Hands-On Walkthrough 3

Adding Calculations to a Model

Cardinal Directions

turning oceans of data into lines of sight

Chapter 1. How to Create Calculated Columns in Power BI

1. Go to the **Table** view or **Data** view
2. Select the table that holds the data to be used.
3. Click **New Column** on the Modeling tab
4. Type in the DAX formula
5. Press Enter

> **NOTE** Calculated columns are computed row-by-row and stored in the table, unlike measures which calculate dynamically.

Author based Calculated Columns

```
_Last, First = Author[Last Name] & ", " & Author[First Name]
```

```
_First Initial = LEFT(Author[First Name],1)
```

```
_Last Letter = RIGHT(Author[First Name],1)
```

```
_Mid Name = MID(Author[First Name],2,3) -- Get 3 characters starting at position 2
(characters 2▯4):
```

Edition based Calculated Columns

```
_Extended Print Run = Edition[Print Run Size (k)]*1000
```

Chapter 2. If-Then-Else

We’ve already done examples like these

IsExpensive

```
IsExpensive = IF ( Edition[Price] > 30, "Expensive", "Regular" )
```

DiscountTier

```
DiscountTier =
IF ( Edition[Price] >= 40, 0.20,
IF ( Edition[Price] >= 25, 0.10, 0.00 )
)
```

More Options

SWITCH - An Alternative to IF

FormatLabel * This will require changing the name of the column to BookFormat.

```
FormatLabel =
SWITCH (
Edition[Format],
"Hardcover","HB",
"Graphic","GP"
"Trade paperback","TR"
"Mass market paperback","MM"
"Other"
)
```

PriceBand

```
PriceBand =
SWITCH (
TRUE(),
Edition[Price] < 10, "Under $10",
Edition[Price] < 20, "$10-$19.99",
Edition[Price] < 30, "$20-$29.99",
">= $30"
)
```

NULL Values . Create *SafePrice

- Fills missing data (NULL) with a default value.

```
SafePrice = COALESCE ( Edition[Price], 0 )
```

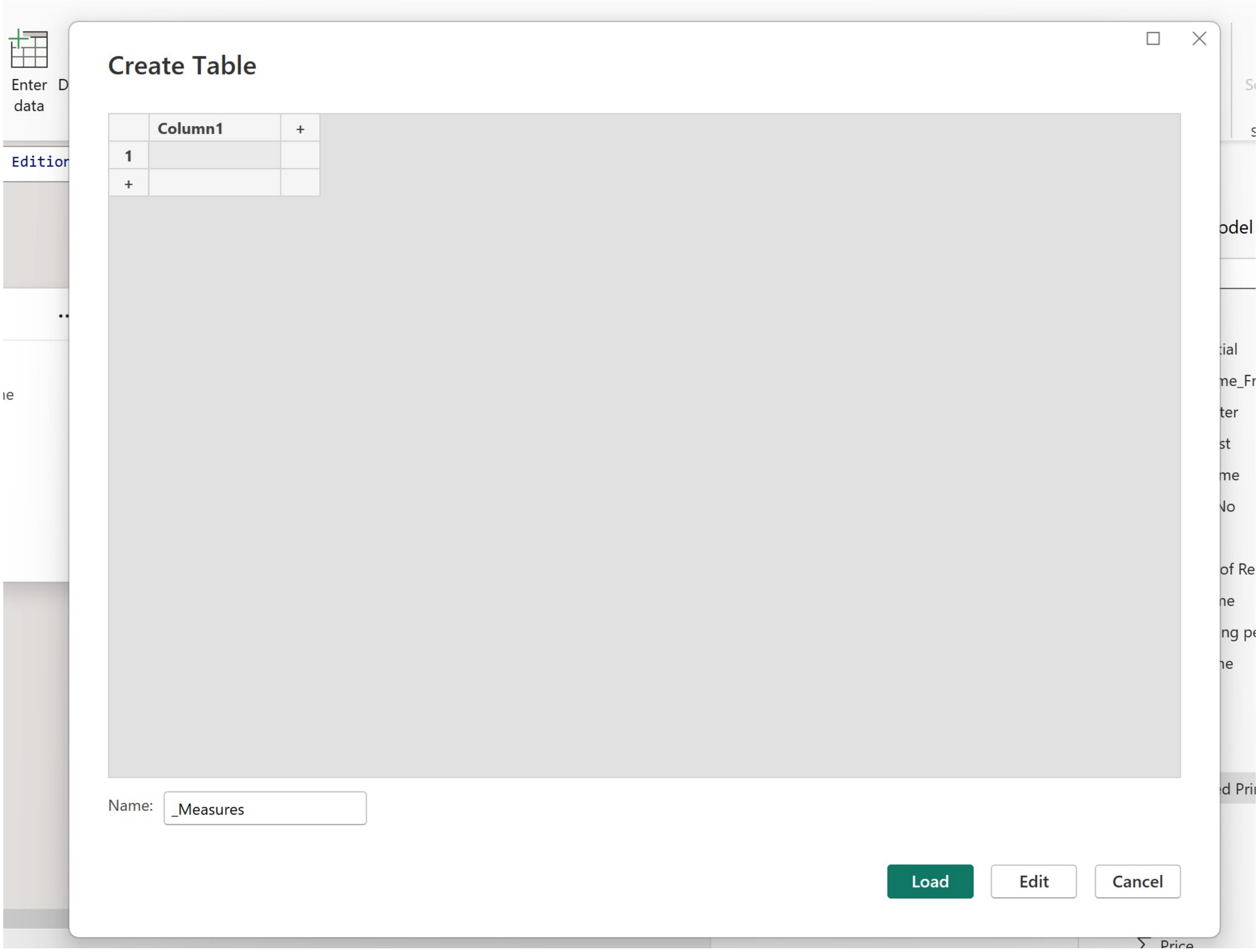
Chapter 3. How to Create Calculated Measures in Power BI

1. Go to the **Model** view
2. Click **New Measure**
3. Type the DAX formula
4. Press Enter

NOTE	These measures will automatically calculate based on any filters or slicers you apply in your reports!

