**Создание функции отмены свертывания столбцов в Power Query**

Это фрагмент книги [Гил Равив. Power Query в Excel и Power BI: сбор, объединение и преобразование данных](https://baguzin.ru/wp/?p=24451).

[Предыдущий раздел](https://baguzin.ru/wp/?p=24596) [К содержанию](https://baguzin.ru/wp/?p=24451#content)  Следующий раздел

В предыдущем разделе был приведен универсальный алгоритм отмены свертывания столбцов таблицы с уровнями иерархии N\*M. Отмена свертывания столбцов настолько полезна, что имеет смысл написать функцию. Такая функция позволит быстро обрабатывать исходные таблицы, берущие своё происхождение из сводных таблиц любой иерархии. С помощью такой функции ручное выполнение последовательности шагов объединяется в одно преобразование, которое будет пригодно для любой таблицы.



Рис. 1. Исходная таблица с иерархией 3х3

#### Создание запросов (списков) RowFields, ColumnFields и ValueField

Начнем с результатов предыдущего раздела. Загрузите книгу C07E01 - Solution.xlsx. Откройте её. Пройдите *Данные* –> *Получить данные* –> *Запустить редактор* *Power Query*. Откройте пустой запрос. Правой кнопкой мыши щелкните на панели *Запросы*, пройдите *Новый запрос* –> *Другие источники* –> *Пустой запрос*. Переименуйте новый запрос в *RowFields* и скопируйте следующий код в строку формул для создания списка полей строк:

= {"Country", "State/Region", "City"}

Создайте второй пустой запрос. Переименуйте его в *ColumnFields* и скопируйте в строку формул следующий код для создания списка полей столбцов:

= {"Color", "Parent Category", "Category"}

Важно именно так назвать новые запросы. Далее они будут использоваться в коде функции. Обратите внимание, что N представляет длину запроса *RowFields*, а буква M – длину запроса *ColumnFields*.

Создайте еще один пустой запрос. Переименуйте его в *ValueField*, и скопируйте следующий код в строку формул:

= "Revenue"

#### Удаление шагов Changed Type

Шаги *Changed Type* (*Измененный тип*) предназначены именно для текущего источника данных и не помогают обобщать запрос для обработки произвольной таблицы. Выберите запрос *Revenues*. Найдите третий шаг *Changed Type*. Данный шаг генерируется автоматически при импорте исходной таблицы в редактор Power Query. Если в дальнейшем будете работать с таблицами меньшего размера, в которых нет одного из перечисленных здесь столбцов, запрос не будет выполняться. Удалить этот шаг. Найдите шаг *Changed Type 1*. Удалите его. Найдите шаг *Changed Type 2*. Этот шаг был сгенерирован при разделении столбца Атрибут на три столбца. Этот шаг также можно удалить, поскольку он не будет применяться в некоторых ситуациях. Например, для преобразования отмены свертывания столбцов при двух уровнях иерархии нет смысла применять код, изменяющий типы трех столбцов. Удалите этот шаг.

#### Модификация шага Filled Down

Изменим формулы, которые явно ссылаются на имена столбцов, и обобщим их для формирования динамических ссылок на столбцы, исходя из значений длин запросов *RowFields* и *ColumnFields*.

Выберите запрос *Revenues* и шаг *Filled Down*. В строке формул появится следующий код:

= Table.FillDown(Revenues\_DefinedName,{"Column1", "Column2"})

Заполняются значениями первые N – 1 столбцы. Модифицируем код, изменив жестко закодированный список {"Column1", "Column2"} на динамический список имен первых N – 1 столбцов в текущей таблице.

Прежде всего рассмотрим, почему в этой формуле применяется шаг Revenues\_DefinedName вместо шага *Navigation*. Посмотрев на панель Примененные шаги, заметим, что шаг *Filled Down* находится ниже шага *Navigation*. Обычно в строке формул отображается ссылка на имя таблицы, которое совпадает с именем предыдущего шага. Но в данном случае видно, что в формуле шаг Revenues\_DefinedName применяется вместо шага *Navigation*. Чтобы понять, в чем дело, выберите команду *Расширенный редактор* на вкладке *Главная* и обратите внимание на следующую строку:

Revenues\_DefinedName = Source {[Item="Revenues", Kind="DefinedName"]}[Data],

Как видите, корректным идентификатором предыдущей строки для применения преобразования *Filled Down* является Revenues\_DefinedName (данный идентификатор применяется перед первым знаком равенства). Теперь, когда известно, как обратиться к предыдущему шагу, изменим шаг *Filled Down*, чтобы в общем случае заполнить N – 1 столбцов.

