

# СПРАВОЧНИК DAX ФУНКЦИЙ НА РУССКОМ ЯЗЫКЕ

## ДЛЯ POWER BI И POWER PIVOT

БУДУЕВ АНТОН  
[biprosto.ru](http://biprosto.ru)



Дорогие друзья, всем привет! Меня зовут Будуев Антон, автор данного справочника DAX функций для Power BI и Power Pivot. Давайте немного познакомимся.

Скажу сразу – я не являюсь каким-то первоисточником, создателем или разработчиком языка DAX.

Я практик, и все, что связано со сквозной BI аналитикой (Power BI, DAX и прочее) - все это я приобретаю на основе своего личного опыта, на основе личной практики внедрения BI отчетов в своем

бизнесе.

Естественно, чтобы все это внедрять у себя – я много учусь. А так как на русскоязычном пространстве очень мало каких-либо материалов по BI аналитике, то в основном, приходится обучаться за рубежом.

И именно поэтому, я и написал данный справочник. Во-первых, чтобы облегчить повседневную работу себе, а во-вторых, чтобы помочь многим из Вас.

Данный справочник содержит основной список (! но не весь...) DAX функций, которые наиболее часто встречаются в работе. В дальнейшем справочник будет обновляться и дополняться новыми функциями. При обновлении справочника Вам на емейл придет соответствующее письмо.

Наши социальные сети: [Вконтакте](#), [FaceBook](#), [Инстаграм](#), [YouTube](#)

## P.S.

Друзья, хотите большего?

Хотите закрыть свои вопросы с непониманием языка DAX?

Хотите научиться писать формулы в Power BI и Power Pivot быстро, чтобы освободить свое личное время?

Приглашаю Вас в новый, большой, подробный, пошаговый курс **«DAX — это просто»** с практикой и самостоятельными заданиями.

В курсе изложено мое простое авторское видение конструктора формул DAX для Power BI и Power Pivot.

Вы на практике вместе со мной, начиная с самых основ и продолжая далее углубляться небольшими шажками, отработаете навыки работы с языком функций и формул DAX с самого начала (с момента создания модели данных) и до самого конца — до разбора сложных витиеватых формул.

Язык DAX преподнесен как простой конструктор, состоящий из нескольких блоков, которые имеют свое определенное, конкретное предназначение. Сочетая различными способами эти блоки, Вы, при помощи конструктора формул DAX, с легкостью сможете решать любые (простые или сложные) аналитические задачи.

Очень важно, что в данном курсе язык DAX рассматривается с точек зрения сразу обеих популярных платформ — и Power BI, и Power Pivot, с разбором тонкостей и нюансов его применения в каждой из платформ.

👉 Итак, узнать все подробности об этом курсе, **а также заказать его со скидкой 50% !!!** Вы можете **[по данной ссылке](#)**

## Оглавление

<b>1. ТЕКСТОВЫЕ ФУНКЦИИ DAX</b>	9
DAX ФУНКЦИЯ BLANK	10
DAX ФУНКЦИЯ CONCATENATE	12
DAX ФУНКЦИЯ CONCATENATEX	14
DAX ФУНКЦИЯ FORMAT	16
DAX ФУНКЦИЯ VALUE	24
DAX ФУНКЦИЯ FIND	25
DAX ФУНКЦИЯ SEARCH	28
DAX ФУНКЦИЯ FIXED	31
DAX ФУНКЦИИ LEFT И RIGHT	33
DAX ФУНКЦИЯ MID	35
DAX ФУНКЦИЯ LEN	38
DAX ФУНКЦИИ LOWER И UPPER	39
DAX ФУНКЦИЯ REPLACE	41
DAX ФУНКЦИЯ SUBSTITUTE	44
DAX ФУНКЦИЯ REPT	46
DAX ФУНКЦИЯ TRIM	47
DAX ФУНКЦИЯ EXACT	48
<b>2. ЛОГИЧЕСКИЕ ФУНКЦИИ DAX</b>	49
DAX ФУНКЦИЯ IF	50
DAX ФУНКЦИИ TRUE И FALSE	53
DAX ФУНКЦИЯ SWITCH	55
DAX ФУНКЦИЯ IFERROR	58
DAX ФУНКЦИИ AND И OR	59
DAX ФУНКЦИЯ NOT	63
<b>3. МАТЕМАТИЧЕСКИЕ ФУНКЦИИ DAX</b>	65
DAX ФУНКЦИЯ DIVIDE	66
DAX ФУНКЦИЯ QUOTIENT	69
DAX ФУНКЦИЯ MOD	71
DAX ФУНКЦИЯ SQRT	75
DAX ФУНКЦИЯ POWER	76

DAX ФУНКЦИЯ ABS	77
DAX ФУНКЦИЯ SIGN	78
DAX ФУНКЦИЯ EXP	79
DAX ФУНКЦИЯ FACT	80
DAX ФУНКЦИИ LN, LOG, LOG10	81
DAX ФУНКЦИЯ PI	82
DAX ФУНКЦИЯ INT	83
DAX ФУНКЦИЯ TRUNC	84
DAX ФУНКЦИЯ ROUND	85
DAX ФУНКЦИЯ ROUNDDOWN	86
DAX ФУНКЦИЯ ROUNDUP	88
DAX ФУНКЦИЯ MROUND	89
DAX ФУНКЦИЯ FLOOR	91
DAX ФУНКЦИЯ CEILING	93
DAX ФУНКЦИЯ ISO.CEILING	95
DAX ФУНКЦИИ RAND и RANDBETWEEN	96
DAX ФУНКЦИЯ RADIANS	97
<b>4. СТАТИСТИЧЕСКИЕ ФУНКЦИИ (АГРЕГИРОВАНИЕ)</b>	98
DAX ФУНКЦИЯ SUM	99
DAX ФУНКЦИЯ SUMX	101
DAX ФУНКЦИЯ MAX	104
DAX ФУНКЦИЯ MAXA	106
DAX ФУНКЦИЯ MAXX	107
DAX ФУНКЦИИ MIN, MINA, MINX	109
DAX ФУНКЦИИ AVERAGE, AVERAGEA, AVERAGEX	111
DAX ФУНКЦИИ COUNT, COUNTA, COUNTX, COUNTAX	114
DAX ФУНКЦИЯ COUNTBLANK	117
DAX ФУНКЦИЯ DISTINCTCOUNT	118
DAX ФУНКЦИЯ COUNTROWS	119
<b>5. ФУНКЦИИ ДАТЫ И ВРЕМЕНИ В DAX</b>	121
DAX ФУНКЦИЯ TODAY	122
DAX ФУНКЦИЯ NOW	123

DAX ФУНКЦИЯ DATE.....	124
DAX ФУНКЦИЯ DATEVALUE.....	125
DAX ФУНКЦИИ YEAR, MONTH, DAY.....	127
DAX ФУНКЦИЯ TIME.....	129
DAX ФУНКЦИЯ TIMEVALUE.....	131
DAX ФУНКЦИИ HOUR, MINUTE и SECOND.....	132
DAX ФУНКЦИЯ WEEKDAY.....	134
DAX ФУНКЦИЯ WEEKNUM.....	136
DAX ФУНКЦИЯ YEARFRAC.....	138
<b>6. ФУНКЦИИ ЛОГИКИ ОПЕРАЦИЙ С ДАТОЙ В DAX (TIME INTELLIGENCE) .....</b>	<b>140</b>
DAX ФУНКЦИЯ CALENDARAUTO.....	141
DAX ФУНКЦИЯ CALENDAR.....	145
DAX ФУНКЦИЯ DATESBETWEEN.....	146
DAX ФУНКЦИЯ DATESINPERIOD.....	148
DAX ФУНКЦИЯ DATEADD.....	151
DAX ФУНКЦИЯ PARALLELPERIOD.....	154
DAX ФУНКЦИЯ SAMEPERIODLASTYEAR.....	159
DAX ФУНКЦИИ PREVIOUSYEAR, PREVIOUSQUARTER, PREVIOUSMONTH, PREVIOUSDAY ..	161
DAX ФУНКЦИИ NEXTYEAR, NEXTQUARTER, NEXTMONTH и NEXTDAY.....	166
DAX ФУНКЦИИ DATESYTD, DATESQTD и DATESMTD.....	171
DAX ФУНКЦИИ TOTALYTD, TOTALQTD, TOTALMTD.....	176
DAX ФУНКЦИИ CLOSINGBALANCEYEAR, CLOSINGBALANCEQUARTER и CLOSINGBALANCEMONTH.....	183
DAX ФУНКЦИИ OPENINGBALANCEYEAR, OPENINGBALANCEQUARTER и OPENINGBALANCEMONTH.....	188
DAX ФУНКЦИИ ENDOFYEAR, ENDOFQUARTER и ENDOFMONTH.....	189
DAX ФУНКЦИИ STARTOFYEAR, STARTOFQUARTER, STARTOFMONTH.....	189
DAX ФУНКЦИИ FIRSTDATE и LASTDATE.....	194
DAX ФУНКЦИЯ EOMONTH.....	198
DAX ФУНКЦИЯ EDATE.....	200
<b>7. ФУНКЦИИ ФИЛЬТРОВ В DAX .....</b>	<b>202</b>
DAX ФУНКЦИЯ CALCULATE.....	203

DAX ФУНКЦИЯ CALCULATETABLE.....	207
DAX ФУНКЦИЯ FILTER.....	209
DAX ФУНКЦИЯ ALL.....	214
DAX ФУНКЦИЯ ALLEXCEPT.....	224
DAX ФУНКЦИЯ ALLSELECTED.....	226
DAX ФУНКЦИЯ ALLNOBLANKROW.....	228
DAX ФУНКЦИЯ VALUES.....	230
DAX ФУНКЦИЯ DISTINCT.....	232
DAX ФУНКЦИЯ RELATED.....	235
DAX ФУНКЦИЯ RELATEDTABLE.....	239
<b>8. ТАБЛИЧНЫЕ ФУНКЦИИ DAX.....</b>	<b>241</b>
DAX ФУНКЦИЯ ADDCOLUMNS.....	242
DAX ФУНКЦИЯ DATATABLE.....	246
DAX ФУНКЦИЯ SUMMARIZE.....	249
DAX ФУНКЦИЯ SUMMARIZECOLUMNS.....	254
DAX ФУНКЦИЯ GROUPBY.....	257
DAX ФУНКЦИЯ ROW.....	260
DAX ФУНКЦИЯ TOPN.....	262
DAX ФУНКЦИЯ EXCEPT.....	264
DAX ФУНКЦИЯ INTERSECT.....	266
DAX ФУНКЦИЯ UNION.....	268
DAX ФУНКЦИЯ NATURALINNERJOIN.....	270
DAX ФУНКЦИЯ NATURALLEFTOUTERJOIN.....	273
DAX ФУНКЦИЯ GENERATESERIES.....	275
DAX ФУНКЦИЯ CROSSJOIN.....	277
DAX ФУНКЦИИ GENERATE И GENERATEALL.....	278
<b>9. ИНФОРМАЦИОННЫЕ ФУНКЦИИ DAX.....</b>	<b>281</b>
DAX ФУНКЦИЯ ISBLANK.....	282
DAX ФУНКЦИЯ ISNUMBER.....	283
DAX ФУНКЦИЯ ISEVEN.....	284
DAX ФУНКЦИИ ISTEXT, ISNONTEXT.....	285
DAX ФУНКЦИЯ ISERROR.....	287



DAX ФУНКЦИЯ ISLOGICAL .....	288
DAX ФУНКЦИИ HASONEVALUE И HASONEFILTER .....	289
DAX ФУНКЦИИ ISFILTERED И ISCROSSFILTERED .....	293
<b>10. ПРОЧИЕ ФУНКЦИИ DAX</b> .....	296
DAX ФУНКЦИИ FIRSTNONBLANK И LASTNONBLANK .....	297
DAX ФУНКЦИЯ EARLIER .....	300
VAR И RETURN – ПЕРЕМЕННЫЕ В DAX .....	306
DAX ФУНКЦИЯ RANKX .....	311
DAX ФУНКЦИЯ ERROR .....	317

Внешние данные Вставка Настройка

CALCULATE =  
CALCULATE (  
SUMX(  
'Заявки'  
'Заявки'  
);  
'Заявки'

[Онлайн-видеокурс]

# DAX - ЭТО ПРОСТО

(для Power BI и Power Pivot)

Узнать подробнее о курсе

\* простой авторский взгляд на конструктор формул DAX  
для Power BI и Power Pivot



# 1. ТЕКСТОВЫЕ ФУНКЦИИ

## DAX

## DAX функция BLANK

BLANK () — возвращает пустое значение.

Синтаксис:

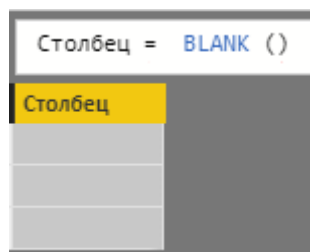
BLANK ()

BLANK () — это самая простая из [всех функций языка DAX](#), которая не имеет никаких параметров и возвращает просто пустое значение. Но, несмотря на свою простоту, она очень часто используется во многих [формулах в Power BI](#), зачастую тогда, когда нужно вывести вместо какой-то ошибки просто пустую ячейку.

Для примера, если мы в [Power BI Desktop](#) на основе нее создадим вычисляемый столбец по следующей формуле:

Столбец = BLANK ()

то, в результате увидим пустой столбец:

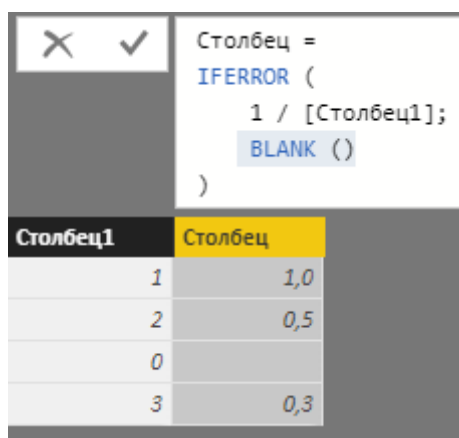


Естественно, в одиночку эту функцию использовать нет никакого смысла. Как я уже писал выше, зачастую она используется в формулах, где нужно заменить ошибку пустым значением. Например, когда происходит деление на 0, то возвращается ошибка, потому что на 0 делить нельзя. И в этом случае, когда происходит деление на 0, можно вывести вместо ошибки просто пустое значение, как в формуле ниже:

```
Столбец =  
IFERROR (  
    1 / [Столбец1];  
    BLANK ()  
)
```

В этой формуле мы воспользовались еще одной DAX функцией [IFERROR](#), которая выполняет выражение, указанное в первом параметре (в нашем случае, это «1» деленное на значения из столбца [Столбец1]) и если ошибки нет, то возвращает значение, получившееся во время выполнения выражения.

Если ошибка есть (например, деление на 0), то она запускает в работу свой второй параметр, где в нашем случае находится функция BLANK, которая, в свою очередь, возвратит пустое значение, как в примере ниже в Power BI:



Столбец1	Столбец
1	1,0
2	0,5
0	
3	0,3

В этом примере в 3 строчке столбца [Столбец1] значение 0, поэтому, когда выполнятся деление «1» на значение столбца, то в 3 строке возникает ошибка и вместо нее BLANK выводит пустое значение.

# DAX функция CONCATENATE

CONCATENATE () — производит объединение двух текстовых строк в одну единую.

Синтаксис:

CONCATENATE ("Текст 1"; "Текст 2")

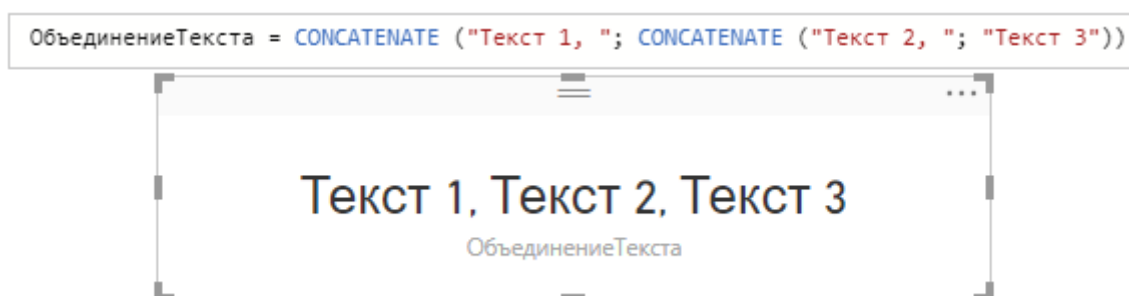
Где, «Текст» — это строка, содержащая текст или число. Также, в качестве параметров могут быть ссылки на столбцы с текстовым типом данных.

В качестве примера [формулы](#) рассмотрим DAX функцию CONCATENATE, вложенную саму в себя:

Объединение Текста =

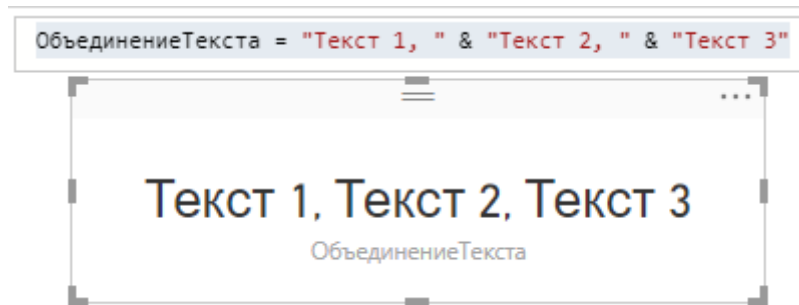
```
CONCATENATE (  
    "Текст 1, ";  
    CONCATENATE ("Текст 2, "; "Текст 3")  
)
```

Результатом выполнения этой формулы будет следующая единая текстовая строка:



Аналогом [функции](#) CONCATENATE является оператор объединения текста [в языке DAX](#) «амперсанд и» — &:

Объединение Текста = "Текст 1, " & "Текст 2, " & "Текст 3"



# DAX функция CONCATENATEX

CONCATENATEX () — объединяет результат выражения, вычисленного для каждой строки таблицы.

Синтаксис:

CONCATENATEX ('Таблица'; Выражение; "Разделитель")

Где:

- 'Таблица' — таблица, содержащая строки, для которых будет вычислено выражение
- Выражение — выражение, вычисляемое для каждой строки таблицы
- «Разделитель» — разделитель объединяемых частей текста (необязательный элемент)

Рассмотрим пример формулы на основе DAX функции CONCATENATEX.

В [Power BI Desktop](#) имеется исходная таблица «Города», содержащая столбец, в каждой строчке которого, прописан только один город:

Город
Москва
Санкт-Петербург
Москва
Красноярск
Санкт-Петербург

Задача — создать единую текстовую строку, где будут объединены города из всех строк исходной таблицы, с разделителем «, » (запятая и пробел).

Для решения этой задачи хорошо подойдет функция CONCATENATEX, в первый параметр которой, мы пропишем исходную таблицу «Города». Во втором параметре, в качестве выражения, просто укажем исходный столбец [Город].



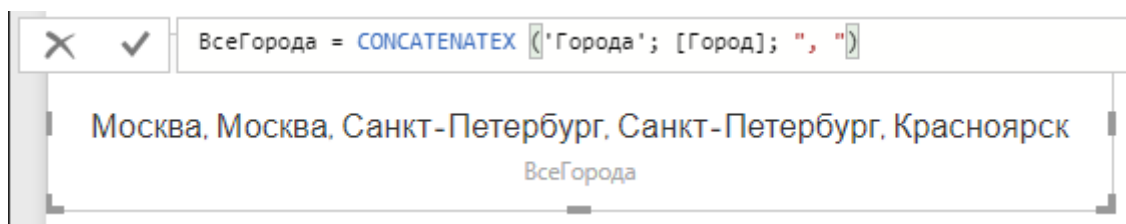
Тогда, когда это выражение будет выполняться для каждой строки исходной таблицы, результатом будет возвращение значения из ячейки этого столбца.

В третьем параметре функции CONCATENATEX мы пропишем разделитель «, » (запятая и пробел).

Итого, у нас получится следующая формула:

Все Города = CONCATENATEX ('Города'; [Город]; ", ")

Результатом выполнения этой формулы в Power BI будет единая строка с перечислением всех городов через запятую:



# DAX функция FORMAT

DAX функция `FORMAT ()` — меняет различные типы данных (числа, дата, время) на текстовый тип данных с возможностью задать нужный формат вывода информации.

Синтаксис:

`FORMAT (Значение; Формат)`

Где:

- Значение — значение с типом данных число, дата, время
- Формат — шаблон формата

## Шаблоны формата чисел

Ниже приведены основные и часто используемые шаблоны формата чисел, а также примеры формул на их основе:

- «General Number» — выводит число

Пример: `FORMAT (12345,678; "General Number")`

Результат: 12345,678

- «Currency» — денежный формат. Выводит число, разделяя его на группы разрядов на основании параметров локали ПК

Пример: `FORMAT (12345,678; "Currency")`

Результат: 12 345,68 Руб

- «Fixed» — округляет число до 2 знаков после запятой

Пример: `FORMAT (12345,67867; "Fixed")`

Результат: 12345,68

- «Standard» — выводит число, разделенное на группы разрядов, округляя его до 2 знаков после запятой

Пример: `FORMAT (12345,67867; "Standard")`

Результат: 12 345,68

- «Percent» — умножает число на 100, преобразуя его в проценты (%). После запятой число округляется до 2 знаков

Пример: `FORMAT (0,67867; "Percent")`

Результат: 67,87%

- «Scientific» — преобразует число в научный вид

Пример: `FORMAT (1234567; "Scientific")`

Результат: 1,23E+06

- «Yes/No» — выводит «Да», если число не равно 0 и «Нет», если равно 0

Пример: а) `FORMAT (1234567; "Yes/No")`

б) `FORMAT (0; "Yes/No")`

Результат а: да

Результат б: нет

- «True/False» — выводит «TRUE» (Истина), если число не равно 0 и «FALSE» (Ложь), если равно 0

Пример: а) `FORMAT (1234567; "True/False")`

б) `FORMAT (0; "True/False")`

Результат а: Истина

Результат б: Ложь

- «On/Off» — выводит «Вкл», если число не равно 0 и «Выкл», если равно 0

Пример: а) `FORMAT (1234567; "On/Off")`

б) `FORMAT (0; "On/Off")`

Результат а: Вкл

Результат б: Выкл

## Шаблоны формата дат

Ниже приведены основные и часто используемые шаблоны формата дат, а также примеры формул на их основе:

- «D» — выводит номер дня месяца (без нулей)

Пример: а) `FORMAT (DATE (2018;10;02); "D")`

б) `FORMAT (DATE (2018;10;23); "D")`

Результат а: 2

Результат б: 23

- «DD» — выводит номер дня месяца (с нулём)

Пример: а) `FORMAT (DATE (2018;10;02); "DD")`

б) `FORMAT (DATE (2018;10;23); "DD")`

Результат а: 02

Результат б: 23

- «DDD» — преобразует исходную дату в день недели (сокращенное название — Пт, Сб, Вс)

Пример: `FORMAT(DATE(2018;10;02); "DDD")`

Результат: Вт

- «DDDD» — преобразует исходную дату в день недели (полное название)

Пример: `FORMAT(DATE(2018;10;02); "DDDD")`

Результат: вторник

- «M» — выводит номер месяца в году (без нулей)

Пример: а) `FORMAT (DATE (2018;01;23); "M")`

б) `FORMAT (DATE (2018;11;23); "M")`

Результат а: 1

Результат б: 11

- «MM» — выводит номер месяца в году (с нулём)

Пример: а) `FORMAT (DATE (2018;01;23); "MM")`

б) `FORMAT (DATE (2018;11;23); "MM")`

Результат а: 01

Результат б: 11

- «MMM» — преобразует исходную дату в месяц (сокращенное название — дек)

Пример: `FORMAT (DATE (2018;11;23); "MMM")`

Результат: ноя

- «MMMM» — преобразует исходную дату в месяц (полное название)

Пример: `FORMAT (DATE (2018;11;23); "MMMM")`

Результат: Ноябрь

- «уу» — выводит год в формате 2 цифр

Пример: а) `FORMAT (DATE (2018;11;23); "уу")`

б) `FORMAT (DATE (2001;11;23); "уу")`

Результат а: 18

Результат б: 01

- «уууу» — выводит год в формате 4 цифр

Пример: `FORMAT (DATE (2018;11;23); "уууу")`

Результат: 2018

## Шаблоны формата времени

Ниже приведены основные и часто используемые шаблоны формата времени, а также примеры формул на их основе:

- «Н» — преобразует исходное время в часы (без нулей)

Пример: `FORMAT (TIME (05;50;16); "Н")`

Результат: 5

- «НН» — преобразует исходное время в часы (с нулём)

Пример: `FORMAT (TIME (05;50;16); "НН")`

Результат: 05

- «т» — преобразует исходное время в минуты (без нулей)

Пример: `FORMAT (TIME (10;07;16); "НН:m")`

Результат: 10:7

- «тт» — преобразует исходное время в минуты (с нулём)

Пример: `FORMAT (TIME (10;07;16); "НН:mm")`

Результат: 10:07



- «s» — преобразует исходное время в секунды (без нулей)

Пример: `FORMAT (TIME (10;18;03); "s")`

Результат: 3

- «ss» — преобразует исходное время в секунды (с нулём)

Пример: `FORMAT (TIME (10;18;03); "ss")`

Результат: 03

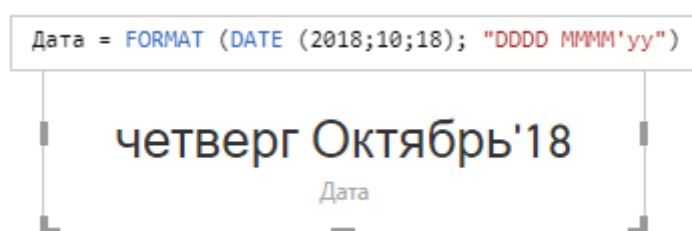
## Совместное использование шаблонов

Вышеприведенные шаблоны функции FORMAT можно использовать как одиночно, так и совмещая их как это будет удобно Вам в конкретном случае.

Например, совместив сразу несколько шаблонов дат друг с другом и разделив их нужными нам разделителями, мы получим удобный для нас текстовый вывод даты в [Power BI](#):

Объединенный шаблон форматов дат: `"DDDD MMMM'yy"`

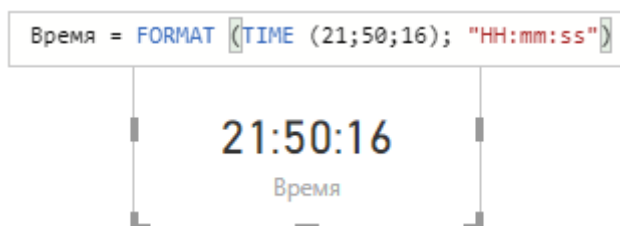
И, как результат, мы получим следующий вывод даты:



Или мы можем объединить сразу несколько шаблонов времени друг с другом, также разделив их нужным разделителем:

Объединенный шаблон форматов времени: `FORMAT (TIME (21;50;16); "HH:mm:ss")`

Как результат, в Power BI мы получим следующий вывод времени:



## DAX функция FORMAT и первый параметр

Во всех выше приведенных примерах мы в первый параметр вставляли либо просто значение, либо другую [DAX функцию](#), возвращающую также значение.

Но, в реальной работе [в DAX формулах](#) чаще всего нужно будет указывать там ссылку на столбец, содержащий какие-то значения. Поэтому, давайте рассмотрим еще один пример использования функции FORMAT, только теперь в качестве первого параметра будем указывать столбец.

Итак, в Power BI у меня имеется исходная таблица со столбцом дат:

Даты
1 января 2018 г.
20 мая 2018 г.
7 ноября 2018 г.
19 июля 2019 г.
30 сентября 2019 г.

Создадим в [Power BI Desktop](#) во вкладке «Моделирование» два вычисляемых столбца по следующим формулам с использованием рассматриваемой нами функции:

Формула 1: Год = FORMAT ('Таблица'[Даты]; "yyyy")

Формула 2: Месяц = FORMAT ('Таблица'[Даты]; "mmmm")

Как итог, мы получили таблицу, где вместо одного столбца с простыми датами появились еще 2 дополнительных столбца со значениями годов и месяцев.

Даты	Год	Месяц
1 января 2018 г.	2018	Январь
20 мая 2018 г.	2018	Май
7 ноября 2018 г.	2018	Ноябрь
19 июля 2019 г.	2019	Июль
30 сентября 2019 г.	2019	Сентябрь

Теперь по этим 2 столбцам мы уже можем свободно создавать в [DAX](#) формулы с фильтрами по годам или месяцам. Например, при помощи функции FILTER (подробно ознакомиться с функцией FILTER Вы можете [в данной статье](#)):

```
Год2018 =
FILTER(
    'Таблица';
    'Таблица'[Год] = "2018"
)
```

FILTER нам возвращает таблицу, отфильтрованную по значению «2018» в созданном нами вычисляемом столбце на основе функции FORMAT:

Даты	Год	Месяц
1 января 2018 г.	2018	Январь
20 мая 2018 г.	2018	Май
7 ноября 2018 г.	2018	Ноябрь

## DAX функция VALUE

VALUE () — преобразует число, записанное в текстовом формате в настоящее число (в числовом типе данных). В некотором смысле, можно сказать, что VALUE — обратная функция для функции [FORMAT](#).

Синтаксис:

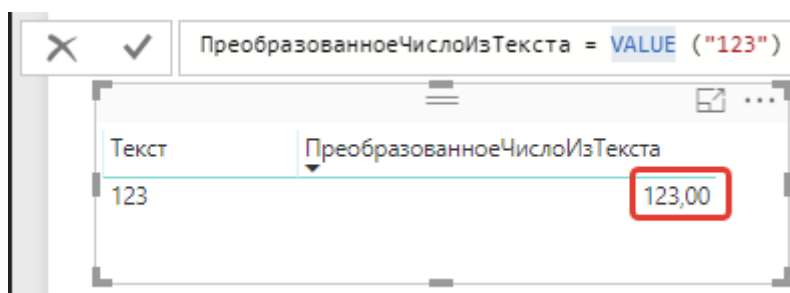
VALUE ("Число")

Где, «Число» — это значение числа (или столбец со значениями чисел), записанное текстом (в текстовом типе данных)

Пример [формулы](#) на основе DAX функции VALUE:

Преобразованное Число Из Текста = VALUE ("123")

Результатом выполнения этой формулы в [Power BI Desktop](#), будет число 123 в числовом формате:



ПреобразованноеЧислоИзТекста = VALUE ("123")	
Текст	ПреобразованноеЧислоИзТекста
123	123,00

Из визуализации мы видим, что, действительно, VALUE преобразовала текст в числовой формат данных, так как число 123.00 в ячейке расположено справа. А по умолчанию, в Power BI числа в таблицах располагаются справа, а текст слева (для примера, рядом в таблице показан текст, который в ячейке находится слева).

## DAX функция FIND

FIND () — ищет один текст в составе другого текста с учетом регистра букв и выводит номер позиции первого символа найденного текста в составе символов другого текста.

Синтаксис:

FIND ("Текст 1"; "Текст 2"; Номер Позиции; Значение Нет)

Где:

- Текст 1 — символы текста, которые нужно найти.

Используются подстановочные шаблоны: ? — один любой символ, \* — много любых символов.

Если в тексте нужно найти знак вопроса (?) или знак звездочка (\*), то перед ними нужно поставить значок тильды ~

- Текст 2 — текст, в котором идет поиск
- Номер Позиции — номер позиции того символа, с которого нужно начать поиск
- Значение Нет — (необязательный параметр) числовое значение или значение [BLANK](#) (), которое нужно вывести, если текст не найден

Примеры [формул](#) на основе DAX функции FIND.

В [Power BI Desktop](#) имеется исходная таблица по товару «Обувь», в которой размещен столбец [Кроссовки] с перечислением брендов кроссовок:

Кроссовки
Кроссовки NIKE
Кроссовки Adidas
Кроссовки ASICS
Кроссовки Patrol

Попробуем в этом столбце при помощи функции FIND найти бренд NIKE и вывести номер позиции первого символа найденного текста, то есть номер позиции символа N. Для этого, создадим в этой таблице второй столбец по следующей формуле:

Поиск NIKE = FIND ("NIKE"; [Кроссовки]; 1; BLANK())

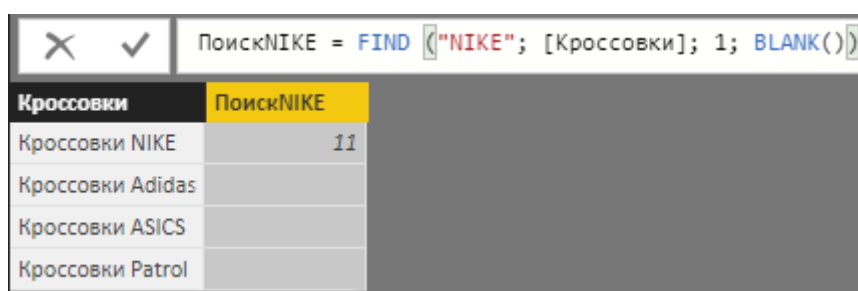
В первом параметре мы прописали тот текст, который мы ищем.

Во втором параметре указали исходный столбец с текстовыми данными, где будет происходить поиск.

В третьем параметре прописали номер позиции символа, с которого начнется поиск в исходном тексте. То есть, в исходном столбце [Кроссовки] поиск начнется с самого первого символа.

В последнем четвертом параметре указали функцию BLANK (), которая выведет пустое значение, если текст не будет найден.

Результатом выполнения этой формулы в Power BI будет создан следующий столбец:



Кроссовки	ПоискNIKE
Кроссовки NIKE	11
Кроссовки Adidas	
Кроссовки ASICS	
Кроссовки Patrol	

То есть, в первой строке функция FIND нашла искомое выражение и вывела номер позиции первого символа, равный 11. Во всех остальных строках — пусто, так как там искомого текстового выражения «NIKE» — нет и в дело вступил четвертый параметр функции FIND, где размещена функция BLANK, возвращающая пустое значение.



Давайте изменим регистр букв искомого выражения с верхнего на нижний, то есть, так:

Поиск NIKE = FIND («nike»; [Кроссовки]; 1; BLANK())

И посмотрим результат в Power BI:



Кроссовки	ПоискNIKE
Кроссовки NIKE	
Кроссовки Adidas	
Кроссовки ASICS	
Кроссовки Patrol	

Как мы видим, FIND не нашла текст «nike», написанное в малом регистре, так как эта функция учитывает регистры текста.

## DAX функция SEARCH

SEARCH () — ищет один текст в составе другого текста без учета регистра букв и выводит номер позиции первого символа найденного текста в составе символов другого текста.

Синтаксис:

SEARCH ("Текст 1"; "Текст 2"; Номер Позиции; Значение Нет)

Где:

- Текст 1 — символы текста, которые нужно найти.

Используются подстановочные шаблоны: ? — один любой символ, \* — много любых символов.

Если в тексте нужно найти знак вопроса (?) или знак звездочка (\*), то перед ними нужно поставить значок тильды ~

- Текст 2 — текст, в котором идет поиск
- Номер Позиции — номер позиции того символа, с которого нужно начать поиск
- Значение Нет — (необязательный параметр) числовое значение или значение BLANK (), которое нужно вывести, если текст не найден

Пример формулы на основе DAX функции SEARCH.

В качестве примера продолжим рассматривать ситуацию, которую мы разбирали в примере выше.

Напишем формулу поиска текста «NIKE» при помощи функции SEARCH:

Поиск NIKE = SEARCH ("NIKE"; [Кроссовки]; 1; BLANK())

Результат будет таким:

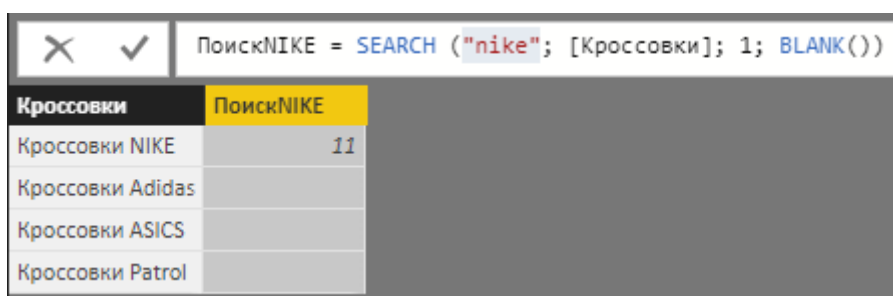


Кроссовки	ПоискNIKE
Кроссовки NIKE	11
Кроссовки Adidas	
Кроссовки ASICS	
Кроссовки Patrol	

Как мы видим, SEARCH нашла искомый текст. А также вывела номер позиции его первого символа N, равный 11. Теперь, как и в примере выше, изменим текст, который нужно найти, а именно изменим его регистр на нижний, то есть так:

Поиск NIKE = SEARCH («nike»; [Кроссовки]; 1; BLANK())

И проверим результат:



Кроссовки	ПоискNIKE
Кроссовки NIKE	11
Кроссовки Adidas	
Кроссовки ASICS	
Кроссовки Patrol	

А результат уже не тот, как был с функцией FIND. Если FIND в этом случае вывел пустое значение, то SEARCH нашла искомый текст, несмотря на то, что регистр букв не совпадал.

В этом и есть различие между двумя функциями FIND и SEARCH — первая функция учитывает регистр букв, а вторая — не учитывает.

Если требуется вернуть найденный текст, то после того, как рассматриваемые функции вернут номер позиции первого символа, можно воспользоваться текстовой функцией [MID](#), которая выведет найденный текст.

То же самое, если найденный текст Вам нужно заменить – то после того, как FIND или SEARCH вернут номер позиции первого символа, нужно воспользоваться текстовой функцией [REPLACE](#), которая произведет замену найденных символов.

## DAX функция FIXED

FIXED () — возвращает в текстовом формате округленное число (до указанного значения десятичного числа)

Синтаксис:

FIXED (Число; Округление; Разряд)

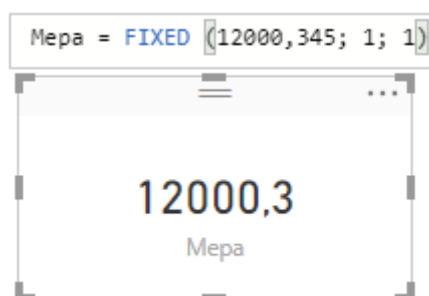
Где:

- Число – исходное десятичное число (столбец с числовыми значениями)
- Округление – (необязательный параметр) до скольких знаков после запятой округлить десятичное число. Если число не указано, то по умолчанию 2
- Разряд – (необязательный параметр) если указано 0, то создать разряд чисел из пробелов, если указано 1, то не создавать разряд. По умолчанию — 0

Пример формулы на основе DAX функции FIXED.

Мера = FIXED (12000,345; 1; 1)

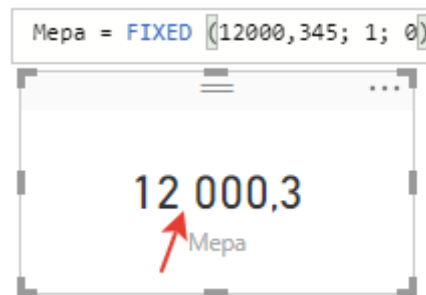
То есть, в этой мере мы задали округление до 1 знака после запятой и не разделять число на разряды. В Power BI эта мера, созданная на основе FIXED, выдаст следующий результат:



Если же мы изменим формулу:

Мера = FIXED (12000,345; 1; 0)

То есть, вместо значения 1 в третьем параметре установим значение 0, что означает «разделить число на разряды», то получим соответствующий результат:





## DAX функции LEFT и RIGHT

LEFT () — выводит нужное количество символов с левой стороны.

RIGHT () — выводит нужное количество символов с правой стороны.

Синтаксис:

LEFT ("Текст"; Количество Символов)

RIGHT ("Текст"; Количество Символов)

Где:

- Текст – исследуемый текст или столбец с текстовыми данными
- Количество Символов – количество символов, которые нужно вывести

Рассмотрим примеры формул на основе DAX функций LEFT и RIGHT.

В [Power BI Desktop](#) имеется исходная таблица по товарам «Обувь», содержащая один столбец [Кроссовки], в котором расположена в единую строку информация по виду обуви (кроссовки), бренду и по сезону коллекции:

Кроссовки
Кроссовки NIKE коллекц'17
Кроссовки Adidas коллекц'17
Кроссовки NIKE коллекц'18
Кроссовки Patrol коллекц'17

Задача – создать дополнительные вычисляемые столбцы в исходной таблице. Отдельно с информацией по виду обуви и отдельно по сезонам коллекций.

Так как вид обуви (слово «кроссовки») в каждой строке находится слева, а сезон коллекции (слово «коллекц'17») — справа и оба этих слова нужно вывести в отдельные столбцы, то для этого очень хорошо подойдут рассматриваемые функции LEFT и RIGHT.

Давайте напишем соответствующие [формулы](#):

Вид Обуви = LEFT ([Кроссовки]; 9)

Сезон Коллекции = RIGHT ([Кроссовки]; 10)

В качестве первых параметров в этих функциях мы указали ссылку на исходный столбец, который содержит исследуемые текстовые данные.

Во вторых параметрах мы прописали количество символов, которые нужно вывести — 9 и 10, так как слово «кроссовки» состоит из 9 символов, а «коллекц'17» из 10.

В итоге, в Power BI мы получили следующий результат:

СезонКоллекции = RIGHT ([Кроссовки]; 10)		
Кроссовки	ВидОбуви	СезонКоллекции
Кроссовки NIKE коллекц'17	Кроссовки	коллекц'17
Кроссовки Adidas коллекц'17	Кроссовки	коллекц'17
Кроссовки NIKE коллекц'18	Кроссовки	коллекц'18
Кроссовки Patrol коллекц'17	Кроссовки	коллекц'17

## DAX функция MID

MID () — выводит часть текста заданной длины с учетом первоначальной позиции.

Синтаксис:

MID ("Текст"; Позиция; Количество Символов)

Где:

- Текст – исследуемый текст или столбец с текстовыми данными
- Позиция – номер позиции символа, с которого нужно выводить текст
- Количество Символов – количество символов, которые нужно вывести

Рассмотрим пример работы формулы на основе DAX функции MID.

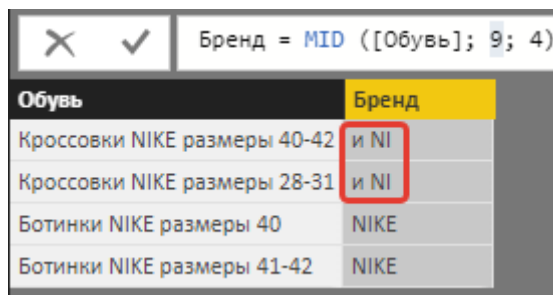
В модели данных Power BI имеется исходная таблица по товарам в магазине обуви. В таблице имеется один столбец, в котором в единую строку расписана информация по виду обуви, бренду и размеру:

Обувь
Кроссовки NIKE размеры 40-42
Кроссовки NIKE размеры 28-31
Ботинки NIKE размеры 40
Ботинки NIKE размеры 41-42

Задача — на основе имеющейся информации, требуется создать в исходной таблице столбец с именем бренда. Попробуем это реализовать при помощи функции MID:

Бренд = MID ([Обувь]; 9; 4)

Но, данная формула, в Power BI работает неправильно:



Обувь	Бренд
Кроссовки NIKE размеры 40-42	и NI
Кроссовки NIKE размеры 28-31	и NI
Ботинки NIKE размеры 40	NIKE
Ботинки NIKE размеры 41-42	NIKE

Так как, на самом деле, мы не можем просто так, числом указать номер стартовой позиции вывода текста, потому что перед брендом NIKE расположены разные слова с разной длиной и нам каким-то образом нужно узнать в каждом конкретном случае номер стартовой позиции.

Справится с этой проблемой нам поможет другая DAX функция — FIND, которая найдет среди всей строки заданное нами слово «NIKE» и вернет ее стартовую позицию среди всего текста. Подробно прочитать о работе функции FIND Вы можете [в этой статье](#).

Итак, исправим нашу формулу:

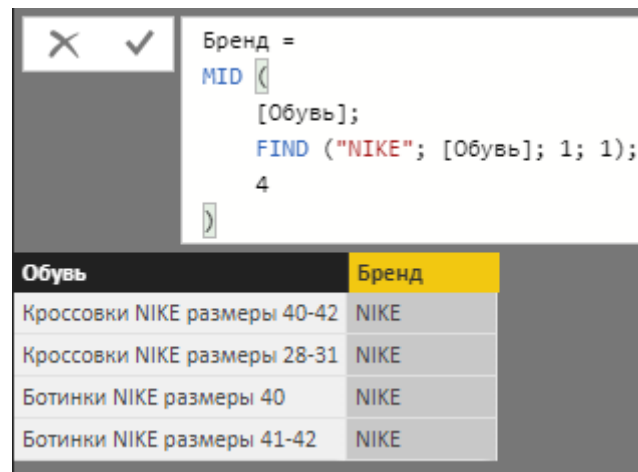
```
Бренд =  
MID (  
    [Обувь];  
    FIND ("NIKE"; [Обувь]; 1; 1);  
    4  
)
```

В первом параметре формулы мы указали столбец с текстовой информацией, которую мы исследуем.

Во втором параметре мы разместили DAX функцию FIND, которая найдет слово «NIKE» и вернет его стартовую позицию

В третьем параметре мы указали количество выводимых символов.

В результате выполнения этой формулы на основе DAX функций MID и FIND в исходной таблице в Power BI Desktop будет создан следующий вычисляемый столбец:



Бренд =

```
MID  
[Обувь];  
FIND ("NIKE"; [Обувь]; 1; 1);  
4
```

Обувь	Бренд
Кроссовки NIKE размеры 40-42	NIKE
Кроссовки NIKE размеры 28-31	NIKE
Ботинки NIKE размеры 40	NIKE
Ботинки NIKE размеры 41-42	NIKE

## DAX функция LEN

LEN () — возвращает количество символов в текстовой строке.

Синтаксис:

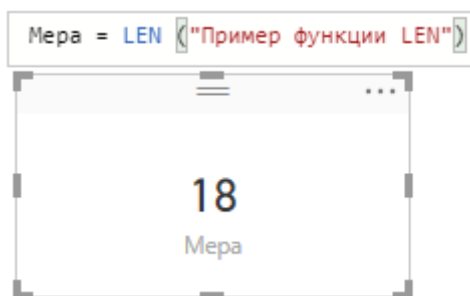
LEN ("Текст")

Где, «Текст» — любое текстовое значение, выражение или столбец с текстовыми данными. Пробелы также включаются в подсчет символов.

LEN — очень простая текстовая функция, основное предназначение которой, простой подсчет символов в текстовой строке в Power BI:

Мера = LEN ("Пример функции LEN")

Результатом выполнения этой формулы на основе DAX функции LEN, будет подсчитано количество символов, равное 18:



## DAX функции LOWER и UPPER

LOWER () – возвращает исходный текст полностью в нижнем регистре.

UPPER () – возвращает исходный текст полностью в верхнем регистре.

Синтаксис:

LOWER ("Текст")

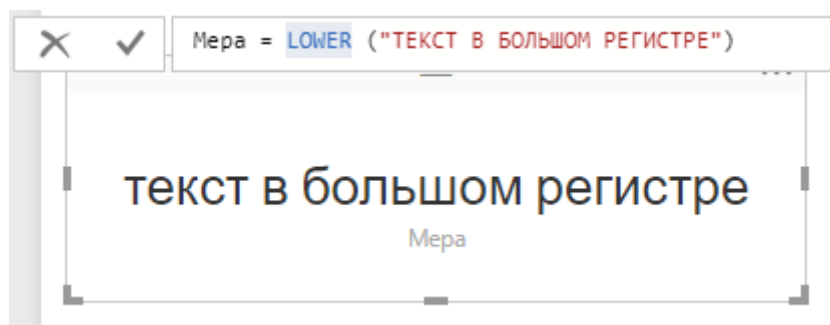
UPPER ("Текст")

Где, «Текст» — текст или столбец, содержащий текстовые значения.

Примеры формул на основе DAX функций LOWER и UPPER:

Мера = LOWER ("ТЕКСТ В БОЛЬШОМ РЕГИСТРЕ")

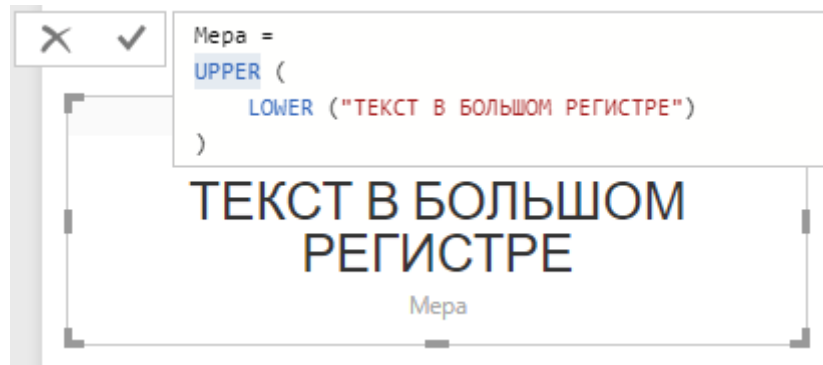
В формуле текст записан полностью в большом регистре, но функция LOWER изменит этот регистр на малый и в визуализации в Power BI отобразится текст малыми буквами:



Теперь, обернем эту формулу в функцию UPPER:

Мера =  
UPPER (  
    LOWER ("ТЕКСТ В БОЛЬШОМ РЕГИСТРЕ")  
)

Тогда произойдет интересный момент, изначально текст был записан большими буквами, LOWER поменяла регистр букв на малый, но затем, UPPER вернула все обратно, заменив малый регистр букв на большой:





## DAX функция REPLACE

REPLACE () — заменяет один текст другим, с учетом стартовой позиции и количества символов.

Синтаксис:

REPLACE ("Текст 1"; Позиция; Количество; "Текст 2")

Где:

- Текст 1 – старый текст, который нужно изменить
- Текст 2 – новый текст, которым нужно заменить старый
- Позиция – номер позиции символа, с которого нужно заменять старый текст
- Количество – количество символов, которые нужно заменить

Пример формулы на основе DAX функции REPLACE.

В [Power BI Desktop](#) имеется исходная таблица «Спорт Товары», содержащая перечисление спортивных курток бренда NIKE:


СпортивнаяОдежда
Куртка NICE
Штаны NICE
Шорты NICE

Но, в этой таблице наименование бренда записано с ошибкой — NICE. Задача — в исходной таблице создать столбец с перечислением спортивной одежды без ошибок в наименовании бренда. Попробуем решить эту задачу при помощи DAX функции REPLACE, которая сможет заменить слово NICE на NIKE.

Запишем формулу, где в первом параметре REPLACE вставим ссылку на исходный столбец с текстовой информацией, во втором параметре укажем позицию, с которой нужно начать замену символов, равную 7. В третьем параметре пропишем количество символов для замены — 4 и в четвертом параметре само новое слово «NIKE»:

Спортивная Одежда NEW = REPLACE ([СпортивнаяОдежда]; 7 ; 4; "NIKE")

Но, в Power BI, к сожалению, результат не тот, который мы ожидали:



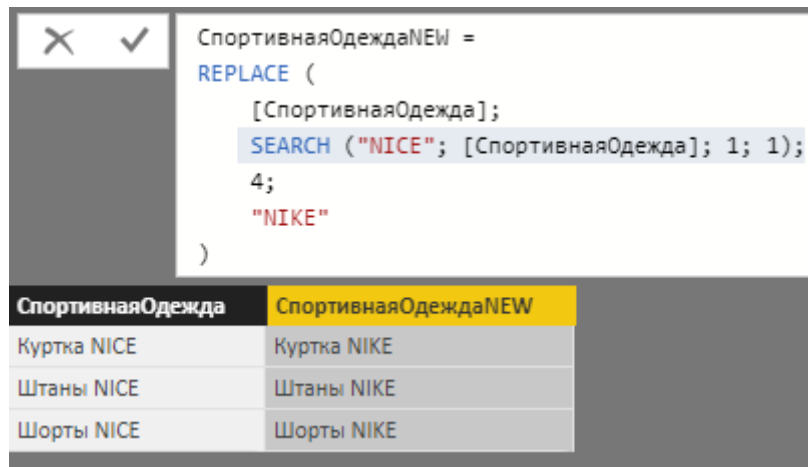
СпортивнаяОдежда	Столбец
Куртка NICE	КурткаNIKEE
Штаны NICE	Штаны NIKE
Шорты NICE	Шорты NIKE

А именно, в первой строке наименование бренда написано слитно с первым словом и вообще, имя бренда опять написано неправильно. Вся эта ошибка возникла из-за того, что мы в формуле указали 7 позицию символа, с которого нужно начать замену. Но, 7 позиция верна только для 2 и 3 строки, а для первой нужна 8 позиция, так как первое слово в первой строке длиннее на 1 символ, чем во 2 и 3 строке.

Чтобы исправить эту ошибку, дополнительно воспользуемся еще одной [функцией](#) языка DAX — SEARCH (подробно ознакомиться с этой функцией Вы можете [в данной статье](#)), которая способна найти искомое слово с ошибкой и вернуть его стартовую позицию. Исправим нашу формулу:

Спортивная Одежда NEW =  
REPLACE (  
    [СпортивнаяОдежда];  
    SEARCH ("NICE"; [СпортивнаяОдежда]; 1; 1);  
    4;  
    "NIKE"  
)

Проверим работу этой формулы в Power BI:



СпортивнаяОдеждаNEW =  
REPLACE (  
[СпортивнаяОдежда];  
SEARCH ("NICE"; [СпортивнаяОдежда]; 1; 1);  
4;  
"NIKE"  
)

СпортивнаяОдежда	СпортивнаяОдеждаNEW
Куртка NICE	Куртка NIKE
Штаны NICE	Штаны NIKE
Шорты NICE	Шорты NIKE

Как мы видим, теперь ошибка в наименовании бренда исправлена.

## DAX функция SUBSTITUTE

SUBSTITUTE () — заменяет один текст другим с учетом регистра.

