



DAX for Power BI

Sample manual - first two chapters



TABLE OF CONTENTS (1 of 4)

1	GETTING STARTED	Page
1.1	Introducing DAX	6
	<i>Where is DAX used?</i>	6
	<i>How DAX is Used 1 - Calculated Columns</i>	6
	<i>How DAX is Used 2 – Measures</i>	7
	<i>How DAX is Used 3 – Queries</i>	7
1.2	The Construct-a-Creature Database	8
	<i>The Database Tables and Relationships</i>	8
1.3	Column Storage	9
	<i>Row versus Column Storage</i>	9
	<i>Data Compression</i>	9
	<i>Implications for Loading Data</i>	10

2	WRITING DAX	Page
2.1	Calculated Columns	11
	<i>Referring to Columns/Fields</i>	11
	<i>Referring to Tables</i>	12
	<i>Fully Qualified References</i>	12
2.2	Writing DAX	13
	<i>Laying out your Formulae</i>	13
	<i>Using Multiple Lines</i>	13
	<i>Pressing the TAB Key</i>	14
	<i>Comments</i>	14
2.3	DAX Syntax	15
	<i>Functions and Arguments</i>	15
	<i>Mathematical Operators</i>	16
	<i>Concatenating Text</i>	16

3	DAX STUDIO	Page
3.1	Using DAX Studio	17
	<i>Installing DAX Studio</i>	17
	<i>Connecting to your Data Model</i>	17
3.2	Five Uses of DAX Studio	18
	<i>Use 1 - Getting at DAX Functions</i>	18
	<i>Use 2 – Writing DAX Queries</i>	18
	<i>Use 3 – Better Formatting</i>	19
	<i>Use 4 – Saving DAX</i>	19
	<i>Use 5 – Getting at Internal Data</i>	19

4	TESTING CONDITIONS	Page
4.1	Testing Single Conditions	20
	<i>The IF Function</i>	20
	<i>Relational Operators</i>	20
	<i>Logical Operators</i>	21
	<i>Using IN to Test if Items Exist in a List</i>	21
4.2	The SWITCH Function	22

5	LINKING TABLES	Page
5.1	The RELATED Function	23
5.2	Dealing with Blanks	24
	<i>BLANK Arithmetic</i>	25
5.3	The RELATEDTABLE Function	26

6	TRAPPING ERRORS	Page
6.1	Using the DIVIDE Function	27
6.2	Using IFERROR	28
	<i>Generating your Own Errors using ERROR</i>	28

7	WORKING WITH DATA TYPES	Page
7.1	DAX Data Types	29
7.2	Scalar Date Functions	30
7.3	Scalar Text Functions	31
	<i>Finding and Replacing Text</i>	31
	<i>Converting Text</i>	31
	<i>Formatting Text</i>	32
	<i>Getting the Length of and Extracting Text</i>	32
7.4	Scalar Number Functions	33

8	MEASURES	Page
8.1	Introduction to Measures	34
	<i>What Measures Are</i>	34
	<i>Examples of Measures</i>	34
8.2	Creating a Measures Table	35
8.3	Creating Measures	36
8.4	Quick Measures	37
	<i>Starting a Quick Measure</i>	37
	<i>Creating the Base Value</i>	38
	<i>Setting any Filtering</i>	38
	<i>Using a Quick Measure</i>	39
8.5	DAX Aggregation Functions	40
8.6	Aggregating Expressions	41
	<i>The Problem</i>	41
	<i>Why the Simple Solution Won't Work</i>	41
	<i>The Answer – X-Suffix Functions</i>	42
	<i>Syntax of AggregateX Functions</i>	42
8.7	Calculating Ratios	43
	<i>Counting Rows using the COUNTROWS Function</i>	43
	<i>Creating Ratios: the Fields Needed</i>	43
	<i>The Final Matrix</i>	44
	<i>Summing Ratios Wouldn't Work</i>	44

TABLE OF CONTENTS (2 of 4)

9	FILTER CONTEXT	Page
9.1	Our Simple Example	45
9.2	How Filter Context Works	46
	<i>What We're Working Towards</i>	46
	<i>Step 1 – Assembling the Data</i>	47
	<i>Step 2 – Working out the Filter Context</i>	48
	<i>Step 3 – Getting the Filtered Data for the Context</i>	49
	<i>Step 4 – Aggregating the Data</i>	49