Вот исходная формула шага *Filled Down*:

= Table.FillDown(Revenues\_DefinedName,{"Column1", "Column2"}),

Изменим ее на:

= Table.FillDown(

Revenues\_DefinedName,

List.FirstN(

Table.ColumnNames(Revenues\_DefinedName),

List.Count(RowFields) – 1

)

)

Функция Table.FillDown ранее получала статический список в качестве второго аргумента. Для превращения его в динамический список сначала необходимо получить список всех имен столбцов таблицы. Это можно сделать с помощью функции Table.ColumnNames на предыдущем шаге Revenues\_DefinedName:

Table.ColumnNames(Revenues\_DefinedName)

Функция Table.ColumnNames возвращает все имена столбцов для текущей таблицы: Column1, Column2, Column3 и т. д. Для возвращения только первых N – 1 столбцов необходимо вычислить значение N – 1. В данном случае N представляет счетчик списка RowFields:

List.Count(RowFields)

Поэтому N – 1 будет иметь следующий вид:

List.Count(RowFields) - 1

Теперь, когда известно, как вычислить N – 1, можно получить имена первых N – 1 столбцов, применяя функцию List.FirstN, которая возвращает список с первыми элементами исходного списка. Функция List.FirstN имеет два аргумента: список ввода и количество первых элементов для выборки. Можно сформировать список {"Column1", "Column2"} с помощью следующего кода:

List.FirstN(Table.ColumnNames(Revenues\_DefinedName), List.Count(RowFields) - 1)

Теперь эта формула применяется в качестве второго аргумента в обновленной функции Table.FillDown:

= Table.FillDown(

Revenues\_DefinedName,

List.FirstN(

Table.ColumnNames(Revenues\_DefinedName),

List.Count(RowFields) – 1

)

)

#### Модификация шага Merged Columns

Выберите шаг *Merged Columns*. Этот шаг сгенерирован при объединении столбцов. В строке формул жестко закодированная ссылка на список {"Column1", "Column2", "Column3"}:

= Table.CombineColumns(

#"Filled Down",

{"Column1", "Column2", "Column3"},

Combiner.CombineTextByDelimiter(":", QuoteStyle.None),

"Merged"

)

Для получения динамической ссылки на N имен столбцов исходной таблицы можно применить тот же код, что и выше, но на этот раз с N в качестве второго аргумента (вместо N – 1). Итоговая формула для шага *Merged Columns*:

= Table.CombineColumns(

#"Filled Down",

List.FirstN(

Table.ColumnNames(Revenues\_DefinedName),

List.Count(RowFields)

),

Combiner.CombineTextByDelimiter(":", QuoteStyle.None),"Merged"

)

#### Модификация шага Filled Down 1

Этот шаг сгенерирован ранее для второй операции заполнения. Вот его код:

= Table.FillDown(#"Transposed Table",{"Column1", "Column2"})

Изменим жестко закодированные имена столбцов с учетом динамической версии. На этот раз, поскольку таблица транспонирована и обработаны поля столбцов, необходимо заполнить первые M – 1 столбца списка *ColumnFields*. Вот модифицированный код:

= Table.FillDown(

#"Transposed Table",

List.FirstN(

Table.ColumnNames(#"Transposed Table"),

List.Count(ColumnFields)-1

)

)

#### Модификация шага Unpivoted Other Columns

Выберите шаг *Unpivoted Other Columns*. В строке формул будут отображены следующие жестко закодированные столбцы:

= Table.UnpivotOtherColumns(

#"Promoted Headers",

{

"::Color",

"::Parent Category",

"Country:State/Region:City \ Category"

},

"Attribute", "Value"

)

При работе с интерфейсом эти столбцы были выделены как первые M столбцов таблицы. Для динамического получения первых M столбцов применим следующий код:

= Table.UnpivotOtherColumns(

#"Promoted Headers",

List.FirstN(

Table.ColumnNames(#"Promoted Headers"),

List.Count(ColumnFields)

),

"Attribute",
ValueField
)

Использован запрос *ValueField* вместо названия столбца *Value* в качестве нового имени для значений. Напомним, что имя *ValueField = Revenue.*

#### Модификация шага Split Column by Delimiter

Выберите шаг Split *Column by Delimiter*. Ему соответствует код, который разбивает столбец Attribute на N столбцов:

= Table.SplitColumn(

#"Unpivoted Other Columns",

"Attribute",

Splitter.SplitTextByDelimiter(":", QuoteStyle.Csv),

{"Attribute.1", "Attribute.2", "Attribute.3"}

)

Третий аргумент функции Table.SplitColumn определяет два важных элемента: число столбцов для разделения и названия разделенных столбцов. Вместо названий Attribute.X, заданных по умолчанию, можно использовать поля строк Country, State/Region, City, которые определены в запросе RowFields. Вот модифицированный код:

= Table.SplitColumn(

#"Unpivoted Other Columns",

"Attribute",

Splitter.SplitTextByDelimiter(":", QuoteStyle.Csv),

RowFields

)