Синтаксис:

SUBSTITUTE ("Текст 1"; "Текст 2"; "Текст 3"; Номер Вхождения)

Где:

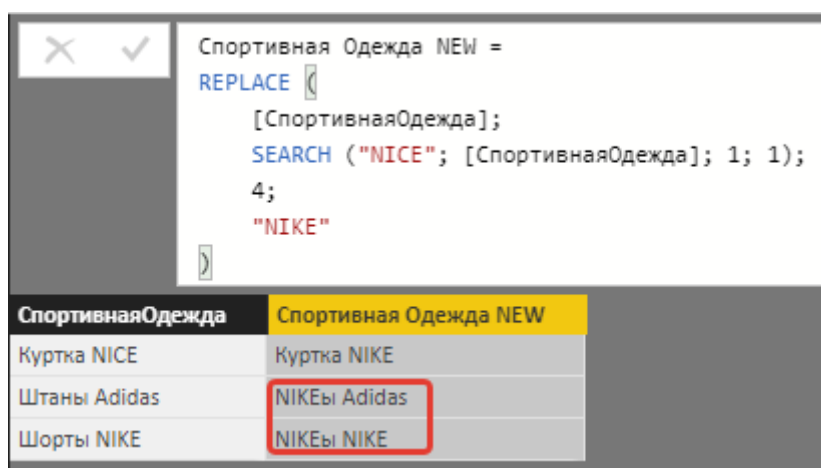
- Текст 1 – исходный текст (столбец с текстовыми данными)
- Текст 2 – часть текста, которую нужно изменить
- Текст 3 – новая часть текста
- Номер вхождения — (необязательный параметр) вхождение (повторы) того слова, которое нужно заменить. Если параметр не указан, то по умолчанию заменяются все вхождения (повторы) искомого слова

Рассмотрим формулу на основе DAX функции SUBSTITUTE, продолжая пример, который мы рассматривали выше, но сейчас, немного усложним исходные данные.

В Power BI имеется такая же исходная таблица по спортивным товарам, но сейчас там расположены разные бренды, и ошибка в наименовании бренда NIKE допущена только в одной строке:

СпортивнаяОдежда	
Куртка	NICE
Штаны	Adidas
Шорты	NIKE

Если мы попытаемся изменить ошибку при помощи функции REPLACE, которую рассматривали выше, то из-за того, что в ней нужно указывать точный номер позиции первого символа для замены, ошибку здесь изменить невозможно:



Спортивная Одежда NEW =  
`REPLACE ([СпортивнаяОдежда];  
SEARCH ("NICE"; [СпортивнаяОдежда]; 1; 1);  
4;  
"NIKE")`

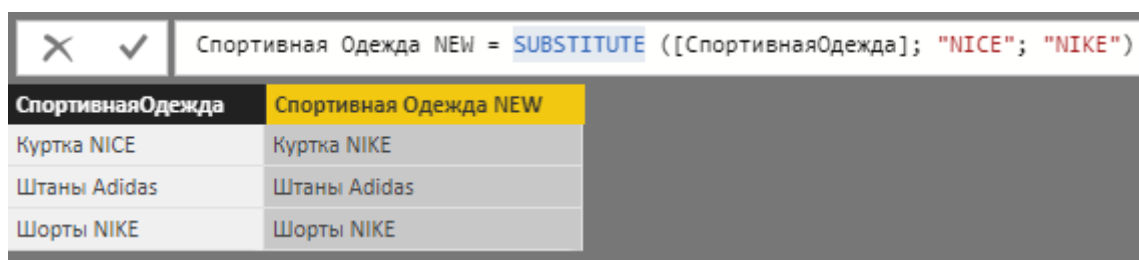
СпортивнаяОдежда	Спортивная Одежда NEW
Куртка NICE	Куртка NIKE
Штаны Adidas	NIKEы Adidas
Шорты NIKE	NIKEы NIKE

И тут, нам на помощь приходит еще одна текстовая функция [языка DAX](#)— SUBSTITUTE, в которой уже нет необходимости указывать какие-либо стартовые позиции. В ней просто нужно указать именно то конкретное слово, которое нужно изменить. Напишем формулу с участием функции SUBSTITUTE:

Спортивная Одежда NEW = SUBSTITUTE ([СпортивнаяОдежда]; "NICE"; "NIKE")

В первом параметре этой функции мы указали исходный столбец с текстовыми данными, во втором — искомое текстовое выражение и в третьем — текст для замены.

В итоге, в Power BI Desktop мы можем наблюдать созданный вычисляемый столбец с информацией без каких-либо ошибок:



Спортивная Одежда NEW = `SUBSTITUTE ([СпортивнаяОдежда]; "NICE"; "NIKE")`

СпортивнаяОдежда	Спортивная Одежда NEW
Куртка NICE	Куртка NIKE
Штаны Adidas	Штаны Adidas
Шорты NIKE	Шорты NIKE

## DAX функция REPT

REPT () — функция повтора текста с заданным количеством повторов.

Синтаксис:

REPT ("Текст"; Количество Повторов)

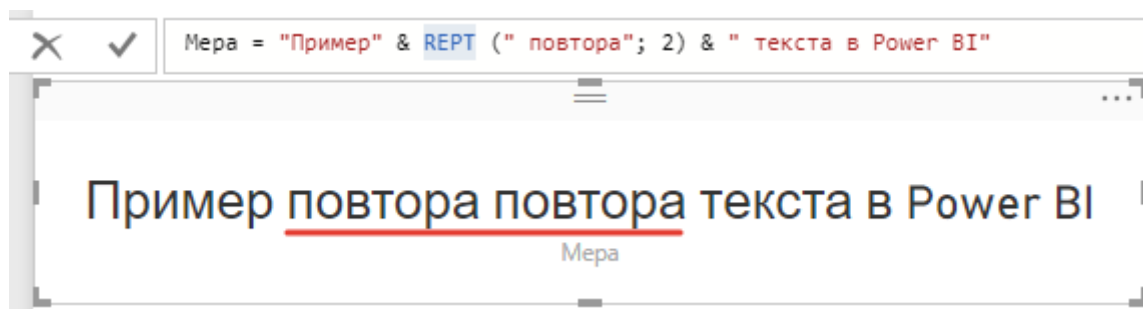
Где:

- Текст – исходный текст для повтора
- Количество Повторов – число повторов исходного текста

Пример формулы на основе DAX функции REPT:

Мера = "Пример" & REPT (" повтора"; 2) & " текста в Power BI"

В этой формуле мы использовали конструкцию объединения текстов при помощи оператора &, входящего в [язык DAX](#). И внутри этого текста вставили повтор слова при помощи функции REPT. Результатом выполнения этой формулы, будет единая текстовая строка с внутренним повтором одного слова:



## DAX функция TRIM

TRIM () — удаляет все лишние пробелы из текста, кроме одинарных пробелов между словами

Синтаксис:

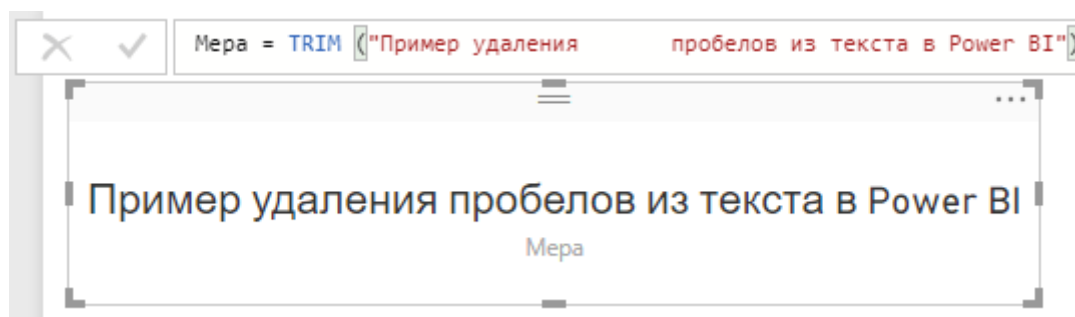
TRIM ("Текст")

Где, «Текст» — любое текстовое значение, выражение или столбец с текстовыми данными.

Пример формулы на основе DAX функции TRIM:

Мера = TRIM ("Пример удаления      пробелов из текста в Power BI")

Результатом выполнения этой формулы на основе функции TRIM, будет текстовая строка с единичными пробелами между словами:



Как мы видим из примера, функция TRIM удалила все лишние множественные пробелы из текста.

## DAX функция EXACT

EXACT () — сравнение двух текстовых строк. Если строки одинаковые, то функция возвращает TRUE, если разные, то FALSE. EXACT учитывает регистр букв, но не учитывает их формат.

Синтаксис:

EXACT ("Текст 1"; "Текст 2")

Где, «Текст 1» и «Текст 2» — сравниваемые тексты или столбцы, содержащие текстовые значения.

Пример формулы на основе DAX функции EXACT.

Мера = EXACT (""; "")

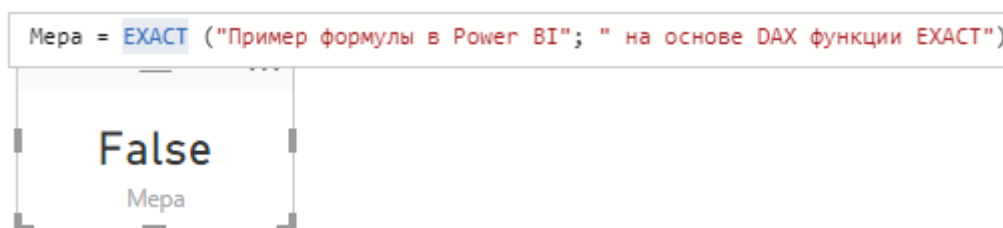
Данная формула вернет значение TRUE, так как оба текста одинаковые, а вернее, просто пустые:



Но, если мы изменим формулу, а точнее, введем разные тексты:

Мера = EXACT ("Пример формулы в Power BI"; " на основе DAX функции EXACT")

то, естественно, EXACT вернет значение FALSE:





## 2. ЛОГИЧЕСКИЕ ФУНКЦИИ DAX

## DAX функция IF

IF () — если. Выполняет проверку условия. Если условие равно логическому значению TRUE, то [функция](#) выполняет одно выражение. Если же условие равно логическому значению FALSE, то функция выполняет второе выражение.

Синтаксис:

IF (Условие; Выражение 1; Выражение 2)

Где:

- Условие — логическое условие, результат которого равен либо значению TRUE, либо — FALSE
- Выражение 1 — выражение, которое вступит в работу, если результат проверки условия будет равен TRUE
- Выражение 2 — выражение, которое вступит в работу, если результат проверки условия будет равен FALSE

Пример [формулы](#) на основе DAX функции IF.

В модели данных [Power BI Desktop](#) имеется исходная таблица, состоящая из 2 столбцов, содержащих числовые значения. Причем, в одной из строк второго столбца находится значение, равное 0:

Столбец1	Столбец2
1	1
2	2
3	0
4	4

Создадим в этой таблице третий вычисляемый столбец, на основе формулы деления первого столбца на второй:

✕ ✓ Столбец = [Столбец1] / [Столбец2]		
Столбец1	Столбец2	Столбец
1	1	1
2	2	1
3	0	∞
4	4	1

В результате, в ячейке третьей строки у нас возникла ошибка деления на 0 и Power BI вывел там знак бесконечности.

Для того, чтобы обработать эту ошибку, воспользуемся рассматриваемой DAX функцией IF (условие если). То есть, если ошибок нет, то функция IF должна вернуть вычисляемое выражение, если ошибки есть, то она должна вывести пустое значение. Новая формула деления будет такой:

```
Столбец =
IF (
    ISERROR ([Столбец1] / [Столбец2]);
    BLANK();
    [Столбец1] / [Столбец2]
)
```

В этой формуле, кроме функции проверки условия IF, мы воспользовались еще несколькими функциями [языка DAX](#). А именно, функцией [ISERROR](#), которая проверила входящее внутрь нее выражение (в нашем случае, деления одного столбца на другой), и если в этом выражении возникает ошибка, то она возвращает TRUE, а иначе FALSE.

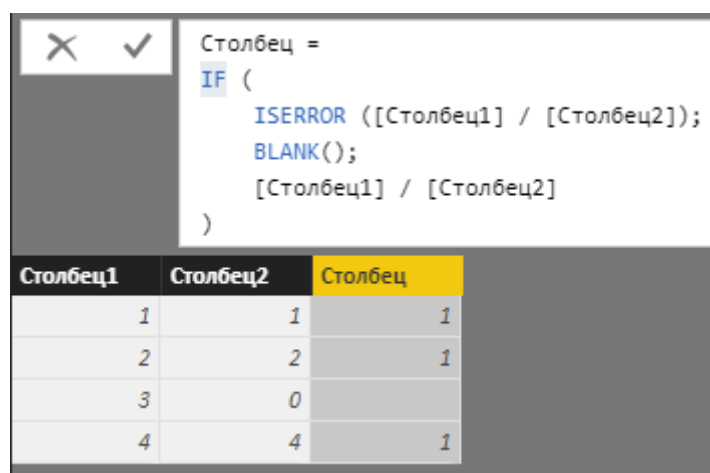
А также, мы воспользовались функцией [BLANK](#), которая возвращает пустое значение.

Как итог, если прочитать всю эту формулу на основе функции IF, то получается следующее: если во время деления происходит ошибка, то возвращается логическое значение TRUE, если ошибки нет, то FALSE.

Соответственно, функция IF будет выполнять выражение из второго параметра, тогда, когда условие возвратит TRUE (в нашем случае, ошибку) и выражение из третьего параметра, тогда, когда условие возвратит FALSE (в нашем случае, ошибки нет).

И именно поэтому, если при делении будет ошибка деления на 0, возвратится пустое значение, так как во втором параметре функции IF вызывается BLANK. И, если ошибки деления нет, то возвратится само деление столбцов из третьего параметра IF.

Посмотрим на выполнение этой формулы в Power BI, чтобы убедиться в правильности наших измышлений:



Столбец =

```
IF (
    ISERROR ([Столбец1] / [Столбец2]);
    BLANK();
    [Столбец1] / [Столбец2]
)
```

Столбец1	Столбец2	Столбец
1	1	1
2	2	1
3	0	
4	4	1

Как мы видим, ошибка обработана, и функция IF при помощи двух других DAX функций вывела на ее месте пустое значение.

## DAX функции TRUE и FALSE

TRUE () — возвращает логическое значение TRUE (истина, да, правда, правильно).

FALSE () — возвращает логическое значение FALSE (ложь, нет, неправда, неправильно).

Синтаксис:

TRUE ()

FALSE ()

Пример формулы на основе DAX функций TRUE () и FALSE ().

Данные функции обычно используются для каких-то промежуточных вычислений, поэтому, пример у нас будет соответствующий.

В Power BI имеется исходная таблица с числовыми данными:

Столбец1
1
2
0
3

Допустим, что для каких-то промежуточных целей, в этой таблице нам нужно создать еще один столбец, который будет содержать значения TRUE, если исходный столбец содержит числовые значения неравные 0. И значения FALSE, если исходный столбец содержит числовые значения равные 0. Напишем соответствующую формулу на основе DAX функций TRUE () и FALSE ():

Столбец =

IF (

[Столбец1] <> 0;

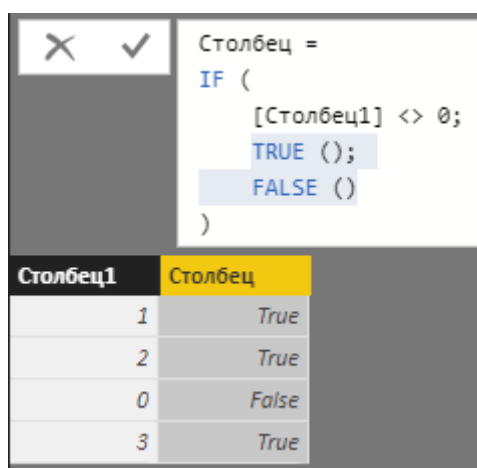
TRUE ();

FALSE ()

)

Естественно, так как в этой формуле нам нужно проверять условие, то мы, дополнительно, использовали ранее рассмотренную функцию IF.

Результатом выполнения этой формулы с участием функций TRUE () и FALSE () в Power BI был создан вычисляемый столбец с соответствующими логическими значениями TRUE и FALSE для промежуточных целей:



Столбец1	Столбец
1	True
2	True
0	False
3	True

## DAX функция SWITCH

SWITCH () — переключатель. Сравнивает результат вычисления выражения с заранее подготовленными значениями и возвращает итоговый результат, относительно найденного подготовленного значения.

Синтаксис:

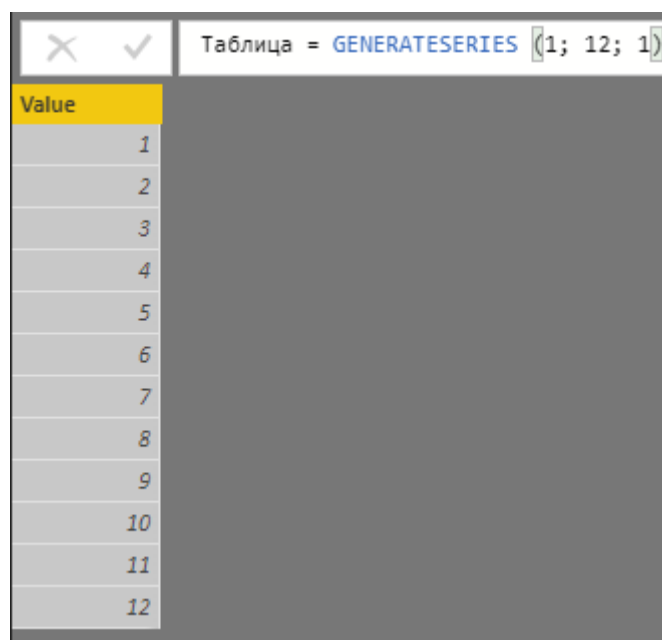
```
SWITCH (  
    Выражение;  
    Значение 1; Результат 1;  
    Значение 2; Результат 2;  
    Значение N; Результат N;  
    Иначе  
)
```

Где:

- Выражение — вычисляемое выражение, результат которого сравнивается с заранее подготовленными значениями
- Значение — заранее подготовленное значение
- Результат — заранее подготовленный результат, который возвратится, если результат вычисления выражения будет равен соответствующему заранее подготовленному значению
- Иначе — заранее подготовленный результат, который возвратится, если результат вычисления выражения не будет равен ни одному заранее подготовленному значению

Пример формулы на основе DAX функции SWITCH.

В Power BI имеется вычисляемая таблица, созданная на основе DAX функции [GENERATESERIES](#). Таблица имеет 12 строк с числовыми значениями от 1 до 12:



Value
1
2
3
4
5
6
7
8
9
10
11
12

Напишем формулу вычисляемого столбца с использованием функции SWITCH, в котором будут отображаться названия месяцев, соответствующие цифрам из столбца [Value] :

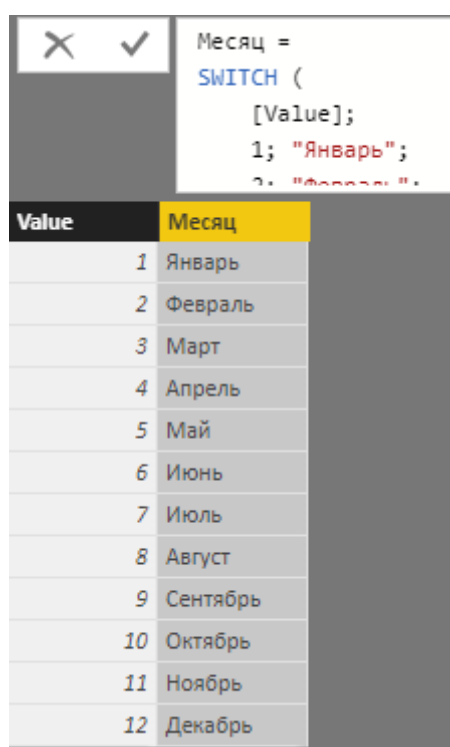
```
Месяц =  
SWITCH (  
    [Value];  
    1; "Январь";  
    2; "Февраль";  
    3; "Март";  
    4; "Апрель";  
    5; "Май";  
    6; "Июнь";  
    7; "Июль";  
    8; "Август";  
    9; "Сентябрь";  
    10; "Октябрь";  
    11; "Ноябрь";  
    "Декабрь"  
)
```



В первом параметре функции SWITCH, в качестве исполняемого выражения, мы разместили ссылку на столбец [Value], который содержит числовые значения от 1 до 12. В последующих строках формулы у нас заранее заготовлены возможные значения (1-11) результата вычисления столбца [Value] и рядом с ними соответствующие итоговые результаты (наименования месяцев).

Если функция SWITCH не найдет в этих значениях (1-11) того результата, который получится в вычислении столбца [Value] (в нашем примере, это 12), то, как итог, выведется значение «Иначе» из последнего параметра, то есть, в нашем случае, текст «Декабрь».

В итоге, в Power BI, на основе этой формулы будет создан следующий вычисляемый столбец:



Value	Месяц
1	Январь
2	Февраль
3	Март
4	Апрель
5	Май
6	Июнь
7	Июль
8	Август
9	Сентябрь
10	Октябрь
11	Ноябрь
12	Декабрь

Как мы можем наблюдать из скриншота, создание столбца на основе работы DAX функции SWITCH прошло удачно и каждой цифре соответствует свое наименование месяца.

## DAX функция IFERROR

IFERROR () — если ошибка. Производит вычисление выражения и если во время вычисления возникла ошибка, то функция выводит значение из второго параметра, если ошибок нет, то возвращается результат вычисления самого выражения.

Синтаксис: IFERROR (Выражение; Значение Если Ошибка)

Пример формулы 1: IFERROR (6 / 2; BLANK() )

Результат: 3

В итоге возвратился результат вычисления самого выражения, так как само выражение «6 / 2» вычисляется без ошибок и равно 3.

Пример формулы 2: IFERROR (6 / 0; BLANK() )

Результат: пусто

Так как на 0 делить нельзя, то результатом вычисления выражения будет ошибка и в этом случае IFERROR выведет значение из второго параметра, где в нашем случае стоит функция [BLANK](#), которая, в свою очередь, выводит пустое значение.

То есть, функцией IFERROR можно обрабатывать ошибки в формулах, где возможно деление на 0. Но, кроме этого, можно при помощи нее застраховываться и от любых других ошибок, возникающих при выполнении формул в DAX.

## DAX функции AND и OR

AND () — и. Возвращает логическое значение TRUE, если оба параметра, входящие в эту функцию, в режиме «и», также, возвращают TRUE. Если же хотя бы один параметр возвращает FALSE, то сама функция, также, возвращает логическое значение FALSE. Аналогом в языке DAX является логический оператор && (и).

OR () — или. Возвращает логическое значение TRUE, если хотя бы один из двух параметров, входящих в эту функцию, также, возвращает логическое значение TRUE. Если же оба параметра возвращают FALSE, то сама функция, также, возвращает логическое значение FALSE. Аналогом в языке DAX является логический оператор || (или).

Синтаксис:

AND (Значение 1; Значение 2)

OR (Значение 1; Значение 2)

Данные функции обычно используют тогда, когда нужно в каких-либо условиях или фильтрах использовать в одном условии или параметре сразу 2 или более условий. Если нужно больше 2 условий, то тогда эти функции вкладываются друг в друга, например, так:

AND (Значение 1; AND (Значение 2; Значение 3))

OR (Значение 1; OR (Значение 2; Значение 3))

Но, в этих случаях предпочтительнее использовать уже не данные функции AND или OR, а соответствующие операторы DAX, например, так:

Сочетание в режиме "и": Значение 1 && Значение 2 && Значение 3

Сочетание в режиме "или": Значение 1 || Значение 2 || Значение 3

Примеры формул на основе DAX функций AND и OR.

В Power BI имеется следующая исходная таблица с числовыми значениями:

Столбец1
1
2
0
3

В качестве примера рассмотрим ситуацию, в которой нам понадобилось создать столбец для промежуточных вычислений, содержащий в своих строках значения из исходного столбца, но только те, которые больше 2 и меньше 4. Для создания этой формулы нам понадобятся функции IF и AND:

```
Столбец =  
IF (  
    AND ([Столбец1] > 2; [Столбец1] < 4);  
    [Столбец1];  
    BLANK ()  
)
```

В этой формуле при помощи DAX функции AND у нас получилось создать двойное условие. Если оно истинно, то возвращаются значения из исходного столбца, если оно не выполняется, то выводится пустое значение функцией BLANK.

Результат этой формулы в Power BI будет следующим:

✕ ✓		Столбец =	
		IF (	
		AND ([Столбец1] > 2; [Столбец1] < 4);	
		[Столбец1];	
		BLANK ()	
		)	
Столбец1	Столбец		
1			
2			
0			
3	3		

Если же мы изменим эту формулу, а именно, заменим AND функцией OR:

```
Столбец =  
IF (  
    OR ([Столбец1] > 2; [Столбец1] < 4);  
    [Столбец1];  
    BLANK ()  
)
```

то, в данном случае, у нас условия будут сочетаться в режиме «или», то есть, значения исходного столбца должны быть или больше 2, или меньше 4:

✕ ✓		Столбец =	
		IF (	
		OR ([Столбец1] > 2; [Столбец1] < 4);	
		[Столбец1];	
		BLANK ()	
		)	
Столбец1	Столбец		
1	1		
2	2		
0	0		
3	3		

Как мы видим из скриншота в Power BI — при таком условии, возвращаются все значения из исходного столбца, так как значение 3 удовлетворяет условию больше 2 и значения 0, 1, 2 — удовлетворяют условию меньше 4.

## DAX функция NOT

NOT () — заменяет логические значения друг другом. Если переданное в функцию выражение равно логическому значению FALSE, то функция возвращает TRUE и наоборот, если переданное в функцию выражение равно логическому значению TRUE, то функция возвращает FALSE. Иначе говоря, меняет FALSE на TRUE и наоборот.

Синтаксис:

NOT (Выражение)

Какого-то большого практического смысла в функции NOT нет, но, все же, она позволяет несколько упростить чтение логики.

Например, когда мы выше рассматривали функцию IF, то составляли такую формулу:

```
IF (  
    ISERROR ([Столбец1] / [Столбец2]);  
    BLANK();  
    [Столбец1] / [Столбец2]  
)
```

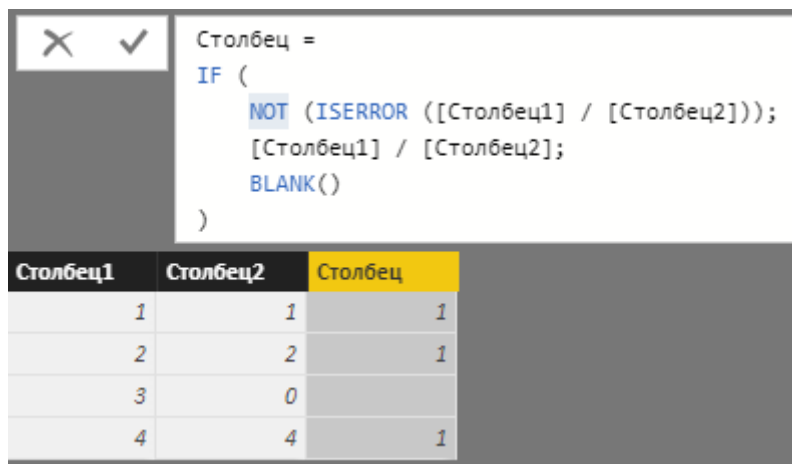
Обычно, функцию IF читают так: выполнить второй параметр, если условие истинно (TRUE) и выполнить третий параметр, если условие ложно (FALSE).

Но, в нашей формуле, изначально в условии проверяется наличие ошибки. И если есть ошибка, то возвращается истина (TRUE), что несколько нелогично, если смотреть с человеческой стороны... логичнее понимать так — если есть ошибка, то значит ложь (FALSE).

Ситуацию, в этом случае, может помочь исправить функция NOT, которая заменит TRUE на FALSE в самом условии. В итоге, формула будет такой:

```
Столбец =  
IF (  
    NOT (ISERROR ([Столбец1] / [Столбец2]));  
    [Столбец1] / [Столбец2];  
    BLANK()  
)
```

Теперь, с точки зрения человеческого понимания, формула стала более логичной и понятной. При этом, в Power BI результат выполнения самой формулы с учетом функции NOT не изменился:



Столбец1	Столбец2	Столбец
1	1	1
2	2	1
3	0	
4	4	1



# 3. МАТЕМАТИЧЕСКИЕ ФУНКЦИИ DAX

## DAX функция DIVIDE

DIVIDE () — производит деление с обработкой ошибки «деление на 0». Обработка ошибки заключается в выводе альтернативного результата в случае возникновения ситуации деления на ноль.

Синтаксис:

DIVIDE (Делимое Число; Делитель; Альтернатива)

Где, альтернатива — (необязательный параметр) значение, которое нужно вывести в случае ошибки деления на ноль (0). По умолчанию выводится пустое значение [BLANK \(\)](#).

Пример формулы на основе DAX функции DIVIDE.

В [Power BI Desktop](#) имеется исходная таблица «Реклама», содержащая по каждой дате затраты на рекламу и прибыль, полученную от продаж с этой рекламы:

Дата	Затраты	Прибыль
14 августа 2018 г.	12000	30000
15 августа 2018 г.	0	18000
16 августа 2018 г.	5000	20000

Задача — создать в Power BI меру расчета коэффициента ROI (окупаемости затрат на рекламу). Данный коэффициент рассчитывается как сумма всей прибыли деленная на сумму всех затрат.

Сумму значений мы можем рассчитать при помощи DAX функции [SUM](#).

В итоге, формула расчета ROI будет такой:

$$ROI = SUM ('Реклама'[Прибыль]) / SUM ('Реклама'[Затраты])$$

То есть, мы сложили всю прибыль, сложили все затраты и затем разделили сумму прибыли на сумму затрат.

Вроде бы, все хорошо, но, если мы вынесем эту меру в [отчеты Power BI](#) и посмотрим ROI по дням, то в визуализации в одной из строк будет отображаться непонятное слово «Бесконечность»:

ROI = SUM ('Реклама'[Прибыль]) / SUM ('Реклама'[Затраты])

Дата	Затраты	Прибыль	ROI
14 августа 2018 г.	12000	30000	2,50
15 августа 2018 г.	0	18000	Бесконечность
16 августа 2018 г.	5000	20000	4,00
Всего	17000	68000	4,00

А все дело в том, что у нас произошла ошибка деления на ноль — прибыль 18000 была разделена на затраты, равные 0. И Power BI вместо этой ошибки вывела значение «Бесконечность».

Для того, чтобы исправить эту ситуацию, в формуле расчета ROI вместо обычного деления нужно использовать рассматриваемую DAX функцию DIVIDE. Она позволит нам произвести деление и обработать все ошибки, возникающие при делении на 0. И вместо значения «Бесконечность» вывести то значение, которое нам нужно, например, пустое значение BLANK ().

В итоге, [формула](#) расчета ROI на основе функции DIVIDE будет такая:

```
ROI =
DIVIDE (
    SUM ('Реклама'[Прибыль]);
    SUM ('Реклама'[Затраты])
)
```

Где, в первом параметре функции DIVIDE мы указали сумму прибыли, которую нужно разделить, во втором параметре — сумму затрат, на которую делится сумма прибыли. Третий параметр указывать не

стали, так как по умолчанию он равен функции BLANK (), что нам и нужно.

В результате, визуализация в Power BI теперь работает правильно:

```
ROI =  
DIVIDE (  
    SUM ('Реклама' [Прибыль]);  
    SUM ('Реклама' [Затраты])  
)
```

Дата	Затраты	Прибыль	ROI
14 августа 2018 г.	12000	30000	2,50
15 августа 2018 г.	0	18000	
16 августа 2018 г.	5000	20000	4,00
Всего	17000	68000	4,00

Используйте функцию деления DIVIDE всегда, когда имеется хоть малейший потенциал значения нуля в делителе Вашей формулы.

## DAX функция QUOTIENT

QUOTIENT () — выполняет деление чисел, входящих в параметры функции и возвращает целочисленную часть от деления.

Синтаксис:

QUOTIENT (Делимое Число; Делитель)

Пример формулы на основе DAX функции QUOTIENT.

В Power BI имеется исходная таблица «Общая Длительность Звонков», содержащая в себе информацию по общей сумме секунд всех разговоров менеджеров:

Менеджер	КоличествоСекунд
Петров	16784
Сидоров	18569

Требуется рассчитать это количество в целых часах.

Для этого, общее количество секунд нужно разделить на количество секунд в часе (3600). Но, в результате этого деления получится дробное число, а нам нужна только целая часть результата деления. В этой ситуации нам поможет функция QUOTIENT.

В итоге, формула расчета общей длительности звонков в целых часах будет такой:

Длительность В Часах =

```
QUOTIENT (  
    SUM ('ОбщаяДлительностьЗвонков'[КоличествоСекунд]);  
    3600  
)
```

И в отчете Power BI по каждому менеджеру эта мера выведет количество целых часов, которые затратили менеджеры на звонки:

```
ДлительностьВЧасах =  
QUOTIENT (  
    SUM ('ОбщаяДлительностьЗвонков' [КоличествоСекунд]);  
    3600  
)
```

Менеджер	ДлительностьВЧасах
Петров	4
Сидоров	5
Всего	9

# DAX функция MOD

MOD () — возвращает остаток от деления со знаком делителя.

Синтаксис:

MOD (Делимое Число; Делитель)

В качестве разбора формулы на основе DAX функции MOD продолжим рассматривать прошлый пример с расчетом длительности разговора менеджеров по телефону в целых часах на основе общей суммы в секундах.

Разделив общее количество секунд на 3600 при помощи QUOTIENT, мы получили целую часть от деления. Но, также, при этом делении мы можем получить и остаток от этого целочисленного деления (в нашем случае, это оставшееся количество секунд за вычетом целых часов из общего количества секунд).

И сделать это можно функцией MOD:

Остаток Секунд =

```
MOD (
    SUM ('ОбщаяДлительностьЗвонков'[КоличествоСекунд]);
    3600
)
```

Давайте проверим эту формулу в Power BI:

ОстатокСекунд =			
MOD (			
SUM ('ОбщаяДлительностьЗвонков'[КоличествоСекунд]);			
3600			
)			

Менеджер	КоличествоСекунд	ДлительностьВЧасах	ОстатокСекунд
Петров	16784	4	2384
Сидоров	18569	5	569
Всего	35353	9	2953

Действительно, если мы рассмотрим менеджера Петров из визуализации выше, то 16784 (общее количество секунд) — 14400 секунд (4 часа \* 3600) = 2384 (остаток секунд). И функция MOD нам также вывела данное значение 2384.

Теперь, это получившееся значение (остаток секунд) можно еще раз разделить функцией QUOTIENT на 60 и мы получим целое количество минут из этого остатка секунд:

```

Остаток В Минутах =
QUOTIENT (
    MOD (
        SUM ('ОбщаяДлительностьЗвонков'[КоличествоСекунд]);
        3600
    );
    60
)
    
```

В Power BI вычисление этой формулы будет таким:

ОстатокВМинутах =			
QUOTIENT(			
MOD (			
SUM ('ОбщаяДлительностьЗвонков'[КоличествоСекунд]);			
3600			
);			
60			
)			

Менеджер	КоличествоСекунд	ДлительностьВЧасах	ОстатокВМинутах
Петров	16784	4	39
Сидоров	18569	5	9
Всего	35353	9	49

Давайте проверим все вычисления на примере менеджера Петрова: общее количество секунд у него 16784, а 4 часа и 39 минут, это 16740 секунд (60\*(4\*60+39)). Все правильно 16740 входит в общее количество секунд 16784.



Теперь, осталось вычислить окончательный остаток в секундах:

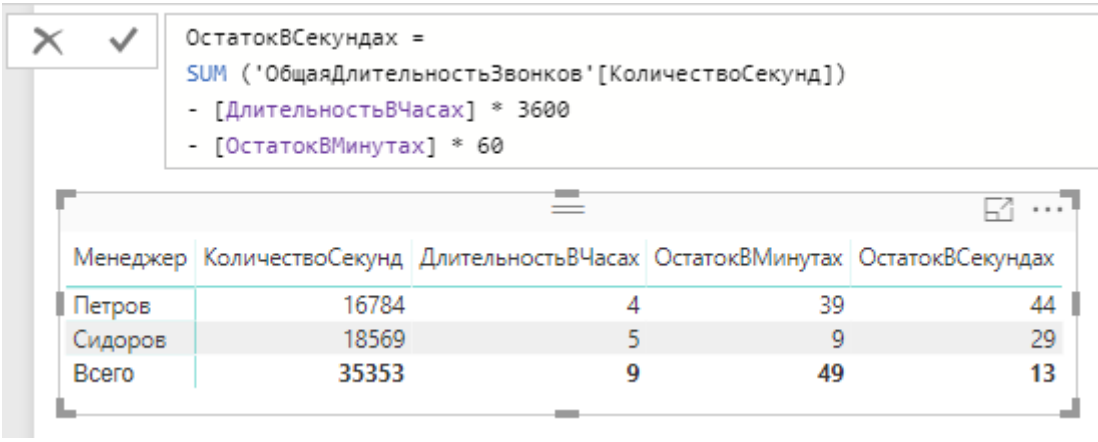
ОстатокВСекундах =

SUM ('ОбщаяДлительностьЗвонков'[КоличествоСекунд])

- [ДлительностьВЧасах] \* 3600

- [ОстатокВМинутах] \* 60

И в Power BI все это будет выглядеть так:



The screenshot shows the Power BI interface. At the top, a formula bar displays the DAX formula for 'ОстатокВСекундах':

```
ОстатокВСекундах =  
SUM ('ОбщаяДлительностьЗвонков'[КоличествоСекунд])  
- [ДлительностьВЧасах] * 3600  
- [ОстатокВМинутах] * 60
```

Below the formula bar is a table with the following data:

Менеджер	КоличествоСекунд	ДлительностьВЧасах	ОстатокВМинутах	ОстатокВСекундах
Петров	16784	4	39	44
Сидоров	18569	5	9	29
Всего	35353	9	49	13

Напоследок, осталось навести некий «косметический дизайн» — совместим часы, минуты и секунды в единое значение «часы : минуты : секунды», для этого, воспользуемся оператором объединения [в языке DAX](#) — & и текстом с двоеточием («:»):

ДлительностьЗвонков =

[ДлительностьВЧасах] & ":" & [ОстатокВМинутах] & ":" & [ОстатокВСекундах]

Итоговая визуализация, демонстрирующая совместную работу DAX функций QUOTIENT и MOD в Power BI будет такая:

ДлительностьЗвонков =  
[ДлительностьВЧасах] & ":" & [ОстатокВМинутах] & ":" & [ОстатокВСекундах]

Менеджер	ДлительностьЗвонков
Петров	4:39:44
Сидоров	5:9:29
Всего	9:49:13

Таким образом, общее количество секунд, затраченное менеджером на звонки мы превратили в соответствующее отображение в часах, минутах и секундах.

Давайте проверим все вычисления, так сказать, на калькуляторе, на примере менеджера Сидорова:

Длительность звонков = 5:9:29, то есть — 5 часов, 9 минут и 29 секунд, что равно 18569 секунд ( $5 \cdot 3600 + 9 \cdot 60 + 29$ ). А это, в свою очередь, равно исходной сумме секунд по менеджеру Сидоров (18569 секунд).

## DAX функция SQRT

SQRT () — [функция](#) квадратного корня из числа.

Синтаксис:

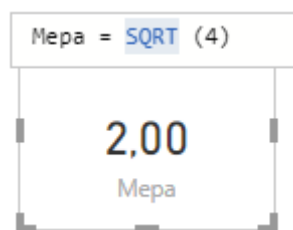
**SQRT (Число)**

Где, число — числовое значение или столбец, содержащий числовые значения.

Пример [формулы](#) на основе DAX функции SQRT:

**Мера = SQRT (4)**

Результатом выполнения этой формулы с участием SQRT будет значение, равное 2:



## DAX функция POWER

POWER () — возводит число в степень.

Синтаксис:

POWER (Число; Степень)

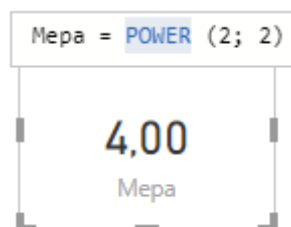
Где:

- число — числовое значение или столбец, содержащий числовые значения
- степень — число необходимой степени

Пример формулы на основе DAX функции POWER:

Мера = POWER (2; 2)

Результатом выполнения этой формулы (2 в степени 2) с участием функции POWER будет значение, равное 4:



## DAX функция ABS

ABS () — функция абсолютного значения числа. Иначе говоря, преобразует отрицательное число в положительное.

Синтаксис:

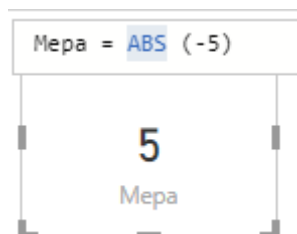
ABS (Число)

Где, число — числовое значение или столбец, содержащий числовые значения.

Пример формулы на основе DAX функции ABS:

Мера = ABS (-5)

Результатом выполнения этой формулы с участием функции ABS будет положительное значение, равное 5:



## DAX функция SIGN

**SIGN ()** — возвращает знак числа в столбце или числа, получившегося в результате вычисления выражения. Значение 1 соответствует положительному числу, значение 0 соответствует нулю и -1 соответствует отрицательному числу.

Синтаксис:

**SIGN (Число)**

Где, число — числовое значение или столбец, содержащий числовые значения.

Примеры формул на основе DAX функции SIGN:

Мера 1 = SIGN (-5)

Мера 2 = SIGN (0)

Мера 3 = SIGN (5)

Результатом выполнения этих формул с участием функции SIGN будут 3 значения -1 (соответствует отрицательному значению), 0 (соответствует нулевому значению), 1 (соответствует положительному значению):

Мера1 = SIGN (-5)		
-1	0	1
Мера1	Мера2	Мера3

## DAX функция EXP

EXP () — возводит число E (2,71828182845904) в нужную степень.

Синтаксис:

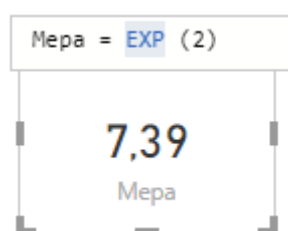
EXP (Степень)

Где, степень — числовое значение или столбец, содержащий числовые значения нужной степени

Пример формулы на основе DAX функции EXP:

Мера = EXP (2)

Результатом выполнения этой формулы с участием функции EXP будет значение, равное 7.39 (2,71828182845904 в степени 2):



## DAX функция FACT

FACT () — факториал числа (произведение последовательности целых чисел начиная с 1 и до указанного числа).

Синтаксис:

FACT (Число)

Где, число — числовое значение или столбец, содержащий числовые значения, указывающее для какого числа производить факториал.

Пример формулы на основе DAX функции FACT:

Мера = FACT (3)

Результатом выполнения этой формулы с участием функции FACT будет значение, равное 6 (произведение ряда последовательных чисел 1\*2\*3):





## DAX функции LN, LOG, LOG10

LN () — вычисляет натуральный логарифм числа по константе E (2,71828182845904)

LOG () — вычисляет логарифм числа по заданному в параметре функции основанию.

LOG10 () — вычисляет логарифм числа по основанию 10.

Синтаксис:

LN (Число)

LOG (Число; Основание)

LOG10 (Число)

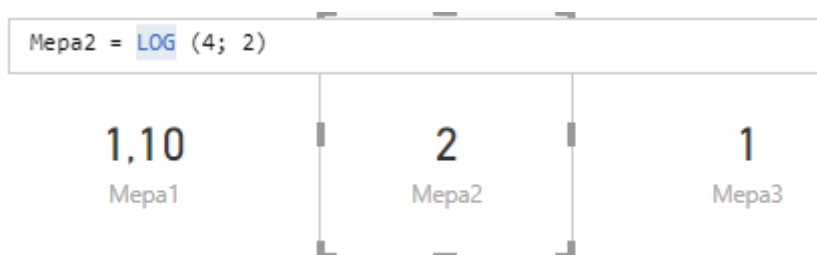
Примеры формул на основе DAX функций LN, LOG и LOG10:

Мера 1 = LN (3)

Мера 2 = LOG (4; 2)

Мера 3 = LOG10 (7)

Результатом выполнения этих формул с участием функций LN, LOG и LOG10 будут значения, равные 1.1, 2, 1:

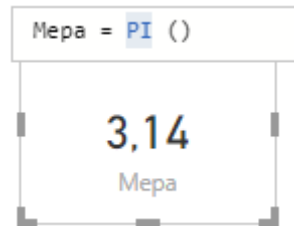


## DAX функция PI

PI () — возвращает значение Пи (3,14159265358979)

Синтаксис:

PI ()



## DAX функция INT

INT () — производит округление числа до ближайшего целого в сторону уменьшения.

Синтаксис:

INT (число)

Где, число — числовое значение или столбец с числовыми значениями.

Для демонстрации округления дробных чисел [функцией](#) INT, рассмотрим примеры формул на основе исходной таблицы в [Power BI Desktop](#), содержащей разные положительные и отрицательные числовые значения:

Числа
233,786
212,375
-212,375
-233,786

Создадим в этой таблице вычисляемый столбец на основе DAX функции INT по следующей [формуле](#):

Пример INT = INT ([Числа])

Результатом выполнения этой формулы будет создан столбец с округленными исходными числами:

Пример INT = INT ([Числа])	
Числа	Пример INT
233,786	233
212,375	212
-212,375	-213
-233,786	-234

Из скриншота мы видим, что INT, действительно, округлила числа до ближайшего целого числа в меньшую сторону, то есть, число 233,786 округлилось до 233, а — 233,786 уже округлилось до -234.

## DAX функция TRUNC

TRUNC () — округляет число до целого, обрезая его дробную часть

Синтаксис: TRUNC (Число; ТО)

Где:

- Число — числовое значение или столбец с числовыми значениями
- ТО — (необязательный параметр) число, задающее точность округления. По умолчанию = 0. Если ТО положительное, то оно отсекает дробную часть, если отрицательное, то округляет правую часть целого числа относительно запятой, если равно 0, то возвращает само целое число, без дроби

Рассмотрим примеры формул с участием DAX функции TRUNC на основе все той же исходной таблицы в Power BI:

Пример TRUNC (1) = TRUNC ([Числа]; 1)

Пример TRUNC (0) = TRUNC ([Числа]; 0)

Пример TRUNC (-1) = TRUNC ([Числа]; -1)

Пример TRUNC (-2) = TRUNC ([Числа]; -2)

Пример TRUNC (-3) = TRUNC ([Числа]; -3)

Пример TRUNC (1) = TRUNC ([Числа]; 1)						
Числа	Пример TRUNC (1)	Пример TRUNC (0)	Пример TRUNC (-1)	Пример TRUNC (-2)	Пример TRUNC (-3)	
233,786	233,7	233	230	200	0	
212,375	212,3	212	210	200	0	
-212,375	-212,3	-212	-210	-200	0	
-233,786	-233,7	-233	-230	-200	0	

Как мы видим из примера, при значении точности округления 1, отсекается дробная часть и от нее остается 1 цифра, если ТО равно 0, то отсекается полностью вся дробь, и если отрицательная, числа по правую сторону от запятой округляются до нуля.

# DAX функция ROUND

ROUND () — производит математическое округление числа до указанной точности округления.

ROUND очень похожа на функцию TRUNC, только TRUNC — просто обрезает дробную часть, а ROUND именно округляет математически.

Синтаксис: **ROUND (Число; ТО)**

Где:

- Число — числовое значение или столбец с числовыми значениями
- ТО — число, задающее точность округления. Если ТО положительное, то оно математически округляет дробную часть, если отрицательное, то округляет правую часть целого числа относительно запятой. Если равно 0, то возвращает математически округленное целое число, без дроби

Рассмотрим примеры формул с участием DAX функции ROUND на основе все той же исходной таблицы в Power BI:

ROUND (2) = ROUND ([Числа]; 2)

ROUND (1) = ROUND ([Числа]; 1)

ROUND (0) = ROUND ([Числа]; 0)

ROUND (-1) = ROUND ([Числа]; -1)

ROUND (-2) = ROUND ([Числа]; -2)

ROUND (-3) = ROUND ([Числа]; -3)

ROUND (2) = ROUND ([Числа]; 2)						
Числа	ROUND (2)	ROUND (1)	ROUND (0)	ROUND (-1)	ROUND (-2)	ROUND (-3)
233,786	233,79	233,8	234	230	200	0
212,375	212,38	212,4	212	210	200	0
-212,375	-212,38	-212,4	-212	-210	-200	0
-233,786	-233,79	-233,8	-234	-230	-200	0

## DAX функция ROUNDDOWN

ROUNDDOWN () — производит округление числа вниз до указанной точности округления.

ROUNDDOWN похожа на функцию ROUND, но ROUND — производит математическое округление (и вверх, и вниз), а ROUNDDOWN, исходя из названия — округляет всегда вниз.

Также, ROUNDDOWN похожа на функцию INT, так как INT, также, округляет число вниз, но всегда только до целого числа, в то время как, ROUNDDOWN может округлять не только до целого числа, но и до дробного, управляя точностью округления.



Синтаксис: **ROUNDDOWN (Число; ТО)**

Где:

- Число — числовое значение или столбец с числовыми значениями
- ТО — число, задающее точность округления. Если ТО положительное, то оно округляет дробную часть, если отрицательное, то округляет правую часть целого числа относительно запятой, если равно 0, то возвращает округленное целое число, без дроби

Рассмотрим примеры формул с участием DAX функции ROUNDDOWN на основе все той же исходной таблицы в Power BI:

**ROUNDDOWN (1) = ROUNDDOWN ([Числа]; 1)**  
**ROUNDDOWN (0) = ROUNDDOWN ([Числа]; 0)**  
**ROUNDDOWN (-1) = ROUNDDOWN ([Числа]; -1)**  
**ROUNDDOWN (-2) = ROUNDDOWN ([Числа]; -2)**  
**ROUNDDOWN (-3) = ROUNDDOWN ([Числа]; -3)**

 		ROUNDDOWN (-3) = <a href="#">ROUNDDOWN</a> ([Числа]; -3)				
Числа	ROUNDDOWN (1)	ROUNDDOWN (0)	ROUNDDOWN (-1)	ROUNDDOWN (-2)	ROUNDDOWN (-3)	
233,786	233,7	233	230	200	0	
212,375	212,3	212	210	200	0	
-233,786	-233,7	-233	-230	-200	0	
-212,375	-212,3	-212	-210	-200	0	

# DAX функция ROUNDUP

ROUNDUP () — производит округление числа вверх до указанной точности округления.

ROUNDUP похожа на функцию ROUNDDOWN, но ROUNDDOWN — производит округление только вниз, а ROUNDUP, исходя из названия — округляет всегда вверх.

Синтаксис: **ROUNDUP** (Число; ТО)

Где:

- Число — числовое значение или столбец с числовыми значениями
- ТО — число, задающее точность округления. Если ТО положительное, то оно округляет дробную часть, если отрицательное, то округляет правую часть целого числа относительно запятой, если равно 0, то возвращает округленное целое число, без дроби

Рассмотрим примеры формул с участием DAX функции ROUNDUP на основе все той же исходной таблицы в Power BI:

ROUNDUP (2) = ROUNDUP ([Числа]; 2)

ROUNDUP (1) = ROUNDUP ([Числа]; 1)

ROUNDUP (0) = ROUNDUP ([Числа]; 0)

ROUNDUP (-1) = ROUNDUP ([Числа]; -1)

ROUNDUP (-2) = ROUNDUP ([Числа]; -2)

ROUNDUP (-3) = ROUNDUP ([Числа]; -3)

ROUNDUP (-3) = ROUNDUP ([Числа]; -3)						
Числа	ROUNDUP (2)	ROUNDUP (1)	ROUNDUP (0)	ROUNDUP (-1)	ROUNDUP (-2)	ROUNDUP (-3)
233,786	233,79	233,8	234	240	300	1000
212,375	212,38	212,4	213	220	300	1000
-233,786	-233,79	-233,8	-234	-240	-300	-1000
-212,375	-212,38	-212,4	-213	-220	-300	-1000



# DAX функция MROUND

MROUND () — выполняет округление числа, кратно заданному значению точности.

Синтаксис:

MROUND (Число; Кратное)

Где:

- Число — числовое значение или столбец с числовыми значениями
- Кратное — число, задающее значение, кратно которому будет округляться исходное число

! — Исходное число и кратное число, входящие в параметры функции MROUND должны быть одного знака — или оба положительные, или оба отрицательные, иначе возвратится ошибка.

Рассмотрим примеры формул с участием DAX функции MROUND на основе все той же исходной таблицы в Power BI, но отрицательные и положительные значения которой, мы разделим на две отдельные таблицы, так как исходные числа и кратные числа в параметрах MROUND должны быть с одним знаком:

MROUND (5) = MROUND ([Числа]; 5)

MROUND (2) = MROUND ([Числа]; 2)

MROUND (0,5) = MROUND ([Числа]; 0,5)

MROUND (0,2) = MROUND ([Числа]; 0,2)

MROUND (0) = MROUND ([Числа]; 0)

MROUND (0) = MROUND ([Числа]; 0)					
Числа	MROUND (5)	MROUND (2)	MROUND (0,5)	MROUND (0,2)	MROUND (0)
233,786	235	234	234	233,8	0
212,375	210	212	212,5	212,4	0


MROUND (-5) = MROUND ([Числа]; -5)

MROUND (-2) = MROUND ([Числа]; -2)

MROUND (-0,5) = MROUND ([Числа]; -0,5)

MROUND (-0,2) = MROUND ([Числа]; -0,2)

MROUND (0) = MROUND ([Числа]; 0)

		MROUND (-5) = MROUND ([Числа]; -5)			
Числа	MROUND (-5)	MROUND (-2)	MROUND (-0,5)	MROUND (-0,2)	MROUND (0)
-233,786	-235	-234	-234	-233,8	0
-212,375	-210	-212	-212,5	-212,4	0

Как мы можем наблюдать из примеров в Power BI, действительно, округление происходит кратно значению, указанному во втором параметре MROUND (например, -233,786 округляется кратно -5, как итог = -235)

# DAX функция FLOOR

FLOOR () — выполняет округление числа в меньшую к нулю сторону, кратно заданному значению точности.

Синтаксис: **FLOOR** (Число; Кратное)

Где:

- Число — числовое значение или столбец с числовыми значениями
- Кратное — число, задающее значение, кратно которому будет округляться исходное число

! — Если исходное число положительное, то кратное число, входящее в параметры функции FLOOR, также, должно быть положительным. Если исходное число отрицательное, то кратное число может быть как отрицательным, так и положительным.

! — Кратное число не может равняться 0

Рассмотрим примеры формул с участием DAX функции FLOOR на основе все той же исходной таблицы в Power BI, но, как и в случае с примером функции MROUND, отрицательные и положительные значения исходной таблицы мы разделим на две отдельные таблицы:

**FLOOR (2) = FLOOR ([Числа]; 2)**

**FLOOR (1) = FLOOR ([Числа]; 1)**

**FLOOR (0,1) = FLOOR ([Числа]; 0,1)**

**FLOOR (0,01) = FLOOR ([Числа]; 0,01)**

FLOOR (0,01) = FLOOR ([Числа]; 0,01)				
Числа	FLOOR (2)	FLOOR (1)	FLOOR (0,1)	FLOOR (0,01)
233,786	232	233	233,7	233,78
212,375	212	212	212,3	212,37

И примеры с отрицательными исходными значениями:

FLOOR (-0,1) = FLOOR ([Числа]; -0,1)

FLOOR (-1) = FLOOR ([Числа]; -1)

FLOOR (-2) = FLOOR ([Числа]; -2)

FLOOR (-0,1) = FLOOR ([Числа]; -0,1)			
Числа	FLOOR (-0,1)	FLOOR (-1)	FLOOR (-2)
-233,786	-233,7	-233	-232
-212,375	-212,3	-212	-212

## DAX функция CEILING

CEILING () — выполняет округление числа в большую сторону до ближайшего целого, кратно значению из второго параметра.

