# От ловушек к робастным запросам в Power Query

Это фрагмент книги Гил Равив. Power Query в Excel и Power BI: сбор, объединение и преобразование данных.

Предыдущий раздел К содержанию Следующий раздел

10 ловушек, рассмотренные в этой главе, подробно обсуждаются в блоге автора книги.

## *Причины и следствия ловушек*

В основе редактора Power Query лежат два основополагающих принципа:

- Пользовательский интерфейс переводит шаги в код.
- Предварительный просмотр данных помогает формировать логику преобразования.

Эти принципы весьма удобны, но… они же являются виновниками большинства ошибок, которые приводят к сбоям при обновлении или потере данных. Одна из частых ситуаций, приводящих к сбоям обновления, возникает при изменении имени столбца в исходной таблице. А потеря данных может возникать из-за некорректной фильтрации.

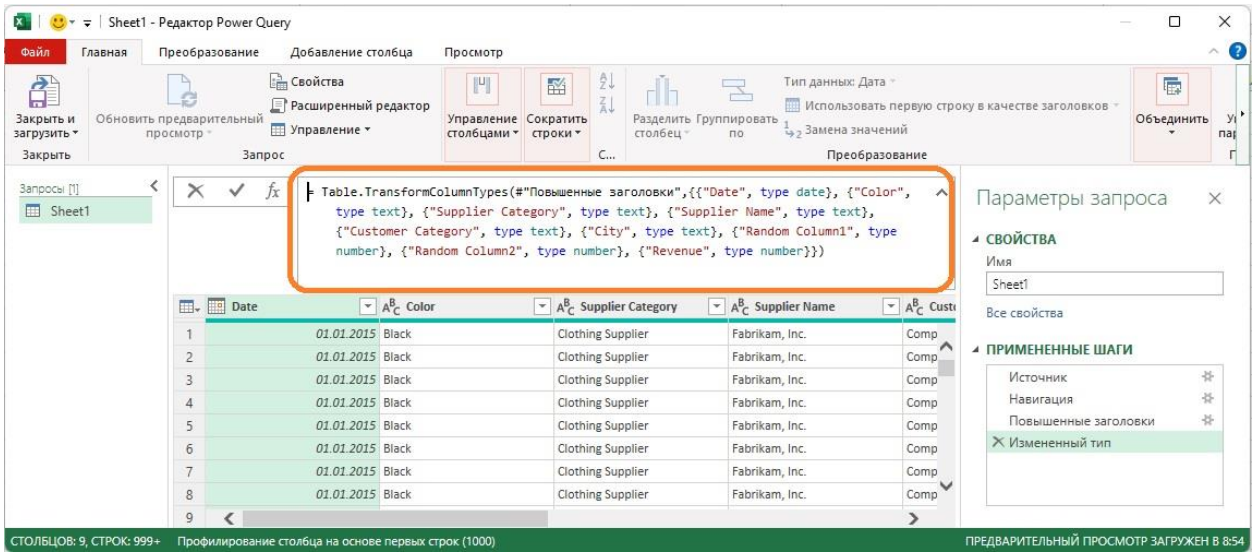


Рис. 1. Жесткая ссылка на имена столбцов

Прежде чем перейти к подводным камням и методам их преодоления, рассмотрим несколько первоочередных принципов. Эти принципы помогают минимизировать влияние ловушек. Они подразделяются на три темы: осведомленность, лучшие практики, модификация кода.

## *Осведомленность*

Следуйте этим рекомендациям после каждого шага преобразования:

- Убедитесь, что данные на панели предварительного просмотра выглядят правильно. Прокрутите вниз для просмотра большего числа строк. Прокрутите вправо, чтобы увидеть все столбцы.
- Всегда отображайте строку формул. Это имеет решающее значение для предотвращения попадания в ловушки.
- Просмотрите код в строке формул. Обратите внимание на жестко закодированные элементы в формуле, такие как имена и значения столбцов. Например, если удалить столбец Column1 в запросе, то можно заметить в строке формул, что столбец на самом деле назван Column1 (с завершающим пробелом в названии). Если владелец источника данных заметит это, то вполне вероятно сделает исправление, и обновление потерпит неудачу. Просмотр строки формул повысит вашу осведомленность и позволит предотвратить сбои в будущем.

## *Лучшие практики*

Придерживайтесь нескольких хорошо зарекомендовавших себя методик:

- Сохраняйте копии отчета до обновления. Если произошел сбой при обновлении можно сравнить отчеты и выявить причину сбоев.

- Общайтесь с коллегами, предоставляющими внешние данные. Убедитесь, что коллеги осведомлены о ваших отчетах, взаимодействуйте с ними, чтобы они знали, что их работа является частью вашей отчетности.
- Некоторых ловушек можно избежать, сделав правильный выбор операции (например, удаление столбца) или, наоборот, избегая выполнения некоторых ненужных преобразований (например, изменение типа или порядка столбцов).

### *Модификации кода*

Имеется ряд модификаций, которые можно выполнить на уровне формул для создания надежных запросов. И хотя существует множество таких модификаций, наиболее распространенный способ основан на функции Table.ColumnNames. Она позволяет преобразовать ссылку на жестко закодированное имя столбца в динамическую ссылку, которая не дает сбоев.

### *Ловушка 1. Игнорирование строки формул*

Если в редакторе PQ строка формул не активна, пройдите *Просмотр –> Строка формул*. С помощью строки формул легко обнаружить статические ссылок на имена столбцов. Загрузите файл C10E01.xlsx и сохраните его в папке C:\Data\C10\. Откройте новую книгу Excel. Пройдите *Данные –> Получить данные –> Из файла –> Из книги*.

Выберите файл C10E01.xlsx и щелкните *Импорт*. В окне *Навигатор* выберите Shit1, кликните *Преобразовать данные* (см. рис. 1). В редакторе PQ в строке формул видно, что шаг *Измененный тип* ссылается на имена столбцов. Эти значения имен – именно то, что следует искать в строке формул, если желаете избежать ловушек.

Видно, что имеются два столбца с именами *Random Column1* и *Random Column2*. Предположим, что эти столбцы содержат случайные числа, и они не нужны в вашем отчете. В реальных сценариях вы встретите имена столбцов, которые могут иметь произвольный или временный контекст. Можно сохранить их в отчете или удалить, но на шаге *Измененный тип* становится ясно, что код уже жестко запрограммирован со ссылкой на эти столбцы. Если эти столбцы не важны, то велика вероятность, что в будущем вы не найдете их в исходной таблице, поскольку владелец внешнего источника данных может их удалить.

Пройдите *Главная –> Закрыть и загрузить*. Сохраните файл под именем C10E01 - Solution.xlsx. Откройте файл C10E01.xlsx и удалите столбцы *Random Column1* и *Random Column2*. Сохраните файл C10E01.xlsx. Вернитесь к файлу C10E01 - Solution.xlsx. Пройдите *Данные –> Обновить всё*. Обновление завершится ошибкой:


Рис. 2. Ошибка обновления

Как можно справиться с описанным сценарием с помощью трех основных принципов, упомянутых ранее: осведомленность, лучшие практики и модификации кода? Изучив строку формул, вы увидите, что названия столбцов жестко закодированы. Вы можете обратиться к владельцам данных и сообщить им о существовании таких столбцов. Попросить их не изменять структуру данных или информировать вас, если они удалят или переименуют.

Наверное, самое эффективное – удалить части формулы:

{"Random Column1", type number}, {"Random Column2", type number},

Вот более надежная версия формулы, которая игнорирует случайные столбцы:

= Table.TransformColumnTypes(#"Повышенные заголовки",{{"Date", type date}, {"Color", type text}, {"Supplier Category", type text}, {"Supplier Name", type text}, {"Customer Category", type text}, {"City", type text}, {"Revenue", type number}})

В этой формуле всё еще имеется множество столбцов, которые могут быть удалены или переименованы в будущем. Действительно ли нужно ссылаться на все эти столбцы? Это – суть второй ловушки.