#### Модификация шага Renamed Columns

Выберите шаг *Renamed Columns*. Его код:

= Table.RenameColumns(

#"Split Column by Delimiter",

{

{"::Color", "Color"},

{"::Parent Category", "Parent Category"},

{"Country:State/Region:City \ Category", "Category"},

{"Attribute.1", "Country"},

{"Attribute.2", "State/Region"},

{"Attribute.3", "City"},

{"Value", "Revenue"}

}

)

Этот код переименовывает столбцы, передавая список списков в качестве второго аргумента функции Table.RenameColumns. Каждый внутренний список представляет пару текстовых значений: первый текст — это имя существующего столбца, а второй — имя нового столбца.

Обратите внимание, что на шаге *Unpivoted Other Columns* столбец *Revenue* уже был правильно переименован. А на шаге *Split Column by Delimiter* были верно названы поля строк. Вот как выглядит запрос на шаге *Split Column by Delimiter:*



Рис. 2. Названия некоторых столбцов уже правильные

Теперь нужно переименовать первые M столбцов. Для получения списка названий первых M столбцов выполните следующий код:

List.FirstN(

Table.ColumnNames(#"*Split Column by Delimiter*"),

List.Count(ColumnFields)

)

Для получения новых названий можно просто выполнить запрос ColumnFields.

Каким образом создать список списков, в котором каждый элемент из первого имени столбца M связан с полями столбцов? Для этого можно применить функцию языка M List.Zip, которая получает два списка в качестве входных данных и возвращает один список списков с парами из обоих списков. Например, следующие входные данные:

List.Zip({"A","B","C"}, {"1", "2", "3"})

возвращают такой список:

{{"A","1"}, {"B", "2"}, {"C", "3"}}

Таким образом, применяя функцию List.Zip для получения названий первых M столбцов таблицы для полей столбцов, имеем следующий код:

List.Zip(

{

List.FirstN(

Table.ColumnNames(#"*Split Column by Delimiter*"),

List.Count(ColumnFields)

),

ColumnFields

}

)

и получим желаемый список списков, который используется в оригинальной формуле:

{

{"::Color", "Color"},

{"::Parent Category ", "Parent Category "},

{"Country:State/Region:Country \ Category", "Category"}

}

Итого обновленный код для шага *Renamed Columns*:

= Table.RenameColumns(

#"Split Column by Delimiter",

List.Zip(

{

List.FirstN(

Table.ColumnNames(#"*Split Column by Delimiter*"),

List.Count(ColumnFields)

),

ColumnFields

}

)

)

#### Восстановление шага Changed Type

Ранее мы удалили несколько шагов *Changed Type* (*Измененный тип*). Теперь пришло время явно изменить тип столбца Revenue на Десятичное число. Если выполнить эту операцию с помощью пользовательского интерфейса, то базовый код в строке формул получит следующий вид:

= Table.TransformColumnTypes(#"Renamed Columns",{{"Revenue", type number}})

Измените жестко закодированное значение "Revenue" на ValueField:

= Table.TransformColumnTypes(#"Renamed Columns",{{ValueField, type number}})

#### Придание универсальности запросу

Пришло время отсоединить обобщенную последовательность отмены свертывания столбцов от ее источника. Можно заметить, что первые два шага зависят от конкретной рабочей книги и рабочего листа. Также можно обобщить эти шаги и запустить последовательность отмены свертывания столбцов из любого источника данных.

Щелкните правой кнопкой мыши на шаге *Filled Down* и выберите команду *Извлечь предыдущий*. В окне *Извлечение шагов* выберите *Имя нового запроса* – *Source*, нажмите Ok. Создается новый запрос, который загрузит исходную таблицу из файла Excel. Обобщенный запрос запустится как ссылка на новый запрос.

#### Преобразование запроса в функцию

Если кликнуть правой кнопкой мыши на запрос *Source*, можно найти опцию *Создать функцию*... Однако мы создадим функцию с помощью расширенного редактора, поскольку окно диалога *Создать функцию* не слишком успешно может применяться для работы со списками.

Щелкните правой кнопкой мыши на запросе *Revenues*, и выберите *Расширенный редактор*. Отобразившийся на экране код уже включает в себя модифицированную версию запроса. Для преобразования запроса в функцию введите над строкой let:

(Source, RowFields, ColumnFields, ValueField)=>

Удалите строку:

Source = Source,

Щелкните *Готово* и переименуйте запрос *Revenues* в FnUnpivotSummarizedTable.

