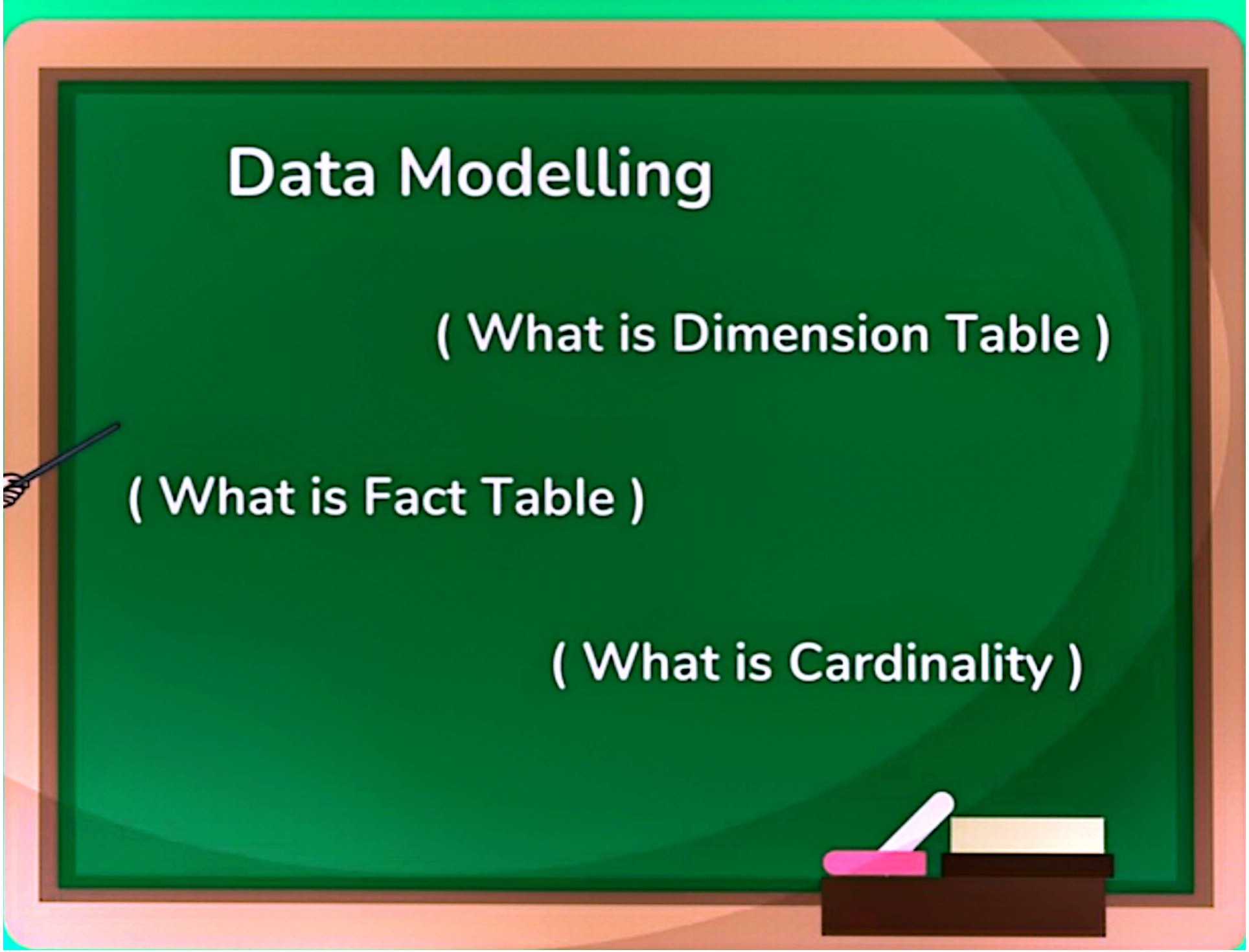
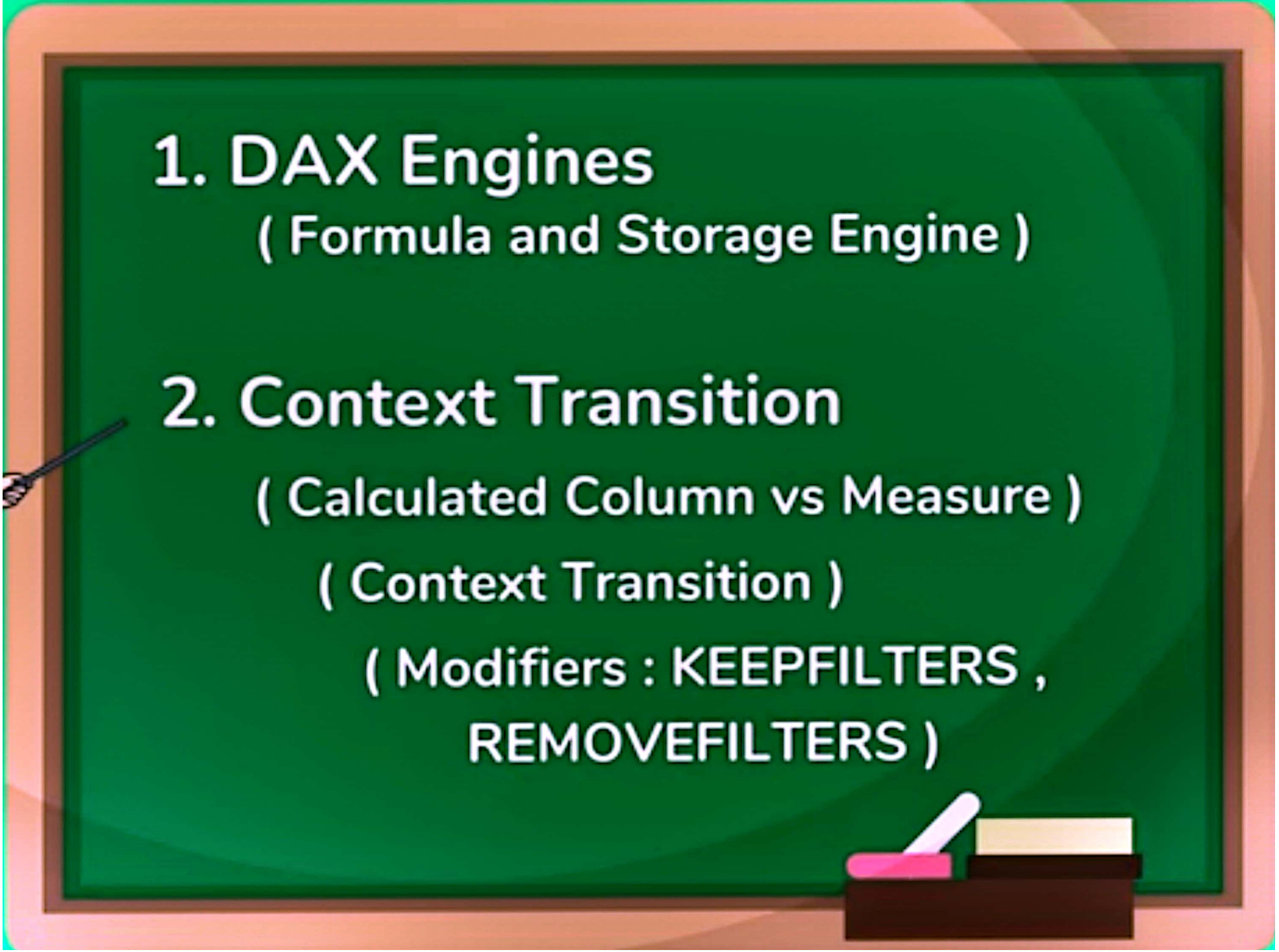
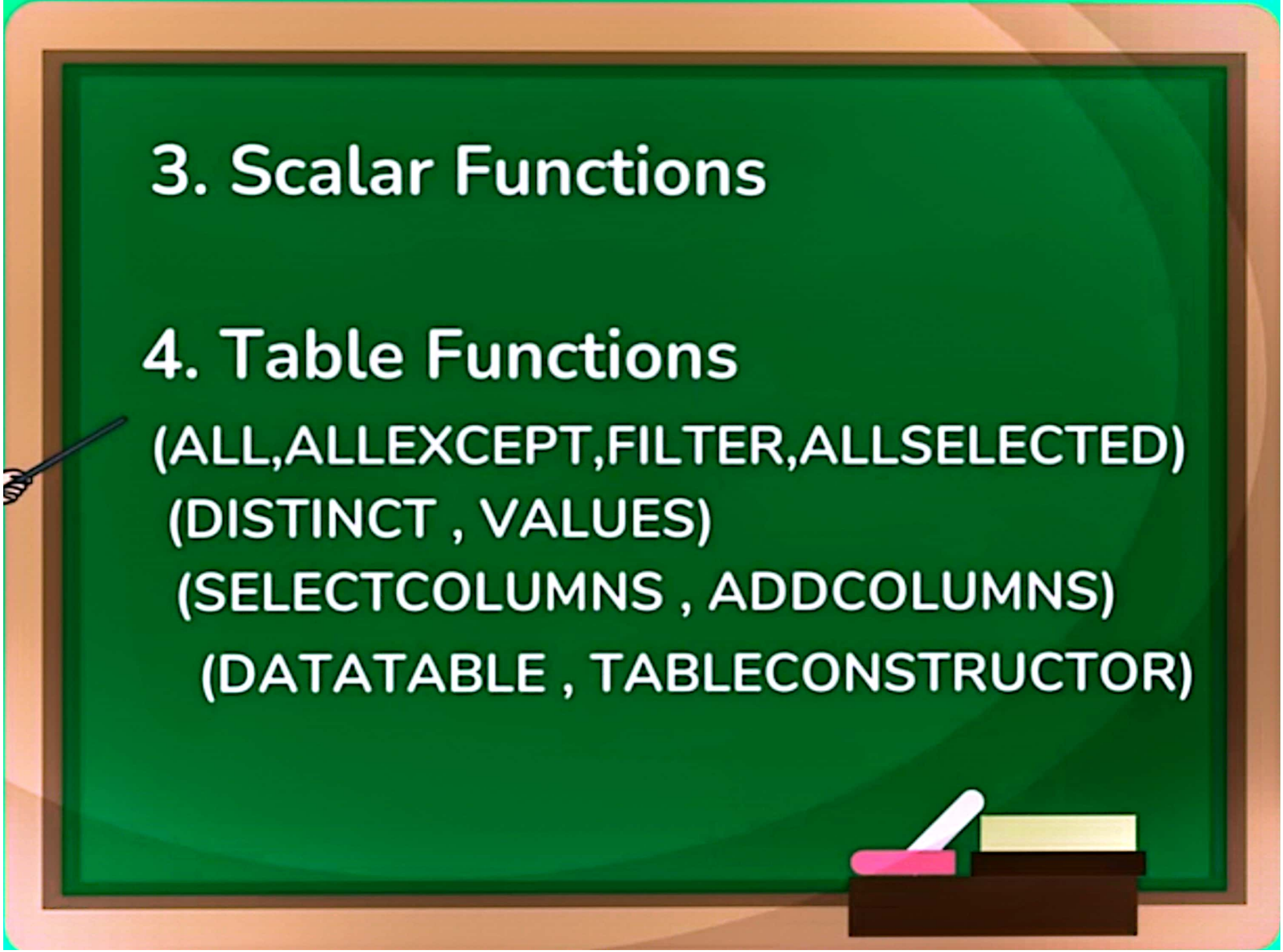
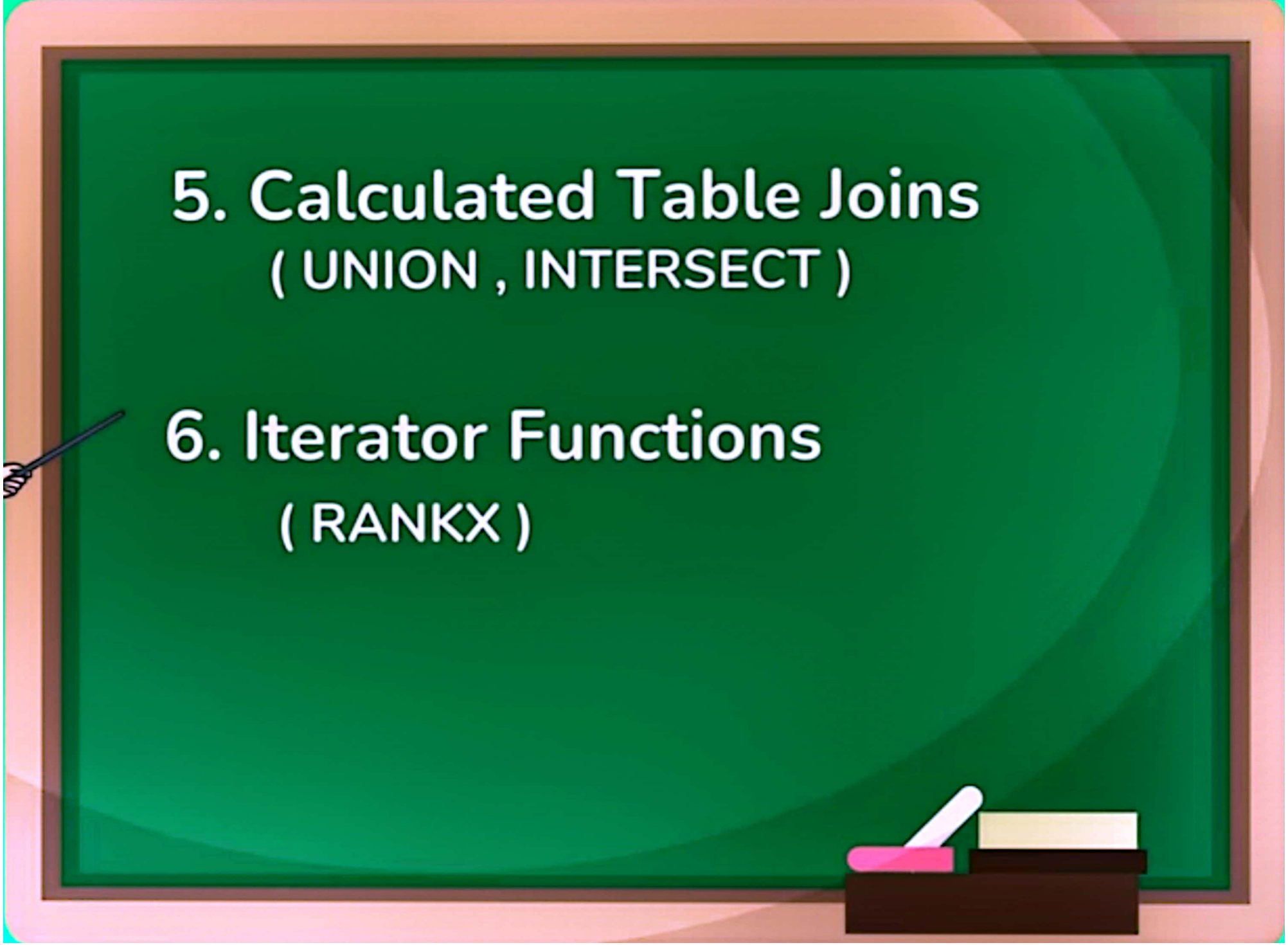
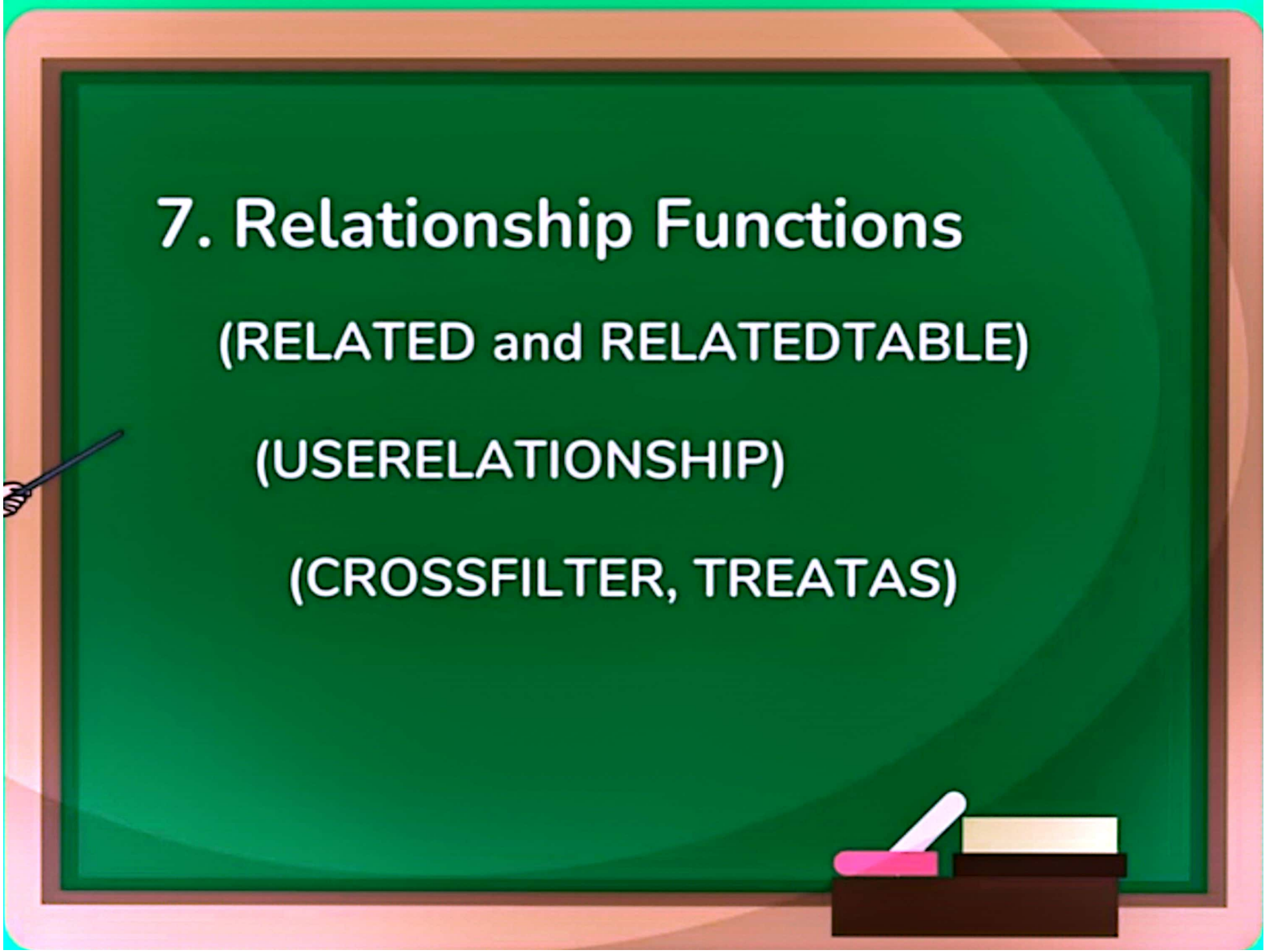
DAX COURSE ROADMAP

