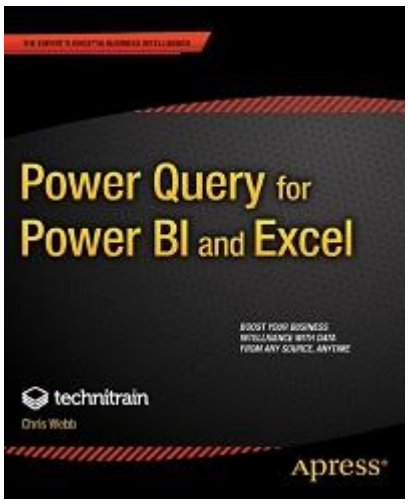
Крис Уэбб. Введение в язык M Power Query

Крис Уэбб (Chris Webb) является признанным гуру в области Power BI, Power Pivot, Power Query и языка M. Он ведет блог с 2004 года, и за это время написал более 1000 постов. У него около 19 тыс. подписчиков. Крис опубликовал 5 книг, и среди них книга, посвященная Power Query. С момента публикации книги прошло 8 лет и за это время интерфейс Power Query претерпел значительные изменения. Тем, кто только начинает знакомиться с Power Query, я рекомендую более современные издания. Начать можно с краткого введения – Марк Мур. Power Query. А далее изучить Кен Пульс и Мигель Эскобар. Язык M для Power Query, Николай Павлов. Скульптор данных в Excel с Power Query и Гил Равив. Power Query в Excel и Power BI: сбор, объединение и преобразование данных. В то же время, язык M за 8 лет изменился незначительно. Мне представляется, что перевод пятой главы, являющейся введением в язык M, будет актуален и интересен.

Chris Webb. Power Query for Power Bl and Excel. – Apress, 2014. – 268 p.



Самая большая проблема языка M с точки зрения пользователя Excel заключается в том, что M мало похож на формулы Excel или код VBA. M – функциональный язык, подобный F#. Вам придется выучить новый синтаксис, новые функции и новые способы выполнения знакомых задач. С другой стороны, маловероятно, что вам понадобится писать много M-кода. В большинстве случаев пользовательский интерфейс решит все ваши задачи. И только в самых сложных ситуациях вам нужно будет отредактировать код, который сгенерирован редактором. И уж совсем редко понадобится написать M-код для шага самостоятельно.

Здесь вы получите начальные сведения о языке M и его синтаксисе, научитесь писать выражения M, познакомитесь с часто используемыми объектами, такими как таблицы, списки и записи, узнаете, как создавать собственные функции в M, чтобы использовать их несколько раз внутри запроса или даже в разных запросах.

Запись кода на языке M в редакторе запросов

Есть два места, где вы можете создавать и редактировать выражения, используемые запросом: в строке формул и в окне *Расширенного редактора*. Кроме того, если вы хотите написать код с нуля (или скопировать код, написанный кем-то), вы можете в Excel пройти по меню *Данные –> Получить данные –> Из других источников –> Пустой запрос*. Если же окно редактора Power Query уже открыто, пройдите *Главная –> Создать источник –> Другие источники –> Пустой запрос*.

Строка формул

Строка формул в редакторе запросов Power Query позволяет просматривать и редактировать M-код для существующего шага, а также создавать новые шаги в запросе, используя M-выражения.

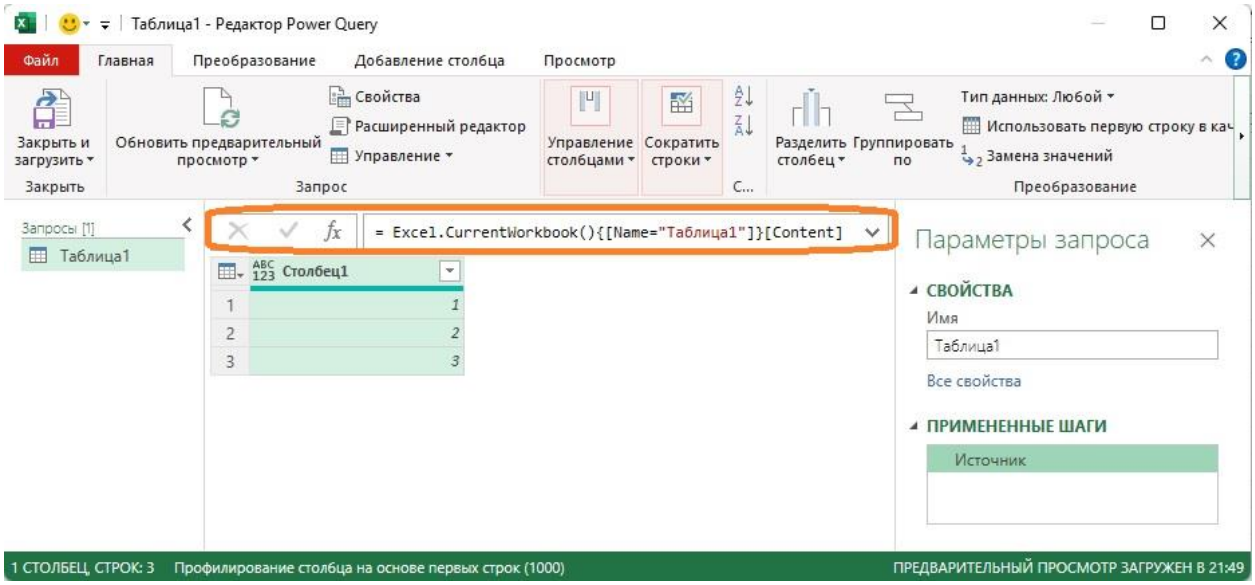


Рис. 1. Строка формул в окне редактора запросов

В строке формул всегда отображается выражение M для шага вашего запроса, который в данный момент выбран на панели ПРИМЕНЕННЫЕ ШАГИ в правой части редактора запросов. После того, как вы выбрали шаг, вы можете щелкнуть внутри строки формул, чтобы отредактировать M-код; когда вы закончите, вы можете либо нажать клавишу *Enter* на клавиатуре, либо нажать кнопку проверки (галочку) слева от строки формул, чтобы сохранить внесенные изменения. После этого на панели предварительного просмотра вы увидите новый результат шага. Если вы вносите изменения, а затем хотите отменить их, вы должны нажать крестик слева от строки формул.

Некоторые шаги на панели ПРИМЕНЕННЫЕ ШАГИ имеют значок шестеренки рядом с названием шага, и щелчок по этому значку позволит отредактировать настройку для этого шага, используя тот же диалог, который вы использовали для создания шага. Если вы отредактируете код M для шага в строке формул и внесете синтаксически правильное изменение, но не поддерживаемое в пользовательском интерфейсе, значок шестеренки исчезнет, и после этого у вас не будет другого выбора, кроме как отредактировать шаг с помощью строки формул.

Вы также можете создать новый шаг в запросе, нажав кнопку f_x слева от строки формул. Будет добавлен новый шаг, выражением которого является лишь название шага, который был выбран на панели ПРИМЕНЕННЫЕ ШАГИ, когда вы нажали на кнопку. Это означает, что шаг вернет точно такой же результат, как и предыдущий шаг в запросе. Теперь вы можете отредактировать код M в строке формул, а также изменить имя шага.

Окно расширенного редактора

Код M для всего запроса можно просмотреть и отредактировать в окне Расширенного редактора. Чтобы открыть Расширенный редактор, в окне PQ пройдите *Главная –> Расширенный редактор*:

Листинг 1[1]

```
let
   Источник = Excel.CurrentWorkbook(){[Name="Таблица1"]}[Content],
   #"Добавлен пользовательский объект" = Table.AddColumn(
      Источник,
      "Умножить на 2",
      each [Столбец1]*2
   ),
   #"Строки с примененным фильтром" = Table.SelectRows(
      #"Добавлен пользовательский объект",
      each ([Столбец1] <> 3)
   )
in
   #"Строки с примененным фильтром"
```

[1] Номер листинга соответствует номеру запроса (таблицы) в приложенном файле Excel

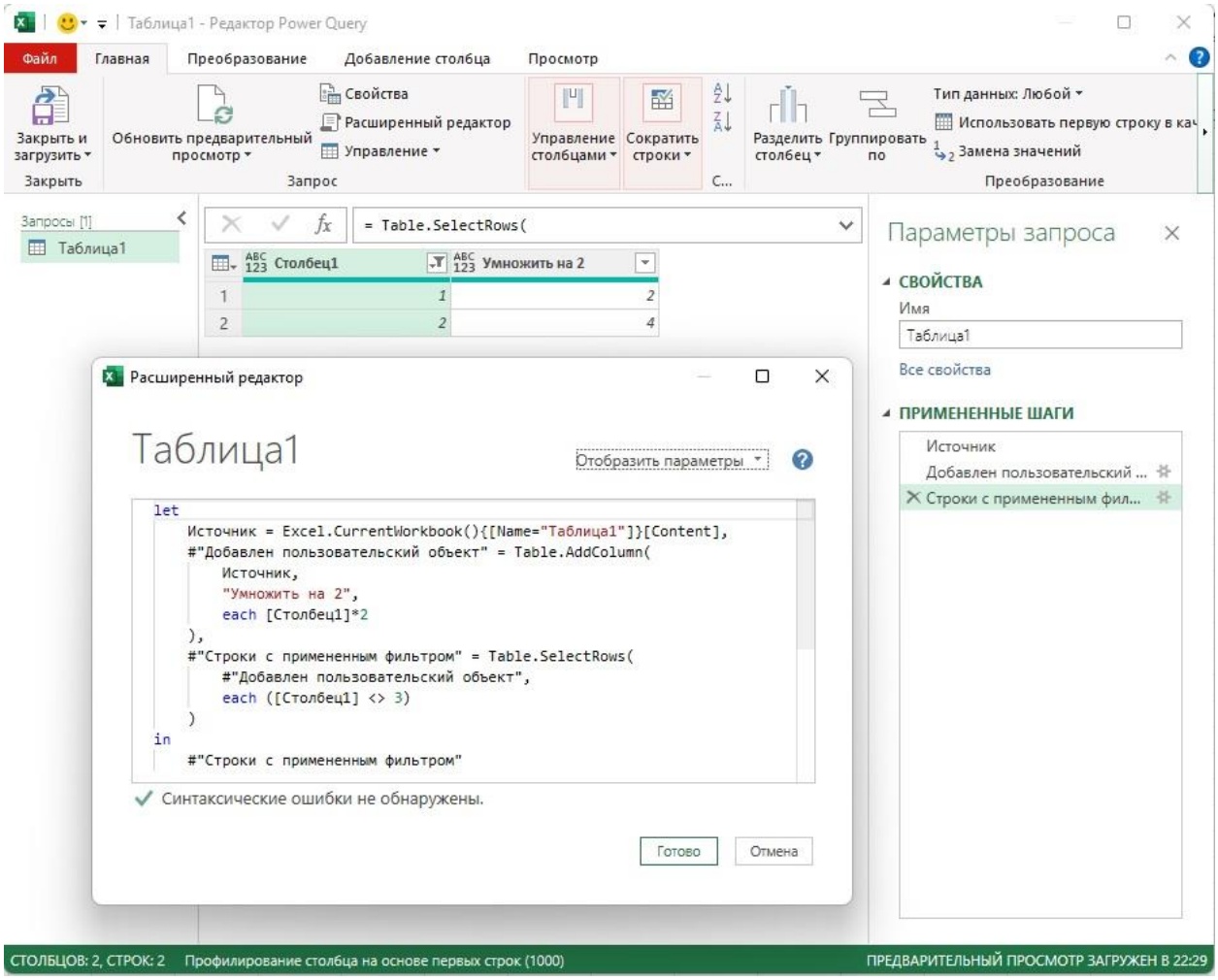


Рис. 2. Окно Расширенного редактора

Со времени написания книги расширенный редактор претерпел значительные изменения. И сейчас в нем работает автодополнение (IntelliSense). При наборе формулы всплывает подсказка по аргументам функции и небольшая справка. Внизу окна отражается сообщение, всё ли Ok с синтаксисом запроса. В случае ошибки текст сообщения отчасти проясняет ситуацию. Если кликнуть *Показать ошибку*, будет подчеркнуто место с ошибкой (более-менее точно).