10	ROW CONTEXT	Page
10.1	Row Context for Calculated Columns	50
10.2	Iterator Functions	51
	<i>Normal Aggregate Functions Use Filter Context</i>	51
	<i>Iterator Functions Use Row Context</i>	52

11	THE CALCULATE FUNCTION	Page
11.1	Syntax of the CALCULATE Function	53
11.2	Removing a Constraint	53
	<i>Our Example</i>	54
	<i>A Quick Note on Ratios</i>	54
	<i>The Formula for this Example</i>	55
	<i>How this Works</i>	55
11.3	Removing Multiple Constraints	56
	<i>Using Multiple ALL Functions</i>	56
	<i>Using ALLEXCEPT</i>	57
11.4	Replacing a Constraint	58
	<i>Filter Context Revisited – Column Storage</i>	58
	<i>How Replacing Filter Context Really Works</i>	59
11.5	Using ALLSELECTED	60
11.6	Context Transition	61

12	VARIABLES	Page
12.1	Referring to Measures within Measures	62
12.2	Creating Variables	63
12.3	Lazy Evaluation and its Implications	64
12.4	Storing Tables in Variables	65
12.5	Debugging using Variables	66

13	THE FILTER FUNCTION	Page
13.1	The Basic FILTER Function	67
	<i>Using CALCULATE as an Alternative to FILTER</i>	67
13.2	FILTER as an Iterator Function	68
	<i>Starting Off – Our Example</i>	68
	<i>Getting the Filter Context</i>	68
	<i>Row Context within this Filter Context</i>	69
	<i>Deriving the Final Result</i>	69
13.3	Multiple Conditions in FILTER Functions	70
	<i>Combining Conditions using && and </i>	70
	<i>Combining Conditions using AND / OR</i>	70
	<i>Combining Conditions by Nesting the FILTER Function</i>	71
13.4	Using ALL and FILTER	72
13.5	FILTER and CALCULATE aren't Equivalent	72

14	THE VALUES FUNCTION	Page
14.1	Introducing the VALUES Function	74
14.2	Detecting the Number of Values	75
	<i>The HASONEVALUE Function</i>	75
	<i>Using COUNTROWS to Count VALUES</i>	76
14.3	Using VALUES to Modify Filter Context	77
	<i>The Obvious Way doesn't Work</i>	77
	<i>Using the VALUES Function to Solve the Problem</i>	77
14.4	Parameter Tables	78
14.5	Dynamic Titles using ISFILTERED	79
	<i>Dynamic Titles for Single-Value Filters</i>	79
	<i>Dynamic Titles for Multi-Value Filters</i>	80

15	CALENDAR TABLES	Page
15.1	What are Calendar Tables?	81
	<i>Requirements for a Calendar Table</i>	81
	<i>Why you Need a Calendar Table</i>	81
15.2	Creating a Calendar	82
	<i>Step 1 – Getting the Calendar Data</i>	82
	<i>Step 2 – Loading and Linking to the Calendar Table</i>	83
	<i>Step 3 – Mark your Table as a Date Table</i>	83
	<i>Step 4 – Setting the Year as Text</i>	84
	<i>Step 5 - Setting a Sort Month</i>	84
15.3	Date Granularity	85
15.4	Special Days	86

TABLE OF CONTENTS (3 of 4)

16	MULTIPLE DATE TABLES	Page
16.1	The Problem, and Two Solutions	87
	<i>Repeat the Table or the Relationship?</i>	87
16.2	Solution One: Duplicate the Calendar Table	88
	<i>Step 1 - Importing and Linking to the Calendar Tables</i>	88
	<i>Step 2 – Renaming Tables and Fields</i>	89
	<i>Step 3 – Using your Multiple Calendars</i>	89
16.3	Solution Two: Duplicate the Relationship	90
	<i>Creating the Duplicate Relationships</i>	90
	<i>Interlude - The CALCULATETABLE Function</i>	91
	<i>The USERELATIONSHIP Function</i>	91
	<i>Our Measures</i>	92
16.4	CROSSFILTER Function	93
	<i>One Solution – Change the Relationships Permanently</i>	93
	<i>A Better Solution – Use DAX to Temporarily Cross-Filter</i>	94
	<i>Multiple Cross-Filtering</i>	94