Синтаксис:

CEILING (Число; Кратное)

Где:

- Число — числовое значение или столбец с числовыми значениями
- Кратное — число, задающее значение, кратно которому будет округляться исходное число

! — Если исходное число положительное, то кратное число, входящее в параметры функции CEILING, также, должно быть положительным. Если исходное число отрицательное, то кратное число может быть как отрицательным, так и положительным.

! — Если исходное число отрицательное и кратное число, также, отрицательное, то округление происходит в меньшую сторону.

Рассмотрим примеры формул с участием DAX функции CEILING на основе все той же исходной таблицы в Power BI, но, отрицательные и положительные значения исходной таблицы мы разделим на две отдельные таблицы:

CEILING (0) = CEILING ([Числа]; 0)

CEILING (1) = CEILING ([Числа]; 1)

CEILING (2) = CEILING ([Числа]; 2)

CEILING (2) = CEILING ([Числа]; 2)			
Числа	CEILING (0)	CEILING (1)	CEILING (2)
233,786	0	234	234
212,375	0	213	214

CEILING (2) = CEILING ([Числа]; 2)

CEILING (1) = CEILING ([Числа]; 1)

CEILING (0) = CEILING ([Числа]; 0)

CEILING (-1) = CEILING ([Числа]; -1)

CEILING (-2) = CEILING ([Числа]; -2)

CEILING (-2) = CEILING ([Числа]; -2)					
Числа	CEILING (2)	CEILING (1)	CEILING (0)	CEILING (-1)	CEILING (-2)
-233,786	-232	-233	0	-234	-234
-212,375	-212	-212	0	-213	-214

Как мы можем наблюдать из примера, действительно, в обычной ситуации CEILING округляет число в большую сторону до ближайшего целого кратно значимости второго параметра. Но, когда исходное значение отрицательное и кратное тоже отрицательное, округление происходит в обратную (меньшую) сторону.

# DAX функция ISO.CEILING

ISO.CEILING () — выполняет округление числа в большую сторону до ближайшего целого, кратно значению из второго параметра.

Синтаксис:

ISO.CEILING (Число; Кратное)

Где:

- Число — числовое значение или столбец с числовыми значениями
- Кратное — число, задающее значение, кратно которому будет округляться исходное число

ISO.CEILING — по всем параметрам идентична функции CEILING, за исключением того фактора, когда исходное число отрицательное и кратное число, также, отрицательное — ISO.CEILING, в отличие от CEILING, округление производит в большую сторону.

Рассмотрим примеры формул с участием DAX функции ISO.CEILING на основе все той же исходной таблицы в Power BI:

ISO.CEILING (2) = ISO.CEILING ([Числа]; 2)

ISO.CEILING (1) = ISO.CEILING ([Числа]; 1)

ISO.CEILING (0) = ISO.CEILING ([Числа]; 0)

ISO.CEILING (-1) = ISO.CEILING ([Числа]; -1)

ISO.CEILING (-2) = ISO.CEILING ([Числа]; -2)

ISO.CEILING (-2) = ISO.CEILING ([Числа]; -2)					
Числа	ISO.CEILING (2)	ISO.CEILING (1)	ISO.CEILING (0)	ISO.CEILING (-1)	ISO.CEILING (-2)
233,786	234	234	0	234	234
212,375	214	213	0	213	214
-212,375	-212	-212	0	-212	-212
-233,786	-232	-233	0	-233	-232

# DAX функции RAND И RANDBETWEEN

RAND () — возвращает случайное число от 0 до 1 или равное 0.

RANDBETWEEN () — возвращает случайное число между двумя числами, прописанными в параметрах функции.

Синтаксис:

RAND ()

RANDBETWEEN (Число от; Число до)

Где, числа «от» и «до» — это числа, между которыми возвратится случайное число в Power BI.

Примеры формул случайных чисел на основе DAX функций RAND и RANDBETWEEN:

Мера 1 = RAND ()

Мера 2 = RANDBETWEEN (10; 30)

В итоге, RAND и RANDBETWEEN возвратили случайные числа 0.31 и 16:





## DAX функция RADIANS

RADIANS () — преобразует градусы в радианы.

Синтаксис:

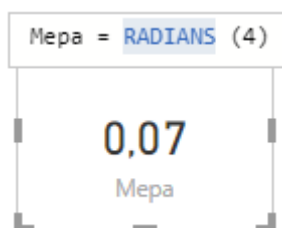
**RADIANS (Значение угла)**

Где, значение угла — угол в градусах, который необходимо преобразовать в радианы.

Пример формулы на основе DAX функции RADIANS:

**Мера = RADIANS (4)**

Результатом выполнения этой формулы с участием функции RADIANS будет значение, равное 0.07:



# 4. СТАТИСТИЧЕСКИЕ ФУНКЦИИ (АГРЕГИРОВАНИЕ)

## DAX функция SUM

SUM () — производит сумму всех чисел в столбце. Если строка в столбце содержит значение с иным типом данных, нежели числовой, то для работы SUM возвращается пустое значение.

Синтаксис: SUM ([Столбец])

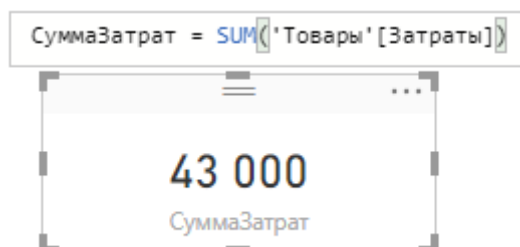
Пример [DAX формулы](#) с использованием функции SUM: в [Power BI](#) имеется исходная таблица заявок по товарам

Товар	Затраты	Продажи
Товар1	15000	50000
Товар2	18000	40000
Товар1	10000	35000

Напишем формулу расчета суммы затрат по всем товарам:

Сумма Затрат = SUM ('Товары'[Затраты])

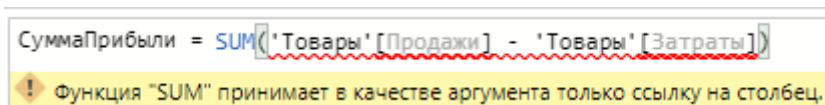
Как итог, SUM выдаст сумму всего столбца [Затраты], равную 43000:



Теперь попробуем при помощи SUM вычислить сумму прибыли, то есть, нам нужно из продаж вычесть затраты и затем сложить всю прибыль. Попробуем написать пример формулы:

Сумма Прибыли = SUM ('Товары'[Продажи] - 'Товары'[Затраты])

Но, Power BI у нас заругался и выдал ошибку:



Все правильно, ведь в качестве входящего параметра SUM принимает только 1 столбец. А мы в формуле выше мало того, что указали 2 столбца, так еще и написали выражение расчета разницы, что в SUM не допустимо вообще.

Для того, чтобы решить эту задачу нужно воспользоваться второй функцией суммирования в DAX — SUMX.

## DAX функция SUMX

SUMX () — вычисляет сумму результатов построчного выполнения выражения.

Синтаксис:

SUMX ('Таблица'; Выражение)

Где:

- 'Таблица' — исходная таблица или табличное выражение, по строкам которой будет вычисляться выражение из второго параметра функции
- Выражение — любое выражение, которое необходимо выполнить по строкам таблицы, входящей в первый параметр функции

SUMX работает в 2 действия. Для начала, вычисляется выражение из второго параметра функции для каждой строки таблицы, входящей в первый параметр. По мере построчного вычисления, результаты временно запоминаются в некой виртуальной таблице, а затем SUMX вторым действием сложит все результаты из этой виртуальной таблицы и выдаст нам итоговую сумму. После этого, временная виртуальная таблица будет стерта из памяти.

Давайте рассмотрим работу SUMX на практике, доработав формулу с ошибкой из примера выше:

Сумма Прибыли =

SUMX(

'Товары';

'Товары'[Продажи] - 'Товары'[Затраты]

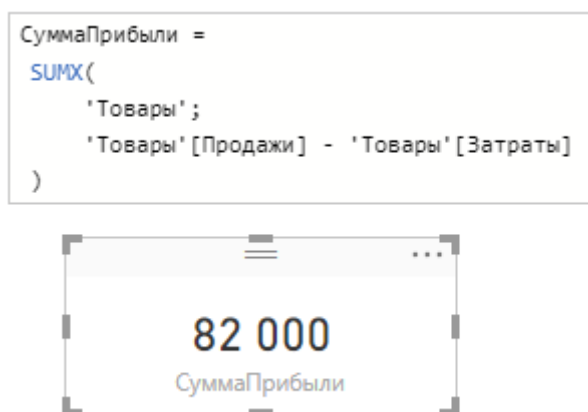
)

В первом параметре функции SUMX мы указали исходную таблицу, по строкам которой будет вычисляться выражение из второго параметра. А во втором параметре, собственно, мы расписали расчет прибыли.

Для начала, SUMX произведет расчет прибыли индивидуально по каждой строке таблицы 'Товары' и запишет результаты во временную виртуальную таблицу. Табличная модель этого процесса будет выглядеть так:

Товар	Затраты	Продажи		ВиртуальнаяТаблица
Товар1	15000	50000	→ 50000 - 15000 = 35000 →	35000
Товар2	18000	40000	→ 40000 - 18000 = 22000 →	22000
Товар1	10000	35000	→ 35000 - 10000 = 25000 →	25000

Затем, вторым действием, SUMX сложит все значения из этой временной виртуальной таблицы и выдаст итоговый результат суммы прибыли, равный 82000. После вычисления суммы виртуальная таблица удалится из памяти.



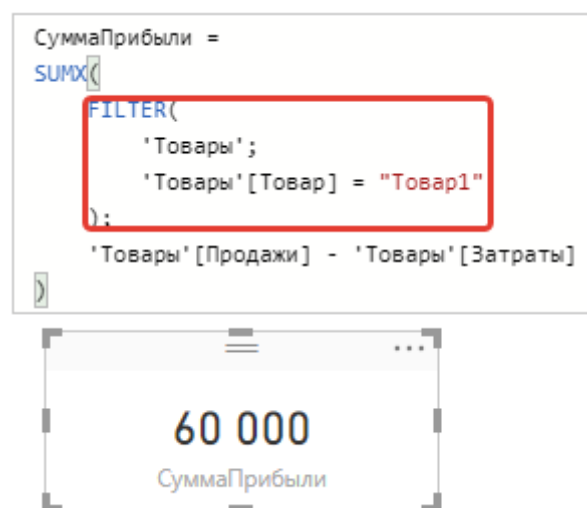
Естественно, что функции SUMX или SUM чаще всего используются в формулах в составе с другими функциями DAX. Например, SUMX можно встроить в параметр какой-то [другой функции](#) (к примеру, в [CALCULATE](#)) или же, наоборот, в параметры самой SUMX может быть встроена какая-то функция.

Для примера, рассмотрим взаимодействие SUMX и [FILTER](#). Так как первым параметром SUMX является таблица, то ее легко можно отфильтровать при помощи FILTER.

Если рассмотреть продолжение примера выше, то тогда можно составить формулу, где будет высчитываться прибыль только по товару 1:

```
Сумма Прибыли =  
SUMX(  
    FILTER(  
        'Товары';  
        'Товары'[Товар] = "Товар1"  
    );  
    'Товары'[Продажи] - 'Товары'[Затраты]  
)
```

И результатом взаимодействия SUMX и FILTER становится значение прибыли по 1 товару, равное 60000:



## DAX функция MAX

DAX функция MAX () — возвращает максимум среди числовых значений. MAX работает в двух режимах и эту особенность необходимо всегда учитывать в ее работе. Режимы работы зависят от вида синтаксиса.

Синтаксис:

- MAX ([Столбец]) — возвращает максимум среди значений столбца (особенность — не учитывает контекст строки)
- MAX (Выражение 1; Выражение 2) — возвращает максимум среди значений из двух параметров функции. Если работа производится в таблице, то MAX работает построчно, вычисляя максимум в рамках только одной строки (особенность — учитывает контекст строки)

Пример работы формул в [Power BI](#) на основе функции MAX: имеется исходная таблица с числовыми значениями в двух столбцах

Столбец1	Столбец2
5	6
2	8
10	5

Добавим в [Power BI Desktop](#) во вкладке «Моделирование» третий вычисляемый столбец на основе следующей [DAX](#) формулы с использованием MAX:

Столбец3 = MAX([Столбец1])

Исходя из того, что MAX в формуле прописана по синтаксису первого типа, то максимум вычисляется среди значений всего столбца, в данном примере по [Столбец1], и это значение равно 10. Эта формула



будет вычисляться без учета контекста строки и именно поэтому, в нашем вычисляемом столбце во всех ячейках результат выполнения функции MAX будет одним и тем же:

✕ ✓ Столбец3 = MAX([Столбец1])		
Столбец1	Столбец2	Столбец3
5	6	10
2	8	10
10	5	10

Если же мы изменим формулу и напомним ее согласно второму типу синтаксиса MAX:

Столбец3 = MAX([Столбец1]; [Столбец2])

то результат вычисления третьего столбца уже будет совсем другой, а именно, MAX будет вычислять максимум среди двух столбцов в рамках только одной строки. Поэтому результат будет таким:

✕ ✓ Столбец3 = MAX([Столбец1]; [Столбец2])		
Столбец1	Столбец2	Столбец3
5	6	6
2	8	8
10	5	10

## DAX функция MAXA

MAXA () — вычисляет максимум с обработкой пустых ячеек и ячеек, содержащих логический тип данных. Причем TRUE считается как 1, а FALSE — как 0.

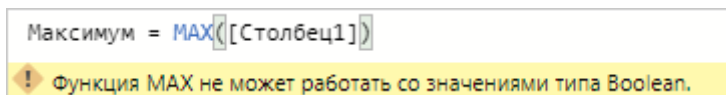
Синтаксис:

MAXA ([Столбец])

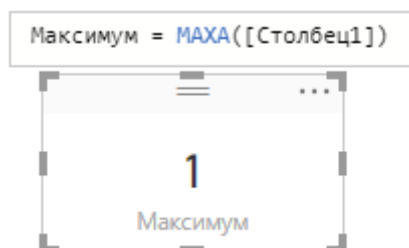
Пример: имеется исходная таблица со столбцом, который содержит логические значения

Столбец1
True
False

Если мы попытаемся рассчитать максимум по этому столбцу с использованием функции MAX, то выйдет ошибка, так как MAX не работает с логическими значениями



Но, если мы исправим формулу и пропишем там MAXA, которая учитывает логические значения, то максимум рассчитается, и он будет равен 1, так как MAXA считает TRUE, равным 1, а FALSE — равным 0:



## DAX функция MAXX

MAXX () — выводит максимальное значение из результатов построчного вычисления выражения.

Синтаксис:

MAXX ('Таблица'; Выражение)

Где:

- 'Таблица' — исходная таблица или табличное выражение, по строкам которой будет вычисляться выражение из второго параметра функции
- Выражение — любое выражение, которое необходимо выполнить по строкам таблицы, входящей в первый параметр функции

Первым действием MAXX вычислит выражение из второго параметра по каждой строке таблицы из первого параметра, а затем вычислит максимальное значение из получившихся результатов первого действия.

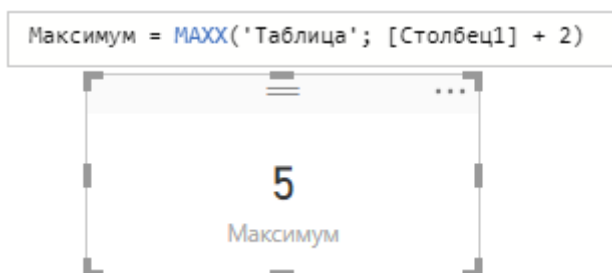
Пример: имеется исходная таблица, состоящая из одного столбца с числовыми данными

Столбец1
1
2
3

Запишем следующую формулу:

Максимум = MAXX ('Таблица'; [Столбец1] + 2)

Для начала MAXX вычислит выражение «[Столбец1] + 2» по каждой строке таблицы, то есть к значению каждой строки [Столбец1] прибавит 2, и только после этого, на основе получившихся результатов, вычислит максимум, который будет равен 5:



## DAX функции MIN, MINA, MINX

DAX функции MIN (), MINA () и MINX () работают полностью аналогично соответствующим функциям максимумов, только вместо самых больших значений вычисляют самые маленькие значения. Поэтому, прежде чем знакомится с функциями группы MIN, рекомендую просмотреть материал выше с разбором и примерами работы функций группы MAX.

1. MIN () — вычисляет минимум среди числовых значений. Также, как и функция MAX, MIN работает в двух режимах, зависящих от вида синтаксиса.

Синтаксис:

- MIN ([Столбец]) — возвращает минимум среди значений столбца (не учитывает контекст строки)
- MIN (Выражение 1; Выражение 2) — возвращает минимум среди значений из двух параметров функции. Если работа производится в таблице, то MIN работает построчно, вычисляя минимум в рамках только одной строки (учитывает контекст строки)

2. MINA () — возвращает минимум с обработкой пустых ячеек и ячеек, содержащих логический тип данных. TRUE считается как 1, а FALSE — как 0.

Синтаксис: MINA ([Столбец])

3. MINX () — выводит минимальное значение из результатов построчного вычисления выражения.

Синтаксис: MINX ('Таблица'; Выражение)

Вычисления производит в два действия — сначала вычисляет выражение из второго параметра по каждой строке таблицы из

первого параметра, а потом возвращает минимум по получившимся результатам первого действия.

Примеры работы [DAX формул](#) на основе функций группы MIN мы разбирать в Power BI или Excel (Powerpivot) не будем, так как они полностью идентичны работе примеров, которые мы разбирали выше по группе функций MAX, только вместо максимумов в данном случае будут вычисляться минимумы.

# DAX функции AVERAGE, AVERAGEA, AVERAGEX

1. DAX функция AVERAGE () — высчитывает среднее арифметическое значение числовых данных столбца (сумма значений деленное на количество этих значений):

- если в столбце текстовый тип данных, то AVERAGE возвращает пустое значение
- если в столбце пустые ячейки или логический тип данных, то в расчет среднего они не берутся
- если в столбце находятся ячейки с 0, то такие ячейки учитываются, как обычные

Синтаксис: AVERAGE ([Столбец])

2. DAX функция AVERAGEA () — высчитывает среднее арифметическое значение числовых данных столбца с учетом текстовых и логических типов данных:

- логические выражения равные TRUE (Правда) обрабатываются как 1
- логические выражения равные FALSE (Ложь) обрабатываются как 0
- текстовый тип данных обрабатывается как 0

Синтаксис: AVERAGEA ([Столбец])

3. DAX функция AVERAGEX () — вычисляет среднее арифметическое среди результатов построчного выполнения выражения.

Синтаксис: AVERAGEX ('Таблица'; Выражение)

Где:

- 'Таблица' — исходная таблица или табличное выражение, по строкам которой будет вычисляться выражение из второго параметра функции
- Выражение — любое выражение, которое необходимо выполнить по строкам таблицы, входящей в первый параметр функции

Функция AVERAGEX работает в 2 этапа. На первом этапе вычисляется выражение из второго параметра для каждой строки таблицы, указанной в первом параметре. На втором этапе AVERAGEX высчитывает среднее значение по данным, получившимся на первом этапе работы.

## Пример работы формулы на основе AVERAGE

Рассмотрим практический пример работы [DAX формулы](#) с использованием AVERAGE в программе [Power BI Desktop](#).

Имеется исходная таблица, содержащая числовые значения:

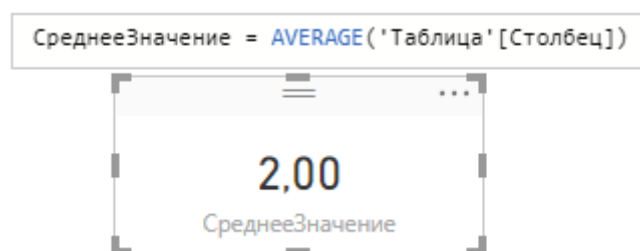
Столбец
1
2
3

Рассчитаем среднее значения столбца при помощи DAX функции AVERAGE:

Среднее Значение = AVERAGE('Таблица'[Столбец])

Результатом выполнения этой формулы будет значение 2:





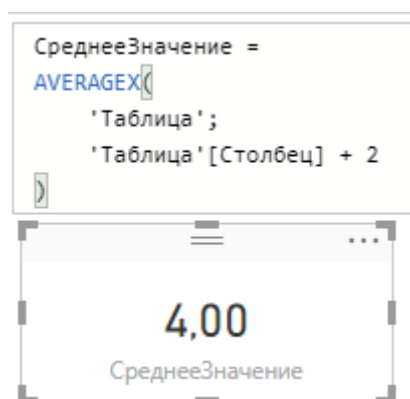
Немного усложним формулу нашего примера заменив AVERAGE функцией AVERAGEX:

```
Среднее Значение =  
AVERAGEX(  
    'Таблица';  
    'Таблица'[Столбец] + 2  
)
```

В этом случае данная DAX формула будет работать в 2 этапа. Для начала, AVERAGEX выполнит выражение «'Таблица'[Столбец] + 2» из второго параметра функции для каждой строки 'Таблица', указанной в первом параметре. Полученные результаты она запишет во внутреннюю память, в некую временную виртуальную таблицу:

ВиртуальнаяТаблица	
	3
	4
	5

И затем, уже на основе полученных результатов, записанных во временной памяти, AVERAGEX рассчитает среднее арифметическое значение, которое будет равно 4:



# DAX функции COUNT, COUNTA, COUNTX, COUNTAX

Итак, все эти функции COUNT, COUNTA, COUNTX и COUNTAX — отвечают за подсчет количества ячеек в Power BI и Power Pivot, но содержание ячеек в каждом из этих вариантов различается.

1. COUNT () — подсчитывает в столбце количество ячеек, которые содержат в себе числовое значение. В качестве числового значения признаются числа, даты и число, записанное в текстовом типе данных. Если в строке учитываемых значений нет, то функция выдаст 0. Если в таблице отсутствуют строки, то COUNT выдаст пустое значение.

Синтаксис: COUNT ([Столбец])

2. COUNTA () — подсчитывает непустые ячейки в столбце. То есть, количество тех ячеек, которые в себе содержат хоть какое-то значение: числа, даты, любой текст или значения логического типа.

Синтаксис: COUNTA ([Столбец])

3. COUNTX () — подсчитывает количество строк, содержащие в себе числовое значение, получившееся в результате построчного выполнения выражения. В качестве числового значения признаются числа, даты и число, записанное в текстовом типе данных.

Синтаксис: COUNTX ('Таблица'; Выражение)

Где:

- 'Таблица' — исходная таблица или табличное выражение, по строкам которой будет вычисляться выражение из второго параметра функции

- Выражение — любое выражение, которое необходимо выполнить по строкам таблицы, входящей в первый параметр функции

После вычисления выражения по каждой строке таблицы, на основе получившихся построчных результатов, COUNTX подсчитывает количество тех самых числовых значений.

- COUNTX () — работает точно также, как и функция выше COUNT, но в данном случае уже на основе получившихся построчных результатов, COUNTX подсчитывает непустые значения (числа, даты, текст, значения логического типа)

Синтаксис: COUNTX ('Таблица'; Выражение)

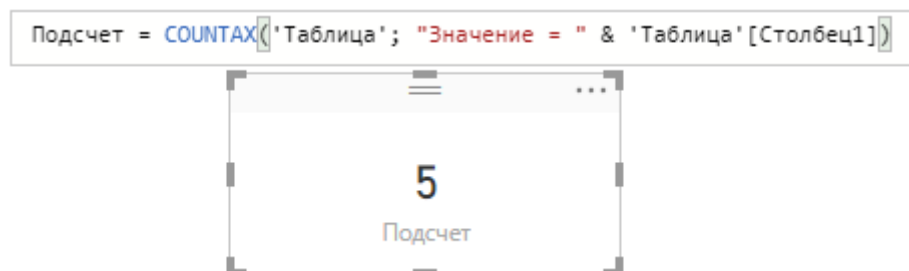
Пример формулы: в [Power BI](#) имеется простейшая таблица, состоящая из 1 столбца, в строках которого содержатся значения чисел, даты и текста:

Столбец1
1
2
3
14.08.2018
текст

Записав следующую [DAX формулу](#) с участием COUNTX ():

Подсчет = COUNTX('Таблица'; "Значение = " & 'Таблица'[Столбец1])

В качестве результата получим количество = 5



Почему? Потому что, COUNTX сначала выполнит выражение из второго параметра по каждой строке таблицы, входящей в первый параметр функции. Будет подсчитано количество непустых значений, полученных в результате вычисления выражения.

параметр и создаст некую виртуальную таблицу с получившимися результатами:

Виртуальный Стобец
Значение = 1
Значение = 2
Значение = 3
Значение = 14.08.2018
Значение = текст

В получившихся результатах везде будут текстовые значения, потому что DAX при работе оператора сложения текста & — все другие типы данных автоматически преобразует в текстовые.

А так как COUNTAX подсчитывает ячейки с любыми типами значений, в том числе и с текстовым типом, то в итоге возвращает количество, равное 5.

# DAX функция COUNTBLANK

COUNTBLANK () — подсчитывает количество пустых ячеек в столбце.

Синтаксис:

COUNTBLANK ([Столбец])

Пример: имеется таблица с одним столбцом, содержащая не только разные типы данных (числа, текст, даты), но и пустые ячейки:

Столбец1
1
2
3
14.08.2018
текст

Если мы составим следующую формулу с участием COUNTBLANK:

Подсчет = COUNTBLANK('Таблица'[Столбец1])

То COUNTBLANK вернет ответ: количество пустых ячеек = 2

Подсчет = COUNTBLANK('Таблица'[Столбец1])
2
Подсчет

# DAX функция DISTINCTCOUNT

Данную функцию почему-то очень часто называют неправильно, в два слова DISTINCT COUNT, хотя правильно писать в одно единое слово DISTINCTCOUNT.

Итак, DISTINCTCOUNT () — подсчитывает количество уникальных значений ячеек в столбце

Синтаксис: **DISTINCTCOUNT** ([Столбец])

Пример: имеется таблица, где в одном из столбцов перечисляются менеджеры:

Менеджеры
Петров
Сидоров
Петров
Воснецова

Если мы подсчитаем количество уникальных фамилий менеджеров при помощи следующей формулы:

Подсчет = **DISTINCTCOUNT** ('Таблица'[Менеджеры])

То ответ будет таким: количество уникальных фамилий менеджеров = 3

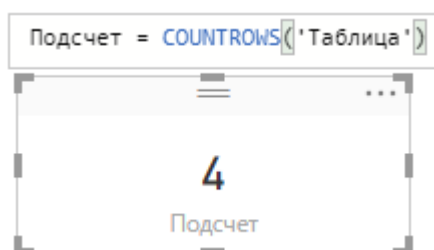
Подсчет = <b>DISTINCTCOUNT</b> ('Таблица'[Менеджеры])
3
Подсчет

# DAX функция COUNTROWS

COUNTROWS () — часто используемая на практике функция, которая подсчитывает количество строк в таблице.

Синтаксис: **COUNTROWS** ('Таблица')

Пример: если мы подсчитаем количество строк в таблице «Менеджеры» из примера выше, то COUNTROWS выдаст ответ 4 строки.



## COUNT и другие функции (CALCULATE, FILTER, IF)

Функции группы COUNT (подсчет количества) сами по себе используются довольно редко, обычно они нужны для каких-либо промежуточных вычислений в партнерстве с другими функциями. Например, функции группы COUNT используются совместно с [CALCULATE](#), [FILTER](#), с условиями «если» [IF](#) и другими функциями DAX.

Ну, как пример, можно подсчитать количество строк в таблице после применения фильтра функции FILTER:

```
Подсчет =  
COUNTROWS(  
    FILTER(  
        'ОбщиеПродажи';  
        'ОбщиеПродажи'[Отдел] = "Первый отдел"  
    )  
)
```

Или после изменения фильтра при использовании CALCULATE:

```
Подсчет =  
CALCULATE(  
    COUNTROWS('ОбщиеПродажи');  
    'ОбщиеПродажи'[Отдел] = "Первый отдел"  
)
```



## 5. ФУНКЦИИ ДАТЫ И ВРЕМЕНИ В DAX

## DAX функция TODAY

TODAY () — сегодня. Возвращает текущую дату.

Данная функция очень простая и не содержит в себе ни одного параметра.

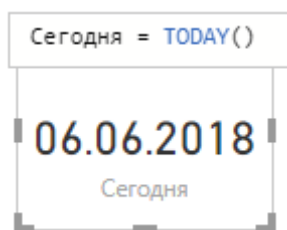
Синтаксис:

TODAY ()

Пример [формулы](#):

Сегодня = TODAY ()

Результатом выполнения этой формулы будет текущее сегодняшнее число:



## DAX функция NOW

NOW () — сейчас. Возвращает текущую дату и время.

NOW — это простейшая [функция](#), не содержащая параметров.

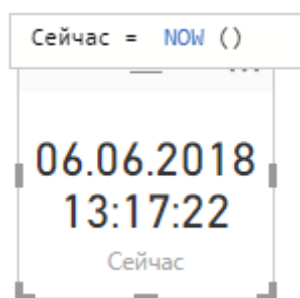
Синтаксис:

NOW ()

Пример [формулы](#):

Сейчас = NOW ()

Результатом выполнения этой простейшей формулы будет текущее время и сегодняшнее число:



## DAX функция DATE

DATE () — возвращает в формате datetime прописанную в параметрах дату.

Синтаксис: DATE (год; месяц; день)

! — Если месяц не соответствует числам 1-12, то лишние (недостающие) месяца добавляются (убавляются) к году. ! — Если дни не соответствуют числам 1-31 в месяце, то лишние (недостающие) дни добавляются (убавляются) к месяцу и к году.

Примеры формул с использованием функции DATE.

Пример 1: DATE (2018; 13; 23)

Результат 1: 23.01.2019

Пример 2: DATE (2018; -1; 23)

Результат 2: 23.11.2017

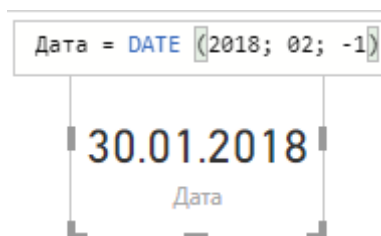
Пример 3: DATE (2018; 01; 32)

Результат 4: 01.02.2018

Пример 4: DATE (2018; 02; -1)

Результат 4: 30.01.2018

Пример выполнения формулы в Power BI на основе DAX функции DATE будет выглядеть так:



## DAX функция DATEVALUE

DATEVALUE () — используя локаль даты ПК, возвращает в формате datetime прописанную в параметрах текстовую дату.

Синтаксис:

DATEVALUE ("Дата")

«Дата» — может быть записана в различных вариантах, в том числе, и сокращенных.

Рассмотрим функцию DATEVALUE на примерах формул:

Пример 1: DATEVALUE ("2018-01-01")

Результат 1: 01.01.2018

Пример 2: DATEVALUE ("2018.01.01")

Результат 2: 01.01.2018

Пример 3: DATEVALUE ("2018/01/01")

Результат 3: 01.01.2018

Пример 4: DATEVALUE ("2018,01,01")

Результат 4: 01.01.2018

Пример 5: DATEVALUE ("2018 01 01")

Результат 5: 01.01.2018

Пример 6: DATEVALUE ("2018 12")

Результат 6: 01.12.2018

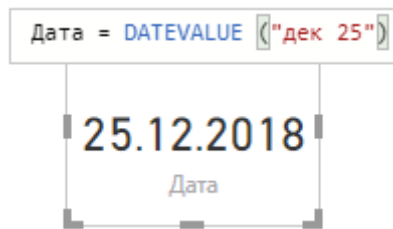
Пример 7: DATEVALUE ("декабрь 2018")

Результат 7: 01.12.2018

Пример 8: DATEVALUE ("23 дек")

Результат 8: 23.12.2018

В Power BI формула на основе DATEVALUE работать будет так:



# DAX функции YEAR, MONTH, DAY

YEAR (), MONTH () и DAY () — возвращают год, месяц и день в формате чисел из значения даты.

Синтаксис:

YEAR ([Дата])

MONTH ([Дата])

DAY ([Дата])

Где [Дата] — столбец, содержащий даты, либо дата в текстовом или datetime форматах.

Примеры формул на основе функций YEAR, MONTH и DAY.

Пример 1: YEAR (DATE (2018; 10; 18))

Результат 1: 2018

Пример 2: YEAR ("декабрь 2017")

Результат 2: 2017

Пример 3: MONTH (DATE (2018; 10; 18))

Результат 3: 10

Пример 4: MONTH ("18 декабря")

Результат 4: 12

Пример 5: DAY (DATE (2018; 10; 13))

Результат 5: 13

Пример 6: DAY ("17 декабря")

Результат 6: 17

В большинстве случаев эти функции в Power BI или PowerPivot будут использоваться со внутренним параметром [Дата], то есть, значение даты будет указано в столбце. Давайте рассмотрим соответствующий пример.

В [Power BI Desktop](#) имеется исходная таблица с датами:

Дата
1 декабря 2018 г.
5 января 2019 г.
10 февраля 2019 г.

Создадим в этой таблице 3 вычисляемых столбца по следующим формулам:

Год = YEAR ([Дата])

Месяц = MONTH ([Дата])

День = DAY ([Дата])

В результате, в Power BI получим следующую расширенную таблицу:

✕ ✓		Год = YEAR ([Дата])		
Дата	Год	Месяц	День	
1 декабря 2018 г.	2018	12	1	
5 января 2019 г.	2019	1	5	
10 февраля 2019 г.	2019	2	10	



## DAX функция TIME

TIME () — возвращает в формате datetime прописанное в параметрах время.

Синтаксис:

TIME (часы; минуты; секунды)

! — Если часы указаны числом, выходящим за пределы 23, то в качестве часа будет указан остаток от деления этого числа на 24 ! — Если минуты указаны числом, выходящим за пределы 59, то это значение будет пересчитано в часы и минуты ! — Если секунды указаны числом, выходящим за пределы 59, то это значение будет пересчитано в часы, минуты и секунды

Примеры формул с использованием функции TIME.

Пример 1: TIME (24; 0; 0)

Результат 1: 0:00:00

Пример 2: TIME (25; 0; 0)

Результат 2: 1:00:00

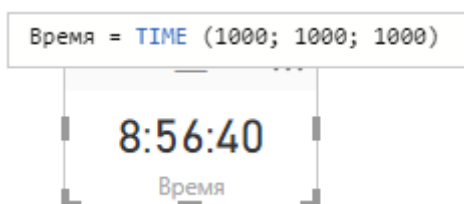
Пример 3: TIME (19; 70; 23)

Результат 3: 20:10:23

Пример 4: TIME (19; 10; 70)

Результат 4: 19:11:10

В [Power BI Desktop](#) формула на основе TIME работать будет так:



В данном примере TIME автоматически преобразовала указанные значения часов, минут и секунд (1000) в соответствующие правильные значения времени.

## DAX функция TIMEVALUE

TIMEVALUE () — используя локаль времени ПК, возвращает в формате datetime прописанное в параметрах текстовое время.

Синтаксис:

TIMEVALUE ("Время")

Рассмотрим DAX функцию TIMEVALUE на примерах формул:

Пример 1: TIMEVALUE ("12:10:10")

Результат 1: 12:10:10

Пример 2: TIMEVALUE ("12:00")

Результат 2: 12:00:00

# DAX функции HOUR, MINUTE И SECOND

HOUR (), MINUTE () и SECOND () — возвращают часы, минуты и секунды в формате чисел из значения времени.

Синтаксис:

HOUR ([Время])

MINUTE ([Время])

SECOND ([Время])

Где [Время] — столбец, содержащий время, либо время в текстовом или datetime форматах.

Примеры формул на основе функций HOUR, MINUTE и SECOND.

Пример 1: HOUR (TIME (19; 11; 23))

Результат 1: 19

Пример 2: HOUR ("22:00")

Результат 2: 22

Пример 3: MINUTE (TIME (19; 11; 23))

Результат 3: 11

Пример 4: MINUTE ("22:18")

Результат 4: 18

Пример 5: SECOND (TIME (19; 11; 23))

Результат 5: 23

Пример 6: SECOND ("22:18:16")

Результат 6: 16

Рассмотрим еще один пример, где в качестве параметра в этих DAX функциях содержится столбец со значениями времени.

В Power BI Desktop имеется исходная таблица со значениями времени:

Время
22:00:17
23:10:00
2:24:19

Создадим в исходной таблице столбцы [Часы], [Минуты] и [Секунды] по формулам:

Часы = HOUR ([Время])

Минуты = MINUTE ([Время])

Секунды = SECOND ([Время])

Как результат, в Power BI мы получим таблицу времени:

Часы = HOUR ([Время])				
Время	Часы	Минуты	Секунды	
22:00:17	22	0	17	
23:10:00	23	10	0	
2:24:19	2	24	19	

## DAX функция WEEKDAY

WEEKDAY () — возвращает день недели в формате чисел 1-7 (0-6). По умолчанию неделя начинается с воскресенья (1) и заканчивается субботой (7).

Синтаксис:

WEEKDAY (Дата; Начало Недели)

Где:

- Дата — дата в формате datetime. Дату необходимо вводить формулой DATE() или другим выражением, возвращающим формат datetime
- Начало Недели (по умолчанию — 1): 1 — начало недели в воскресенье (1) и конец в субботу (7) ; 2 — начало недели в понедельник (1) и конец в воскресенье (7); 3 — начало недели в понедельник (0) и конец в воскресенье (6)

Примеры формул на основе DAX функции WEEKDAY.

Пример 1: WEEKDAY (DATE (2018; 10; 13)) - начало недели в воскресенье (1)

Результат 1: 7 (суббота)

Пример 2: WEEKDAY (DATE (2018; 10; 13); 1) - начало недели в воскресенье (1)

Результат 2: 7 (суббота)

Пример 3: WEEKDAY (DATE (2018; 10; 13); 2) - начало недели в понедельник (1)

Результат 3: 6 (суббота)

Пример 4: WEEKDAY (DATE (2018; 10; 13); 3) - начало недели в понедельник (0)

Результат 4: 5 (суббота)

В качестве параметров даты в WEEKDAY можно вставлять столбец со значениями даты. Давайте рассмотрим такой пример формулы.

В Power BI имеется исходная таблица с датами, где 1 января 2018 — понедельник:

Дата
1 января 2018 г.
2 января 2018 г.
3 января 2018 г.

Добавим в эту таблицу 2 вычисляемых столбца на основе следующих формул с участием DAX функции WEEKDAY:

Номер Дня Недели 1 = WEEKDAY ([Дата]; 1)

Номер Дня Недели 2 = WEEKDAY ([Дата]; 2)

То есть, в первой формуле начало недели начинается в воскресенье (1), а во второй формуле — в понедельник (1).

В итоге результат будет таким:

НомерДняНедели 1 = WEEKDAY ([Дата]; 1)		
Дата	НомерДняНедели 1	НомерДняНедели 2
1 января 2018 г.	2	1
2 января 2018 г.	3	2
3 января 2018 г.	4	3

В столбце на основе первой формулы 1 января (понедельник) равен числу 2, так как начало недели в воскресенье (1). А в столбце на основе второй формулы 1 января (понедельник) равен числу 1, так как начало недели, также, в понедельник (1).

## DAX функция WEEKNUM

WEEKNUM () — возвращает номер недели года (относительно начала года).

Синтаксис:

WEEKNUM (Дата; Начало Недели)

Где:

- Дата — дата в формате datetime. Дату необходимо вводить формулой DATE() или другим выражением, возвращающим формат datetime
- Начало Недели: 1 — неделя начинается с воскресенья (по умолчанию); 2 — неделя начинается с понедельника

Примеры формул на основе DAX функции WEEKNUM.

Пример 1: WEEKNUM (DATE (2018; 10; 13))

Результат 1: 41

Пример 2: WEEKNUM (DATE (2018; 10; 13); 1)

Результат 2: 41

Пример 3: WEEKNUM (DATE (2018; 10; 13); 2)

Результат 3: 41

Как и в функции WEEKDAY, в качестве параметров даты в WEEKNUM, также можно вставить столбец со значениями даты. Давайте рассмотрим такой пример формулы.



В Power BI Desktop имеется исходная таблица с датами, где 1 апреля 2018 — это воскресенье:

Дата
1 апреля 2018 г.
2 апреля 2018 г.
3 апреля 2018 г.

Добавим в эту таблицу 2 вычисляемых столбца на основе следующих формул с участием [DAX функции](#) WEEKNUM:

Неделя Года 1 = WEEKNUM ([Дата]; 1)

Неделя Года 2 = WEEKNUM ([Дата]; 2)

Исходя из синтаксиса, в первой формуле начало недели начинается в воскресенье, а во второй формуле — в понедельник.

В итоге результат будет таким:

НеделяГода2 = WEEKNUM ([Дата]; 2)		
Дата	НеделяГода1	НеделяГода2
1 апреля 2018 г.	14	13
2 апреля 2018 г.	14	14
3 апреля 2018 г.	14	14

Так как в столбце, рассчитанном на основе первой формулы, начало недели в воскресенье, а в таблице у нас представлены 3 дня — воскресенье (1 апреля), понедельник (2 апреля), вторник (3 апреля), то у этих всех 3-х дней один номер недели = 14.

В столбце, рассчитанном на основе второй формулы, начало недели в понедельник. И именно поэтому, 1 апреля (воскресенье) имеет номер недели 13, а у 2 и 3 апреля, номер недели уже 14.

## DAX функция YEARFRAC

YEARFRAC () — вычисляет долю указанного периода дат в целом году.

Синтаксис:

YEARFRAC (Стартовая Дата; Конечная Дата; Базис)

Где, «Стартовая Дата» и «Конечная Дата» — даты в формате datetime, а «Базис» — способ вычисления дня (необязательный параметр).

Базис:

- параметр = 0 — американский стандарт (NASD), 30 дней / 60 дней
- параметр = 1 — фактический период / фактический период
- параметр = 2 — фактический период / 360 дней
- параметр = 3 — фактический период / 365 дней
- параметр = 4 — европейский стандарт, 30 дней / 360 дней

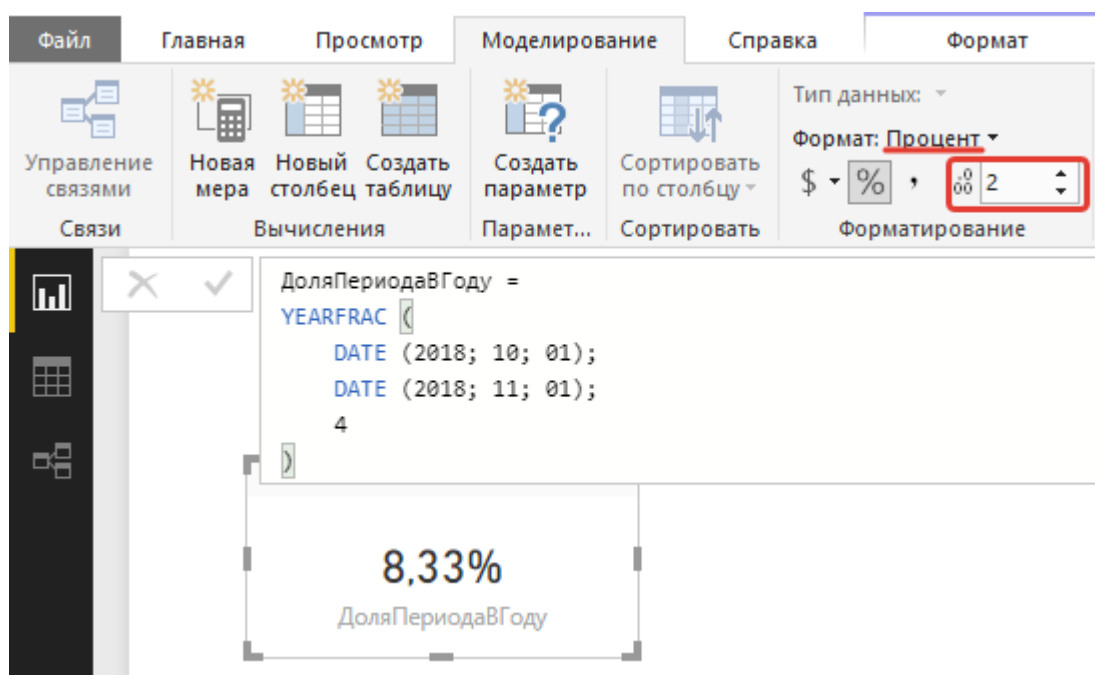
Рассмотрим пример [формулы](#) на основе DAX функции YEARFRAC.

Доля Периода В Году =

```
YEARFRAC (  
    DATE (2018; 10; 01);  
    DATE (2018; 11; 01);  
    4  
)
```

В данной формуле начальную и конечную даты мы задали при помощи DAX функции [DATE](#). Период между этими двумя датами составляет ровно 1 месяц. В качестве базиса (способа вычисления дня) я взял европейский стандарт (4).

Как результат, формула на основе YEARFRAC вывела долю этого периода относительно целого года = 8.33 %:

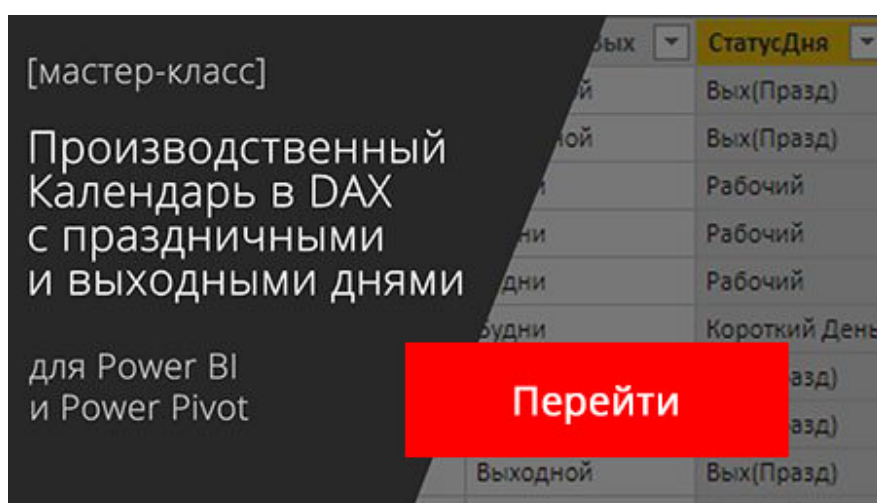


У созданной меры в [Power BI Desktop](#) нужно поменять формат на % и вывести два знака после запятой.

## 6. ФУНКЦИИ ЛОГИКИ ОПЕРАЦИЙ С ДАТОЙ В DAX (TIME INTELLIGENCE)

# DAX функция CALENDARAUTO

CALENDARAUTO () — автоматический календарь в [DAX](#). Возвращает таблицу с одним столбцом [Date], содержащим непрерывные даты от начала и до конца года. Рассчитывается исходя из тех дат, которые находятся во всех таблицах модели данных.



То есть, CALENDARAUTO в модели данных ищет самую старую (минимальную) и самую новую (максимальную) даты. И на основе года из этих дат, строит полный календарь от 1 января того года, который указан в самой старой (минимальной) дате и до 31 декабря того года, который указан в самой новой (максимальной) дате из всех таблиц модели данных.

! — Если в модели данных нет значений даты, то функция CALENDARAUTO возвратит ошибку.

! — Будьте внимательны, CALENDARAUTO для расчетов берет любые даты, например, даты дней рождения. Поэтому, если в Вашей модели данных будет человек с днем рождения в 1925 году, то функция создаст календарь, начиная с 1925 года и далее до самой последней даты в модели данных.

Синтаксис:

CALENDARAUTO ()

CALENDARAUTO (Номер Месяца)

Где, Номер Месяца — это номер месяца в году.

Если в CALENDARAUTO параметров нет, то возвращается календарь, кратный целому году с 1 января по 31 декабря.

Если в параметрах прописан номер месяца, то возвращается календарь, кратный целому году (12 месяцев), где первый месяц выведенного года равен следующему месяцу, после номера месяца, который прописан в параметре функции.

Разберем работу DAX функции CALENDARAUTO на основе нескольких примеров [формул](#).

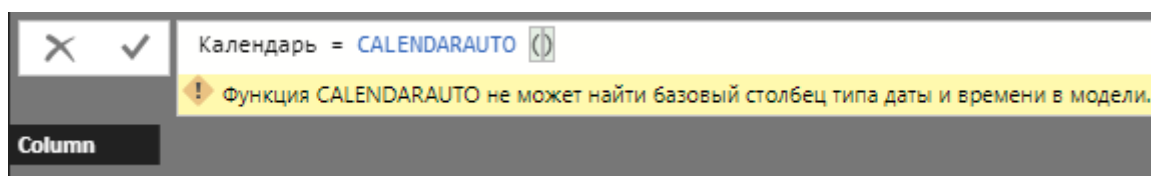
В модели данных Power BI имеется всего одна таблица «Заявки», содержащая 2 столбца [Менеджер] и [Сумма Продажи]:

Менеджер	СуммаПродажи
Петров	10000
Сидоров	15000

Во вкладке «Моделирование» в [Power BI Desktop](#) создадим вычисляемую таблицу «Календарь» на основе формулы с CALENDARAUTO:

Календарь = CALENDARAUTO ()



В данном случае, так как в модели данных значений дат нет вообще, то [DAX функция](#) CALENDARAUTO выдаст ошибку:



Давайте изменим исходную таблицу в модели данных и добавим туда столбец, содержащий даты обработок заявок:

Менеджер	СуммаПродажи	ДатаЗаявки
Петров	10000	25 декабря 2017 г.
Сидоров	15000	15 января 2018 г.



В этот раз функция CALENDARAUTO отработала как нужно и возвратила календарь дат с 1 января 2017 по 31 декабря 2018 года, так как год у минимальной даты 2017, а год у максимальной даты — 2018:

 	Календарь = CALENDARAUTO ()
Date	
01.01.2017	
02.01.2017	
03.01.2017	
...	
29.12.2018	
30.12.2018	
31.12.2018	

Теперь, изменим формулу выше и добавим в CALENDARAUTO параметр — номер месяца 4:

Календарь = CALENDARAUTO (4)

Результат в Power BI будет следующим:

		Календарь = CALENDARAUTO (4)
Date		
01.05.2017		
02.05.2017		
<hr/>		
28.04.2018		
29.04.2018		
30.04.2018		

Так как в этой формуле в CALENDARAUTO был указан номер месяца 4, то функция, также вывела полных 2 года, только не с 1 января по 31

декабря, а в самом начале с пропуском 4 месяцев, то есть, период в 2 года здесь перечислен с 1 мая 2017 по 30 апреля 2018 года.



## DAX функция CALENDAR

CALENDAR () — календарь, создаваемый вручную. Возвращает таблицу с одним столбцом [Date], содержащим непрерывные даты от стартовой до конечной дат, указанных в параметрах.

Синтаксис:

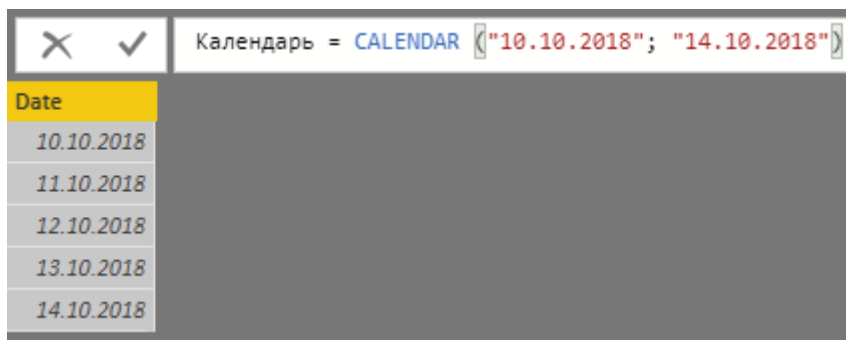
CALENDAR (Стартовая Дата; Конечная Дата)

Где, стартовые и конечные даты — это любые выражения, возвращающие значения в формате datetime (в том числе, ссылки на исходные столбцы с датами).

Пример формулы на основе DAX функции CALENDAR:

Календарь = CALENDAR ("10.10.2018"; "14.10.2018")

Результатом данной формулы с CALENDAR будет таблица небольшого календаря, состоящая из 5 дней:



Date
10.10.2018
11.10.2018
12.10.2018
13.10.2018
14.10.2018

## DAX функция DATESBETWEEN

DATESBETWEEN () — ограничитель календаря по произвольным датам. Создает ограниченную таблицу из дат, начиная со стартовой и заканчивая конечной датами, расположенными в исходном столбце.

Синтаксис:

DATESBETWEEN ([Дата]; Стартовая Дата; Конечная Дата)

Где:

- [Дата] — столбец дат в исходной таблице
- Стартовая и Конечная Даты — дата в формате datetime

! — Стартовые и конечные даты, указанные в параметрах функции, входят в состав выводимых дат.

Пример формулы на основе DAX функции DATESBETWEEN.

В Power BI имеется исходная таблица «Продажи», содержащая даты и сумму продаж за 2 года с 1.01.2017 по 31.12.2018. Но, продажи имеются не за каждый день. Обратите внимание, что 30 декабря 2018 года отсутствует, потому что продаж не было:

ДатаПродажи	СуммаПродажи
1 января 2017 г.	31083
2 января 2017 г.	32305
3 января 2017 г.	20425
28 декабря 2018 г.	62910
29 декабря 2018 г.	65906
31 декабря 2018 г.	48099

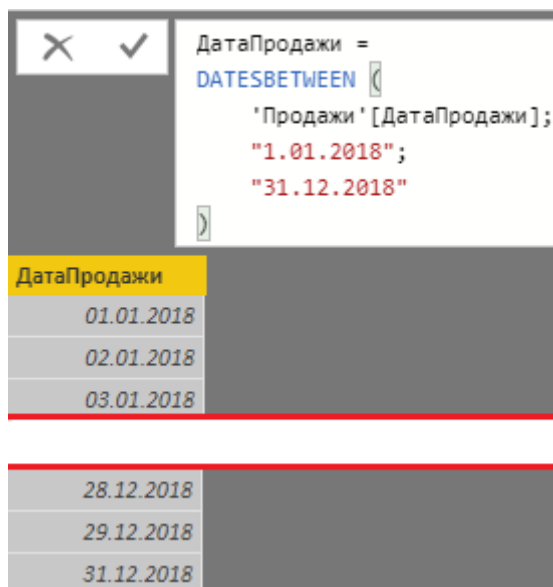
В определенных расчетах нам могут понадобиться даты этих продаж, но не все, а ограниченные, например, только за 2018 год. Ограничить даты можно многими способами, начиная от пользовательских фильтров в разделе «Отчеты» в [Power BI Desktop](#), заканчивая разнообразными формулами и [функциями](#) внутри самого языка DAX.