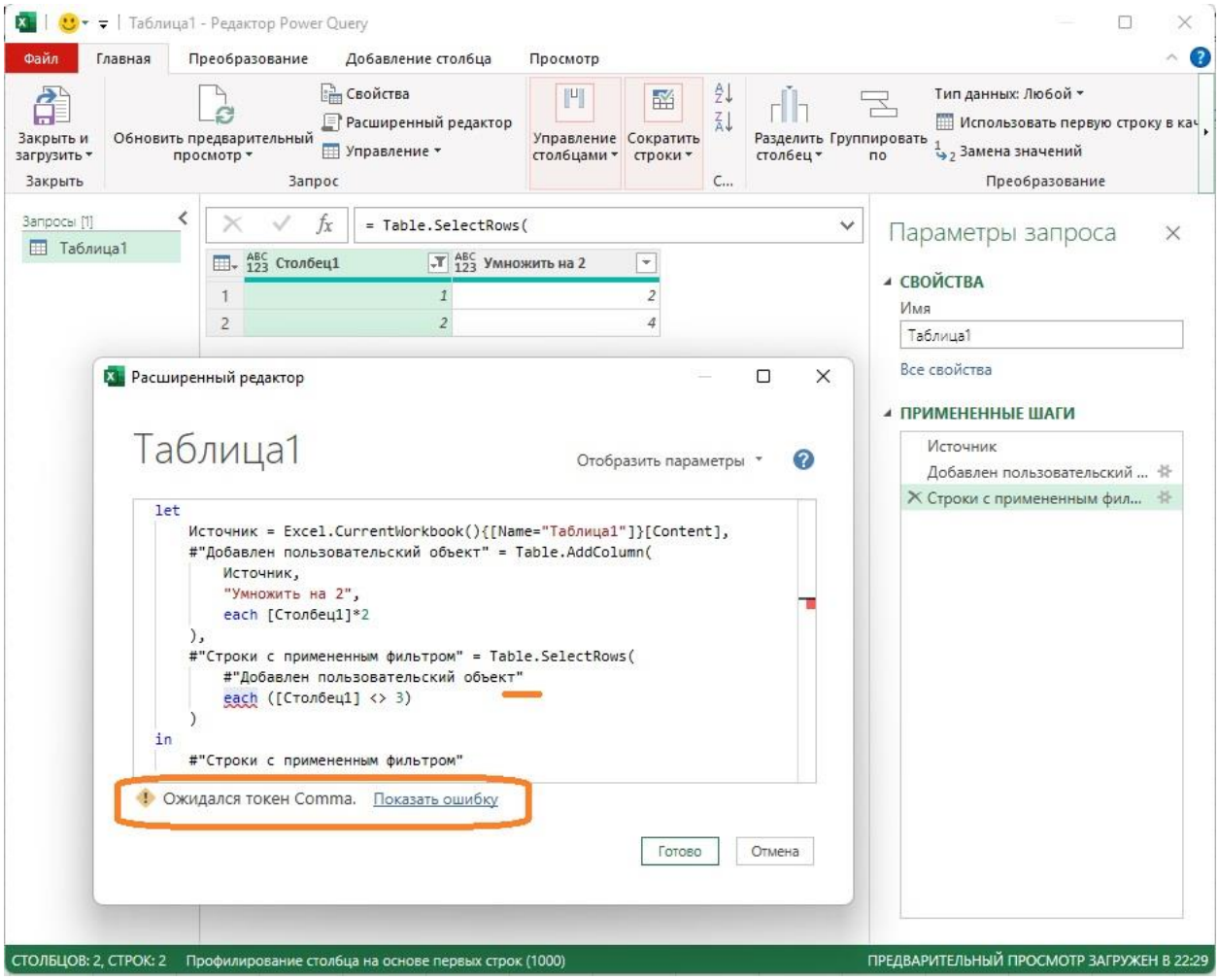


Рис. 3. Сообщения об ошибке

Когда вы закончите работу в Расширенном редакторе, вы можете нажать *Готово*, чтобы сохранить изменения и закрыть окно, или *Отменить*, чтобы закрыть окно без сохранения изменений.

Понятия языка M

Корпорация Майкрософт дает подробную спецификацию Язык формул Power Query M. Это важный документ, и любой, кто изучает M, должен иметь его под рукой. Но… он длинный и сложный. Настоящая заметка не является заменой этого документа, а является введением к изучению языка.

Выражения, значения и оператор let

В языке M два важных понятия: выражения и значения. *Значения* – это числа, текст или более сложные объекты, такие как таблицы. *Выражения* возвращают значения при их вычислении. Например, выражение 10+10 возвращает значение 20. Менее очевидно, что каждый запрос Power Query является одним выражением M, а значение, возвращаемое этим выражением, является выходными данными запроса.

Как запрос Power Query может быть одним выражением M, если он состоит из нескольких шагов? Это возможно с помощью оператора *let*,[2] который позволяет разбить одно выражение на несколько меньших. Рассмотрим таблицу в Excel:

[2] В английском языке используется термин *let expression*. Но многие русскоязычные источники относят *let* к операторам.

	A	B
1	Месяц	Продажи
2	янв	2
3	фев	3
4	мар	5
5	апр	3
6	май	6
7	июн	8
8	июл	7
9	авг	6
10	сен	6
11	окт	9
12	ноя	8
13	дек	5

Рис. 4. Умная таблица в Excel

Импортируем эту таблицу в Power Query, отфильтруем ее, оставив строки с *Продажами* более 5, и, наконец, отсортируем в порядке убывания по столбцу *Продажи*. У нас будет запрос с четырьмя шагами, имена которых видны в области ПРИМЕНЕННЫЕ ШАГИ:

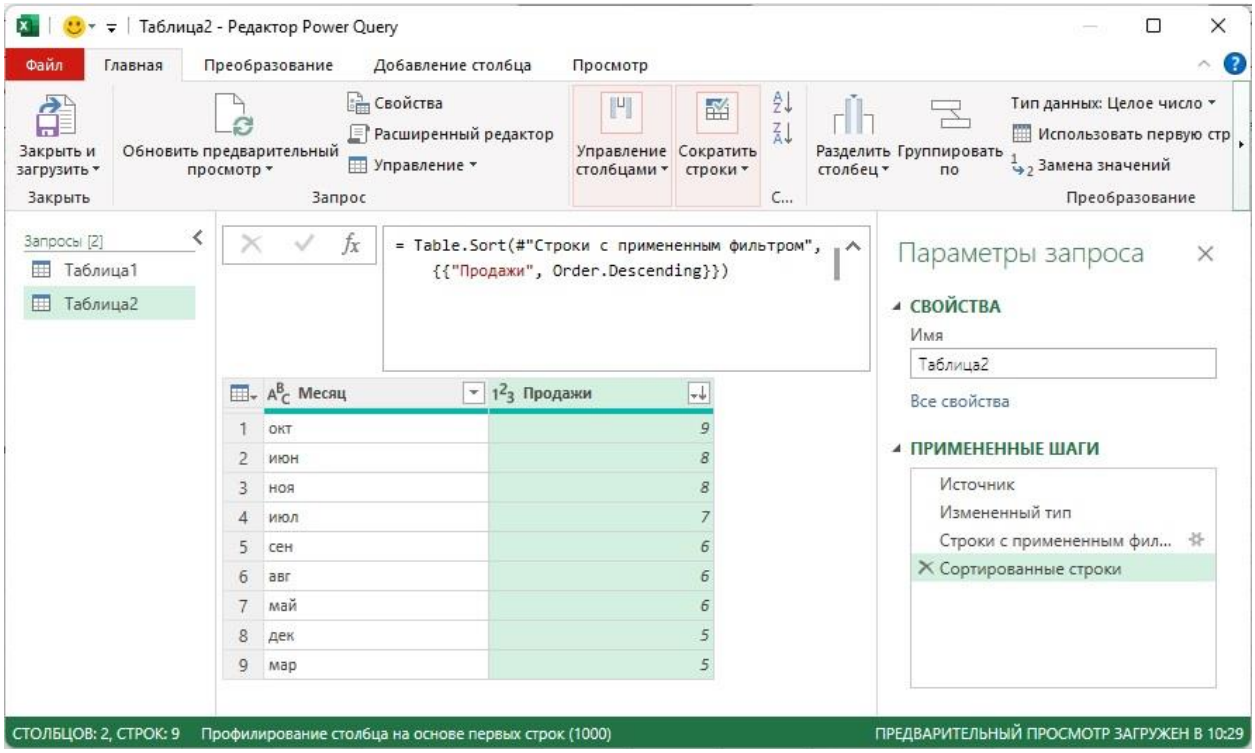


Рис. 5. Запрос из четырех шагов

Если открыть *Расширенный редактор,* вы увидите, что код M состоит из инструкции *let*:

Листинг 2

```
let
  Источник = Excel.CurrentWorkbook(){[Name="Таблица2"]}[Content],
  #"Измененный тип" = Table.TransformColumnTypes(
    Источник,
    {
      {"Месяц", type text},
      {"Продажи", Int64.Type}
    }
  ),
  #"Строки с примененным фильтром" = Table.SelectRows(
    #"Измененный тип",
    each [Продажи] >= 5
  ),
```

```
  #"Сортированные строки" = Table.Sort(
    #"Строки с примененным фильтром",
    {{"Продажи", Order.Descending}}
  )
in
  #"Сортированные строки"
```

Имена всех шагов присутствуют в коде. Каждый шаг представляет собой переменную и записан в виде

Имя шага = выражение

Каждый шаг отделен от предыдущего запятой (кроме последнего). Все шаги определены внутри *let*. Каждая переменная возвращает результат выражения M для этого шага. Переменные могут обращаться к значениям, возвращаемым другими переменными. Например, в листинге 2 шаг *#"Сортированные строки"* ссылается на значение, возвращаемое шагом *#"Строки с примененным фильтром"*, а шаг *#"Строки с примененным фильтром"*, в свою очередь, ссылается на значение, возвращаемое на шаге *#"Измененный тип"*. При этом все три переменные возвращают значения, которые являются таблицами. Оператор *let* возвращает значение, заданное после *in*, которое в нашем примере является именем последней переменной в списке *#"Сортированные строки"*.

В принципе оператор *let* может возвращать результат любого выражения: он может возвращать результат любой переменной в своем списке переменных, или он может возвращать результат выражения, которое не ссылается ни на одну из переменных. Кроме того, переменные могут ссылаться на любую другую переменную внутри конструкции *let…in*, а не только на переменную, объявленную непосредственно перед ней. В том числе можно ссылаться на переменные, объявленные позже в коде. Тем не менее, желательно сохранять порядок переменных в коде ради удобочитаемости.

Если вы перемещаете переменные внутри *let* слишком интенсивно, редактор запросов не сможет отобразить отдельные шаги на панели ПРИМЕНЕННЫЕ ШАГИ, хотя запрос по-прежнему будет работать. Также важно отметить, что шаг вычисляется только в том случае, если возвращаемое им значение используется другим шагом или является конечным результатом запроса. В принципе код M работает от конца к началу. Выполняется попытка вычислить возвращаемое выражение, далее выражения, на которые ссылается конечное выражение и т.д. Так что, если какая-то переменная не включена в эту цепочку, то она и не будет вычислена. Поэтому язык M иногда называют *ленивым*.

Хотя синтаксис M довольно прост, есть несколько особенностей языка, которые стоит упомянуть, прежде чем вы начнете его изучать.

Стандартная библиотека

M поставляется с большим количеством встроенных функций, называемых стандартной библиотекой. Все эти функции перечислены в спецификации Microsoft. Возвращаясь к листингу 2, Excel.CurrentWorkbook(), Table.SelectRows() и Table.Sort() являются примерами функций из стандартной библиотеки. С каждым выпуском число функций в стандартной библиотеке растет.

Текст справки и примеры функций можно просмотреть в стандартной библиотеке, создав в запросе шаг, возвращающий эту функцию. Например, если создать шаг со следующим выражением (обратите внимание, что в конце имени функции нет скобок)…

```
= Table.SelectRows
```

… то отразится справка для функции. Это не то же самое, что вызов функции. Шаг возвращает саму функцию. Если вы решите вызвать функцию, вы можете нажать кнопку *Вызвать*.

