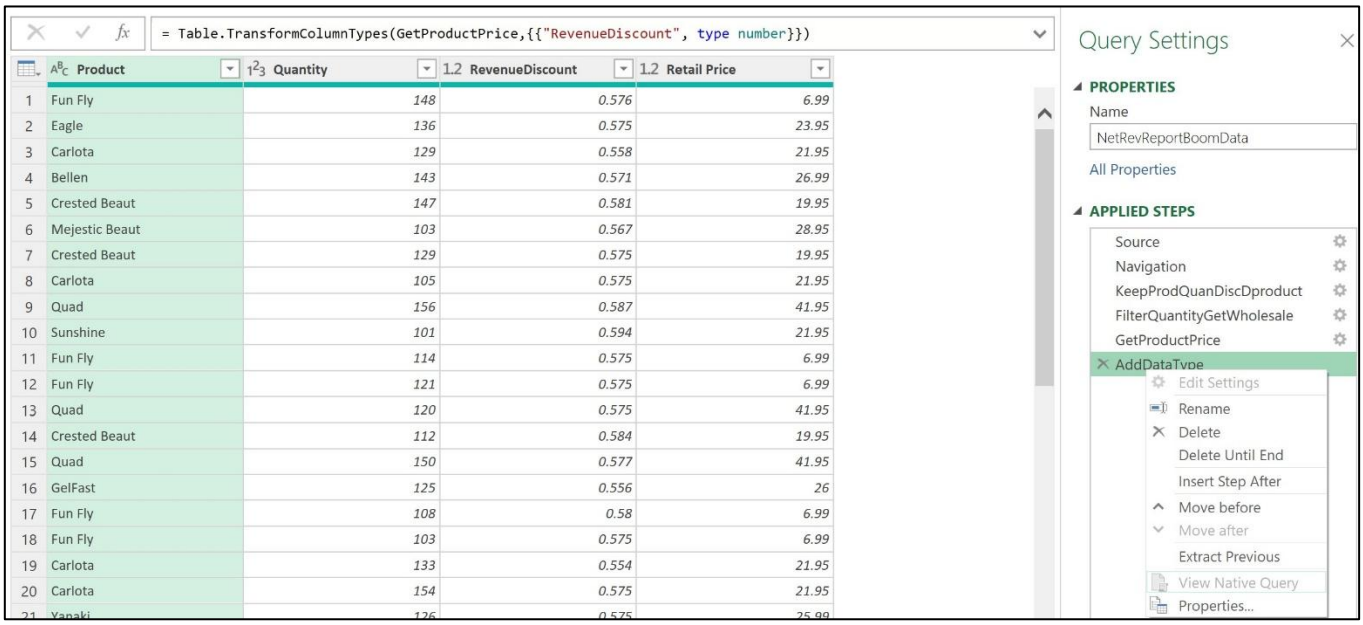


4) We therefore do as many query steps as we can until we MUST add the data type. In the picture below, we created the three steps to remove columns, remove rows, and expand the dProduct column to get the product price, but because the next step is calculating Net Revenue, we must have the RevenueDiscount as a number.



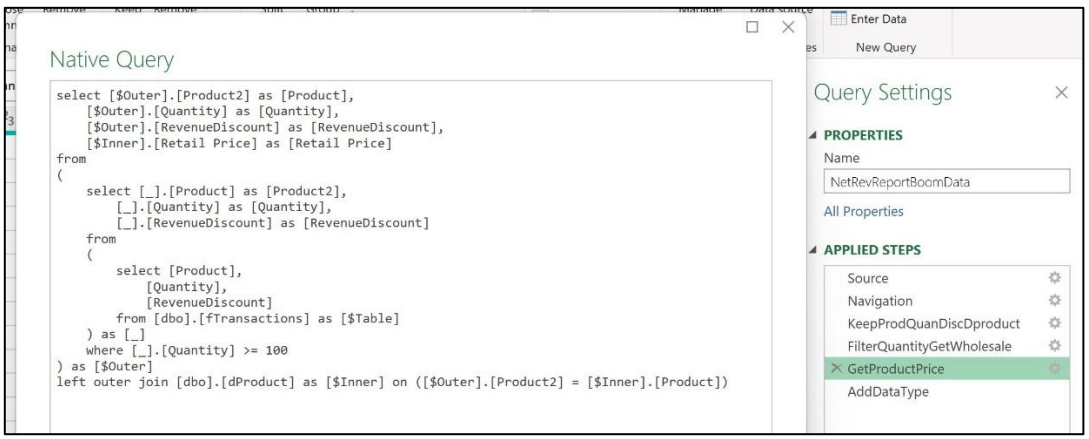
	Product	Quantity	RevenueDiscount	Retail Price
1	Fun Fly	148	0.576	6.99
2	Eagle	136	0.575	23.95
3	Carlota	129	0.558	21.95
4	Bellen	143	0.571	26.99
5	Crested Beaut	147	0.581	19.95
6	Mejestic Beaut	103	0.567	28.95
7	Crested Beaut	129	0.575	19.95
8	Carlota	105	0.575	21.95
9	Quad	156	0.587	41.95
10	Sunshine	101	0.594	21.95
11	Fun Fly	114	0.575	6.99

