PREVIEW

SECOND EDITION

DAX PATTERNS

The most comprehensive collection of ready-to-use solutions in DAX for Power BI, Analysis Services, and Power Pivot

Alberto Ferrari
Marco Russo

sqlbi

Introduction

At SQLBI we have a beautiful job: we are world-wide trainers and consultants. We meet thousands of people all over the world every year: a crowd of very diverse persons, sharing the same passion for Business Intelligence and DAX. We are asked to solve scenarios of various complexity by our students and customers.

Say a student approaches you because they need to compute the number of new customers for their report. You solve the problem once, twice, three times… And at some point, you feel that the next time you need to answer the same question, you would love to have a ready-to-use solution. This is the reason we started the daxpatterns.com website in 2013. We started collecting patterns that repeat themselves. We created a collection of DAX formulas aimed at solving the most frequently-asked questions we receive. At that time, the goal was not to write a new book. Instead, our goal was to create some sort of memory bank for the solutions we would find. We thought we would be the main users of our own website.

As is often the case, real-life does not go according to plan. This time, for the better. The website had a tremendous success. Users downloaded the samples and achieved two different goals: they found a ready-to-use solution to their problems, and they improved their DAX skills based on the formulas we authored. Because of the different file formats, we included samples for Excel 2010 and Excel 2013 – the latter still works with later versions of Excel. Eventually, we collected the content of the website into a book. That was the first edition of DAX Patterns. It was at the end of 2015. At the time, we had not yet published the first edition of The Definitive Guide to DAX. Therefore, we included a short introduction to DAX in the DAX Patterns book.

Many things changed over the following five years. DAX evolved with many useful features. Most importantly, Power BI hit the market and the number of users adopting DAX grew at an exponential rate. Today, most of the DAX users create a Power BI solution. When we published the first edition of this book, Power BI had not even been announced yet.

During these five years, the process of collecting patterns continued. We met more students, we solved more problems, we also got better and better at DAX. Plus, we now had thousands of users who were able to provide feedback on previous patterns. Studying user comments gave us a better picture of what our readers needed. In parallel, we went on to publish two editions of The Definitive Guide to DAX. At that point, there was no longer a reason to be teaching DAX in a book about patterns.

Long story short, it started to make a lot of sense to author a new version of both the DAX Patterns website and book. We rolled up our sleeves and created the book you are reading right now.

We did not use any of the content from the previous book. We wanted a fresh start. The entire library of code is rewritten from scratch, using the latest DAX and Power BI features and adapting the code to Excel 2019 when necessary.

In this new edition we made several choices:

- We greatly increased the share of the book dedicated to time intelligence calculations. Time intelligence is by far the most widely studied topic. Therefore, it made sense to increase the number of time-related calculations and patterns.
- Similarly, the New and returning customers pattern was an absolute hit. We gave that pattern a bigger share of the book as well, increasing the number of formulas and models to compute new and returning customers.
- We increased the number of patterns, adding several that – in our experience – are likely to be useful to our readers.
- We decided to cut out a few patterns. For example, the chapter about statistical calculations was useful back in 2015, because of the lack of statistical functions in DAX. Since then, DAX introduced many new functions to compute the formulas that were explained in that chapter. There is no need for that content in 2020.
- We no longer provide code snippets. In the previous book, most of the code was shown including placeholders for the columns that readers were likely to change. We no longer do that. We show code that works, because you often have to adapt the data model and other details in the formula. We felt this would make the code more readable and easier to use and to adapt to your model.
- We optimized every single formula. All the code you see in these patterns has been thoroughly reviewed for performance. This is not to say that these patterns are the very best. They are the best we could come up with. If you can make the code better and faster, let us know! The comment sections on the website are the right place to provide your feedback.
- We created a Power BI and an Excel version of each sample file. In the book, we include pictures of Power BI reports showing the results of the code, but the examples you can download are available in both formats: Power BI and Excel.
- We improved the readability of the eBook version of DAX Patterns. This meant keeping the code formatting intact regardless of the eBook reader size.