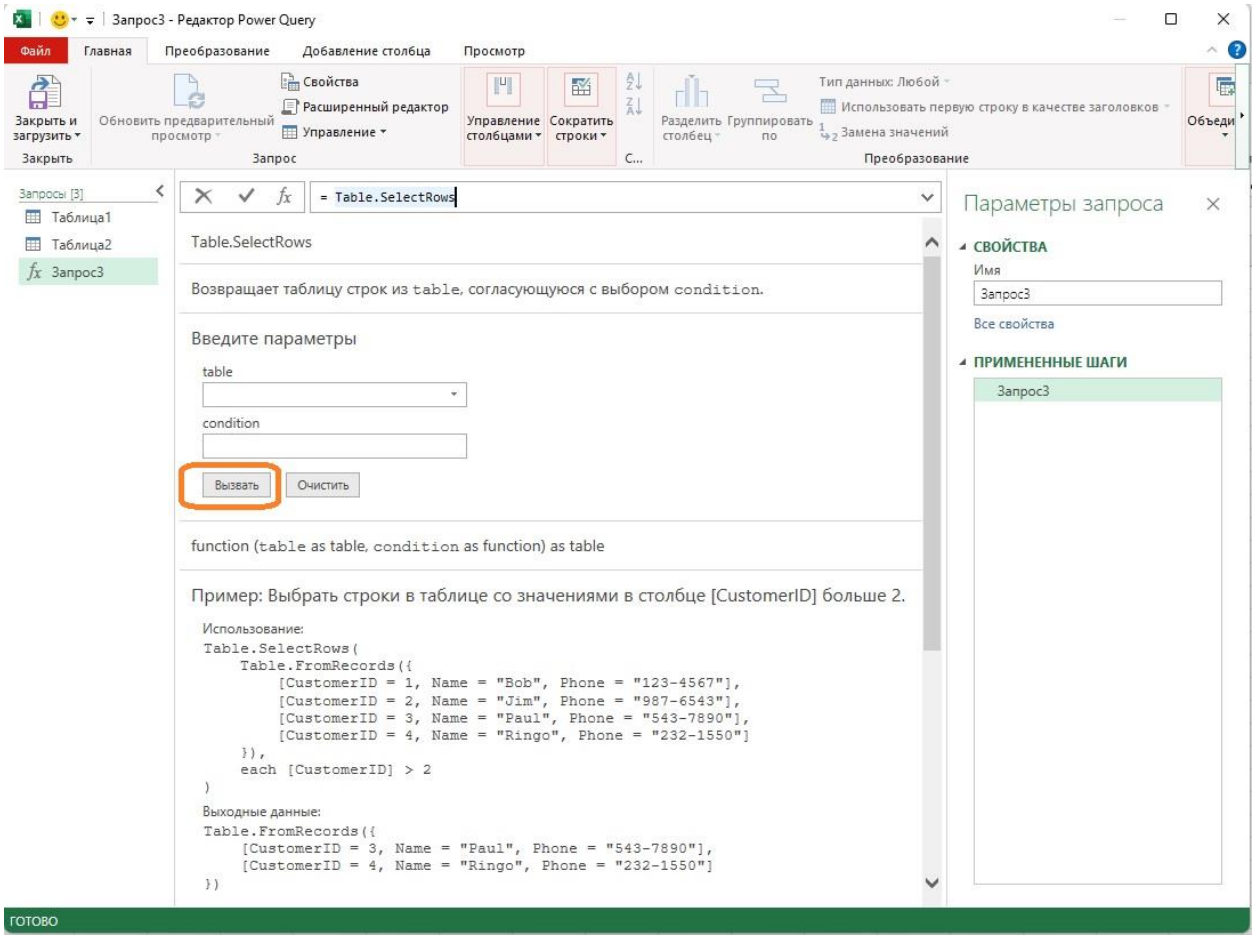


Рис. 6. Справка функции *Table.SelectRows*

Чувствительность к регистру

M чувствителен к регистру. Это требует повышенной аккуратности. Так например, Power Query распознает Excel.CurrentWorkbook() как стандартную функцию, а Excel.Currentworkbook() не будет распознана, поскольку использована строчная *w*.

Типы

Каждое значение в M имеет тип, будь то простой тип, такой как число, или структурированный тип, такой как таблица. Тип имеют и все переменные. Вам не нужно объявлять тип значения – M будет динамически определять его.

Строгая типизация означает, например, что выражение…

= "The number " & "1"

… работает нормально и возвращает текст

"Номер 1"

… а выражение…

= "The number " & 1

выдает ошибку, поскольку оператор & (который объединяет два текстовых значения) не может использоваться с текстовым значением и числом.

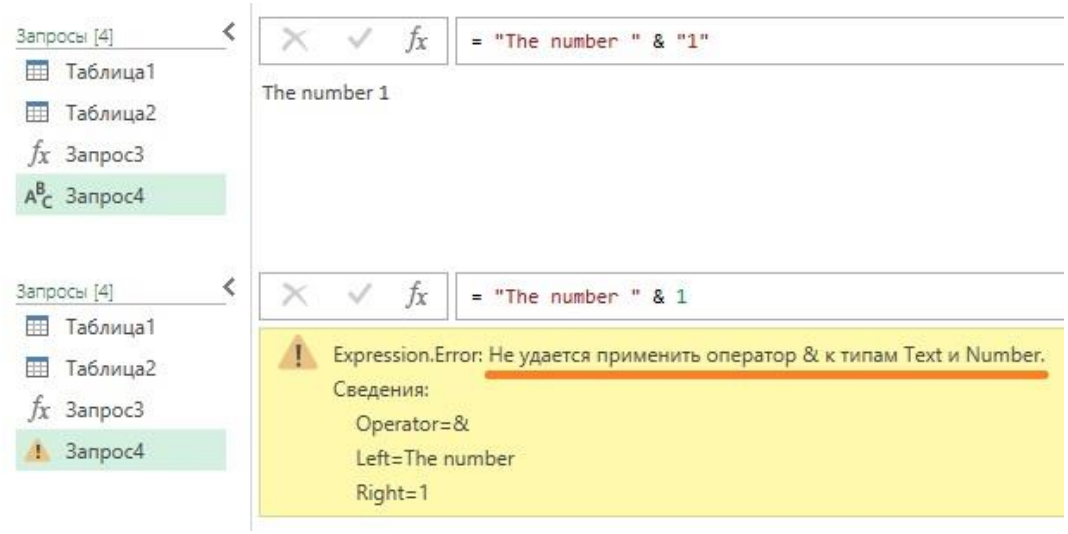


Рис. 7. Строгая типизация языка M может приводить к неожиданным ошибкам

В стандартной библиотеке доступно множество функций для преобразования одного типа в другой, и в нашем примере функция Number.ToText может быть использована для приведения числа в текст:

= "The number " & Number.ToText(1)

Вы можете проверить, является ли значение определенным типом, используя оператор *is*. Например, выражение…

= 1 is number

…вернет TRUE, а…

= 1 is text

… вернет FALSE.

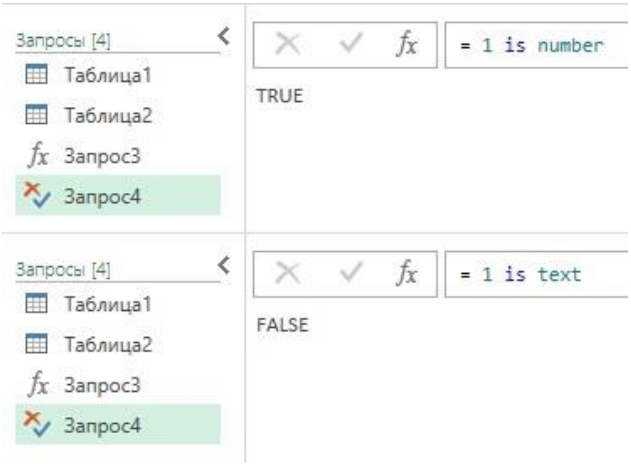


Рис. 8. Оператор *is* позволяет проверить, обладает ли значение определенным типом

Объявление типов date, time, datetime, datetimezone и duration

Если необходимо объявить значения указанных типов, можно использовать встроенные функции #date(), #datetime(), #datetimezone() и #duration(). Например, для #datetime() используется синтаксис:

#datetime(year as number, month as number, day as number, hour as number, minute as number, second as number) as datetime

Рис. 9. Объявление даты и времени

Комментарии

Существует два способа добавления комментариев к коду M: однострочным комментариям предшествует //, а многострочные комментарии начинается с /* и заканчивается */. Код в следующем листинге эквивалентен коду листинга 2, но дополнен комментариями:

Листинг 6

```
let
  //Загрузка данных из таблицы на листе Excel
  Источник = Excel.CurrentWorkbook(){[Name="Таблица2"]}[Content],
  //Автоматический шаг, изменяющий тип столбцов
  #"Измененный тип" = Table.TransformColumnTypes(
    Источник,
    {
      {"Месяц", type text},
      {"Продажи", Int64.Type}
    }
  ),
  //Фильтрация по столбцу "Продажа" для значений >= 5
  #"Строки с примененным фильтром" = Table.SelectRows(
    #"Измененный тип",
    each [Продажи] >= 5
  ),
  //Сортировка по столбцу "Продажа" по убыванию
  #"Сортированные строки" = Table.Sort(
    #"Строки с примененным фильтром",
    {{"Продажи", Order.Descending}}
  )
in
  #"Сортированные строки"
```

К сожалению, комментарии видны только в расширенном редакторе. В строке формул, как правило, они не отображаются. Если вы хотите, чтобы комментарий отражался в строке формул, его следует встроить в выражение:

```
  #"Сортированные строки" = Table.Sort(
    /* это встроенный комментарий */
    #"Строки с примененным фильтром",
    {{"Продажи", Order.Descending}}
  )
```

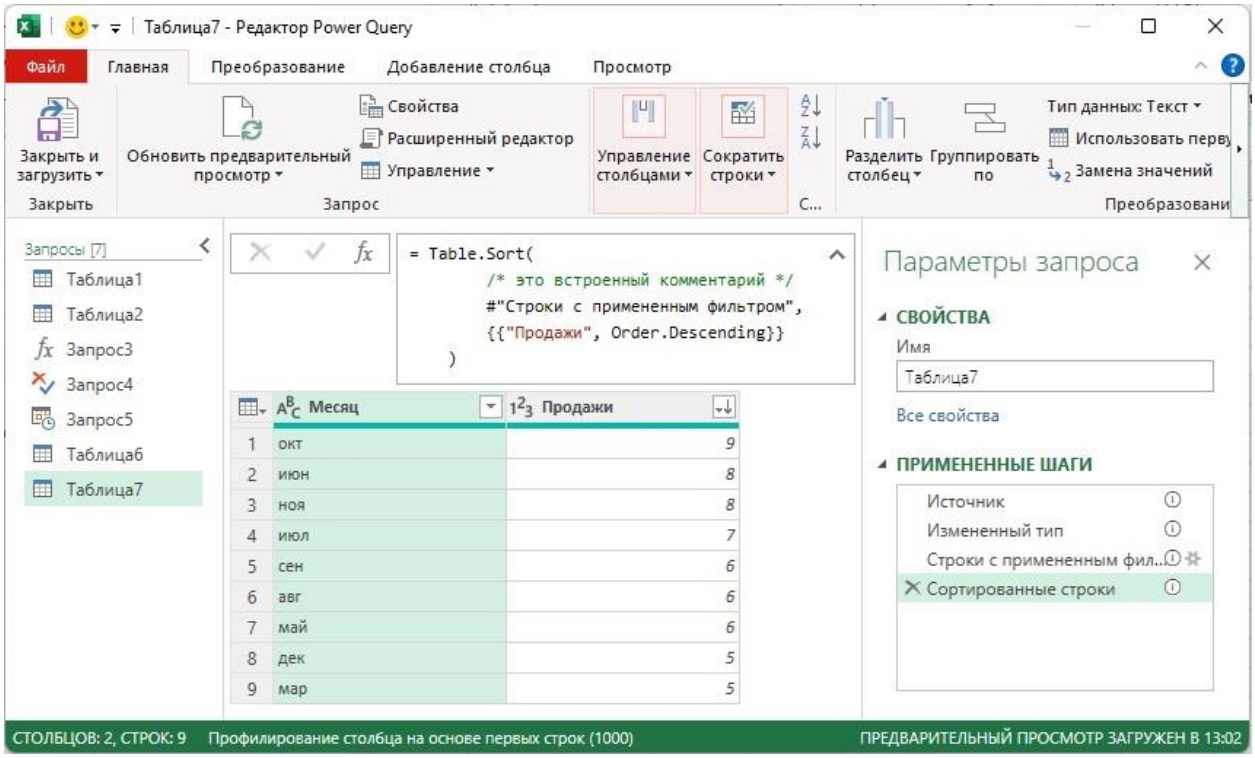


Рис. 10. Встроенный комментарий

Захват ошибок

При работе с интерфейсом в редакторе PQ существует стандартная опция фильтрации строк, содержащих значения ошибок. Однако гораздо лучше ловить ошибки на уровне выражений. Это можно сделать, используя конструкцию…

try ... otherwise expression

Ниже показана таблица в Excel, в которой один столбец содержит числа, а другой – смесь чисел и текста.