Так как в этой статье мы разбираем DATESBETWEEN, то давайте разберем пример ограничения при помощи этой функции, тем более что она, как раз, для этого и существует.

Создадим во вкладке «Моделирование» в Power BI Desktop вычисляемую таблицу по следующей [формуле](#) на основе DATESBETWEEN:

```
ДатаПродажи =  
DATESBETWEEN (  
    'Продажи'[ДатаПродажи];  
    "1.01.2018";  
    "31.12.2018"  
)
```

Этой формулой мы создали таблицу с датами продаж, ограниченными только 2018 годом:



ДатаПродажи
01.01.2018
02.01.2018
03.01.2018
...
28.12.2018
29.12.2018
31.12.2018

Обратите внимание, что DATESBETWEEN возвратила именно тот набор дат, который был в исходном столбце, то есть, 30 декабря 2018 года в этой таблице тоже нет.

## DAX функция DATESINPERIOD

DATESINPERIOD () — ограничитель календаря по типовым интервалам. Создает ограниченную таблицу из дат, начиная со стартовой даты, расположенной в исходном столбце и продолжая в течение указанного типового интервала (количеством дней, месяцев, кварталов, лет).

Синтаксис:

DATESINPERIOD ([Дата]; Стартовая Дата; Количество Интервалов; Интервал)

Где:

- [Дата] — столбец дат в исходной таблице
- Стартовая Дата — дата в формате datetime (указанная дата входит в состав выводимых дат)
- Количество Интервалов — целое число, характеризующее количество интервалов
- Интервал — наименование интервала: year (год), quarter (квартал), month (месяц), day (день)

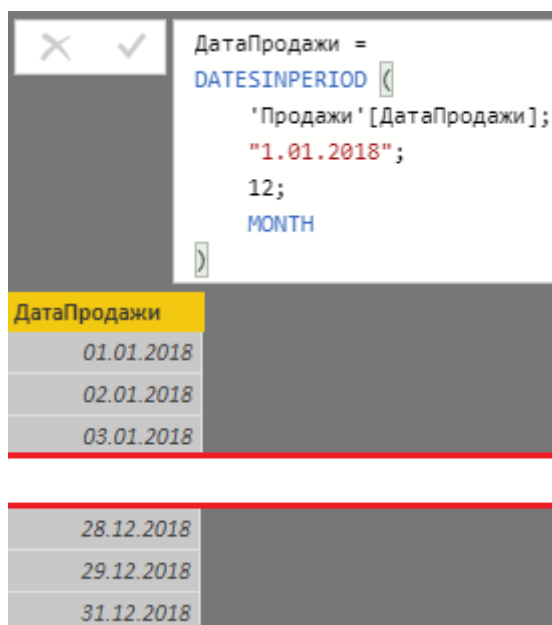
DATESINPERIOD очень похожа на выше рассматриваемую функцию в первом разделе этой статьи. Разница заключается лишь в том, что DATESINPERIOD ограничивает исходный набор дат именно определенными целыми интервалами.

Продолжая рассматривать практический пример, который мы разбирали с первой функцией выше, давайте напишем, формулу, ограничивающую даты продажи 2018 годом, но теперь, на основе DAX функции DATESINPERIOD:

```
ДатаПродажи =  
DATESINPERIOD (  
    'Продажи'[ДатаПродажи];  
    "1.01.2018";  
    12;  
    MONTH  
)
```

В первом параметре мы указали исходный столбец, во втором — стартовую дату, с которой нужно начать ограничение, в третьем — количество периодов, в нашем случае, 12. Ну и, в четвертом — наименование самого периода — я выбрал «MONTH», то есть, месяц.

Результатом выполнения этой формулы на основе DATESINPERIOD, получился столбец из дат, начиная с 1 января 2018 года и далее, продолжая на период в 12 месяцев, до 31 декабря 2018 года:



ДатаПродажи
01.01.2018
02.01.2018
03.01.2018
...
28.12.2018
29.12.2018
31.12.2018

Обращаю Ваше внимание, что и в данном случае, как это было и с первой рассматриваемой функцией, 30 декабря здесь также отсутствует, так как рассматриваемые в этой статье обе функции не выводят список из непрерывных дат, а выводят именно то, что имеется в исходном столбце.

Сами по себе DATESBETWEEN и DATESINPERIOD обычно просто так не используют. А вставляют их в другие формулы в сочетании с другими функциями.

Также, не нужно путать столбцы из дат, которые создают DATESBETWEEN либо DATESINPERIOD с календарем. Для создания календаря есть специализированные функции, ссылку на них я уже давал выше. А рассматриваемые функции в этой статье — являются именно ограничителем либо исходного столбца дат, либо большого непрерывного календаря.

## DAX функция DATEADD

DATEADD () — создает таблицу со столбцом из дат, сдвинутых назад (в прошлое) или вперед (в будущее) на заданное количество интервалов от даты текущего контекста.

Синтаксис:

DATEADD ([Дата]; Количество Интервалов; Интервал)

Где:

- [Дата] — столбец из дат или выражений, возвращающих даты
- Количество Интервалов — целое число, характеризующее количество интервалов, которое нужно добавить (вычесть) к дате в текущем контексте. Если указано положительное число — то интервалы добавляются, если указано отрицательное число — то интервалы вычитаются
- Интервал — тип интервала: year (год), quarter (квартал), month (месяц), day (день)

! — Для правильной работы функции необходимо в качестве ее первого параметра [Дата] использовать столбец из таблицы непрерывных дат в [Power BI](#), то есть, создавать отдельную связанную таблицу «Календарь» с непрерывным перечислением всех дат.

Разберем работу DAX функции DATEADD на примере нескольких [формул](#).

В [Power BI Desktop](#) имеется таблица с датами «Календарь»:

Дата
1 января 2018 г.
2 января 2018 г.
3 января 2018 г.
4 января 2018 г.
5 января 2018 г.

Для создания формул сравнения текущих показателей с прошлыми, необходимо внутри этих формул, относительно текущей даты, как-то получить соответствующую дату в прошлом. Для этого воспользуемся рассматриваемой функцией DATEADD.

И для примера того, как она сдвигает даты, создадим в таблице «Календарь» вычисляемый столбец на основе следующей формулы:

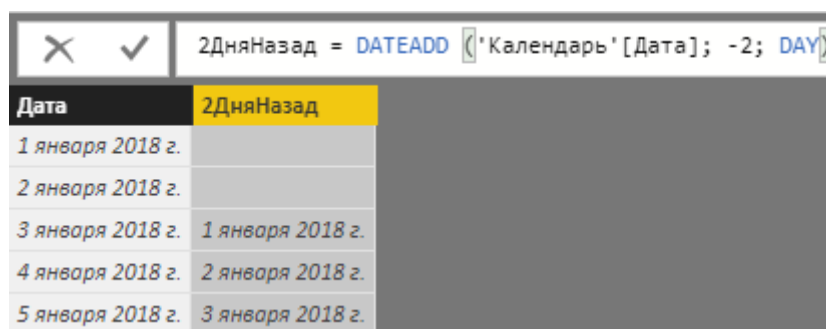
**2 Дня Назад = DATEADD ('Календарь'[Дата]; -2; DAY)**

В первом параметре этой формулы мы указали ссылку на столбец даты в календаре (календарь должен быть связан с той таблицей фактов, по которой Вы рассчитываете показатели).

Во втором параметре — количество интервалов, на которое нужно сдвинуть даты, причем значение отрицательное, то есть, сдвиг будет в прошлое.

В третьем — тип самого интервала (день).

Как результат выполнения этой формулы на основе DATEADD, получился столбец дат, сдвинутый в прошлое на два дня назад:



Дата	2ДняНазад
1 января 2018 г.	
2 января 2018 г.	
3 января 2018 г.	1 января 2018 г.
4 января 2018 г.	2 января 2018 г.
5 января 2018 г.	3 января 2018 г.

То есть, каждой текущей дате из столбца [Дата] соответствует своя дата из прошлого на 2 дня назад (для текущей даты 5 января соответствует дата из прошлого 3 января).

Если мы в формуле укажем в качестве количества интервалов положительное число:



1 День Вперед = DATEADD ('Календарь'[Дата]; 1; DAY)

то сдвиг произойдет в будущее:

✕ ✓ 1ДеньВперед = DATEADD ('Календарь'[Дата]; 1; DAY)	
Дата	1ДеньВперед
1 января 2018 г.	2 января 2018 г.
2 января 2018 г.	3 января 2018 г.
3 января 2018 г.	4 января 2018 г.
4 января 2018 г.	5 января 2018 г.
5 января 2018 г.	

То есть, каждой текущей дате из столбца [Дата] соответствует своя дата из будущего на 1 день вперед (для текущей даты 1 января соответствует дата из будущего 2 января).

## DAX функция PARALLELPERIOD

PARALLELPERIOD () — возвращает таблицу из дат, смещенных во времени вперед или назад параллельно текущей дате в текущем контексте на заданное количество интервалов.

Синтаксис:

PARALLELPERIOD ([Дата]; Количество Интервалов; Интервал)

Где:

- [Дата] — столбец из дат или выражений, возвращающих даты
- Количество Интервалов — целое число, характеризующее количество интервалов, которое нужно добавить (вычесть) к дате в текущем контексте. Если указано положительное число — то интервалы добавляются, если указано отрицательное число — то интервалы вычитаются
- Интервал — тип интервала: year (год), quarter (квартал), month (месяц)

! — Для правильной работы функции необходимо в качестве ее первого параметра [Дата] использовать столбец из таблицы непрерывных дат в Power BI, то есть, создавать отдельную связанную таблицу «Календарь» с непрерывным перечислением всех дат.

В общем и целом, функции DATEADD и PARALLELPERIOD практически одинаковы, за исключением того, что:

1. в PARALLELPERIOD нет интервала day (день)
2. PARALLELPERIOD возвращает параллельный период полностью (например, от начала до конца месяца), тогда как DATEADD возвращает конкретно только тот интервал, который задан в изначальном столбце дат.

Например, изначальный набор дат будет отфильтрован датами от 16 до 25 октября. И если мы обеими функциями попытаемся вернуть набор дат на 1 месяц назад, то DATEADD возвратит даты только с 16 до 25 сентября, а PARALLELPERIOD — уже возвратит даты всего предыдущего месяца от начала и до конца, то есть, с 1 по 30 сентября. Давайте разберем разницу работы DATEADD и PARALLELPERIOD на конкретном практическом примере.

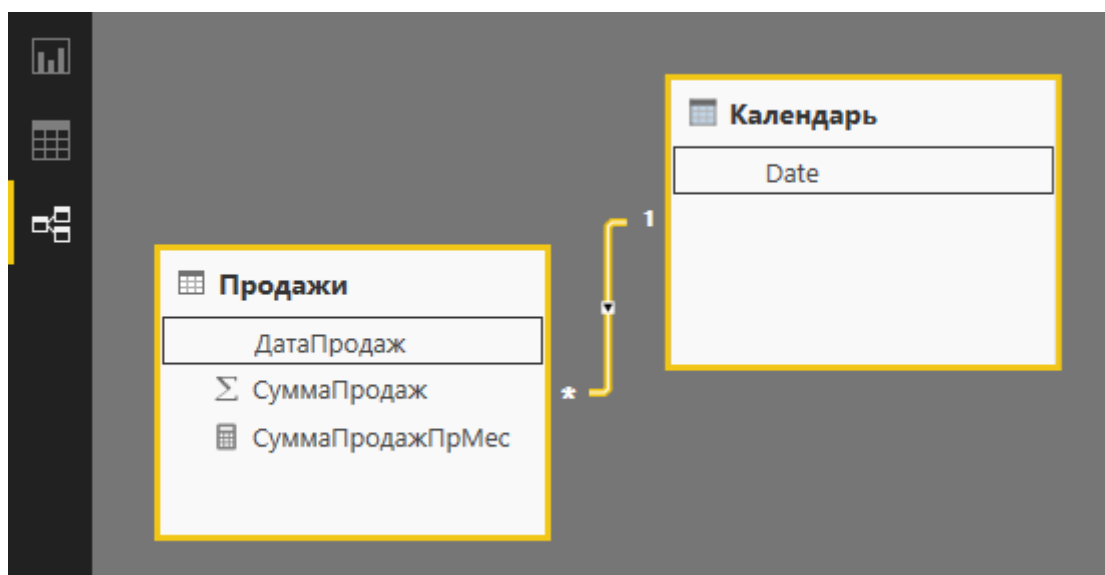
В Power BI Desktop имеется исходная таблица по продажам за 2018 год (с 16.01.2018 по 31.12.2018):

ДатаПродаж	СуммаПродаж
16 января 2018 г.	25 148
17 января 2018 г.	37 713
18 января 2018 г.	50 051
19 января 2018 г.	54 574
28 декабря 2018 г.	55 879
29 декабря 2018 г.	67 277
30 декабря 2018 г.	37 153
31 декабря 2018 г.	58 241

Задача состоит в следующем — в рамках сводной таблицы, напротив текущего месяца, получить сумму продаж за предыдущий месяц. Для этого подойдут обе рассматриваемые выше функции.

Для корректной работы DATEADD и PARALLELPERIOD мы не можем ссылаться на столбец дат, который находится в таблице продаж. Для них нужна совершенно отдельная таблица неразрывных дат «Календарь». В разборе синтаксисов функций я об этом писал (напоминаю, как создавать таблицы неразрывных дат (календари) разбирается [в этой статье](#)).

Итак, в модели данных я создал отдельный «Календарь» и связал его со столбцом [Дата Продаж] в таблице «Продажи»:



Теперь можно приступать к созданию формул:

```
ПрМесDA =  
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    DATEADD ('Календарь'[Date]; -1; MONTH)  
)
```

```
ПрМесPP =  
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    PARALLELPERIOD ('Календарь'[Date]; -1; MONTH)  
)
```

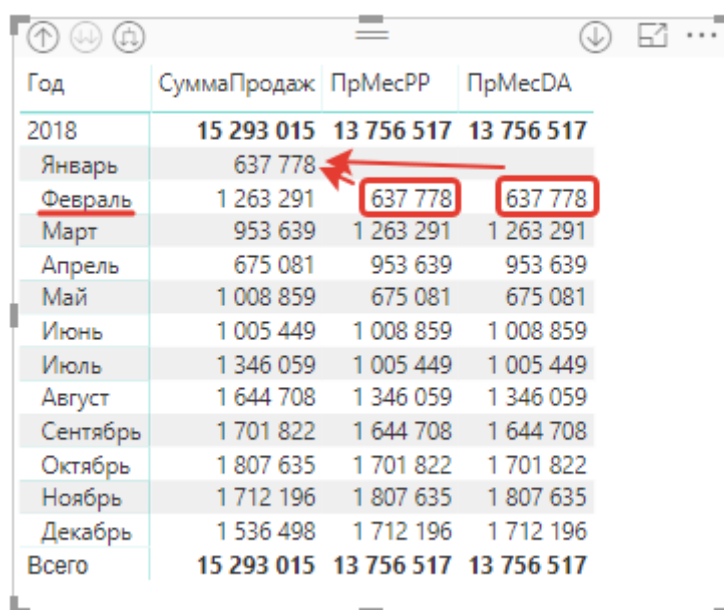
Обе формулы, которые мы записали выше, рассчитывают сумму прибыли за предыдущий месяц:

- за суммирование отвечает функция [SUM](#), которая суммирует все значения в столбце [СуммаПродаж]
- за перемещение периода отвечают функции DATEADD и PARALLELPERIOD, которые перемещают даты из календаря на 1

месяц назад, причем, напомним, «Календарь» связан с таблицей «Продажи» по столбцам дат

- и, за само вычисление суммы, под условием перемещения дат на месяц назад, отвечает DAX функция [CALCULATE](#), внутрь которой помещены все функции, участвующие в работе

Посмотрим, как поведут себя эти формулы в Power BI Desktop:



Год	СуммаПродаж	ПрМесPP	ПрМесDA
2018	15 293 015	13 756 517	13 756 517
Январь	637 778		
Февраль	1 263 291	637 778	637 778
Март	953 639	1 263 291	1 263 291
Апрель	675 081	953 639	953 639
Май	1 008 859	675 081	675 081
Июнь	1 005 449	1 008 859	1 008 859
Июль	1 346 059	1 005 449	1 005 449
Август	1 644 708	1 346 059	1 346 059
Сентябрь	1 701 822	1 644 708	1 644 708
Октябрь	1 807 635	1 701 822	1 701 822
Ноябрь	1 712 196	1 807 635	1 807 635
Декабрь	1 536 498	1 712 196	1 712 196
Всего	15 293 015	13 756 517	13 756 517

Как мы видим, формулы отработали как нужно и для текущего месяца (например, февраль) дают сумму продаж за предыдущий месяц (январь).

Обе функции DATEADD и PARALLELPERIOD пока работают абсолютно одинаково, так в чем же их различие?

А различие в том, что DATEADD — возвращает дату, которая была месяц назад. А PARALLELPERIOD весь период, который был месяц назад (в данном случае, период равен месяцу, так как в параметрах указан MONTH).

Давайте это подтвердим на нашем примере и в созданной матрице спустимся в иерархии дат до дней, тогда мы тут же заметим разницу в работе этих двух функций (для удобства демонстрации я создал две

копии визуализаций: на первой показан январь по дням, на второй — февраль по дням):

Год	СуммаПродаж	ПрМесPP	ПрМесDA
2018	15 293 015	13 756 517	13 756 517
Январь	637 778		
16	25 148		
17	37 713		
18	50 051		
19	54 574		
20	42 542		
21	30 952		
22	46 024		
23	48 289		
24	39 186		
25	27 288		
26	32 081		
27	57 345		
28	45 494		
29	36 650		
30	42 566		
31	21 875		
Февраль	1 263 291	637 778	637 778
Всего	15 293 015	13 756 517	13 756 517

Год	СуммаПродаж	ПрМесPP	ПрМесDA
31	21 875		
Февраль	1 263 291	637 778	637 778
1	41 386	637 778	
2	59 162	637 778	
3	57 079	637 778	
4	41 429	637 778	
5	62 707	637 778	
6	53 951	637 778	
7	48 176	637 778	
8	25 689	637 778	
9	23 343	637 778	
10	27 039	637 778	
11	33 746	637 778	
12	46 306	637 778	
13	60 905	637 778	
14	45 372	637 778	
15	65 812	637 778	
16	48 375	637 778	25 148
17	53 236	637 778	37 713
Всего	15 293 015	13 756 517	13 756 517

То есть, теперь, когда мы спустились в матрице к отображению дней, то DATEADD возвращает для каждого текущего дня сумму продаж именно по этому же дню, но из прошлого месяца. А PARALLELPERIOD возвращает для каждого текущего дня сумму продаж уже не по этому же дню из прошлого месяца, а именно сумму продаж за весь прошлый период (в данном случае, за месяц).

## DAX функция

# SAMEPERIODLASTYEAR

SAMEPERIODLASTYEAR () — возвращает таблицу из дат, смещенных на 1 год назад относительно дат текущего контекста.

Синтаксис:

**SAMEPERIODLASTYEAR ([Дата])**

На самом деле, SAMEPERIODLASTYEAR это упрощенный вариант DAX функции DATEADD, а именно, упрощенный ее вариант с конкретными настройками параметров:

**SAMEPERIODLASTYEAR ([Дата]) = DATEADD ([Дата]; -1; year)**

И нужна эта функция только для того, чтобы упростить написание кода [на языке DAX](#) в Power BI.

Посмотрим на практике как работают формулы на основе этих двух функций:

```
ПрГодDA =  
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    DATEADD ('Календарь'[Date]; -1; YEAR)  
)
```

```
ПрМесSPLY =  
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    SAMEPERIODLASTYEAR ('Календарь'[Date])  
)
```

Год	СуммаПродаж	ПрГодDA	ПрМесSPLY
2017	11 071 856		
Январь	968 651		
Февраль	895 176		
Март	1 055 249		
Апрель	928 462		
Май	578 847		
Июнь	453 297		
Июль	631 893		
Август	707 628		
Сентябрь	756 559		
Октябрь	1 154 008		
Ноябрь	1 512 104		
Декабрь	1 429 982		
2018	15 946 758	11 071 856	11 071 856
Январь	1 291 521	968 651	968 651
Февраль	1 263 291	895 176	895 176
Март	953 639	1 055 249	1 055 249
Апрель	675 081	928 462	928 462
Всего	27 018 614	11 071 856	11 071 856

Как мы видим, формула на основе SAMEPERIODLASTYEAR, как и на основе DATEADD, вывела значение суммы продаж за январь предыдущего года. Соответственно, в тех случаях, когда нам нужно вычислить какие-то значения за прошлый год, то лучше просто воспользоваться SAMEPERIODLASTYEAR, так как она позволяет быстрее и легче написать DAX код в Power BI.



# DAX функции PREVIOUSYEAR, PREVIOUSQUARTER, PREVIOUSMONTH, PREVIOUSDAY

[Все функции](#) группы PREVIOUS возвращают таблицу со столбцом из дат предыдущего периода на основе даты текущего контекста. Где, предыдущий период в:

- PREVIOUSYEAR () — равен предыдущему году
- PREVIOUSQUARTER () — равен предыдущему кварталу
- PREVIOUSMONTH () — равен предыдущему месяцу
- PREVIOUSDAY () — равен предыдущему дню

Синтаксис:

PREVIOUSYEAR ([Дата]; "Конец Года")

PREVIOUSQUARTER ([Дата])

PREVIOUSMONTH ([Дата])

PREVIOUSDAY ([Дата])

Где:

- [Дата] — столбец из дат или выражений, возвращающих даты
- «Конец Года» — текстовая дата, записанная в виде «02/01» («день/месяц»). Определяет дату окончания года (по умолчанию 31 декабря). Необязательный параметр.

! — Для бесперебойной работы группы функций PREVIOUS необходимо в качестве их параметров [Дата] использовать столбец из календаря непрерывных дат в [Power BI](#), то есть, создавать отдельную связанную таблицу «Календарь» с непрерывным перечислением всех дат.

## Пример формул на основе функций PREVIOUSYEAR, PREVIOUSQUARTER, PREVIOUSMONTH и PREVIOUSDAY

Для понимания сути работы группы функций PREVIOUS, рассмотрим на практике несколько примеров в Power BI.

Сперва, разберем простой пример на основе DAX функции PREVIOUSDAY.

В [Power BI Desktop](#) имеется исходная таблица дат «Календарь»:

Дата
1 января 2018 г.
2 января 2018 г.
3 января 2018 г.

Создадим в этой таблице вычисляемый столбец по следующей [формуле](#) с участием PREVIOUSDAY:

Предыдущий День = PREVIOUSDAY ('Календарь'[Дата])

Результатом выполнения этой формулы будет созданный в исходной таблице второй столбец:

<div><div>✕</div><div>✓</div></div> <div>ПредыдущийДень = PREVIOUSDAY ('Календарь' [Дата])</div>	
Дата	ПредыдущийДень
1 января 2018 г.	
2 января 2018 г.	1 января 2018 г.
3 января 2018 г.	2 января 2018 г.

где, для текущего дня (например, 2 января) из столбца [Дата], соответствует предыдущий день (1 января).

Для чего это все может быть полезно?

Сами по себе функции PREVIOUSYEAR, PREVIOUSQUARTER, PREVIOUSMONTH и PREVIOUSDAY использовать в одиночку бесполезно. Они становятся нужными в сочетании [с другими функциями языка DAX](#).

Например, тогда, когда для конкретного периода в текущем контексте нам нужно рассчитать сумму продаж предыдущего периода. И здесь, как раз, функции PREVIOUS нам очень хорошо подходят.

Давайте разберем на практике соответствующие примеры.

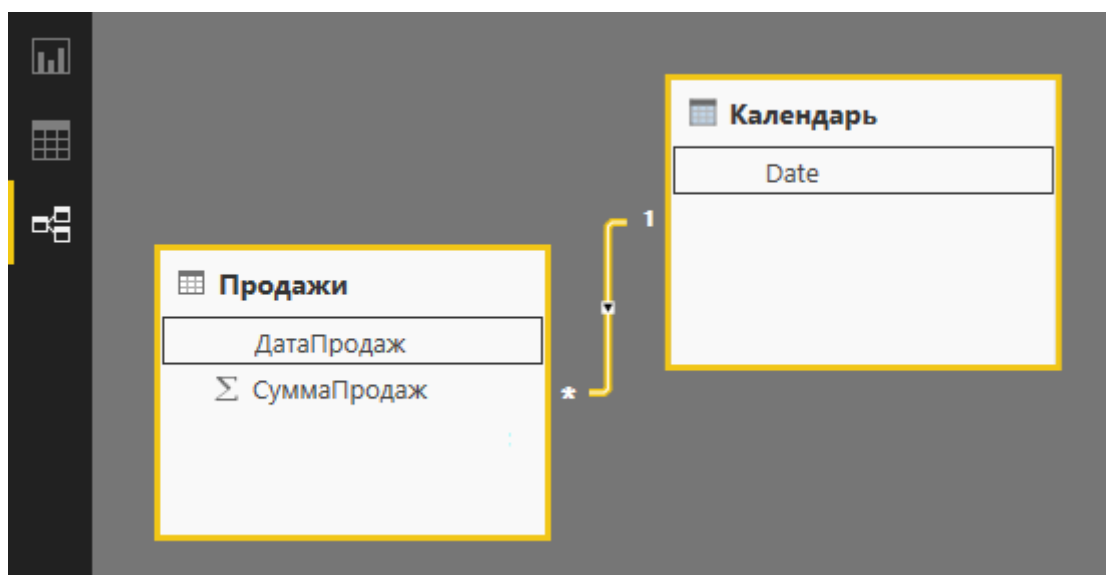
В Power BI Desktop имеется исходная таблица продаж за 2 года (с 01.01.2017 по 31.12.2018):

ДатаПродаж	СуммаПродаж
1 января 2017 г.	39 667
2 января 2017 г.	36 635
3 января 2017 г.	43 966
4 января 2017 г.	37 451
28 декабря 2018 г.	55 075
29 декабря 2018 г.	67 277
30 декабря 2018 г.	37 153
31 декабря 2018 г.	58 241

Задача — рассчитать суммы продаж за предыдущие периоды относительно периодов текущего контекста.

Для правильной работы PREVIOUSYEAR, PREVIOUSQUARTER, PREVIOUSMONTH и PREVIOUSDAY в качестве столбца [Дата] мы не можем вставлять столбец дат, который находится в исходной таблице продаж. Для них нужна совершенно отдельная таблица неразрывных дат «Календарь» (как создавать календари неразрывных дат Вы можете прочитать [в этой статье](#)).

Поэтому, создадим в модели данных новую вычисляемую таблицу «Календарь» и свяжем ее с основной исходной таблицей «Продажи» по столбцу [Дата Продаж]:



Далее, напомним формулы расчета суммы продаж за предыдущие периоды на основе группы функций PREVIOUS:

Предыдущий Год =

```
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    PREVIOUSYEAR ('Календарь'[Date])  
)
```

Предыдущий Квартал =

```
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    PREVIOUSQUARTER ('Календарь'[Date])  
)
```

Предыдущий Месяц =

```
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    PREVIOUSMONTH ('Календарь'[Date])  
)
```

```

Предыдущий День =
CALCULATE (
    SUM ('Продажи'[СуммаПродаж]);
    PREVIOUSDAY ('Календарь'[Date])
)
    
```

Где:

- сумма продаж рассчитывается при помощи DAX функции [SUM](#)
- предыдущие периоды рассчитываются при помощи рассматриваемой группы PREVIOUS
- условие «вычисление суммы только за предыдущий период» создается функцией [CALCULATE](#), внутрь которой вложены все остальные функции, участвующие в работе

Результатом выполнения всех формул, будут следующие сводные таблицы в Power BI Desktop:

Год	СуммаПродаж	ПредыдущийГод
2017	11 071 856	
2018	15 946 758	11 071 856
Всего	27 018 614	

Год	СуммаПродаж	ПредыдущийКвартал
2017	11 071 856	
Кв. 1	2 919 076	
Кв. 2	1 960 606	2 919 076
Кв. 3	2 096 080	1 960 606
Кв. 4	4 096 094	2 096 080
Всего	27 018 614	

Год	СуммаПродаж	ПредыдущийДень
2017	11 071 856	
Январь	968 651	
1	39 667	
2	36 635	39 667
3	43 966	36 635
4	37 451	43 966
5	26 716	37 451
6	26 508	26 716
Всего	27 018 614	

Год	СуммаПродаж	ПредыдущийМесяц
2017	11 071 856	
Январь	968 651	
Февраль	895 176	968 651
Март	1 055 249	895 176
Апрель	928 462	1 055 249
Всего	27 018 614	

Из примера мы видим, что все функции из группы PREVIOUS отлично справились со своей задачей и везде возвращены суммы продаж соответствующего предыдущего периода.

# DAX функции NEXTYEAR, NEXTQUARTER, NEXTMONTH и NEXTDAY

Все функции группы NEXT возвращают таблицу со столбцом из дат следующего периода на основе даты текущего контекста. Где, следующий период в:

- NEXTYEAR () — равен следующему году
- NEXTQUARTER () — равен следующему кварталу
- NEXTMONTH () — равен следующему месяцу
- NEXTDAY () — равен следующему дню

Синтаксис:

NEXTYEAR ([Дата]; "Конец Года")

NEXTQUARTER ([Дата])

NEXTMONTH ([Дата])

NEXTDAY ([Дата])

Где:

- [Дата] — столбец из дат или выражений (табличных, логических и прочих), возвращающих даты
- «Конец Года» — (необязательный параметр) текстовая дата, записанная в виде «15/06» («день/месяц»). Определяет дату окончания года (по умолчанию «31/12», то есть, 31 декабря).

! — Для безошибочной работы группы функций NEXT необходимо в качестве их параметров [Дата] использовать столбец из календаря непрерывных дат в Power BI, то есть, создавать отдельную связанную таблицу «Календарь» с непрерывным перечислением всех дат.

## Пример формул на основе функций NEXTYEAR, NEXTQUARTER, NEXTMONTH и NEXTDAY

В качестве практики рассмотрим несколько примеров в Power BI.

Сначала разберем простой пример на основе DAX функции NEXTDAY.

Итак, в [Power BI Desktop](#) имеется исходная таблица «Календарь»:

Дата
1 января 2018 г.
2 января 2018 г.
3 января 2018 г.

Для наглядности примера создадим в этой таблице второй столбец со следующими днями, относительно дат в столбце [Дата]. Для этого составим следующую формулу с участием NEXTDAY:

Следующий День = NEXTDAY ('Календарь'[Дата])

В итоге эта формула возвратит ожидаемый результат:

✕	✓	СледующийДень = NEXTDAY ('Календарь'[Дата])	
Дата	СледующийДень		
1 января 2018 г.	2 января 2018 г.		
2 января 2018 г.	3 января 2018 г.		
3 января 2018 г.			

где, для текущей даты (например, 1 января) в столбце [Дата] соответствует следующий день (2 января).

Все эти 4 функции NEXTYEAR, NEXTQUARTER, NEXTMONTH и NEXTDAY просто так, в одиночку не применяют. А используют совместно с [другими функциями в DAX](#) для создания сложных [формул](#), мер и вычислений.

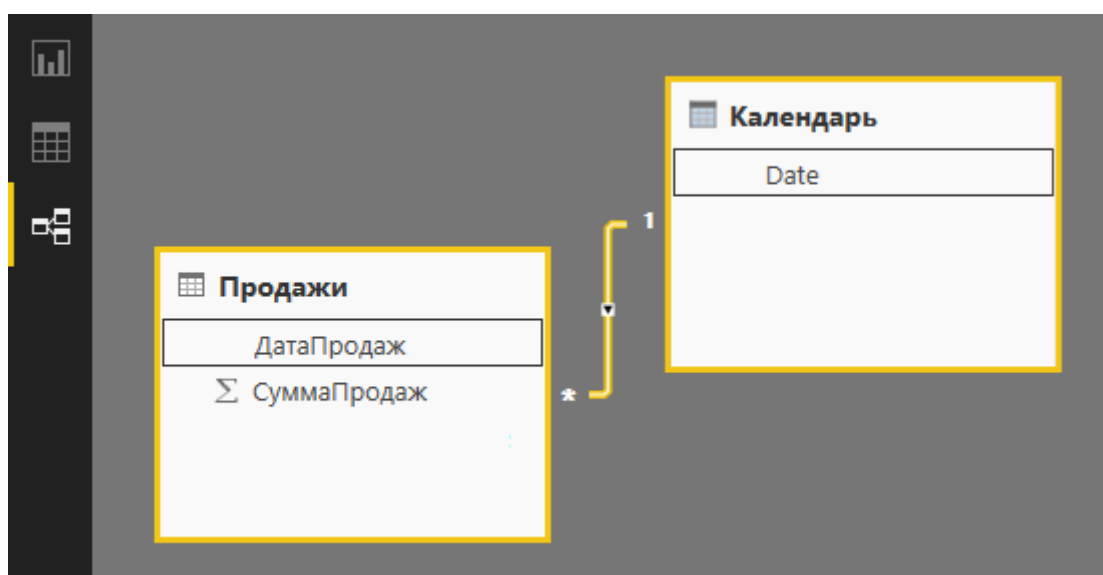
Соответственно, давайте попробуем разобрать примеры более сложных формул [на языке DAX](#), а конкретно, формулы,

рассчитывающие для даты из текущего контекста сумму прибыли следующего (будущего) периода.

В Power BI имеется исходная таблица с информацией по продажам с 1 января 2017 года по 31 декабря 2018 года:

ДатаПродаж	СуммаПродаж
1 января 2017 г.	39 667
2 января 2017 г.	36 635
3 января 2017 г.	43 966
4 января 2017 г.	37 451
28 декабря 2018 г.	33 075
29 декабря 2018 г.	67 277
30 декабря 2018 г.	37 153
31 декабря 2018 г.	58 241

Так как в параметрах функций NEXTYEAR, NEXTQUARTER, NEXTMONTH и NEXTDAY нельзя использовать ссылку на исходный столбец с датами продаж из таблицы «Продажи», а нужно ссылаться на отдельный календарь из непрерывных дат, то создадим в модели данных отдельную таблицу из непрерывных дат «Календарь» и свяжем ее с исходной таблицей «Продажи» (как создавать календари с непрерывными датами Вы можете прочитать [в этой статье](#)):





Теперь, можно создать формулы мер, рассчитывающих значение суммы продаж в следующем году, квартале, месяце или дне относительно даты текущего контекста:

Следующий Год =

```
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    NEXTYEAR ('Календарь'[Date])  
)
```

Следующий Квартал =

```
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    NEXTQUARTER ('Календарь'[Date])  
)
```

Следующий Месяц =

```
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    NEXTMONTH ('Календарь'[Date])  
)
```

Следующий День =

```
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    NEXTDAY ('Календарь'[Date])  
)
```

Где:

- сумму продаж мы рассчитали при помощи DAX функции [SUM](#)
- следующие периоды возвратили при помощи рассматриваемых функций из группы NEXT

- условие «расчет суммы продаж только за следующие периоды относительно дат текущего контекста» накладывается при помощи функции [CALCULATE](#)

Чтобы проверить работу этих формул, создадим на основе них в Power BI Desktop, сводные таблицы:

The image displays four screenshots of Power BI PivotTables, each illustrating a different time-based calculation function. Arrows indicate the relationship between the current period and the next period.

**Table 1: NEXTYEAR**

Год	СуммаПродаж	СледующийГод
2017	11 071 856	15 946 758
2018	15 946 758	
Всего	27 018 614	

**Table 2: NEXTQUARTER**

Год	СуммаПродаж	СледующийКвартал
2017	11 071 856	3 508 451
Кв. 1	2 919 076	1 960 606
Кв. 2	1 960 606	2 096 080
Кв. 3	2 096 080	4 096 094
Кв. 4	4 096 094	3 508 451
Всего	27 018 614	

**Table 3: NEXTMONTH**

Год	СуммаПродаж	СледующийМесяц
2017	11 071 856	1 291 521
Январь	968 651	895 176
Февраль	895 176	1 055 249
Март	1 055 249	928 462
Апрель	928 462	578 847
Всего	27 018 614	

**Table 4: NEXTDAY**

Год	СуммаПродаж	СледующийДень
2017	11 071 856	26 412
Январь	968 651	43 391
1	39 667	36 635
2	36 635	43 966
3	43 966	37 451
4	37 451	26 716
5	26 716	26 508
6	26 508	32 179
Всего	27 018 614	

В сводных таблицах мы можем наблюдать работу функций NEXTYEAR, NEXTQUARTER, NEXTMONTH и NEXTDAY — везде, в рамках периода текущего контекста, возвращены суммы продаж за следующий период.

То есть, для 2017 года возвращена сумма прибыли за следующий 2018 год, для января возвращена сумма прибыли за следующий месяц (февраль) и так далее.

# DAX функции DATESYTD, DATESQTD И DATESMTD

DATESYTD () — создает таблицу, со столбцом из дат от начала года до текущего дня в рамках текущего контекста.

DATESQTD () — создает таблицу, со столбцом из дат от начала квартала до текущего дня в рамках текущего контекста.

DATESMTD () — создает таблицу, со столбцом из дат от начала месяца до текущего дня в рамках текущего контекста.

Синтаксис:

DATESYTD ([Дата]; "Конец Года")

DATESQTD ([Дата])

DATESMTD ([Дата])

Где:

- [Дата] — столбец из дат или выражений, возвращающих даты
- «Конец Года» — текстовая дата, записанная в виде «01/05» («день/месяц»). Определяет дату окончания года (по умолчанию 31 декабря). Необязательный параметр.

! — Для безошибочной работы функций DATESYTD, DATESQTD и DATESMTD необходимо в качестве их параметров [Дата] использовать столбец из календаря непрерывных дат в [Power BI](#), то есть, создавать отдельную связанную таблицу «Календарь» с непрерывным перечислением всех дат.

## Пример формул на основе группы time intelligence функций DATES (YTD, QTD и MTD)

В [Power BI Desktop](#) имеется модель данных, состоящая из 2 связанных таблиц: таблицы фактов «Продажи» и таблицы измерений (справочник) «Календарь»:

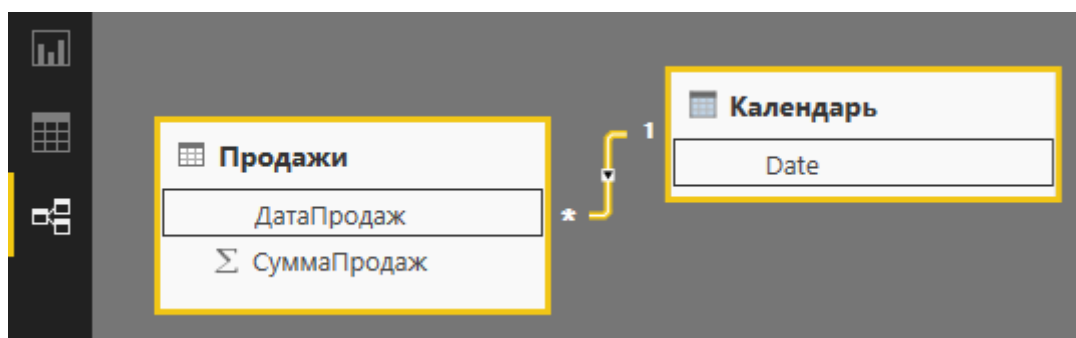


Таблица «Продажи» содержит даты и суммы продаж за 2 года, начиная с 1 января 2017 года, заканчивая 31 декабря 2018 года:

ДатаПродаж	СуммаПродаж
1 января 2017 г.	39 667
2 января 2017 г.	36 635
3 января 2017 г.	43 966
4 января 2017 г.	37 451
...	...
28 декабря 2018 г.	55 073
29 декабря 2018 г.	67 277
30 декабря 2018 г.	37 153
31 декабря 2018 г.	58 241

Таблица «Календарь» содержит в себе неразрывный набор дат за эти 2 года с шагом в 1 день. Она не является исходной. Календарь был создан мною специально (как создавать календари в Power BI читайте [в этой статье](#)), так как функции DATESYTD, DATESQTD и DATESMTD не могут работать напрямую со столбцом дат в таблице «Продажи» и для их функционирования нужна отдельная связанная таблица с неразрывными датами:

Календарь = CALENDARAUTO()	
Date	
1 января 2017 г.	
2 января 2017 г.	
3 января 2017 г.	
4 января 2017 г.	
28 декабря 2018 г.	
29 декабря 2018 г.	
30 декабря 2018 г.	
31 декабря 2018 г.	

В качестве примера работы рассматриваемых функций, давайте напишем [формулы](#) расчета накопительной суммы продаж от начала года, от начала квартала, от начала месяца до текущей даты текущего контекста. А затем, выведем эти формулы в визуализациях в Power BI Desktop.

Итак, формулы расчета накопительной суммы продаж на основе функции DATESYTD, DATESQTD и DATESMTD будут следующими:

```
Итого С Начала Года =  
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    DATESYTD ('Календарь'[Date])  
)
```

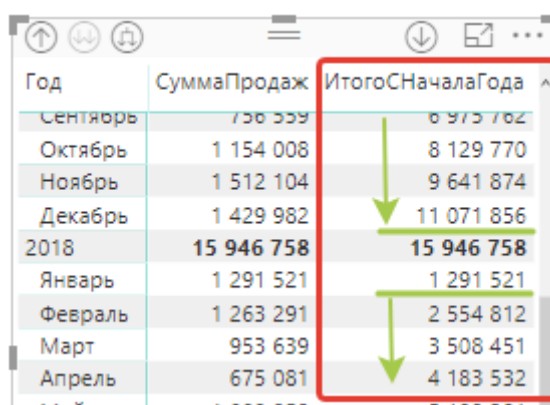
```
Итого С Начала Квартала =  
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    DATESQTD ('Календарь'[Date])  
)
```

```
Итого С Начала Месяца =  
CALCULATE (  
    SUM ('Продажи'[СуммаПродаж]);  
    DATESMTD ('Календарь'[Date])  
)
```

Где:

- сумму продаж мы посчитали при помощи функции [SUM](#)
- таблицы с соответствующим набором дат нам возвратили, рассматриваемые в этой статье, time intelligence функции
- и, соблюдение условия «расчет суммы продаж накопительным итогом с начала периода и до текущей даты» нам позволила создать DAX функция [CALCULATE](#), внутри которой мы и встроили остальные функции из наших формул

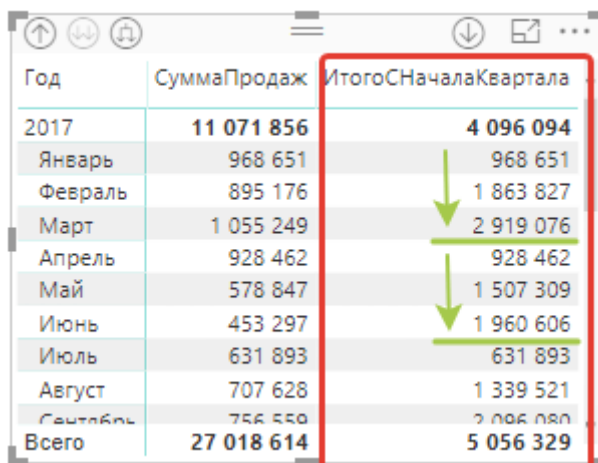
Итак, результатом выполнения формулы на основе функции DATESYTD, в Power BI будет сводная таблица, где, действительно, сумма продаж накапливается каждый месяц в течение всего года, и как только год закончился, сумма сбросилась и далее со следующего года она опять начала накапливаться:



Год	СуммаПродаж	ИтогоСНачалаГода
Сентябрь	736 559	6 975 762
Октябрь	1 154 008	8 129 770
Ноябрь	1 512 104	9 641 874
Декабрь	1 429 982	11 071 856
2018	15 946 758	15 946 758
Январь	1 291 521	1 291 521
Февраль	1 263 291	2 554 812
Март	953 639	3 508 451
Апрель	675 081	4 183 532

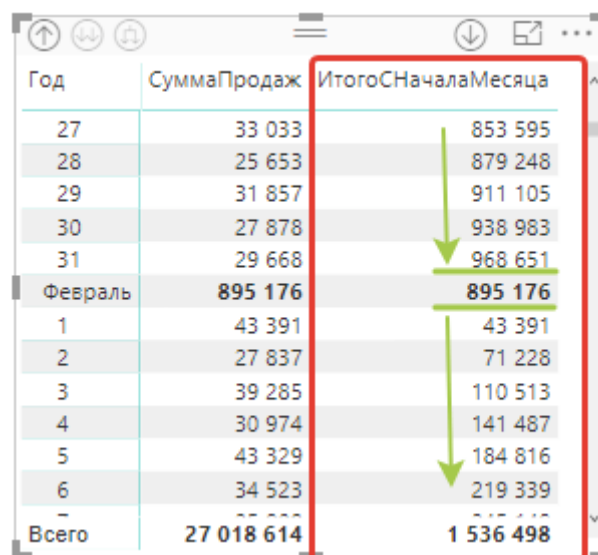
С формулой на основе функции DATESQTD та же самая история — в Power BI будет сводная таблица, где сумма продаж накапливается каждый месяц в течение конкретного квартала, и как только квартал

заканчивается, сумма сбрасывается и далее со следующего квартала она опять начинает накапливаться:



Год	СуммаПродаж	ИтогоСНачалаКвартала
2017	11 071 856	4 096 094
Январь	968 651	968 651
Февраль	895 176	1 863 827
Март	1 055 249	2 919 076
Апрель	928 462	928 462
Май	578 847	1 507 309
Июнь	453 297	1 960 606
Июль	631 893	631 893
Август	707 628	1 339 521
Сентябрь	756 550	2 096 071
Всего	27 018 614	5 056 329

Соответственно, точно такой же результат будет и у формулы на основе функции DATESMTD — в Power BI будет сводная таблица, где сумма продаж накапливается каждый день в течение всего месяца, и как только месяц заканчивается, сумма сбрасывается и далее со следующего месяца она вновь начинает накапливаться:



Год	СуммаПродаж	ИтогоСНачалаМесяца
27	33 033	853 595
28	25 653	879 248
29	31 857	911 105
30	27 878	938 983
31	29 668	968 651
Февраль	895 176	895 176
1	43 391	43 391
2	27 837	71 228
3	39 285	110 513
4	30 974	141 487
5	43 329	184 816
6	34 523	219 339
Всего	27 018 614	1 536 498

# DAX функции TOTALYTD, TOTALQTD, TOTALMTD

TOTALYTD () — вычисляет заданное выражение от начала года до текущего дня в рамках текущего контекста.

TOTALQTD () — вычисляет заданное выражение от начала квартала до текущего дня в рамках текущего контекста.

TOTALMTD () — вычисляет заданное выражение от начала месяца до текущего дня в рамках текущего контекста.

Синтаксис:

TOTALYTD (Выражение; [Дата]; Фильтр; "Конец Года")

TOTALQTD (Выражение; [Дата]; Фильтр)

TOTALMTD (Выражение; [Дата]; Фильтр)

Где:

- Выражение — выражение, возвращающее единственное скалярное значение
- [Дата] — столбец из дат или выражений, возвращающих даты
- Фильтр — (необязательный параметр) необходимые фильтры для вычисления выражения
- «Конец Года» — (необязательный параметр) текстовая дата, записанная в виде «01/07» («день/месяц»). Определяет дату окончания года (по умолчанию «31/12»)

! — Для корректной работы функций TOTALYTD, TOTALQTD и TOTALMTD необходимо в качестве их параметров [Дата] использовать столбец из календаря непрерывных дат в [Power BI](#), то есть, создавать отдельную связанную таблицу «Календарь» с непрерывным перечислением всех дат.



## Пример формул на основе time intelligence функций группы TOTAL (YTD, QTD и MTD)

В модели данных [Power BI Desktop](#) имеется исходная таблица «Продажи-Затраты», содержащая даты и по каждой дате, соответственно, сумму продаж и сумму затрат за 2 года (с 1.01.2017 по 31.12.2018):

Дата	Продажи	Затраты
1 января 2017 г.	33931	19554
2 января 2017 г.	22204	10213
3 января 2017 г.	34820	13955
4 января 2017 г.	35674	18955
28 декабря 2018 г.	33473	19128
29 декабря 2018 г.	44153	13756
30 декабря 2018 г.	57104	17656
31 декабря 2018 г.	47107	16437

Задача состоит в следующем: создать [отчет в Power BI](#), который бы показывал накопленную прибыль компании за весь текущий период. А если конкретнее, то в отчете должны быть три визуализации вида «Матрица» и в каждой визуализации отображена накопленная прибыль за 3 периода (в первой таблице — итог по прибыли за текущий год, во второй — итог по прибыли за текущий квартал и в третьей — накопленный итог прибыли за текущий месяц).

Так как накопленную сумму прибыли в этом примере нужно рассчитывать от начала текущего года / квартала / месяца до текущей даты, то это все можно легко решить при помощи, рассматриваемых в этой статье, time intelligence функций языка DAX: TOTALYTD, TOTALQTD и TOTALMTD. Именно они в Power BI рассчитывают любые выражения в рамках конкретного периода текущего контекста.

В исходной таблице суммы прибыли организации у нас нет, поэтому, нам ее нужно рассчитать. Для этого, в качестве параметра «Выражение» в функциях TOTALYTD, TOTALQTD и TOTALMTD нужно

прописать формулы расчета самой прибыли, которая равняется разнице сумм продаж и затрат. И рассчитаем мы ее при помощи еще одной функции в DAX — SUMX, о которой Вы можете подробно прочитать [в этой статье](#).

И перед тем, как написать итоговые формулы, нам осталось обсудить еще один момент. Дело в том, что функции TOTALYTD, TOTALQTD и TOTALMTD не работают в Power BI с датами в исходных таблицах.

В синтаксисе этих функций я писал, что для их использования нужна специальная отдельная таблица «Календарь» с неразрывными датами, связанная с нужной исходной таблицей. Как создавать календари неразрывных дат в Power BI Вы можете прочитать [в этой статье](#).