5) As shown below, we add the Data Type step, but the View Native Query option is greyed out. This means all the remaining steps will be greyed out also.

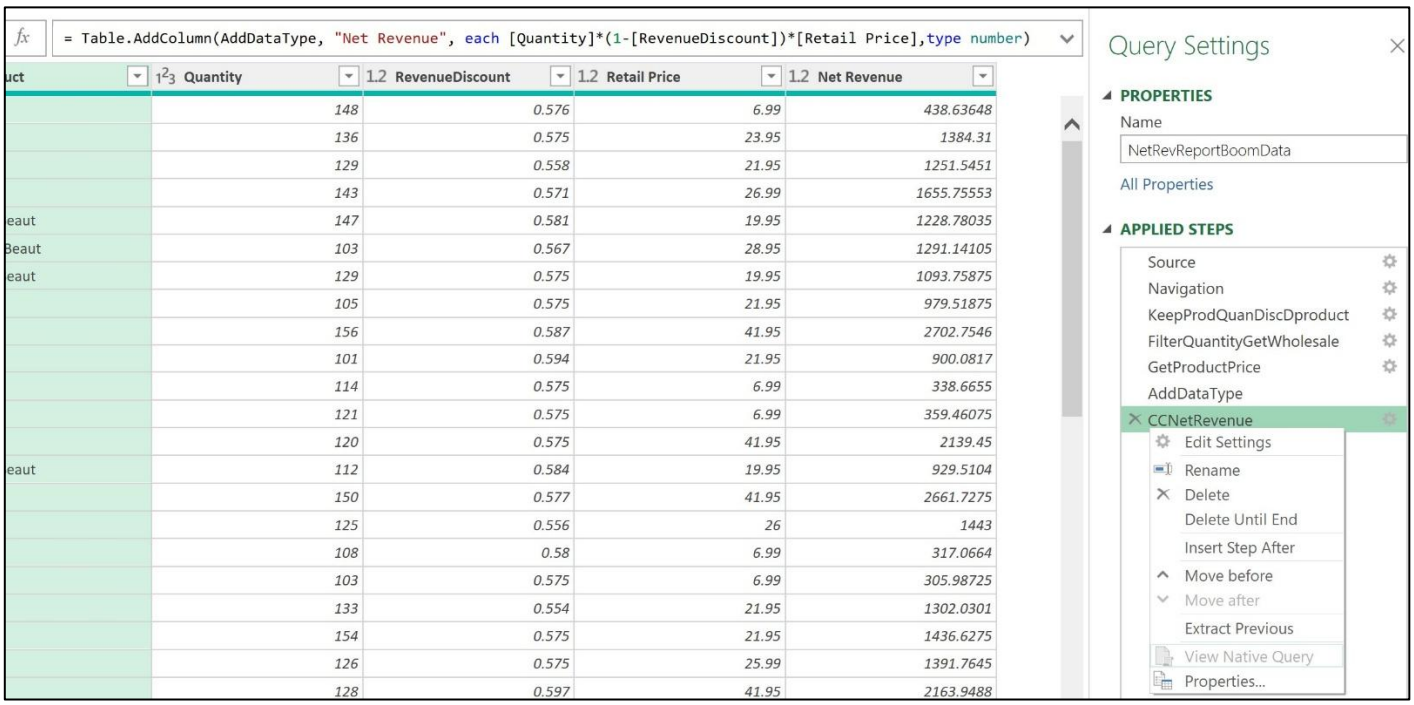


	Product	Quantity	RevenueDiscount	Retail Price
1	Fun Fly	148	0.576	6.99
2	Eagle	136	0.575	23.95
3	Carlota	129	0.558	21.95
4	Bellen	143	0.571	26.99
5	Crested Beaut	147	0.581	19.95
6	Mejestic Beaut	103	0.567	28.95
7	Crested Beaut	129	0.575	19.95
8	Carlota	105	0.575	21.95
9	Quad	156	0.587	41.95
10	Sunshine	101	0.594	21.95
11	Fun Fly	114	0.575	6.99
12	Fun Fly	121	0.575	6.99
13	Quad	120	0.575	41.95
14	Crested Beaut	112	0.584	19.95
15	Quad	150	0.577	41.95
16	GelFast	125	0.556	26
17	Fun Fly	108	0.58	6.99
18	Fun Fly	103	0.575	6.99
19	Carlota	133	0.554	21.95
20	Carlota	154	0.575	21.95
21	Yanaki	126	0.575	25.99