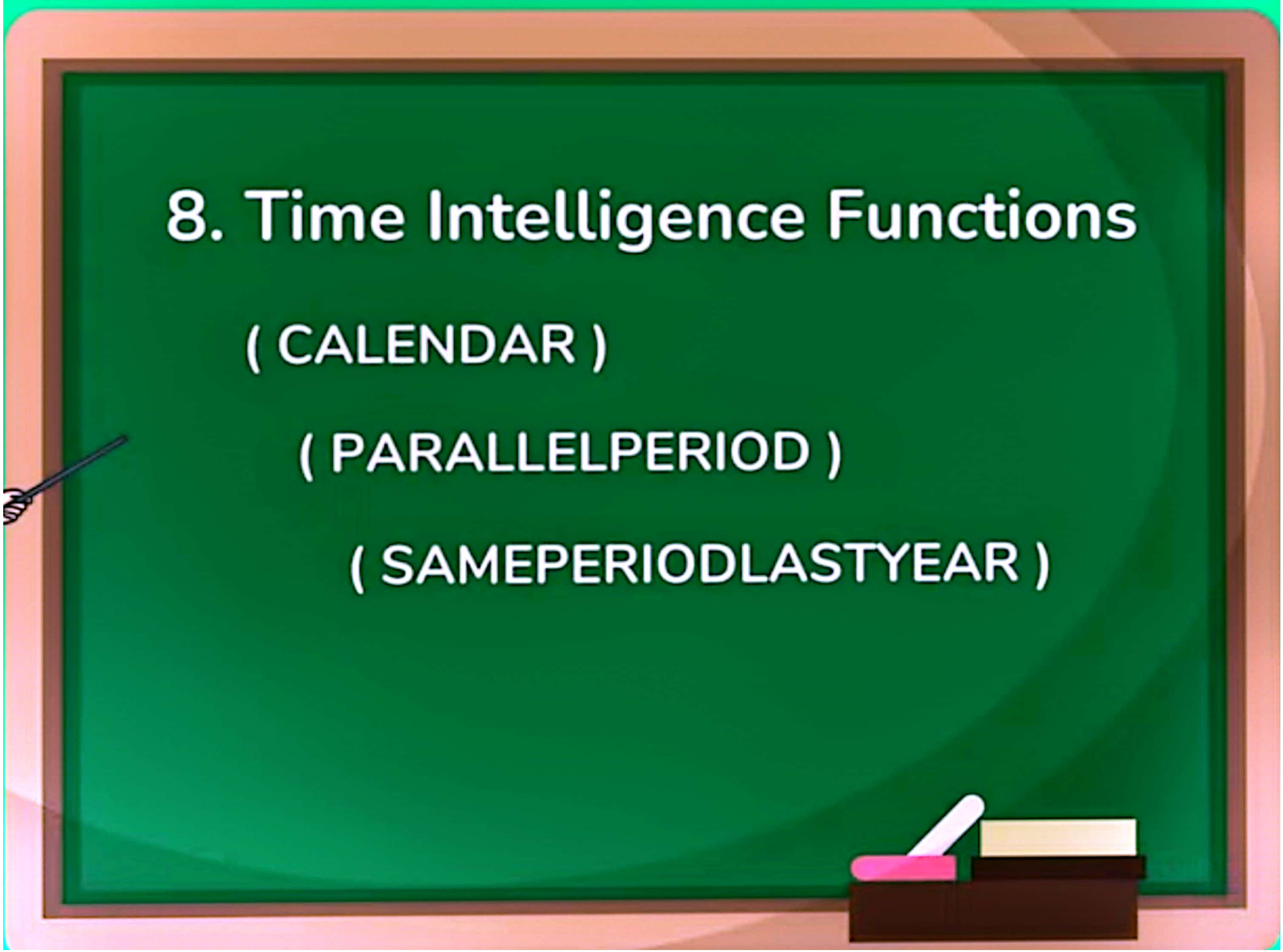

Data Modelling
(What is Dimension Table)
(What is Fact Table)
(What is Cardinality)


1. DAX Engines
(Formula and Storage Engine)
2. Context Transition
(Calculated Column vs Measure)
(Context Transition)
(Modifiers : KEEPFILTERS ,
REMOVEFILTERS)