17	HOW TIME INTELLIGENCE FUNCTIONS WORK	Page
17.1	Our Example	95
17.2	Filter Context Reminder	96
17.3	Year-to-Date using CALCULATE	97
17.4	Year-to-Date using Time-Intelligence Functions	98
	<i>The DATESYTD Function</i>	98
	<i>The TOTALYTD Function</i>	98

18	DAX DATE FUNCTIONS	Page
18.1	Contents of the Chapter	99
18.2	Period to Date	100
	<i>Using DATESYTD, DATESQTD and DATESMTD</i>	100
	<i>Using TOTALYTD, TOTALQTD and TOTALMTD</i>	100
18.3	Changing the Financial Year End	101
	<i>Functions with a Year End Date Argument</i>	101
	<i>Displaying Data for Different Financial Year Ends</i>	102
18.4	Referencing Previous Periods	103
	<i>The SAMEPERIODLASTYEAR Function</i>	103
	<i>The DATEADD Function</i>	103
18.5	Parallel Periods	104
18.6	Moving Averages	105
	<i>Definition of a Moving Average</i>	105
	<i>Moving Average using DATESINPERIOD and LASTDATE</i>	106
	<i>Moving Average using DATESBETWEEN, NEXTDATE and LASTDATE</i>	106
18.7	Semi-Additive Measures	107
	<i>Useful Semi-Additive Functions</i>	107
	<i>Using the FIRSDATE and LASTDATE Functions</i>	107
	<i>Using FIRSTNONBLANK and LASTNONBLANK</i>	108
	<i>Detecting Relationships in FIRSTNONBLANK / LASTNONBLANK</i>	108

19	RANKING	Page
19.1	The RANKX Function	109
	<i>Syntax of the Rank Function</i>	109
	<i>Intellisense for the RANKX Function</i>	109
19.2	RANKX for Calculated Columns	110
19.3	Ranking Measures (Existing Columns)	111
	<i>The Most Common Problem – Omitting ALL</i>	111
	<i>The Solution using ALL</i>	111
19.4	Ranking using Aggregate Calculations	112
	<i>RANKX is an Iterator Function</i>	112
19.5	Ranking with Context	113
	<i>Suppressing Totals</i>	113
	<i>Ranking over Selected Items</i>	113

TABLE OF CONTENTS (4 of 4)

20	THE EARLIER FUNCTION	Page
20.1	Case Study of the EARLIER Function	114
	<i>Our Example</i>	114
	<i>An Outline of the EARLIER Function</i>	114
	<i>Row Context within Filter Context</i>	115
	<i>The Final Formula</i>	116
20.2	Another Example – Running Totals	117
20.3	Using Variables instead of the EARLIER Function	118
	<i>Ranking Sales using Variables</i>	118
	<i>Running Totals using Variables</i>	118

21	BANDING	Page
21.1	What is Banding?	119
	<i>Creating and Loading a Banding Table</i>	119
21.2	Creating a Banding Formula	120
21.3	Sorting the Bands	121

22	PARENT-CHILD HIERARCHIES	Page
22.1	What is a Parent-Child Hierarchy?	122
22.2	Creating a Parent-Child Hierarchy	123
	<i>Step 1 – Create a List of Parent Ids (the PATH Function)</i>	123
	<i>Step 2 – Working out the Path Depth (the PATHLENGTH Function)</i>	123
	<i>Step 3 – Create a Measure Showing the Number of Levels</i>	123
	<i>Step 4 - Finding Managers at Each Level (PATHITEM and LOOKUPVALUE)</i>	124
	<i>Step 5 – Creating a Hierarchy</i>	125
	<i>Step 6 – Creating your Visual</i>	125

CHAPTER 1 - GETTING STARTED

1.1 Introducing DAX

The *DAX* language (standing loosely for *Data Analysis eXpressions*) allows you to create calculated columns, measures and queries (an example of each is shown in this section).

Where is DAX used?

You can write DAX within the following programs:

Program	Notes
<i>Power BI</i>	Power BI is a standalone application which allows you to create business intelligence reports, and publish them to a website or server.
<i>PowerPivot</i>	PowerPivot is an add-in within Excel which allows you to combine data from multiple data sources, and present this in a pivot table.
<i>SQL Server Analysis Services (Tabular)</i>	SSAS Tabular allows you to combine data from lots of different data sources, apply security to it to control who sees what and then allow employees of your organisation to share the resulting data model.