6) If we right-click the previous step and select View Native Query option, as shown below, this is the SQL code that is sent back to the SQL Server Database.

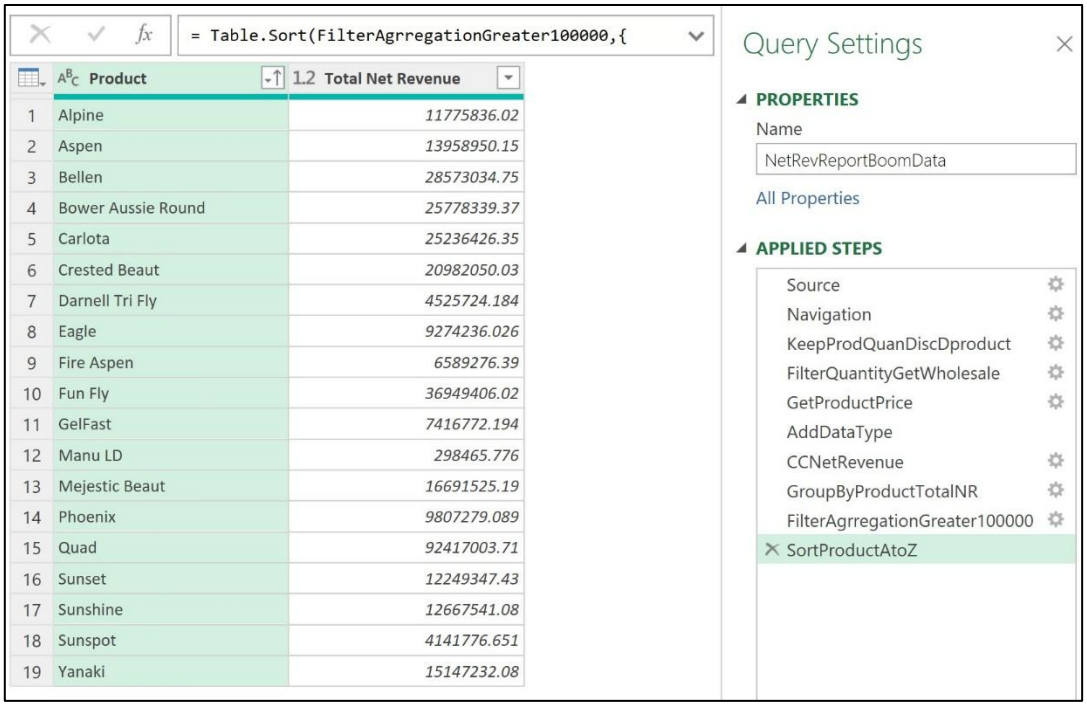


7) After creating the Net Revenue column, this step shows a greyed out View Native Query option also.



uct	Quantity	RevenueDiscount	Retail Price	Net Revenue
	148	0.576	6.99	438.63648
	136	0.575	23.95	1384.31
	129	0.558	21.95	1251.5451
	143	0.571	26.99	1655.75553
eaut	147	0.581	19.95	1228.78035
Beaut	103	0.567	28.95	1291.14105
eaut	129	0.575	19.95	1093.75875
	105	0.575	21.95	979.51875
	156	0.587	41.95	2702.7546
	101	0.594	21.95	900.0817
	114	0.575	6.99	338.6655
	121	0.575	6.99	359.46075
	120	0.575	41.95	2139.45
eaut	112	0.584	19.95	929.5104
	150	0.577	41.95	2661.7275
	125	0.556	26	1443
	108	0.58	6.99	317.0664
	103	0.575	6.99	305.98725
	133	0.554	21.95	1302.0301
	154	0.575	21.95	1436.6275
	126	0.575	25.99	1391.7645
	128	0.597	41.95	2163.9488