3. Scalar Functions
4. Table Functions
(ALL,ALLEXCEPT,FILTER,ALLSELECTED)
(DISTINCT , VALUES)
(SELECTCOLUMNS , ADDCOLUMNS)
(DATATABLE , TABLECONSTRUCTOR)


5. Calculated Table Joins
(UNION , INTERSECT)
6. Iterator Functions
(RANKX)


7. Relationship Functions
(RELATED and RELATEDTABLE)
(USERELATIONSHIP)
(CROSSFILTER, TREATAS)

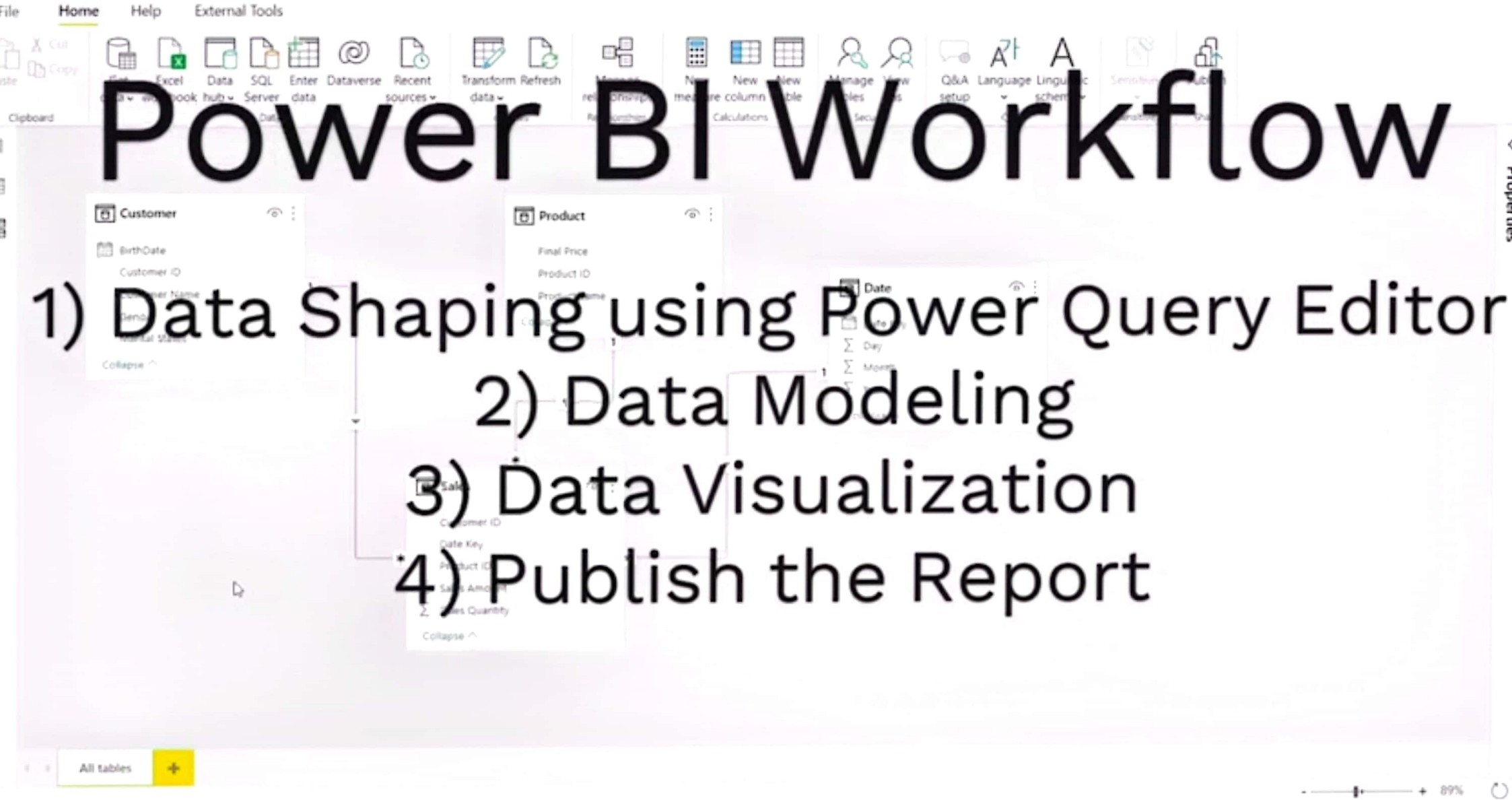

8. Time Intelligence Functions
(CALENDAR)
(PARALLELPERIOD)
(SAMEPERIODLASTYEAR)

What is DAX ??

Data Analysis
Expression

- Data Manipulation
- Data Aggregation

To Derive the
Insights


File
Home
Help
External Tools
Power BI Workflow
1) Data Shaping using Power Query Editor
2) Data Modeling
3) Data Visualization
4) Publish the Report
Customer
Product
Date
All tables

Data Modelling

Cardinality: Cardinality concept should always be kept in mind while doing data modeling.

- cardinality is basically the way over dimention and fact are related with each other.

- **one (1) to Many (*):** relationship between dimention and fact. example: one customer can purchase multiple items.
  one product can be sale multiple time

  one to many relationship is mostly used while during data modeling.

- one to one
- many to many:

* How dimention and fact relate with each other?
  - By primary key & Foreign key

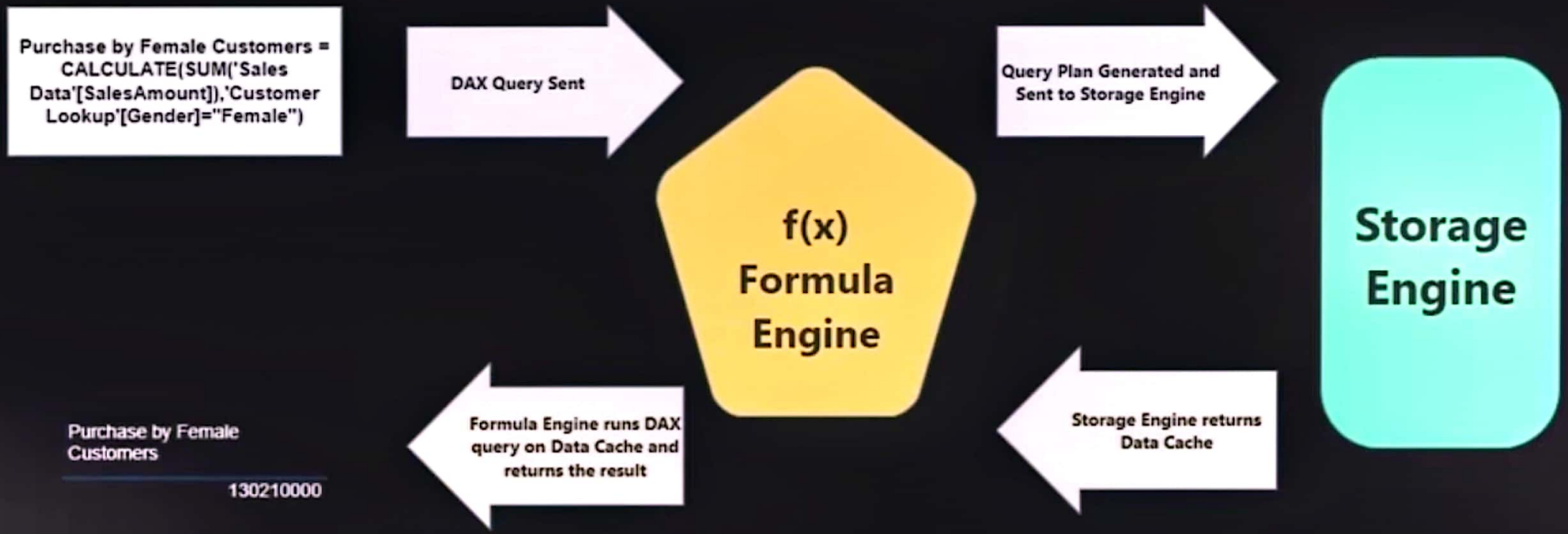
* Filter propogation / Filter Flow: Filter flow says how one table filtering another table.
  - Direction of Filter flow is from one side to many side (one-to-many) Relationship.

Dax engine: Dax Query execution. Dax query execution using formula & storage Engine.

Basicaly there one two types of engine work in Dax

1)- Formula engine: Query Recieve & interpret
2). Storage engine.: Compress and Encodes the data before storing.

DAX Query Evaluation



Dax Query Evolution process:(field)

When we write a Dax query this Dax query sent to (Formula Engine). When request pass to the formula engine. formula engine then interpret the request and for it then generates a query plan. The query plan created by the formula engine is passed to storage engine. from the formula engine. Query plan contains instruction about what the ~~data~~ data is needed and how the data can be captured. storage engine accept query plan instruction (necessary data). Return necessary data in the form of cash, to our formula engine. The formula engine will execute/ run ~~Dax query and~~ it a Dax query against the data cache. while query will execute against data cache out put will generate.

There are two types of storage engine.

1) Vertipaq : data connection import → all data table store in vertipaq
2) Direct query: data connection Direct-query: Data is note store in memory. Data will exist it own connecting SQL / Excel / one. we just query it.

SQL Server database

Server ⓘ

Database (optional)

Data Connectivity mode ⓘ

- Import
- DirectQuery

Advanced options

OK Cancel

Vertipaq Engine / Vertipaq storage engine:

- Vertipaq application for imported Data models only stored in - memory.

- How vertipaq storage engine store data in memory?
  - we know sql store data in memory row wise.
  vertipaq storage engine store data <u>column wise</u>.

- The advantage of vertipaq storage engine due to store data in column wise, its easy to emphasize on aggregated data, its easy to scan. (single scan).
  for multiple column operation it is easy to sequentially scan column. Evaluation of Dax query become easy for colum wise data storing (vertipaq Engine).

Region	Product Name	Sales(Cr)
North	Mobile	150000
North	Television	35000
North	AC	12000
North	Refrigerator	9000
West	Mobile	530000
West	Television	7000
West	AC	90000
West	Refrigerator	12000
South	Mobile	230000
South	Television	67000
South	AC	3400
South	Refrigerator	12000
East	Mobile	34000
East	Television	1200
East	AC	550
East	Refrigerator	5000

VertiPaq Engine stores data as individual columns

Storing values of Column adjacent to each other

Evaluation of DAX queries Will be very fast against the data stored In columnar format

Region
North
North
North
North
West
West
West
West
South
South
South
South
East

Product Name
Mobile
Television
AC
Refrigerator
Mobile
Television
AC
Refrigerator
Mobile
Television
AC
Refrigerator
Mobile

Sales(Cr)
150000
35000
12000
9000
530000
7000
90000
12000
230000
67000
3400
12000
34000

Scan 1

Scan Product Name Column and Identify rows where Product Name = "Mobile"

Region
North
North
North
North
West
West
West
West
South
South
South
South
East
East
East
East

Product Name	Sales(Cr)
Mobile	150000
Television	35000
AC	12000
Refrigerator	9000
Mobile	530000
Television	7000
AC	90000
Refrigerator	12000
Mobile	230000
Television	67000
AC	3400
Refrigerator	12000
Mobile	34000
Television	1200
AC	550
Refrigerator	5000

H
12000
230000
67000
3400
12000
34000
1200
550
5000

VertiPaq Engine stores data as individual columns

Storing values of Column adjacent to each other

Evaluation of DAX queries Will be very fast against the data stored In columnar format

Scan 2

Scan Sales Column corresponding to rows identified in Scan 1 and perform summation of values in Sales Column

	H
	12000
	230000
	67000
	3400
	12000
	34000
	1200
	550
	5000

VertiPaq Engine stores data as individual columns

Storing values of Column adjacent to each other

Evaluation of DAX queries Will be very fast against the data stored In columnar format

Region	Product Name	Sales(Cr)
North	Mobile	150000
North	Television	35000
North	AC	12000
North	Refrigerator	9000
West	Mobile	530000
West	Television	7000
West	AC	90000
West	Refrigerator	12000
South	Mobile	230000
South	Television	67000
South	AC	3400
South	Refrigerator	12000
East	Mobile	34000
East	Television	1200
East	AC	550
East	Refrigerator	5000

* In vertipaq storage engine data is store in memory (ram). If we do not store our data without compress form our ram consume maximum. thats why vertipaq storage engine compress data by applying encoding approaches. As Benifit we get ram free, scanning time (min). output generate fast.

vertipaq storage engine use the approaches to compress the data:-

- value encoding
- hash encoding
- run length encoding.