	A	B
1	Числа	Числа, как текст
2	1	4
3	2	Apples
4	3	Oranges
5	4	1

Рис. 11. Таблица, содержащая числа и текст

Импортируем таблицу в Power Query. Редактор PQ автоматически добавит шаг *Измененный тип…*

= Table.TransformColumnTypes(Источник,{{"Числа", Int64.Type}, {"Числа, как текст", type text}})

… в котором назначит столбцу *Числа* тип Int64 (64-разрядное целое число), а столбцу *Числа, как текст* – тип text. Если теперь добавить настраиваемый столбец…

```
#"Добавлен пользовательский объект" = Table.AddColumn(
    #"Измененный тип",
    "Пользовательский",
    each [Числа]+Number.FromText([#"Числа, как текст"])
)
```

… получим ошибку:

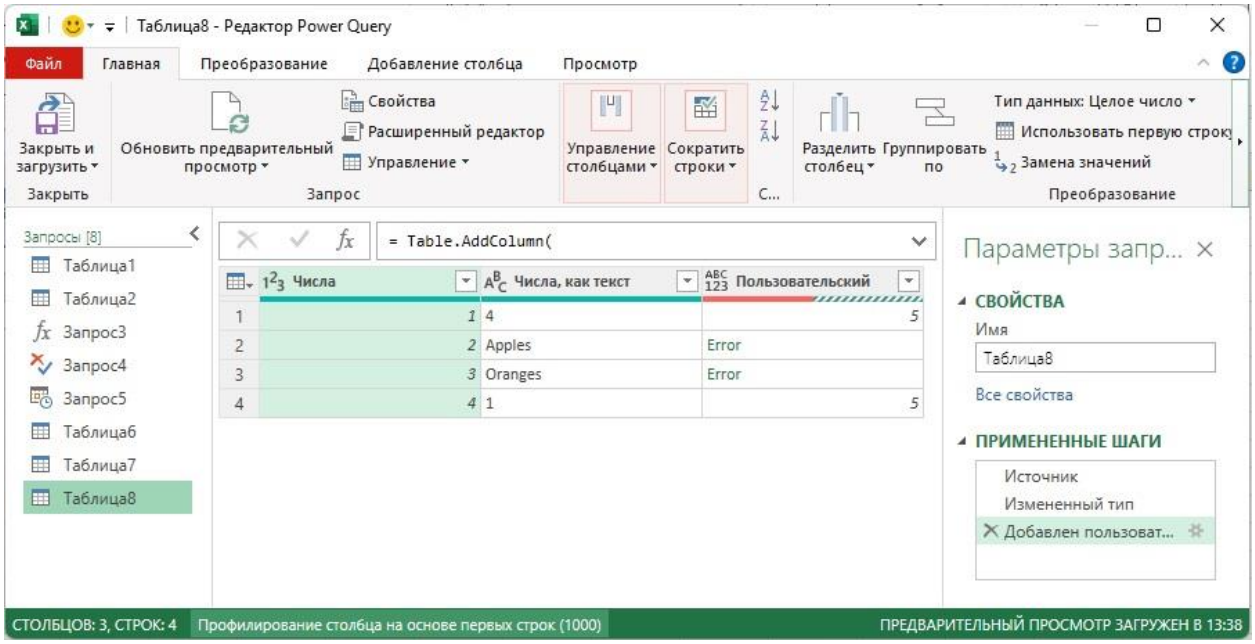


Рис. 12. Настраиваемый столбец, содержащий значения ошибок

Функция Number.FromText() пытается взять текстовое значение и преобразовать его в число. Если текстовое значение не может быть преобразовано, функция возвращает ошибку.

Изменим выражение, которое используется при добавлении настраиваемого столбца:

```
#"Добавлен пользовательский объект" = Table.AddColumn(
    #"Измененный тип",
    "Пользовательский",
    each [Числа] +
    (
        try Number.FromText([#"Числа, как текст"])
        otherwise 0
    )
)
```

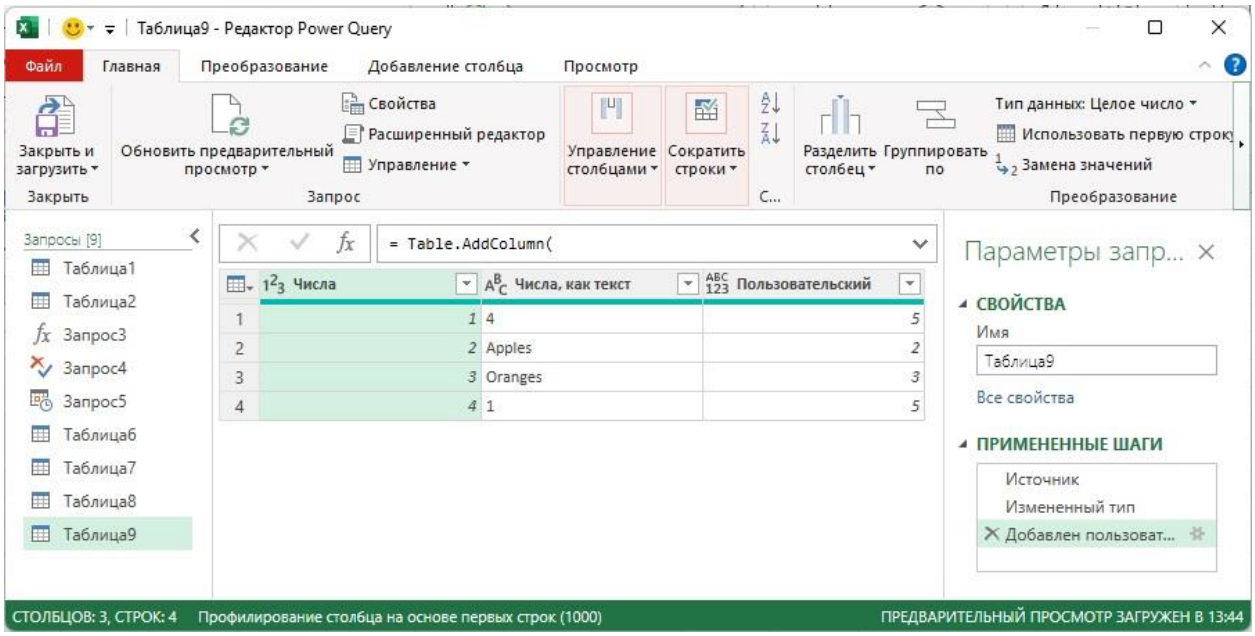


Рис. 13. Настраиваемый столбец, в котором ошибки были захвачены

Выражение *try* вычисляет переданное ему выражение, и, если ошибка не возникает, возвращает значение этого выражения. Если же возникает ошибка, то возвращается значение, указанное в предложении *other*.

Условная логика

Условная логика в M использует конструкцию…

if ... then ... else

Продолжая предыдущий пример, добавим условный столбец:

```
#"Условный столбец добавлен" = Table.AddColumn(
    #"Добавлен пользовательский объект",
    "Пользовательский.1",
    each if [Числа] <= 2 then "2 или менее" else "Более 2"
)
```

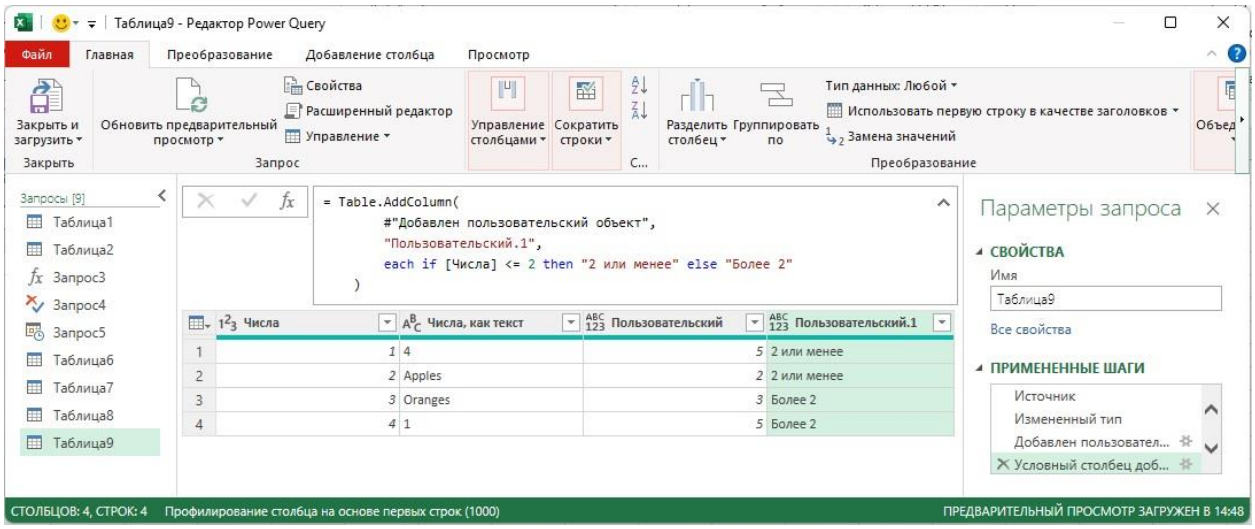


Рис. 14. Условный столбец, использующий логику if...then...else

Аналога конструкции *Case*, имеющейся в других языках, в M нет, но можно использовать вложенные выражения *if*. Например:

if [Числа] > 3 then "Более 3" else if [Числа] > 2 then "Более 2" else "2 или менее"

Списки

Выше были кратко описаны примитивные типы данных – числа, даты, текст и др. Как правило, именно они отражаются в ячейках. Но, работая в PQ, вы также могли видеть в ячейках значения типа *table* или *record*. Это структурированные типы – объекты, которые содержат много значений, связанных друг с другом определенным образом.

Список — это упорядоченная последовательность значений, в чем-то похожая на массив в других языках программирования. Элементы в списке могут быть любого типа, и у вас даже могут быть списки списков. Списки полезны как средство для достижения цели: вы будете использовать их в сложных вычислениях, но маловероятно, что список будет конечным результатом запроса.

Определение списков вручную

Списки можно определить вручную как набор значений разделенных запятыми, окруженный фигурными скобками. Например, {1,2,3} определяет список значений 1, 2 и 3, а {"A", "B", "C"} определяет список со значениями A, B и C. Можно определить список последовательных целых чисел, используя синтаксис *x..y*. Например, {1..4} возвращает список значений 1, 2, 3 и 4. Можно задать пустой список {}. Каждый элемент в списке может быть списком, поэтому {{1,2},{3,4}} определяет список, содержащий два элемента, каждый из которых является списком, содержащим два элемента.