### *Ловушка 2. Измененные типы*

Чтобы избежать второй ловушки обратитесь к шагу *Измененный тип*, который был создан автоматически. Самое простое и наиболее распространенное решение, позволяющее избежать этой ловушки, состоит в удалении шага *Измененный тип.*

Уточним, в чем состоит суть проблемы. Вам иногда необходимы правильные типы столбцов. Без определения типов в некоторых столбцах нельзя выполнять арифметические операции или операции с датами. Учитывая это обстоятельство, редактор Power Query неявно определяет типы всех столбцов в таблице. При отсутствии этого автоматического шага, если вы забудете явно изменить тип, то не сможете, например, рассчитать сумму по столбцу. Вместо того чтобы использовать автоматический шаг *Измененный тип*, удалите его и задайте типы вручную.

> «Чем позже, тем лучше» – полезный девиз для изменения типов в Power Query. Рекомендуется изменять типы столбцов в редакторе Power Query, а не полагаться на автоматический шаг *Измененный тип*, и чем позже вы явно измените типы, тем лучше.

Изменяя тип на последнем шаге сценария, вы получите следующие преимущества:

- Изменение типа требует определенных вычислительных усилий. Если вы уменьшите число строк (применяя фильтры), то сократите время обновления запроса, выполняя изменение типа после шага фильтрации.
- Изменение типов может привести к ошибкам из-за того, что в ячейках содержатся значения, которые невозможно преобразовать в новый тип. Чем позже подобные ошибки появятся в цепочке этапов преобразования, тем проще их устранить. (Пример ошибки, которую трудно обнаружить из-за применения шага *Измененный тип* на раннем этапе, описывается в заметке.)
- Изменение типа на ранних стадиях может не сохраняться в некоторых сценариях. Например, если добавить все файлы из папки, то изменение типа, выполненное на уровне образца запроса, не распространяется на добавленные результаты.

Power Query позволяет отключать автоопределение и изменение типов, но сделать это можно лишь для текущего файла Excel. Опции глобальной настройки нет. В редакторе Power Query пройдите *Файл –> Параметры и настройки –> Параметры запроса*. Снимите галку *Определять типы и заголовки столбцов для неструктурированных источников.*

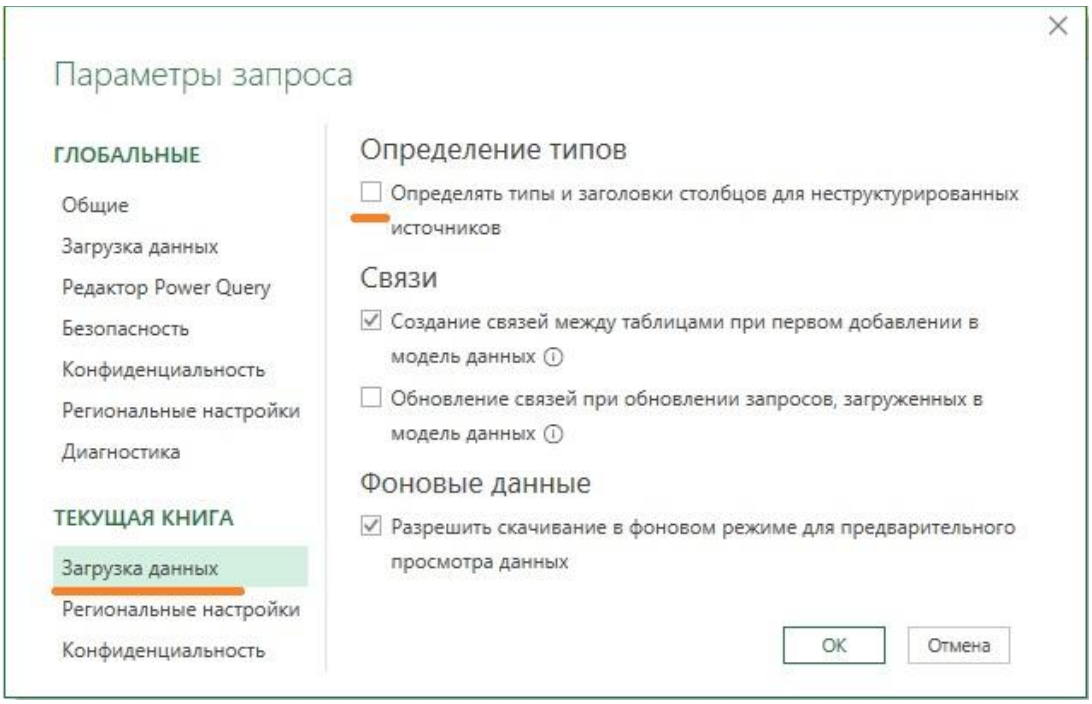


Рис. 3. Отключите автоматическую проверку и изменение типов

Теперь вы должны постоянно помнить и изменять типы данных вручную. Это относится ко всем нетекстовым столбцам. В нашем примере необходимо изменить типы столбцов *Date* и *Revenue*. Это можно также сделать, отредактировав автоматически созданный шаг *Измененный тип:*

```
= Table.TransformColumnTypes(
        #"Повышенные заголовки",
        {
                {"Date", type date},
                {"Revenue", type number}
        }
)
```

Если выполняется обработка таблиц, которые включают большое число столбцов для выполнения явного преобразования типов и возникает необходимость в автоматическом способе определения типов столбцов без ссылки на имена столбцов, удобное выражение M можно найти в следующей заметке.

### *Ловушка 3. Небезопасная фильтрация*

Рассмотрим базовый сценарий, демонстрирующий ошибку фильтрации. Допустим, вас попросили проанализировать влияние на бизнес прекращения импорта товаров, окрашенных в черный цвет. Поэтому необходимо отфильтровать запрос и выбрать товары, окрашенные не в черный цвет.

Откройте новую книгу Excel. Пройдите *Данные –> Получить данные –> Из файла –> Из книги*. Выберите файл C10E01.xlsx и щелкните *Импорт*. В окне *Навигатор* выберите Sheet1, кликните *Преобразовать данные*. Удалите шаг *Измененный тип*. Измените тип столбца *Date* на Дата, а тип столбца *Revenue* – на *Десятичное число*. Отфильтруйте все товары, окрашенные в черный цвет, щелкнув на элементе управления фильтра в столбце *Color* и отменив выбор флажка Black.