- value encoding: Compressing using value encoding in vertipaq.

- value encoding is basically applies on Integer, whole number, currency datatypes ~~columns~~ columns to compress data. vertipaq storage engine study at first integers/ whole/ currency column → Identify some relationship. To store data in small bit

pe

Quantity Sold

VertiPaq Compression Techniques

Value Encoding

Reduce the amount of memory
Required to store the data after
applying
The compression techniques

Quantity Sold
7059
7017
7123
7289
7244
7119
7126
7219
7170
7290
7043
7024
7035
7065
7236
7290

VertiPaq engine scans the sample of data
In a particular column
And then apply different encoding
Methods to compress the data

Value Encoding is used on Integer Column

Min Value 7017

Quantity Sold (Compressed Data)
42
0
106
272
227
102
109
202
153
273
26
7
18
48
219
273

7290

2^(n)>= N

Max Range of Values it can store	8192

13 Bits

273

Max Range of Values it can store	512

9 Bits

- The formula for store integer value as bit

$= 2^n > n$ where n is distinct value.

example $2^{13} \rightarrow$ 13 bit can store max = 8192

above example is given see the column value can be modified right by subtracting min value to minimize bit/storage.

(13 bit to 9 bit)

value encoding apply more various techniques to minimize rom space use by minimizing bit.

- Hash Encoding: Applied to column having string, int64, or floating point values.

In this kinds of scenario vertipaq engine at first see cardinality of any column. Cardinality (unique value present in a column). vertipaq engine rather than storing these string values at first create dictionaries using unique values (integer). then store only integer value corresponding their unique cardinality. Size of string values does not matter. uniqueness of column value matters as it decides how many bit will be required to store the data.

Hash encoding

Hash Encoding

Applicable to Columns having

String
Int64
Or
Floating Point Values

Dictionary Created

Index	Product Name
0	Mobile
1	Television
2	AC
3	Refrigerator

Product Name
Mobile
Television
AC
Refrigerator
Mobile
Television
AC
Refrigerator
Mobile
Television
AC
Refrigerator
Mobile
Television
AC
Refrigerator

After applying Hash Encoding

Product Index	Row#
0	1
1	2
2	3
3	4
0	5
1	6
2	7
3	8
0	9
1	10
2	11
3	12
0	13
1	14
2	15
3	16

VertiPaq Engine created table with index value and row value

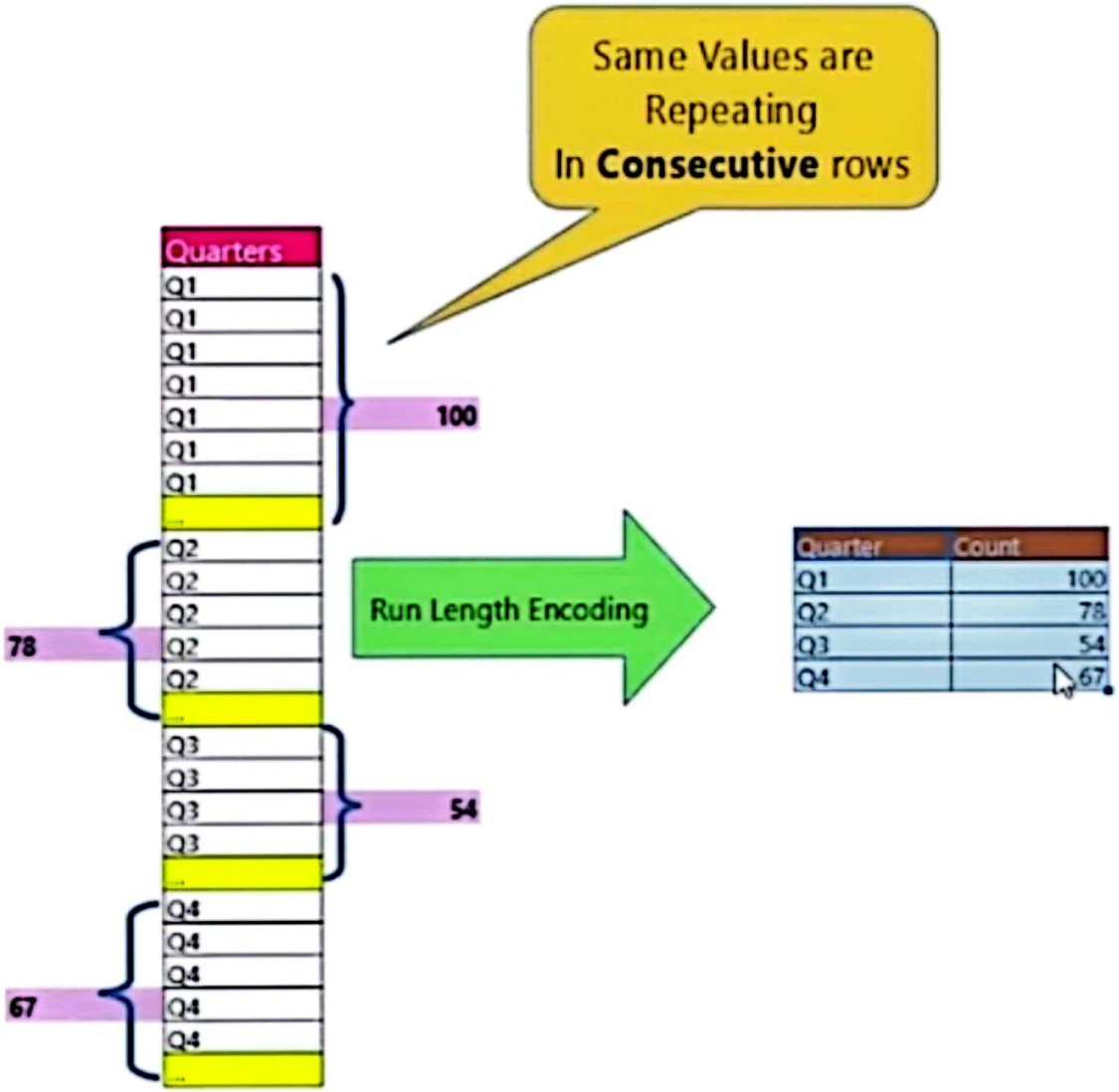
Storing Integer Values in place of String values

Saves the overall storage space significantly

Run Length encoding

Run Length Encoding
Reduces the size of Data Model by avoiding Repeated(Duplicate) Values

VertiPaq automatically Sorts data at the time of import or Refresh to achieve optimal compression



- Run Length Encoding: Sample values are repeating in consecutive rows.
  - Duplicate values must be stored consecutively / Adjacently before applying Run Length Encoding.

- Vertipaq engine can apply multiple encoding to one column based on its requirment: Basically more than one encoding can applicable to a particular column in order to ~~again~~ organize in memory best way.

- 4 Factors Determine compression of a column.
  1) cardinality of a column. (unique value)
  2) Number / Redundant rows
  3) Count of rows
  4) Datatypes of column.

4 Factors Determine Compression of a Column

1) Cardinality of Column
2) Number of redundant Rows
3) Count of rows
4) Data Type of Column

Country Name
Australia
Australia
Australia
Australia
Canada
Canada
Canada
Canada
India
India
India
India
United States of America
United States of America
United States of America
United States of America