DAX initially looks similar to Excel, but you will quickly realise that it is actually very different!

How DAX is Used 1 - Calculated Columns

A *calculated column* is like a formula in Excel:

PurchaseDate	ProductId	CentreId	Quantity	Price	Sales value
21 July 2021	4	367	1	4.99	4.99
21 July 2021	19	244	1	3.99	3.99
21 July 2021	4	375	1	4.99	4.99
21 July 2021	10	101	1	3.99	3.99

This DAX calculated column gives the sales value for each row of a purchases table, by multiplying the quantity of items bought by the price paid.

As we will see in this manual, a calculated column is evaluated for each row of a table; DAX creates a *row context* for each separate calculation.

How DAX is Used 2 – Measures

Much of this courseware will be devoted to creating *measures* like this one, to calculate the value of sales for any region, year or other constraint:

Measures use the same language, but are usually more complicated (and always involve some aggregation).

This measure is showing that total sales for the **East Anglia** region and **Air** environment are £1,268.71. As we will see, each measure is calculated for a particular combination of constraints called the *filter context*.

```

1 Total sales = SUMX(
2
3 // sum the value of all sales for each
4 // cell in the current filter context
5 Purchase,
6 [Price] * [Quantity]
7 )
    
```

RegionName	Air	Land	Water	Total
East Anglia	1,268.71	13,640.28	2,050.91	16,959.90
East Midlands	2,609.87	26,289.48	5,033.67	33,933.02
London	4,899.45	54,018.62	9,136.50	68,054.57
North	2,874.62	25,980.55	4,334.49	33,189.66
North West	7,950.17	69,751.18	12,614.71	90,316.06
South East	10,697.97	102,171.04	17,507.12	130,376.13
South West	2,616.48	23,769.80	4,506.69	30,892.97
West Midlands	4,702.33	49,116.82	8,878.73	62,697.88
Yorkshire & Humberside	6,231.78	47,396.27	8,509.78	62,137.83
Total	43,851.38	412,134.04	72,572.60	528,558.02

How DAX is Used 3 – Queries

As well as for creating calculated columns and measures, you can also use DAX to create queries to get data out of a Power BI, PowerPivot or SSAS Tabular data model:

```

1 // list out the regions
2 EVALUATE
3 Region
4 ORDER BY
5 Region[RegionName]
6
    
```

RegionId	RegionName
1	East Anglia
2	East Midlands
3	London
4	North
5	North West
6	South East
7	South West
8	West Midlands
9	Yorkshire & Humberside

The DAX query language is similar to SQL, but whereas SQL is used to get data out of a relational database, DAX is used to get data out of a data model.

This query lists out all of the regions in a data model, alphabetically by region name.