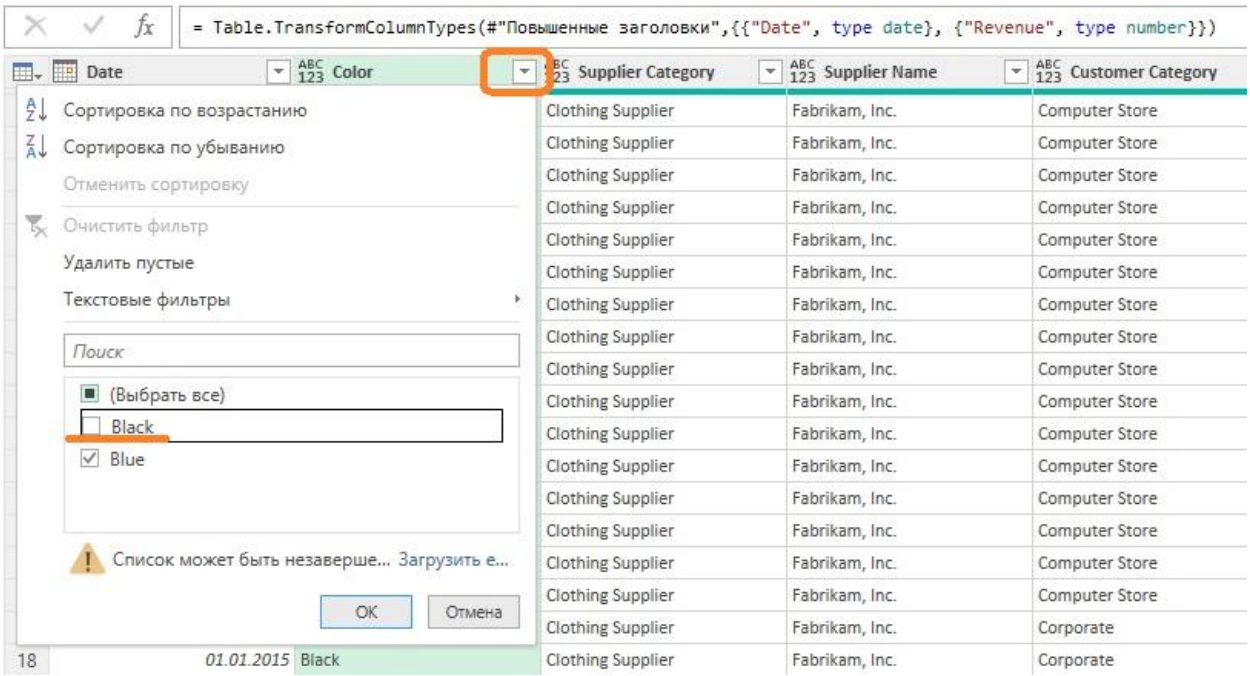


Рис. 4. Отмена выделения флажка Black приводит к нежелательным результатам

После выполнения этого шага фильтрации можно ожидать, что в качестве выходных данных будут применяться все цвета, кроме черного. Однако код шага…

```
= Table.SelectRows(#"Измененный тип", each ([Color] = "Blue"))
```

…говорит о другом. Поскольку в текущем запросе присутствуют только два цвета, отмена выбора флажка Black приводит к неверному коду. В результате отфильтрованы только товары синего цвета. Если в будущем появятся иные цвета, они не попадут в выборку. Следует исправить формулу:

```
= Table.SelectRows(#"Измененный тип", each ([Color] <> "Black"))
```

Вместо редактирования формулы можно тоньше настроить фильтр. Удалите шаг *Строки с примененным фильтром*. Снова выберите элемент управления фильтром для столбца *Color*. На панели *Фильтр* выберите команду *Текстовые фильтры –> Не равно –> Black*. Щелкните Ok. Убедитесь, что шагу соответствует корректная формула M. Всегда лучше использовать текстовые фильтры, чем выбирать значения на панели ввода текста.

Можете изучить файл решения C10E02 - Solution.xlsx.

### *Логика, определяющая условия фильтрации*

При выборе значений на панели *Фильтр* PQ следует двум правилам:

- Если число выбранных значений меньше или равно числу невыбранных значений, редактор Power Query создает *позитивное* условие, применяя знак равенства для выбранных значений.
- Если число выбранных значений больше числа невыбранных значений, редактор Power Query создает *отрицательное* условие, применяя знак неравенства для невыбранных значений.

Рассмотрим пример столбца, содержащего значения от 1 до 6. Создайте новый файл Excel. Пройдите *Данные –> Получить данные –> Из других источников –> Пустой запрос*. В строке формул введите:

```
= {1..6}
```

Пройдите *Средства для списков –> Преобразование –> В таблицу*. Щелкните Ok. Выберите элемент управления фильтром для столбца *Column1*. В окне *Фильтр* снимите галочку для значения 1 и закройте окно. В строке формул отразится:

```
= Table.SelectRows(#"Преобразовано в таблицу", each ([Column1] <> 1))
```

Повторите фильтрацию, отменив выбор для значений 1 и 2:

```
= Table.SelectRows(#"Преобразовано в таблицу", each ([Column1] <> 1 and [Column1] <> 2))
```

Далее отмените выбор значений 1, 2 и 3. Теперь строка формул содержит *позитивное* условие:

```
= Table.SelectRows(
        #"Преобразовано в таблицу",
        each (
                [Column1] = 4 or
                [Column1] = 5 or
                [Column1] = 6
        )
)
```

Это потенциально неверное условие. Корректное условие должно оставаться отрицательным:

```
= Table.SelectRows(
        #"Преобразовано в таблицу",
        each (
                [Column1] <> 1 or
                [Column1] <> 2 or
                [Column1] <> 3
        )
)
```

Позитивное условие приведет к ошибке при появлении новых значений. Представьте, что в ваших данных в будущем появятся значения 7, 8, 9 и 10. При позитивном условии обновление приведет к тому, что по-прежнему в фильтр попадут только значения 4, 5 и 6.

### *Поиск значений на панели Фильтр*

Откройте новую книгу Excel. Пройдите *Данные –> Получить данные –> Из файла –> Из книги*. Выберите файл C10E01.xlsx и щелкните *Импорт*. В окне *Навигатор* выберите Sheet1, кликните *Преобразовать данные*. Удалите шаг *Измененный тип*. Измените тип столбца *Date* на Дата, а тип столбца *Revenue* – на *Десятичное число*. Допустим вы хотите исключить строки, относящиеся к

городу *Baldwin City*. Выберите элемент управления фильтром для столбца *City*. Поскольку список городов довольно длинный в поле поиска начните вводить *Ba*...

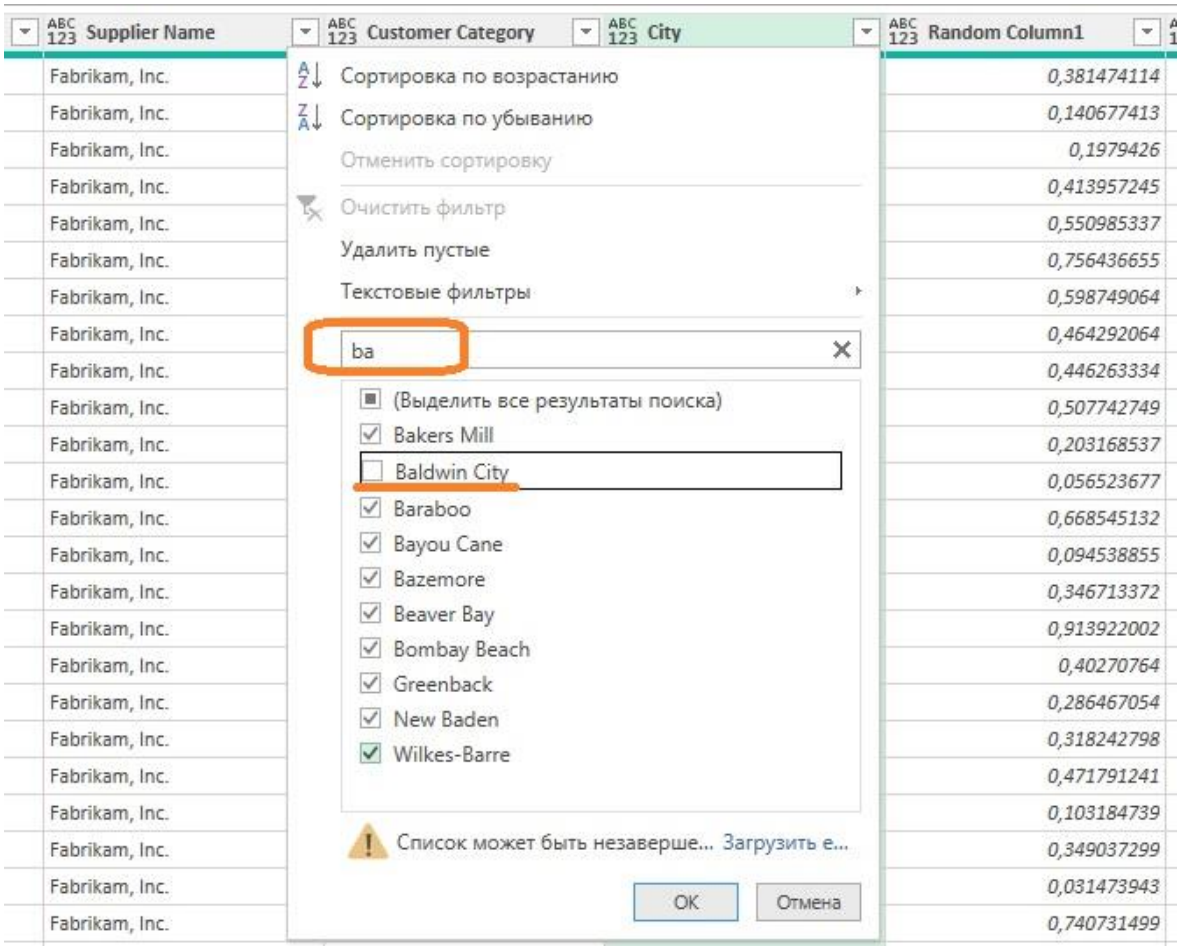


Рис. 5. Поиск значений на панели *Фильтр*

Список сократится. Снимите выделение для *Baldwin City*. Нажмите Ok.  К сожалению, при просмотре строки формул оказывается, что полученное выражение содержит неверное положительное условие:

```
= Table.SelectRows(
        #"Измененный тип",
        each (
                [City] = "Bakers Mill" or
                [City] = "Baraboo" or
                [City] = "Bayou Cane" or
                [City] = "Bazemore" or
                [City] = "Beaver Bay" or
                [City] = "Bombay Beach" or
                [City] = "Greenback" or
                [City] = "New Baden" or
                [City] = "Wilkes-Barre"
        )
)
```

Выбранными оказались все города, содержащие *Ba*, кроме *Baldwin City.* А все другие города будут отфильтрованы.