Create a New Table to Hold Calculated Measures

1. Go to the **Model** view
2. Click **Enter Data** on the **Home** ribbon.
3. Name the new table **_Measures**.
4. Press Load.



Name: _Measures

The Basics

Total Quantity Sold

```
Total Quantity Sold = SUM(Q1_Sales[QUANTITY])
```

Average Discount

```
Average Discount = AVERAGE(Q1_Sales[Discount])
```

Order Count

```
Order Count = DISTINCTCOUNT(Q1_Sales[OrderID])
```

Chapter 4. Using Iterator Functions

These are Iterator Aggregate functions. These are used when there is a formula as part of the aggregation.

Basic Aggregate Functions

- `Total Quantity Sold = SUM('Q1-Q1_Sales'[QUANTITY])`
- `Average Discount = AVERAGE('Q1-Q1_Sales'[Discount])`
- `Order Count = DISTINCTCOUNT('Q1-Q1_Sales'[OrderID])`
- `Product Count = COUNT(Products_Dim[ProductID])`
- `Customer Count = COUNTROWS(Customers_Dim)`

Iterator Functions in DAX

An **iterator function** loops over the rows of a table, creates **row context**, evaluates an **expression for each row**, then returns either an **aggregate value** or a **new table**.

Two Big Groups of Iterators

1. Scalar aggregators with “X” (return a single value)
    - `SUMX ( table, expression )`
    - `AVERAGEX ( table, expression )`
    - `MINX / MAXX ( table, expression )`
    - `COUNTX ( table, expression )`
    - `RANKX ( table, expression [, value] )`
    - `CONCATENATEX ( table, expression [, delimiter] )`
2. Table-shaping iterators (return a table)
    - `FILTER ( table, condition )`
    - `ADDCOLUMNS ( table, name, expression, … )`
    - `SUMMARIZE`, `GENERATE`, `CROSSJOIN` (and related)

Line Amount

```
_Line Amount = sumx(Q1_Sales,Q1_Sales[QUANTITY]* RELATED(Edition[Price]))
```

Line Total with Sales Tax (8.25%)

```
Line Total with Sales Tax (8.25%) = round(Q1_Sales[Line Total]*(1+0.0825),2)
```

What happens in _Line Amount:

SUMX(Q1_Sales, …) iterates row by row over Q1_Sales.

For each row, it calculates Q1_Sales[Quantity] * RELATED(Edition[Price]).

RELATED(Edition[Price]) pulls in the price from the Edition table via the relationship.

SUMX then adds all those row-level products together to return one number per filter context (for example, per book, per month, per customer).

More Examples

```
_Revenue After Discount =
SUMX(
    Q1_Sales,
    Q1_Sales[QUANTITY] *
    RELATED(Edition[Price]) *
    (1 - IF(ISBLANK(Q1_Sales[DISCOUNT]), 0, Q1_Sales[DISCOUNT]))
)
```

```
_Avg Sale per Order =
AVERAGEX(
    VALUES(Q1_Sales[ORDER_ID]),
    CALCULATE(
        SUMX(
            Q1_Sales,
            Q1_Sales[QUANTITY] * RELATED(Edition[Price])
        )
    )
)
```

```
Lowest Price Sold =
MINX(
    Q1_Sales,
    RELATED(Edition[Price])
)
```

```
Highest Price Sold =
MAXX(
    Q1_Sales,
    RELATED(Edition[Price])
)
```

```
_Unique Books Sold =
COUNTX(
    VALUES(Q1_Sales[ISBN]),
    Q1_Sales[ISBN]
)
```

Chapter 5. SUM vs SUMX

You use `SUM` when you are adding up a single numeric column, and `SUMX` when you need to sum an expression evaluated row by row (often involving multiple columns or `RELATED`).

5.1. When to Use SUM

Use `SUM` when:

- Summing a single numeric column.
- There is no per-row formula needed, just “add these numbers up”.
- The column already represents the value to aggregate.

5.2. Examples

Total quantity sold:

```
Total Quantity :=
SUM ( Q1_Sales[Quantity] )
```

Total line amount when you already have a `LineAmount` column:

```
Total Line Amount (column) :=
SUM ( Q1_Sales[LineAmount] )
```

In these examples:

- Look at all visible rows of `Q1_Sales`.
- Add the values in that one column.
- Respects filters (for example, by date, product, or customer).

You cannot do something like:

```
SUM ( Q1_Sales[Quantity] * Edition[Price] )
```

because `SUM` only takes a single column, not an expression.

5.3. When to Use SUMX

Use `SUMX` when:

- The need is to calculate something per row, then sum it.
- The calculation involves multiple columns or `RELATED` values.
- There is no existing column that already stores the final value to aggregate.

General pattern:

```
MeasureName :=
SUMX (
    TableToIterate,
    ExpressionUsingThatRow
)
```

Chapter 6. How relationships from Tableau

In Tableau

When you define joins, you usually end up with a single flattened data source (at least conceptually), and calculations can often treat all columns as if they are in one big table.

In Power BI

- Tables like Edition and Q1_Sales stay as separate tables in the model.
- A relationship (for example, Edition[ISBN] 1 → many Q1_Sales[ISBN]) controls how filters flow between them.
- DAX then uses:
  - Filter context plus relationships, and
  - Row context plus functions like RELATED / RELATEDTABLE

It feels more like “linked tables with filter propagation,” not “one big joined table.”