Step 1) DE

Country Name	Index
Australia	0
Canada	1
India	2
United States of America	3

Index	Row
0	1
0	2
0	3
0	4
1	5
1	6
1	7
1	8
2	9
2	10
2	11
2	12
3	13
3	14
3	15
3	16

Step 2) RLE

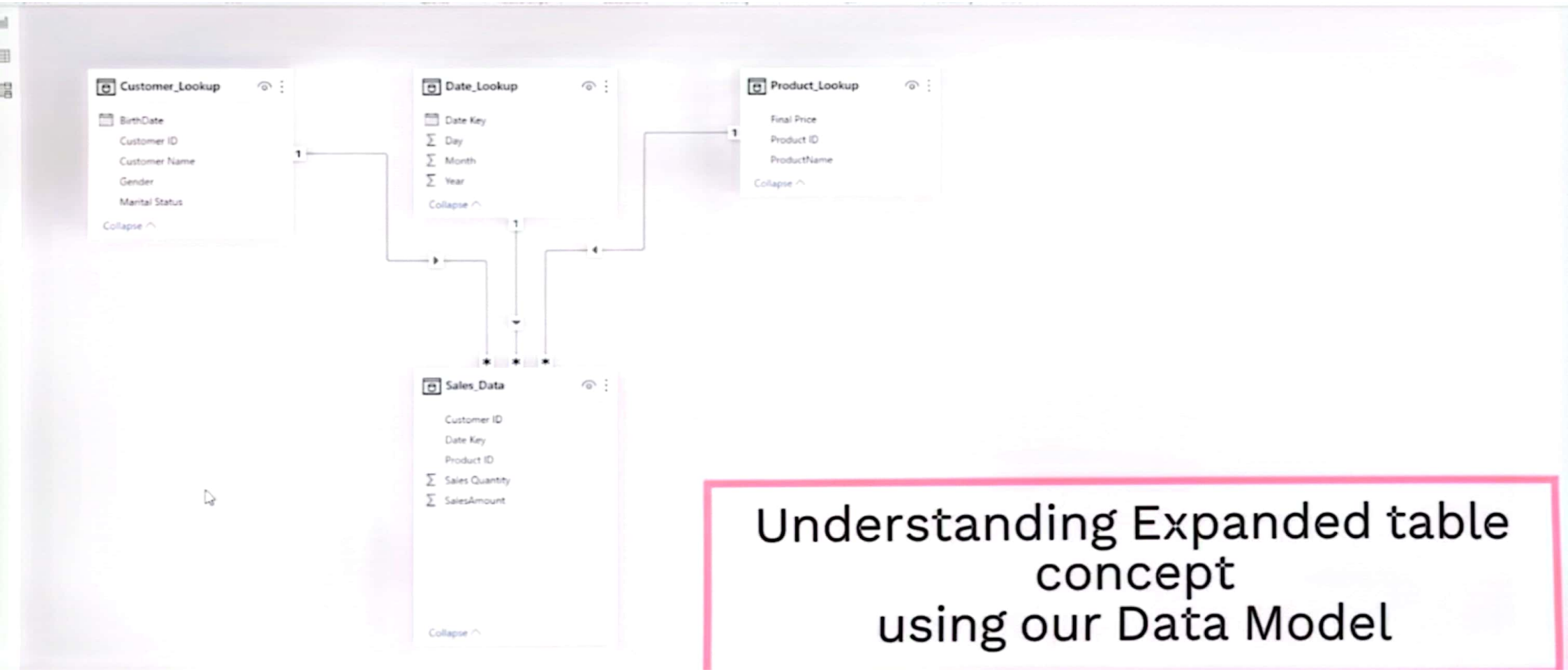
Index	Count
0	4
1	4
2	4
3	4

Hash/Dictionary + Run Length Encoding

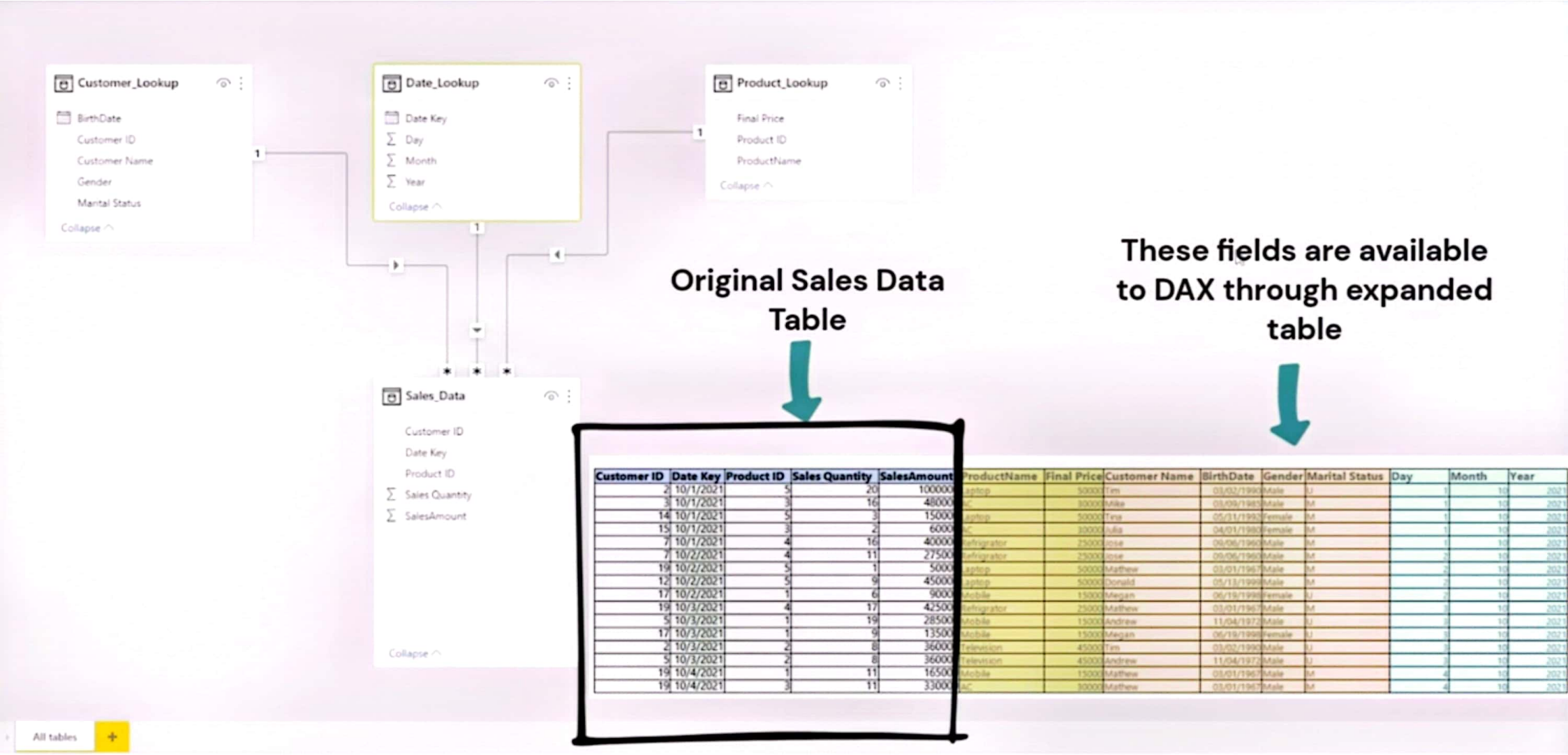
Expanded Tables

Many times we ab talk about context in Dax. Filter context, row context. we talk about how filter context works, The base of all comes from Expanded Tables.

- Expanded tables is key concept to understand before written writting Dax queries.
- Basicaly when a dimention table to make model (relationship) with Fact table (1 to many (*)) relationship then internally in Fact table one Exp- Expanded table is exist with both of dimetion and foct table colums.
- Filter context always pass on from one (1) side to many (*) side. (dimention table (1) → Fact table (*))
- which table in (*) many side that have expanded table.
- Filter context is propogate from dimention table to fact table

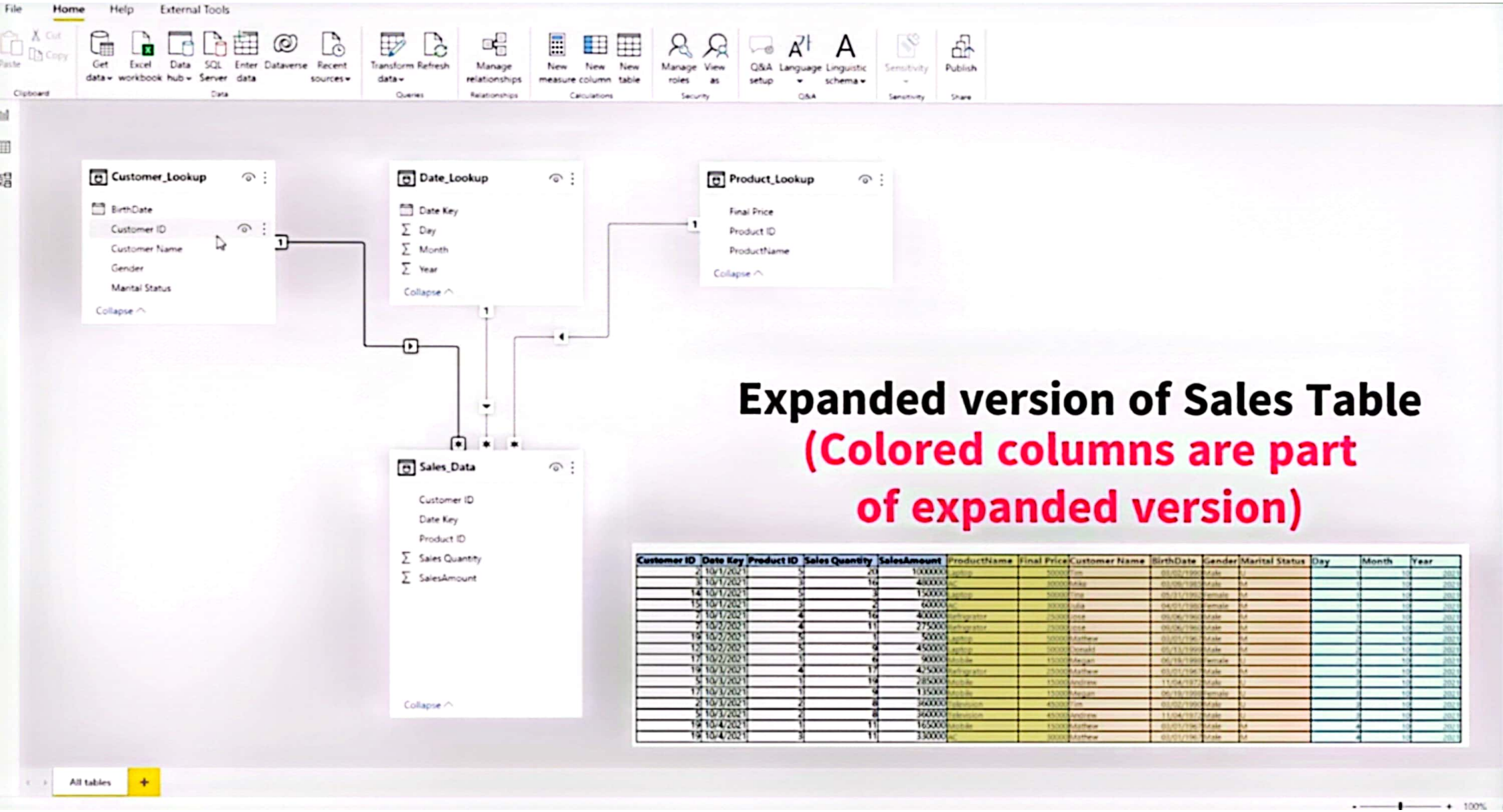


Understanding Expanded table concept using our Data Model



Customer ID	Date Key	Product ID	Sales Quantity	SalesAmount
2	10/1/2021	5	20	100000
3	10/1/2021	3	16	48000
14	10/1/2021	5	3	15000
15	10/1/2021	3	2	6000
7	10/1/2021	4	16	40000
7	10/2/2021	4	11	27500
19	10/2/2021	5	1	5000
12	10/2/2021	5	9	45000
17	10/2/2021	1	6	9000
19	10/3/2021	4	17	42500
5	10/3/2021	1	19	28500
17	10/3/2021	1	9	13500
2	10/3/2021	2	8	36000
5	10/3/2021	2	8	36000
19	10/4/2021	1	11	16500
19	10/4/2021	3	11	33000

ProductName	Final Price	Customer Name	BirthDate	Gender	Marital Status	Day	Month	Year
Laptop	50000	Tim	03/02/1990	Male	U	1	10	2021
AC	30000	Mike	03/09/1985	Male	M	1	10	2021
Laptop	50000	Tina	05/11/1997	Female	M	1	10	2021
AC	30000	Julia	04/01/1980	Female	M	1	10	2021
Refrigrator	25000	Jose	09/06/1960	Male	M	1	10	2021
Refrigrator	25000	Jose	09/06/1960	Male	M	2	10	2021
Laptop	50000	Mathew	03/01/1967	Male	M	2	10	2021
Laptop	50000	Donald	05/13/1999	Male	M	2	10	2021
Mobile	15000	Megan	06/19/1998	Female	U	2	10	2021
Refrigrator	25000	Mathew	03/01/1967	Male	M	3	10	2021
Mobile	15000	Andrew	11/04/1972	Male	U	3	10	2021
Mobile	15000	Megan	06/19/1998	Female	U	3	10	2021
Television	45000	Tim	03/02/1990	Male	U	3	10	2021
Television	45000	Andrew	11/04/1972	Male	U	3	10	2021
Mobile	15000	Mathew	03/01/1967	Male	M	4	10	2021
AC	30000	Mathew	03/01/1967	Male	M	4	10	2021


File
Home
Help
External Tools
Get data
Excel workbook
Data hub
SQL Server
Enter data
Dataverse
Recent sources
Transform data
Refresh
Manage relationships
New measure
New column
New table
Manage roles
View as
Q&A setup
Language
Linguistic schema
Sensitivity
Publish
Customer_Lookup
BirthDate
Customer ID
Customer Name
Gender
Marital Status
Collapse
Date_Lookup
Date Key
Day
Month
Year
Collapse
Product_Lookup
Final Price
Product ID
ProductName
Collapse
Sales_Data
Customer ID
Date Key
Product ID
Sales Quantity
SalesAmount
Collapse
Expanded version of Sales Table
(Colored columns are part of expanded version)
Customer ID | Date Key | Product ID | Sales Quantity | SalesAmount | ProductName | Final Price | Customer Name | BirthDate | Gender | Marital Status | Day | Month | Year
All tables
100%

Expanded Tables

Customer Name	Female_Customers_Purchase	Sum of SalesAmount
Andrew		15755000
Angela	12090000	12090000
Donald		19510000
Ebony	24860000	24860000
Hosse		25880000
Jasmine	21255000	21255000
Jason		18110000
Jennifer	16175000	16175000
Jose		16675000
Julia	21190000	21190000
Mathew		18090000
Megan	17790000	17790000
Mickey		19370000
Mike		13430000
Nathan		15135000
Simon		17435000
Tim		17585000
Tina	16850000	16850000
Victor		17925000
Total	**130210000**	**345110000**