Для устранения ошибки измените формулу:

```
= Table.SelectRows(#"Измененный тип", each ([City] <> "Baldwin City"))
```

Сценарии, описанные в этом разделе, весьма распространены, поэтому попасть в третью ловушку довольно легко. Лучшая защита – применение окна *Текстовые фильтры*.

### *Ловушка 4. Переупорядочение столбцов*

При переупорядочении используется функция Table.ReorderColumns со ссылкой на все имена столбцов. Это ослабляет запрос и увеличивает вероятность ошибок обновления в будущем, если столбцы будут переименованы или удалены из исходных данных. Старайтесь не использовать переупорядочение ради наглядности.

Но существуют сценарии, которые требуют определенного порядка столбцов в запросе. Например, в главе 4 переупорядочение требовалось, чтобы переместить вычисляемый столбец в начало таблицы, что позволило после транспонирования использовать его в качестве заголовков. Если переупорядочение необходимо, применяйте его после того, как в отчете останутся только те столбцы, которые действительно необходимы.

Откройте новую книгу Excel. Пройдите *Данные –> Получить данные –> Из файла –> Из книги*. Выберите файл C10E01.xlsx и щелкните *Импорт*. В окне *Навигатор* выберите Sheet1, кликните *Преобразовать данные*. Удалите шаг *Измененный тип*. Переместите столбец *City* на место второго столбца, а *Revenue* – на место третьего столбца:

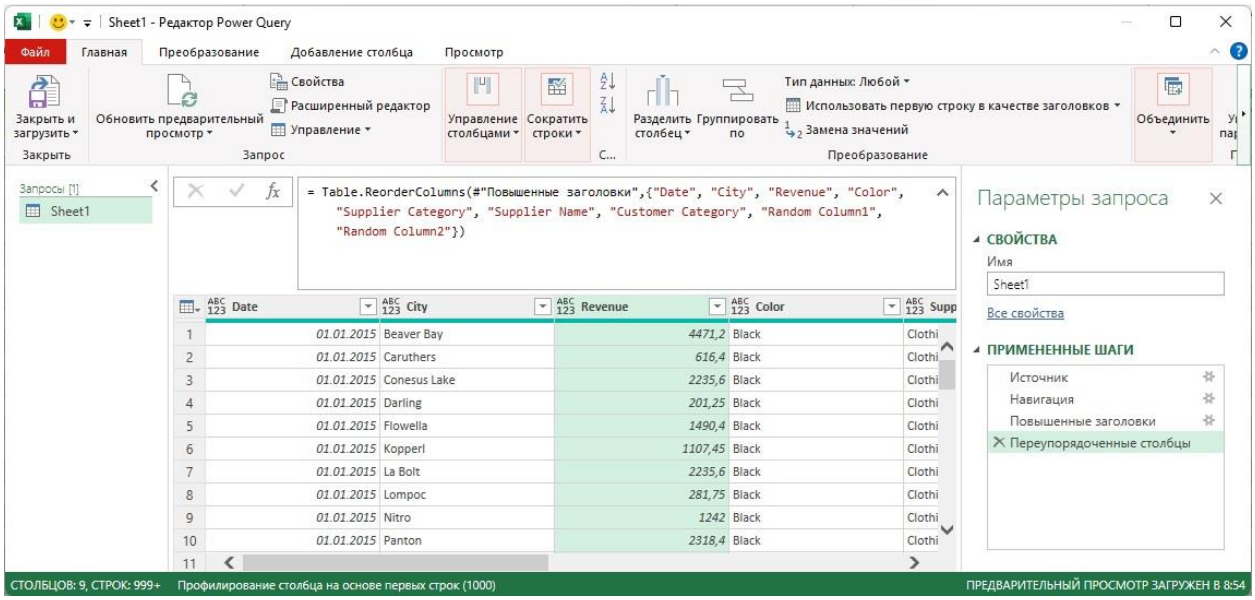


Рис. 6. Переупорядочение столбцов *City* и *Revenue*

Изучите код в строке формул:

```
= Table.ReorderColumns(
        #"Повышенные заголовки",
        {
                "Date",
                "City",
                "Revenue",
                "Color",
                "Supplier Category",
                "Supplier Name",
                "Customer Category",
                "Random Column1",
                "Random Column2"
        }
)
```

Понятно, что формула не будет обновляться, если имена столбцов изменятся. Измените формулу: Удалите из формулы фрагмент…

```
= Table.ReorderColumns(
        #"Повышенные заголовки",
        List.InsertRange(
                List.Difference(
```

```
                Table.ColumnNames(#"Повышенные заголовки"),
                {"City", "Revenue"}
            ),
            1,
            {"City", "Revenue"}
        )
)
```

Функция List.InsertRange получает список в качестве входных данных и вставляет другой список с определенным смещением от нуля. Например, если имеется список A и нужно добавить поля *City* и *Revenue* в качестве второго и третьего пунктов списка A, можно применить следующую формулу:

```
List.InsertRange(A, 1, {"City", "Revenue"}
```

Осталось исключить из списка A столбцы *City* и *Revenue.* Для этого используется функция List.Difference. Она принимает один список в качестве первого аргумента, а другой список – в качестве второго аргумента и возвращает новый список со всеми элементами из первого списка, которых нет во втором списке (левостороннее объединение двух списков).

Таким образом, если B – список имен столбцов, то следующая функция вернет все названия столбцов, кроме *City* и *Revenue*:

```
A = List.Difference(B, {"City", "Revenue"})
```

Теперь при указании функции Table.ColumnNames в следующей формуле вместо B можно собрать все части для формирования полного выражения:

```
B = Table.ColumnNames(#"Повышенные заголовки")
```

И наконец, при объединении всех элементов вместе получим окончательную формулу.