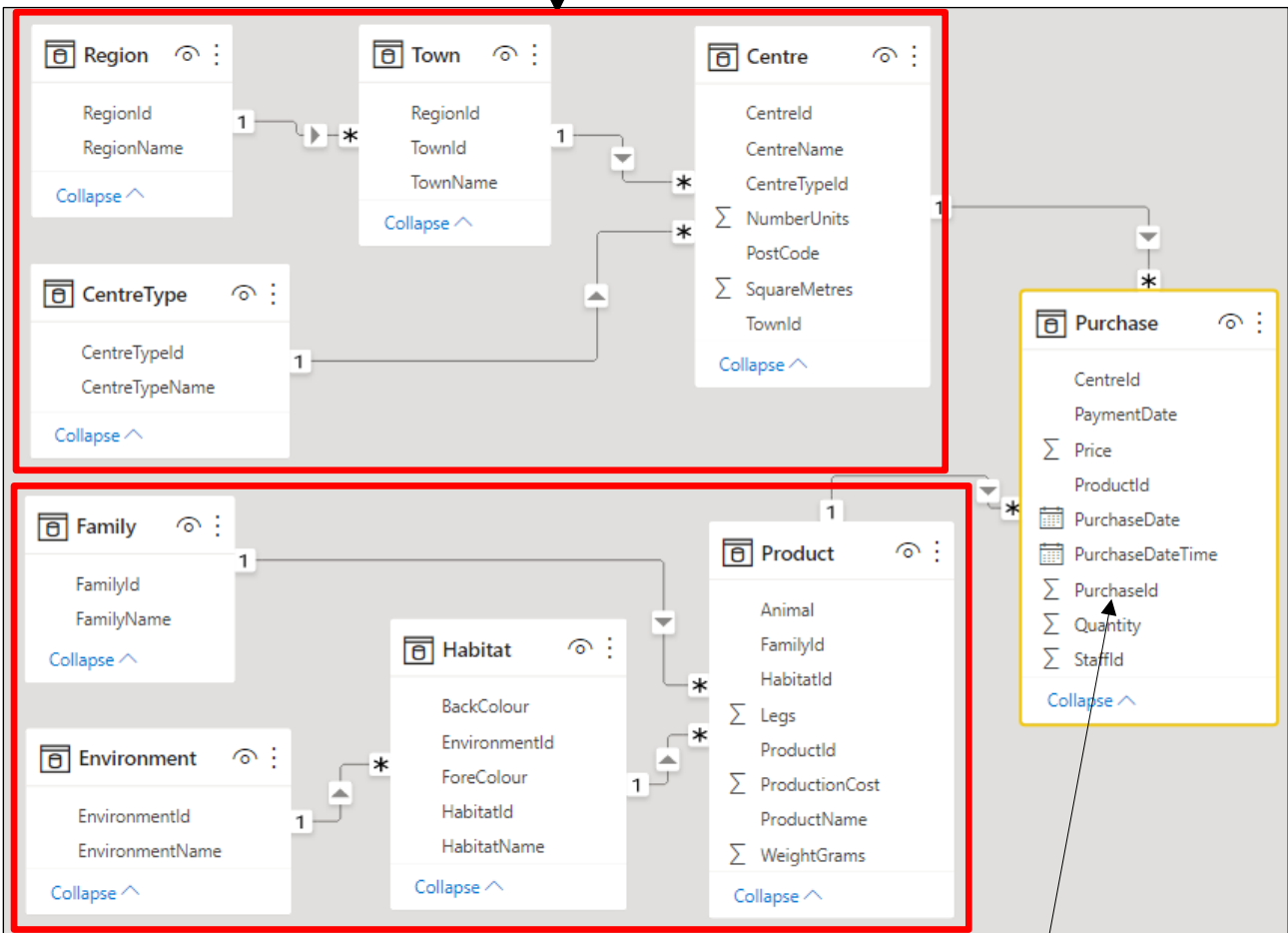
1.2 The Construct-a-Creature Database

This courseware uses data from the (fictitious) Wise Owl subsidiary *Construct-a-Creature* (a retail chain loosely modelled on *Build-a-Bear*, but with a wider range of animals available for purchase).

The Database Tables and Relationships

Here are the tables in the *Construct-a-Creature* (CAC) database:

There are four geographical tables to do with where purchases took place, giving the shopping centre, type of shopping centre, town and region.



There are also four tables to do with the biological classification of each animal (for example, a frog is an amphibian which lives in a fresh water habitat in a watery environment).

The purchases table specifies how many of each product were bought in each location in each transaction.

1.3 Column Storage

To understand how DAX works in Power BI, and how to tweak it, it's vital to understand how the underlying engine (called either *VertiPaq* or *xVelocity*, depending on what you read) stores data.

Row versus Column Storage

Most databases (such as SQL Server) use a row-based storage algorithm:

Typically each row in a table is stored as a record, and is accessed by its primary key (unique identifier).

ProductId	ProductName	Animal	HabitatId	Legs	FamilyId
1	Sammy	Snake	1	0	1
2	Pokyo	Penguin	4	2	3
3	Fenella	Frog	3	4	4
4	Layla	Lemur	2	2	5

Power BI models, by contrast, store data by column:

In Power BI columns are stored separately, which makes any calculation summing or otherwise aggregating this column run much more quickly.

ProductId	ProductName	Animal	HabitatId	Legs	FamilyId
1	Sammy	Snake	1	0	1
2	Pokyo	Penguin	4	2	3
3	Fenella	Frog	3	4	4
4	Layla	Lemur	2	2	5
5	Dave	Dachsund	1	4	5
6	Kylie	Camel	5	4	5
7	Jeremy	Jackdaw	7	2	3
8	Faye	Fox	6	4	5

Data Compression

Duplicate column values are only ever stored once. Thus the **FamilyId** and **Legs** columns above might be stored something like this:

Column	Dictionary	Values
FamilyId	1,2,3,4,5,6	0,2,3,4,4,4,2,4,2,1,4,5,4,2,0,3,4,4,4
Legs	0,2,4,6	0,1,2,1,2,2,1,2,1,0,2,3,2,1,0,2,2,2,2



This shows that the lower the cardinality of a column (ie the smaller the number of distinct values there are, and hence the more duplication there is), the more efficiently the data will be stored.