Работа со списками в редакторе запросов

Когда вы переходите к шагу в запросе, который возвращает список, редактор запросов в области предварительного просмотра покажет содержимое этого списка, и на ленте появится вкладка *Средства для списков*:

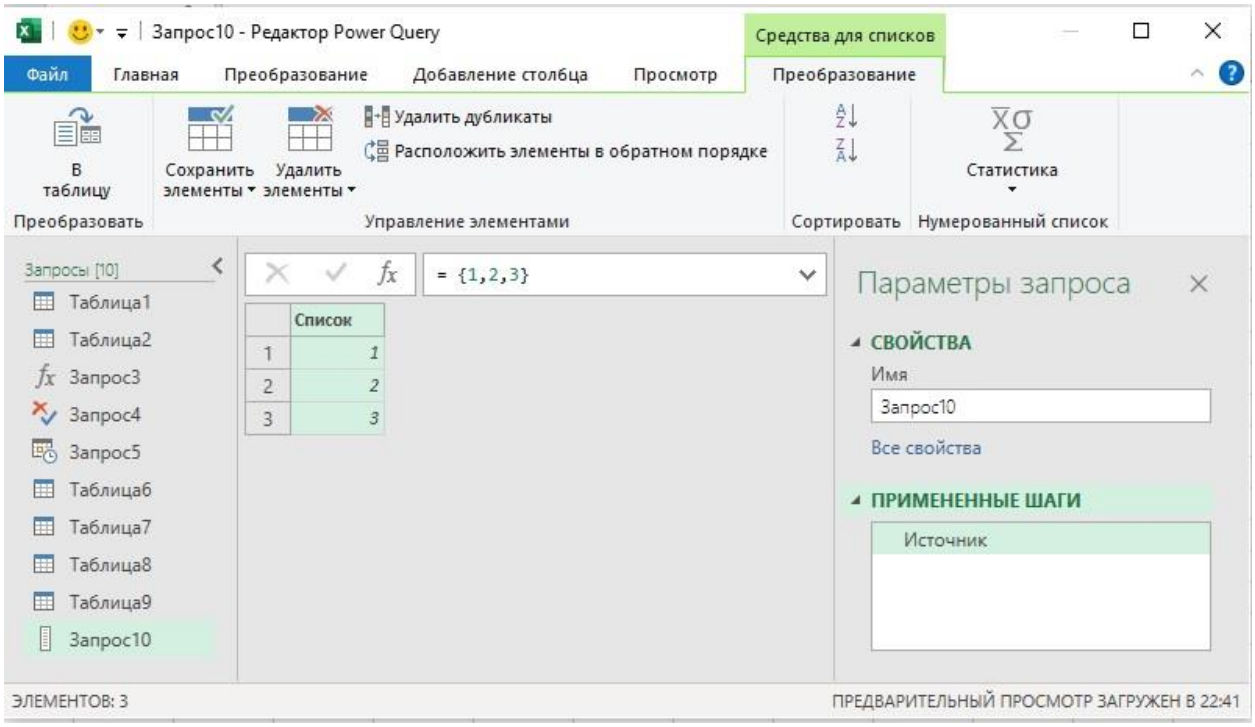


Рис. 15. Список в редакторе запросов

При нажатии кнопки *В таблицу* будет создан новый шаг, который преобразует список в таблицу.

Функции, создающие списки

Для создания более сложных числовых последовательностей можно использовать функцию List.Numbers():

List.Numbers(start as number, count as number, optional increment as nullable number) as list

Параметр *start* – это число, с которого нужно начать, *count* – количество возвращаемых значений в списке, *increment* – разница между значениями в списке. Например, List.Numbers(5,4,3) возвращает список {5,8,11,14}. Аналогично, List.Dates() возвращает диапазон дат, поэтому List.Dates(#date(2014,1,1), 3, #duration(1,0,0,0)) возвращает список, содержащий первые три даты в январе 2014 года.

Существует целый ряд других стандартной функций, которые преобразуют значения в списки. Например, Table.ToList() преобразует таблицу в список, а Table.Column() возвращает все значения столбца таблицы в виде списка.

Агрегирование значений в списках

Одна из основных причин, по которой вы захотите сохранить последовательность чисел в виде списка, заключается в том, чтобы иметь возможность агрегировать эти значения. Все основные методы агрегирования поддерживаются в M. Функция List.Count() возвращает число элементов в списке, а функция List.Sum() возвращает сумму чисел в списке. Другие функции агрегирования: List.Product(), List.Average(), List.Mode(), List.StandardDeviation(), List.Max() и List.Min(). Некоторые из этих функций работают и со списками, содержащими типы данных отличные от числовых. Например, List.Sum() суммирует значения типа duration (длительности).

Сортировка списков

Для сортировки элементов в списке можно использовать функцию List.Sort():

List.Sort(list as list, optional comparisonCriteria as any) as list

Первый параметр – это список для сортировки, а второй – управляет тем, как происходит сортировка. В большинстве случаев все, что вам нужно будет указать во втором параметре, это Order.Descending или Order.Ascending, хотя можно использовать и более сложные критерии упорядочения. Например List.Sort({1,2,3,4,5}, Order.Descending) возвращает список {5,4,3,2,1}, а List.Sort({"Helen", "Natasha", "Mimi", "Chris"}, Order.Ascending) возвращает список {"Chris", "Helen",

"Mimi", "Natasha"}. List.Reverse() изменяет порядок элементов в списке, так что List.Reverse({3,5,4}) возвращает {4,5,3}.

Фильтрация списков

Существует несколько функций, которые можно использовать для фильтрации элементов в списке. List.First() и List.Last() возвращают первый и последний элементы в списке, а List.FirstN() и List.LastN() возвращают списки, которые представляют собой заданное количество значений из начала и конца списка. List.Distinct() возвращает список уникальных значений.

Более сложные сценарии фильтрации можно обрабатывать с помощью функции List.Select():

List.Select(list as list, selection as function) as list

Первым параметром является список, подлежащий фильтрации; вторым – выражение, которое является функцией. Эта тема будет подробно рассмотрена далее, но пока вам будет приятно узнать, что базовый синтаксис для этого параметра довольно прост. Например…

List.Select({1,2,3,4,5}, each _>2)

…возвращает список {3,4,5}. Оператор *each* определяет безымянную функцию, единственный параметр которой представлен символом подчеркивания _ и имеет тип any. Поэтому each _>2 является функцией, которая возвращает значение ИСТИНА, когда переданное ей значение больше 2 и ЛОЖЬ в противном случае. Когда выражение each _>2 используется во втором параметре функции List.Select(), каждый элемент списке последовательно поступает на вход выражения each _>2. А возвращаются только элементы, для которых функция возвращает ИСТИНА.

Записи

Думайте о записи как о таблице, в которой только одна строка. Запись – это упорядоченная последовательность полей, где каждое поле имеет имя и одно значение (любого типа). Запись можно определить вручную, как разделенный запятыми список пар поле/значение, заключенных в квадратные скобки. Например:

[firstname="Chris", lastname="Webb", gender="Male", town="Amersham"]

Как и в случае со списком, при просмотре шага, возвращающего запись, в поле предварительного просмотра редактора запросов можно увидеть каждое поле записи:

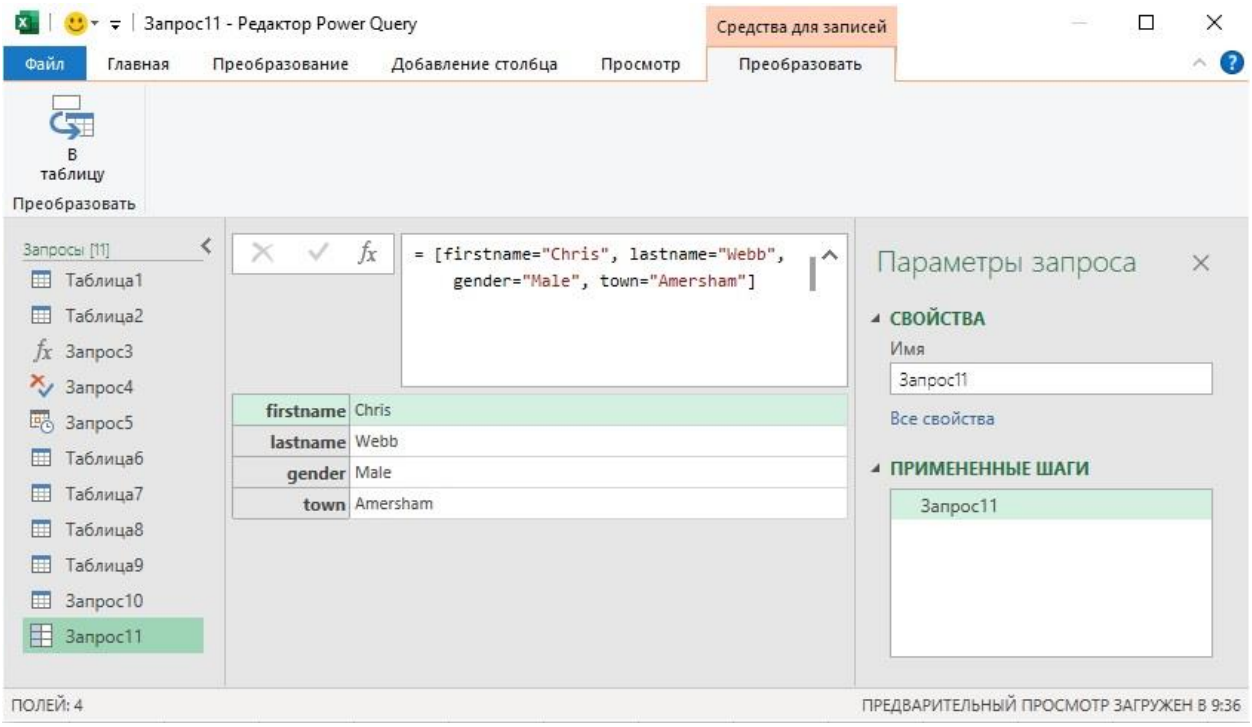


Рис. 16. Запись в редакторе запросов

Нажатие кнопки *В таблицу* на панели инструментов преобразует запись в таблицу с одной строкой для каждого поля:

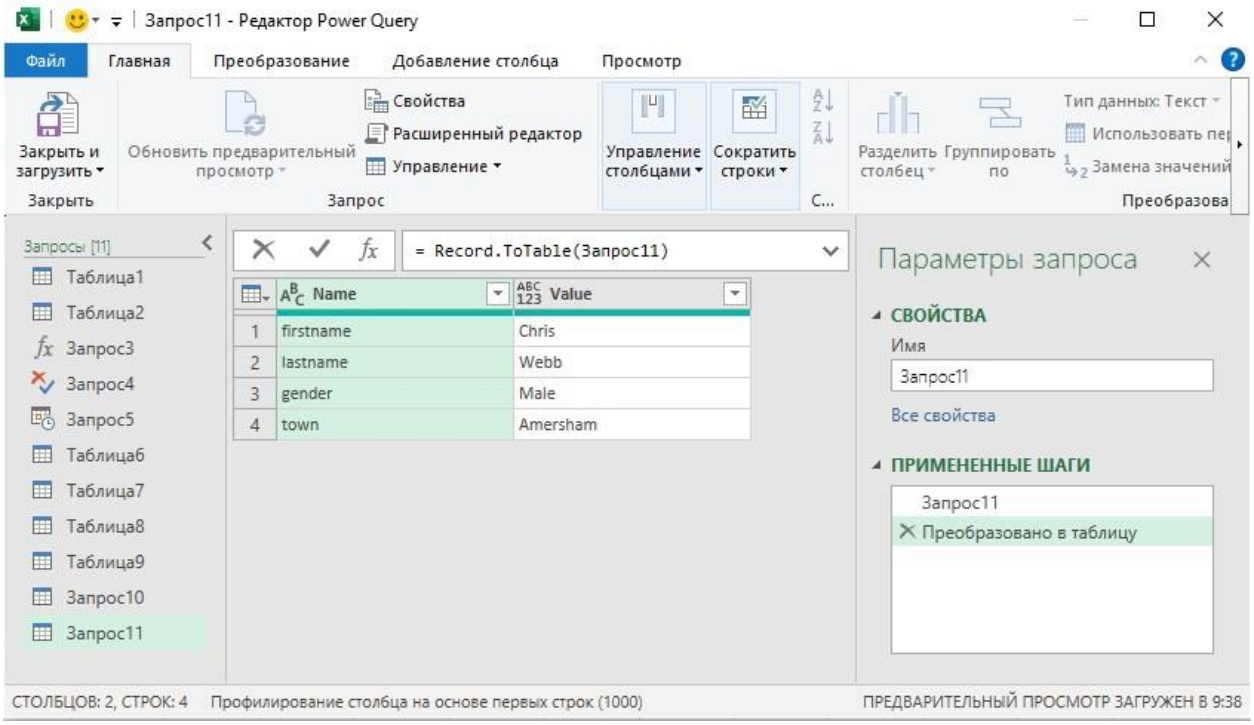


Рис. 17. Таблица, созданная из записи

Таблицы

Таблица является наиболее важным из всех структурированных типов. В большинстве случаев данные, импортируемые в запрос, находятся в табличной форме; большинство шагов запроса возвращают таблицы; и вы почти всегда возвращаете таблицы из запросов. Таблица состоит из данных, организованных в строки и столбцы, где каждый столбец имеет уникальное имя и содержит данные определенного типа.

Создание таблиц

Если в строке формул вы посмотрите на код M для первого шага в большинстве запросов вы увидите выражение, которое возвращает таблицу из источника данных, находящегося вне Power Query. Например, в листинге 1 шаг *Источник* использует выражение…

```
Источник = Excel.CurrentWorkbook(){[Name="Таблица1"]}[Content],
```

…для возврата содержимого *Таблица1* в текущей книге Excel. Стандартная библиотека содержит множество подобных функций, которые позволяют извлекать таблицы данных из разных источников.