### *Пользовательская функция FnReorderSubsetOfColumns*

Можно упростить, описанный выше подход, с помощью пользовательской функции. Откройте книгу Excel, созданную в предыдущем разделе. Пройдите *Данные –> Получить данные –> Из других источников –> Пустой запрос*. В редакторе PQ пройдите *Главная –> Расширенный редактор*. Вставьте код:

```
(tbl as table, reorderedColumns as list, offset as number)
    as table =>
        Table.ReorderColumns(
            tbl,
            List.InsertRange(
                List.Difference(
                    Table.ColumnNames(tbl),
                    reorderedColumns
                ),
                offset,
                reorderedColumns
            )
        )
```

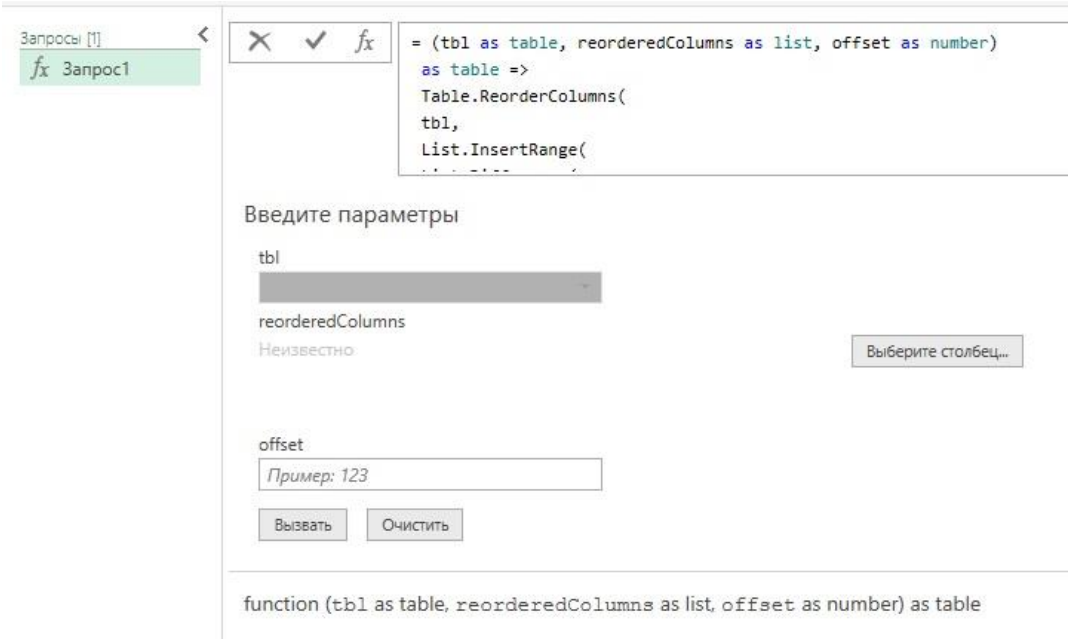


Рис. 7. Пользовательская функция

Переименуйте *Запрос1* в *FnReorderSubsetOfColumns*.

Вернитесь к запросу *Sheet1*, удалите шаг *Переупорядоченные столбцы*. Щелкните на значке **$f_x$** в строке формул для создания нового шага. Введите:

= FnReorderSubsetOfColumns(#"Повышенные заголовки", {"City", "Revenue"}, 1)

Функция FnReorderSubsetOfColumns получает таблицу, подмножество переупорядоченных имен столбцов в виде списка и индекс, начинающийся с нуля. Затем выполняется переупорядочение очень надежным способом, без ссылок на имена других столбцов. Файлы решения C10E03 - Solution.xlsx.

> Есть и другие способы реализации функции Table.ReorderColumns, которые позволяют избежать ошибок обновления. Можно задать третий аргумент, который позволит игнорировать пропущенные поля или добавлять нулевые значения для пропущенных полей. И хотя эти методы предотвращают сбои обновления, они могут приводить к неожиданным результатам. Подробнее см. заметку.

### *Ловушка 5. Удаление и выделение столбцов*

Хотя удаление ненужных столбцов является важным моментом при формировании эффективных отчетов (меньшее число столбцов приводит к сокращению объема памяти и размера файла), при удалении столбца вы ухудшаете запрос, подвергая его риску будущих сбоев при обновлениях. Всякий раз, удаляя столбец, вы рискуете, что в будущем обновление может завершиться ошибкой, если удаленный столбец будет отсутствовать во внешнем источнике данных.

Чтобы снизить риск сбоев при обновлениях, можно следовать простой рекомендации: сосредоточьтесь на столбцах, которые необходимо сохранить, а не на тех, которые следует удалить. Редактор Power Query позволяет удалять или сохранять столбцы. И хотя более естественно нажать на кнопку *Удалить*, во многих случаях предпочтительнее выбирать столбцы, которые желательно сохранить.

Откройте новую книгу Excel. Пройдите *Данные –> Получить данные –> Из файла –> Из книги*. Выберите файл C10E01.xlsx и щелкните *Импорт*. В окне *Навигатор* выберите Sheet1, кликните *Преобразовать данные*. Удалите шаг *Измененный тип*. Выберите столбцы *Random Column1* и *Random Column2* и нажмите *Удалить*. Обратите внимание на код в строке формул:

= Table.RemoveColumns(#"Повышенные заголовки",
{"Random Column1", "Random Column2"})

В будущем, если исходная таблица не будет содержать один из этих столбцов, то обновление не пройдет. Удалим случайные столбцы другим способом и повысим робастность запроса.

Удалите шаг *Удаленные столбцы*. Пройдите *Главная –> Выбрать столбцы* и отмените выделение столбцов *Random Column1* и *Random Column2*.

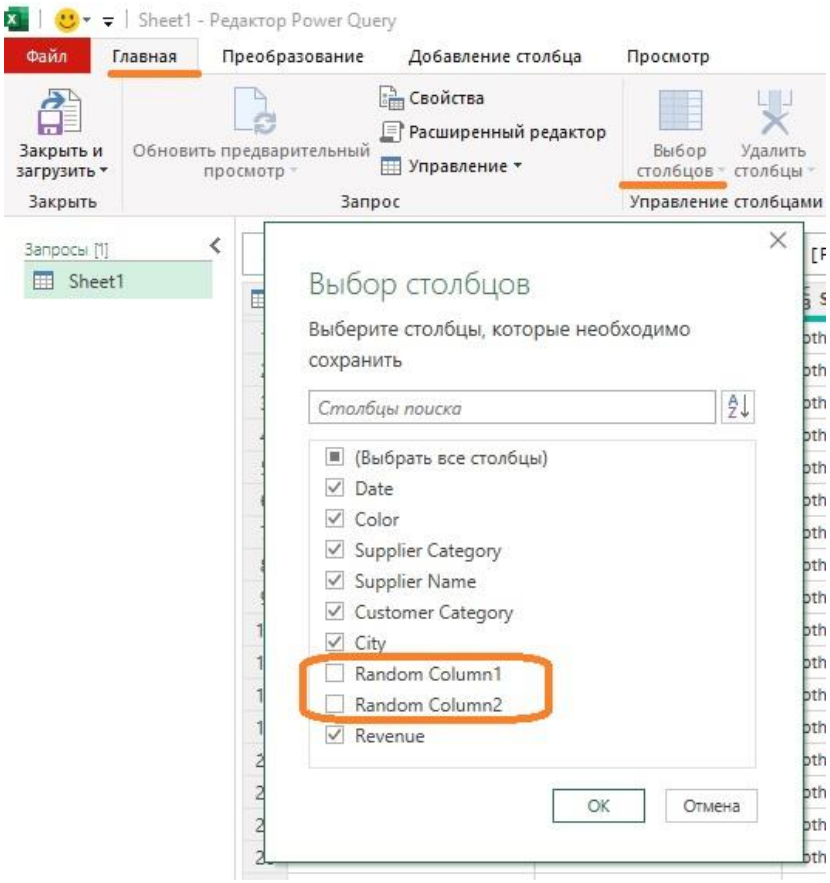


Рис. 8. Выбор столбцов

Нажмите Ok. Изучите строку формул:

```
= Table.SelectColumns(#"Повышенные заголовки",
{"Date", "Color", "Supplier Category", "Supplier Name",
"Customer Category", "City", "Revenue"})
```

### *Игнорирование пропущенного столбца*

В некоторых ситуациях необходимо сохранить очень большое число столбцов, а риск удаления нескольких столбцов может быть ниже, чем проблемы, связанные с указанием столь большого числа столбцов. Часто не имеет значения, удаляются или выбираются столбцы. Вы имеете дело с внешними источниками данных, которые могут меняться. Для предотвращения ошибок обновления существует необязательный третий аргумент, который можно задать в функциях Table.RemoveColumns и Table.SelectColumns, что позволит игнорировать ошибки вместо того, чтобы прекращать обновление.

Аргумент MissingField.Ignore игнорирует пропущенный столбец, MissingField.UseNull сохраняет имя столбца при ошибках, но заполняет его нулями. Аргумент MissingField.UseNull более практичен, чем MissingField.Ignore, поскольку гарантирует, что выбранные имена столбцов будут включены в конечные результаты. Однако оба варианта могут привести к ошибкам, которые трудно обнаружить. Таким образом, ошибка обновления может оказаться предпочтительнее неожиданных результатов, которые вы получите, задав эти аргументы.