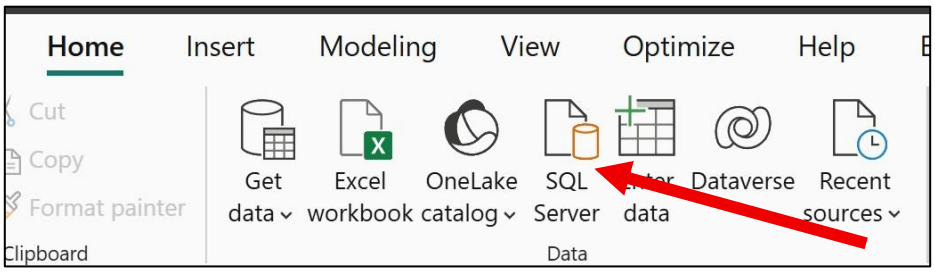
2) After creating the remaining query steps, as shown below, we get our final report. The amounts are a few dollars off because when I built the database and changed a few numbers. But the report is the same in all other regards.



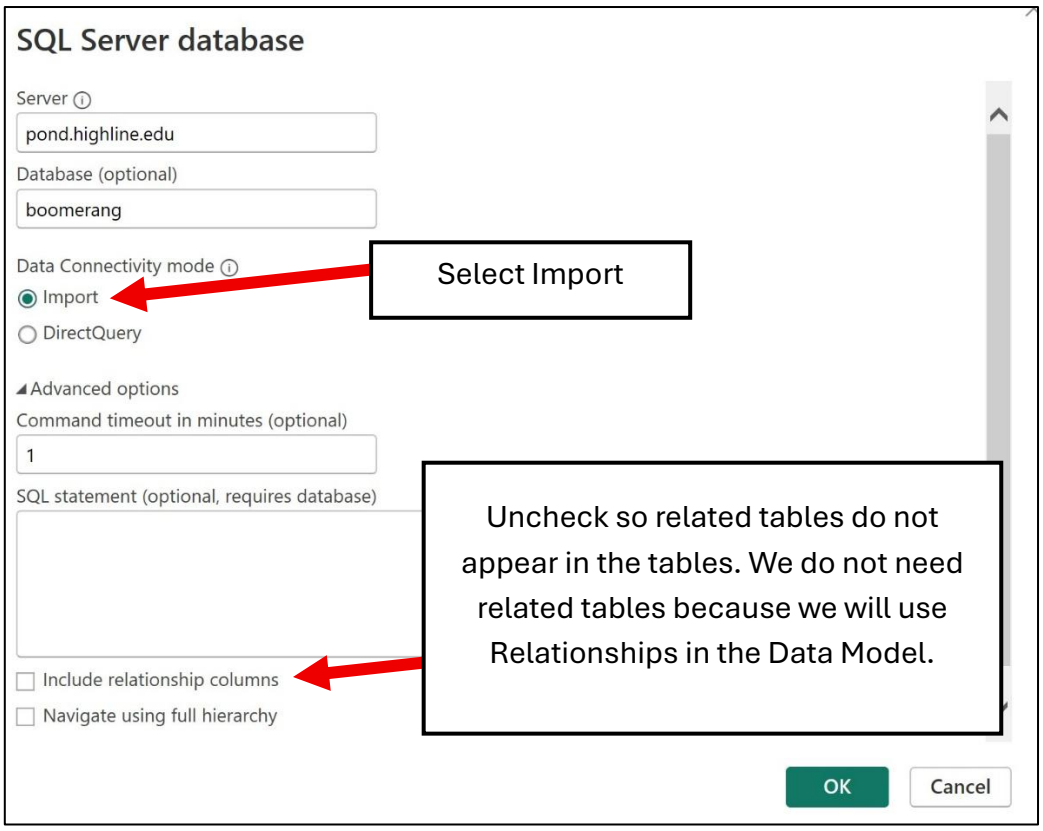
	Product	Total Net Revenue
1	Alpine	11775836.02
2	Aspen	13958950.15
3	Bellen	28573034.75
4	Bower Aussie Round	25778339.37
5	Carlota	25236426.35
6	Crested Beaut	20982050.03
7	Darnell Tri Fly	4525724.184
8	Eagle	9274236.026
9	Fire Aspen	6589276.39
10	Fun Fly	36949406.02
11	GelFast	7416772.194
12	Manu LD	298465.776
13	Mejestic Beaut	16691525.19
14	Phoenix	9807279.089
15	Quad	92417003.71
16	Sunset	12249347.43
17	Sunshine	12667541.08
18	Sunspot	4141776.651
19	Yanaki	15147232.08