Expanded Version of Sales Data Table contains Gender column

Customer ID	Date Key	Product ID	Sales Quantity	SalesAmount	ProductName	Final Price	Customer Name	BirthDate	Gender	Marital Status	Day	Month	Year
2	10/1/2021	5	20	10000000	Laptop	500000	Tim	03/02/1981	Male		1	10	2021
3	10/1/2021	3	16	480000	AC	30000	Mike	01/09/1981	Male		1	10	2021
14	10/1/2021	5	3	1500000	Laptop	500000	Tina	05/31/1981	Female		1	10	2021
15	10/1/2021	3	2	60000	AC	30000	Julia	04/01/1981	Female		1	10	2021
7	10/1/2021	4	16	400000	Refrigerator	25000	Jose	09/06/1981	Male		1	10	2021
7	10/2/2021	4	11	275000	Refrigerator	25000	Jose	09/06/1981	Male		2	10	2021
19	10/2/2021	5	1	500000	Laptop	500000	Mathew	03/01/1981	Male		2	10	2021
12	10/2/2021	5	9	4500000	Laptop	500000	Donald	03/13/1981	Male		2	10	2021
17	10/2/2021	1	6	900000	Mobile	150000	Megan	08/19/1981	Female		2	10	2021
19	10/3/2021	4	17	425000	Refrigerator	25000	Mathew	03/01/1981	Male		3	10	2021
5	10/3/2021	1	19	2850000	Mobile	150000	Andrew	11/04/1981	Male		3	10	2021
17	10/3/2021	1	9	1350000	Mobile	150000	Megan	08/19/1981	Female		3	10	2021
2	10/3/2021	2	8	3600000	Television	450000	Tim	03/02/1981	Male		3	10	2021
5	10/3/2021	2	8	3600000	Television	450000	Andrew	11/04/1981	Male		3	10	2021
19	10/4/2021	1	11	1650000	Mobile	150000	Mathew	03/01/1981	Male		4	10	2021
19	10/4/2021	3	11	330000	AC	30000	Mathew	03/01/1981	Male		4	10	2021

Page 1

Calculated Columns vs Measures

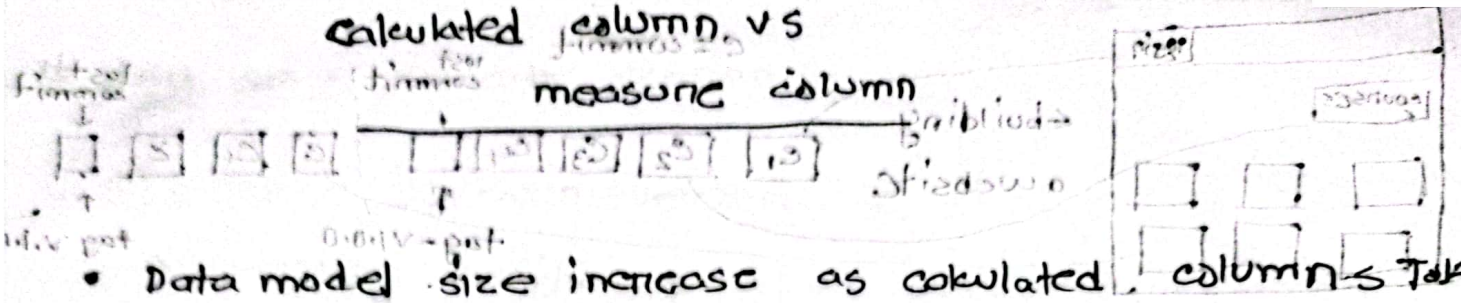
Calculated Columns

- Stored In-Memory
- Evaluation of values at the time of definition and when data refresh happens
- Used for Extension of some feature of table
- Complex Calculated Column - Although we will have large processing time initially but afterwards query time will be less

Measures

- Used for defining calculation
- Will not consume any memory unless used in visual
- They are stored as code only.

Calculated column vs measure column



- Data model size increase as calculated columns takes spaces. (consume memory)
- Measure do not consume memory as only source code is saved in Data Model; (consume only cpu)

= Measure is efficient but why we use calculated columns, if we have measure ??

-90% time use measure. but performing complex and large calculations we will use calculated columns to enchance user experience who is viewing the report.

- Measures are computed on the fly, Hence when complex and large calculation are present they take more time to display the value.
- If we go to display very complex calculation under measure, and dataset large as well Then it would take time to render our report. (this kind of scinario we use calculated column)

- Default context of our calculated column is row wise (way of evaluation)

  . that means calculated colum context default row wise works.

- Calculate() Transition from Row context to Filter context.

  - Basicaly calculate column evolution does in row wise context.

  we can also evaluate calculate column with filter contex. By **Calculate()** Function.

  CALCULATE(Expression, Filter 1, Filter 2 · — -Filter n)

  example. CALCULATE(SUM(sales[sales amount]))

- In measure filter context is automatically occur.

  measure have by default Filter context

  . if you use CALCULATE() function it of filter similar as measure (inbuilt filter context)

measure table

- key advantage of separate measure table is organize Data Model tab.
- To create a measure table click on enter data option from Home tab.
- A measure table will create where you can create diff diff measure.
- To shift measure you have created in another tables. you want to shift all deorganized measure to measure table.

just click on measure. A measure tool ribbon will open. then Home table option → change Home table by click in measure table.

CALCULATE( ) Function.

CALCULATE(<Expression>[,<filter1>[,filter-2[,...]]])

1) Accept boolean filter expression
2) Accept table filter expression

example

CALCULATE(SUM(sales_amount), customer_name = "Andrew")

- calculate() function is basically change filter context.

- Dax engine treats filter arguments in calculate function as Tables not conditions.

example:

*own measure

▷ Sales_Amount(product_lookup) = CALCULATE(SUM(sales_(sales amount)), product[product_name] = "mobile")

- Dax engine transform this logical expression into table.

* Dax measure

▷ Sales_amount(productwise) = CALCULATE(sum[sales[sales amount]], Filter(All(product_lookup), product[product_name] = "Mobile"))

↑ In backend Dax use long syntax.

* Filter context overwrite.

Filter (All(product_lookup), product_lookup[product_name] = "mobile")

↓ This all function overwrite external features. (outer filter will) ignore

↓ inner filter.

- Filter overwrite when outer filter column and inner filter column is same.

Calculate modifier : KEEPFILTERS(Expression)

This function take initial filter context (external filter) and then ⟶ calculate() then apply over it.

Basicaly from existing one filter layer ⇒ again another filter layer.

GFC ⟶ calculate() ⟶ modifiers change

visual slicer

Types of Calculate Modifiers

1) Modify Filters (KeepFILTERS, REmove FILTERS etc)
2) To access inactive relationship (USERELATIONSHIP)
3) To change Filter flow (CROSSFILTER): CROSS FILTER change the direction of Filter propagation.

① KEEPFILTERS:

mobile_soles =

CALCULATE([Total_Soles], KEEPFILTERS(product_lookup[P_nom]="Mob

KEPFITERS Apply Filter only given filter item.

- If outer column is diff from measure, KEEPFILTER and calculate() inter filter act same).
- Keep filter show only outer filter and inner filter intersection output.


Data
Queries
Insert
Calculations
Sensitivity
Share
Calculate Modifier: KEEPFILTERS(Expression)
Product_Name | Mobile_Sales (Without KEEPFILTERS) | Mobile_Sales (With KEEPFILTERS)
AC 34065000
Laptop 34065000
Mobile 34065000 34065000
Refrigerator 34065000
Television 34065000
Total 34065000 34065000
Customer_Name | Mobile_Sales (With KEEPFILTERS) | Mobile_Sales (Without KEEPFILTERS)
Nathan 870000 870000
Tim 960000 960000
Jasmine 1050000 1050000
Jason 1170000 1170000
Tina 1395000 1395000
Angela 1425000 1425000
Ebony 1575000 1575000
Donald 1650000 1650000
Mike 1770000 1770000
Victor 1815000 1815000
Mathew 1830000 1830000
Julia 2040000 2040000
Simon 2070000 2070000
Andrew 2100000 2100000
Jose 2145000 2145000
Total 34065000 34065000
Visualizations
Build visual
Filters
Values
Add data fields here
Drill through
Cross-report
Keep all filters
Add drill-through fields here
Fields
Search
Measures_Table
Mobile_Sales (With KEEPFILTERS)
Mobile_Sales (Without KEEPFILTERS)
Total_Sales_Amount
Customer_Lookup
Birth_Date
Customer_ID
Customer_Name
Gender
Marital_Status
Date_Lookup
Product_Lookup
Sales_Data

Remove FILTERS expression

REMOVEFILTERS(Table Name or Column-name, [column name])

- Remove FILTER is basicaly used for delecon filters.

For cleon filters from a partiewlon Table / column }

while * It should be kept in mind while inserting

column or name under REMOVEFILTER( )

you can only insert a partiewlon tables columns.

not from different table different columns.

- The difference between REMOVEFILTERS() and ALL( )

Function is RF can work as calculat() modifiers.

and can use under calculate.

ALL() Function can work as Table function)

essentially both wunk similarly

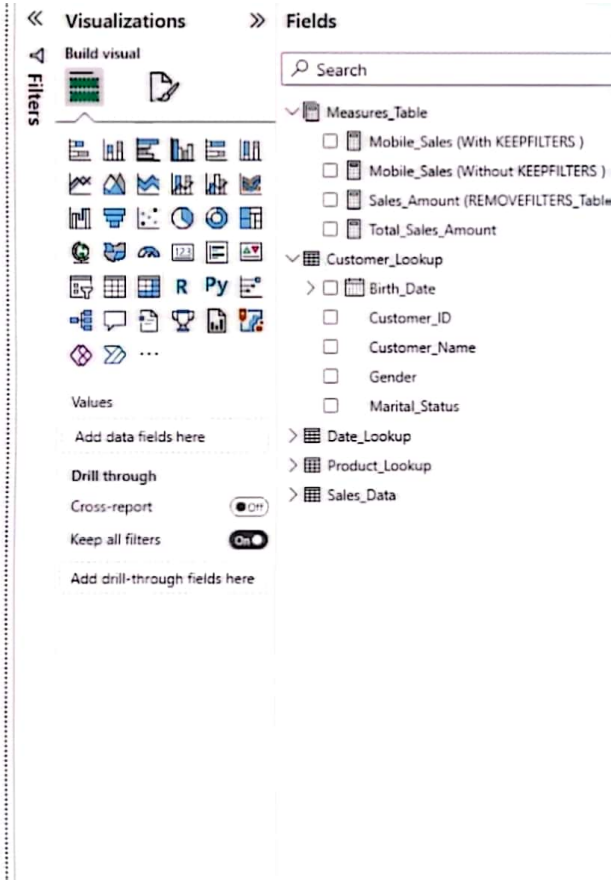
- When you apply RF on whole table then no

filter will apply on that table.

Calculate Modifier : REMOVEFILTERS(TableNameorColumnName,[ColumnName],[....])