### *Выделение или удаление столбцов на основе их расположения*

Во многих сценариях удаление или выбор столбцов на основе их расположения является более точным, чем обращение к ним по имени. Используя функцию Table.ColumnNames для получения списка всех имен столбцов и функцию List.Range для формирования подмножества столбцов, можно выбрать любое подмножество столбцов в зависимости от их расположения.

Следующие формулы удаляют первый столбец в таблице:

= Table.RemoveColumns(Source, List.First(Table.ColumnNames(Source)))

= Table.RemoveColumns(Source, Table.ColumnNames(Source){0})

Заменив функцию FirstN на List.FirstN, получите удаление N первых столбцов таблицы:

= Table.RemoveColumns(Source, List.FirstN(Table.ColumnNames(Source), N))

Следующая формула удаляет последний столбец таблицы:

= Table.RemoveColumns(Source, List.Last(Table.ColumnNames(Source), 1))

А эта формула сохраняет второй и третий столбцы в таблице:

= Table.SelectColumns(Source, List.Range(Table.ColumnNames(Source), 1, 2))

Функция List.Range получает список в качестве первого аргумента, смещение от нуля и количество возвращаемых элементов.

Можно выбрать отдельные столбцы. Следующая формула выбирает первый и второй столбец:

= Table.SelectColumns(Source, {Table.ColumnNames(Source){0}, Table.ColumnNames(Source){1}})

### *Выделение или удаление столбцов на основе их имен*

Существует бесчисленное множество возможностей выбора или удаления столбцов в языке M. Следующая формула применяет функцию List.Select к именам столбцов, что позволит удалить столбцы, содержащие подстроку Random:

```
= Table.RemoveColumns(#"Повышенные заголовки",
List.Select(Table.ColumnNames(#"Повышенные заголовки"),
each Text.Contains(_, "Random")))
```

Аналогичный результат можно получить с помощью функции Table.SelectColumns и негативной логики (т.е. можно выбрать все столбцы, которые не содержат "Random"):

```
= Table.SelectColumns(#"Повышенные заголовки",
List.Select (Table.ColumnNames(#"Повышенные заголовки"),
each not Text.Contains(_, "Random")))
```

Файл решения C10E04 - Solution.xlsx.

### *Ловушка 6. Переименование столбцов*

Столбцы принято переименовывать довольно часто. Это дает пользователям удобные имена в отчетах. Всякий раз при переименовании столбца возрастает вероятность сбоев обновления в будущем. Можно повысить надежность запроса, изменяя формулу и избегая ссылок на имена текущих столбцов.

Загрузите файл C10E05.xlsx и сохраните ее в папке C:\Data\C10\. Откройте новую книгу Excel. Пройдите *Данные –> Получить данные –> Из файла –> Из книги*. Выберите файл C10E05.xlsx и щелкните *Импорт*. В окне *Навигатор* выберите Sheet1, кликните *Преобразовать данные*. Удалите шаг *Измененный тип*. Импортированная таблица включает семь столбцов, в названии которых присутствует слово *Random*.

В ручном режиме переименуйте столбцы *Random Column1* и *Random Column2* в *Factor 1* и *Factor 2*. Строка формул содержит код:

```
= Table.RenameColumns(#"Повышенные заголовки",
{{"Random Column1", "Factor 1"},
{"Random Column2", "Factor 2"}})
```

Если исходная таблица больше не включает один из этих столбцов, то обновление завершится неудачей. Представьте, что владелец данных исходной таблицы уведомит вас о том, что он в будущем переименует эти столбцы, но порядок всех столбцов сохранит. Тогда можно сослаться на эти столбцы по их расположению. Например, так:

```
= Table.RenameColumns(#"Повышенные заголовки",
{{Table.ColumnNames (#"Повышенные заголовки"){6}, "Factor 1"},
{Table.ColumnNames(#"Повышенные заголовки"){7}, "Factor 2"}})
```

### *Настраиваемая функция FnRenameColumnsByIndices*

Для того чтобы переименовать столбцы на основе их расположения в таблице, можно написать настраиваемую функцию, позволяющую переименовывать большое подмножество имен столбцов без необходимости отдельно записывать пары для старых и новых имен столбцов. Синтаксис такой функции:

```
= FnRenameColumnsByIndices(#"Повышенные заголовки", {"Factor 1", "Factor 2"}, {6, 7})
```

Подход с применением настраиваемых функций является масштабируемым, позволяя переименовывать большое количество столбцов одновременно и даже импортировать старые и новые имена столбцов из внешнего списка.

Начнем с неэффективного метода. Можно написать код, который задает пары старых и новых имен:

```
= Table.RenameColumns(
        #"Повышенные заголовки",
        {
                {Table.ColumnNames(#"Повышенные заголовки"){6}, "Factor 1"},
                {Table.ColumnNames(#"Повышенные заголовки"){7}, "Factor 2"},
                {Table.ColumnNames(#"Повышенные заголовки"){8}, "Factor 3"},
                {Table.ColumnNames(#"Повышенные заголовки"){9}, "Factor 4"},
                {Table.ColumnNames(#"Повышенные заголовки"){10}, "Factor 5"},
                {Table.ColumnNames(#"Повышенные заголовки"){11}, "Factor 6"},
                {Table.ColumnNames(#"Повышенные заголовки"){12}, "Factor 7"}
        }
)
```

Можно написать более интеллектуальный код:

```
= FnRenameColumnsByIndices(
        #"Повышенные заголовки",
        List.Transform(
                {1..7},
                each "Factor " & Text.From(_)
                ),
        {6..12}
)
```

В этой формуле второй аргумент функции FnRenameColumnsByIndices – динамический список, который создается функцией List.Transform. Первый аргумент функции List.Transform – список индексов от 1 до 7. Функция List.Transform возвращает преобразованный список, объединяющий префикс Factor и соответствующий индекс. Третьим аргументом функции FnRenameColumnsByIndices служит список от 6 до 12 для индексов столбцов для переименования.

Код функции FnRenameColumnsByIndices:

```
(Source as table, ColumnNamesNew as list, Indices as list) =>
let
        ColumnNamesOld = List.Transform(Indices, each Table.ColumnNames(Source){_}),
        ZippedList = List.Zip({ ColumnNamesOld, ColumnNamesNew}),
        #"Переименованные столбцы" = Table.RenameColumns(Source, ZippedList )
in
        #"Переименованные столбцы"
```

Аргументы *Source* соответствуют исходной таблице, ColumnNamesNew – списку новых имен столбцов и индексов для списка индексов в исходной таблице. Первая строка внутри выражения let получает индексы и возвращает имена соответствующих столбцов в таблице *Source*. В следующей строке применяется функция List.Zip для формирования вложенных списков. Каждый вложенный список содержит два элемента – старые и новые имена столбцов, относящиеся к одному и тому же индексу в разных списках. Например, эта функция List.Zip:

List.Zip({"a","b","c"}, {"A", "B", "C"}}

Возвратит следующий список из вложенных списков:

{{"a", "A"}, {"b", "B"}, {"c", "C"}}

Этот формат является обязательным для второго аргумента функции Table.RenameColumns – список вложенных списков, куда входят пары старых и новых имен столбцов, он применяется в третьей строке внутри выражения let.

### *Функция Table.TransformColumnNames*

Существует и другая методика для переименования столбцов. Напомним, что в главе 4 применялась функция Table.TransformColumnNames для переименования всех столбцов. Она заменяла подчеркивания пробелами или выполняла переход к заглавным буквам. Эта функция подойдет в рассматриваемом здесь сценарии для замены имен столбцов:

```
= Table.TransformColumnNames(#"Повышенные заголовки",
each Text.Replace(_, "Random Column", "Factor "))
```

Преимущество этого метода в том, что он успешно переименовывает столбцы, даже если случайные столбцы будут переупорядочены в исходной таблице. Тем не менее следует быть осторожным с логикой переименования, чтобы избежать переименования столбцов там, где не нужно.