Why we published this book

If you are wondering what the differences may be between the content of this book and the content published on daxpatterns.com, we want to assure you that there are no differences. Should you buy the book to obtain extra content? No. The access to the web site is free, where you can read the same content as what you will find in this book and download the sample files for free.

That said, if you enjoy having an offline copy of the patterns, if you enjoy having a printed version, if you would like to have it in your eBook collection, then you should purchase it. This way, you help us keep the business up and running. We were surprised with the number of people who purchased the first edition. This motivated us to further invest into this new version of the website and the pattern. We hope the process will continue!

By visiting the daxpatterns.com website, you will also see that we have recorded a video for each pattern. This is where we go into more depth on how to use the patterns and how the formulas work. These videos are for sale. You can buy all of them, or just the pattern you want to study more. It is an additional service that many people have been asking for; we know some prefer the book, some prefer the video, and many people want both!

How to use this book

What will you find in this book? Each standalone chapter covers a separate pattern and can be read without having read the others. You can read the Currency conversion pattern without having ever looked at the Basket analysis, or at any of the time-related calculations.

Each chapter about a pattern starts with a brief description of the business scenario; it then goes into a more complete description of the solution, along with all the DAX code that needs to be implemented in order to solve the scenario. We kept the description of the code short, using comments in the code to document the measures where needed.

You need separate companion content for the book. At the beginning of each chapter, a short URL points to the corresponding pattern on the daxpatterns.com website. You can download the sample files for Power BI and Excel from the website.

The book is intended to be used as a reference. When you want to implement a pattern, you do not want to read long descriptions: you want to see the code and the reason for it. Therefore, we kept it as compact as possible, keeping the spotlight on the DAX code.

That said, if you want to implement a pattern we strongly suggest that you read the entire chapter before implementing any code. The reason is that we sometimes present multiple solutions and you need to choose the best for your specific scenario. For each pattern we also provide the demo files both in Power BI and Power Pivot for Excel. Sometimes the code of the two versions is slightly different. The book always presents the Power BI solution, which is using the latest features of DAX at the time of printing. Some of those features are not available in Power Pivot – like calculated tables. This is the main reason for the differences.

There is only one exception: time-related calculations. As we said, we gave the time-related calculations more space in the book: we now present four different patterns for time-related calculations. Each of these four patterns is huge. Together, they represent more than 40% of this book. This is why we created an introductory chapter to the time-related calculations, which aims to help you choose the right pattern for your scenario. If you need to implement time-related calculations, make sure to read the introduction first, and then the full chapter covering the pattern you decide to use.

Prerequisites

One word of advice to our readers: this book does not teach DAX.

You are expected to already know DAX to make the best use of these patterns. Most of the patterns show advanced DAX techniques that you are welcome to study and use in your solutions. By reading this book you will not learn DAX. But if you already know DAX, you will likely become a better DAX developer.

We suggest that you use these patterns with the latest version of Power BI or Excel, because DAX evolves and improves over time. We tested the patterns on Power BI June 2020, Excel 2019, and Excel for Microsoft 365 version 2006. Most of the patterns work with earlier versions of Power BI and Excel, but we cannot guarantee this because we did not thoroughly test for all the previous versions.

Acknowledgments

Last, but not least: the acknowledgments section.

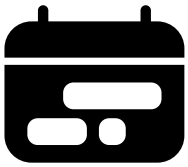
The most important person we want to thank is you. This work was made possible by the discussions we have had over time with readers, users, customers, and students like yourself. Therefore, even without knowing it you have contributed to this content; and if you post comments in our public forums, you will be contributing further.

That said, there are some people who directly contributed to the entire writing process: Daniil Maslyuk meticulously reviewed each pattern, found all the errors we had made and provided invaluable feedback. Claire Costa reviewed our English grammar and readability, making the book more precise and enjoyable. Sergio Murru built the Excel versions of the sample files, which made the patterns available also to Power Pivot for Excel users. Daniele Perilli is the reason behind the book and the website being as beautiful as they are. We are responsible for the content and for any mistake, but if you can read accurate numbers, in good English, in both Excel and Power BI, and with a gorgeous overall presentation, it is thanks to them.