Customer_Name	Total_Sales_Amount	Sales_Amount (REMOVEFILTERS_Table)
Andrew	15755000	345110000
Angela	12090000	345110000
Donald	19510000	345110000
Ebony	24860000	345110000
Hosse	25880000	345110000
Jasmine	21255000	345110000
Jason	18110000	345110000
Jennifer	16175000	345110000
Jose	16675000	345110000
Julia	21190000	345110000
Mathew	18090000	345110000
Megan	17790000	345110000
Mickey	19370000	345110000
Mike	13430000	345110000
Nathan	15135000	345110000
Simon	17435000	345110000
Tim	17585000	345110000
Tina	16850000	345110000
Total	**345110000**	**345110000**

Gender	Sales_Amount (REMOVEFILTERS_Table)
Female	345110000
Male	345110000
Total	**345110000**



syntax for applying Remove Filter on whole table:-

= CALCULATE ([Total-sales] , REMOVEFILTERS (customer_table))

- syntax for applying Remove Filter on columns

= CALCULATE ([Total-sales], REMOVE FILTERS (customer_table [custom_ID

customer_table [customer_name],))

Remove filter use case (To derive % of Total)

SUM vs SUMX

SUM(Sales_Data[Sales_Quantity]) SUMX(Sales_Data, Sales_Data[Sales_Quantity])

- What is aggregate function in Dax.
  - Aggregate functions aggregate column value and return single value

    SUM(), MIN(), MAX(), AVG() those are aggregation functions.
  - Aggregate function can take only single argument (column)

    MIN(Sales_Data[Sales_Amount])

    ' ' can take only single column.

- Advanced version of aggregate is Iterators function.

Iterators functions are aggregate function but they are different from aggregate function.

Iterator like SUMX() are AVGX() MAXX() at first take a table as a input (1st parameter). and then take (2nd parameter). expression

- Iterators are used to perform calculations on multiple columns of same table are related tables

- Suppose we need multiply two column o sum of those multiplied result.

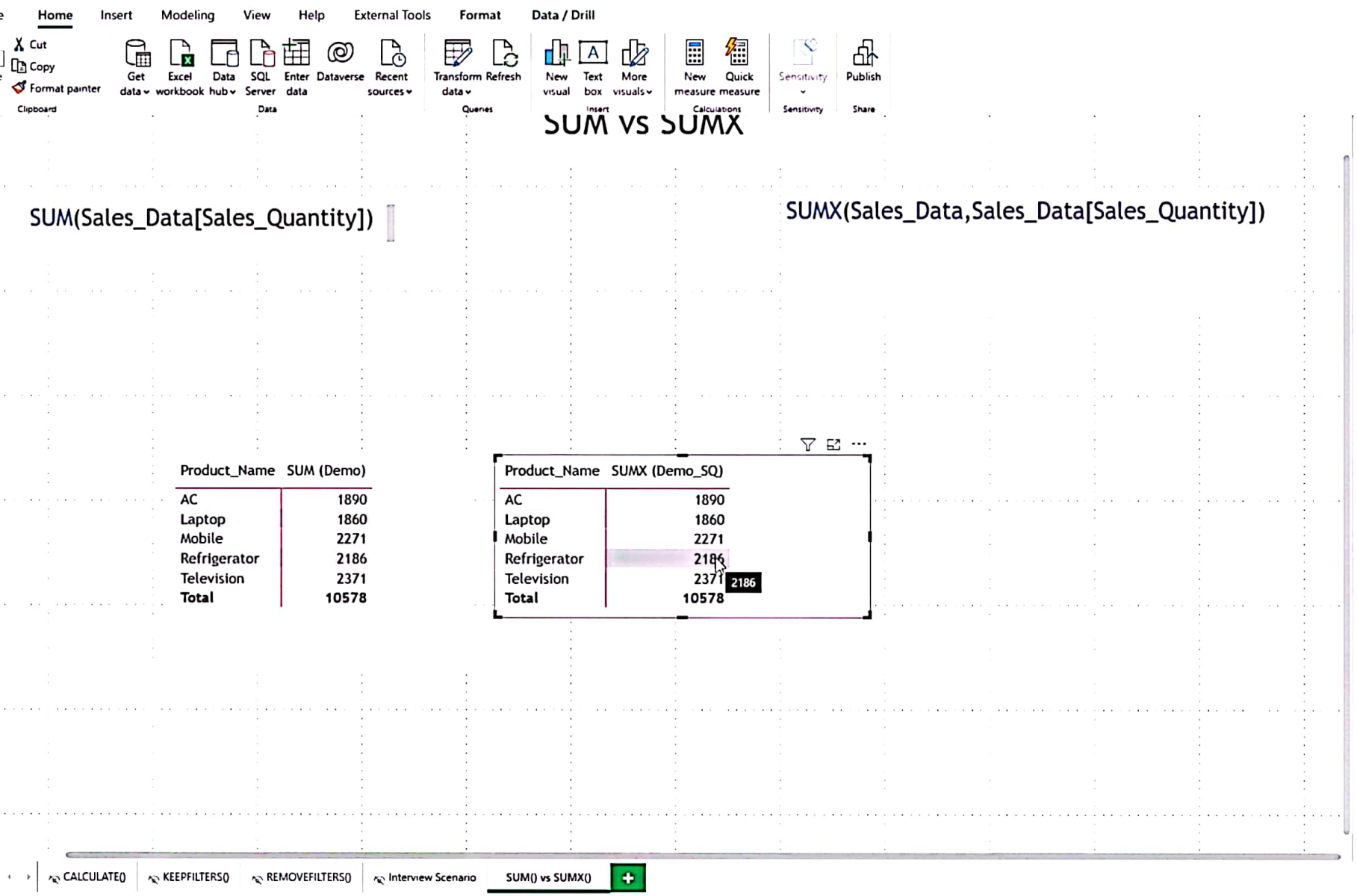
then use - SUMX

- SUMX() Perform Row by Row evolution of expression.

SUMX(Table , expression)

SUMX(Saled_Data , Sales_Data[Amount] / Sales_Data[Quantity])

- SUM() & SUMX() Function only differ in syntax
- Dax Engine interprets SUM() as SUMX() only
- Basicaly Dax engine internally run Iteration function. P.BI gives us simplied syntax for aggregate;


Home
Insert
Modeling
View
Help
External Tools
Format
Data / Drill
Cut
Copy
Format painter
Clipboard
Get data
Excel workbook
Data hub
SQL Server
Enter data
Dataverse
Recent sources
Data
Transform data
Refresh
Queries
New visual
Text box
More visuals
Insert
New measure
Quick measure
Calculations
Sensitivity
Sensitivity
Publish
Share
SUM vs SUMX
SUM(Sales_Data[Sales_Quantity])
SUMX(Sales_Data,Sales_Data[Sales_Quantity])
Product_Name SUM (Demo)
AC 1890
Laptop 1860
Mobile 2271
Refrigerator 2186
Television 2371
Total 10578
Product_Name SUMX (Demo_SQ)
AC 1890
Laptop 1860
Mobile 2271
Refrigerator 2186
Television 2371
2186
Total 10578
Visualizations
Build visual
Filters
Rows
Product_Name
Columns
Add data fields here
Values
SUMX (Demo_SQ)
Drill through
Cross-report
Keep all filters
Add drill-through fields here
Fields
Search
Measures_Table
SUM (Demo)
SUMX (Dem
SUMX (Dem
Total_Sales_
Customer_Lookup
Date_Lookup
Product_Lookup
Final_Price
Product_ID
Product_Nar
Sales_Data
CALCULATE()
KEEPFILTERS()
REMOVEFILTERS()
Interview Scenario
SUM() vs SUMX()

FILTER FUNCTION

FILTER(<table>, <filter>) ← Filter condition

** Returns a Table

** Reduce number of rows based on filter criteria

Table Arguments in Function either accept physical table or virtual table

- Filter Expression passed as 2nd Argument has row context, as it is evaluated for each row.

- Example of physical Table

SUMX (physical Table) = SUMX(Sales_Data, Sales_Data[quantity] * 10)
Table, expression

- Example of vertual table

SUMX (vertual Table) =
vertual filtered table

SUMX(FILTER(Sales_Data, Sales_Data[quantity] < 1000),
Sales_Data[Sales_Amount] / 1000)
expression over filtered table

- FILTER under calculate

= CALCULATE ([Sum_amount], FILTER(Sales_Data, Sales_Data[ID] = 1))

Distinct() Function:

- DISTINCT() Function is a table function.

- Returns single column table of unique values when column is passed as an argument.
- Return single column Table on unique values when column is passed as an argument
- Return unique combination of values when table expression is passed as an argument

- To create a DISTINCT Function
  - click on measure table
  - Then rather than create measure we will create new table.
  - Because DISTINCT() is a Table function

DISTINCT (Table-name/column name)

Values Function:

- VALUES (ColumnName / TableName)

Case-1: When columnname is passed then it returns a single column table of unique values

Case-2: When a table name is passed then it returns
a) Entire table including duplicates
b) Blank rows.

- Major difference between Distinct and value function is "Blank rows", values Function return blank row as well.

ALLEXCEPT() Function

ALLEXCEPT (<TableName>, <ColumnName>, [,<ColumnName> [,...]])

* should be existing table, Table expression not allowed.
* column from referenced table or the table on one side of relationship (Expanded version of referenced table).
* All except function Remove filter from ^ all column Except argumented column of given table.

- Exceptall() function can take colum from fact table and dimention table as well. with standing in fact table. Fact Because Fact table have Expanded table. means all Dimention tables columns one also in fact table.

ALLEXCEPT (<table name> <column name>)

↓

The columns that you dont want to w Filter

* ALLEXCEPT() could be used in 2 scenation

1) As a CALCULATE modifier. w most of the time

2) As a table Function.

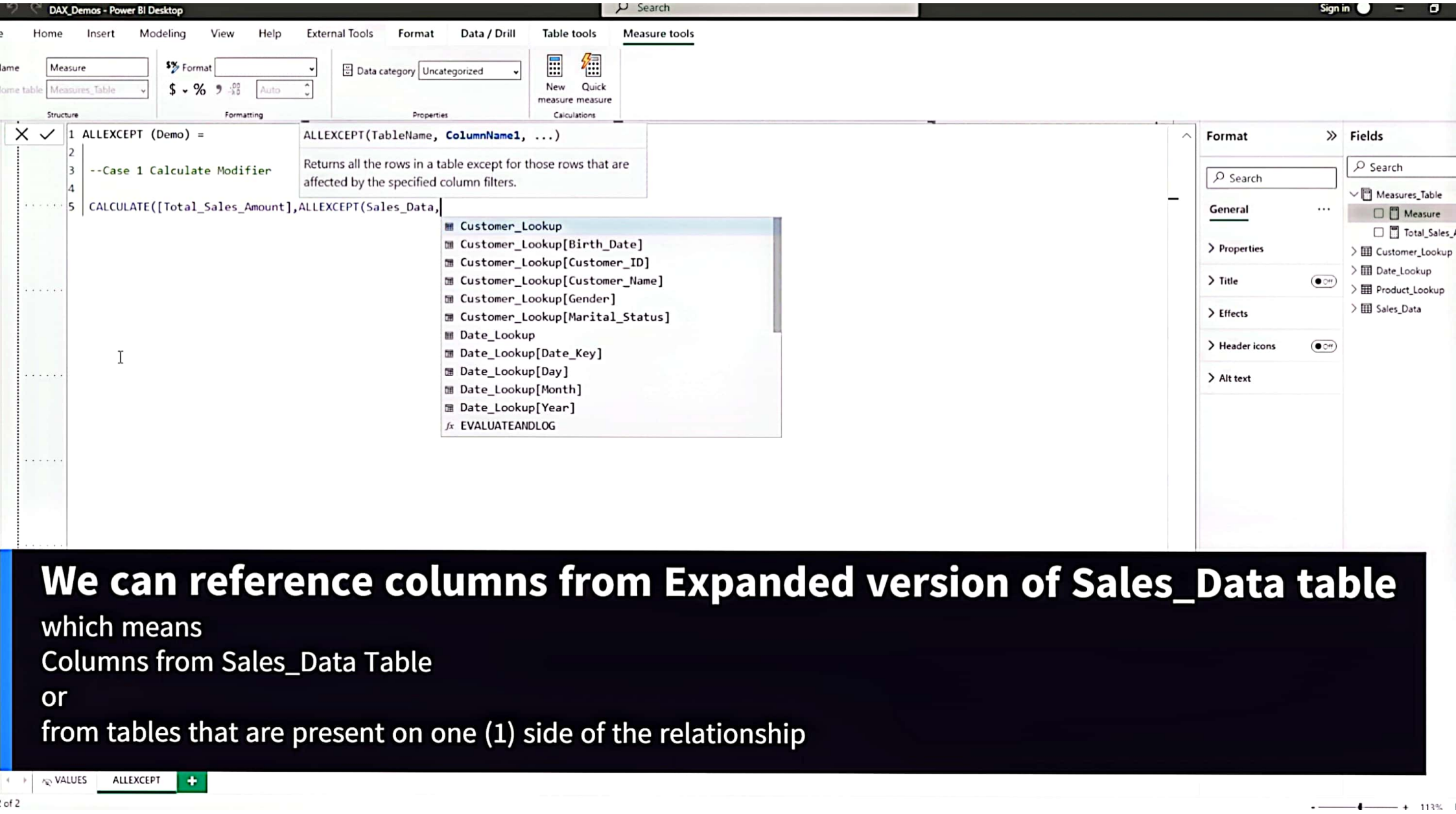
= CALCULATE ([Total-sale-amount], ALLEXCEPT (sales-Data, customer[customer-ID], product[product ID]))

calculate expression

table

columns from diff diff table

- When we use ~~customer~~ to ALLEXCEPT() function or as a Table function it return unique combination of row of the table. But the column input you give those one not show because Logically you remove Filter from those.