Implications for Loading Data

What column storage implies is that you should avoid loading columns with high cardinality (that is, with very little repeated data) unless you need them:

Navigator

Display Options ▾

- Construct a creature.xlsx [9]
 - Centre
 - CentreType
 - Environment
 - Family
 - Habitat
 - Place
 - Product
 - Purchase
 - Region

Purchase

PurchaseId	PurchaseDate	PurchaseDateTime	ProductId	CentreId	Quantity
2	17/12/2015	17/12/2015 14:30:00	14	14	94
10	21/12/2015	21/12/2015 18:05:00	14	14	75
13	22/12/2015	22/12/2015 11:41:00	14	14	67
15	23/12/2015	23/12/2015 16:05:00	14	14	75
16	23/12/2015	23/12/2015 16:15:00	2	2	319
17	27/12/2015	27/12/2015 09:33:00	14	14	361
18	27/12/2015	27/12/2015 12:02:00	2	2	307
22	29/12/2015	29/12/2015 17:42:00	1	1	380
35	04/01/2016	04/01/2016 18:29:00	14	14	363
48	05/01/2016	05/01/2016 09:52:00	14	14	363
54	05/01/2016	05/01/2016 10:13:00	14	14	361
58	05/01/2016	05/01/2016 12:28:00	14	14	361

You should avoid loading the **PurchaseId** column. It isn't used to link to any other table, and it has the highest possible cardinality (each number is unique) so will take up a lot of memory.

The other column to avoid loading, since each time of day is stored as a separate number internally (unless, of course, you want to analyse purchases by the time of day when they occurred).



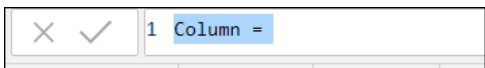
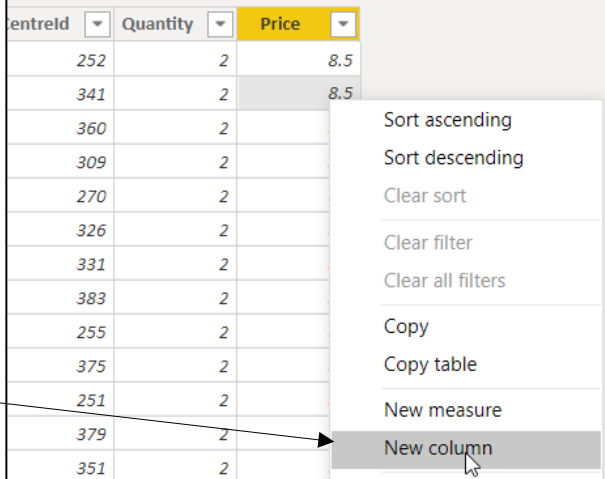
Note that for the example above you have to import the **ProductId** and **CentreId** columns because they are used to link to other tables.

CHAPTER 2 - WRITING DAX

2.1 Calculated Columns

The simplest way to write DAX is as a *calculated column*:

Right-click on a column and choose to insert a new one as here, then start typing into the formula bar which appears:

entrelid	Quantity	Price
252	2	8.5
341	2	8.5
360	2	
309	2	
270	2	
326	2	
331	2	
383	2	
255	2	
375	2	
251	2	
379	2	
351	2	

Referring to Columns/Fields

The easiest way to create a formula in DAX is to use the keyboard:

Sales = [

- [Centrelid]
- [Price]
- [ProductId]
- [PurchaseDate]
- [PurchaseDateTime]
- [PurchaseId]
- [Quantity]

a) After typing a name for your new column, type the opening square bracket symbol [.

b) Type in the first letter of the field you want to use (here q).

Sales => [q]

c) Either double-click on the field name to insert it, or press **TAB**.

Sales => [Quantity]

Sales = [Quantity] * [Price]

d) Continue in this way to build up the formula you want to create.