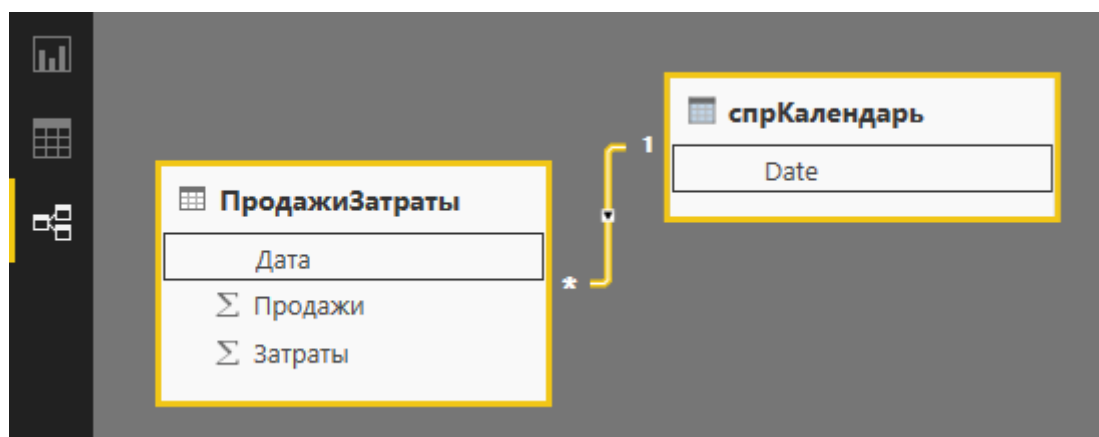
Поэтому, в нашей модели данных я создал таблицу «Справочник Календарь»:

✕

✓

спрКалендарь = CALENDARAUTO()

Date	
1 января 2017 г.	
2 января 2017 г.	
3 января 2017 г.	
4 января 2017 г.	
5 января 2017 г.	
6 января 2017 г.	
7 января 2017 г.	
8 января 2017 г.	
9 января 2017 г.	
10 января 2017 г.	
11 января 2017 г.	
12 января 2017 г.	
13 января 2017 г.	
14 января 2017 г.	
15 января 2017 г.	
16 января 2017 г.	
17 января 2017 г.	
18 января 2017 г.	
19 января 2017 г.	
20 января 2017 г.	
21 января 2017 г.	
22 января 2017 г.	
23 января 2017 г.	
24 января 2017 г.	
25 января 2017 г.	
26 января 2017 г.	
27 января 2017 г.	
28 января 2017 г.	
29 января 2017 г.	
30 января 2017 г.	
31 января 2017 г.	
1 февраля 2017 г.	
2 февраля 2017 г.	
3 февраля 2017 г.	
4 февраля 2017 г.	
5 февраля 2017 г.	
6 февраля 2017 г.	
7 февраля 2017 г.	
8 февраля 2017 г.	
9 февраля 2017 г.	
10 февраля 2017 г.	
11 февраля 2017 г.	
12 февраля 2017 г.	
13 февраля 2017 г.	
14 февраля 2017 г.	
15 февраля 2017 г.	
16 февраля 2017 г.	
17 февраля 2017 г.	
18 февраля 2017 г.	
19 февраля 2017 г.	
20 февраля 2017 г.	
21 февраля 2017 г.	
22 февраля 2017 г.	
23 февраля 2017 г.	
24 февраля 2017 г.	
25 февраля 2017 г.	
26 февраля 2017 г.	
27 февраля 2017 г.	
28 февраля 2017 г.	
1 марта 2017 г.	
2 марта 2017 г.	
3 марта 2017 г.	
4 марта 2017 г.	
5 марта 2017 г.	
6 марта 2017 г.	
7 марта 2017 г.	
8 марта 2017 г.	
9 марта 2017 г.	
10 марта 2017 г.	
11 марта 2017 г.	
12 марта 2017 г.	
13 марта 2017 г.	
14 марта 2017 г.	
15 марта 2017 г.	
16 марта 2017 г.	
17 марта 2017 г.	
18 марта 2017 г.	
19 марта 2017 г.	
20 марта 2017 г.	
21 марта 2017 г.	
22 марта 2017 г.	
23 марта 2017 г.	
24 марта 2017 г.	
25 марта 2017 г.	
26 марта 2017 г.	
27 марта 2017 г.	
28 марта 2017 г.	
29 марта 2017 г.	
30 марта 2017 г.	
31 марта 2017 г.	
1 апреля 2017 г.	
2 апреля 2017 г.	
3 апреля 2017 г.	
4 апреля 2017 г.	
5 апреля 2017 г.	
6 апреля 2017 г.	
7 апреля 2017 г.	
8 апреля 2017 г.	
9 апреля 2017 г.	
10 апреля 2017 г.	
11 апреля 2017 г.	
12 апреля 2017 г.	
13 апреля 2017 г.	
14 апреля 2017 г.	
15 апреля 2017 г.	
16 апреля 2017 г.	
17 апреля 2017 г.	
18 апреля 2017 г.	
19 апреля 2017 г.	
20 апреля 2017 г.	
21 апреля 2017 г.	
22 апреля 2017 г.	
23 апреля 2017 г.	
24 апреля 2017 г.	
25 апреля 2017 г.	
26 апреля 2017 г.	
27 апреля 2017 г.	
28 апреля 2017 г.	
29 апреля 2017 г.	
30 апреля 2017 г.	
1 мая 2017 г.	
2 мая 2017 г.	
3 мая 2017 г.	
4 мая 2017 г.	
5 мая 2017 г.	
6 мая 2017 г.	
7 мая 2017 г.	
8 мая 2017 г.	
9 мая 2017 г.	
10 мая 2017 г.	
11 мая 2017 г.	
12 мая 2017 г.	
13 мая 2017 г.	
14 мая 2017 г.	
15 мая 2017 г.	
16 мая 2017 г.	
17 мая 2017 г.	
18 мая 2017 г.	
19 мая 2017 г.	
20 мая 2017 г.	
21 мая 2017 г.	
22 мая 2017 г.	
23 мая 2017 г.	
24 мая 2017 г.	
25 мая 2017 г.	
26 мая 2017 г.	
27 мая 2017 г.	
28 мая 2017 г.	
29 мая 2017 г.	
30 мая 2017 г.	
31 мая 2017 г.	
1 июня 2017 г.	
2 июня 2017 г.	
3 июня 2017 г.	
4 июня 2017 г.	
5 июня 2017 г.	
6 июня 2017 г.	
7 июня 2017 г.	
8 июня 2017 г.	
9 июня 2017 г.	
10 июня 2017 г.	
11 июня 2017 г.	
12 июня 2017 г.	
13 июня 2017 г.	
14 июня 2017 г.	
15 июня 2017 г.	
16 июня 2017 г.	
17 июня 2017 г.	
18 июня 2017 г.	
19 июня 2017 г.	
20 июня 2017 г.	
21 июня 2017 г.	
22 июня 2017 г.	
23 июня 2017 г.	
24 июня 2017 г.	
25 июня 2017 г.	
26 июня 2017 г.	
27 июня 2017 г.	
28 июня 2017 г.	
29 июня 2017 г.	
30 июня 2017 г.	
1 июля 2017 г.	
2 июля 2017 г.	
3 июля 2017 г.	
4 июля 2017 г.	
5 июля 2017 г.	
6 июля 2017 г.	
7 июля 2017 г.	
8 июля 2017 г.	
9 июля 2017 г.	
10 июля 2017 г.	
11 июля 2017 г.	
12 июля 2017 г.	
13 июля 2017 г.	
14 июля 2017 г.	
15 июля 2017 г.	
16 июля 2017 г.	
17 июля 2017 г.	
18 июля 2017 г.	
19 июля 2017 г.	
20 июля 2017 г.	
21 июля 2017 г.	
22 июля 2017 г.	
23 июля 2017 г.	
24 июля 2017 г.	
25 июля 2017 г.	
26 июля 2017 г.	
27 июля 2017 г.	
28 июля 2017 г.	
29 июля 2017 г.	
30 июля 2017 г.	
31 июля 2017 г.	
1 августа 2017 г.	
2 августа 2017 г.	
3 августа 2017 г.	
4 августа 2017 г.	
5 августа 2017 г.	
6 августа 2017 г.	
7 августа 2017 г.	
8 августа 2017 г.	
9 августа 2017 г.	
10 августа 2017 г.	
11 августа 2017 г.	
12 августа 2017 г.	
13 августа 2017 г.	
14 августа 2017 г.	
15 августа 2017 г.	
16 августа 2017 г.	
17 августа 2017 г.	
18 августа 2017 г.	
19 августа 2017 г.	
20 августа 2017 г.	
21 августа 2017 г.	
22 августа 2017 г.	
23 августа 2017 г.	
24 августа 2017 г.	
25 августа 2017 г.	
26 августа 2017 г.	
27 августа 2017 г.	
28 августа 2017 г.	
29 августа 2017 г.	
30 августа 2017 г.	
31 августа 2017 г.	
1 сентября 2017 г.	
2 сентября 2017 г.	
3 сентября 2017 г.	
4 сентября 2017 г.	
5 сентября 2017 г.	
6 сентября 2017 г.	
7 сентября 2017 г.	
8 сентября 2017 г.	
9 сентября 2017 г.	
10 сентября 2017 г.	
11 сентября 2017 г.	
12 сентября 2017 г.	
13 сентября 2017 г.	
14 сентября 2017 г.	
15 сентября 2017 г.	
16 сентября 2017 г.	
17 сентября 2017 г.	
18 сентября 2017 г.	
19 сентября 2017 г.	
20 сентября 2017 г.	
21 сентября 2017 г.	
22 сентября 2017 г.	
23 сентября 2017 г.	
24 сентября 2017 г.	
25 сентября 2017 г.	
26 сентября 2017 г.	
27 сентября 2017 г.	
28 сентября 2017 г.	
29 сентября 2017 г.	
30 сентября 2017 г.	
1 октября 2017 г.	
2 октября 2017 г.	
3 октября 2017 г.	
4 октября 2017 г.	
5 октября 2017 г.	
6 октября 2017 г.	
7 октября 2017 г.	
8 октября 2017 г.	
9 октября 2017 г.	
10 октября 2017 г.	
11 октября 2017 г.	
12 октября 2017 г.	
13 октября 2017 г.	
14 октября 2017 г.	
15 октября 2017 г.	
16 октября 2017 г.	
17 октября 2017 г.	
18 октября 2017 г.	
19 октября 2017 г.	
20 октября 2017 г.	
21 октября 2017 г.	
22 октября 2017 г.	
23 октября 2017 г.	
24 октября 2017 г.	
25 октября 2017 г.	
26 октября 2017 г.	
27 октября 2017 г.	
28 октября 2017 г.	
29 октября 2017 г.	
30 октября 2017 г.	
31 октября 2017 г.	
1 ноября 2017 г.	
2 ноября 2017 г.	
3 ноября 2017 г.	
4 ноября 2017 г.	
5 ноября 2017 г.	
6 ноября 2017 г.	
7 ноября 2017 г.	
8 ноября 2017 г.	
9 ноября 2017 г.	
10 ноября 2017 г.	
11 ноября 2017 г.	
12 ноября 2017 г.	
13 ноября 2017 г.	
14 ноября 2017 г.	
15 ноября 2017 г.	
16 ноября 2017 г.	
17 ноября 2017 г.	
18 ноября 2017 г.	
19 ноября 2017 г.	
20 ноября 2017 г.	
21 ноября 2017 г.	
22 ноября 2017 г.	
23 ноября 2017 г.	
24 ноября 2017 г.	
25 ноября 2017 г.	
26 ноября 2017 г.	
27 ноября 2017 г.	
28 ноября 2017 г.	
29 ноября 2017 г.	
30 ноября 2017 г.	
1 декабря 2017 г.	
2 декабря 2017 г.	
3 декабря 2017 г.	
4 декабря 2017 г.	
5 декабря 2017 г.	
6 декабря 2017 г.	
7 декабря 2017 г.	
8 декабря 2017 г.	
9 декабря 2017 г.	
10 декабря 2017 г.	
11 декабря 2017 г.	
12 декабря 2017 г.	
13 декабря 2017 г.	
14 декабря 2017 г.	
15 декабря 2017 г.	
16 декабря 2017 г.	
17 декабря 2017 г.	
18 декабря 2017 г.	
19 декабря 2017 г.	
20 декабря 2017 г.	
21 декабря 2017 г.	
22 декабря 2017 г.	
23 декабря 2017 г.	
24 декабря 2017 г.	
25 декабря 2017 г.	
26 декабря 2017 г.	
27 декабря 2017 г.	
28 декабря 2017 г.	
29 декабря 2017 г.	
30 декабря 2017 г.	
31 декабря 2017 г.	
1 января 2018 г.	
2 января 2018 г.	
3 января 2018 г.	
4 января 2018 г.	
5 января 2018 г.	
6 января 2018 г.	
7 января 2018 г.	
8 января 2018 г.	
9 января 2018 г.	
10 января 2018 г.	
11 января 2018 г.	
12 января 2018 г.	
13 января 2018 г.	
14 января 2018 г.	
15 января 2018 г.	
16 января 2018 г.	
17 января 2018 г.	
18 января 2018 г.	
19 января 2018 г.	
20 января 2018 г.	
21 января 2018 г.	
22 января 2018 г.	
23 января 2018 г.	
24 января 2018 г.	
25 января 2018 г.	
26 января 2018 г.	
27 января 2018 г.	
28 января 2018 г.	
29 января 2018 г.	
30 января 2018 г.	
31 января 2018 г.	
1 февраля 2018 г.	
2 февраля 2018 г.	
3 февраля 2018 г.	
4 февраля 2018 г.	
5 февраля 2018 г.	
6 февраля 2018 г.	
7 февраля 2018 г.	
8 февраля 2018 г.	
9 февраля 2018 г.	
10 февраля 2018 г.	
11 февраля 2018 г.	
12 февраля 2018 г.	
13 февраля 2018 г.	
14 февраля 2018 г.	
15 февраля 2018 г.	
16 февраля 2018 г.	
17 февраля 2018 г.	
18 февраля 2018 г.	
19 февраля 2018 г.	
20 февраля 2018 г.	
21 февраля 2018 г.	
22 февраля 2018 г.	
23 февраля 2018 г.	
24 февраля 2018 г.	
25 февраля 2018 г.	
26 февраля 2018 г.	
27 февраля 2018 г.	
28 февраля 2018 г.	
1 марта 2018 г.	
2 марта 2018 г.	
3 марта 2018 г.	
4 марта 2018 г.	
5 марта 2018 г.	
6 марта 2018 г.	
7 марта 2018 г.	
8 марта 2018 г.	
9 марта 2018 г.	
10 марта 2018 г.	
11 марта 2018 г.	
12 марта 2018 г.	
13 марта 2018 г.	
14 марта 2018 г.	
15 марта 2018 г.	
16 марта 2018 г.	
17 марта 2018 г.	
18 марта 2018 г.	
19 марта 2018 г.	
20 марта 2018 г.	
21 марта 2018 г.	
22 марта 2018 г.	
23 марта 2018 г.	
24 марта 2018 г.	
25 марта 2018 г.	
26 марта 2018 г.	
27 марта 2018 г.	
28 марта 2018 г.	
29 марта 2018 г.	
30 марта 2018 г.	
31 марта 2018 г.	
1 апреля 2018 г.	
2 апреля 2018 г.	
3 апреля 2018 г.	
4 апреля 2018 г.	
5 апреля 2018 г.	
6 апреля 2018 г.	
7 апреля 2018 г.	
8 апреля 2018 г.	
9 апреля 2018 г.	
10 апреля 2018 г.	
11 апреля 2018 г.	
12 апреля 2018 г.	
13 апреля 2018 г.	
14 апреля 2018 г.	
15 апреля 2018 г.	
16 апреля 2018 г.	
17 апреля 2018 г.	
18 апреля 2018 г.	
19 апреля 2018 г.	
20 апреля 2018 г.	
21 апреля 2018 г.	
22 апреля 2018 г.	
23 апреля 2018 г.	
24 апреля 2018 г.	
25 апреля 2018 г.	
26 апреля 2018 г.	
27 апреля 2018 г.	
28 апреля 2018 г.	
29 апреля 2018 г.	
30 апреля 2018 г.	
1 мая 2018 г.	
2 мая 2018 г.	
3 мая 2018 г.	
4 мая 2018 г.	
5 мая 2018 г.	
6 мая 2018 г.	
7 мая 2018 г.	
8 мая 2018 г.	
9 мая 2018 г.	
10 мая 2018 г.	
11 мая 2018 г.	
12 мая 2018 г.	
13 мая 2018 г.	
14 мая 2018 г.	
15 мая 2018 г.	
16 мая 2018 г.	
17 мая 2018 г.	
18 мая 2018 г.	
19 мая 2018 г.	
20 мая 2018 г.	
21 мая 2018 г.	
22 мая 2018 г.	
23 мая 2018 г.	
24 мая 2018 г.	
25 мая 2018 г.	
26 мая 2018 г.	
27 мая 2018 г.	
28 мая 2018 г.	
29 мая 2018 г.	
30 мая 2018 г.	
31 мая 2018 г.	
1 июня 2018 г.	
2 июня 2018 г.	
3 июня 2018 г.	
4 июня 2018 г.	
5 июня 2018 г.	
6 июня 2018 г.	
7 июня 2018 г.	
8 июня 2018 г.	
9 июня 2018 г.	
10 июня 2018 г.	
11 июня 2018 г.	
12 июня 2018 г.	
13 июня 2018 г.	
14 июня 2018 г.	
15 июня 2018 г.	
16 июня 2018 г.	
17 июня 2018 г.	
18 июня 2018 г.	
19 июня 2018 г.	
20 июня 2018 г.	
21 июня 2018 г.	
22 июня 2018 г.	
23 июня 2018 г.	
24 июня 2018 г.	
25 июня 2018 г.	
26 июня 2018 г.	
27 июня 2018 г.	
28 июня 2018 г.	
29 июня 2018 г.	
30 июня 2018 г.	
1 июля 2018 г.	
2 июля 2018 г.	
3 июля 2018 г.	
4 июля 2018 г.	
5 июля 2018 г.	
6 июля 2018 г.	
7 июля 2018 г.	
8 июля 2018 г.	
9 июля 2018 г.	
10 июля 2018 г.	
11 июля 2018 г.	
12 июля 2018 г.	
13 июля 2018 г.	
14 июля 2018 г.	
15 июля 2018 г.	



Итак, теперь мы, наконец, можем написать сами [формулы](#) расчета накопленной прибыли за текущий период на основе функций TOTALYTD, TOTALQTD и TOTALMTD:

```
Прибыль =  
SUMX (  
    'ПродажиЗатраты';  
    'ПродажиЗатраты'[Продажи] - 'ПродажиЗатраты'[Затраты]  
)
```

```
Итоговая Прибыль Год =  
TOTALYTD (  
    [Прибыль];  
    'спрКалендарь'[Date]  
)
```

```
Итоговая Прибыль Квартал =  
TOTALQTD (  
    [Прибыль];  
    'спрКалендарь'[Date]  
)
```

```
Итоговая Прибыль Месяц =
TOTALMTD (
    [Прибыль];
    'спрКалендарь'[Date]
)
```

Для начала, мы создали формулу меры [Прибыль] для того, чтобы, с одной стороны, просто разместить текущий размер прибыли в визуализации Power BI и, с другой стороны, чтобы в 3-х формулах накопленной прибыли не прописывать одни и те же строки несколько раз, тем самым сэкономив и время и место в редакторе формул.

Далее, собственно, расписали, достаточно простые формулы накопленной текущей прибыли по году, кварталу и месяцу. В первом параметре всех 3-х функций мы указали то выражение, которое нужно вычислить. В нашем примере это мера [Прибыль]. Фильтры в формулах мы не использовали просто потому, что в них нет надобности в этом практическом примере. В качестве последнего параметра указали набор непрерывных дат из связанной таблицы «Календарь», по которому эти функции и будут вычислять меру [Прибыль].

Итак, результатом выполнения формулы на основе DAX функции TOTALYTD, в Power BI будет следующая визуализация накопленной годовой прибыли:

Год	Продажи	Затраты	Прибыль	ИтоговаяПрибыльГод
2017	10 938 630	5 444 467	5 494 163	5 494 163
Кв. 1	3 002 232	1 336 278	1 665 954	1 665 954
Кв. 2	2 076 777	1 364 543	712 234	2 378 188
Кв. 3	1 955 855	1 365 782	590 073	2 968 261
Кв. 4	3 903 766	1 377 864	2 525 902	5 494 163
2018	15 447 621	5 539 021	9 908 600	9 908 600
Кв. 1	3 417 278	1 380 146	2 037 132	2 037 132
Кв. 2	2 632 577	1 365 004	1 267 573	3 304 705
Кв. 3	4 483 229	1 396 650	3 086 579	6 391 284
Кв. 4	4 914 537	1 397 221	3 517 316	9 908 600
Всего	26 386 251	10 983 488	15 402 763	9 908 600

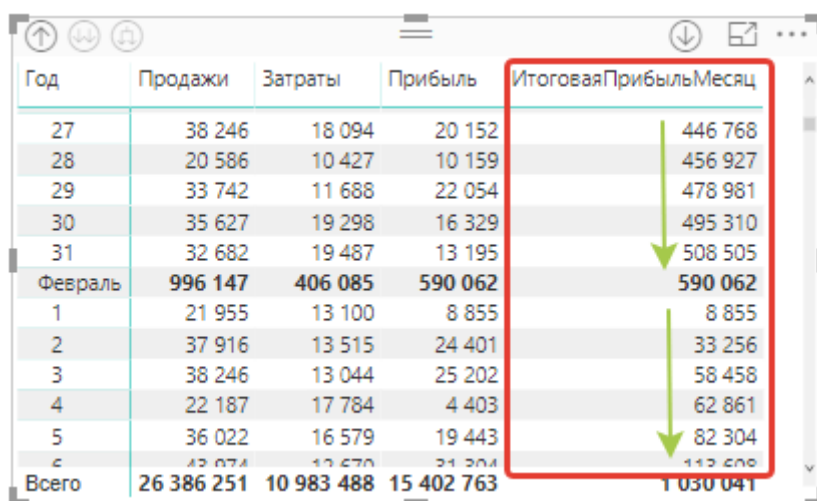
В данной визуализации мы видим, что, действительно, сумма прибыли накапливается общим итогом в течение года, и затем, как начинается новый год, итоговая годовая сумма прибыли обнуляется и накапливается вновь.

Результатом выполнения формулы на основе DAX функции TOTALQTD, в Power BI будет следующая визуализация накопленной поквартальной прибыли:

Год	Продажи	Затраты	Прибыль	ИтоговаяПрибыльКвартал
2017	10 938 630	5 444 467	5 494 163	2 525 902
Кв. 1	3 002 232	1 336 278	1 665 954	1 665 954
Январь	989 648	481 143	508 505	508 505
Февраль	996 147	406 085	590 062	1 098 567
Март	1 016 437	449 050	567 387	1 665 954
Кв. 2	2 076 777	1 364 543	712 234	712 234
Апрель	1 025 575	440 771	584 804	584 804
Май	590 166	465 437	124 729	709 533
Июнь	461 036	458 335	2 701	712 234
Кв. 3	1 055 855	1 365 782	500 073	500 073
Всего	26 386 251	10 983 488	15 402 763	3 517 316

В этой визуализации все также, как и в примере выше, то есть, в первом квартале прибыль накапливается общим итогом каждый месяц, далее, во втором квартале она обнуляется и вновь начинает накапливаться новая поквартальная прибыль.

Результатом выполнения формулы на основе DAX функции TOTALMTD, в Power BI будет следующая визуализация накопленной помесечной прибыли:



Год	Продажи	Затраты	Прибыль	ИтоговаяПрибыльМесяц
27	38 246	18 094	20 152	446 768
28	20 586	10 427	10 159	456 927
29	33 742	11 688	22 054	478 981
30	35 627	19 298	16 329	495 310
31	32 682	19 487	13 195	508 505
Февраль	996 147	406 085	590 062	590 062
1	21 955	13 100	8 855	8 855
2	37 916	13 515	24 401	33 256
3	38 246	13 044	25 202	58 458
4	22 187	17 784	4 403	62 861
5	36 022	16 579	19 443	82 304
6	42 074	17 670	24 404	106 708
Всего	26 386 251	10 983 488	15 402 763	1 030 041

И в этой визуализации, опять все также, как и в примерах выше — прибыль накапливается каждый день общим итогом в течение всего месяца. Затем, как начинается новый месяц, общий итог обнуляется и прибыль начинает накапливать вновь.

## DAX функции

### CLOSINGBALANCEYEAR,

### CLOSINGBALANCEQUARTER И

### CLOSINGBALANCEMONTH

CLOSINGBALANCEYEAR () — вычисляет заданное выражение на последнюю дату года текущего контекста.

CLOSINGBALANCEQUARTER () — вычисляет заданное выражение на последнюю дату квартала текущего контекста.

CLOSINGBALANCEMONTH () — вычисляет заданное выражение на последнюю дату месяца текущего контекста.

Синтаксис:

CLOSINGBALANCEYEAR (Выражение; [Дата]; Фильтр; "Конец Года")

CLOSINGBALANCEQUARTER (Выражение; [Дата]; Фильтр)

CLOSINGBALANCEMONTH (Выражение; [Дата]; Фильтр)

Где:

- Выражение — выражение, возвращающее единственное скалярное значение
- [Дата] — столбец из дат или выражений, возвращающих даты
- Фильтр — (необязательный параметр) необходимые фильтры для вычисления выражения

«Конец Года» — (необязательный параметр) текстовая дата, записанная в виде «01/06» («день/месяц»). Определяет дату окончания года (по умолчанию «31/12»)

Рассмотрим пример формул на основе [DAX функций](#) категории time intelligence CLOSINGBALANCEYEAR и CLOSINGBALANCEQUARTER.

В модели данных [Power BI Desktop](#) имеется исходная таблица «Сальдо счета», в которой на последнюю дату каждого месяца (с января 2018 по ноябрь 2018) прописан остаток баланса на счете организации. При чем, последняя дата баланса 20 ноября 2018 года, то есть, месяц еще не закончился и, соответственно, 4 квартал и год, также, еще не закончены:

Месяц	БалансСчета
31 января 2018 г.	357 800
28 февраля 2018 г.	200 020
31 марта 2018 г.	67 439
30 апреля 2018 г.	730 052
31 мая 2018 г.	1 004 358
30 июня 2018 г.	799 325
31 июля 2018 г.	824 992
31 августа 2018 г.	1 005 385
30 сентября 2018 г.	1 048 325
31 октября 2018 г.	900 352
20 ноября 2018 г.	1 034 083

Задача — создать отчет в Power BI, который бы показывал остаток (сальдо) средств на счете организации на конец каждого квартала и года, а также, на последнее текущее число (в данном примере, последнее текущее число — 20 ноября 2018).

Для этого воспользуемся нашими DAX функциями, которые мы рассматриваем в этой статье — CLOSINGBALANCEYEAR, CLOSINGBALANCEQUARTER, так как они способны вычислить выражение на последнюю известную дату года и квартала. Составим на основе них соответствующие [формулы](#) расчета сальдо баланса счета на последние даты периодов (года и квартала):

```
Сальдо Год =
CLOSINGBALANCEYEAR (
    SUM ('СальдоСчета'[БалансСчета]);
    'СальдоСчета'[Месяц]
)
```



```
Сальдо Квартал =  
CLOSINGBALANCEQUARTER (  
    SUM ('СальдоСчета'[БалансСчета]);  
    'СальдоСчета'[Месяц]  
)
```

В первом параметре этих формул мы прописали само выражение, которое нам нужно вычислить — в данном случае, это сумма по столбцу [Баланс Счета], которую мы вычислили при помощи DAX функции [SUM](#). Но, на самом деле, здесь сумма не высчитывается по всему столбцу, а только по одной его строчке — соответствующей последней известной дате квартала и года. И это ограничение на выражение накладывают уже сами функции CLOSINGBALANCEYEAR, CLOSINGBALANCEQUARTER на основе тех дат, которые поданы во втором параметре этих функций.

И именно потому, что функции накладывают это ограничение, сальдо 4 квартала и сальдо всего года, вычисляемые каждый текущий день, выдают нам итоговое сальдо баланса за эти периоды на основании последней известной даты — 20 ноября.

В итоге, в Power BI мы можем наблюдать следующие визуализации сальдо баланса счета на конец квартала и года:

Год	БалансСчета	СальдоКвартал	СальдоГод
2018	7 972 131	1 034 083	1 034 083
Январь	357 800		
Февраль	200 020		
Март	67 439	67 439	
Апрель	730 052		
Май	1 004 358		
Июнь	799 325	799 325	
Июль	824 992		
Август	1 005 385		
Сентябрь	1 048 325	1 048 325	
Октябрь	900 352		
Ноябрь	1 034 083	1 034 083	1 034 083
Всего	7 972 131	1 034 083	1 034 083

Теперь, рассмотрим еще один пример формулы на основе DAX функции CLOSINGBALANCEMONTH.

В модели данных Power BI имеется таблица «Товары», в которой отображены по определенным датам остатки товара на складе за февраль, март и начало апреля:

Дата	ОстатокТовара
12 февраля 2018 г.	38
16 февраля 2018 г.	17
23 февраля 2018 г.	6
26 февраля 2018 г.	0
2 марта 2018 г.	18
8 марта 2018 г.	46
12 марта 2018 г.	30
17 марта 2018 г.	23
26 марта 2018 г.	3
2 апреля 2018 г.	0
3 апреля 2018 г.	19

Задача — создать отчет в Power BI, показывающий в реальном времени текущие остатки товара за месяц, а также, остатки товара на конец прошлых месяцев.

По сути, формула на основе функции CLOSINGBALANCEMONTH [языка DAX](#) будет идентична тем формулам, которые мы рассматривали выше:

```
Остаток На Конец Месяца =  
CLOSINGBALANCEMONTH (  
    SUM ('Товары'[ОстатокТовара]);  
    'Товары'[Дата]  
)
```

Результатом выполнения этой формулы на основе CLOSINGBALANCEMONTH, в Power BI Desktop будет следующая визуализация остатков товара на складе на конец месяца:

ОстатокНаКонецМесяца =  
CLOSINGBALANCEMONTH (  
 SUM ('Товары'[ОстатокТовара]);  
 'Товары'[Дата]  
)

Год	ОстатокТовара	ОстатокНаКонецМесяца
2018		
Февраль		
12	38	
16	17	
23	6	
26	0	0
Март		
2	18	
8	46	
12	30	
17	23	
26	3	3
Апрель		
2	0	
3	19	19

## DAX функции

### OPENINGBALANCEYEAR,

### OPENINGBALANCEQUARTER И

### OPENINGBALANCEMONTH

OPENINGBALANCEYEAR () — вычисляет заданное выражение на первую дату года текущего контекста.

OPENINGBALANCEQUARTER () — вычисляет заданное выражение на первую дату квартала текущего контекста.

OPENINGBALANCEMONTH () — вычисляет заданное выражение на первую дату месяца текущего контекста.

Синтаксис:

OPENINGBALANCEYEAR (Выражение; [Дата]; Фильтр; "Конец Года")

OPENINGBALANCEQUARTER (Выражение; [Дата]; Фильтр)

OPENINGBALANCEMONTH (Выражение; [Дата]; Фильтр)

Где:

- Выражение — выражение, возвращающее единственное скалярное значение
- [Дата] — столбец из дат или выражений, возвращающих даты
- Фильтр — (необязательный параметр) необходимые фильтры для вычисления выражения
- «Конец Года» — (необязательный параметр) текстовая дата, записанная в виде «01/06» («день/месяц»). Определяет дату окончания года (по умолчанию «31/12»)

Пример формул разбирается выше в предыдущей схожей группе функций.

## DAX функции ENDOFYEAR, ENDOFQUARTER И ENDOFMONTH

+

## DAX функции STARTOFYEAR, STARTOFQUARTER, STARTOFMONTH

ENDOFYEAR () — возвращает последнюю известную дату года в рамках текущего контекста.

ENDOFQUARTER () — возвращает последнюю известную дату квартала в рамках текущего контекста.

ENDOFMONTH () — возвращает последнюю известную дату месяца в рамках текущего контекста.

Синтаксис:

ENDOFYEAR ([Дата]; "Конец Года")

ENDOFQUARTER ([Дата])

ENDOFMONTH ([Дата])

Где:

- [Дата] — столбец из дат или выражений, возвращающих даты
- «Конец Года» — (необязательный параметр) текстовая дата, записанная в виде «01/06» («день/месяц»). Определяет дату окончания года (по умолчанию «31/12»)

STARTOFYEAR () — возвращает первую известную дату года в рамках текущего контекста.

STARTOFQUARTER () — возвращает первую известную дату квартала в рамках текущего контекста.

STARTOFMONTH () — возвращает первую известную дату месяца в рамках текущего контекста.

Синтаксис:

STARTOFYEAR ([Дата]; "Конец Года")

STARTOFQUARTER ([Дата])

STARTOFMONTH ([Дата])

Где:

- [Дата] — столбец из дат или выражений, возвращающих даты
- «Конец Года» — (необязательный параметр) текстовая дата, записанная в виде «01/06» («день/месяц»). Определяет дату окончания года (по умолчанию «31/12»)

### Пример формул на основе функций группы ENDOF и STARTOF (YEAR, QUARTER и MONTH)

В модели данных [Power BI Desktop](#) имеется исходная таблица «Банковский Счет», в которой расположена информация о сальдо баланса банковского счета организации за каждый день с 15 января по 7 апреля 2018 года:

Дата	БалансСчета
15 января 2018 г.	549480
16 января 2018 г.	528180
17 января 2018 г.	444110
18 января 2018 г.	374770
4 апреля 2018 г.	259260
5 апреля 2018 г.	284250
6 апреля 2018 г.	602190
7 апреля 2018 г.	560510

Сперва, для объяснения сути работы всех DAX функций ENDOFYEAR, ENDOFQUARTER, ENDOFMONTH и STARTOFYEAR, STARTOFQUARTER, STARTOFMONTH, рассмотрим простой пример их работы.

Создадим в Power BI Desktop во вкладке «Моделирование» в исходной таблице «Банковский счет» 2 временных вычисляемых столбца на основе следующих [формул](#) с использованием функций STARTOFMONTH и ENDOFMONTH:

Дата Начала Месяца = STARTOFMONTH ('БанковскийСчет'[Дата])

Дата Конца Месяца = ENDOFMONTH ('БанковскийСчет'[Дата])

Результатом выполнения этих формул, в исходной таблице у нас появились 2 вычисляемых столбца:

ДатаКонцаМесяца = ENDOFMONTH ('БанковскийСчет'[Дата])			
Дата	БалансСчета	ДатаНачалаМесяца	ДатаКонцаМесяца
29 января 2018 г.	354150	15 января 2018 г.	31 января 2018 г.
30 января 2018 г.	200980	15 января 2018 г.	31 января 2018 г.
31 января 2018 г.	541020	15 января 2018 г.	31 января 2018 г.
1 февраля 2018 г.	526250	1 февраля 2018 г.	28 февраля 2018 г.
2 февраля 2018 г.	341080	1 февраля 2018 г.	28 февраля 2018 г.
3 февраля 2018 г.	409310	1 февраля 2018 г.	28 февраля 2018 г.
4 февраля 2018 г.	373060	1 февраля 2018 г.	28 февраля 2018 г.

Из этих созданных столбцов мы можем увидеть, что для каждой даты в столбце [Дата] соответствует реально существующая дата начала или конца этого месяца, то есть, для 30 января — дата начала месяца 15 января (так как баланс счета у нас начинается с 15 января) и дата конца месяца — 31 января.

То же самое и для других дат — для 3 февраля, дата начала месяца 1 февраля, а дата конца месяца 28 февраля.

В этом и заключается вся суть DAX функций ENDOFYEAR, ENDOFQUARTER, ENDOFMONTH и STARTOFYEAR, STARTOFQUARTER,

STARTOFMONTH. Но, сами по себе эти функции весьма бесполезны и их нужно использовать совместно [с другими функциями языка DAX](#).

Рассмотрим пример далее, с более сложными формулами.

Создадим в Power BI три меры, рассчитывающие значение сальдо баланса банковского счета организации на начало 3 периодов — на начало месяца, квартала и года. Код формул будет таким:

БалансНаНачалоМесяца =

```
CALCULATE (  
    SUM ('БанковскийСчет'[БалансСчета]);  
    STARTOFMONTH ('БанковскийСчет'[Дата])  
)
```

БалансНаНачалоКвартала =

```
CALCULATE (  
    SUM ('БанковскийСчет'[БалансСчета]);  
    STARTOFQUARTER ('БанковскийСчет'[Дата])  
)
```

БалансНаНачалоГода =

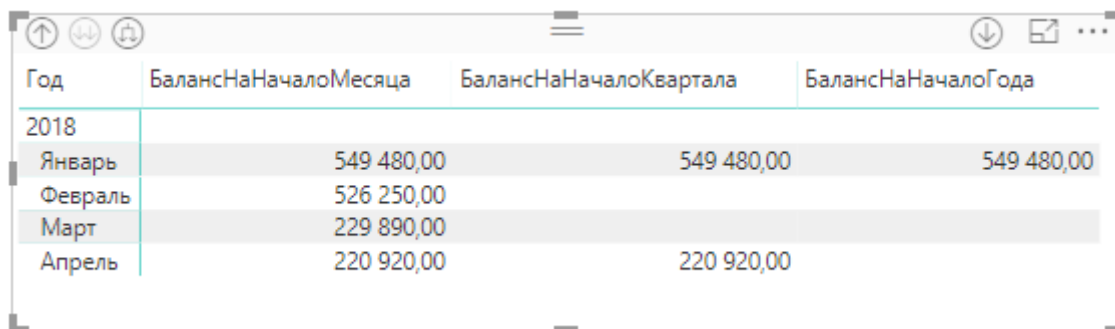
```
CALCULATE (  
    SUM ('БанковскийСчет'[БалансСчета]);  
    STARTOFYEAR ('БанковскийСчет'[Дата])  
)
```

Во всех этих формулах мы используем специальную DAX функцию — CALCULATE (подробнее об этой функции Вы можете прочитать [в этой статье](#)). Эта функция позволяет нам вычислить сумму, рассчитанную функцией [SUM](#), по столбцу [Баланс Счета], но не по всем строкам этого столбца, а только по одной единственной строке, которую возвратят каждая в своей формуле функции STARTOFMONTH, STARTOFQUARTER и STARTOFYEAR, находящиеся во втором



параметре CALCULATE в качестве фильтров. А если быть точнее, то по строкам начала месяца, квартала и года в рамках текущего контекста.

В итоге, [в отчетах Power BI](#), на основе выше приведенных формул, мы можем наблюдать следующую визуализацию:



The screenshot shows a table visualization in Power BI. The table has four columns: 'Год' (Year), 'БалансНаНачалоМесяца' (Balance at the start of the month), 'БалансНаНачалоКвартала' (Balance at the start of the quarter), and 'БалансНаНачалоГода' (Balance at the start of the year). The data is filtered for the year 2018. The rows represent the months of the year: Январь (January), Февраль (February), Март (March), and Апрель (April). The balance values are in Russian rubles (rub).

Год	БалансНаНачалоМесяца	БалансНаНачалоКвартала	БалансНаНачалоГода
2018			
Январь	549 480,00	549 480,00	549 480,00
Февраль	526 250,00		
Март	229 890,00		
Апрель	220 920,00	220 920,00	

Из этой визуализации мы уже можем наблюдать практическую ценность формул на основе DAX функций STARTOFYEAR, STARTOFQUARTER, STARTOFMONTH. То есть, из так называемой, таблицы ежедневных снимков баланса счета, мы получили точные значения сальдо баланса на начало месяца, квартала и года.

# DAX функции

## FIRSTDATE И LASTDATE

FIRSTDATE () — возвращает первую известную дату для указанного столбца дат в текущем контексте. LASTDATE () — возвращает последнюю известную дату для указанного столбца дат в текущем контексте.

Иногда я встречаю, что пользователи [языка DAX](#) в Power BI или Power Pivot пишут данные функции в два отдельных слова: FIRST DATE или LAST DATE, что, естественно, неправильно.

Синтаксис:

FIRSTDATE ([Дата])

LASTDATE ([Дата])

### Пример формул на основе DAX функций FIRSTDATE и LASTDATE

Разберем формулы с участием DAX функций FIRSTDATE и LASTDATE на основе примера продажи некого товара.

В Power BI имеется исходная таблица «Продажи товара», где содержится информация по дате сделки продажи:

ДатаСделки
17 января 2018 г.
23 января 2018 г.
28 января 2018 г.

Для тех или иных задач, для построения каких-то исследовательских [отчетов в Power BI](#), на основании имеющихся данных по продажам, нам требуется получить даты первой и последней сделки по продаже товара. Как Вы понимаете, для этих целей нам подойдут рассматриваемые в этой статье time intelligence

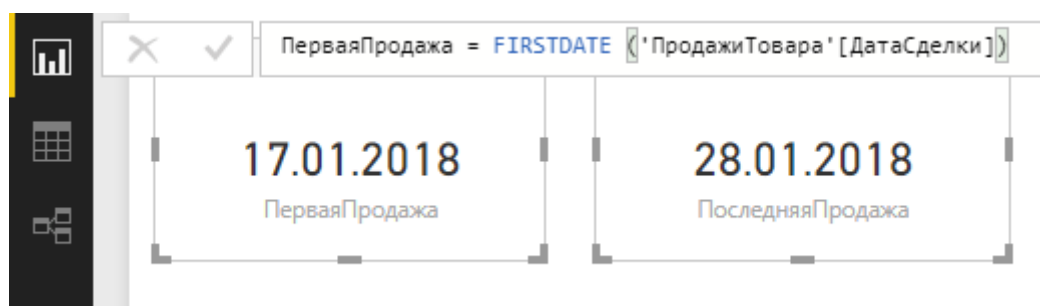
функции FIRSTDATE и LASTDATE, так как они, как раз таки, и возвратят нам эти даты первой и последней продажи.

Итак, составим соответствующие [формулы](#) мер в [Power BI Desktop](#):

Первая Продажа = FIRSTDATE ('ПродажиТовара'[ДатаСделки])

Последняя Продажа = LASTDATE ('ПродажиТовара'[ДатаСделки])

И в отчетах Power BI мы можем пронаблюдать работу формул этих мер:



Теперь, больше не для того, чтобы понять работу функций FIRSTDATE и LASTDATE (они и так простые и с ними все понятно), а для тренировки составления формул на языке DAX, рассмотрим пример с формулой, несколько сложнее предыдущих.

Итак, нам требуется составить такой код формулы, чтобы получить дату последней сделки, но только для тех продаж, которые были совершены до 25 января.

То есть, в нашу исходную прошлую формулу нужно вставить каким-то образом фильтр, который отфильтрует все даты продаж до 25 января. Код формулы будет таким:

```
Последняя Продажа =  
LASTDATE (  
    FILTER (  
        ALL ('ПродажиТовара'[ДатаСделки]);  
        'ПродажиТовара'[ДатаСделки] < DATE (2018; 01; 25)  
    )  
)
```

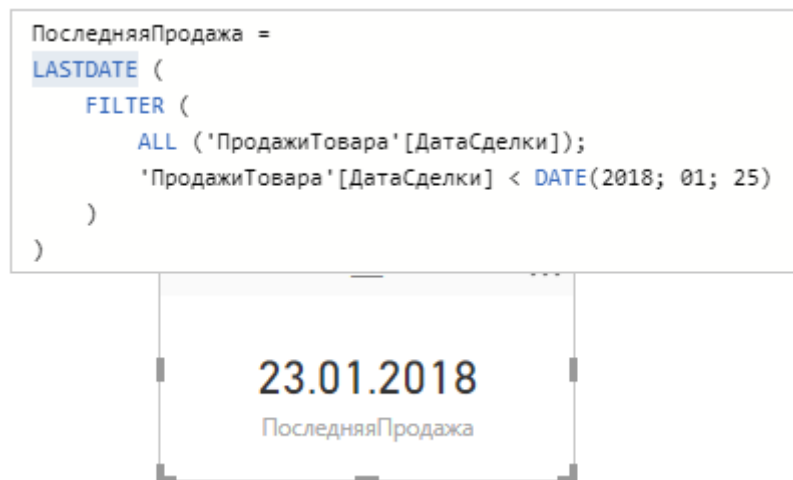
Где, в качестве параметра столбца с датами в функции LASTDATE мы вставили функцию [FILTER](#), которая и отфильтровывает даты сделок до 25 января следующим условием:

```
'ПродажиТовара'[ДатаСделки] < DATE (2018; 01; 25)
```

Для сравнения в условии даты сделок с 25 января мы воспользовались еще одной DAX функцией [DATE](#), которая позволила в этом условии записать саму дату.

Также, пришлось использовать еще одну DAX функцию — [ALL](#). Потому как, функция FILTER, если в ее первый параметр вставить просто таблицу «Продажи Товара», то она возвратит отфильтрованную таблицу, что не подойдет для функции LASTDATE, так как в параметре LASTDATE — должен быть столбец, а не таблица. И функция ALL, внутрь которой мы вставили ссылку на столбец [Дата Сделки], в результате возвратила столбец в виде таблицы. Что подошло как и для FILTER (так как в ней нужна ссылка именно на таблицу), так и для LASTDATE (так как для нее нужна ссылка на столбец).

Итог работы этой формулы мы можем наблюдать в отчетах Power BI:



Как мы видим из этой визуализации в Power BI, дата последней продажи у нас вывелась не 28 января, как в первом примере, а 23 января.

## DAX функция EOMONTH

EOMONTH () — возвращает дату конца месяца с учетом прописанного отступа в месяцах от указанной в параметрах даты.

Синтаксис:

EOMONTH (Дата; Отступ)

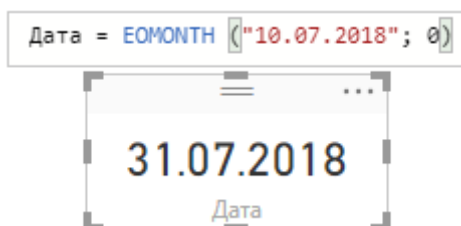
Где:

- Дата — дата, записанная в текстовом или datetime форматах
- Отступ — отступ в месяцах (отрицательное значение — отступ до текущей даты, положительное значение — после текущей даты)

Примеры [формул](#) на основе DAX функции EOMONTH.

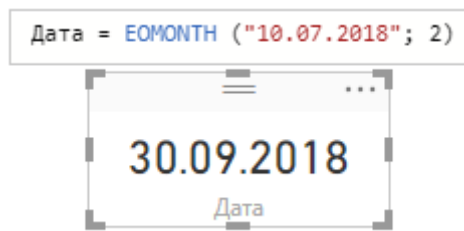
Дата = EOMONTH ("10.07.2018"; 0)

В формуле во втором параметре прописано значение 0, то есть, EOMONTH в [Power BI Desktop](#) выведет конец месяца, по той же самой дате, которая указана в первом параметре:



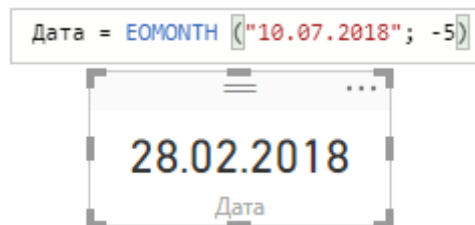
Дата = EOMONTH ("10.07.2018"; 2)

Так как в этой формуле отступ указан в 2 месяца положительным числом, то в итоге, функция возвратит конец сентября, потому что в первом параметре указан июль:



Дата = EOMONTH ("10.07.2018"; -5)

Эта формула, наоборот, возвратит конец февраля, так как отступ в месяцах указан отрицательным числом -5:



## DAX функция EDATE

EDATE () — возвращает дату, прописанную в параметрах и перемещенную на указанное количество месяцев до прописанной даты или после.

Синтаксис:

EDATE (Дата; Отступ)

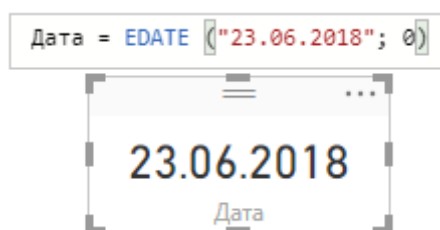
Где:

- Дата — дата, записанная в текстовом или datetime форматах
- Отступ — отступ в месяцах (отрицательное значение — отступ до текущей даты, положительное значение — после текущей даты)

Примеры формул на основе DAX функции EDATE.

Дата = EDATE ("23.06.2018"; 0)

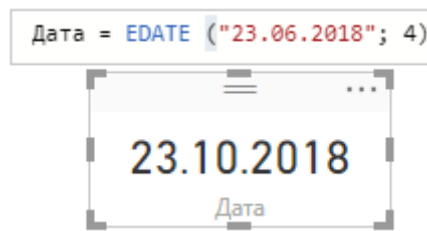
Данная формула возвратит то же самое число, которое указано в первом параметре EDATE, так как отступ равен значению 0:



Дата = EDATE ("23.06.2018"; 4)

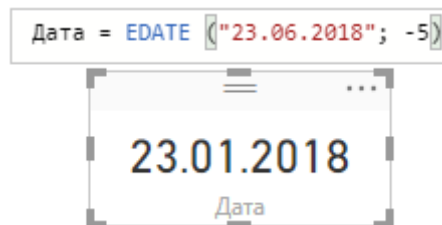
Эта формула возвратит 23 октября, так как именно это число получится при положительном отступе в 4 месяца от 23 июня, указанным в первом параметре:





Дата = EDATE ("23.06.2018"; -5)

Ну и, при отрицательном отступе в 5 месяцев, DAX функция EDATE возвратит в мере в Power BI Desktop 23 января:



# 7. ФУНКЦИИ ФИЛЬТРОВ В DAX

# DAX функция CALCULATE

CALCULATE () — вычисляет выражение, измененное внутренними фильтрами.

Синтаксис:

CALCULATE (Выражение; Фильтр 1; Фильтр 2; ...; Фильтр N)

Где:

- Выражение — то выражение, которое нужно вычислить (обязательный параметр для CALCULATE, без него эта функция работать не будет)
- Фильтр — условия фильтров (необязательный параметр, количество фильтров может быть от 0 до многих и все они сочетаются в режиме «и»)

Фильтры в CALCULATE не могут ссылаться на различные меры или вложенные функции CALCULATE. При этом, в условиях фильтров могут использоваться какие-либо другие функции DAX, вычисляющие одно скалярное значение или создающие запрос уточнения одного скалярного значения.

Если говорить проще про функцию CALCULATE, то она при помощи своих условий фильтров способна дополнить, заменить или полностью удалить все предыдущие наложенные фильтры на выражение, указанное в первом параметре.

## Пример использования функции CALCULATE

Для понимания функционирования и работы [DAX формул](#) на основе CALCULATE, разберем примеры ее работы в программе [Power BI Desktop](#).

Имеется исходная таблица «Общие Продажи»:

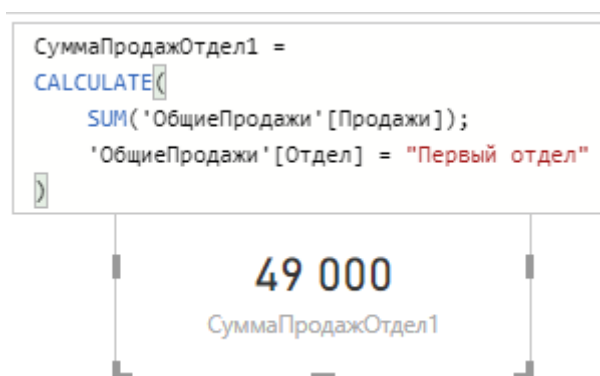
Менеджер	Продажи	Отдел
Смирнов	18000	Первый отдел
Сидоров	31000	Первый отдел
Колесников	16500	Второй отдел
Василькова	24000	Второй отдел
Соколова	38000	Третий отдел

Задача — вычислить сумму продаж только по первому отделу. Создадим в Power BI Desktop во вкладке «Моделирование» меру по следующему коду формулы с участием функций [SUM](#) и CALCULATE:

```
Сумма Продаж Отдел 1 =  
CALCULATE(  
    SUM('ОбщиеПродажи'[Продажи]);  
    'ОбщиеПродажи'[Отдел] = "Первый отдел"  
)
```

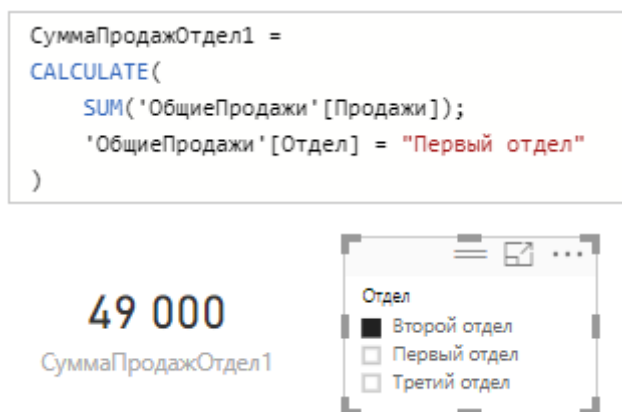
Где, SUM вычислит сумму всех продаж, находясь под созданным функцией CALCULATE фильтром из второго параметра.

Результатом выполнения этого кода формулы будет сумма продаж по первому отделу, равная 49000:



Причем созданный функцией CALCULATE фильтр, будет заменять любой другой фильтр, который может наложить пользователь во вкладке «Отчеты» в Power BI.

В этом можно убедиться на примере ниже, где на срезе «Отделы» установлен пользовательский фильтр «Второй отдел», но CALCULATE полностью заменяет его своим условием и формула высчитывает сумму опять же только по первому отделу, то есть сумма продаж равна 49000:



В качестве выражения в первом параметре CALCULATE можно рассчитывать очень многие формулы и вставлять туда какие-либо другие функции DAX, которые Вам понадобятся в конкретной ситуации.

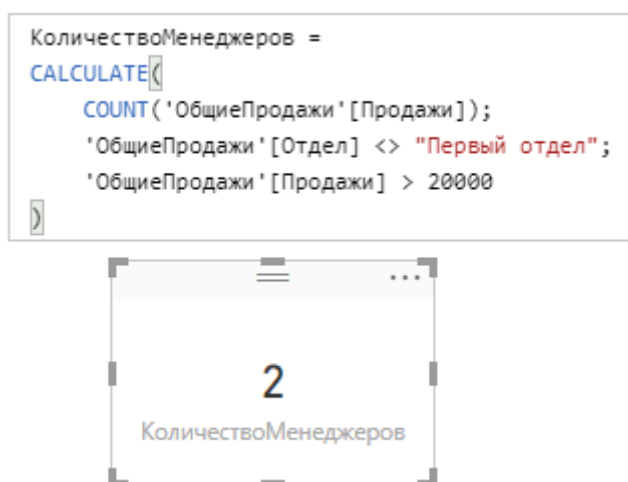
Также, при этом, создавая хитрые переплетения фильтров из второго и последующих параметров CALCULATE, можно формировать очень сложные формулы и необычные вычисления в Power BI и Power Pivot.

В рамках этой ознакомительной статьи сложные примеры мы пока рассматривать не будем, но для закрепления материала, посмотрим еще один небольшой пример формулы, несколько сложнее предыдущего.

Рассчитаем при помощи функции [COUNT](#) количество менеджеров, у которых сумма продаж больше 20000. При условии, что менеджеры не должны принадлежать первому отделу. Условия в этой формуле мы зададим при помощи фильтров CALCULATE:

```
Количество Менеджеров =  
CALCULATE (  
    COUNT ('ОбщиеПродажи'[Продажи]);  
    'ОбщиеПродажи'[Отдел] <> "Первый отдел";  
    'ОбщиеПродажи'[Продажи] > 20000  
)
```

В этой формуле COUNT считает количество менеджеров, находясь под действием сразу двух фильтров от CALCULATE, где столбец [Отдел] не равен значению «Первый отдел» и одновременно с этим в столбце [Продажи] все значения должны быть более 20000. Как итог, формула рассчитает количество менеджеров, равное 2:



Теперь давайте рассмотрим собрата CALCULATE — функцию CALCULATETABLE, которая, как я писал в самом начале по всем своим свойствам полностью аналогична первой функции и различия между ними в том, что первая функция работает с единичными скалярными значениями, а вторая — с табличными выражениями.

# DAX функция CALCULATETABLE

CALCULATETABLE () — вычисляет табличное выражение, измененное внутренними фильтрами.

Синтаксис:

CALCULATETABLE (Табличное выражение; Фильтр 1; Фильтр 2; ...; Фильтр N)

Где:

- Табличное выражение — то табличное выражение, которое нужно вычислить (обязательный параметр для CALCULATETABLE, без него эта функция работать не будет)
- Фильтр — условия фильтров (необязательный параметр, количество фильтров может быть от 0 до многих и все они сочетаются в режиме «и»)

Функцию CALCULATETABLE нельзя путать с функцией фильтра таблиц в DAX — [FILTER](#). Эта функция просто фильтрует таблицы по заданным условиям фильтра. А CALCULATETABLE, в свою очередь, не просто фильтрует, а заменяет или удаляет фильтры, что в итоге дает совершенно другой результат.

Рассмотрим небольшой пример формулы на основе CALCULATETABLE. В качестве исходной таблицы возьмем ту же таблицу по продажам менеджеров, которую брали в примерах выше:

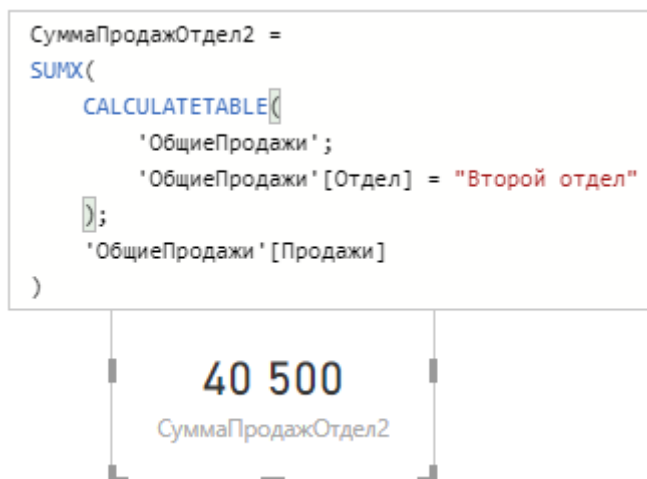
Менеджер	Продажи	Отдел
Смирнов	18000	Первый отдел
Сидоров	31000	Первый отдел
Колесников	16500	Второй отдел
Василькова	24000	Второй отдел
Соколова	38000	Третий отдел

В этом примере также рассчитаем сумму продаж, но только по второму отделу. В данном случае фильтры мы будем устанавливать функцией CALCULATETABLE, а сумму рассчитаем при помощи второй

функции суммирования в DAX — SUMX (подробный разбор функции SUMX Вы можете прочитать [в этой статье](#)):

```
Сумма Продаж Отдел 2 =  
SUMX (  
    CALCULATETABLE (  
        'ОбщиеПродажи';  
        'ОбщиеПродажи'[Отдел] = "Второй отдел"  
    );  
    'ОбщиеПродажи'[Продажи]  
)
```

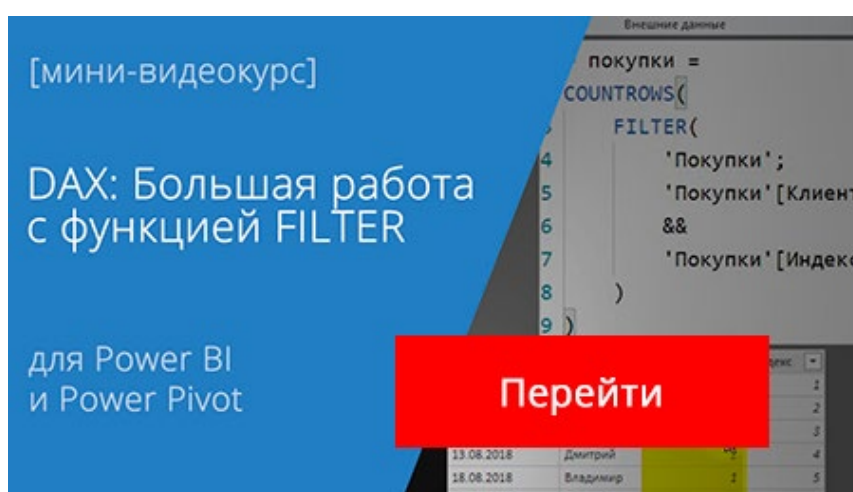
В данной формуле CALCULATETABLE возвращает таблицу, отфильтрованную по условию «Второй отдел», причем этот фильтр будет заменять любой другой пользовательский фильтр, наложенный на этот же столбец [Отдел]. А SUMX рассчитает по этой отфильтрованной таблице сумму по столбцу [Продажи]. В итоге, получится сумма продаж именно по второму отделу, равная 40500:





# DAX функция FILTER

DAX [функция](#) FILTER () — возвращает таблицу, фильтруя исходную таблицу по заданным в параметрах фильтрам. Фильтрация производится по строкам, то есть, возвращаются все столбцы исходной таблицы, а строки только те, которые удовлетворяют условию фильтра.