Enjoy DAX!

CHAPTER 5

Custom time-related calculations



Download sample files: **https://sql.bi/dax-204**

This pattern shows how to compute time-related calculations like year-to-date, same period last year, and percentage growth using a custom calendar. This pattern does not rely on DAX built-in time intelligence functions. All the measures refer to the fiscal calendar because the same measures, with a regular Gregorian calendar, can be obtained using the **Standard time-related calculations** pattern.

There are several scenarios where the DAX built-in functions for time intelligence cannot provide the right answers. For example, if your fiscal year starts on a month other than January, April, July, or October, then you cannot use the DAX time intelligence functions for quarterly-related calculations. In these scenarios, you need to rewrite the time intelligence logic of the built-in functions by using plain DAX functions like FILTER and CALCULATE. Moreover, you must create a *Date* table that contains additional columns to compute time periods like the previous quarter or a whole year. Indeed, the standard time intelligence functions derive this information from the *Date* column in the *Date* table. The custom time-related calculations pattern does not extract the information from the *Date* column and requires additional columns.

The measures in this pattern work on a regular Gregorian calendar with the following assumptions:

- Years and quarters always start on the first day of a month.
- A month is always a calendar month.

In simpler words, this pattern works fine if the fiscal year starts on the first day of a month, and a quarter is made of three regular months. For example, if the fiscal year starts on March 3, or all the fiscal quarters must have 90 days, then the formulas do not work.

An example of a calendar that does not satisfy the requirements of this pattern is a week-based calendar. If you need calculations over periods based on weeks, you should use the **Week-related calculations** pattern.

Introduction to custom time intelligence calculations

The custom time intelligence calculations in this pattern modify the filter context over the *Date* table to

obtain the required result. The formulas are designed to apply filters to the lowest granularity required to improve query performances. For example, a calculation over months works by modifying the filter context at the month level, instead of the individual dates. This technique reduces the cost of computing the new filter and applying it to the filter context. This optimization is especially useful when using DirectQuery, even though it also improves performance on models imported in memory.

Because the pattern does not rely on the standard time intelligence functions, the *Date* table does not have the requirements needed for standard DAX time intelligence functions.

For example, the Mark as Date Table setting is suggested, but not required. The formulas in this pattern do not rely on the automatic REMOVEFILTERS applied over the *Date* table when the *Date* column is filtered. Instead, the *Date* table must contain specific columns required by the measures. Therefore, although you might already have a *Date* table in your model, you must read the next section (Building a *Date* table) to verify that all the required columns are present in the *Date* table.

Building a *Date* table

The *Date* table used for custom time-related calculations is based on the months of the standard Gregorian calendar table. If you already have a *Date* table, you can import the table and – if necessary – extend it to include a set of columns containing the information required by the DAX formulas. We describe these columns later in this section.

If a *Date* table is not available, you can create one using a DAX calculated table. As an example, the following DAX expression defines the *Date* table used in this pattern, which has a fiscal year starting on March 1:

Calculated table

```
Date =
VAR FirstFiscalMonth = 3  -- First month of the fiscal year
VAR FirstDayOfWeek = 0    -- 0 = Sunday, 1 = Monday, ...
VAR FirstSalesDate = MIN ( Sales[Order Date] )
VAR LastSalesDate = MAX ( Sales[Order Date] )
VAR FirstFiscalYear =     -- Customizes the first fiscal year to use
    YEAR ( FirstSalesDate )
    + 1 * ( MONTH ( FirstSalesDate ) >= FirstFiscalMonth && FirstFiscalMonth > 1)
VAR LastFiscalYear =      -- Customizes the last fiscal year to use
    YEAR ( LastSalesDate )
    + 1 * ( MONTH ( LastSalesDate ) >= FirstFiscalMonth && FirstFiscalMonth > 1)
RETURN
GENERATE (
    VAR FirstDay =
        DATE (
            FirstFiscalYear - 1 * (FirstFiscalMonth > 1),
            FirstFiscalMonth,
            1
        )
```

```
    VAR LastDay =
        DATE (
            LastFiscalYear + 1 * (FirstFiscalMonth = 1),
            FirstFiscalMonth, 1
        ) - 1
    RETURN
        CALENDAR ( FirstDay, LastDay ),

    VAR CurrentDate = [Date]
    VAR Yr = YEAR ( CurrentDate )        -- Year Number
    VAR Mn = MONTH ( CurrentDate )       -- Month Number (1-12)
    VAR Mdn = DAY ( CurrentDate )        -- Day of Month
    VAR DateKey = Yr*10000+Mn*100+Mdn
    VAR Wd =                             -- Weekday Number (0 = Sunday, 1 = Monday, ...)
        WEEKDAY ( CurrentDate + 7 - FirstDayOfWeek, 1 )
    VAR WorkingDay =                     -- Working Day (1 = working, 0 = non-working)
        ( WEEKDAY ( CurrentDate, 1 ) IN { 2, 3, 4, 5, 6 } )
    VAR Fyr =                            -- Fiscal Year Number
        Yr + 1 * ( FirstFiscalMonth > 1 && Mn >= FirstFiscalMonth )
    VAR Fmn =                            -- Fiscal Month Number (1-12)
        Mn - FirstFiscalMonth + 1 + 12 * (Mn < FirstFiscalMonth)
    VAR Fqrn =                           -- Fiscal Quarter (string)
        ROUNDUP ( Fmn / 3, 0 )
    VAR Fmqn =
        MOD ( FMn - 1, 3 ) + 1
    VAR Fqr =                            -- Fiscal Quarter (string)
        FORMAT ( Fqrn, "\Q0" )
    VAR FirstDayOfYear =
        DATE ( Fyr - 1 * (FirstFiscalMonth > 1), FirstFiscalMonth, 1 )
    VAR Fydn =
        SUMX (
            CALENDAR ( FirstDayOfYear, CurrentDate ),
            1 * ( MONTH ( [Date] ) <> 2 || DAY ( [Date] ) <> 29 )
        )
    RETURN ROW (
        "DateKey", INT ( DateKey ),
        "Sequential Day Number", INT ( [Date] ),
        "Year Month", FORMAT ( CurrentDate, "mmm yyyy" ),
        "Year Month Number", Yr * 12 + Mn - 1,
        "Fiscal Year", "FY " & Fyr,
        "Fiscal Year Number", Fyr,
        "Fiscal Year Quarter", "F" & Fqr & "-" & Fyr,
        "Fiscal Year Quarter Number", CONVERT ( Fyr * 4 + FQrn - 1, INTEGER ),
        "Fiscal Quarter", "F" & Fqr,
        "Month", FORMAT ( CurrentDate, "mmm" ),
        "Fiscal Month Number", Fmn,
        "Fiscal Month in Quarter Number",  Fmqn,
        "Day of Week", FORMAT ( CurrentDate, "ddd" ),
        "Day of Week Number", Wd,
        "Day of Month Number", Mdn,
        "Day of Fiscal Year Number", Fydn,
        "Working Day", IF ( WorkingDay, "Working Day", "Non-Working Day" )
    )
)
```

The first two variables are useful to customize the beginning of both the fiscal year and the week. The next variables detect the range of fiscal years required, based on the transactions in *Sales*. You can customize *FirstSalesDate* and *LastSalesDate* to retrieve the first and last transaction date in your model, or you can assign the first and last fiscal year in the *FirstFiscalYear* and *LastFiscalYear* variables.

The quarters are computed starting from the first month of the fiscal year. The *Date* table contains hidden columns to support the correct sorting of years, quarters, and months. These hidden columns are populated with sequential numbers that make it easy to apply filters to retrieve previous or following years, quarters, and months, without relying on complex calculations at query time.