> В большинстве случаев достаточно простого переименования, как это обычно и делается. Не особенно углубляйтесь в этот вопрос.

Файл решения C10E05 - Solution.xlsx.

### *Ловушка 7. Разбиение столбца на другие столбцы*

Очередная ловушка поджидает вас при использовании команды *Разделить столбец по разделителю*. Эта операция обычно применяется в двух случаях:

- Базовая операция позволяет разбивать столбец на несколько столбцов. Часто применяется для разделения на дату и время или имя и фамилию.
- Расширенная операция позволяет разделять каждую запись на несколько строк. После этого можно создавать новую таблицу, которая связывает каждое разделяемое значение с его сущностью. Как показано на рис. 9, можно создавать справочную таблицу для исходной таблицы, с помощью которой устанавливается соответствие между кодами продуктов и их цветами.

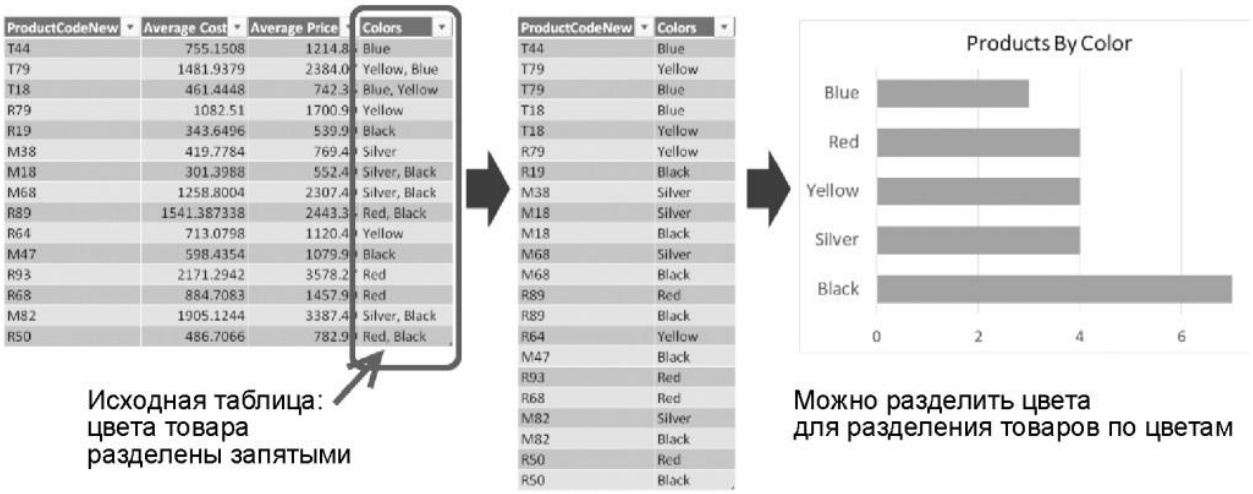


Рис. 9. Разделите столбец Color с данными, записанными через запятую, чтобы узнать, сколько выпущено товаров с разными цветами

Ранее мы уже разделяли столбец *Colors* для установления связи между товарами и цветами, что позволяло определить, сколько выпущено товаров с учетом цвета. Вспомните, что для решения этой проблемы столбец *Colors* разбивался на строки. Следующий сценарий показывает, что может произойти, если вы разбиваете столбец на столбцы, а не на строки.

Загрузите файл C10E06.xlsx и сохраните его в папке C:\Data\ C10\. Откройте новую книгу Excel. Пройдите *Данные –> Получить данные –> Из файла –> Из книги*. Выберите файл C10E05.xlsx и щелкните *Импорт*. В окне *Навигатор* выберите *Products*, кликните *Преобразовать данные*. Удалите шаг *Измененный тип*. На панели *Запросы* щелкните правой кнопки мыши на *Products* и выберите *Ссылка*. Ваша цель состоит в создании новой таблицы, включающей коды и цвета продуктов.

Переименуйте запрос *Products (2)* в *Products and Colors*. Пройдите *Главная –> Выбрать столбцы*, в окне *Выбор столбцов* оставьте галочки напротив *ProductCodeNew* и *Colors*. Щелкните Ok. Выделите столбец *Colors,* пройдите *Преобразование –> Разделить столбец –> По разделителю*. Оставьте настройки окна *Разделить столбец по разделителю* по умолчанию. Щелкните Ok.

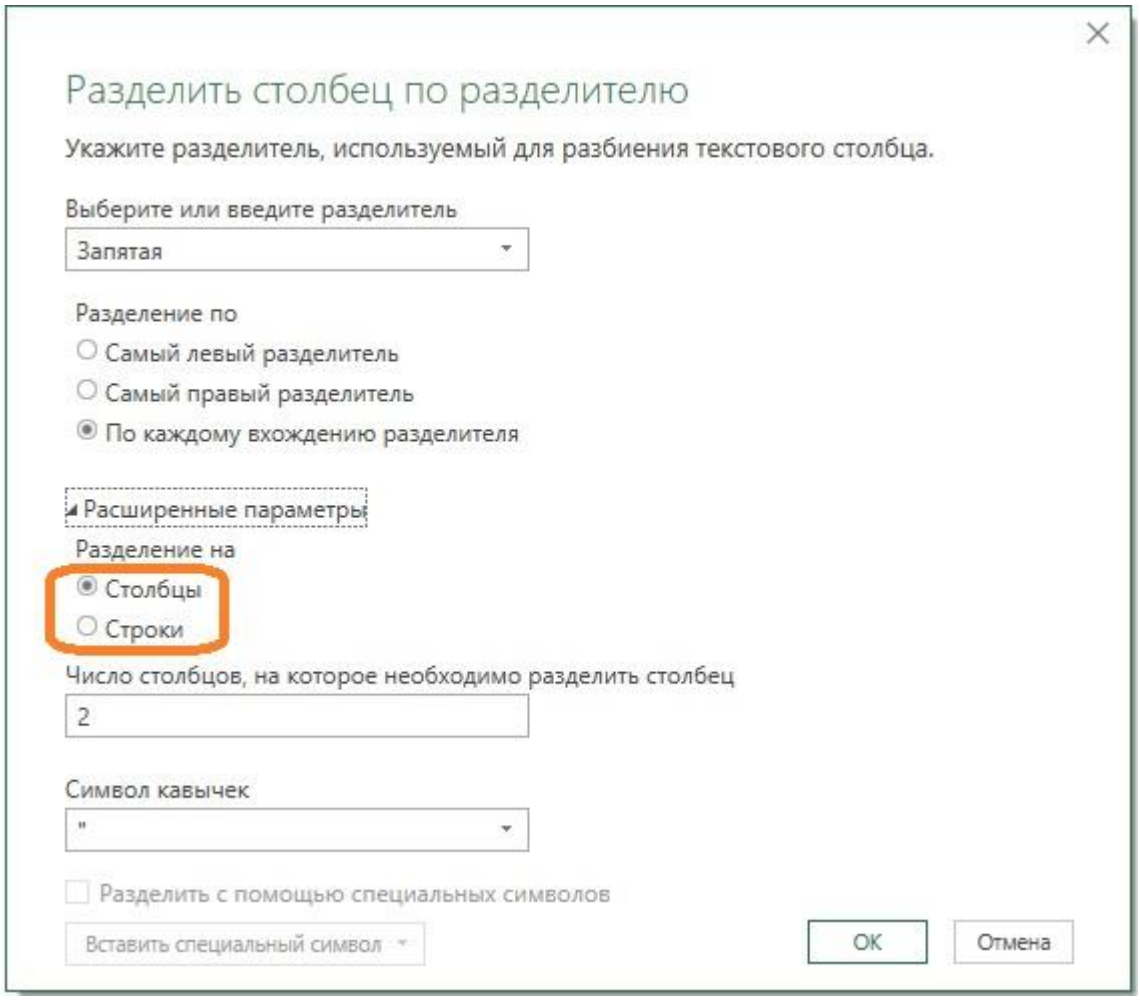


Рис. 10. Окне *Разделить столбец по разделителю*

К сожалению, по умолчанию выполняется разделение на столбцы, а не строки. Столбец *Colors* разделился на *Colors.1* и *Colors.2*. На панели *Примененные шаги* выберите шаг *Разделить столбец по разделителю*. Код в строке формул:

```
= Table.TransformColumnTypes(
#"Разделить столбец по разделителю",
{{"Colors.1", type text}, {"Colors.2", type text}})
```