1 Sales = [Quantity] * [Price]

PurchaseDate	ProductId	Centrelid	Quantity	Price	Sales
21 July 2021	4	367	1	4.99	4.99
21 July 2021	10	223	1	3.99	3.99
21 July 2021	4	375	1	4.99	4.99

e) When you press **ENTER** to create your formula, DAX will calculate it and show the results for every row in the table.

Referring to Tables

If you want to refer to a table, the easiest way to do it is to type in the `'` apostrophe character:

Example = `sumx(`

SUMX(**Table**, Expression)
Returns the sum of an expression evaluat

- ADDCOLUMNS
- ADDMISSINGITEMS
- ALL
- ALLEXCEPT
- ALLNOBLANKROW

When an expression calls for a table ...

... type an apostrophe symbol to bring up a list of just the table names.

Example = `SUMX('`

SUMX(**Table**, Expression)
Returns the sum of an expr

- 'Centre'
- 'CentreType'
- 'Environment'
- 'Family'
- 'Habitat'
- 'Product'
- 'Purchase'
- 'Region'
- 'Town'



Although the method above makes it easy to insert a table name into a formula, the apostrophe characters are optional, and most people miss them out. The exception to this is when your table name is also a reserved word (for example, **Calendar** is a table name which you would have to type as **'Calendar'**).

Fully Qualified References

You can always refer to a column using its full reference:

=TableName[ColumnName]

However, you can often miss out the table name where it is unambiguous from the context. So both of these calculated columns will work:

Here we haven't specified the table name, so DAX assumes that it is the current one.

Sales = [Quantity] * [Price]

Productid	Centreid	Quantity	Price	Sales
4	367	1	4.99	4.99
10	223	1	3.99	3.99

Here, by contrast, we have included the table name, even though it wasn't necessary.

Sales = Purchase[Quantity] * Purchase[Price]

Productid	Centreid	Quantity	Price	Sales
4	367	1	4.99	4.99
10	223	1	3.99	3.99



It's probably best practice always to fully qualify column references in DAX (although the author confesses to frequently taking the lazy way out and omitting them where they're not needed).

2.2 Writing DAX

The more you get into DAX, the longer and more complicated your formulae will become – and the more important it will be to format and comment them properly!

Laying out your Formulae

DAX formulae can quickly become quite long, and hard to read. You can make formulae easier to interpret by *indenting* arguments to functions:

```
Size = IF(
  [SquareMetres] > 10000,
  "Large",
  "Small"
)
```

Using multiple lines and indenting code together make formulae easier to read.

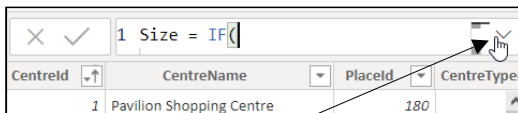


Wise Owl's Hint

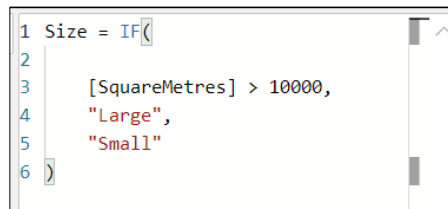
When you copy DAX formulae from Power BI you lose any colours. Because of this all of the formulae in this courseware are copied from DAX Studio, a separate standalone DAX editor. As a result the colours shown will have slightly different shading to those you'll see in Power BI.

Using Multiple Lines

You can use this drop down arrow to give yourself more space to work in:



Click on this drop arrow to give yourself more room for typing.



Too much room, sometimes! Click again to collapse the space.

You can also use the following keys to add carriage returns into a formula:

Key	What it does
Shift +	Add a new line, and also an indentation level if appropriate.
Alt +	Add a new line, but don't indent it.



Wise Owl's Hint

Irritatingly, the one key which doesn't work is just pressing **Enter**. Instead, this makes Power BI create your formula, even if you haven't finished it. You'll then have to sort out the brackets Power BI has thoughtfully added at the end of the formula to make your parentheses balance out!

Pressing the TAB Key

When you've typed in a function or field, the best key that you can press is `Tab`. This is true even if you've typed in the whole name of a function:

SizeVerdict = if

IF(LogicalTest, ResultIfTrue, ResultIfFalse)
Checks whether a condition is TRUE and returns one value if TRUE, and another value if FALSE.

DATEDIFF
IF
IFERROR

Here we've typed in the full function name **IF**, but it's in lower case and we need a bracket to follow it.