Among the many columns, one is worth expanding on. The ***Year Month Number*** column contains the year number multiplied by 12, plus the month. The resulting number is hard to read, but it allows math over months. Given the *Year Month Number* value, you can just subtract 12 to go back one year; this gives you the value of *Year Month Number* corresponding to the same month in the previous year. Many formulas use this characteristic to perform time-shifts.

In order to obtain the right visualization, the **calendar columns** must be configured in the data model as follows – for each column you can see the data type and the format string, followed by a sample value:

- *Date*: Date, m/dd/yyyy (8/14/2007), used as a column to mark as date table (not required)
- *DateKey*: Whole Number, (20070814), used as an alternate key for relationships
- *Sequential Day Number*: Whole Number, Hidden (40040), same value of Date as integer
- *Year Month*: Text (Aug 2007)
- *Year Month Number*: Whole Number, Hidden (24091)
- *Month*: Text (Aug)
- *Fiscal Month Number*: Whole Number, Hidden (6)
- *Fiscal Month in Quarter Number*: Whole Number, Hidden (3)
- *Fiscal Year*: Text (FY 2008)
- *Fiscal Year Number*: Whole Number, Hidden (2008)
- *Fiscal Year Quarter*: Text (FQ2-2008)
- *Fiscal Year Quarter* Number: Whole Number, Hidden (8033)
- *Fiscal Quarter*: Text (FQ2)
- *Day of Fiscal Year Number*: Whole Number, Hidden (167)
- *Day of Month Number*: Whole Number, Hidden (14)

We want to introduce the concept of **filter-safe columns**. In a table, there are columns whose filters need to be preserved. The filters over filter-safe columns are not altered by the time intelligence calculations. They will be affecting the calculations presented in this pattern. The filter-safe columns in our sample table are the following:

- *Day of Week*: ddd (Tue)
- *Day of Week Number*: Whole Number, Hidden (6)
- *Working Day*: Text (Working Day)

We further describe the behavior of filter-safe columns in the next section.

The *Date* table in this pattern has one hierarchy:

- Fiscal: Year (*Fiscal Year*), Quarter (*Fiscal Year Quarter*), Month (*Year Month*)

The columns are designed to simplify the formulas. For example, the *Day of Fiscal Year Number* column contains the number of days since the beginning of the fiscal year, ignoring February 29 in leap years; this number makes it easier to find a corresponding range of dates in the previous year.

The *Date* table must also include a hidden *DateWithSales* calculated column, used by some of the formulas of this pattern:

Calculated column in the Date table

```
DateWithSales =
'Date'[Date] <= MAX ( Sales[Order Date] )
```

The *Date[DateWithSales]* column is *TRUE* if the date is on or before the last date with sales; it is *FALSE* otherwise. In other words, *DateWithSales* is *TRUE* for "past" dates and *FALSE* for "future" dates, where "past" and "future" are relative to the last date with sales.

In case you import a *Date* table, you want to create columns that are similar to the ones we describe in this pattern, in that they should behave the same way.

Understanding filter-safe columns

The *Date* table contains two types of columns: regular columns and filter-safe columns. The regular columns are manipulated by the measures shown in this pattern. The filters over filter-safe columns are always preserved and never altered by the measures of this pattern. An example clarifies this distinction. The *Year Month Number* column is a regular column: the formulas in this pattern have the option of changing its value during their computation.

For example, in order to compute the previous month the formulas change the filter context by subtracting one to the value of *Year Month Number* in the filter context. Conversely, the *Day of Week* column is a filter-safe column. If a user filters Monday to Friday, the formulas do not alter that filter on the day of the week. Therefore, a previous-year measure keeps the filter on the day of the week; it replaces only the filter on calendar columns such as year, month, and date.