Поскольку цель – сопоставить коды товаров и цвета, необходимо использовать преобразование отмены свертывания столбцов для получения таблицы, состоящей из пар код/цвет. Выберете столбец *ProductCodeNew*. Пройдите *Преобразование –> Отменить свертывание столбцов –> Отменить свертывание других столбцов*. Столбцы *Colors.1* и *Colors.2* трансформируются в столбцы *Атрибут* и Значение. Последний столбец включает цвета, которые теперь корректно сопоставлены соответствующим кодам товаров. Некоторые названия цветов начинаются с пробела. Чтобы это исправить, кликните правой кнопкой мыши на столбце *Значение* и выберите *Преобразование –> Усечь*. Альтернативный вариант – на панели *Примененные шаги* выберите

шестеренку возле шага *Разделить столбец по разделителю* и измените разделитель на *Пользовательский*, а затем введите запятую и пробел.

Удалите столбец *Атрибут* и загрузите запрос в таблицу на лист Excel. Можно создать сводную диаграмму и построить гистограмму:

| ProductCodeNew | Атрибут | Значение |
|---|---|---|
| T44 | Colors.1 | Blue |
| T79 | Colors.1 | Yellow |
| T79 | Colors.2 | Blue |
| T18 | Colors.1 | Blue |
| T18 | Colors.2 | Yellow |
| R79 | Colors.1 | Yellow |
| R19 | Colors.1 | Black |
| M38 | Colors.1 | Silver |
| M18 | Colors.1 | Silver |
| M18 | Colors.2 | Black |
| M68 | Colors.1 | Silver |
| M68 | Colors.2 | Black |
| R89 | Colors.1 | Red |
| R89 | Colors.2 | Black |
| R64 | Colors.1 | Yellow |
| M47 | Colors.1 | Black |
| R93 | Colors.1 | Red |
| R68 | Colors.1 | Red |
| M82 | Colors.1 | Silver |
| M82 | Colors.2 | Black |
| R50 | Colors.1 | Red |
| R50 | Colors.2 | Black |

| Названия строк | Количество артикулов |
|---|---|
| Black | 7 |
| Silver | 4 |
| Yellow | 4 |
| Red | 4 |
| Blue | 3 |
| **Общий итог** | **22** |

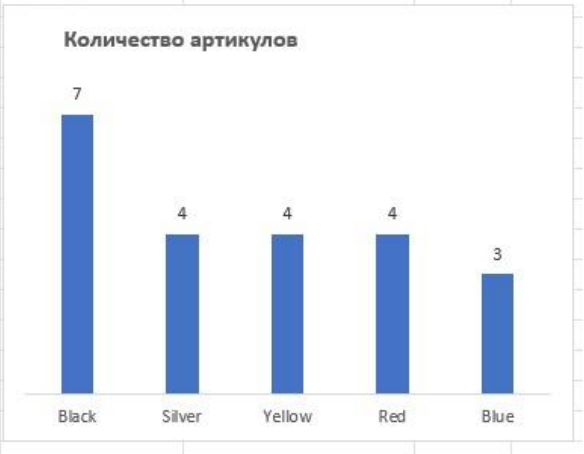


Рис. 11. Анализ цветовой гаммы продукции

Давайте проверим, что произойдет, если какой-то код продукта относится к более чем двум цветам, сохраните отчет и откройте файл C10E06.xslx. В ячейку D3 добавьте еще два цвета к имеющимся двум: Black, Red. Сохраните файл C10E06.xslx и закройте его. Вернитесь к отчету и обновите его. К сожалению, новые цвета не будут включены в отчет по артикулы T79.

Решить проблему можно, разбивая столбец *Colors* на строки. Но если по какой-то причине необходимо разбить столбец на столбцы, а не на строки, то можно отмасштабировать решение, увеличивая число столбцов, на которые нужно выполнить разбиение. Допустим, заранее известно, что количество цветов не может превышать 20. Можно модифицировать формулу на шаге *Разделить столбец по разделителю*:

```
= Table.SplitColumn(Products_Table, "Colors",
Splitter.SplitTextByDelimiter(", ", QuoteStyle.Csv), 20)
```

Заменяя жестко закодированную часть, {"Colors.1", "Colors.2"}, на 20, т. е. на максимальное число столбцов, которое вы ожидаете получить. Это усилит запрос и придаст уверенности, что данные не будут утеряны.

Файлы решения C10E06 - Solution.xlsx. В файле C10E06 - v2.xlsx содержится вторая версия книги C10E06.xlsx, с обновленными значениями цветов. Запросы в файлах решения требуют наличия рабочей книги C10E06 - v2.xlsx в папке C:\Data\C10\.

### *Ловушка 8. Слияние столбцов*

При объединении нескольких столбцов в один формула сначала преобразует все числовые столбцы в текст, а затем объединяет вместе все столбцы. Следующий код генерится автоматически при объединении трех столбцов в исходной таблице, первые из которых являются числовыми (Numeric Column1 и Numeric Column 2), а третий — текстовым (Textual Column3):

```
#"Merged Columns" = Table.CombineColumns(
        Table.TransformColumnTypes(
                Source, {
                        {"Numeric Column1", type text},
```

```
                    {"Numeric Column2", type text}
                },
                "en-US"
        ),
        {"Numeric Column1", "Numeric Column2", "Textual Column3"},
        Combiner.CombineTextByDelimiter(":", QuoteStyle.None),
        "Merged"
)
```

Функция Table.TransformColumnTypes обеспечивает преобразование типов всех числовых столбцов, а затем объединяет соответствующие столбцы. Эта формула может привести к ошибкам обновления. Целесообразнее изменить код и масштабировать его для объединения заданного списка столбцов, не ссылаясь на какие-либо жестко закодированные имена столбцов.

Допустим, имеются имена столбцов для слияния в списке ColumnsToMerge. Модифицированная формула:

```
#"Merged Columns" = Table.CombineColumns(
        Table.TransformColumnTypes(
                Source,
                List.Transform(
                        ColumnsToMerge,
                        each {_, type text}
                ),
                "en-US"
        ),
        ColumnsToMerge,
        Combiner.CombineTextByDelimiter(":", QuoteStyle.None),
        "Merged"
)
```

Основное различие этих двух формул сконцентрировано в следующей части кода:

```
{
        {"Numeric Column1", type text},
        {"Numeric Column2", type text}
}
```

Замена выполняется с помощью функции List.Transform, функция генерит то же преобразование, но без ссылки на имена столбцов:

```
List.Transform(
        ColumnsToMerge,
        each {_, type text}
)
```

Этот код выполняет итерации по каждому имени столбца в ColumnsToMerge и преобразует его в список имен столбцов и текстового типа: {_, type text}. И хотя обычно подобная функция может и не понадобиться, рассмотренный пример служит важной демонстрацией применения функций списка при формировании масштабируемых и надежных версий для ваших автоматически сгенерированных формул.

### *Дополнительные ловушки и методы для создания надежных запросов*

Еще одна ловушка связана с разворачиванием столбцов таблицы. Операция *Развернуть столбцы таблицы* выполняется при объединении нескольких файлов из папки, объединении двух таблиц или при работе с неструктурированными наборами данных, такими как JSON. При развертывании столбцов таблицы необходимо на первом этапе выбрать столбцы для развертывания. В результате редактор Power Query автоматически генерирует формулу с жестко закодированными именами столбцов, а новые имена столбцов могут быть пропущены. Чтобы узнать, как обойти эту ловушку, а также, как избежать пропусков новых столбцов, см. заметку. Еще одна ловушка относится к удалению дубликатов, она рассматривалась ранее.