Pressing the **TAB** key will solve both problems with one keystroke!

SizeVerdict = IF(

IF(LogicalTest, ResultIfTrue, ResultIfFalse)
Checks whether a condition is TRUE and returns one value if TRUE, and another value if FALSE.

Comments

You can't insert comments at the start of DAX formulae:

```
1 // this won't work
2 Size = IF(
3     [SquareMetres] > 10000,
4     "Large",
5     "Small"
6 )
```

// this won't workSize

This formula won't work, because even though the comment syntax is valid DAX will take the comment as the start of the formula, and name the column accordingly!

You can, however, insert them anywhere else, using one of 3 different syntaxes:

You can add comments at the end of formulae using one of these three syntaxes (although until you learn DAX variables it's not obvious why you'd want to do this).

```
Size = IF(
    // condition (bigger than
    // 20,000 square metres)
    [SquareMetres] > 20000,

    // return LARGE if a centre is big
    "Large",

    // otherwise return SMALL
    "Small"
)
```

You can also (more usefully) put comments between arguments – any line beginning with `//` or `--` will be ignored.

```
Size = IF(
    [SquareMetres] > 20000,
    "Large",
    "Small"
)
// this formula categories
// shopping centres by size

-- you can also write comments
-- like this

/*
or even with long comments
like this
*/
```

2.3 DAX Syntax

This section explains the rules you have to follow when creating DAX formulae.

Functions and Arguments

When you type any DAX function, Intellisense will tell you the arguments you need to specify. Here's an example:

A simple DAX function, returning different values if something is true or false.

SizeVerdict = IF(

entreNar	Placeld
million Shop	

IF(**LogicalTest**, ResultIfTrue, [ResultIfFalse])
 Checks whether a condition is met, and returns one value if TRUE, and another value if FALSE.

Here are the *arguments* to this function (the bits of information that you need to specify):

Argument name	Status	What it should contain
LogicalTest	Compulsory	A test to perform to see if something is true or not
ResultIfTrue	Compulsory	What to return if the test returns true
ResultIfFalse	Optional	What to return if the test returns false



You can tell whether an argument is compulsory or optional by seeing whether it is enclosed in square brackets [like this].

Mathematical Operators

You can use the following standard mathematical symbols in DAX expressions:

Symbol	What it means	Example
+ / -	Addition / subtraction	= 3 + 5 - 2 would return 6
* / /	Multiplication / division	= 2 * 6 / 3 would return 4
^	Raising to the power of	= 2 ^ 3 would return 8

Standard rules of arithmetic (BODMAS) apply: so $2 + 3 * 5$ would return 17, since the multiplication would take precedence over the addition.



*Division and multiplication take equal precedence, and are read from left to right. So $15 / 3 * 2$ would return 10, not 2.5.*

Concatenating Text

There are two ways to join text together. You can either use the `&` symbol:

1	StaffName = [FirstName] & " " & [LastName]		
FirstName	LastName	DateJoined	StaffName
Leah	Menzies	13 August 2016	Leah Menzies
Lara	Bhangu	26 March 2011	Lara Bhangu
Suzanna	Pederson	28 August 2012	Suzanna Pederson
Debbie	Scott	03 December 2001	Debbie Scott








































This formula would join together the first name and last name fields, with a space in between.

Or alternatively, the **CONCATENATE** function (although this doesn't work well when you have more than two things that you want to join together):

1	StaffName = CONCATENATE(CONCATENATE([FirstName], " "), [LastName])			
FirstName	LastName	DateJoined	StaffName	
9	Leah	Menzies	13 August 2016	Leah Menzies
1	Lara	Bhangu	26 March 2011	Lara Bhangu
0	Suzanna	Pederson	28 August 2012	Suzanna Pederson

Unlike in Excel, to join more than two things together you have to repeat the function name, making for a messy formula.

What we do!

		Basic training	Advanced training	Systems / consultancy
Office	Microsoft Excel			
	VBA macros			
	Office Scripts			
	Microsoft Access			
Business Intelligence	Power BI			
	Power Apps			
	Power Automate / PAD			
SQL Server	SQL			
	Reporting Services			
	Report Builder			
	Integration Services			
	Analysis Services			
Coding	Visual C# programming			
	VB programming			
	DAX			
	Python			



WiseOwl
Training