To implement this pattern, you must identify which columns need to be treated as filter-safe columns, because filter-safe columns require special handling. The following is the classification of the columns used in the *Date* table of this pattern:

- **Calendar columns**: *Date*, *DateKey*, *Sequential Day Number*, *Year Month*, *Year Month Number*, *Month*, *Fiscal Month Number*, *Fiscal Month in Quarter Number*, *Fiscal Year*, *Fiscal Year Number*, *Fiscal Year Quarter*, *Fiscal Year Quarter Number*, *Fiscal Quarter*, *Day of Fiscal Year Number*, *Day of Month Number* .
- **Filter-safe columns**: *Day of Week*, *Day of Week Number*, *Working Day*.

The special handling of filter-safe columns pertains to the filter context. Every measure in this pattern manipulates the filter context by replacing filters over all the calendar columns, without altering any filter applied to the filter-safe columns. In other words, every measure follows two rules:

- Remove filters on calendar columns;
- Keep filters on filter-safe columns.

The ALLEXCEPT function can implement these requirements; specify the *Date* table in the first argument, and the filter-safe columns in the following arguments:

```
CALCULATE (
    [Sales Amount],
    ALLEXCEPT ( 'Date', 'Date'[Working Day], 'Date'[Day of Week] ),
    ... // Filters over one or more calendar columns
)
```

If the *Date* table did not have any filter-safe column, the filters could be removed by using REMOVEFILTERS over the *Date* table instead of ALLEXCEPT:

```
CALCULATE (
    [Sales Amount],
    REMOVEFILTERS ( 'Date' ),
    ... // Filters over one or more calendar columns
)
```

If your *Date* table does not contain any filter-safe column, then you can use REMOVEFILTERS instead of ALLEXCEPT in all the measures of this pattern. We provide a complete scenario that includes filter-safe columns. Whenever possible, you can simplify it.

While the ALLEXCEPT should include all the filter-safe columns, we skip specifically the hidden filter-safe columns used only to sort other columns. For example, we do not include *Day of Week Number*, which is a hidden column used to sort the *Day of Week* column. The assumption is that the user never applies filters on hidden columns; if this assumption is not true, then the hidden filter-safe columns must also be included in the ALLEXCEPT arguments. You can find an example of the different results of using REMOVEFILTERS and ALLEXCEPT in the **Year-to-date total** section of this pattern.

Controlling the visualization on future dates

Most of the time intelligence calculations should not display values for dates after the last date available. For example, a year-to-date calculation can also show values for future dates, but we want to hide those values. The dataset used in these examples ends on August 15, 2009. Therefore, we consider the month of August 2009, the third quarter of 2009 (Q3-2009), and the year 2009 as the last time periods with data. Any date later than August 15, 2019 is considered future, and we want to hide its values.

In order to avoid showing results in future dates, we use the following *ShowValueForDates* measure. *ShowValueForDates* returns TRUE if the period selected is earlier than the last period with data:

Measure (hidden) in the Date table

```
ShowValueForDates :=
VAR LastDateWithData =
    CALCULATE (
        MAX ( 'Sales'[Order Date] ),
        REMOVEFILTERS ()
    )
VAR FirstDateVisible =
    MIN ( 'Date'[Date] )
VAR Result =
    FirstDateVisible <= LastDateWithData
RETURN
    Result
```

The *ShowValueForDates* measure is hidden. It is a technical measure created to reuse the same logic in many different time-related calculations, and the user should not use *ShowValueForDates* directly in a report. The REMOVEFILTERS function removes filters from all tables in the model, because the purpose is to retrieve the last date used in the *Sales* table regardless of filters.

Naming convention

This section describes the naming convention we adopted to reference the time intelligence calculations. A simple categorization shows whether a calculation:

- Shifts over a period of time, for example the same period in the previous year;
- Performs an aggregation, for example year-to-date; or,
- Compares two time periods, for example this year compared to last year.