Таблицы также можно создавать без внешнего источника внутри запроса. Вы можете использовать функцию #table(), которая принимает два параметра: список, содержащий заголовки столбцов, и список списков, содержащий строки таблицы. Например, выражение…

```
= #table({"Fruit", "Sales"}, {{"Apples", 1}, {"Oranges", 2} })
```

…возвращает таблицу:

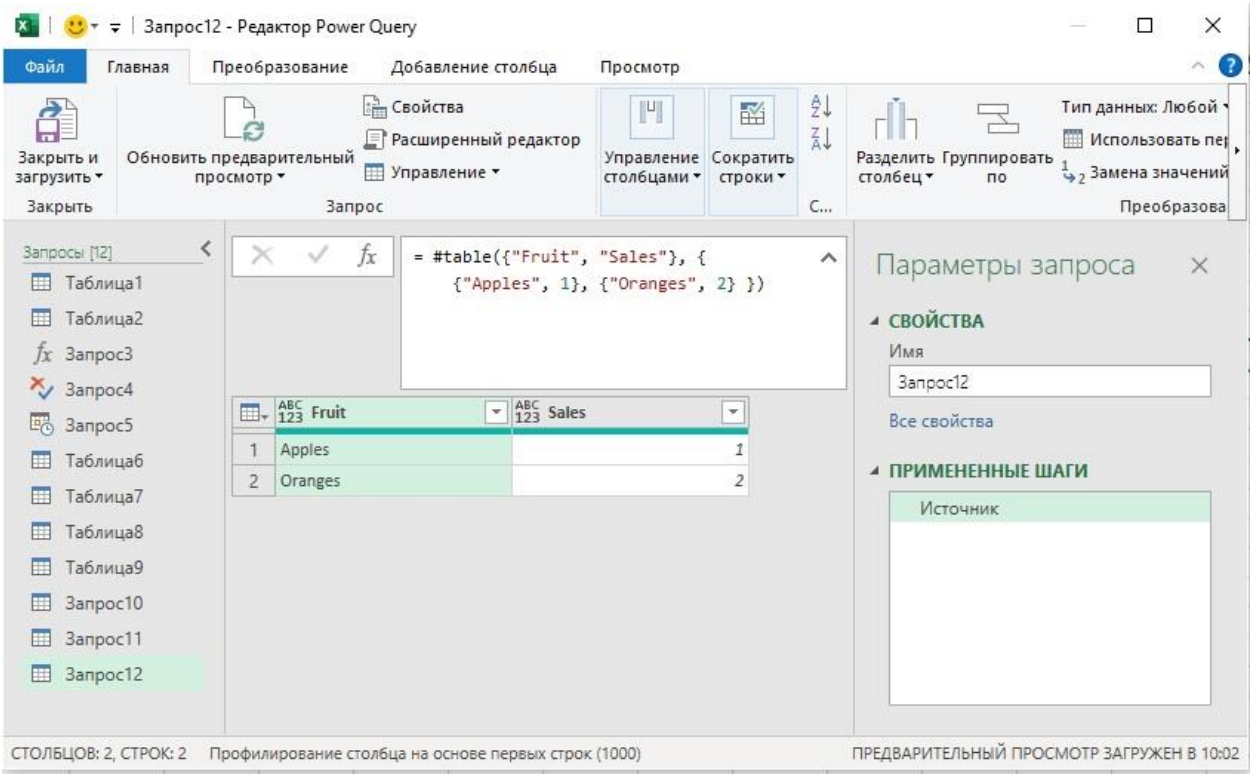


Рис. 18. Таблица, созданная с помощью встроенной функции #table

Все столбцы в таблицах, созданных таким образом, имеют тип *any*. Можно указать иные типы для столбцов, используя запись в первом параметре вместо списка:

#table(type table [Fruit = text, Sales = number], {{"Apples", 1},{"Oranges", 2}})

Для создания таблиц можно использовать и иные функции. Например, Table.FromRows()…

Table.FromRows({ {"Apples", 1}, {"Oranges", 2} }, {"Fruit", "Sales"})

…или Table.FromRecords()…

Table.FromRecords({[Fruit="Apples", Sales=1], [Fruit="Oranges", Sales=2]})

Агрегирование данных в таблицах

Рассмотрим некоторые варианты агрегирования на примеры следующих данных:

	A	B	C
1	Квартал	Месяц	Продажи
2	Q1	янв	2
3	Q1	фев	3
4	Q1	мар	5
5	Q2	апр	3
6	Q2	май	6
7	Q2	июн	8
8	Q3	июл	7
9	Q3	авг	6
10	Q3	сен	6
11	Q4	окт	9
12	Q4	ноя	8
13	Q4	дек	5

Рис. 19. Пример таблицы для агрегирования

Для нахождения числа строк таблицы используется функция Table.RowCount(). Следующий код загружает данные из таблицы Excel и подсчитывает количество строк в ней:

Листинг 13

```
let
   //Загрузка данных из таблицы Excel
   Источник = Excel.CurrentWorkbook(){[Name="Таблица4"]}[Content],
   //Автоматическое изменение типа столбцов
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Квартал", type text},
         {"Месяц", type text},
         {"Продажи", Int64.Type}
      }
   ),
   //Подсчет числа строк в таблице
   CountRowsInTable = Table.RowCount(#"Измененный тип")
in
   CountRowsInTable
```

Этот запрос возвращает число, а не таблицу с одной ячейкой, содержащей значение 12. Хотя если выходные данные запроса загружены на лист Excel или в модель данных, результат запроса будет рассматриваться как таблица с одной строкой и одним столбцом.

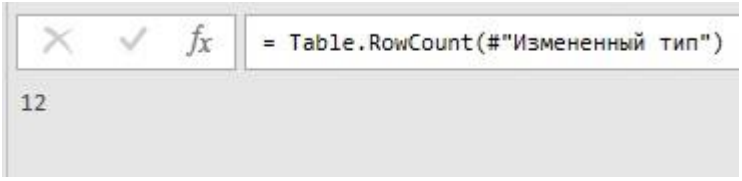


Рис. 20. Выходные данные Table.RowCount в редакторе запросов

Для получения выходных данных типа *table* и/или выполнения других типов агрегирования используется функция Table.Group():

```
Table.Group(
   table as table,
   key as any,
   aggregatedColumns as list,
   optional groupKind as nullable number,
   optional comparer as nullable function
) as table
```

Первый параметр – это таблица, подлежащая агрегированию, второй – список ключевых столбцов для группировки, третий – указывает способ агрегирования данных. Следующий код аналогичен листингу 13, но подсчитает количество строк в таблице с помощью иной функции:

Листинг 14

```
let
   //Загрузка данных из таблицы Excel
   Источник = Excel.CurrentWorkbook(){[Name="Таблица4"]}[Content],
   //Автоматическое изменение типа столбцов
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Квартал", type text},
         {"Месяц", type text},
         {"Продажи", Int64.Type}
      }
   ),
   //Подсчет числа строк в таблице
   GroupedRows = Table.Group(
```

```
    #"Измененный тип",
    {},
    {{"Число строк",
      each Table.RowCount(_),
      type number
    }}
  )
in
  GroupedRows
```

Здесь первый параметр функции Table.Group() – таблица, полученная на предыдущем шаге – #"Измененный тип". Второй параметр – пустой список, так как нет столбца, по которому осуществляется группировка. Третий параметр – список, содержащий один элемент, который сам по себе является списком. Последний содержит три элемента: имя столбца выходной таблицы, счетчик each Table.RowCount() и тип столбца. Вот что будет на выходе:

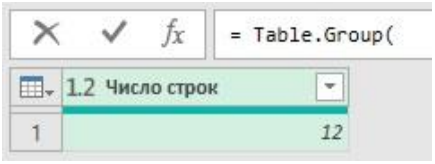


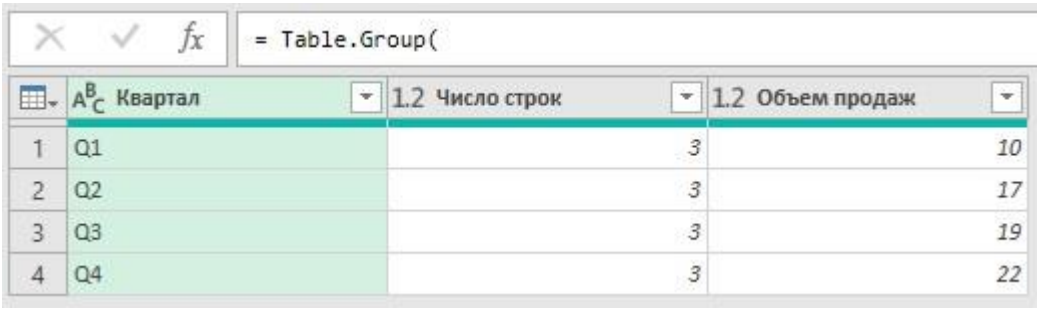
Рис. 21. Результат работы листинга 14

Ниже показан более сложный пример, с группировкой по кварталам:

Листинг 15

```
let
  //Загрузка данных из таблицы Excel
  Источник = Excel.CurrentWorkbook(){[Name="Таблица4"]}[Content],
  //Автоматическое изменение типа столбцов
  #"Измененный тип" = Table.TransformColumnTypes(
    Источник,
    {
      {"Квартал", type text},
      {"Месяц", type text},
      {"Продажи", Int64.Type}
    }
  ),
  //Подсчет числа строк и продаж по кварталам
  GroupedRows = Table.Group(
    #"Измененный тип",
    {"Квартал"},
    {
      {"Число строк",
        each Table.RowCount(_),
        type number
      },
      {"Объем продаж",
        each List.Sum([Продажи]),
        type number
      }
    }
  )
in
  GroupedRows
```

Здесь второй параметр не пустой, а список с одним элементом – "Квартал" – имя столбца, который используется для группировки. Третий параметр включает не один, а два списка. Второй используется для суммирования по столбцу [Продажи]. Вот что будет на выходе:



	Квартал	Число строк	Объем продаж
1	Q1	3	10
2	Q2	3	17
3	Q3	3	19
4	Q4	3	22

Рис. 22. Подсчет и суммирование по кварталам с помощью Table.Group()

У функции Table.Group() есть необязательный четвертый параметр, который управляет типом группировки. Значением по умолчанию для параметра является GroupKind.Global, и это означает, что при группировке по столбцу порядок сортировки таблицы не имеет значения: все строки, связанные с отдельным значением в столбце, агрегируются вместе. GroupKind.Local, другое возможное значение, означает, что Table.Group() учитывает порядок сортировки таблиц и группирует только по непрерывным диапазонам значений. В качестве примера рассмотрим таблицу:

	A	B	C	D
1	Дата	Рабочий/выходной	Продажи	Штуки
2	01.апр.2022	Рабочий	5	2
3	02.апр.2022	Выходной	4	1
4	03.апр.2022	Выходной	2	1
5	04.апр.2022	Рабочий	6	1
6	05.апр.2022	Рабочий	8	5
7	06.апр.2022	Рабочий	6	3
8	07.апр.2022	Рабочий	4	2
9	08.апр.2022	Рабочий	5	1
10	09.апр.2022	Выходной	5	2
11	10.апр.2022	Выходной	2	3
12	11.апр.2022	Рабочий	1	1
13	12.апр.2022	Рабочий	3	1

Рис. 23. Таблица продаж

Найдем количество строк, сгруппированных по столбцу *Рабочий/выходной* (четвертый параметр со значением по умолчанию задан явно для ясности):

Листинг 16

```
let
  Источник = Excel.CurrentWorkbook(){[Name="Таблица5"]}[Content],
  #"Измененный тип" = Table.TransformColumnTypes(
    Источник,
    {
      {"Дата", type datetime},
      {"Рабочий/выходной", type text},
      {"Продажи", Int64.Type},
      {"Штуки", Int64.Type}
    }
  ),
  //Подсчет числа строк по дням недели
  GroupedRows = Table.Group(
    #"Измененный тип",
    {"Рабочий/выходной"},
    {{"Число строк",
      each Table.RowCount(_),
      type number
    }},
```

```
    GroupKind.Global
  )
in
  GroupedRows
```