Обратите внимание, что FnUnpivotSummarizedTable на панели *Запросы* отмечена значком **fx**. После выбора этой функции предварительный просмотр данных больше не отображается. Вместо этого на панели предварительного просмотра отображается панель *Ввод параметров*, которую можно использовать для вызова функции. Однако в данном случае нас это не устраивает, поскольку нельзя заполнить функцию списками. Обратите также внимание, что в разделе *Примененные шаги* весь набор шагов сведен к одному шагу и нельзя изменять функцию средствами пользовательского интерфейса.



Рис. 3. Интерфейс функции

Если функция создается с помощью расширенного редактора, а не с помощью команды пользовательского интерфейса *Создать функцию*, то вне расширенного редактора изменить функцию невозможно.

Однако можно продублировать функцию, а в дублированном запросе можно удалить первую строку или закомментировать ее, добавив символы // в начале строки. Таким образом, удается преобразовать дублированную функцию в редактируемый запрос. После этого можно просмотреть шаги на панели *Примененные шаги* и внести изменения средствами пользовательского интерфейса. Завершив изменение нового запроса, можно скопировать код из расширенного редактора в исходную функцию.

Функция может быть написана как отдельный запрос, как это выполнено в этом случае, или как внутренний элемент внутри запроса. При вызове с аргументами создается новое значение. Можно вызвать функцию, перечислив параметры функции в формате "через запятую" в круглых скобках, за именем функции.

Итак, чего мы достигли к настоящему моменту. Вот аргументы, которые объявлены в функции:

* Source – таблица исходных данных;
* RowFields – список полей строк, в данном случае {"City", "State/Region", "City"};
* ColumnFields – список полей столбцов, в данном случае {"Color", "Parent Category ", "Category"};
* ValueField – имя для столбца Value, в данном случае "Revenue".

Символ "следования" => служит разделом между объявлением интерфейса функции и реализацией функции. Уже завершена часть реализации функции, когда обобщен каждый из соответствующих шагов с помощью параметров Source, RowFields, ColumnFields и ValueField.

Нужно отметить, что заголовок функции также может быть задан с явными типами, которые требуются в качестве входных аргументов, и ожидаемым типом вывода, используя элемент as, за которым следует конкретный тип. Вот пример строго типизированного заголовка функции:

=(Source as table, RowFields as list, ColumnFields as list, ValueField as text) as table =>

Наличие объявленных типов может гарантировать, что при вызове функции с неправильными типами возвращается ошибка. Например, неправильная передача текста "Country, State/Region, City" вместо списка в аргументе RowFields приведет к следующей ошибке:

Expression.Error: We cannot convert the value "Country, State/Region, City" to type List. Details: Value=Country, State/Region, City Type=Type

#### Тестирование функции

Теперь можно использовать функцию для отмены свертывания столбцов в любой обобщенной таблице. Для проверки выполните следующие действия. Щелкните правой кнопкой мыши на запросе *Source* и выберите команду *Ссылка*. Переименуйте новый запрос в *Results*.

В строке формул измените код:

= Source

на

= FnUnpivotSummarizedTable(Source, RowFields, ColumnFields, ValueField)

Нажмите Enter. Вуаля!

Еще раз, что мы сделали. Всю заметку мы долго модифицировали запрос, выполняющий отмену свертывания столбцов. А затем превратили запрос в функцию FnUnpivotSummarizedTable. Далее загрузили «сырой» запрос *Source*:



Рис. 4. «Сырой» запрос *Source*

А затем написали новый запрос *Results* с единственной строкой

= FnUnpivotSummarizedTable(Source, RowFields, ColumnFields, ValueField)

… т.е., просто вызвали функцию с параметрами. И получили идеальную таблицу фактов:



Рис. 5. Таблица фактов

Для придания универсальности удалите запросы *RowFields*, *ColumnFields* и *ValueField.* Выделите запрос *Results* и замените строку:

= FnUnpivotSummarizedTable(Source, RowFields, ColumnFields, ValueField)

на:

= FnUnpivotSummarizedTable(Source,

{"Country", "State/Region", "City"},

{"Color", "Parent Category ", "Category"},

"Revenue"

)

Т.е., внутри функции мы явно задали все её параметры.

Интересно, что эта функция может работать даже со сводной таблицей 1x1, но применять ее в этом аспекте не очень практично.

Чтобы попрактиковаться сохраните копию файла Excel. Откройте ее, импортируйте таблицу *Revenues* из файла C07E02.xlsx. Изучите, сколько уровней иерархии представлено в таблице. Как озаглавить строки, столбцы и значения. Создайте копию запроса *Revenues*, назовите ее *Results2.* Запрос *Results2* содержит единственную строку

= Revenues

Замените ее на код аналогичный этому

= FnUnpivotSummarizedTable(Source,

{"Country", "State/Region", "City"},

{"Color", "Parent Category ", "Category"},

"Revenue"

)

… но с иным источником, и иным количеством и именами полей. Если всё сделаете верно, то получите новую идеальную таблицу фактов. Функция работает!