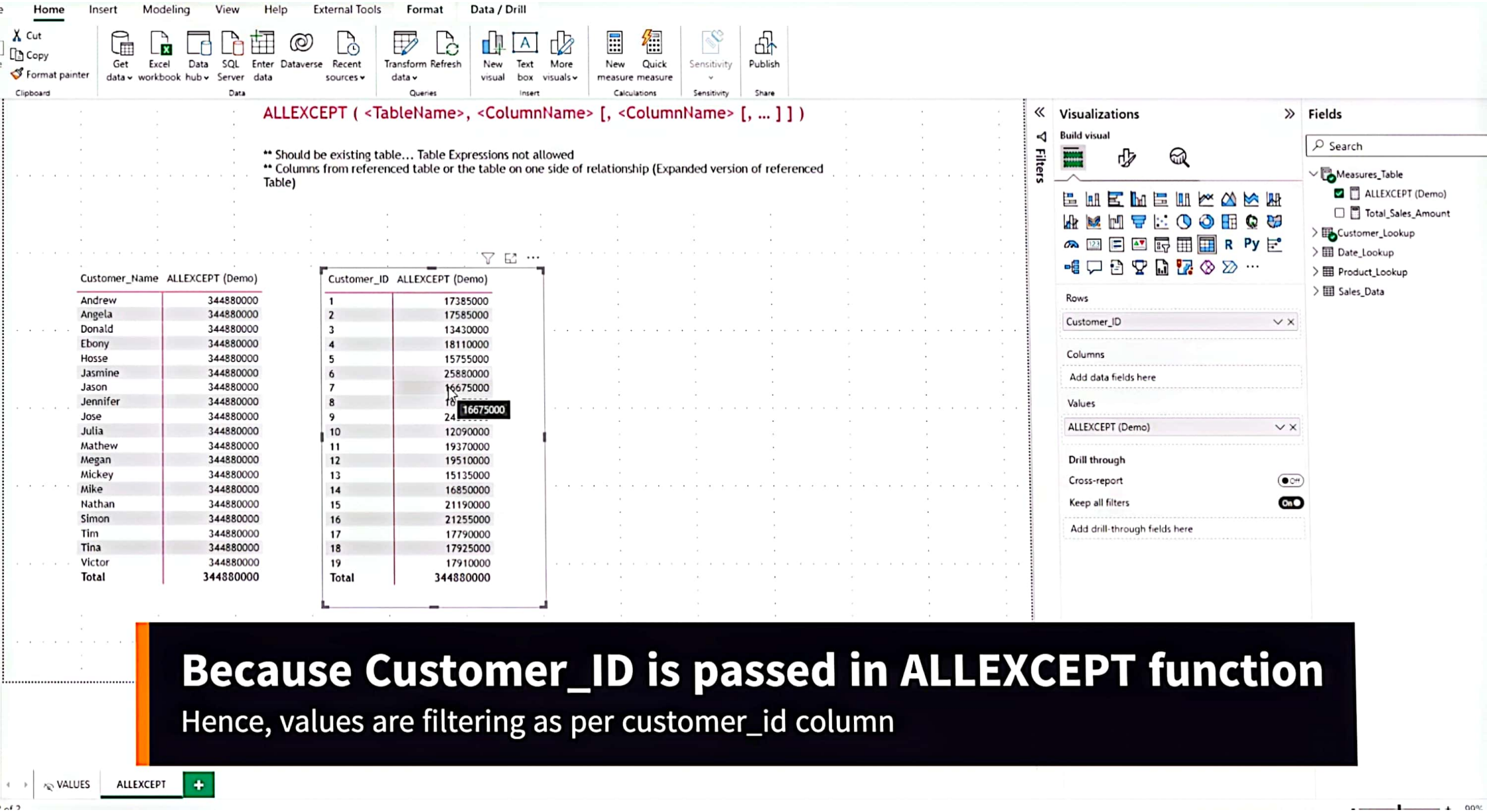
We can reference columns from Expanded version of Sales_Data table

which means

Columns from Sales_Data Table

or

from tables that are present on one (1) side of the relationship


Home
Insert
Modeling
View
Help
External Tools
Format
Data / Drill
Cut
Copy
Format painter
Clipboard
Get data
Excel workbook
Data hub
SQL Server
Enter data
Dataverse
Recent sources
Data
Transform data
Refresh
Queries
New visual
Text box
More visuals
Insert
New measure
Quick measure
Calculations
Sensitivity
Sensitivity
Publish
Share
ALLEXCEPT (<TableName>, <ColumnName> [, <ColumnName> [, ...]])
** Should be existing table... Table Expressions not allowed
** Columns from referenced table or the table on one side of relationship (Expanded version of referenced Table)
Customer_Name ALLEXCEPT (Demo)
Andrew 344880000
Angela 344880000
Donald 344880000
Ebony 344880000
Hosse 344880000
Jasmine 344880000
Jason 344880000
Jennifer 344880000
Jose 344880000
Julia 344880000
Mathew 344880000
Megan 344880000
Mickey 344880000
Mike 344880000
Nathan 344880000
Simon 344880000
Tim 344880000
Tina 344880000
Victor 344880000
Total 344880000
Customer_ID ALLEXCEPT (Demo)
1 17385000
2 17585000
3 13430000
4 18110000
5 15755000
6 25880000
7 16675000
8
9
10 12090000
11 19370000
12 19510000
13 15135000
14 16850000
15 21190000
16 21255000
17 17790000
18 17925000
19 17910000
Total 344880000
16675000
Visualizations
Build visual
Filters
Rows
Customer_ID
Columns
Add data fields here
Values
ALLEXCEPT (Demo)
Drill through
Cross-report
Keep all filters
Add drill-through fields here
Fields
Search
Measures_Table
ALLEXCEPT (Demo)
Total_Sales_Amount
Customer_Lookup
Date_Lookup
Product_Lookup
Sales_Data
Because Customer_ID is passed in ALLEXCEPT function
Hence, values are filtering as per customer_id column
VALUES
ALLEXCEPT

ALL vs ALLSELECTED in DAX

- ALLSELECTED(TableName or ColumnName), [ColumnName-1])

ALLSELECT() function have two usecase

a) use as a CALCULATE modifier

b) As a table function

- ALLSELECTED() function ignore filters comming from the visuals (rows and colums) but respect filter comming from outside i.e. slicer etc.
- keep the filter comming from outside.
- Ignoring Filters that might have been applied inside query.

- ALL() Function in Dax remove all filter (inner query / outside filter).

= CALCULATE([Total-sales-amount], ALLSELECTED (custome [customer.nam

SELECT COLUMNS ( )

SELECTCOLUMNS (<table>, [<name>], <expression> <name>...)

- Its a table function
- Its an iteration function: Scans row by row (row context)
- Returns a table.
- Arguments
  a) Table: physical or virtual
  b) Name: Name of new column
  c) expression: Expression that returns a scalar value
     [column, referenced, Integer or string value]

- Selectcolum is a powerfull function. You can create a table by modify your column name, modifying column value,

physical table
= SELECTCOLUMNS (customer, "cust_nom", customer [customer name],
"Year_of_Birth", YEAR [customer [Birthday]])

virtual table
= SELECTCOLUMN (FILTER (customer, customer [name] = "Mole"),
"name", customer [name], "Birth year", YEAR (customer [Birth]))

Add columns:

ADDCOLUMNS (<table>, <name>, <expression> [<name>, <expression>])

- Its a table Function
- Return the table with [original column + Added column]
  - Table: physical + virtual (Table to which new columns need to be handle)
  - Name: Name of the column (enclose in double quotes)
  - Expression: DAX expression that returns scalar value,
- Difference between SELECTCOLUMNS and blank ADDCOLUMN is SELECTCOLUMNS create only query specified colum to start with a blank table, whereas Add column create query specified column with existing column of table.

= ADDCOLUMNS (Customer, "Custname", Customer[name],

"Birth-year", YEAR(Customer[Birth-Date]))

Datatable Function

DATATABLE (name1, type-1, [name-2, type-2, ...] ← table sub construction

{ value1, value-2 } ← row (col wise value)

- DT use for creating table which will contain static value
- In DT function no column reference, no table reference.

```
DATATABL (Demo) =
    DATATABLE ("column-1", INTEGER, "column-2", STRING, ← Table substructu
    {
        { 1, "ABC" }   ← row one
        { 2, "XYZ" }   ← row 2
    }
    )
```

Table Construction Function:

- using the table construction we design table.
- In table construction not able to specify name of columns.
- Table construction is use to create table of one or more columns
- values (1. Fixed values 2. Expression returning scalar values)

- The Advantage of Table construction over Datatable is we can write expression under table construction)

Fixed vale

```
Table constructor (Demo) =
{
(1,2),
(3,4)
}
```

with the expression

```
Table constructor (demo) =
{
(1,2),
(3,SUM(sales[amount]))
(calculate(expr...)),
('A','B','C')
}
```

UNION vs INTERSECTION

- UNION

= UNION returns all the rows for each of the table.

(Table expression)

- UNION Argument must have same number of columns

Duplicate rows and retained

order of table matter.

- Intersection:

- Returns rows available in both the tables

(Table) expression

' Argument must have same number of columns,

' duplicate row are retained.

order of table matter,

Related Function

RELATED (column-name)

- Return a related value from another table in the Data model.
- Must reference a table that is on the one side of the relationship (1:*)
- Helpfull in accessing columns from expended version of the table.

Use case

- as a measure
- as a calculated column

Reasons

Related_demo =

SUMX ('Sales_Data', 'Sales_Data'[Sales_quantity] * RELATED ('Product_Loo [Final_price]))

- Interviewer might ask you to fetch values from table on side for which you need to use Related Function.

Column :=

Related ('Product_lookup'[Product_name])

By reference we retrieve -

Related Table!

- Used to traverse the relationship from (1) side to many(*) side of the relationship,
- It perform context transition.
- Reference physical table present on many side of the relationship.

- Difference Between Related and RELATED TABLE.

is RELATED() function only return single value.

But when we need to traverse (1 to *) we need to fetch some data from (one to many) then we need RELATED Table.

- Count
- COUNTROWS (Related
- COUNTROWS (RELATED TABLE (Sales-data))