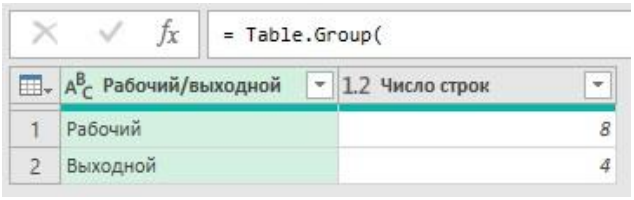


Рис. 24. Число строк с параметром GroupKind.Global

Если четвертый параметр равен GroupKind.Local, то функция Table.Group() будет агрегировать только связанные диапазоны. Как только в столбце *Рабочий/выходной* появляется новое значение, начинается новое агрегирование:

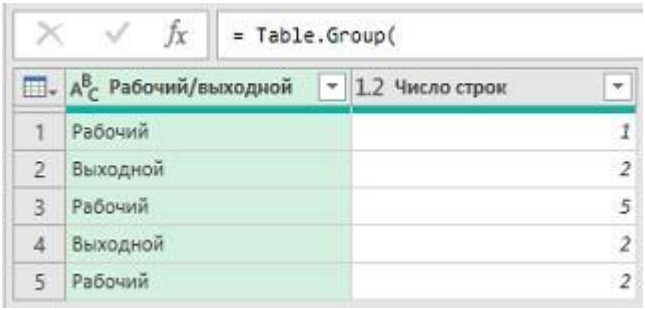


Рис. 25. Результат работы функции Table.Group() с параметром GroupKind.Local

Сортировка таблиц

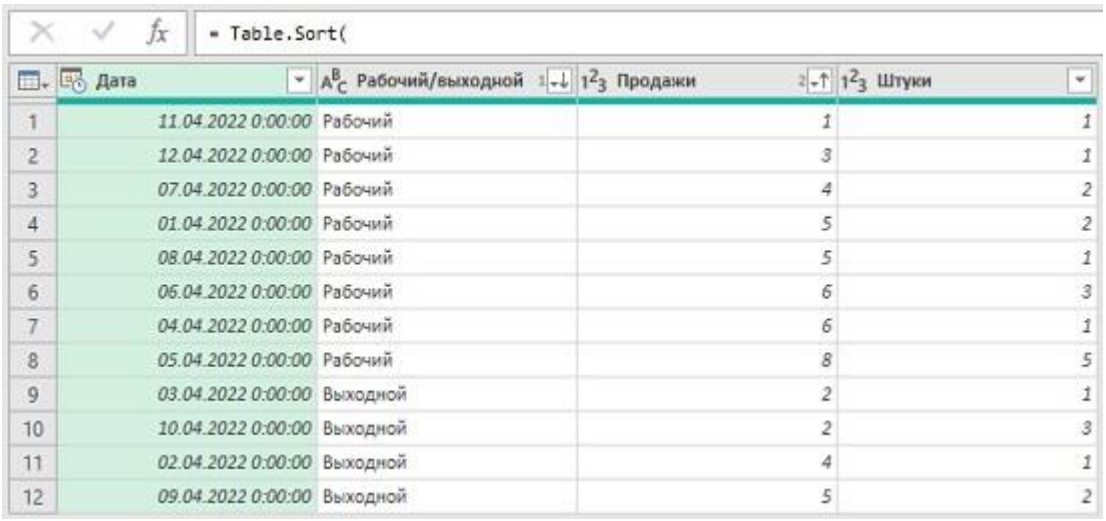
Параметры сортировки таблицы очень похожи на параметры сортировки списка. Table.Sort() сортирует строки в таблице в порядке возрастания или убывания по одному или нескольким столбцам, а Table.ReverseRows() изменяет порядок строк в таблице.

Воспользуемся данными с рис. 23 и отсортируем таблицу сначала в порядке убывания по столбцу *Рабочий/выходной*, а затем в порядке возрастания столбцу *Продажи*:

Листинг 18

```
let
  Источник = Excel.CurrentWorkbook(){[Name="Таблица5"]}[Content],
  #"Измененный тип" = Table.TransformColumnTypes(
    Источник,
    {
      {"Дата", type datetime},
      {"Рабочий/выходной", type text},
      {"Продажи", Int64.Type},
      {"Штуки", Int64.Type}
    }
  ),
  //Сортировка строк
  SortRows = Table.Sort(
    #"Измененный тип",
    {
      {"Рабочий/выходной", Order.Descending},
      {"Продажи", Order.Ascending}
    }
  )
in
  SortRows
```

Обратите внимание, что во втором параметре функции Table.Sort список списков используется для передачи комбинаций столбцов и используемых порядков сортировки. На выходе получим:



	Дата	Рабочий/выходной	Продажи	Штуки
1	11.04.2022 0:00:00	Рабочий	1	1
2	12.04.2022 0:00:00	Рабочий	3	1
3	07.04.2022 0:00:00	Рабочий	4	2
4	01.04.2022 0:00:00	Рабочий	5	2
5	08.04.2022 0:00:00	Рабочий	5	1
6	06.04.2022 0:00:00	Рабочий	6	3
7	04.04.2022 0:00:00	Рабочий	6	1
8	05.04.2022 0:00:00	Рабочий	8	5
9	03.04.2022 0:00:00	Выходной	2	1
10	10.04.2022 0:00:00	Выходной	2	3
11	02.04.2022 0:00:00	Выходной	4	1
12	09.04.2022 0:00:00	Выходной	5	2

Рис. 26. Таблица, отсортированная с помощью Table.Sort()

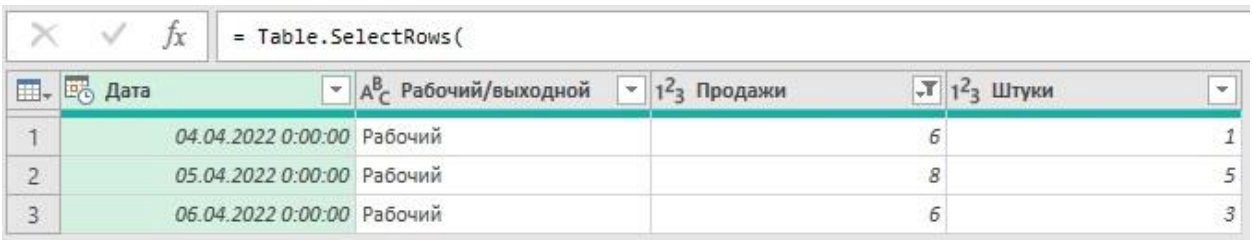
Фильтрация таблиц

Функции фильтрации таблиц аналогичны функциям фильтрации списков. Table.First() и Table.Last() возвращают первую и последнюю строки из таблицы; Table.FirstN() и Table.LastN() возвращают первые и последние N строк из таблицы. Table.SelectRows() работает аналогично List.Select(), хотя во втором параметре можно ссылаться на значение в столбце таблицы без использования нотации _. Например, используя таблицу на рис. 23, можно удалить строки, где столбец *Продажи* содержит значения меньше 6:

Листинг 19

```
let
   Источник = Excel.CurrentWorkbook(){[Name="Таблица5"]}[Content],
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Дата", type datetime},
         {"Рабочий/выходной", type text},
         {"Продажи", Int64.Type},
         {"Штуки", Int64.Type}
      }
   ),
   //Фильтрация строк
   SelectRows = Table.SelectRows(
      #"Измененный тип",
      each [Продажи] > 5
   )
in
   SelectRows
```

Здесь [Продажи] можно использовать для ссылки на значение в столбце *Продажи* для текущей строки при итерации функции по строкам таблицы. На выходе:



	Дата	Рабочий/выходной	Продажи	Штуки
1	04.04.2022 0:00:00	Рабочий	6	1
2	05.04.2022 0:00:00	Рабочий	8	5
3	06.04.2022 0:00:00	Рабочий	6	3

Рис. 27. Таблица, отфильтрованная с помощью Table.SelectRows()

Отмена свертывания столбцов

В языке M для этих целей есть две функции: Table.UnPivot(), отменяющая сведение выбранного списка столбцов, и Table.UnPivotOtherColumns(), которая отменяет свертывание столбцы, кроме выбранных:

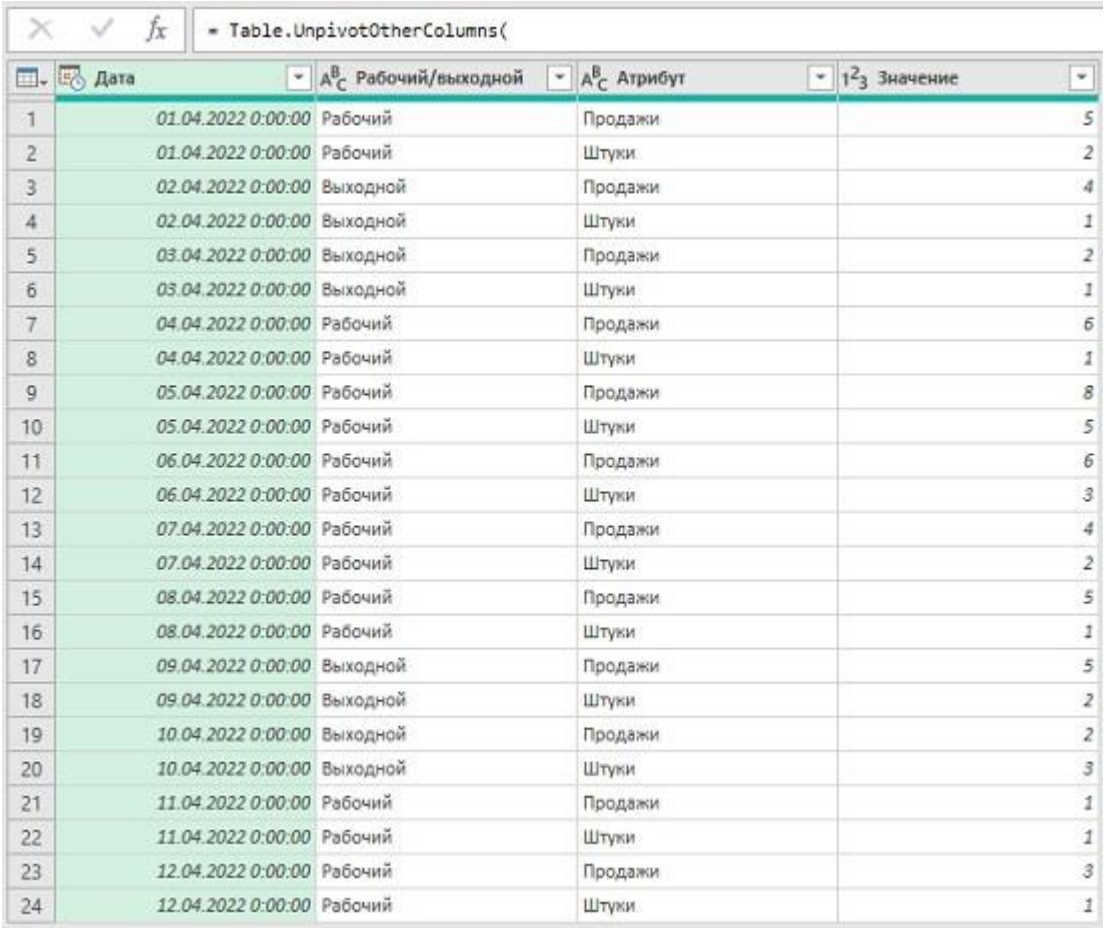
Table.Unpivot(table as table, pivotColumns as list, attributeColumn as text, valueColumn as text) as table

Table.UnpivotOtherColumns(table as table, pivotColumns as list, attributeColumn as text, valueColumn as text) as table

Опять возьмем таблицу с рис. 23. Следующий код вместо столбцов *Продажи* и *Штуки* вернет столбцы атрибутов и значений:

Листинг 20

```
let
   Источник = Excel.CurrentWorkbook(){[Name="Таблица5"]}[Content],
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Дата", type datetime},
         {"Рабочий/выходной", type text},
         {"Продажи", Int64.Type},
         {"Штуки", Int64.Type}
      }
   ),
   //Отмена свертывания столбцов
   Unpivoting = Table.UnpivotOtherColumns(
      #"Измененный тип",
      {"Дата", "Рабочий/выходной"},
      "Атрибут", "Значение"
   )
in
   Unpivoting
```