Синтаксис:

**FILTER** ('Таблица'; Фильтр)

Где:

- 'Таблица' — исходная таблица или табличное выражение, которую необходимо отфильтровать
- Фильтр — логическое выражение, которое сравнивается с каждой строкой таблицы, указанной в первом параметре

Обычно, данную функцию не используют в самостоятельной работе для простого создания отфильтрованной вычисляемой таблицы, хотя в Power BI это возможно.

Так как эта функция в итоге возвращает таблицу, то в основном FILTER () используют внутри самих формул DAX в каких-то промежуточных решениях, например, в составе параметров других функций, которые требуют для своей работы таблицу.

## Пример формулы DAX на основе работы функции FILTER

Разберем работу FILTER на простейшем примере: у нас имеется исходная таблица 'Общие Продажи'

Менеджер	Продажи	Отдел
Смирнов	18000	Первый отдел
Сидоров	31000	Первый отдел
Колесников	16500	Второй отдел
Василькова	24000	Второй отдел
Соколова	38000	Третий отдел

Задача состоит в следующем — нужно создать новую таблицу, которая должна быть уже отфильтрована по условию «Показать только первый отдел». Для решения этой задачи как раз-таки очень хорошо подойдет FILTER.

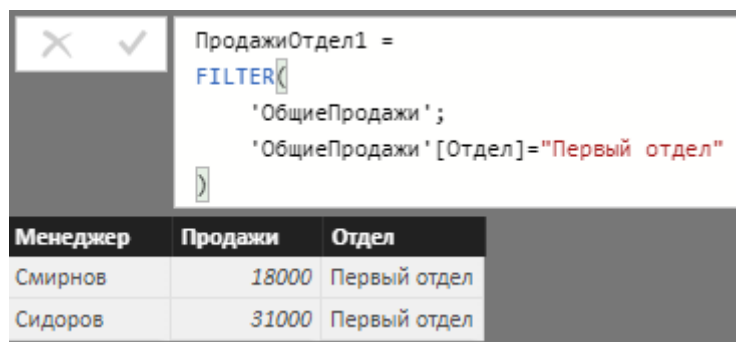
Создадим в [Power BI Desktop](#) вычисляемую таблицу, нажав кнопку «Создать таблицу» во вкладке «Моделирование». Итак, код будет таким:

```
ПродажиОтдел1 =  
FILTER ('ОбщиеПродажи'; 'ОбщиеПродажи'[Отдел] = "Первый отдел")
```

Где, в качестве первого параметра мы указали исходную таблицу 'ОбщиеПродажи', которую нужно отфильтровать. А в качестве второго параметра само условие, по которому значение из текущей строки столбца [Отдел] в этой таблице проверяется на соответствие условию «Первый отдел».

Та строка, которая удовлетворяет этому условию, в итоге возвращается функцией FILTER, а те строки, которые не удовлетворяют — пропускаются.

Итог работы этой формулы, следующий:



The screenshot shows the DAX formula bar with the following formula: `ПродажиОтдел1 = FILTER('ОбщиеПродажи', 'ОбщиеПродажи'[Отдел]="Первый отдел")`. Below the formula bar, a table is displayed with the following data:

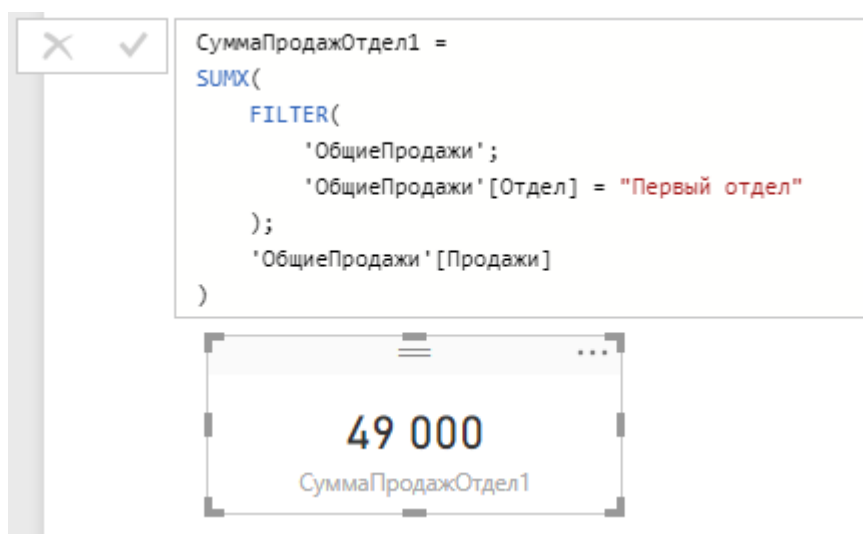
Менеджер	Продажи	Отдел
Смирнов	18000	Первый отдел
Сидоров	31000	Первый отдел

Как я уже писал выше, сама по себе данная функция используется редко, в основном ее используют в составе формул в одновременной работе с другими функциями.

Как пример, если потребуется создать меру, которая уже сразу должна выдать отфильтрованную в самом коде DAX сумму продаж, то в этом случае FILTER уже будет использоваться в качестве входящего параметра в другой функции:

```
СуммаПродажОтдел1 =
SUMX(
    FILTER(
        'ОбщиеПродажи';
        'ОбщиеПродажи'[Отдел] = "Первый отдел"
    );
    'ОбщиеПродажи'[Продажи]
)
```

И, как итог, формула выдает сумму продаж только по первому отделу:



В данном примере FILTER возвращает отфильтрованную таблицу 'ОбщиеПродажи' в первый параметр функции [SUMX](#), которая, в свою очередь, уже по этой отфильтрованной таблице вычисляет сумму столбца [Продажи].

## Функция FILTER и несколько условий фильтров

На просторах Интернета я часто встречаю вопрос о том, как использовать в функции FILTER сразу несколько условий фильтров, ведь параметров для составления условия у нее всего один.

Вариантов здесь два:

1. Либо дополнительно в условии использовать функции [AND\(\)](#) или [OR\(\)](#), или операторы && (и) или || (или). Например, так (в этом примере я соединил два условия в одном при помощи DAX

оператора «или» || ):

ПродажиОтдел1 = FILTER( 'ОбщиеПродажи'; 'ОбщиеПродажи'[Отдел] = "Первый отдел"    'ОбщиеПродажи'[Отдел] = "Третий отдел" )		
Менеджер	Продажи	Отдел
Смирнов	18000	Первый отдел
Сидоров	31000	Первый отдел
Соколова	38000	Третий отдел

Или так (здесь мы объединили в FILTER несколько разных условий при помощи функции «и» AND):

ПродажиОтдел1 = FILTER( 'ОбщиеПродажи'; AND('ОбщиеПродажи'[Отдел] = "Первый отдел"; 'ОбщиеПродажи'[Продажи] > 30000) )		
Менеджер	Продажи	Отдел
Сидоров	31000	Первый отдел

2. Либо вложить FILTER саму в себя, проведя двойную фильтрацию. И, таким образом, мы сможем опять же объединить сразу несколько условий фильтров в рамках одной формулы:

ПродажиОтдел1 = FILTER( FILTER( 'ОбщиеПродажи'; 'ОбщиеПродажи'[Продажи] > 30000 ); 'ОбщиеПродажи'[Отдел] = "Первый отдел" )		
Менеджер	Продажи	Отдел
Сидоров	31000	Первый отдел

## DAX функция ALL

ALL () — возвращает полную исходную таблицу или столбец, игнорируя все, ранее наложенные на них, фильтры. Иначе говоря, удаляет все ранее наложенные на таблицу или столбец фильтры.

Синтаксис:

- **ALL ('Таблица')** — возвращает все строки исходной таблицы
- **ALL ([Столбец 1]; [Столбец 2]; ...; [Столбец N])** — возвращает столбец (столбцы) со всеми уникальными значениями исходного столбца (столбцов)

Исходя из определения синтаксиса функции ALL, который описан выше, можно сказать, что ALL работает в двух разных режимах.

Когда в параметре указана просто таблица — то возвращается исходная таблица в том виде, какая она есть на самом деле.

Когда в параметре функции ALL указан столбец (столбцов может быть от 1 до нескольких), то возвращается уже не полностью исходный столбец (его копия), а только уникальные значения исходного столбца, но с учетом того, что ранее наложенные фильтры удаляются.

Давайте эти моменты мы с Вами закрепим на практическом примере.

Пример будем рассматривать в [Power BI](#), так как в этой программе имеется возможность создавать вычисляемые таблицы. А это для нашего примера важно, потому что пример получится наглядным.

В Excel (Power Pivot), к сожалению, вычисляемые таблицы в модели данных создавать нельзя, поэтому функция ALL там работает «виртуально». То есть, она возвращает таблицы и столбцы в своей виртуальной памяти и использует их только во время самих вычислений.

В то время, как в Power BI, при помощи функции ALL можно создать полноценные физические таблицы и столбцы в самой модели данных.

Итак, рассматриваем пример того, как функция ALL работает, когда в ее параметрах указана таблица и как она работает, когда там указан столбец.

В [Power BI Desktop](#) имеется исходная таблица «Заявки»:

Менеджер	Затраты	Сумма
Петров	20	100
Сидоров	50	150
Воснецова	30	90
Сидоров	5	30
Скворцов	50	130
Петров	60	200
Сидоров	75	250
Воснецова	20	101
Скворцов	5	20
Петров	100	350

Создадим на основе этой таблицы и функции ALL новую вычисляемую таблицу по следующей [DAX формуле](#):

Таблица = ALL ('Заявки')

Так как на таблицу никаких фильтров мы не накладывали, то функции ALL удалять нечего. И, собственно, в любом случае, она возвратила полную исходную таблицу:

Таблица = ALL('Заявки')		
Менеджер	Затраты	Сумма
Петров	20	100
Сидоров	50	150
Воснецова	30	90
Сидоров	5	30
Скворцов	50	130
Петров	60	200
Сидоров	75	250
Воснецова	20	101
Скворцов	5	20
Петров	100	350

Теперь изменим наш DAX код, а именно, вместо таблицы в параметрах функции ALL укажем столбец:

Таблица = ALL ('Заявки'[Менеджер])

Опять же, так как никаких фильтров до этого не было, то функции ALL удалять нечего. И она возвратила исходный столбец. Но не все его значения, а как я это писал выше, только уникальные значения этого столбца:

Таблица = ALL('Заявки'[Менеджер])	
Менеджер	
Петров	
Сидоров	
Воснецова	
Скворцов	

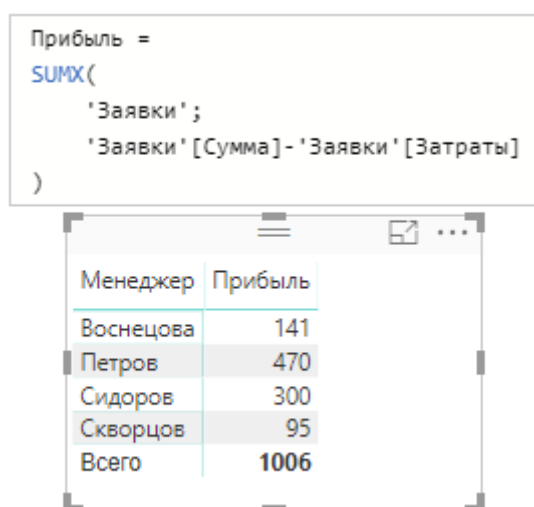
Итак, хорошо, с работой функции ALL [языка DAX](#) мы разобрались, но все же, зачем она нужна в реальной жизни? Теперь, давайте рассмотрим настоящий жизненный пример работы функции ALL.



## Пример формулы с участием DAX функции ALL

Рассматриваем пример на основе все той же исходной таблицы «Заявки».

Во вкладке «Отчеты» в Power BI Desktop у меня уже подготовлен небольшой отчет с формулой расчета прибыли по каждому менеджеру. Для формулы расчета прибыли я использовал DAX функцию [SUMX](#):



The screenshot shows a DAX formula bar at the top and a table visualization below it. The formula is: `Прибыль = SUMX('Заявки', 'Заявки'[Сумма] - 'Заявки'[Затраты])`. The table visualization has two columns: 'Менеджер' and 'Прибыль'. It lists four managers: Воснецова (141), Петров (470), Сидоров (300), and Скворцов (95), with a total row 'Всего' showing 1006.

Менеджер	Прибыль
Воснецова	141
Петров	470
Сидоров	300
Скворцов	95
Всего	1006

Немного поясню, как рассчитывается формула прибыли в этом примере: когда в таблице этой визуализации рассчитывалась ячейка прибыли по менеджеру Воснецова, то в функцию SUMX была подана таблица «Заявки», отфильтрованная строкой этой визуализации, а конкретно, менеджером Воснецова. И именно поэтому, SUMX рассчитала прибыль только по менеджеру Воснецова.

Теперь, о сути задачи примера — нам нужно рассчитать в % вклад каждого менеджера в общий итог прибыли.

Для этого, первым действием, на основе созданной мною ранее меры [Прибыль], рассчитаем для каждой строки таблицы в визуализации общую прибыль всех менеджеров.

А сделать мы это сможем при помощи функций CALCULATE (про работу функции CALCULATE Вы можете прочитать [в этой статье](#)) и ALL. Где

ALL, находясь в составе CALCULATE, будет удалять ранее наложенные фильтры с таблицы «Заявки». Поэтому, созданная мною ранее мера [Прибыль], также, находясь в составе CALCULATE, будет уже рассчитываться на основе таблицы «Заявки», очищенной от всех фильтров.

То есть, теперь в SUMX уже будет подана полная исходная таблица «Заявки», так как ALL удалит все фильтры, которые были наложены строками таблицы визуализации.

Код формулы будет таким:

```
Прибыль% =  
CALCULATE(  
    [Прибыль];  
    ALL ('Заявки')  
)
```

Результат выполнения этой формулы получился тот, который мы и ожидали — наша новая мера рассчитала для каждой строки визуализации общую сумму прибыли:

Прибыль% = CALCULATE( [Прибыль]; ALL('Заявки') )		
Менеджер	Прибыль	Прибыль%
Воснецова	141	1006
Петров	470	1006
Сидоров	300	1006
Скворцов	95	1006
Всего	1006	1006

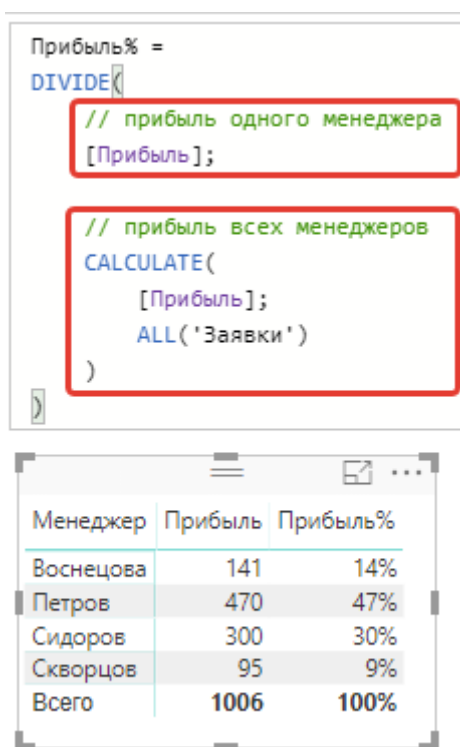
До окончательного результата остался всего один шаг. Теперь мы имеем прибыль по каждому менеджеру и общую прибыль по всем

менеджерам. Для того, чтобы рассчитать прибыль в %, нам всего лишь нужно разделить прибыль по одному менеджеру на всю общую прибыль. В итоге формула будет такой:

```
Прибыль% =  
DIVIDE(  
    // прибыль одного менеджера  
    [Прибыль];  
  
    // прибыль всех менеджеров  
    CALCULATE(  
        [Прибыль];  
        ALL ('Заявки')  
    )  
)
```

Для операции деления в этой формуле я использовал DAX функцию [DIVIDE](#), которая по факту является делением (делит первый параметр на второй параметр) с обработкой ошибки деления на 0.

Результат работы этой формулы, следующий:

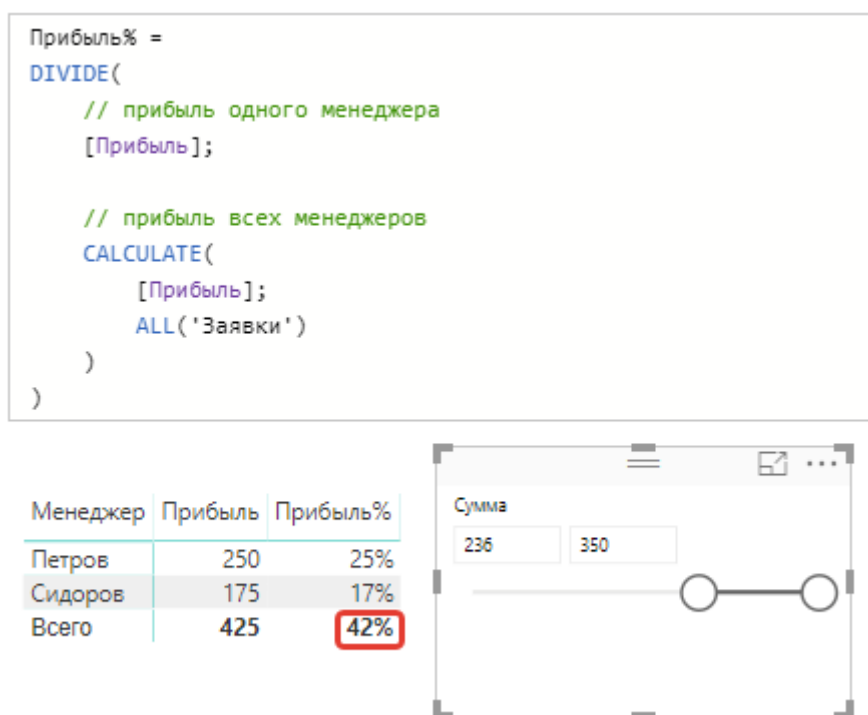


```
Прибыль% =  
DIVIDE(  
    // прибыль одного менеджера  
    [Прибыль];  
  
    // прибыль всех менеджеров  
    CALCULATE(  
        [Прибыль];  
        ALL('Заявки')  
    )  
)
```

Менеджер	Прибыль	Прибыль%
Воснецова	141	14%
Петров	470	47%
Сидоров	300	30%
Скворцов	95	9%
Всего	1006	100%

Вроде бы, если визуально смотреть на отчет, мы добились поставленной задачи — прибыль в % для каждого менеджера у нас рассчитывается.

Но, на самом деле, тут не все так гладко. Если в отчете добавить какой-нибудь срез, например, сделать фильтр по столбцу [Сумма] из таблицы «Заявки», то наша формула уже будет работать некорректно:



А именно, когда мы установили пользовательский фильтр «Сумма больше или равна 236», то мера [Прибыль%] — выдала нам общий результат не 100%, а 42%.

Все дело в том, что в параметре функции ALL мы указали всю таблицу «Заявки» и ALL удаляет все фильтры из всех столбцов этой таблицы.

Соответственно, в нашем примере, ALL удалила фильтры не только по столбцу [Менеджер], но и по столбцу [Сумма].

Так как в результате фильтра по столбцу [Сумма] в таблице визуализации у нас отображаются 2 менеджера, то, соответственно, мы ожидаем расчет 100% прибыли именно по 2 менеджерам. Но, мера [Прибыль%] рассчитывает этот % исходя из всех 4 менеджеров, так как на эту меру не действует фильтр по столбцу [Сумма] — его ведь ALL удалила.

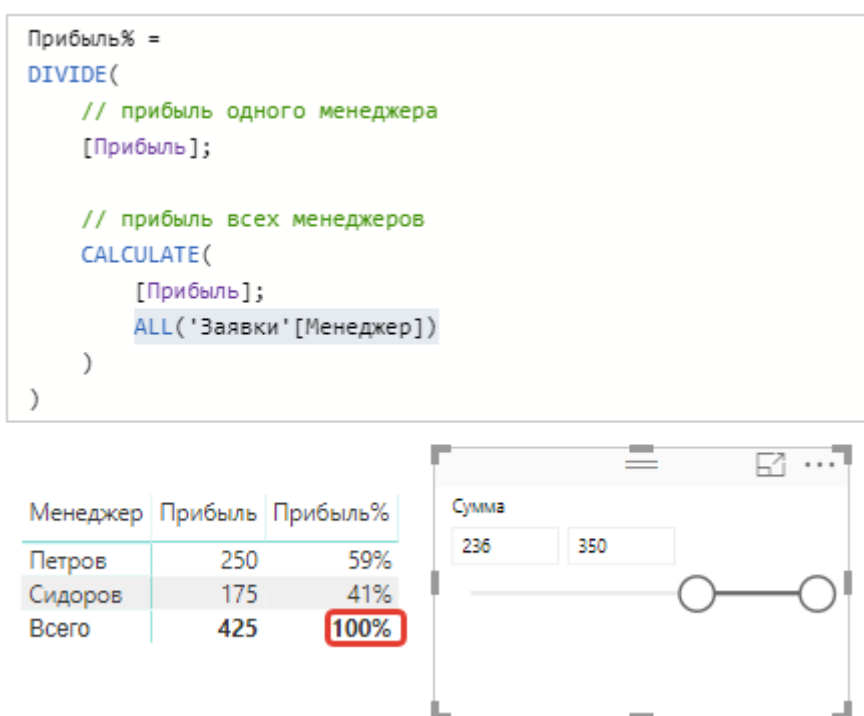
В общем, суть такая — удалять фильтр со столбца [Сумма] нам не нужно, а нужно удалить фильтр только со столбца [Менеджер]. Функция ALL у нас удаляет фильтры со всех столбцов всей таблицы. И

вот этот момент нам и нужно исправить. То есть, нужно в параметрах ALL указать не всю таблицу «Заявки», а только столбец [Менеджеры].

Исправим нашу формулу:

```
Прибыль% =  
DIVIDE(  
    // прибыль одного менеджера  
    [Прибыль];  
  
    // прибыль всех менеджеров  
    CALCULATE(  
        [Прибыль];  
        ALL ('Заявки'[Менеджер])  
    )  
)
```

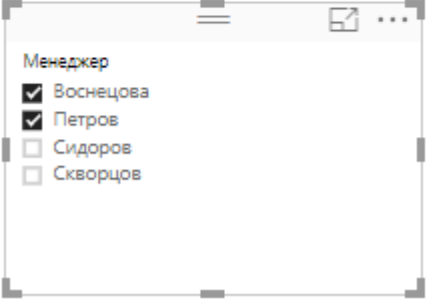
Теперь формула заработала как надо, так как ALL удаляет фильтры только со столбца [Менеджер]:



Но, на этом разбор этого практического примера еще не окончен. Давайте добавим еще один пользовательский срез, но теперь уже по самому столбцу [Менеджер]:

```
Прибыль% =  
DIVIDE(  
    // прибыль одного менеджера  
    [Прибыль];  
  
    // прибыль всех менеджеров  
    CALCULATE(  
        [Прибыль];  
        ALL('Заявки'[Менеджер])  
    )  
)
```

Менеджер	Прибыль	Прибыль%
Воснецова	141	14%
Петров	470	47%
Всего	611	61%



И у нас вновь возникла проблема. Опять общий итог равен не 100%, а 61%. Все дело в том, что функция ALL в нашей формуле удаляет все фильтры со столбца [Менеджер]. Но, нам, с одной стороны, этот фильтр нужен в созданном срезе, а с другой стороны, не нужен в таблице визуализации. Как быть?

Решением будет являться использование другой DAX функции ALLSELECTED.

## DAX функция ALLEXCEPT

ALLEXCEPT () — удаляет все наложенные фильтры с указанной таблицы в первом параметре, кроме тех столбцов, которые указаны во втором и последующих параметрах.

Синтаксис:

ALLEXCEPT ('Таблица'; [Столбец 1]; [Столбец 2]; ...; [Столбец N])

Иногда, в Интернете я встречаю написание этой функции в 2 слова: ALL EXCEPT, что неправильно...

Пример формулы с использованием DAX функции ALLEXCEPT: например, имеется таблица, состоящая из 5 столбцов. Необходимо удалить фильтры из 4 столбцов. Для этого можно использовать функцию ALL:

```
ALL (  
    [Столбец 1];  
    [Столбец 2];  
    [Столбец 3];  
    [Столбец 4]  
)
```

Но, в данном случае, проще использовать DAX функцию ALLEXCEPT, которая также, как и ALL, удаляет фильтры со всей таблицы, кроме указанных столбцов. В нашем примере с 5 столбца удалять фильтры не нужно, поэтому, формулу выше можно записать с участием ALLEXCEPT так:

```
ALLEXCEPT (  
    'Таблица';  
    [Столбец 5]  
)
```



Эта формула удалит все фильтры из всех столбцов таблицы, кроме 5 столбца.

Но, тем не менее, между этими двумя вариантами есть разница. Если мы, в процессе работы, добавим в таблицу 6 столбец, то в варианте использования функции ALL:

```
ALL (  
    [Столбец 1];  
    [Столбец 2];  
    [Столбец 3];  
    [Столбец 4]  
)
```

от фильтров будут очищены 1, 2, 3 и 4 столбцы. 5 и 6 столбцы будут под фильтрами.

А в варианте использования функции ALLEXCEPT:

```
ALLEXCEPT (  
    'Таблица';  
    [Столбец 5]  
)
```

от фильтров будут очищены уже 1, 2, 3, 4 и 6 столбец. А под фильтром останется только пятый столбец.

## DAX функция ALLSELECTED

ALLSELECTED () — удаляет последний наложенный уровень фильтра.

Синтаксис:

- **ALLSELECTED ()** — удаляет последний наложенный уровень фильтра со всех таблиц модели данных
- **ALLSELECTED ('Таблица')** — удаляет последний наложенный уровень фильтра с указанной таблицы
- **ALLSELECTED ([Столбец])** — удаляет последний наложенный уровень фильтра только с одного указанного столбца

В качестве примера формулы с использованием функции ALLSELECTED, продолжим разбирать пример, который рассматривали выше.

Итак, в нашем примере использовались несколько уровней фильтров по столбцу [Менеджер].

Первый уровень фильтра — это срез, который мы создали по менеджерам. Его нам нужно оставить (но функция ALL в примере выше его удаляла).

Второй и последний уровень фильтра — это, непосредственно, сами строки в таблице визуализации, где и рассчитывается значение формулы. Именно этот, последний уровень фильтра, нам и нужно удалить, чтоб в итоге % всегда рассчитывался правильно, несмотря на то, какие бы мы пользовательские срезы не устанавливали.

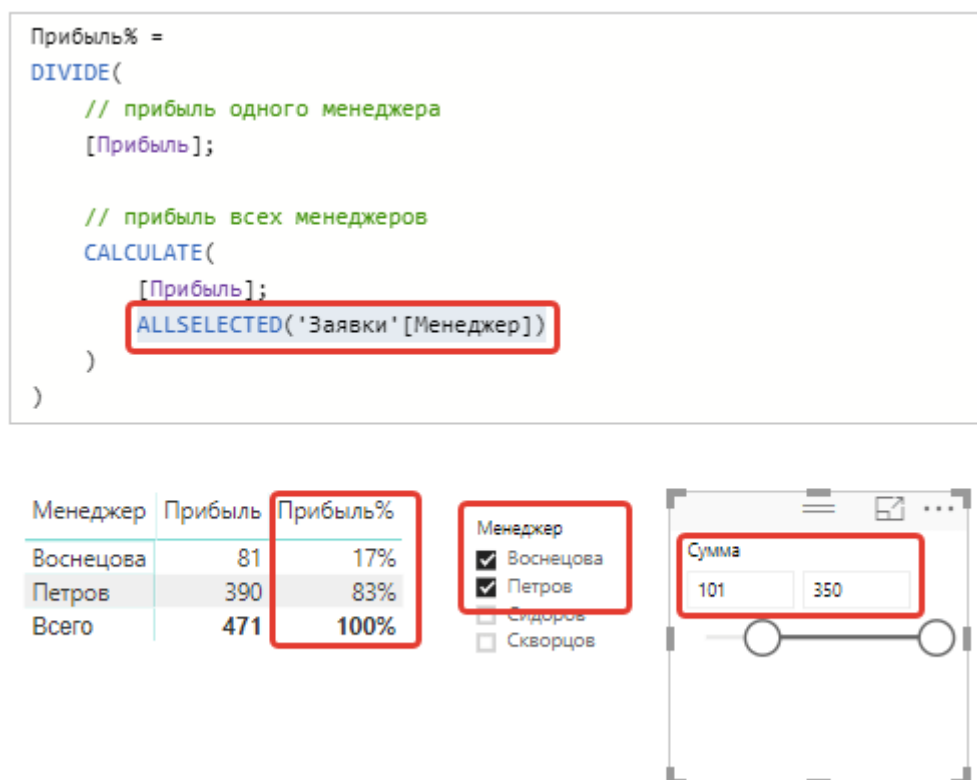
Исправим в примере нашу формулу — ALL заменим на ALLSELECTED:

```

Прибыль% =
DIVIDE(
    // прибыль одного менеджера
    [Прибыль];

    // прибыль всех менеджеров
    CALCULATE(
        [Прибыль];
        ALLSELECTED ('Заявки'[Менеджер])
    )
)
    
```

Вот теперь, все точно работает правильно! DAX функция ALLSELECTED нам в этом помогла и удалила последний наложенный уровень фильтра со столбца [Менеджер] в таблице визуализации. Все % рассчитываются как надо, несмотря на то что мы установили срезы и по менеджерам, и по сумме:



## DAX функция ALLNOBLANKROW

ALLNOBLANKROW () — возвращает полную исходную таблицу или столбец без учета пустых строк, игнорируя все, ранее наложенные фильтры.

Синтаксис:

- **ALLNOBLANKROW ('Таблица')** — возвращает все строки исходной таблицы без учета пустых строк
- **ALLNOBLANKROW ([Столбец 1]; [Столбец 2]; ...; [Столбец N])** — возвращает столбец (столбцы) со всеми уникальными значениями исходного столбца (столбцов) без учета пустых строк

ALLNOBLANKROW () — функция, полностью идентичная функции ALL, за исключением того, что ALLNOBLANKROW не учитывает, автоматически создаваемые DAX, пустые строки.

Ситуация с автоматически создаваемыми DAX пустыми строками возможна тогда, когда между связанными таблицами различаются значения ключевых столбцов.

В примере ниже, в таблице фактов «Заявки» появилась строка с заявкой от менеджера Поклонский. Но, в связанной таблице «Справочник менеджеров» данного менеджера нет. В этом случае DAX автоматически в этой таблице создаст пустую строку:

Заявки

Менеджер  
Σ Затраты  
Σ Сумма

СпрМенеджеры

Менеджер  
Отдел

Менеджер	Затраты	Сумма
Петров	20	100
Сидоров	50	150
Воснецова	30	90
Сидоров	5	30
Скворцов	50	130
Петров	60	200
Сидоров	75	250
Воснецова	20	101
Скворцов	50	130
Петров	100	350

Менеджер	Отдел
Петров	Отдел1
Сидоров	Отдел1
Воснецова	Отдел2
Скворцов	Отдел2

Поклонский 120 500

Именно эту пустую строку ALLNOBLANKROW и не учитывает.

## DAX функция VALUES

VALUES () — функция, как я уже написал во вступлении, которая создает таблицу, состоящую из одного столбца с набором уникальных значений исходной таблицы или столбца.

Синтаксис:

VALUES ('Таблица') или VALUES ([Столбец])

Данная функция достаточно простая, поэтому описывать ее сильно не будем, а сразу перейдем к практическим примерам и построению [DAX формул](#).

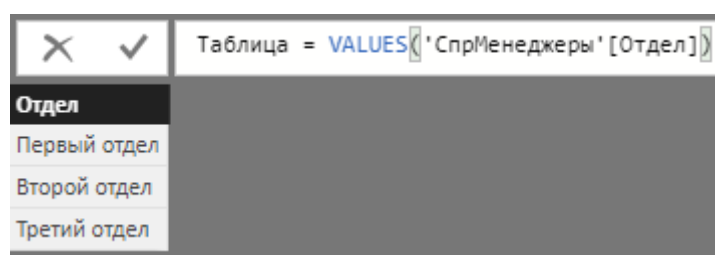
В качестве примера возьмем исходную таблицу «СпрМенеджеры», находящуюся в [Power BI](#) и содержащую информацию по менеджерам и отделам:

Менеджер	Отдел
Смирнов	Первый отдел
Сидоров	Первый отдел
Колесников	Второй отдел
Василькова	Второй отдел
Соколова	Третий отдел

Создадим в [Power BI Desktop](#) во вкладке «Моделирование» вычисляемую таблицу уникальных значений отделов на основе формулы с участием VALUES:

Таблица = VALUES ('СпрМенеджеры'[Отдел])

Как результат, мы увидим столбец с уникальными значениями отделов:

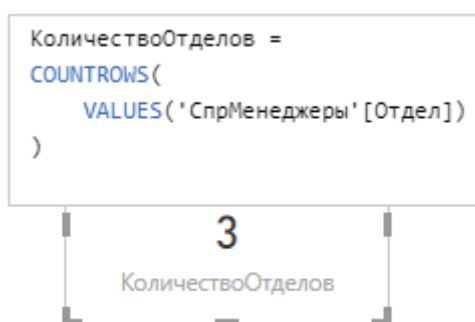


Саму по себе функцию VALUES, как на примере выше, используют достаточно редко. Обычно в формулах она используется в партнерстве с другими [DAX функциями](#).

Например, мы можем в рамках одной формулы объединить VALUES с функцией [COUNTROWS](#), которая считает количество строк. Тем самым, мы можем создать меру, вычисляющую на основе исходной таблицы «СпрМенеджеры» количество отделов в организации:

```
Количество Отделов =  
COUNTROWS (  
    VALUES ('СпрМенеджеры'[Отдел])  
)
```

Результатом выполнения кода этой формулы будет количество отделов, равное 3:



## DAX функция DISTINCT

DISTINCT () — DAX функция, которая возвращает столбец с уникальными значениями исходного столбца или таблицы. В своей практике я эту функцию очень часто использую при создании так называемых таблиц справочников (измерений) на основе исходного столбца из таблицы фактов.

Таблицы фактов обычно содержат очень большой объем данных (миллионы строк), которые могут повторяться. Например, таблица фактов «Заявки», в которой в каждой строке указана информация по одной заявке — сумма, затраты, менеджер, дата сделки и т.д.

Таблицы справочников (измерений) гораздо меньше, количество строк в них может составлять десятки или сотни, в крайних случаях (в больших корпорациях) несколько тысяч. В них содержатся только уникальные значения, расшифровывающие то или иное измерение, например, менеджеров (имя, фамилия, телефон, адрес проживания и т.д.)

Синтаксис:

**DISTINCT ([‘Таблица’]) или DISTINCT ([Столбец])**

В качестве примера работы функции DISTINCT напишем простую формулу, которая создаст в нашей модели данных справочник по менеджерам на основе исходной таблицы фактов «Заявки»:



Менеджер	Затраты	Сумма
Петров	20	100
Сидоров	50	150
Воснецова	30	90
Сидоров	5	30
Скворцов	50	130
Петров	60	200
Сидоров	75	250
Воснецова	20	101
Скворцов	5	20
Петров	100	350

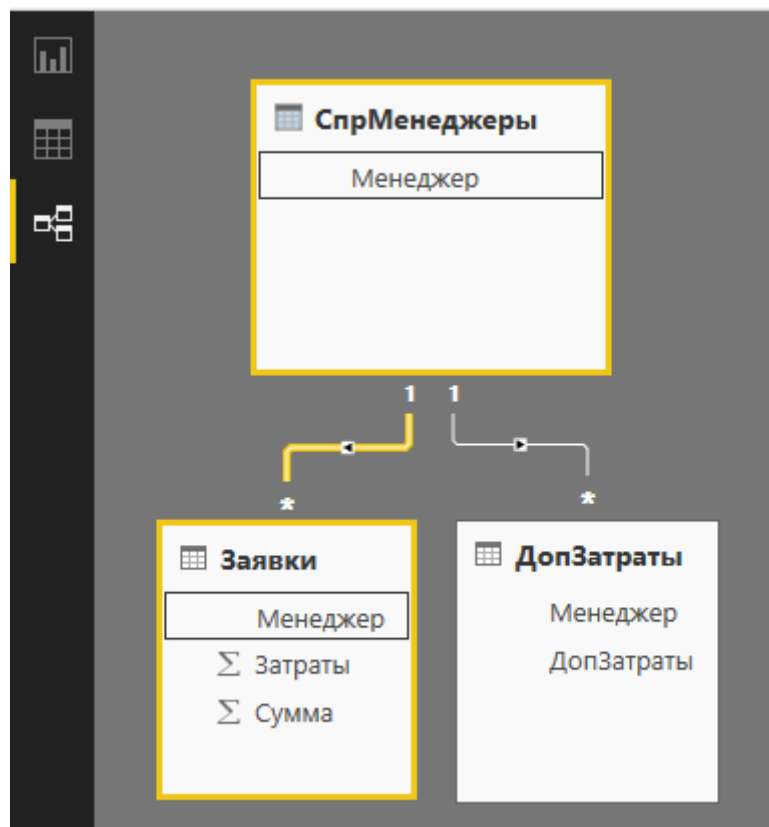
Итак, код формулы в [DAX](#) достаточно простой:

`СпрМенеджеры = DISTINCT ('Заявки'[Менеджер])`

Создав на основе этой формулы вычисляемую таблицу, получим в модели данных справочник по менеджерам:

✕	✓	СпрМенеджеры = <code>DISTINCT('Заявки'[Менеджер])</code>
Менеджер		
Петров		
Сидоров		
Воснецова		
Скворцов		

Далее, в Power BI Desktop во вкладке «Связи» можно будет объединить все таблицы, содержащие столбец [Менеджер], на основе созданного функцией DISTINCT, справочника «СпрМенеджер»:



В результате использования функции **DISTINCT**, мы смогли создать справочник с уникальными значениями менеджеров, при помощи которого, объединили две таблицы фактов «Заявки» и «Дополнительные затраты».

## DAX функция RELATED

RELATED () — находясь в одной таблице, позволяет в рамках контекста строки получить связанное значение из второй таблицы по связи «Многие к одному».

Синтаксис:

RELATED ([Столбец])

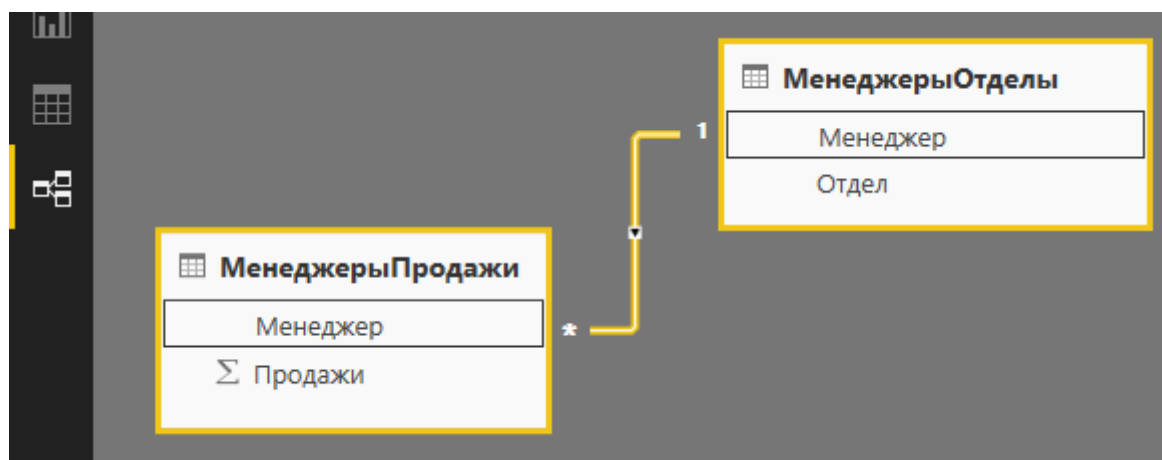
Рассмотрим пример [DAX формулы](#) с участием RELATED.

В [Power BI Desktop](#) имеются 2 исходные таблицы «Менеджеры Продажи» и «Менеджеры Отделы»:

Менеджер	Продажи
Смирнов	18000
Сидоров	31000
Смирнов	13000
Колесников	16500
Василькова	24000
Соколова	38000
Соколова	10000

Менеджер	Отдел
Смирнов	Первый отдел
Сидоров	Первый отдел
Колесников	Второй отдел
Василькова	Второй отдел
Соколова	Третий отдел

Между ними настроена связь по полю «Менеджер» по типу «Многие к одному», то есть, много менеджеров может находиться в одном отделе:



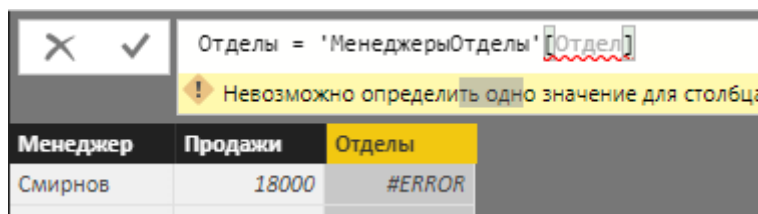
Попробуем в таблицу «Менеджеры Продажи» через связь добавить третий столбец [Отделы]:

Менеджер	Продажи	Отделы
Смирнов	18000	
Сидоров	31000	
Смирнов	13000	
Колесников	16500	
Василькова	24000	
Соколова	38000	
Соколова	10000	

Для этого создадим в Power BI Desktop во вкладке «Моделирование» вычисляемый столбец и постараемся в его формуле прописать столбец [Отделы] из связанной таблицы «Менеджеры Отделы»:

Отделы = 'МенеджерыОтделы'[Отдел]

И по данной формуле у нас всплывает ошибка:



На самом деле все правильно. Ведь мы, находясь в одной таблице, в формуле вычисляемого столбца попытались сослаться на столбец совершенно другой таблицы. И даже несмотря на имеющуюся между ними связь, так значение получить невозможно. Вот тут, как раз таки, и необходимо использовать DAX функцию RELATED.

Давайте перепишем формулу вычисляемого столбца с использованием RELATED:

Отделы = RELATED ('МенеджерыОтделы'[Отдел])

И вот теперь, все прошло удачно. Функция RELATED позволила нам, находясь в одной таблице, дотянуться до значения другой таблицы по связи «Многие к одному» и создать соответствующий новый столбец:

Отделы = RELATED('МенеджерыОтделы'[Отдел])		
Менеджер	Продажи	Отделы
Смирнов	18000	Первый отдел
Сидоров	31000	Первый отдел
Смирнов	13000	Первый отдел
Колесников	16500	Второй отдел
Василькова	24000	Второй отдел
Соколова	38000	Третий отдел
Соколова	10000	Третий отдел

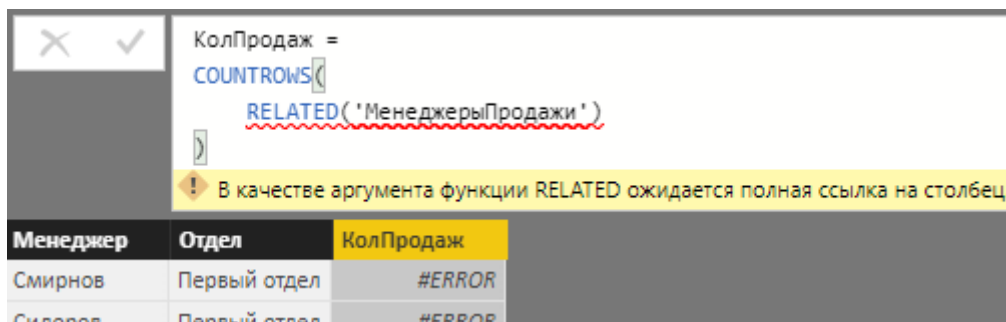
Теперь, давайте рассмотрим противоположную ситуацию. Сейчас мы будем находиться в другой таблице «Менеджеры Отделы» и в ней нам нужно будет создать новый столбец с подсчетом количества продаж каждым менеджером.

То есть, нам нужно подсоединиться через связь к таблице «Менеджеры Продажи» и там посчитать количество продаж каждого менеджера. Это количество мы можем посчитать при помощи DAX функции [COUNTROWS](#), которая считает количество строк. Ну а подсоединяться к другой таблице через связь мы будем с помощью RELATED.

Пропишем данную формулу:

```
КолПродаж =
COUNTROWS(
    RELATED ('МенеджерыПродажи')
)
```

И у нас опять получилась ошибка:



На самом деле ошибка закономерна, так как функция `RELATED` работает по связи «Многие к одному», что у нас было соблюдено в первом примере и что мы нарушили сейчас. Так как в данном примере у нас уже связь другая, а именно «Один ко многим» (один отдел может в себе содержать много менеджеров) и с этой связью `RELATED` уже не работает.

Тут нам на помощь может прийти вторая [функция](#) работы по связям в DAX: `RELATEDTABLE`.

# DAX функция RELATEDTABLE

RELATEDTABLE () — находясь в одной таблице, возвращает связанную таблицу значений из второй таблицы по связи «Один ко многим», где одна строка соответствует многим строкам.

Синтаксис:

**RELATEDTABLE** ('Таблица')

Функция RELATEDTABLE, в отличие от RELATED, уже не возвращает какое-то одно скалярное значение, она возвращает именно связанную таблицу значений, с которыми мы можем что-то сделать, например посчитать количество строк:

	Менеджер	Продажи
1	Смирнов	18000
	Сидоров	31000
2	Смирнов	13000
	Колесников	16500
	Василькова	24000
	Соколова	38000
	Соколова	10000

Менеджер	Отдел
Смирнов	Первый отдел
Сидоров	Первый отдел
Колесников	Второй отдел
Василькова	Второй отдел
Соколова	Третий отдел

Давайте доработаем формулу предыдущего примера и исправим там ошибку, а именно, заменим функцию RELATED на RELATEDTABLE:

```
КолПродаж =
COUNTROWS(
    RELATEDTABLE ('МенеджерыПродажи')
)
```

Теперь у нас все хорошо, пример формулы с RELATEDTABLE отработал отлично и посчитал нам количество продаж по каждому менеджеру:

<div>✕ ✓</div> <div>КолПродаж = COUNTROWS( RELATEDTABLE('МенеджерыПродажи') )</div>		
Менеджер	Отдел	КолПродаж
Смирнов	Первый отдел	2
Сидоров	Первый отдел	1
Колесников	Второй отдел	1
Василькова	Второй отдел	1
Соколова	Третий отдел	2



## 8. ТАБЛИЧНЫЕ ФУНКЦИИ

### DAX

# DAX функция ADDCOLUMNS

ADDCOLUMNS () — название этой функции говорит само за себя: добавить столбец. Она создает вычисляемый столбец в таблице.

На просторах Интернета я часто встречаю, что некоторые пользователи пишут эту формулу так: addcolumn (без последней буквы s) или add columns (в два отдельных слова) — что совершенно неправильно. Правильное написание этой функции ADDCOLUMNS.

Синтаксис:

```
ADDCOLUMNS (  
    'Таблица';  
    "Имя столбца 1"; Выражение 1;  
    "Имя столбца 2"; Выражение 2;  
    "..."; ...;  
    "Имя столбца N"; Выражение N  
)
```

Где:

- 'Таблица' — существующая таблица или табличное выражение к которому добавляется столбец
- «ИмяСтолбца» — имя создаваемого столбца
- Выражение — вычисляемое выражение для каждой строки создаваемого столбца

Функцию ADDCOLUMNS использовать саму по себе нет никакого смысла, она используется только совместно с какими-либо другими [функциями в DAX](#), создавая виртуальный вычисляемый столбец, который будет использоваться другой функцией.

Это нужно для того, чтобы не создавать вычисляемый столбец в физической памяти, чтобы он не занимал место оперативной памяти ПК.

Для понимания сути работы DAX функции ADDCOLUMNS, рассмотрим следующий пример. В модели данных [Power BI Desktop](#) имеется таблица «Продажи», содержащая дату и сумму продажи:

Дата	СуммаПродаж
16 марта 2018 г.	15000
20 марта 2018 г.	60000
24 марта 2018 г.	23000
10 апреля 2018 г.	10000
14 апреля 2018 г.	14000
25 апреля 2018 г.	53000

Задача: написать формулу суммы продаж только за апрель месяца. Без создания дополнительного вычисляемого столбца [Месяц] в физической памяти модели данных.

Для того, чтобы столбец [Месяц] не создавать в физической памяти модели данных, воспользуемся DAX функцией ADDCOLUMNS, которая добавит этот вычисляемый столбец к таблице виртуально. При этом формула суммы продаж за апрель будет ссылаться именно на виртуальный столбец [Месяц], этим самым, мы не будем загружать оперативную память ПК.

Итак, формула с использованием ADDCOLUMNS будет такой:

```
/* 0 */ СуммаАпрель =
/* 1 */ SUMX(
/* 2 */     FILTER(
/* 3 */         ADDCOLUMNS(
/* 4 */             'Продажи';
/* 5 */             "Месяц";
/* 6 */             FORMAT ('Продажи'[Дата]; "MMMM")
/* 7 */         );
/* 8 */         [Месяц] = "Апрель"
/* 9 */     );
/* 10 */ 'Продажи'[СуммаПродаж]
/* 11 */ )
```

Для удобства объяснения этой формулы я каждую строку пронумеровал.

В 0 строке объявляется имя создаваемой меры.

В 1 строке вызывается функция [SUMX](#), которая и будет считать сумму столбца [СуммаПродаж] (в строке 10).

Строки 2-9 в коде формулы являются входящей таблицей в SUMX (первым ее параметром). Эта таблица создается при помощи функции [FILTER](#), в которую, также подается таблица (строки 3-7) и затем эта поданная таблица фильтруется по условию столбец [Месяц] = «Апрель» (строка 8).

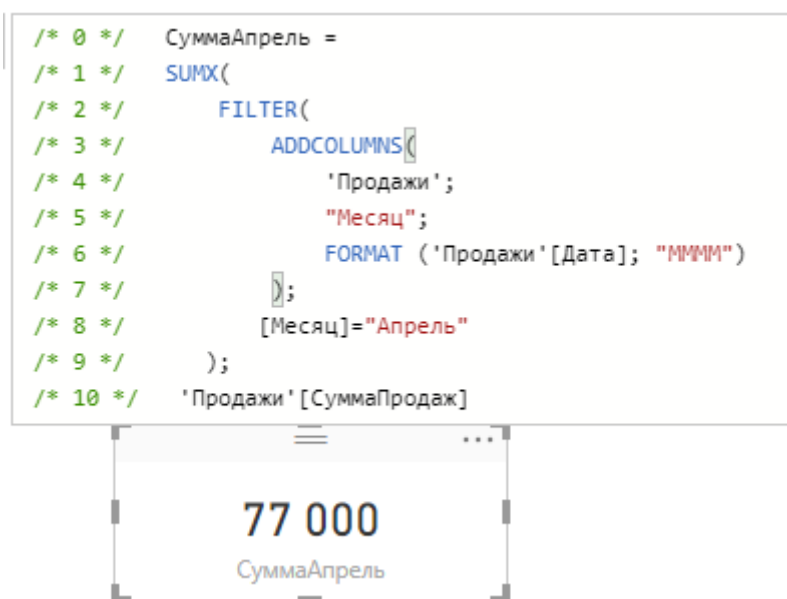
По какому столбцу [Месяц] функция FILTER будет фильтровать таблицу? И какую вообще таблицу она будет фильтровать?

Эта таблица — это не что иное, как исходная таблица «Продажи», в которой был виртуально добавлен столбец [Месяц] при помощи функции ADDCOLUMNS (3-7 строки).

Разберем построчно этот код с ADDCOLUMNS (3-7 строки).

- В качестве первого параметра (4 строка) мы прописали исходную таблицу «Продажи».
- Вторым параметром (строка 5) мы прописали имя создаваемого виртуального столбца [Месяц].
- В третьем параметре (6 строка) указано само выражение для расчета создаваемого столбца. И это выражение состоит из функции [FORMAT](#), которая, в свою очередь, из столбца [Дата] исходной таблицы «Продажи» возвратила наименование месяца. Этот месяц и был прописан в виртуальном столбце, созданным ADDCOLUMNS.

Итог работы этой формулы будет такой:



Итак, мы убедились с Вами, что иногда, в процессе работы в языке DAX, возникают такие моменты, что для расчетов нужно создавать вычисляемые столбцы.

Но, так как вычисляемые столбцы дают нагрузку на оперативную память ПК, то при помощи функции ADDCOLUMNS эти вычисляемые столбцы можно в физической памяти не создавать. Ведь эта функция дает возможность их создать виртуально, не нагружая реальную оперативную память.

## DAX функция DATATABLE

DATATABLE () — возвращает таблицу с нужными данными (позволяет создать таблицу «с нуля»).

Синтаксис:

```
DATATABLE (  
    "Имя столбца 1"; Тип Данных Столбца 1;  
    "Имя столбца 2"; Тип Данных Столбца 2;  
    "Имя столбца N"; Тип Данных Столбца N;  
    {  
        {Значение 1; Значение 2; Значение N};  
        {Значение 1; Значение 2; Значение N};  
        {Значение 1; Значение 2; Значение N};  
        {Значение 1; Значение 2; Значение N}  
    }  
)
```

Где:

- «Имя столбца» — имя создаваемого столбца в таблице
- Тип Данных Столбца — тип данных создаваемого столбца: INTEGER (целые числа), DOUBLE (дробные числа), STRING (текстовая строка), BOOLEAN (логический тип), CURRENCY (денежный тип), DATETIME (дата и время)
- {Значение 1; Значение 2; Значение 3} — строки создаваемой таблицы
- Значение 1 — любое значение или выражение, соответствующие создаваемому столбцу и его типу данных

Пример [формулы](#) на основе использования DAX функции DATATABLE.