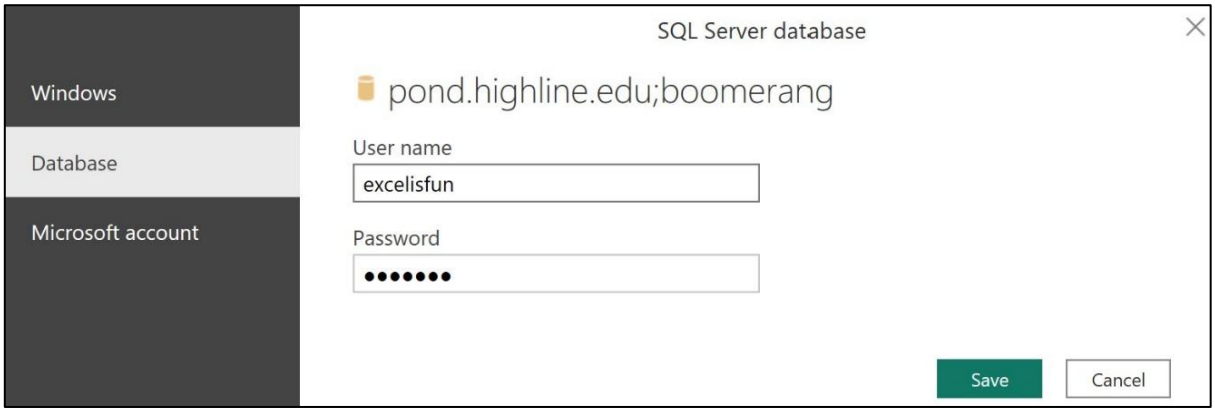
4) The fourth query in Power BI Desktop

1) The goal of this project is to connect to the SQL database and import three tables into the Columnar Database in the Data Model. In Power BI Desktop, the SQL Server button is in the Home tab, Data group:



2) The dialog boxes to enter credentials look the same as in Excel except for there is an option to import or connect directly to the SQL Server Database:



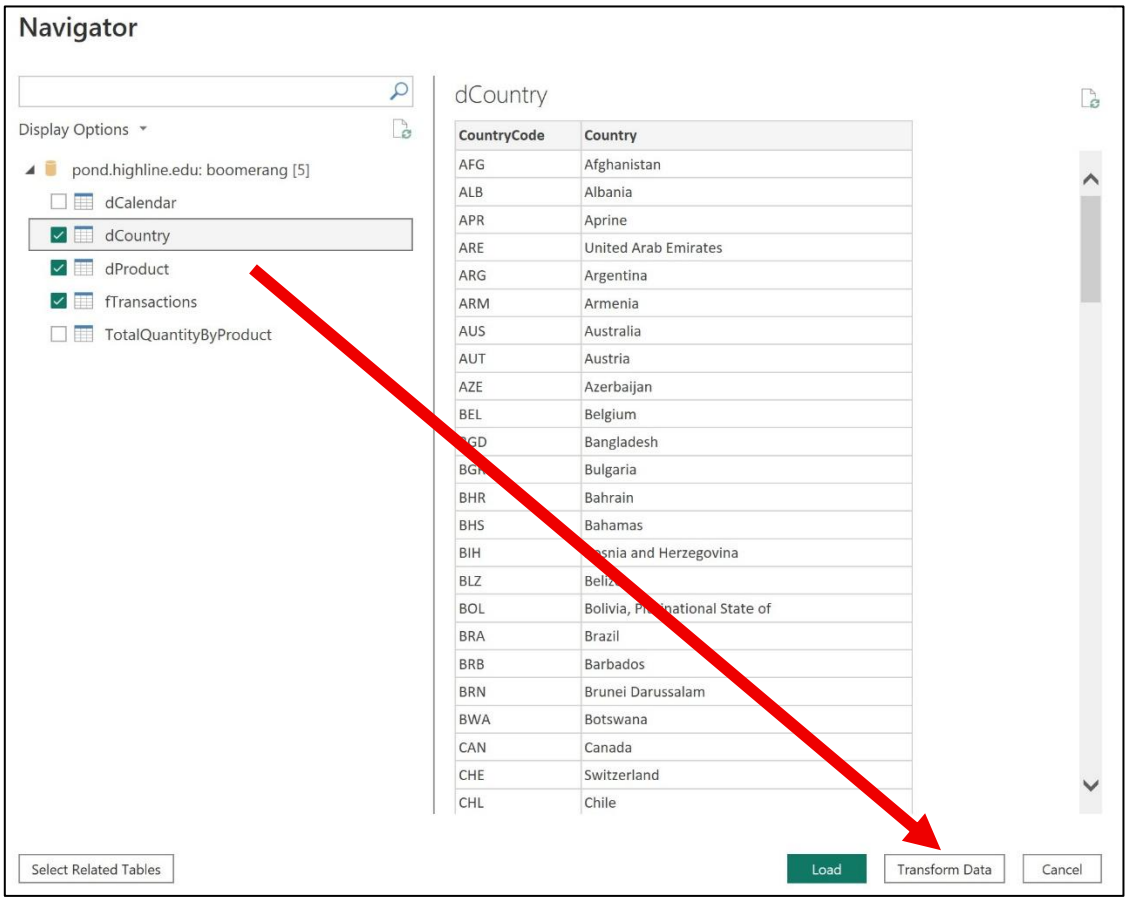


Notes for Direct Query in Power BI

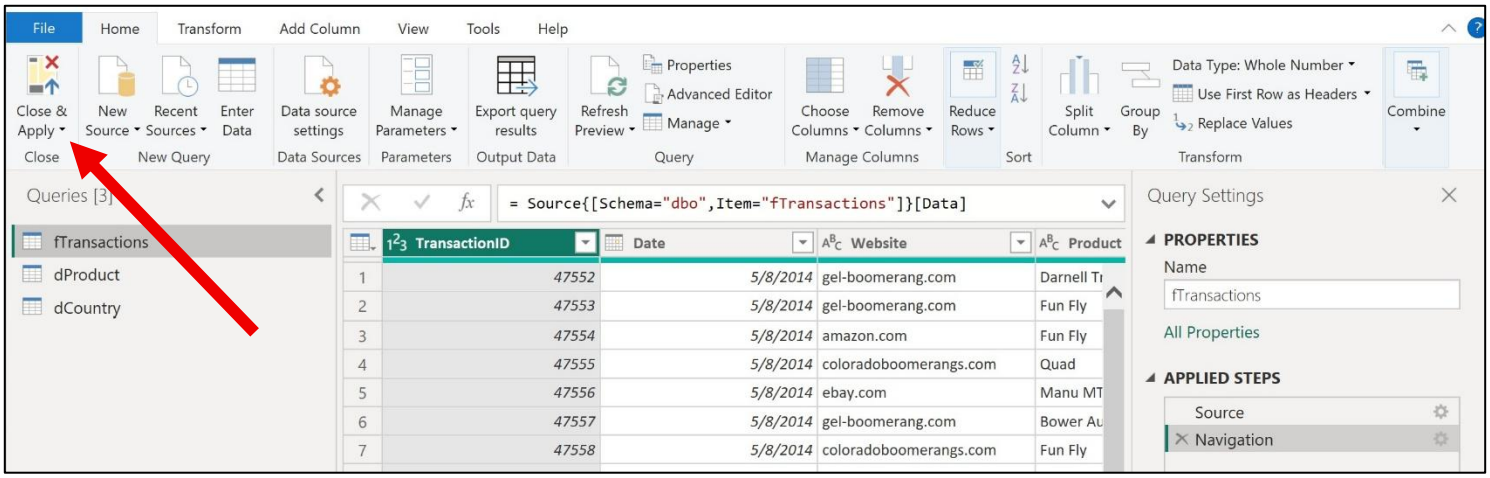
Information about SQL Server Database Direct Query in Power BI:

https://learn.microsoft.com/en-us/power-bi/connect-data/desktop-use-directquery

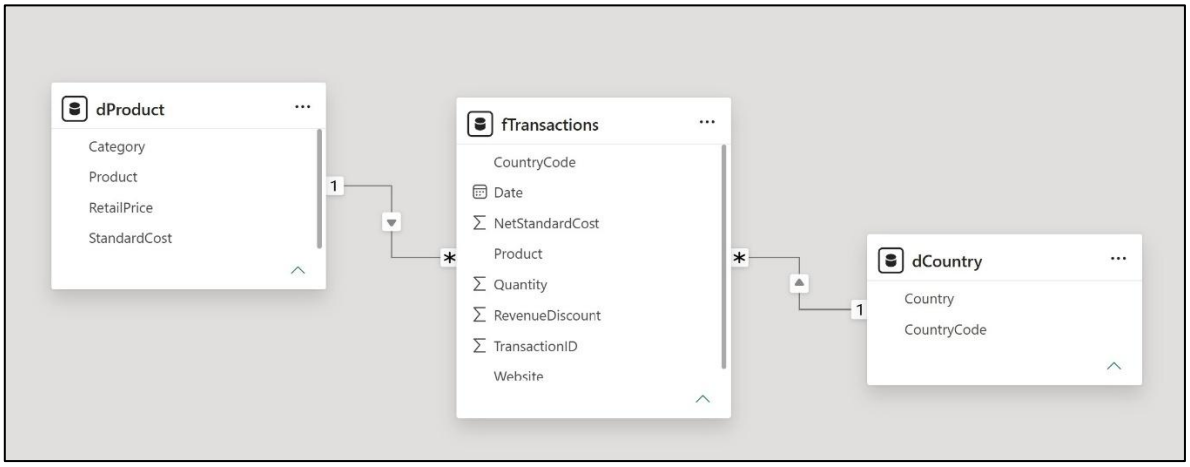
3) In the Navigator dialog box, select the three tables: fTransactions, dProduct and dCountry and then click the Transform Data button:



4) In the Power Query Editor, click the Close & Apply button to load the tables to the Columnar Database in the Data Model:



5) The finished tables with relationships can be viewed in Model View in Power BI Desktop:



Columnar Database

This is the database that is behind the scenes in the Data Model in Excel PowerPivot, Power BI Desktop, Power BI online, and Fabric. It compresses data and works directly with DAX formulas to calculate quickly in Big Data. The picture below shows the compression process:

A **Columnar Database** is a special type of behind-the-scenes data storage location that is loaded into random access memory (RAM) when you open an Excel or Power BI file. A columnar database compresses the data into a smaller size and stores the data one column at a time (hence the "columnar" name). For each column of data that is stored, the columnar database stores a unique list of items. The more unique items in any given column, the larger the size of the stored column in the database. The database also stores a type of mapping that allows it to reconstruct records of data from the columns of unique lists that are stored in the columnar database. This database was specifically designed by Microsoft to work efficiently with big data for data analysis and is part of the Data Model tool.

Original Table with 606 records:

Date	ProductID	SalesRepID	Units
3/19/2021	2	4	80
4/8/2021	2	4	5
4/12/2021	2	4	88
4/12/2021	2	4	70
4/22/2021	2	4	6
5/12/2021	2	4	5
5/24/2021	2	4	1
6/11/2021	2	4	92
6/11/2021	2	4	91
6/17/2021	2	4	209
7/5/2021	2	4	91
7/18/2021	2	4	66
7/20/2021	2	4	110
7/24/2021	2	4	4
8/6/2021	2	4	94
8/18/2021	2	4	2
9/10/2021	2	4	57
10/2/2021	2	4	4
10/2/2021	2	4	1
10/7/2021	2	4	86
10/7/2021	2	4	90
10/18/2021	2	4	72
10/25/2021	2	4	69

How Columnar Database compresses tables of data:

Records in original table:	606
# Columns:	4
Total cells with data: 606*4 =	2424

Total cells in columnar database: 426 + 4 + 4 + 187 =	621

Unique counts for each column:

Count	Count	Count	Count
426	4	4	187

Date	ProductID	SalesRepID	Units
3/19/2021	2	4	80
4/8/2021	3	2	5
4/12/2021	4	3	88
4/22/2021	1	1	70
5/12/2021			6
5/24/2021			1
6/11/2021			92
6/17/2021			91
7/5/2021			209
7/18/2021			66
7/20/2021			110