Acronym	Description	Shift	Aggregation	Comparison
YTD	Year-to-date		X	
QTD	Quarter-to-date		X	
MTD	Month-to-date		X	
MAT	Moving annual total		X	
PY	Previous year	X		
PQ	Previous quarter	X		
PM	Previous month	X		
PYC	Previous year complete	X		
PQC	Previous quarter complete	X		
PMC	Previous month complete	X		
PP	Previous period (automatically selects year, quarter, or month)	X		
PYMAT	Previous year moving annual total	X	X	
YOY	Year-over-year			X
QOQ	Quarter-over-quarter			X
MOM	Month-over-month			X
MATG	Moving annual total growth	X	X	X
POP	Period-over-period (automatically selects year, quarter, or month)			X
PYTD	Previous year-to-date	X	X	
PQTD	Previous quarter-to-date	X	X	
PMTD	Previous month-to-date	X	X	
YOYTD	Year-over-year-to-date	X	X	X
QOQTD	Quarter-over-quarter-to-date	X	X	X
MOMTD	Month-over-month-to-date	X	X	X
YTDOPY	Year-to-date-over-previous-year	X	X	X
QTDOPQ	Quarter-to-date-over-previous-quarter	X	X	X
MTDOPM	Month-to-date-over-previous-month	X	X	X

Contents

Get the full book

Paperback

- Amazon.com
- Amazon.ae
- Amazon.ca
- Amazon.co.jp
- Amazon.co.uk
- Amazon.com.au
- Amazon.com.br
- Amazon.com.mx
- Amazon.de
- Amazon.es
- Amazon.fr
- Amazon.in
- Amazon.it
- Amazon.nl

Kindle

- Amazon.com
- Amazon.ca
- Amazon.co.jp
- Amazon.co.uk
- Amazon.com.au
- Amazon.com.br
- Amazon.com.mx
- Amazon.de
- Amazon.es
- Amazon.fr
- Amazon.in
- Amazon.it
- Amazon.nl

Ebook

- Apple books
- Google books

PDF

- Lulu

Publisher / Editorial Production: SQLBI Corp., Las Vegas, NV, Unites States

Authors: Alberto Ferrari, Marco Russo
Copy Editor: Claire Costa
Technical Editors: Daniil Maslyuk, Sergio Murru
Cover Designer: Daniele Perilli

ISBN: 978-1-7353652-0-6
Library of Congress Control Number: 2020912594

All the samples and files used in this book are available on **www.daxpatterns.com**

All the code in this book has been formatted with **www.daxformatter.com**

DAX PATTERNS
SECOND EDITION

A pattern is a general, reusable solution to a frequent or common challenge.

This book is the second edition of **the most comprehensive collection of ready-to-use solutions in DAX**, that you can use in Microsoft Power BI, Analysis Services Tabular, and Power Pivot for Excel.

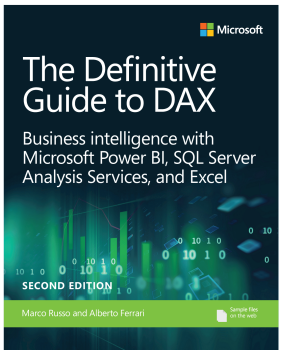
IN THIS BOOK

- Standard time-related calculations
- Month-related calculations
- Week-related calculations
- Custom time-related calculations
- Comparing different time periods
- Semi-additive calculations
- Cumulative total
- Parameters table
- Static segmentation
- Dynamic segmentation
- ABC classification
- New and returning customers
- Related distinct count
- Events in progress
- Ranking
- Hierarchies
- Parent-child hierarchies
- Like-for-like comparison
- Transition matrix
- Survey
- Basket analysis
- Currency conversion
- Budget

www.daxpatterns.com

Alberto Ferrari and **Marco Russo** co-founded SQLBI.com, where they publish frequent articles about DAX and other Microsoft tools.
They are regular speakers at major international conferences such as Microsoft Ignite, PASS Summit, and SQLBits. Both currently teach, consult, and mentor on Microsoft Business Intelligence technologies.

They have written several books, including the best-seller **The Definitive Guide to DAX** published by Microsoft Press.



Find out more at
www.sqlbi.com/books