Так как DATATABLE позволяет создавать вычисляемую таблицу так сказать «с нуля», то для ее работы исходные таблицы можно не

использовать, а прописать нужную информацию прямо в самой формуле. В нашем примере поступим именно так.

В Power BI Desktop во вкладке «Моделирование» создадим вычисляемую таблицу по следующей формуле на основе функции DATATABLE:

```
/*0*/ Таблица =  
/*1*/ DATATABLE (  
/*2*/   "Менеджер"; STRING;  
/*3*/   "Продажи"; CURRENCY;  
/*4*/   {  
/*5*/     {"Сидоров"; 31000};  
/*6*/     {"Петров"; 15000};  
/*7*/     {"Свиридова"; 35000}  
/*8*/   }  
/*9*/ )
```

В данной формуле в качестве параметров DATATABLE во 2 и 3 строках мы прописали имена создаваемых столбцов и рядом их типы данных.

Фигурные скобки в 4 и 8 строках говорят о том, что внутри располагаются строки создаваемой таблицы.

5, 6, 7 номера — это непосредственно 3 строки таблицы, внутри которых расположены значения в соответствии со структурой ранее прописанных столбцов.

Результатом выполнения этой формулы на основе DAX функции DATATABLE, будет созданная в модели данных в Power BI вычисляемая таблица:

<div><div>✕</div><div>✓</div></div>		<pre>/*0*/ Таблица = /*1*/ DATATABLE ( /*2*/     "Менеджер"; STRING; /*3*/     "Продажи"; CURRENCY; /*4*/     { /*5*/         {"Сидоров"; 31000}; /*6*/         {"Петров"; 15000}; /*7*/         {"Свиридова"; 35000} /*8*/     } /*9*/ )</pre>
Менеджер	Продажи	
Сидоров	31 000 ₽	
Петров	15 000 ₽	
Свиридова	35 000 ₽	



## DAX функция SUMMARIZE

SUMMARIZE () — создает сводную таблицу с агрегированными итогами по выбранным группам.

Синтаксис:

```
SUMMARIZE (  
    'Таблица';  
    [Столбец 1]; [Столбец 2]; ...; [Столбец N];  
    "Имя столбца 1"; Выражение 1;  
    "Имя столбца 2"; Выражение 2;  
    ...; ...;  
    "Имя столбца N"; Выражение N  
)
```

Где:

- 'Таблица' — исходная существующая таблица или выражение, возвращающее таблицу, значения которой мы хотим сгруппировать
- [Столбец] — столбец для группировки
- «Имя столбца» — имя создаваемого столбца для значений группировки
- Выражение — вычисляемое выражение для значений группировки

### Пример использования DAX функции SUMMARIZE

Пример формулы на основе работы SUMMARIZE мы будем рассматривать в [Power BI](#). Так как сводную таблицу мы будем создавать в физической модели данных, а это возможно только в Power BI.

В Excel (Power Pivot) создавать вычисляемые таблицы в модели данных, к сожалению, нельзя. Поэтому SUMMARIZE там создает сводные таблицы только виртуально, во время непосредственного вычисления формулы.

Итак, в [Power BI Desktop](#) у нас имеется исходная таблица «Продажи Менеджеров», где каждая строка соответствует конкретной продаже:

Менеджер	Продажи
Сидоров	31000
Петров	15000
Свиридова	35000
Свиридова	35000
Свиридова	35000
Петров	15000
Сидоров	31000

В Power BI Desktop во вкладке «Моделирование» создадим в модели данных вычисляемую сводную таблицу при помощи [DAX формулы](#) с участием рассматриваемой функции SUMMARIZE:

```
Сводные Продажи По Менеджерам =  
SUMMARIZE (  
    'ПродажиМенеджеров';  
    'ПродажиМенеджеров'[Менеджер];  
    "СводныеПродажи";  
    SUM('ПродажиМенеджеров'[Продажи])  
)
```

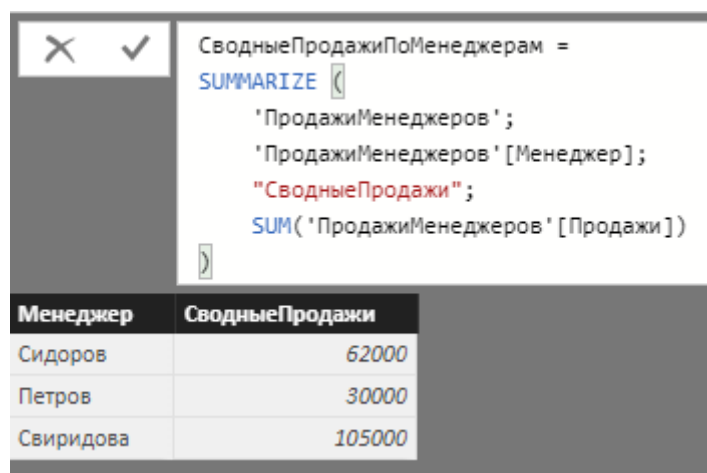
Где, в качестве первого параметра мы указали исходную таблицу «Продажи Менеджеров», значения которой мы хотим сгруппировать.

В качестве второго параметра мы прописали тот столбец, по которому будет происходить группировка значений.

В третьем параметре мы указали имя создаваемого столбца, в котором расположатся агрегированные значения.

Ну и, в четвертом параметре само выражение агрегации на основе DAX функции [SUM](#).

То есть, сводная таблица, создаваемая функцией SUMMARIZE, будет состоять из столбца с категорией [Менеджер], по которой сводится информация и из нового столбца [Сводные Продажи], где будут размещены значения агрегации (сумма продаж по менеджерам):



The screenshot shows the DAX formula bar with the following code:

```
СводныеПродажиПоМенеджерам =  
SUMMARIZE(  
    'ПродажиМенеджеров';  
    'ПродажиМенеджеров'[Менеджер];  
    "СводныеПродажи";  
    SUM('ПродажиМенеджеров'[Продажи])  
)
```

Below the formula bar, a preview table is displayed:

Менеджер	СводныеПродажи
Сидоров	62000
Петров	30000
Свиридова	105000

Также, у рассматриваемой функции имеются еще и дополнительные параметры, рассмотрим их ниже.

## Дополнительные параметры SUMMARIZE — ROLLUP и ROLLUPGROUP

Если мы столбцы для группировки, указанные в параметрах, дополнительно обернем в синтаксическое выражение ROLLUP (ROLLUPGROUP ([Столбец])), то в добавок к сводным агрегационным значениям мы еще получим итоги по группам.

Синтаксис:

```
SUMMARIZE (  
    'Таблица';  
    ROLLUP ( ROLLUPGROUP ([Столбец 1]; [Столбец 2]; ...; [Столбец N]));  
    "Имя столбца 1"; Выражение 1;  
    "Имя столбца 2"; Выражение 2;  
    ...; ...;  
    "Имя столбца N"; Выражение N  
)
```

Рассмотрим пример формулы с использованием ROLLUP (ROLLUPGROUP ()).

В Power BI имеется исходная таблица «Продажи Менеджеров»:

Менеджер	Продажи	Отдел
Сидоров	31000	Первый отдел
Петров	15000	Второй отдел
Свиридова	35000	Второй отдел
Свиридова	35000	Второй отдел
Свиридова	35000	Второй отдел
Петров	15000	Второй отдел
Сидоров	31000	Первый отдел

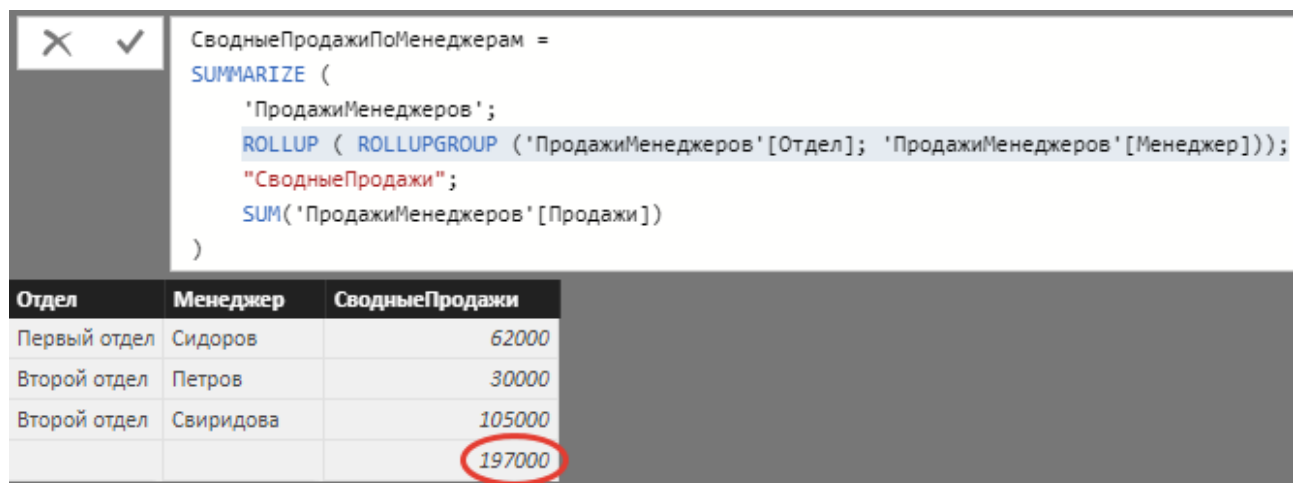
При помощи SUMMARIZE создадим новую сводную таблицу, только теперь, в качестве столбцов, по которым будем группировать агрегированные значения, мы укажем 2 столбца [Отдел] и [Менеджер].

А также, при помощи дополнительных параметров ROLLUP и ROLLUPGROUP, подведем общий итог по получившимся агрегированным значениям. Ниже код этой формулы:

Сводные Продажи По Менеджерам =

```
SUMMARIZE (  
    'ПродажиМенеджеров';  
    ROLLUP ( ROLLUPGROUP ('ПродажиМенеджеров'[Отдел];  
        'ПродажиМенеджеров'[Менеджер]));  
    "СводныеПродажи";  
    SUM('ПродажиМенеджеров'[Продажи])  
)
```

Результатом выполнения выше приведенной формулы будет сводная таблица с подведением общего итога:



The screenshot shows the DAX formula bar with the following formula:

```
СводныеПродажиПоМенеджерам =  
SUMMARIZE (  
    'ПродажиМенеджеров';  
    ROLLUP ( ROLLUPGROUP ('ПродажиМенеджеров'[Отдел]; 'ПродажиМенеджеров'[Менеджер]));  
    "СводныеПродажи";  
    SUM('ПродажиМенеджеров'[Продажи])  
)
```

Below the formula bar, a summary table is displayed with the following data:

Отдел	Менеджер	СводныеПродажи
Первый отдел	Сидоров	62000
Второй отдел	Петров	30000
Второй отдел	Свиридова	105000
		197000

## DAX функция

# SUMMARIZECOLUMNS

SUMMARIZECOLUMNS () — очень похожая DAX функция на GROUPBY, а тем более на SUMMARIZE. Она также создает сводную таблицу, но, в данном случае, с возможностью фильтрации группируемых столбцов.

Синтаксис:

```
SUMMARIZECOLUMNS (  
    [Столбец 1]; [Столбец 2]; ...; [Столбец N];  
    Фильтр;  
    "Имя Столбца 1"; Выражение 1;  
    "Имя Столбца 1"; Выражение 1;  
    "..."; ...;  
    "Имя Столбца N"; Выражение N  
)
```

Синтаксис функции SUMMARIZECOLUMNS очень похож на синтаксис GROUPBY, за исключением лишь той разницы, что:

- в первом параметре не нужно прописывать исходную таблицу;
- в выражении мы можем использовать не только X функции, но и любые другие функции агрегирования
- здесь появился еще один параметр — фильтр, по которому производится фильтрация столбцов для группировки

Рассмотрим пример формулы на основе DAX функции SUMMARIZECOLUMNS. Как и выше, пример мы будем рассматривать в Power BI на основе все той же исходной таблицы «Продажи Менеджеров»:

Менеджер	Продажи	Отдел
Сидоров	31000	Первый отдел
Петров	15000	Второй отдел
Свиридова	35000	Второй отдел
Свиридова	35000	Второй отдел
Свиридова	35000	Второй отдел
Петров	15000	Второй отдел
Сидоров	31000	Первый отдел

Создадим в Power BI Desktop во вкладке «Моделирование» новую вычисляемую таблицу и пропишем там следующую формулу с участием функции SUMMARIZECOLUMNS:

```
СводныеПродажиПоМенеджерам =  
SUMMARIZECOLUMNS (  
    'ПродажиМенеджеров'[Отдел]; 'ПродажиМенеджеров'[Менеджер];  
    FILTER ('ПродажиМенеджеров'; 'ПродажиМенеджеров'[Менеджер] <>  
        "Петров");  
    "Продажи";  
    SUM ('ПродажиМенеджеров'[Продажи])  
)
```

В первой строке мы прописали столбцы [Отдел] и [Менеджер], по которым будет происходить группировка значений.

Во второй строке прописали фильтр, созданный на основе DAX функции [FILTER](#). Данная функция фильтрует исходную таблицу «Продажи Менеджеров» по столбцу [Менеджер], где его значения не должны быть равны значению «Петров».

В третьей строке указали имя нового столбца, в котором будут прописаны новые агрегированные значения.

В четвертой строке само выражение агрегации на основе функции [SUM](#), которая сложит все продажи по категориям группировки.

Итак, результатом выполнения формулы выше на основе работы DAX функции SUMMARIZECOLUMNS будет следующая сводная таблица:

<div>✕ ✓</div> <div>СводныеПродажиПоМенеджерам = SUMMARIZECOLUMNS (     'ПродажиМенеджеров'[Отдел]; 'ПродажиМенеджеров'[Менеджер];     FILTER ('ПродажиМенеджеров'; 'ПродажиМенеджеров'[Менеджер] &lt;&gt; "Петров");     "Продажи";     SUM('ПродажиМенеджеров'[Продажи]) )</div>		
Отдел	Менеджер	Продажи
Первый отдел	Сидоров	62000
Второй отдел	Свиридова	105000

Как мы видим, создалась сводная таблица по продажам менеджеров, но без менеджера Петров, так как его мы отфильтровали в самой формуле SUMMARIZECOLUMNS.



## DAX функция GROUPBY

GROUPBY () — создает сводную таблицу, сгруппированную по указанным столбцам (название столбцов конфигурируется из названия самой исходной таблицы и названия исходного столбца для группировки).

Также, часто я встречаю раздельное написание этой функции, как DAX GROUP BY, что неправильно...

Синтаксис:

```
GROUPBY (  
    'Таблица';  
    [Столбец 1]; [Столбец 2]; ...; [Столбец N];  
    "Имя столбца 1"; Выражение 1;  
    "Имя столбца 2"; Выражение 2;  
    "..."; ...;  
    "Имя столбца N"; Выражение N  
)
```

Где:

- 'Таблица' — исходная существующая таблица или табличное выражение, значения которых мы хотим сгруппировать
- [Столбец] — столбец для группировки
- «Имя столбца» — имя создаваемого столбца для значений группировки
- Выражение — вычисляемое выражение для значений группировки

Выражение обязательно должно содержать статистическую [DAX функцию](#) формата X ([SUMX](#), [MAXX](#), [AVERAGEX](#)...), во входных параметрах которой, в качестве таблицы подставляется служебное выражение CURRENTGROUP ().

Давайте разберем все параметры GROUPBY, в том числе и служебное выражение CURRENTGROUP на примере формулы.

Для разбора примера создадим в модели данных вычисляемую таблицу по формуле с участием GROUPBY.

Так как в Excel (Power Pivot) в модели данных создавать вычисляемые таблицы нельзя, то пример будем рассматривать на основе [Power BI](#) — в ней можно создавать физические вычисляемые таблицы. А в Excel вычисляемые таблицы создаются только виртуально, во время вычисления самих формул.

Итак, в [Power BI Desktop](#) имеется исходная таблица «Продажи Менеджеров»:

Менеджер	Продажи	Отдел
Сидоров	31000	Первый отдел
Петров	15000	Второй отдел
Свиридова	35000	Второй отдел
Свиридова	35000	Второй отдел
Свиридова	35000	Второй отдел
Петров	15000	Второй отдел
Сидоров	31000	Первый отдел

Создадим во вкладке «Моделирование» вычисляемую таблицу на основе [формулы](#) с участием DAX функции GROUPBY:

Сводные Продажи По Менеджерам =

GROUPBY (

    'ПродажиМенеджеров';

    'ПродажиМенеджеров'[Отдел]; 'ПродажиМенеджеров'[Менеджер];

    "Продажи";

    SUMX (

        CURRENTGROUP ();

        'ПродажиМенеджеров'[Продажи]

    )

)

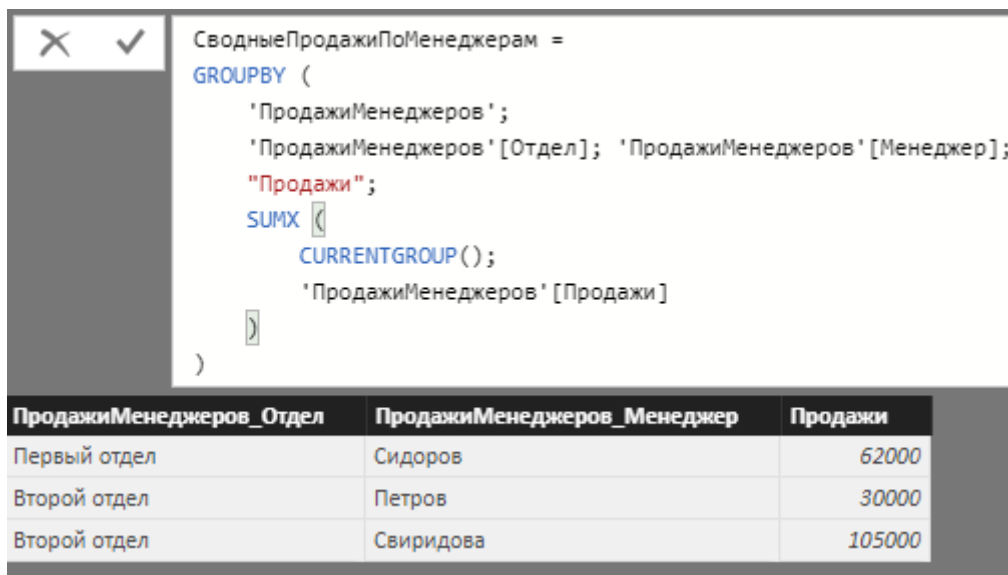
Где, в качестве первого параметра мы прописали исходную таблицу в [DAX](#), из которой будут браться все значения.

Во втором и третьем параметрах (вторая строка параметров) мы прописали столбцы [Отдел] и [Менеджер]. Именно по ним и будет происходить группировка всех значений.

В четвертом параметре (третья строка параметров) мы прописали название нового столбца, в котором расположатся агрегированные значения в создаваемой сводной таблице.

В пятом параметре, согласно синтаксису GROUPBY, расположилась вложенная функция SUMX на основе которой, будет рассчитываться агрегированная сумма всех продаж по группам. В качестве входящей таблицы в SUMX указана служебное выражение CURRENTGROUP.

Итак, результатом выполнения формулы на основе DAX функции GROUPBY будет следующая сводная таблица:



СводныеПродажиПоМенеджерам =  
 GROUPBY (  
 'ПродажиМенеджеров';  
 'ПродажиМенеджеров'[Отдел]; 'ПродажиМенеджеров'[Менеджер];  
 "Продажи";  
 SUMX (  
 CURRENTGROUP();  
 'ПродажиМенеджеров'[Продажи]  
 )  
 )

ПродажиМенеджеров_Отдел	ПродажиМенеджеров_Менеджер	Продажи
Первый отдел	Сидоров	62000
Второй отдел	Петров	30000
Второй отдел	Свиридова	105000

В этой созданной таблице при помощи GROUPBY нам удалось собрать все продажи воедино по каждому менеджеру каждого отдела в организации.

## DAX функция ROW

ROW () — функция итогов. Создает таблицу из одной строки с итогами по выбранным столбцам исходной таблицы (итоги по [сумме](#), [количеству](#), [средним значениям](#) и так далее).

Синтаксис:

```
ROW (  
"Имя столбца 1"; Выражение 1;  
"Имя столбца 2"; Выражение 2;  
"Имя столбца N"; Выражение N  
)
```

Где:

- «Имя столбца» — имя столбца в создаваемой новой таблице
- Выражение — выражение агрегации для вычисления значения столбца

Пример формулы на основе DAX функции ROW.

В Power BI имеется исходная таблица «Продажи Менеджеров»:

Менеджер	Продажи	Отдел
Смирнов	18000	Первый отдел
Сидоров	31000	Первый отдел
Колесников	16500	Второй отдел
Василькова	24000	Второй отдел
Соколова	38000	Третий отдел

Создадим новую таблицу с итогами по всем исходным столбцам: количество менеджеров, количество отделов и сумма всех продаж.

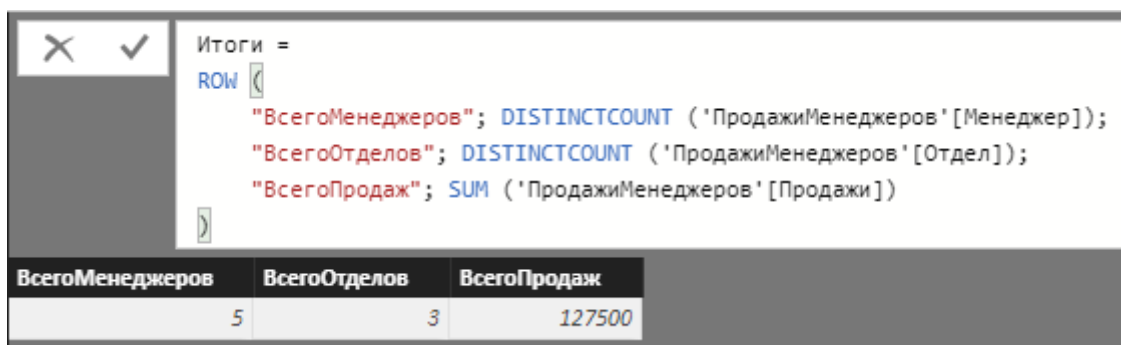
Для этого, в [Power BI Desktop](#) во вкладке «Моделирование» создадим вычисляемую таблицу по следующей формуле с участием функции ROW, которая и создаст итоги по исходным столбцам:

```
Итоги =  
ROW (  
    "ВсегоМенеджеров"; DISTINCTCOUNT ('ПродажиМенеджеров'[Менеджер]);  
    "ВсегоОтделов"; DISTINCTCOUNT ('ПродажиМенеджеров'[Отдел]);  
    "ВсегоПродаж"; SUM ('ПродажиМенеджеров'[Продажи])  
)
```

На основе этой формулы функция ROW создаст вычисляемую таблицу в модели данных Power BI, состоящую из 3 столбцов:

- значения столбца [ВсегоМенеджеров] будут рассчитываться при помощи функции [DISTINCTCOUNT](#), которая посчитает уникальное количество значений в исходном столбце [Менеджер]
- тоже самое и со вторым столбцом [ВсегоОтделов]
- значения третьего столбца [ВсегоПродаж] будут рассчитаны с помощью другой [DAX функции](#) — [SUM](#), которая сложит все цифровые значения из исходного столбца [Продажи], то есть, посчитает итоговую сумму всех продаж

Результатом выполнения этой [формулы](#) на основе работы DAX функции ROW, будет следующая таблица итогов:



The screenshot shows the DAX formula bar with the following formula:

```
Итоги =  
ROW (  
    "ВсегоМенеджеров"; DISTINCTCOUNT ('ПродажиМенеджеров'[Менеджер]);  
    "ВсегоОтделов"; DISTINCTCOUNT ('ПродажиМенеджеров'[Отдел]);  
    "ВсегоПродаж"; SUM ('ПродажиМенеджеров'[Продажи])  
)
```

Below the formula bar, a table is displayed with the following data:

ВсегоМенеджеров	ВсегоОтделов	ВсегоПродаж
5	3	127500

## DAX функция TOPN

TOPN () — создает таблицу из первых N строк исходной таблицы, с возможностью выбора столбца исходной таблицы, по которой будут выбираться первые N строк.

Синтаксис:

TOPN (N строк; 'Таблица'; [Столбец]; Порядок Сортировки)

Где:

- N строк — число первых строк (если 0, то возвратится пустая таблица)
- 'Таблица' — исходная таблица
- [Столбец] — столбец, по которому выбирается количество N строк (необязательный параметр)
- Порядок Сортировки — порядок сортировки выбора N строк (ASC — по возрастанию, DESC — по убыванию)

! — Результаты в самой выводимой таблице по выводу не сортируются

! — При наличии одинаковых значений в исходной таблице, в создаваемой таблице выводятся все строки с одинаковыми значениями

Пример формулы на основе DAX функции TOPN.

В Power BI имеется та же исходная таблица «Продажи Менеджеров», пример на основе которой, мы разбирали выше:

Менеджер	Продажи	Отдел
Смирнов	18000	Первый отдел
Сидоров	31000	Первый отдел
Колесников	16500	Второй отдел
Василькова	24000	Второй отдел
Соколова	38000	Третий отдел

Создадим таблицу с топ 3 максимальными продажами. Для этого воспользуемся функцией TOPN и напишем следующую формулу:

```
Топ3МаксимальныеПродажи =  
TOPN (  
    3;  
    'ПродажиМенеджеров';  
    'ПродажиМенеджеров'[Продажи];  
    DESC  
)
```

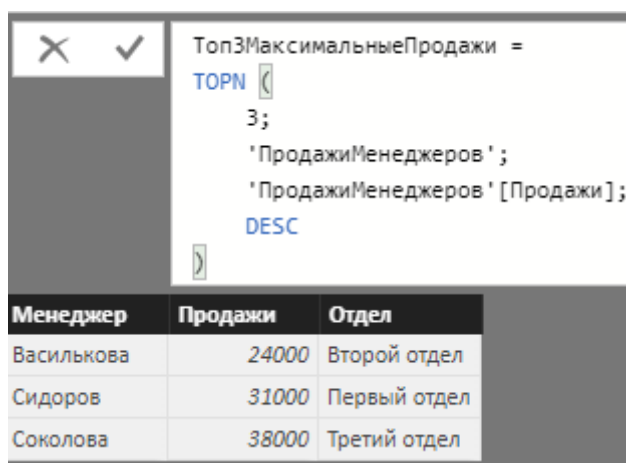
В этой формуле первый параметр функции TOPN имеет значение 3 — то есть, выведется 3 строки.

Второй параметр описывает название исходной таблицы.

Третий параметр указывает на исходный столбец, по которому нужно будет произвести сортировку для выбора 3 строк.

Четвертый параметр содержит служебное слово DESC, которое означает сортировку выборки по убыванию.

В итоге, на основании всех указанных параметров DAX функции TOPN, создастся новая таблица из 3 строк с самыми максимальными значениями в столбце [Продажи] исходной таблицы «Продажи Менеджеров»:



Топ3МаксимальныеПродажи = TOPN 3; 'ПродажиМенеджеров'; 'ПродажиМенеджеров'[Продажи]; DESC		
Менеджер	Продажи	Отдел
Василькова	24000	Второй отдел
Сидоров	31000	Первый отдел
Соколова	38000	Третий отдел

## DAX функция EXCEPT

EXCEPT () — создает таблицу из строк левой таблицы, которых нет в правой. В обеих таблицах должна быть идентичная структура столбцов.

Синтаксис:

EXCEPT ('Левая Таблица'; 'Правая Таблица')

Пример формулы с использованием DAX функции EXCEPT.

В модели данных Power BI имеются 2 таблицы с одинаковой структурой столбцов «Продажи Менеджеров» и «Топ 3 Максимальные Продажи»:

Менеджер	Продажи	Отдел
Сидоров	31000	Первый отдел
Петров	15000	Второй отдел
Свиридова	35000	Второй отдел
Колесников	18000	Второй отдел
Воснецова	25000	Первый отдел

Менеджер	Продажи	Отдел
Свиридова	35000	Второй отдел
Сидоров	31000	Первый отдел
Воснецова	25000	Первый отдел

Задача — создать новую таблицу с оставшимися менеджерами и их продажами, не входящих в Топ 3. То есть, нужно создать таблицу из строк первой таблицы, которых нет во второй. И с решением данной задачи нам отлично поможет DAX функция EXCEPT.

Напишем соответствующую формулу с участием EXCEPT:

Оставшиеся Продажи = EXCEPT ('ПродажиМенеджеров'; 'Топ3МаксПродажи')

Результатом выполнения этой формулы с функцией EXCEPT, будет являться новая таблица со строками из «Продажи Менеджеров», которых нет в «Топ 3 Макс Продажи»:



ОставшиесяПродажи = EXCEPT ( 'ПродажиМенеджеров' ; 'Топ3МаксПродажи' )		
Менеджер	Продажи	Отдел
Петров	15000	Второй отдел
Колесников	18000	Второй отдел

## DAX функция INTERSECT

INTERSECT () — создает таблицу из строк левой таблицы, которые присутствуют в правой таблице (пересечение строк двух таблиц с сохранением дубликатов). В обеих таблицах должна быть идентичная структура столбцов.

Синтаксис:

INTERSECT ('Левая Таблица'; 'Правая Таблица')

Пример формулы с использованием DAX функции INTERSECT.

В модели данных имеются 2 таблицы с одинаковой структурой столбцов «Города Где Прибыль Больше 1 млн» и «Города Где Магазинов Меньше 5»:

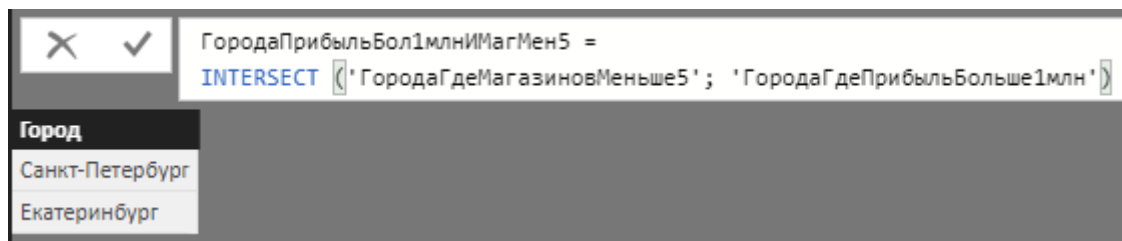
Город	Город
Москва	Санкт-Петербург
Санкт-Петербург	Екатеринбург
Екатеринбург	Новосибирск
	Астрахань

Требуется создать таблицу, в которой должна быть информация о городах, имеющих общую прибыль по городу более 1 млн и при этом количество магазинов в городе меньше 5. То есть, нам нужно создать таблицу из строк первой таблицы, которые есть во второй. А для этого, хорошо подойдет рассматриваемая DAX функция INTERSECT.

Напишем пример формулы на основе INTERSECT:

ГородаПрибыльБол1млнИМагМен5 =  
INTERSECT ('ГородаГдеМагазиновМеньше5'; 'ГородаГдеПрибыльБольше1млн')

И итог работы данной формулы на основе DAX функции INTERSECT следующий:



То есть, INTERSECT создала в модели данных таблицу, состоящую из двух строк: города Санкт-Петербург и Екатеринбург, в которых общей прибыли больше 1 млн, и при этом, магазинов в каждом городе менее 5.

# DAX функция UNION

UNION () — создает новую таблицу, объединяя любое количество таблиц с единой структурой столбцов. То есть, объединяет строки нескольких одинаковых таблиц в одну единую таблицу.

Синтаксис:

UNION ('Таблица 1'; 'Таблица 2'; ...; 'Таблица N')

Пример [формулы](#) с использованием DAX функции UNION.

Примеры мы будем рассматривать в [Power BI](#), так как в этой программе имеется возможность физического создания вычисляемых таблиц в модели данных — что нам нужно для наглядности демонстрации примеров.

В надстройке Excel (Powerpivot) в самой модели данных вычисляемые таблицы создавать нельзя. Они в Power Pivot создаются только виртуально, во время самого вычисления формулы.

Итак, в [Power BI Desktop](#) имеется несколько исходных таблиц с одинаковой структурой столбцов — «Продажи Отдел 1», «Продажи Отдел 2», «Продажи Отдел 3»:

Менеджер	Продажи	Отдел
Сидоров	31000	Первый отдел
Смирнов	18000	Первый отдел

Менеджер	Продажи	Отдел
Колесников	16500	Второй отдел
Василькова	24000	Второй отдел

Менеджер	Продажи	Отдел
Соколова	38000	Третий отдел

Объединим все эти таблицы в одну общую при помощи DAX функции UNION, создав во вкладке «Моделирование» в Power BI Desktop, вычисляемую таблицу на основе следующей формулы:

Общие Продажи = UNION ('ПродажиОтдел1'; 'ПродажиОтдел2'; 'ПродажиОтдел3')

Как итог вычисления этой формулы на основе UNION, будет создана общая таблица по продажам всех отделов:

ОбщиеПродажи = UNION ( 'ПродажиОтдел1' ; 'ПродажиОтдел2' ; 'ПродажиОтдел3' )		
Менеджер	Продажи	Отдел
Смирнов	18000	Первый отдел
Сидоров	31000	Первый отдел
Колесников	16500	Второй отдел
Василькова	24000	Второй отдел
Соколова	38000	Третий отдел

# DAX функция

## NATURALINNERJOIN

NATURALINNERJOIN () — создает новую таблицу, объединяя столбцы левой и правой таблицы и выводит только те строки, которые имеют одно и то же значение. [Функция](#) работает на основе внутренних DAX связей в Power BI или Power Pivot.

Синтаксис:

NATURALINNERJOIN ('Левая Таблица'; 'Правая Таблица')

! — Названия столбцов в левой и правой таблицах должны быть разными ! — Обе таблицы должны быть объединены внутренней связью в DAX

Пример формулы на основе DAX функции NATURALINNERJOIN.

В [Power BI Desktop](#) имеются две исходные таблицы «Общие Продажи» и «Справочник Менеджеры»:

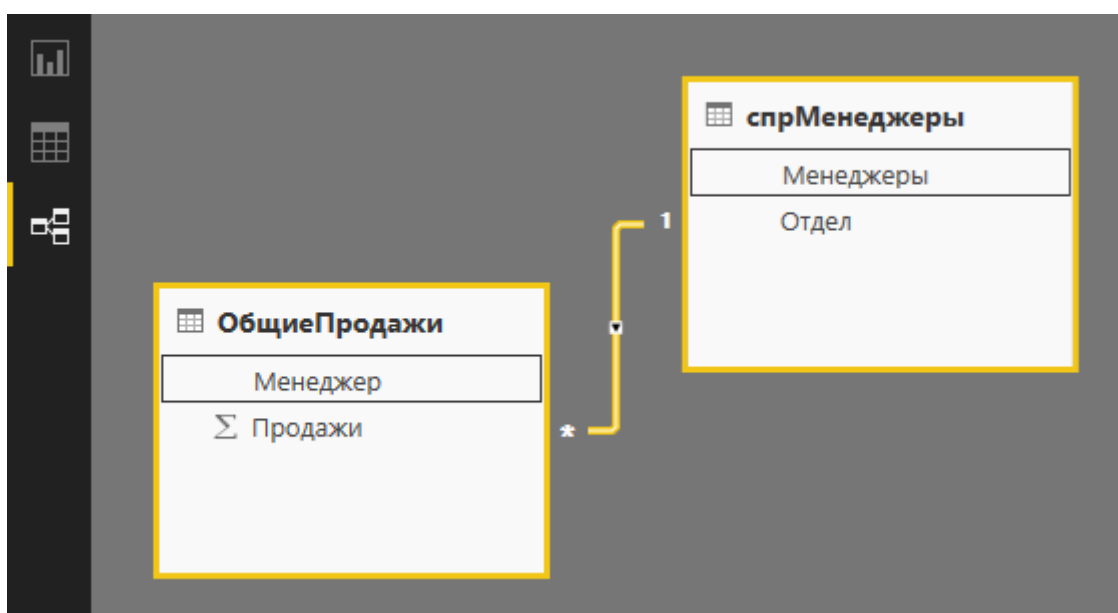
Менеджер	Продажи
Петров	18000
Петров	40000
Сидоров	15000
Соколова	32000
Сидоров	10000
Петров	30000

Менеджеры	Отдел
Петров	Первый отдел
Сидоров	Второй отдел
Соловьева	Второй отдел

В первой перечислены все продажи, которые совершаются в организации менеджерами и директором. В данном случае директором является Соколова.

Во второй перечислены все менеджеры организации. Так как Соколова не является менеджером, то она отсутствует в этом справочнике менеджеров.

Обе таблицы связаны внутренней связью DAX в Power BI:



Задача: создать новую таблицу с продажами отделов (только по менеджерам, исключая продажи директора). То есть, нужно создать таблицу из общих строк первой таблицы «Общие Продажи», и второй «Справочник Менеджеры».

С этой задачей легко справится функция `NATURALINNERJOIN`, так как между таблицами настроена внутренняя связь в DAX.

Напишем соответствующую [формулу](#) с использованием `NATURALINNERJOIN`:

**Продажи Отделов = `NATURALINNERJOIN` ('спрМенеджеры'; 'ОбщиеПродажи')**

Результатом выполнения этой формулы будет новая вычисляемая таблица в Power BI Desktop:

ПродажиОтделов = <code>NATURALINNERJOIN</code> ('спрМенеджеры'; 'ОбщиеПродажи')			
Менеджеры	Отдел	Менеджер	Продажи
Петров	Первый отдел	Петров	18000
Петров	Первый отдел	Петров	30000
Петров	Первый отдел	Петров	40000
Сидоров	Второй отдел	Сидоров	10000
Сидоров	Второй отдел	Сидоров	15000

В созданной таблице мы уже не увидим продажи директора Соколовой, так как NATURALINNERJOIN возвращает только те строки, которые есть в обеих исходных таблицах. А Соколова содержится только в одной.

Аналогично, мы не увидим менеджера Соловьеву, так как у нее нет ни одной продажи.



## DAX функция

# NATURALLEFTOUTERJOIN

NATURALLEFTOUTERJOIN () — создает новую таблицу, объединяя столбцы левой и правой таблицы и выводит все строки из правой таблицы, а также из левой, которые имеют одно и то же значение с правой. Функция работает на основе внутренних DAX связей в Power BI или Power Pivot.

Синтаксис:

**NATURALLEFTOUTERJOIN ('Левая Таблица'; 'Правая Таблица')**

! — Названия столбцов в левой и правой таблицах должны быть разными

! — Обе таблицы должны быть объединены внутренней связью в DAX

Пример формулы на основе DAX функции NATURALLEFTOUTERJOIN.

Рассмотрим пример на основе все тех же двух исходных таблиц «Общие Продажи» и «Справочник Менеджеры», которые мы рассматривали в примере выше.

Только теперь у нас задача несколько иная — вывести всех менеджеров из всех отделов и их продажи, если они есть. То есть, нам в новой таблице нужно вывести все строки из таблицы «Справочник Менеджеры», а также, все строки из «Общие Продажи», которые соответствуют «Справочник Менеджеры».

Формула с использованием DAX функции NATURALLEFTOUTERJOIN будет следующей:

**ПродажиОтделов = NATURALLEFTOUTERJOIN ('спрМенеджеры'; 'ОбщиеПродажи')**

И результат выполнения этой формулы будет такой:

✕ ✓		ПродажиОтделов = <b>NATURALLEFTOUTERJOIN</b> ( 'спрМенеджеры' ; 'ОбщиеПродажи' )	
Менеджеры	Отдел	Менеджер	Продажи
Петров	Первый отдел	Петров	18000
Петров	Первый отдел	Петров	30000
Петров	Первый отдел	Петров	40000
Сидоров	Второй отдел	Сидоров	10000
Сидоров	Второй отдел	Сидоров	15000
Соловьева	Второй отдел		

В отличие от первого примера, где мы рассматривали функцию NATURALINNERJOIN, в данном случае, NATURALLEFTOUTERJOIN вывела еще одну строку с менеджером Соловьева и ее пустыми продажами. Так как в таблице «Справочник Менеджеров» этот менеджер есть, а продаж в таблице «Общие Продажи» — нет.

Аналогично с директором Соколовой — NATURALLEFTOUTERJOIN также не вывела ее продажи, так как Соколовой нет в справочнике по менеджерам.

## DAX функция GENERATESERIES

GENERATESERIES () — создает таблицу последовательных значений (ряд чисел).

Синтаксис:

GENERATESERIES (Стартовое Число; Конечное Число; Инкремент)

Где:

- Стартовое Число — стартовое число последовательности
- Конечное Число — конечное число последовательности
- Инкремент — приращение (шаг, увеличение) последовательности (! — инкремент должен быть больше 0)

Пример формулы на основе [DAX функции](#) GENERATESERIES.

Создадим в [Power BI Desktop](#) во вкладке «Моделирование» вычисляемую таблицу на основе следующей формулы с использованием функции GENERATESERIES:

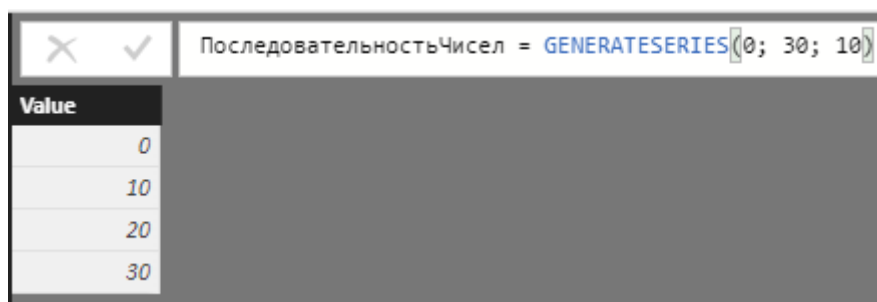
Последовательность Чисел = GENERATESERIES (0; 30; 10)

В первом параметре мы указали значение = 0, соответственно, наш ряд чисел начнется с 0.

Во втором параметре значение 30, соответственно, ряд чисел закончится числом 30.

И в третьем параметре прописали инкремент = 10, то есть, каждое новое число будет больше предыдущего на 10.

Как итог вычисления этой формулы на основе GENERATESERIES, в модели данных [Power BI](#) создастся таблица со столбцом ряда чисел:



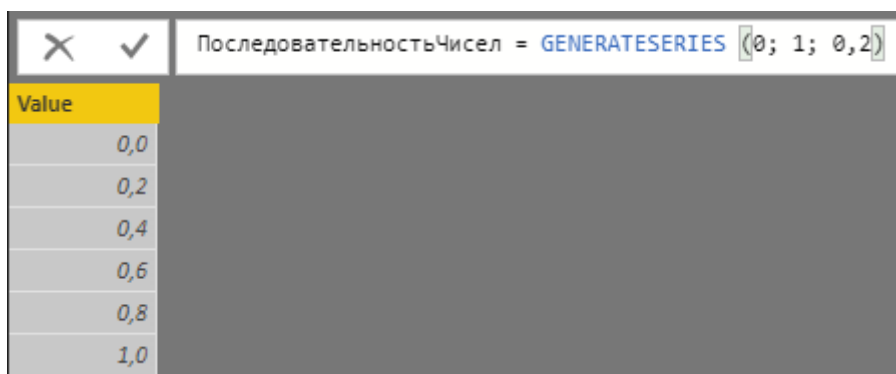
ПоследовательностьЧисел = GENERATESERIES(0; 30; 10)

Value
0
10
20
30

Для закрепления практики, напишем еще одну [DAX формулу](#):

Последовательность Чисел = GENERATESERIES (0; 1; 0,2)

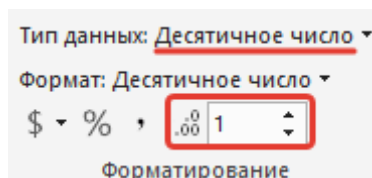
Исходя из прописанных параметров, у нас должен получиться ряд чисел начиная от 0, заканчивая 1, с инкрементом 0.2, то есть каждое последующее число будет больше предыдущего на 0.2:



ПоследовательностьЧисел = GENERATESERIES (0; 1; 0,2)

Value
0,0
0,2
0,4
0,6
0,8
1,0

Единственное, в этом примере нужно не забыть у созданного столбца поменять тип данных на «Десятичное число». Так как при создании, возможно, тип данных будет «Целое число». А также, число десятичных знаков поставить 1, вместо 0:



# DAX функция CROSSJOIN

CROSSJOIN () — создает таблицу из всех пересечений строк всех таблиц, входящих в параметры функции.

Синтаксис:

CROSSJOIN ('Таблица 1'; 'Таблица 2'; 'Таблица N')

Где: Таблица — исходная таблица или табличное выражение

! — Названия столбцов во всех таблицах должны быть разными

Рассмотрим пример формулы на основе DAX функции CROSSJOIN.

В Power BI Desktop имеются 3 исходные таблицы «Год», «Месяц», «День»:

Год	Месяцы	Дни
2018	январь	1
2019	февраль	2

Напишем формулу календаря из пересечения всех строк года, месяца и дня на основе функции CROSSJOIN:

Календарь = CROSSJOIN ('Год'; 'Месяц'; 'День')

Результатом выполнения этой формулы будет следующее пересечение всех строк:

Календарь = CROSSJOIN ('Год'; 'Месяц'; 'День')		
Год	Месяцы	Дни
2018	январь	1
2018	февраль	1
2018	январь	2
2018	февраль	2
2019	январь	1
2019	февраль	1
2019	январь	2
2019	февраль	2

# DAX функции GENERATE и GENERATEALL

GENERATE () и GENERATEALL () — создают таблицу из всех пересечений строк двух таблиц, входящих в параметры функций. То есть, для каждой строки первой таблицы возвращаются (по очереди вычисляются) все строки из второй.

Иногда, на просторах Интернета я встречаю, что некоторые пользователи несколько неверно прописывают имя этих функций, добавляя второе слово: generate table. Слово table здесь не нужно.

Синтаксис:

GENERATE ('Таблица 1'; 'Таблица 2')

GENERATEALL ('Таблица 1'; 'Таблица 2')

Где: Таблица — исходная таблица или табличное выражение

! — Названия столбцов в обеих таблицах должны быть разными

Обе функции GENERATE и GENERATEALL работают практически одинаково, за исключением того, как они обрабатывают пустую вторую таблицу.

Давайте сначала разберемся как они обе функционируют, а затем рассмотрим различия в их работе.

Примеры [формул](#) на основе DAX функций GENERATE и GENERATEALL.

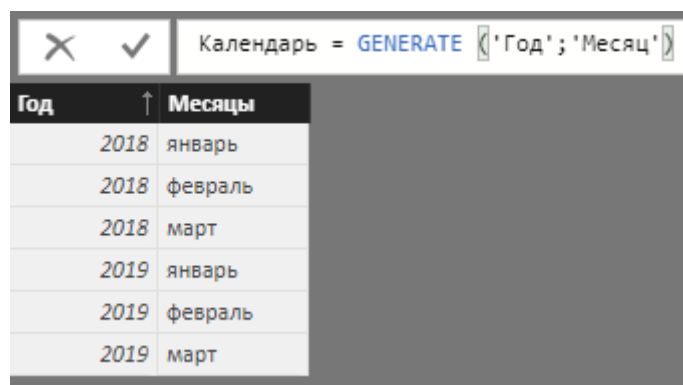
В [Power BI Desktop](#) имеются 2 исходные таблицы «Год» и «Месяцы»:

Год	Месяцы
2018	январь
2019	февраль
	март

Создадим при помощи функции GENERATE формулу, возвращающую календарь из пересечений строк всех годов со всеми имеющимися месяцами:

Календарь = GENERATE ('Год'; 'Месяц')

Результатом выполнения этой формулы будет перечисление всех 3 месяцев по каждому году:

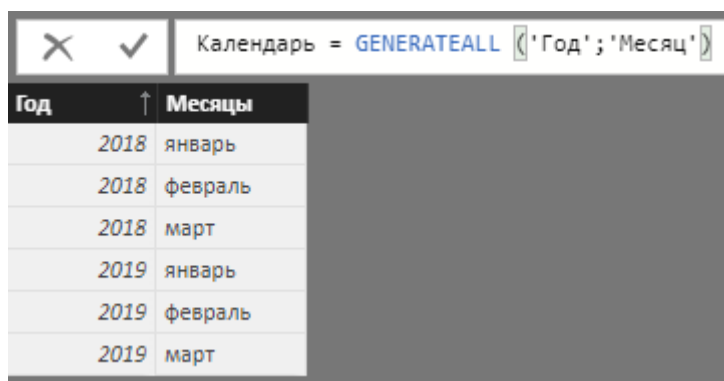


The screenshot shows the DAX formula bar with the formula: Календарь = GENERATE ('Год'; 'Месяц'). Below the formula bar, a table is displayed with two columns: 'Год' (Year) and 'Месяцы' (Months). The table contains the following data:

Год	Месяцы
2018	январь
2018	февраль
2018	март
2019	январь
2019	февраль
2019	март

Итог будет абсолютно тем же самым, если вместо GENERATE мы пропишем DAX функцию GENERATEALL:

Календарь = GENERATEALL ('Год'; 'Месяц')



The screenshot shows the DAX formula bar with the formula: Календарь = GENERATEALL ('Год'; 'Месяц'). Below the formula bar, a table is displayed with two columns: 'Год' (Year) and 'Месяцы' (Months). The table contains the following data:

Год	Месяцы
2018	январь
2018	февраль
2018	март
2019	январь
2019	февраль
2019	март

Теперь давайте поговорим о разнице этих 2 функций. А вся разница, как я уже писал выше, заключается в обработке пустой второй таблицы.

Рассмотрим пример, когда вторая таблица «Месяц» полностью пустая:

Год	Месяцы
2018	
2019	

При тех же формулах:

Календарь = GENERATE ('Год';'Месяц')

Календарь = GENERATEALL ('Год';'Месяц')

В этот раз результат будет разным.

GENERATE — возвратит вообще пустую таблицу:

✕	✓	Календарь = GENERATE ('Год';'Месяц')
Год	↑	Месяцы

А DAX функция GENERATEALL возвратит таблицу, состоящую из двух столбцов, где будут прописаны года, а месяца останутся пустыми:

<div>✕ ✓</div> <div>Календарь = GENERATEALL ( 'Год'; 'Месяц' )</div>	
Год	Месяцы
2018	
2019	



## 9. ИНФОРМАЦИОННЫЕ ФУНКЦИИ DAX

## DAX функция ISBLANK

ISBLANK () — проверяет переданное в параметрах значение. Если значение пусто, то функция возвращает TRUE, в противном случае возвращает FALSE.

Синтаксис:

ISBLANK (Значение)

Пример формулы на основе DAX функции ISBLANK.

В [Power BI Desktop](#) имеется исходная таблица, содержащая в своих строках цифровые и пустые значения:

Столбец1
12
54

Создадим в этой таблице второй столбец на основе функции ISBLANK и при помощи нее проверим исходный столбец на наличие пустых ячеек. [Формула](#) будет такой:

Проверка = ISBLANK ([Столбец1])

В результате мы получили информацию о наличии пустой ячейки, в виде значения TRUE во второй строке:

✕ ✓ Проверка = ISBLANK ([Столбец1])	
Столбец1	Проверка
12	False
	True
54	False

## DAX функция ISNUMBER

ISNUMBER () — проверяет переданное в параметрах значение. Если значение является числом, то функция возвращает TRUE, во всех остальных случаях возвращает FALSE

Синтаксис:

ISNUMBER (Значение)

Пример формулы на основе DAX функции ISNUMBER.

Если мы рассмотрим ту же самую исходную таблицу, содержащую как числовые, так и пустые значения и проверим наличие числовых значений следующей формулой:

Проверка = ISNUMBER ([Столбец1])

то, результат будет обратный тому результату, который мы получали в примере выше. Так как, в этом случае, ISNUMBER вернет значение TRUE по тем ячейкам, где находятся числа, а где пустое значение, там ISNUMBER уже вернет FALSE:

Проверка = ISNUMBER ([Столбец1])	
Столбец1	Проверка
12	True
	False
54	True

## DAX функция ISEVEN

ISEVEN () — проверяет переданное в параметрах число. Если число четное, то функция возвращает TRUE, во всех остальных случаях возвращает FALSE

Синтаксис:

ISEVEN (Число)

Пример формулы на основе DAX функции ISEVEN.

В Power BI имеется исходная таблица, содержащая разные числовые значения:

Столбец1
-5
-2
0
1
2

Создадим в этой таблице второй столбец на основе функции ISEVEN по следующей формуле:

Проверка = ISEVEN ([Столбец1])

В результате ISEVEN проинформировала нас о том, в какой ячейке четное число (TRUE), а в какой нечетное (FALSE):

✕	✓	Проверка = ISEVEN ([Столбец1])	
Столбец1	Проверка		
-5	False		
-2	True		
0	True		
1	False		
2	True		

# DAX функции ITEXT, ISNONTEXT

ISTEXT () — проверяет переданное в параметрах значение. Если значение является текстом, то функция возвращает TRUE, во всех остальных случаях возвращает FALSE.

ISNONTEXT () — проверяет переданное в параметрах значение. Если значение не является текстом, то функция возвращает TRUE, если значение является текстом, то возвращает FALSE

То есть, функции ITEXT и ISNONTEXT обратные друг другу.

Синтаксис:

ISTEXT (Значение)

ISNONTEXT (Значение)

Примеры формул на основе DAX функций ITEXT и ISNONTEXT.

Так как в Power BI в рамках одного столбца мы не можем размещать разные типы данных (текст и не текст), то примеры формул рассмотрим на основе мер и простых единичных значений, входящих в параметры функций ITEXT и ISNONTEXT:

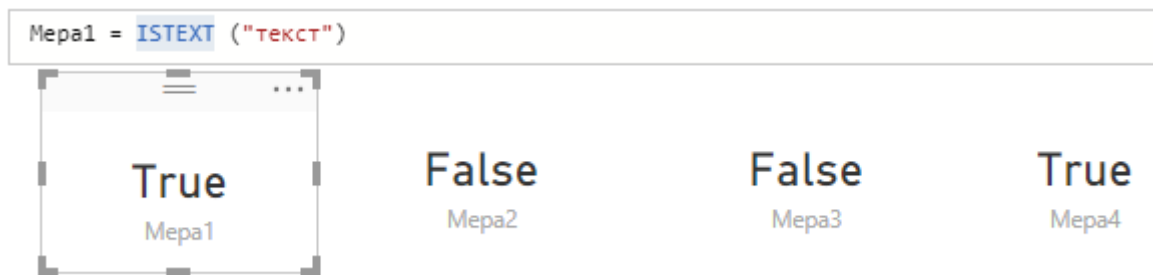
Мера 1 = ITEXT ("текст")

Мера 2 = ITEXT (123)

Мера 3 = ISNONTEXT ("текст")

Мера 4 = ISNONTEXT (123)

Результаты выполнения этих мер по каждой из функций ITEXT и ISNONTEXT, естественно, будут противоположны друг другу. Там где текст, ITEXT возвратит TRUE, а ISNONTEXT возвратит FALSE, там где число — все будет наоборот:



## DAX функция ISERROR

ISERROR () — относится к информационным функциям DAX. Она выводит значение TRUE (Истина), если значение, входящее в ее параметр, вычисляется с ошибкой, а также, значение FALSE (Ложь), если ошибок нет.

Синтаксис:

ISERROR (Значение)

Пример формулы 1: ISERROR (6 / 2)

Результат 1: FALSE (Ложь)

Пример формулы 2: ISERROR (6 / 0)

Результат 2: TRUE (Истина)

В первой формуле ISERROR выдала значение FALSE (Ложь), потому что выражение «6 / 2» вычисляется без ошибки. Тогда как, во втором случае выражение «6 / 0» вычисляется с ошибкой и поэтому ISERROR выдала значение TRUE (Истина).

Если ISERROR дополнить функцией условия «если» IF, то получится полный аналог [DAX функции](#), которую мы рассматривали выше — IFERROR:

IFERROR =

IF (

ISERROR (Выражение);

"Значение Если Ошибка"

Выражение

)

## DAX функция ISLOGICAL

ISLOGICAL () — проверяет переданное в параметрах значение. Если значение является логическим, то функция возвращает TRUE, во всех остальных случаях возвращает FALSE

Синтаксис:

ISLOGICAL (Значение)

Пример формулы на основе DAX функции ISLOGICAL.

Точно также, как и в примере выше, так как в рамках одного столбца в Power BI разные типы данных разместить нельзя (логические и не логические), рассмотрим примеры формул в рамках простых мер:

Мера 1 = ISLOGICAL ("текст")

Мера 2 = ISLOGICAL (FALSE)

То есть, мера 1 содержит текст, а мера 2 — логическое значение FALSE. Посмотрим, что ISLOGICAL возвратит в эти двух случаях:



Как мы видим, в мере 1, где содержится текст, ISLOGICAL возвратила значение FALSE, а в мере 2, где находится логическое значение, функция уже возвратила TRUE.



# DAX функции HASONEVALUE и HASONEFILTER

HASONEVALUE () — проверяет столбец и если в результате всех перекрестных фильтров в нем осталось одно единственное значение (одна строка), то функция возвращает TRUE, во всех остальных случаях — FALSE.

HASONEFILTER () — проверяет столбец и если в результате прямого фильтра в нем осталось одно единственное значение (одна строка), то функция возвращает TRUE, во всех остальных случаях — FALSE.

Синтаксис:

HASONEVALUE ([Столбец])

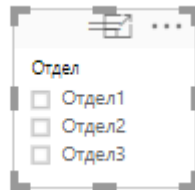
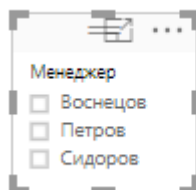
HASONEFILTER ([Столбец])

Пример [формул](#) на основе DAX функций HASONEVALUE и HASONEFILTER.

В [Power BI Desktop](#) имеется исходная таблица «Продажи Менеджеров», содержащая столбцы [Отдел], [Менеджер], [Продажи]:

Отдел	Менеджер	Продажи
Отдел1	Петров	200000
Отдел2	Сидоров	250000
Отдел3	Воснецов	100000

Давайте [в отчетах Power BI](#) создадим два среза по полям [Менеджер] и [Отдел]:



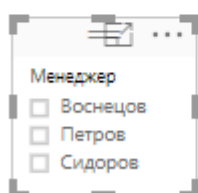
Теперь, напишем 2 формулы на основе функций HASONEVALUE и HASONEFILTER по столбцу [Менеджер], то есть, при помощи этих функций будем исследовать данное поле на количество оставшихся значений (строк) в условиях наложенных фильтров:

HASONEFILTER = HASONEFILTER ('ПродажиМенеджеров'[Менеджер])

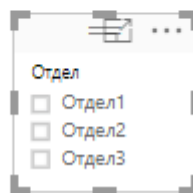
HASONEVALUE = HASONEVALUE ('ПродажиМенеджеров'[Менеджер])

Посмотрим, какие значения возвратят эти функции [языка DAX](#), когда фильтры на исходную таблицу не наложены, то есть, в столбце [Менеджер] находится более одного значения (больше одной строки):

**False**  
HASONEFILTER



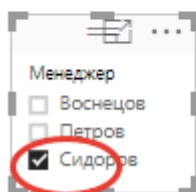
**False**  
HASONEVALUE



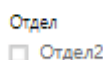
Обе функции возвращают значение FALSE, так как на столбец [Менеджер] не наложено никаких фильтров, а значит, в нем более одного значения.

Давайте поставим прямой фильтр на это поле:

**True**  
HASONEFILTER



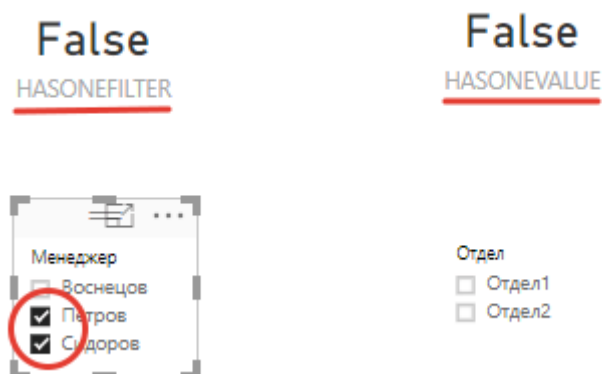
**True**  
HASONEVALUE



В результате, обе функции возвращают логическое значение TRUE, так как на столбец [Менеджер] наложен прямой фильтр, фильтрующий его до одного конкретного значения «Сидоров», а функция HASONEFILTER возвращает TRUE как раз тогда, когда наложен прямой фильтр и в столбце осталось одно конкретное значение.

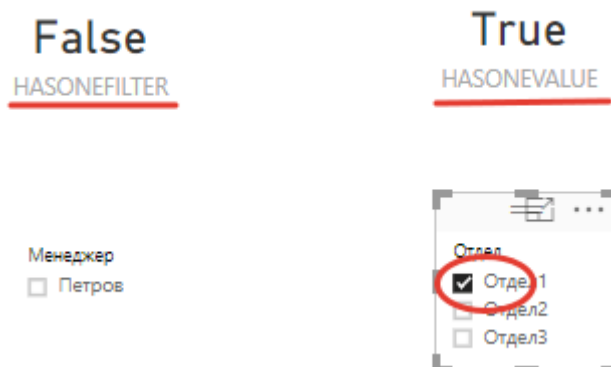
HASONEVALUE, в свою очередь, также, возвращает TRUE, так как она учитывает любые перекрестные фильтры.

Теперь, расширим этот фильтр до 2 значений «Сидоров» и «Петров»:



Как мы можем наблюдать из отчета Power BI, в этой ситуации функции HASONEVALUE и HASONEFILTER вновь возвращают логическое значение FALSE несмотря на то, что на столбец [Менеджер] до сих пор наложен прямой фильтр. Все дело в том, что этот фильтр уже оставляет не одно единственное значение, а больше (в данном случае два).

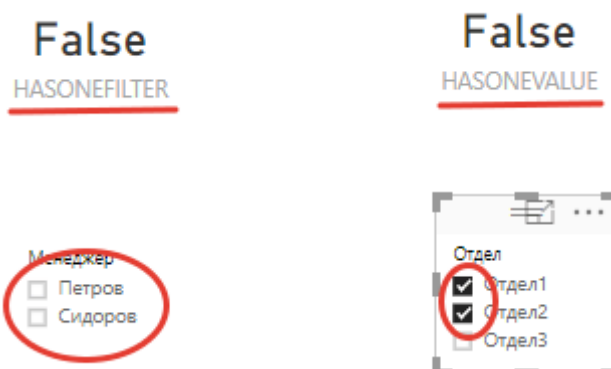
Теперь, снимем все прямые фильтры со столбца [Менеджер] и создадим срез по второму полю [Отдел]. Этим самым, столбец [Менеджер] также будет отфильтрован, но уже не прямым фильтром, а косвенным:



В данном примере значение TRUE по столбцу [Менеджер] возвращает только HASONEVALUE, так как эта функция учитывает любые перекрестные фильтры. А в этом случае, на столбец [Менеджер], как раз-таки, распространяется перекрестный фильтр от поля [Отдел] и в результате этого косвенного фильтра, в столбце [Менеджер] осталось одно значение «Петров».

А функция HASONEFILTER возвращает FALSE, даже несмотря на то, что в поле [Менеджер] всего одно значение, так как эта функция учитывает только прямые фильтры, которых на столбце сейчас нет.

Если же, мы расширим фильтр по отделам до двух значений, то косвенный фильтр, распространяющийся на столбец [Менеджер], также, будет расширен до 2 значений. И обе функции HASONEVALUE, HASONEFILTER будут вновь возвращать логическое значение FALSE, так как TRUE они возвращают именно в той ситуации, когда в результате прямых или косвенных фильтров остается одно конкретное значение (одна строка):



# DAX функции ISFILTERED и ISCROSSFILTERED

ISFILTERED () — проверяет столбец и, если он находится под воздействием прямых фильтров, возвращает логическое значение TRUE, во всех остальных случаях — FALSE.

ISCROSSFILTERED () — проверяет столбец и, если он находится под воздействием прямых или косвенных (кросс) фильтров, возвращает логическое значение TRUE, во всех остальных случаях — FALSE.

Синтаксис:

ISFILTERED ([Столбец])

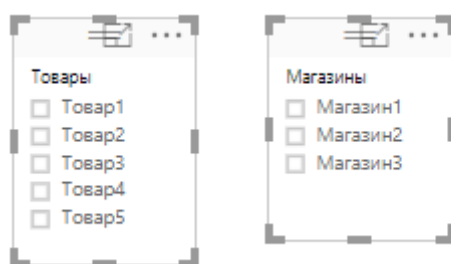
ISCROSSFILTERED ([Столбец])

Пример [формул](#) на основе DAX функций ISFILTERED и ISCROSSFILTERED.

В модели данных [Power BI Desktop](#) имеется исходная таблица «Товары По Магазинам», содержащая столбцы [Товары] и [Магазины]:

Товары	Магазины
Товар1	Магазин1
Товар2	Магазин1
Товар3	Магазин2
Товар4	Магазин2
Товар5	Магазин3

В разделе [отчетов в Power BI](#) создадим два среза по обоим полям [Товары] и [Магазины] исходной таблицы:



Теперь, напомним формулы на основе DAX функций ISFILTERED и ISCROSSFILTERED, при помощи которых, мы сможем отслеживать наличие фильтров на столбце [Товары]:

ISFILTERED = ISFILTERED ('ТоварыПоМагазинам'[Товары])

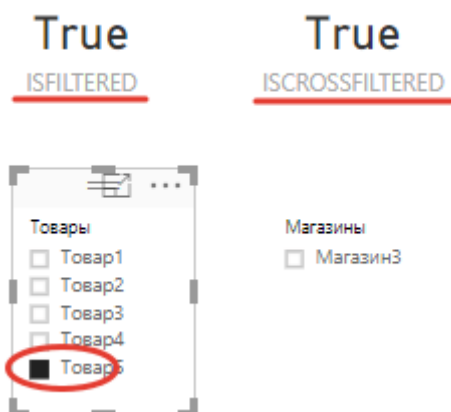
ISCROSSFILTERED = ISCROSSFILTERED ('ТоварыПоМагазинам'[Товары])

Посмотрим, какие значения возвратят эти функции при пустых фильтрах (срезах):



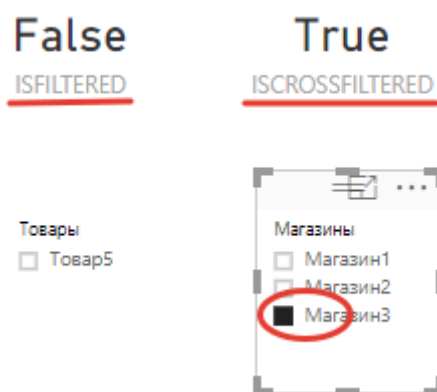
Из визуализации в Power BI видно, что обе эти функции возвратили логическое значение FALSE. Все верно, ведь на столбце [Товары] нет никаких фильтров.

Давайте исправим эту ситуацию. Создадим срез (прямой фильтр) по полю [Товары]:



В данной ситуации DAX функции ISFILTERED и ISCROSSFILTERED уже возвращают значение TRUE. И это правильно, так как, и та и другая функция учитывают прямые наложенные фильтры. А у нас, как раз, на столбец [Товары] наложен прямой фильтр.

Теперь, этот прямой фильтр уберем и отфильтруем другой столбец [Магазины]. При этом, [Товары], также отфильтруются. Но, это уже будет не прямой фильтр, а косвенный, распространяющийся от столбца [Магазины], так как оба столбца находятся в рамках одной и той же таблицы. Посмотрим какие значения ISFILTERED и ISCROSSFILTERED возвратят в этот раз:



Так как в этой ситуации на столбце [Товары] нет прямого фильтра, а только косвенный, то функция ISFILTERED уже возвращает логическое значение FALSE, так как она учитывает только прямые фильтры.

В свою очередь, DAX функция ISCROSSFILTERED по-прежнему возвращает TRUE, так как она учитывает кроме прямых фильтров, также и косвенные.

# 10. ПРОЧИЕ ФУНКЦИИ

## DAX



# DAX функции FIRSTNONBLANK и LASTNONBLANK

FIRSTNONBLANK () — возвращает первое известное значение из столбца, где вычисленное выражение является непустым.

LASTNONBLANK () — возвращает последнее известное значение из столбца, где вычисленное выражение является непустым.

Синтаксис:

FIRSTNONBLANK ([Столбец]; Выражение)

LASTNONBLANK ([Столбец]; Выражение)

Пример [формул](#) на основе DAX функций FIRSTNONBLANK и LASTNONBLANK.

В [Power BI Desktop](#) имеется исходная таблица «Продажи», в которой находится информация по дате и сумме продажи:

Дата	СуммаПродажи
15 июля 2018 г.	5000
16 июля 2018 г.	12000
17 июля 2018 г.	3000
18 июля 2018 г.	24000
19 июля 2018 г.	7000
20 июля 2018 г.	12099
21 июля 2018 г.	13000

Требуется [в отчетах Power BI](#) вывести даты первой и последней продажи. И сделать это, буквально в одно действие, можно при помощи DAX функций FIRSTNONBLANK и LASTNONBLANK. Давайте напишем соответствующие формулы мер:

```
Дата Первой Продажи = FIRSTNONBLANK ('Продажи'[Дата];  
SUM('Продажи'[СуммаПродажи]))
```

```
Дата Последней Продажи = LASTNONBLANK ('Продажи'[Дата];  
SUM('Продажи'[СуммаПродажи]))
```

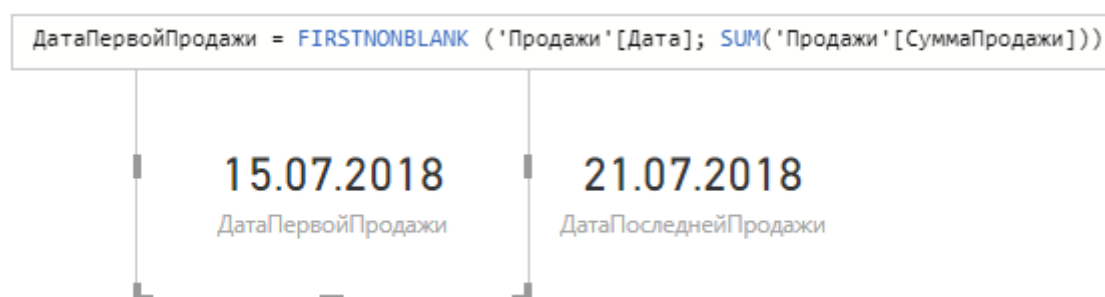
Где, в первом параметре мы прописали исходный столбец с датами продаж. Именно из этого столбца рассматриваемые функции и будут брать первые и последние известные значения.

Во втором параметре мы прописали выражение (в данном случае, сумма продаж, вычисленная при помощи функции [SUM](#)), для которого будет анализироваться результат вычисления на то, непустое ли значение получилось.

Если в рамках данного контекста, результат выражения является не пустым, то возвращается первое (последнее) известное значение из первого параметра.

Если же результат выражения в данном контексте имеет значение «пусто», то значение из первого параметра возвращаться не будет.

Итак, результатом выполнения этих формул в Power BI на основе DAX функций FIRSTNONBLANK и LASTNONBLANK, будут получены даты первой и последней продажи в рамках текущего контекста:



Если мы сравним исходную таблицу и получившиеся отчеты по датам первой и последней продажи, то увидим, что функции FIRSTNONBLANK и LASTNONBLANK отработали правильно и

вернули дату первой продажи — 15.07.2018, а дату последней продажи — 21.07.2018.

Если же, мы изменим контекст вычисления, а именно, введем фильтр дат, например от 16.07.2018 до 19.07.2018, то рассматриваемые функции FIRSTNONBLANK и LASTNONBLANK вернут соответствующие этому контексту фильтра измененные даты первой и последней продажи:

**16.07.2018**                      **19.07.2018**  
ДатаПервойПродажи                      ДатаПоследнейПродажи



## DAX функция EARLIER

EARLIER () — возвращает по указанному в параметрах столбцу, его значение на высших уровнях контекста строки.

Синтаксис:

EARLIER ([Столбец]; Номер Уровня Контекста)

Пример [формулы](#) на основе DAX функции EARLIER.

В [Power BI Desktop](#) имеется исходная таблица «Продажи Менеджеров», содержащая столбцы [Менеджер] и [Продажи]:

Менеджер	Продажи
Петров	120000
Сидоров	350000
Скворцова	185300
Воснецова	530745

Задача — создать столбец в исходной таблице с рангом каждого менеджера по сумме его продаж относительно продаж других менеджеров. Иначе говоря, нужно пронумеровать каждого менеджера, где 1-й номер — это менеджер, который принес самую максимальную сумму продаж, а 4-й номер — менеджер, который принес самую минимальную сумму продаж.

Давайте писать соответствующую формулу постепенно, шаг за шагом.

Номера рангов в этом столбце мы будем считать при помощи функции [COUNTROWS](#), которая подсчитывает количество строк в таблице. Напишем первый этап нашей формулы:

Ранг Прибыли = COUNTROWS ('ПродажиМенеджеров')

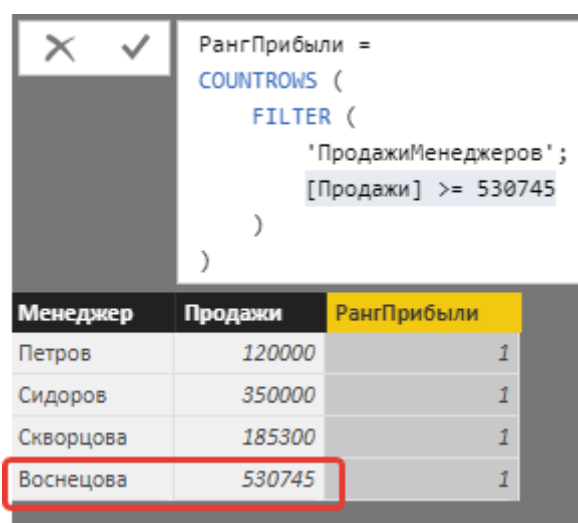
РангПрибыли = COUNTROWS ('ПродажиМенеджеров')		
Менеджер	Продажи	РангПрибыли
Петров	120000	4
Сидоров	350000	4
Скворцова	185300	4
Воснецова	530745	4

Количество строк мы получили, но, нам нужно, чтобы в каждой строке были номера от 1 до 4, а не значения 4 во всех строках. Для этого, при расчете количества строк в исходной таблице, нам эту исходную таблицу каждый раз (для каждой строки) нужно фильтровать. Например, так:

```
Ранг Прибыли =
COUNTROWS (
    FILTER (
        'ПродажиМенеджеров';
        [Продажи] >= 530745
    )
)
```

Для фильтрации исходной таблицы мы применили [еще одну функцию в языке DAX](#) — [FILTER](#). В качестве фильтра в этой функции мы прописали условие, где продажи должны быть больше или равны 530745.

Результатом выполнения этой функции будут следующие значения:

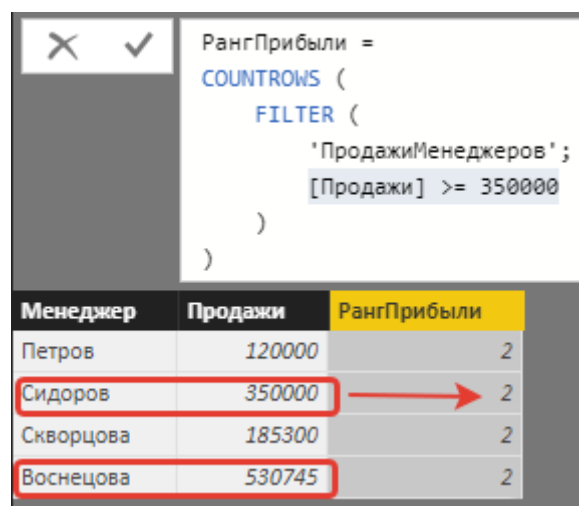


РангПрибыли =  
COUNTROWS (  
    FILTER (  
        'ПродажиМенеджеров';  
        [Продажи] >= 530745  
    )  
)

Менеджер	Продажи	РангПрибыли
Петров	120000	1
Сидоров	350000	1
Скворцова	185300	1
Воснецова	530745	1

То есть, при расчете количества строк в исходной таблице у нас эта таблица отфильтровалась до одной единственной строки, которая удовлетворяет условию фильтра «[Продажи] >= 530745». И COUNTROWS вывела количество строк, равное 1.

Если мы условие фильтра поменяем следующим образом: «[Продажи] >= 350000», то результат уже будет таким:



РангПрибыли =  
COUNTROWS (  
    FILTER (  
        'ПродажиМенеджеров';  
        [Продажи] >= 350000  
    )  
)

Менеджер	Продажи	РангПрибыли
Петров	120000	2
Сидоров	350000	2
Скворцова	185300	2
Воснецова	530745	2

В данной ситуации условию фильтра уже удовлетворяют 2 строки и поэтому, COUNTROWS вывела количество строк исходной таблицы 2.

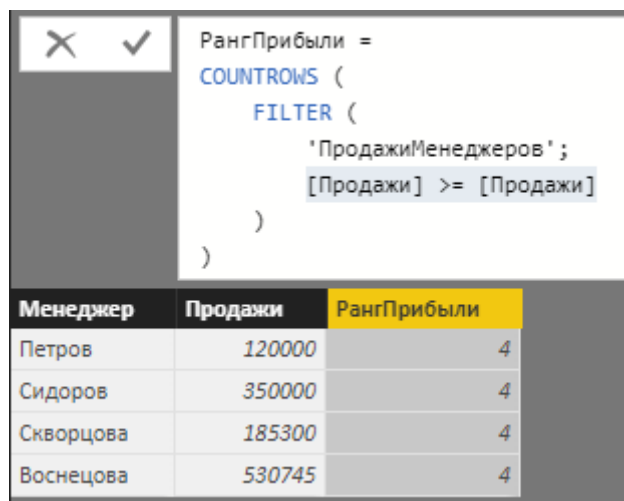
Причем имеется интересный факт из этих двух выше приведенных примеров: когда мы фильтровали таблицу по условию «[Продажи] >= 530745» (а это максимальная сумма) — то результатом подсчета строк было значение 1, когда условие изменили на «[Продажи] >= 350000» (а

это вторая по величине сумма), результат подсчета строк был уже 2. И это уже похоже на то, что нам нужно, то есть ранг менеджеров по прибыли.

Из того факта, который я выше описал, получается, что нам нужно для расчета ранга для каждой конкретной строки в условии фильтра столбец [Продажи] сравнивать с этим же столбцом [Продажи] в этой же конкретной строке. Давайте попробуем написать такую формулу:

```
Ранг Прибыли =
COUNTROWS (
    FILTER (
        'ПродажиМенеджеров';
        [Продажи] >= [Продажи]
    )
)
```

Но, к сожалению, у нас ничего не получилось:



РангПрибыли =  
COUNTROWS (  
 FILTER (  
 'ПродажиМенеджеров';  
 [Продажи] >= [Продажи]  
 )  
)

Менеджер	Продажи	РангПрибыли
Петров	120000	4
Сидоров	350000	4
Скворцова	185300	4
Воснецова	530745	4

Почему так? Потому что, когда формула считала значение для 1 строки (менеджер Петров), то DAX видел только одну эту строку. И в столбце [Продажи] было только одно значение 120000. Но, как только DAX спустился внутрь самой формулы до функции FILTER, то эта функция,

прежде чем фильтровать, возвратила полную исходную таблицу и в условие фильтра уже был подан полный столбец [Продажи].

Это эффект, так называемого уровня контекста строки. Изначально был первый уровень контекста строки, но внутри формулы функция FILTER создала второй уровень контекста строки и DAX про первый уровень полностью «забыл».

! — Еще раз повторяюсь, что для полного понимания работы функции EARLIER для начала нужно очень хорошо понимать контексты строк в DAX и уровни этих контекстов. Что, к сожалению, в рамках этой одной статьи рассмотреть просто невозможно.

Исходя из контекстов строк языка DAX, в условии фильтра столбец [Продажи], находящийся на втором уровне контекста строки, сравнивался с этим же столбцом [Продажи] опять же на втором уровне контекста строки, а нужно, чтобы возвратилось значение столбца [Продажи] с первого уровня контекста строки.

А это, как раз таки, может сделать, рассматриваемая в этой статье, функция EARLIER. Исправим нашу формулу, добавив в нее функцию EARLIER:

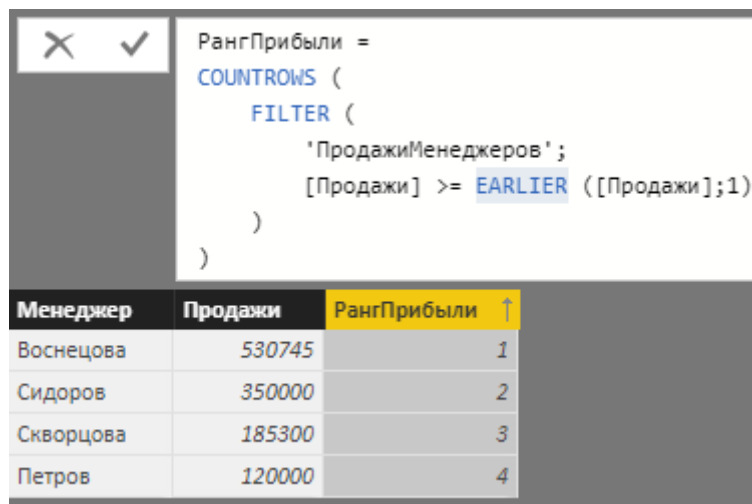
```
Ранг Прибыли =  
COUNTROWS (  
    FILTER (  
        'ПродажиМенеджеров';  
        [Продажи] >= EARLIER ([Продажи];1)  
    )  
)
```

В первый параметр функции EARLIER мы прописали столбец [Продажи], по которому нужно вернуть значения из верхнего уровня контекста строки. Во втором параметре мы прописали значение 1, то



есть, значение нужно вернуть из этого столбца на 1 уровень контекста строки выше.

И вот теперь, все заработало как надо:



The screenshot shows the DAX formula bar with the following formula:

```
РангПрибыли =  
COUNTROWS (  
    FILTER (  
        'ПродажиМенеджеров';  
        [Продажи] >= EARLIER ([Продажи];1)  
    )  
)
```

Below the formula bar is a table with the following data:

Менеджер	Продажи	РангПрибыли ↑
Воснецова	530745	1
Сидоров	350000	2
Скворцова	185300	3
Петров	120000	4

Почему? Потому, что EARLIER возвратила значение столбца [Продажи] из контекста строки на уровень выше. И значения столбца [Продажи] на втором уровне контекста строки, созданного функцией FILTER, сравнивались с единственным значением столбца [Продажи] первого уровня контекста строки, который был создан самой исходной таблицей.

Понимаю, что сходу, прочитав эту статью, понять смысл EARLIER у Вас может сразу не получиться. Попробуйте смоделировать все примеры, которые мы рассматривали в этой статье, самостоятельно. Шаг за шагом. Тогда, думаю, понимание смысла функционирования EARLIER к Вам должно прийти.

# VAR И RETURN – переменные в DAX

Любая переменная в DAX всегда состоит из обязательного синтаксиса, давайте разберем его.

Синтаксис:

VAR Name 1 = Код Переменной 1

VAR Name 2 = Код Переменной 2

RETURN

Name 1 \* Name 2

Где:

- Name — имя создаваемой переменной
- Код Переменной — любая [DAX формула](#), характеризующая саму переменную
- VAR — служебное слово (специальная функция), всегда предшествующая объявлению имени переменной при ее создании
- RETURN — служебное слово (специальная функция), оканчивающая создание (объявление) блока переменных, после него идет основной код формулы
- Name 1 \* Name 2 — основной код формулы (меры, вычисляемого столбца или таблицы)

Для чего нужны переменные в DAX (Power BI)?

Переменные:

- визуально упрощают сложный код формулы в DAX
- уменьшают время расчета сложного кода, который имеет повторяемые участки, так как переменная рассчитывается

только 1 раз и ей можно заменить повторяемые участки кода в основной формуле

- «запоминают контекст строки», в котором переменная была вычислена

Рассмотрим применение переменных (VAR и RETURN) в Power BI на основе очень простого, но показательного примера DAX формулы.

В [Power BI Desktop](#) имеется исходная таблица «Товары», содержащая столбцы [Товар], [Затраты], [Количество] и [Цена]:

Товар	Затраты	Количество	Цена
Товар1	300	100	1000
Товар2	500	50	1500
Товар1	350	10	1000

Необходимо в этой таблице рассчитать прибыль по каждой строке товара. Прибыль равняется цене за вычетом затрат, помноженная на количество. Напишем эту простую формулу:

Прибыль =  
([Цена] - [Затраты])  
\* [Количество]

Данная формула состоит из двух простейших действий:

1. ([Цена] — [Затраты]), то есть, в этом действии высчитывается прибыль по одной единице товара
2. \* [Количество], то есть, результат первого действия (прибыль по одной единице товара) помножается на количество товара в конкретной строке таблицы

Посмотрим, как эта формула отработала в Power BI:

<div> <span>✕</span> <span>✓</span> <div> Прибыль =  ([Цена] - [Затраты])  * [Количество] </div> </div>				
Товар	Затраты	Количество	Цена	Прибыль
Товар1	300	100	1000	70000
Товар2	500	50	1500	50000
Товар1	350	10	1000	6500

Теперь, давайте изменим эту формулу, применив в ней переменные.

Скажу сразу, что на самом деле, здесь переменные вовсе не нужны, но на этом простом примере очень легко понять функционирование переменных в DAX.

Итак, формула с использованием переменной в Power BI будет такая:

```

Прибыль =
VAR p1 = [Цена] - [Затраты]
RETURN
p1 * [Количество]
    
```

В данной формуле мы создали одну переменную. Ее имя — p1.

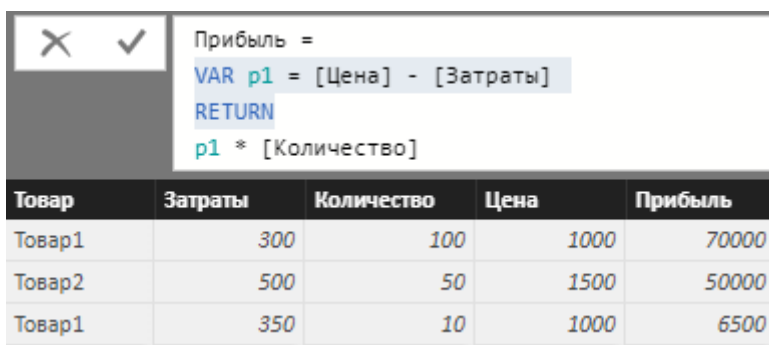
Перед созданием (объявлением) самой переменной, обязательно нужно прописать функцию (служебное слово) VAR, что мы и сделали. Дословно, VAR говорит движку DAX, что далее пойдет имя и код переменной.

И в самом деле, после VAR мы написали имя переменной p1, а затем, через знак равно (=) код самой переменной: [Цена] — [Затраты].

Далее, как мы закончили описывать все наши переменные (а у нас она всего одна), мы прописали функцию (служебное слово) RETURN, которое сообщает движку DAX, что все коды переменных закончились и далее пойдет основной код формулы вычисляемого столбца.

В основном коде вычисляемого столбца, после RETURN мы столбец [Количество] помножили на переменную p1. А переменная p1 это не что иное, как [Цена] — [Затраты].

В итоге, в Power BI вычисляемый столбец на основе формулы с использованием переменных имеет ровно те же значения, что и без них:



The screenshot shows the DAX formula bar with the following code:

```
Прибыль =
VAR p1 = [Цена] - [Затраты]
RETURN
p1 * [Количество]
```

Below the formula bar is a table with the following data:

Товар	Затраты	Количество	Цена	Прибыль
Товар1	300	100	1000	70000
Товар2	500	50	1500	50000
Товар1	350	10	1000	6500

Теперь, пойдем еще дальше и объявим вторую переменную:

```
Прибыль =
VAR p1 = [Цена] - [Затраты]
VAR p2 = [Количество]
RETURN
p1 * p2
```

В этой новой версии формулы мы создали вторую переменную с именем p2. Как и должно быть по правилу синтаксиса, перед созданием (объявлением) второй переменной мы прописали функцию (служебное слово) VAR. Далее имя переменной, знак равно и сам код переменной, который равен значениям столбца [Количество].

После того, как были объявлены все переменные (в нашем случае их две) мы прописали RETURN, которое, напоминаю, говорит о том, что объявление переменных закончено и далее пойдет основной код меры, вычисляемого столбца или таблицы. Как, собственно, и в нашей формуле — далее пошел основной код нашего вычисляемого столбца [Прибыль].

А основной код у нас стал совсем простым: переменная p1 помножается на переменную p2, то есть, результат разницы столбцов [Цена] и [Затраты] помножается на значения столбца [Количество].

Опять же, в Power BI эта версия формулы вычисляемого столбца [Прибыль] с двумя переменными имеет точно такой же результат, как и ранее:

<div>Прибыль = VAR p1 = [Цена] - [Затраты] VAR p2 = [Количество] RETURN p1 * p2</div>				
Товар	Затраты	Количество	Цена	Прибыль
Товар1	300	100	1000	70000
Товар2	500	50	1500	50000
Товар1	350	10	1000	6500

Как мы видим, из всего примера, который разобрали выше, переменные действительно помогают разложить некую сложную формулу на простые части. В данном случае, у нас формула и так была простая, но в рамках обучения, мы ее сумели еще разложить на 2 простые части (2 переменные).

Напоминаю, что это не единственная польза переменных. Например, в сложном коде могут встречаться одни и те же части, повторяемые несколько раз. И использование переменных позволяет не только сократить длину кода, написав этот код один раз в переменной, и далее просто ссылаться на имя этой переменной, так и сократить время вычисления этой большой формулы. Так как повторяемый код, прописанный переменной вычисляется только один раз, а далее, при вызове имени переменной из основного кода, просто уже будет возвращаться вычисленный ранее результат переменной.

## DAX функция RANKX

RANKX () — возвращает ранг выражения, вычисляемого в рамках текущего контекста исходного списка значений для каждой строки текущей таблицы.

Синтаксис:

RANKX ('Таблица'; Выражение; Значение; Порядок; Равные Значения)

Где:

- Таблица — исходная таблица или табличное выражение
- Выражение — любое выражение, вычисляемое для каждой строки исходной таблицы с целью формирования полного списка возможных значений для ранжирования
- Значение — (необязательный параметр) любое выражение, ранг которого необходимо найти. По умолчанию, если этот параметр не указан, вместо этого параметра возвращается значение выражения из текущей строки
- Порядок — (необязательный параметр) вид ранжирования параметра «Значение»: 1) DESC — упорядоченный порядок рангов по убыванию (по умолчанию); 2) ASC — по возрастанию
- Равные Значения — (необязательный параметр) способ определения ранга при наличии равных значений:
  - 1) Skip — (по умолчанию) ранг значения, идущего после ряда одинаковых значений, определяется как ранг равных значений плюс количество этих равных значений. Например, имеется 3 равных значения с рангом 5, следующее значение будет иметь ранг 8 (5+3);
  - 2) Dense — ранг значения, идущего после ряда одинаковых значений, определяется как следующий ранг. Например,

имеется 3 равных значения с рангом 5, следующее значение будет иметь ранг 6

## Пример формулы на основе DAX функции RANKX

В [Power BI Desktop](#) имеется исходная таблица «Продажи Менеджеров»:

Менеджер	Продажи
Петров	120000
Сидоров	350000
Скворцова	185300
Воснецова	530745

Требуется создать меру ранжирования менеджеров, исходя из сумм их продаж. Иначе говоря, нужно создать ранг по продажам, где, ранг 1 соответствует самой большой сумме продаж, а ранг 2 и далее, меньшим суммам продаж.

Воспользуемся рассматриваемой DAX функцией RANKX и попробуем на основе нее составить код формулы для меры [Ранг]:

```
Ранг =  
RANKX (  
    'ПродажиМенеджеров';  
    SUM ('ПродажиМенеджеров'[Продажи])  
)
```

В данной формуле, согласно синтаксису функции RANKX, мы использовали только 2 обязательных параметра, остальные (необязательные) мы не стали прописывать.

В качестве первого параметра мы прописали ссылку на исходную таблицу «Продажи Менеджеров».



Во втором параметре указали нужное нам выражение, которое должно рассчитываться для каждой строки исходной таблицы, указанной в первом параметре. А именно, произвели суммирование значений столбца [Продажи] при помощи функции [SUM](#) (функция суммирования [в языке DAX](#)).

Посмотрим, как отработала, созданная нами мера расчета ранга, [в отчетах Power BI](#):

```
Ранг =  
RANKX (  
    'ПродажиМенеджеров';  
    SUM ('ПродажиМенеджеров' [Продажи])  
)
```

Менеджер	Продажи	Ранг
Воснецова	530745	1
Петров	120000	1
Сидоров	350000	1
Скворцова	185300	1

Как мы видим из визуализации в Power BI, наша формула на основе функции RANKX отработала неверно и возвратила для каждого менеджера один и тот же ранг, равный 1.

Причины этой ошибки 2:

1. В этой визуализации при расчете конкретной ячейки ранга, значение столбца [Менеджер] является фильтром исходной таблицы.

Например, рассмотрим строку с менеджером Петров — при расчете ранга Петрова, исходная таблица будет автоматически отфильтрована только этим менеджером и сумма продаж возвратится только по одному Петрову.

Соответственно, ранг рассчитается только исходя из одного этого менеджера и, естественно, ранг будет равен 1. И так по каждому менеджеру.

Решить эту проблему мы сможем при помощи еще одной функции языка DAX — ALLSELECTED (прочитать информацию об этой функции Вы можете [в этой статье](#)), обернув в эту функцию исходную таблицу из первого параметра. Этим самым, мы удалим все фильтры, которые наложались на столбец [Менеджер] строками самой визуализации.

2. Так как выражение во втором параметре вычисляется по каждой строке исходной таблицы, указанной в первом параметре, то необходимо передать контекст этой строки из первого параметра во второй параметр.

Но, проблема в том, что функция SUM не принимает контекст строки.

Чтобы решить эту вторую проблему, нужно обернуть функцию SUM в другую DAX функцию — [CALCULATE](#), которая сможет передать контекст строки из таблицы в выражение. И тогда функция SUM рассчитает сумму продаж в рамках этого контекста строки, то есть, рассчитает сумму продаж только по одному конкретному менеджеру.

Итак, исправим нашу формулу согласно этим двум пунктам:

```
Ранг =  
RANKX (  
    ALLSELECTED ('ПродажиМенеджеров');  
    CALCULATE ( SUM ('ПродажиМенеджеров'[Продажи]))  
)
```

Испробуем эту доработанную меру в Power BI:

```

Ранг =
RANKX (
    ALLSELECTED ('ПродажиМенеджеров');
    CALCULATE ( SUM ('ПродажиМенеджеров'[Продажи]))
)

```

Менеджер	Продажи	Ранг
Воснецова	530745	1
Сидоров	350000	2
Скворцова	185300	3
Петров	120000	4

Теперь, наша мера расчета ранга по продажам менеджеров работает как надо. У менеджера Воснецовой сумма продаж составляет 530745, что является самой наивысшей суммой и поэтому у этой суммы продаж ранг 1. А у менеджера Петров самая минимальная сумма продаж, равная 120000 и именно поэтому, у его продаж ранг 4.

Если же мы хотим поменять порядок ранжирования, чтобы у самой минимальной суммы был ранг 1, а у самой максимальной суммы — ранг 4, то в параметрах функции RANKX нужно добавить 4 параметр «Порядок» и поставить у него значение ASC (по возрастанию). Тогда формула будет такой:

```

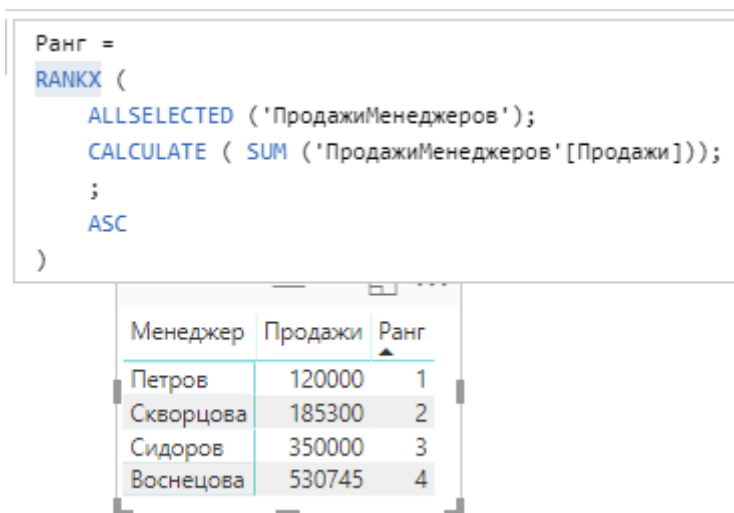
Ранг =
RANKX (
    ALLSELECTED ('ПродажиМенеджеров');
    CALCULATE ( SUM ('ПродажиМенеджеров'[Продажи]));
    ;
    ASC
)

```

Исходя из синтаксиса функции RANKX в этой формуле в параметрах мы указали 3 параметра — первый, второй и четвертый. Третий и пятый мы пропустили. Этим самым и объясняется лишняя

точка с запятой перед ASC в новой измененной формуле, то есть, это и есть пропущенный необязательный третий параметр.

Итак, посмотрим на результат в Power BI:



The screenshot shows a DAX formula in the Power BI interface. The formula is:

```
Ранг =  
RANKX (  
    ALLSELECTED ('ПродажиМенеджеров');  
    CALCULATE ( SUM ('ПродажиМенеджеров'[Продажи]));  
    ;  
    ASC  
)
```

Below the formula, a table is displayed with the following data:

Менеджер	Продажи	Ранг
Петров	120000	1
Скворцова	185300	2
Сидоров	350000	3
Воснецова	530745	4

Теперь, как мы видим, ранг 1 соответствует самой минимальной сумме продаж, а ранг 4 — самой максимальной.

## DAX функция ERROR

ERROR () — останавливает выполнение DAX кода и выводит заранее определенную пользователем ошибку (предупреждение).

Синтаксис:

ERROR («Текст ошибки»)

Пример: в [Power BI](#) имеется исходная таблица с перечислением товаров и их количеством

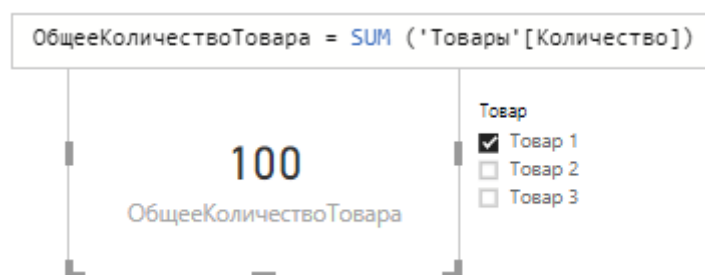
Товар	Количество
Товар 1	100
Товар 2	130
Товар 3	200

Суть задачи: создать такую меру, чтобы она всегда вычисляла общее количество товара, и пользователь не мог наложить никаких фильтров на это вычисление. Если пользователь накладывает фильтры, нужно остановить вычисление меры и в [Power BI Desktop](#) выдать пользователю ошибку (предупреждение).

Общее количество можно рассчитать при помощи DAX функции [SUM](#):

Общее Количество Товара = SUM ('Товары'[Количество])

Данная формула действительно сможет посчитать общее количество товара, но она также легко подвержена пользовательским фильтрам, что по условию задачи нам не нужно:

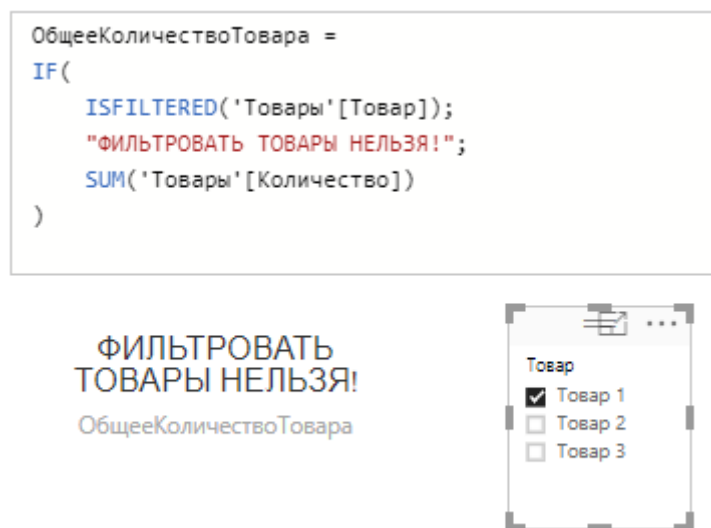


Тогда мы можем изменить формулу выше и сумму рассчитать под следующим условием: если наложен какой-либо фильтр, то выдать пользователю предупреждение, если фильтра нет, то рассчитать количество.

Все это легко решается при помощи функций [IF](#) (условия «если») и `ISFILTERED` (проверяет на наличие фильтров):

```
Общее Количество Товара =  
IF(  
    ISFILTERED('Товары'[Товар]);  
    "ФИЛЬТРОВАТЬ ТОВАРЫ НЕЛЬЗЯ!";  
    SUM('Товары'[Количество])  
)
```

Получившаяся формула вполне рабочая, если мы выберем какой-либо товар, то нам действительно выйдет предупреждение:

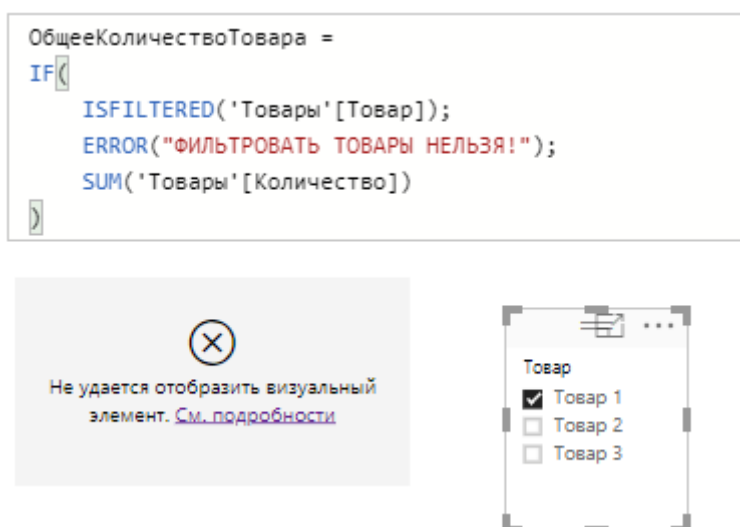


В принципе, мы задачу практически решили. Но, можно пойти еще дальше и при пользовательском фильтре не то чтобы просто вывести предупреждение, а вообще остановить работу [DAX формулы](#) и тем самым действительно обратить внимание пользователя к ошибке.

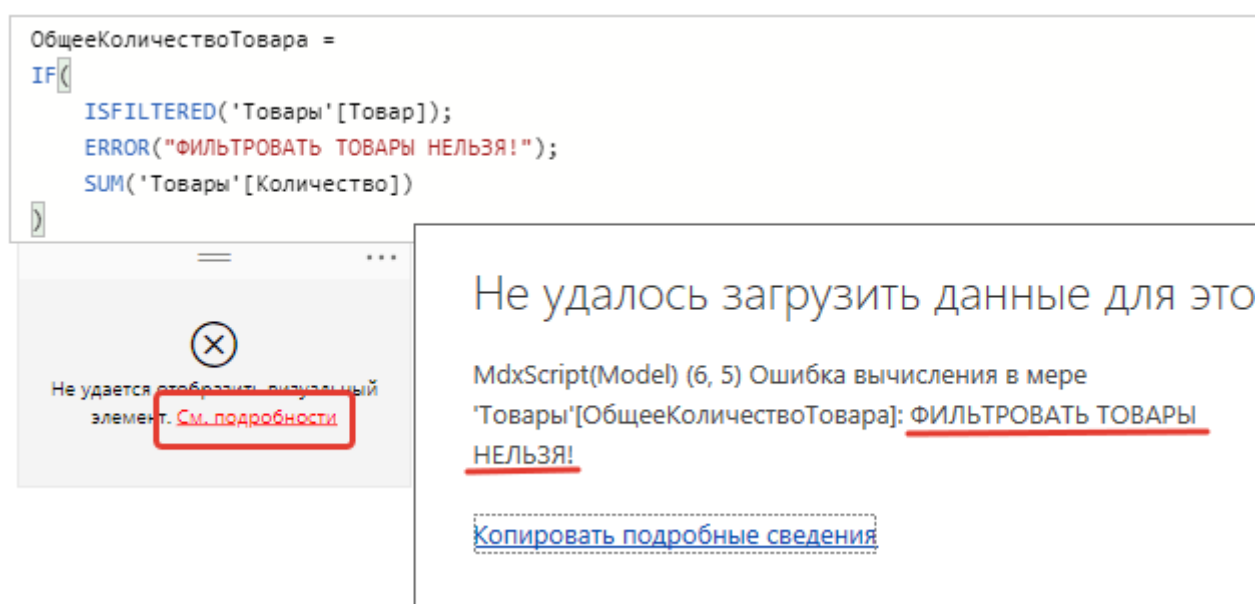
И это как раз-таки можно реализовать при помощи функции `ERROR`, прописав внутри нее наш текст предупреждения, и вставив `ERROR` вместо текста предупреждения в формуле выше:

```
Общее Количество Товара =  
IF(  
    ISFILTERED('Товары'[Товар]);  
    ERROR("ФИЛЬТРОВАТЬ ТОВАРЫ НЕЛЬЗЯ!");  
    SUM('Товары'[Количество])  
)
```

Тогда, если пользователь наложит фильтр, то DAX формула остановит свою работу:

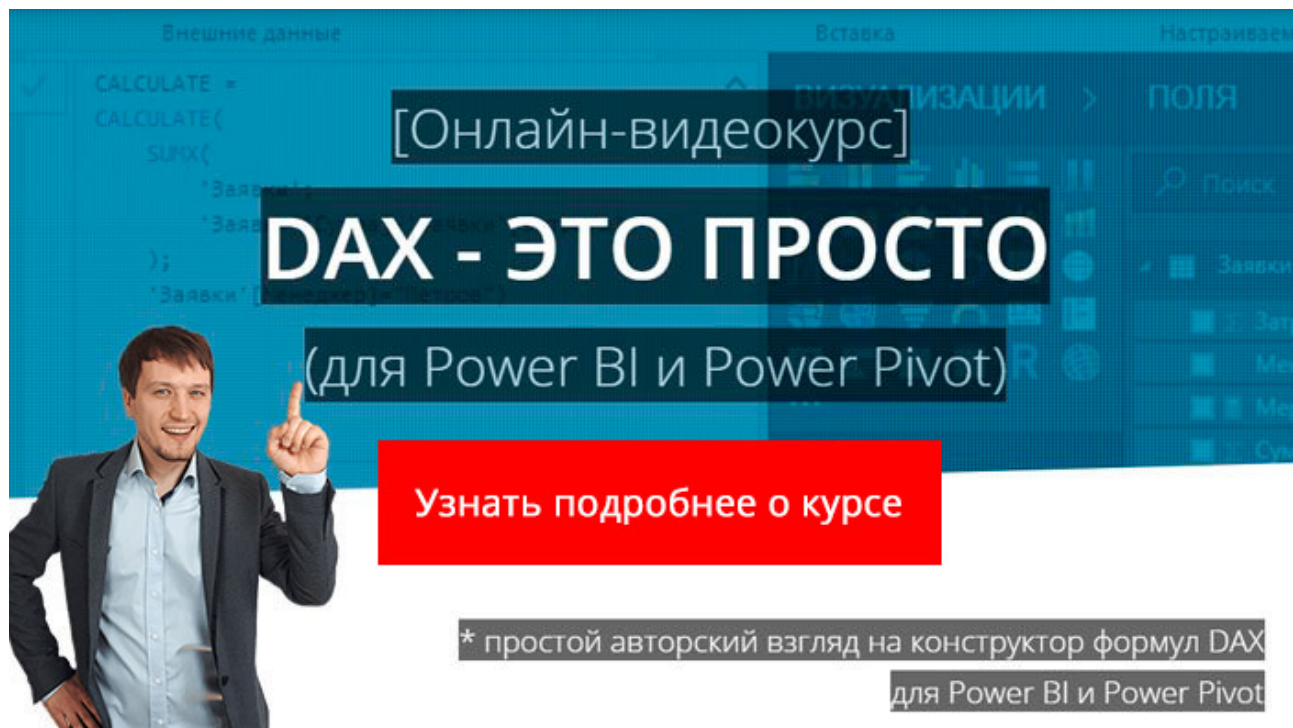


И при нажатии на визуализации в Power BI на ссылку «См. подробности», выйдет текст самого предупреждения, который мы прописывали в ERROR:



Если убрать все фильтры, то, соответственно, формула рассчитает общее количество товаров и ни каких предупреждений от ERROR не будет.





[Онлайн-видеокурс]

# DAX - ЭТО ПРОСТО

(для Power BI и Power Pivot)

[Узнать подробнее о курсе](#)

\* простой авторский взгляд на конструктор формул DAX  
для Power BI и Power Pivot