= Table.UnpivotOtherColumns(

	Дата	Рабочий/выходной	Атрибут	Значение
1	01.04.2022 0:00:00	Рабочий	Продажи	5
2	01.04.2022 0:00:00	Рабочий	Штуки	2
3	02.04.2022 0:00:00	Выходной	Продажи	4
4	02.04.2022 0:00:00	Выходной	Штуки	1
5	03.04.2022 0:00:00	Выходной	Продажи	2
6	03.04.2022 0:00:00	Выходной	Штуки	1
7	04.04.2022 0:00:00	Рабочий	Продажи	6
8	04.04.2022 0:00:00	Рабочий	Штуки	1
9	05.04.2022 0:00:00	Рабочий	Продажи	8
10	05.04.2022 0:00:00	Рабочий	Штуки	5
11	06.04.2022 0:00:00	Рабочий	Продажи	6
12	06.04.2022 0:00:00	Рабочий	Штуки	3
13	07.04.2022 0:00:00	Рабочий	Продажи	4
14	07.04.2022 0:00:00	Рабочий	Штуки	2
15	08.04.2022 0:00:00	Рабочий	Продажи	5
16	08.04.2022 0:00:00	Рабочий	Штуки	1
17	09.04.2022 0:00:00	Выходной	Продажи	5
18	09.04.2022 0:00:00	Выходной	Штуки	2
19	10.04.2022 0:00:00	Выходной	Продажи	2
20	10.04.2022 0:00:00	Выходной	Штуки	3
21	11.04.2022 0:00:00	Рабочий	Продажи	1
22	11.04.2022 0:00:00	Рабочий	Штуки	1
23	12.04.2022 0:00:00	Рабочий	Продажи	3
24	12.04.2022 0:00:00	Рабочий	Штуки	1

Рис. 28. Таблица после отмены свертывания столбцов

Сведение столбцов

Видимо, в 2013–2014 гг., когда Крис писал книгу, операция, обратная отмене свертывания столбцов, была недоступна в интерфейсе. В 2022 г., когда я пишу эту заметку, такая опция есть: *Преобразование –> Столбец сведения*. Но, и тогда, и сейчас «за кулисами» стоит функция Table.Pivot():

```
Table.Pivot(
   table as table,
   pivotValues as list,
   attributeColumn as text,
   valueColumn as text,
   optional aggregationFunction as nullable function
) as table
```

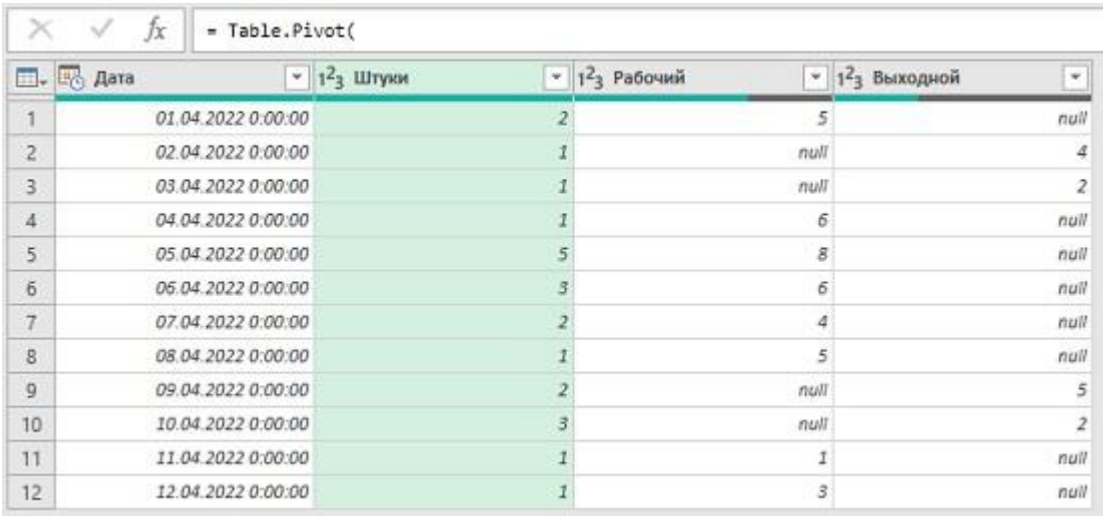
Используя таблицу на рис. 23, применим сведение столбцов, чтобы вместо одного столбца *Продажи* у нас было два: один для продаж в будний день и один для продаж в выходные:

Листинг 21

```
let
   Источник = Excel.CurrentWorkbook(){[Name="Таблица5"]}[Content],
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Дата", type datetime},
         {"Рабочий/выходной", type text},
         {"Продажи", Int64.Type},
         {"Штуки", Int64.Type}
      }
   ),
   #"Сведенный столбец" = Table.Pivot(
```

```
    #"Измененный тип",
    {"Рабочий", "Выходной"},
    "Рабочий/выходной",
    "Продажи"
  )
in
  #"Сведенный столбец"
```

Результат запроса представлен на рис. 29. Обратите внимание, что на столбец *Штуки* операция сведения не повлияла.



= Table.Pivot(

	Дата	Штуки	Рабочий	Выходной
1	01.04.2022 0:00:00	2	5	null
2	02.04.2022 0:00:00	1	null	4
3	03.04.2022 0:00:00	1	null	2
4	04.04.2022 0:00:00	1	6	null
5	05.04.2022 0:00:00	5	8	null
6	06.04.2022 0:00:00	3	6	null
7	07.04.2022 0:00:00	2	4	null
8	08.04.2022 0:00:00	1	5	null
9	09.04.2022 0:00:00	2	null	5
10	10.04.2022 0:00:00	3	null	2
11	11.04.2022 0:00:00	1	1	null
12	12.04.2022 0:00:00	1	3	null

Рис. 29. Таблица после сведения по столбцу *Продажи*

Вместо жесткого кодирования списка во втором аргументе функции Table.Pivot() можно использовать функцию List.Distinct(), которая возвращает уникальные значения из списка:

Листинг 22

```
let
  Источник = Excel.CurrentWorkbook(){[Name="Таблица5"]}[Content],
  #"Измененный тип" = Table.TransformColumnTypes(
    Источник,
    {
      {"Дата", type datetime},
      {"Рабочий/выходной", type text},
      {"Продажи", Int64.Type},
      {"Штуки", Int64.Type}
    }
  ),
  #"Сведенный столбец" = Table.Pivot(
    #"Измененный тип",
    List.Distinct(#"Измененный тип"[#"Рабочий/выходной"]),
    "Рабочий/выходной",
    "Продажи"
  )
in
  #"Сведенный столбец"
```

Такой подход является более гибким, а запросы, написанные в таком стиле, более робастными, устойчивыми к возможным последующим изменениям исходных данных.

Пятый параметр функции Table.Pivot(), который в данном случае опущен, это функция для агрегирования данных при операции сведения. Опущенный параметр означает, что сведение столбца выполнено без агрегирования. Если бы в таблице на рис. 23 было несколько строк с одной датой, то агрегирование понадобилось (если мы хотим, чтобы в итоговой таблице была

только одна строка для каждой даты). Для суммирования значений можно использовать функцию List.Sum()…

= Table.Pivot(#"Измененный тип", List.Distinct(#"Измененный тип"[#"Рабочий/выходной"]), "Рабочий/выходной", "Продажи", List.Sum)

… для нахождения максимума – List.Max()…

= Table.Pivot(#"Измененный тип", List.Distinct(#"Измененный тип"[#"Рабочий/выходной"]), "Рабочий/выходной", "Продажи", List.Max)

… и др. стандартные функции. Более того, можно использовать пользовательские функции.

Таблицы и ключи

Хотя этот факт скрыт, таблицы в Power Query могут иметь первичный и внешний ключи. Таблицы, импортированные из источников данных, таких как реляционные базы данных, имеют ключи, определенные для них автоматически. Вы также можете определить свои собственные ключевые столбцы в любой создаваемой таблице. Пользовательский интерфейс не отображает, какие столбцы в таблице являются ключевыми, но можно использовать функцию Table.Keys() для возврата списка ключевых столбцов в таблице.

Ключи можно добавить в таблицу с помощью функции Table.AddKey(). Вернемся к таблице на рис. 23. Определим первичный ключ для столбца *Дата*:

Листинг 23

```
let
   Источник = Excel.CurrentWorkbook(){[Name="Таблица5"]}[Content],
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Дата", type datetime},
         {"Рабочий/выходной", type text},
         {"Продажи", Int64.Type},
         {"Штуки", Int64.Type}
      }
   ),
   DefineDateKey = Table.AddKey(#"Измененный тип", {"Дата"}, true)
in
   DefineDateKey
```

Обратите внимание, что функция Table.AddKey() не проверяет, являются ли значения в столбце *Дата* уникальными, поэтому вы должны убедиться в этом другим способом.

Выполнение некоторых операций с таблицей в качестве побочного эффекта создает первичный ключ. Например, выбор столбца в интерфейсе редактора запросов и нажатие кнопки *Удалить дубликаты*, помимо прочего, делает этот столбец ключевым. Я убедился в этом запросив ключи сразу после импорта таблицы с рис. 23 и после удаления дубликатов в столбце *Продажи*:

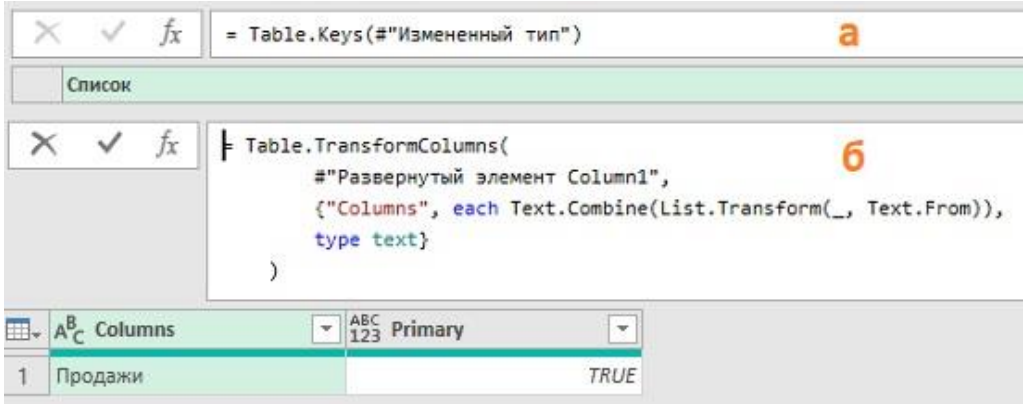


Рис. 30. Ключевые столбцы: а) после импорта таблицы, б) после удаления дубликатов в столбце *Продажи* (см. запрос с именем *Таблица24* в приложенном Excel-файле)

Если ключи не создаются автоматически, то преимущества их определения в рамках запроса мало очевидны. Тем не менее, ключи будут влиять синтаксис ссылок на отдельные строки в таблице (см. ниже первый абзац после рис. 36).

Ссылки на отдельные значения

На отдельные значения в таблицах, записях или списках можно ссылаться с помощью операторов выбора и проекции.

Ссылки на элементы в списках

Учитывая, что список по своей сути упорядочен (индексирован), можно ссылаться на элемент в списке, используя индекс от нуля. Например, если взять список {"A", "B", "C"}, то выражение {"A", "B", "C"}{0} возвращает "A", а выражение {"A", "B", "C"}{2} возвращает "C". Если вы используете индекс, который не существует в списке, {"A", "B", "C"} {4}, по умолчанию будет возвращена ошибка; но если вы добавите ? оператор в конце выражения, {"A", "B", "C"}{4}? , вместо ошибки будет возвращено нулевое значение.

Ссылки на строки в таблицах

На строки в таблицах можно ссылаться по индексу так же, как и на элементы в списке, хотя вместо отдельного значения возвращается запись, представляющая собой строку. Ниже показана таблица, которую мы используем для иллюстрации синтаксиса ссылки на строку.

	A	B	C
1	Фрукты	Продажи	Штуки
2	Яблоки	10	7
3	Апельсины	20	8
4	Персики	30	9

Рис. 31. Простая таблица

Импортируйте таблицу в PQ. Теперь, чтобы вернуть первую строку, воспользуйтесь кодом:

Листинг 25

```
let
   Источник = Excel.CurrentWorkbook(){[Name="Таблица6"]}[Content],
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Фрукты", type text},
         {"Продажи", Int64.Type},
         {"Штуки", Int64.Type}
      }
   ),
   FirstRow = #"Измененный тип"{0}
in
   FirstRow
```

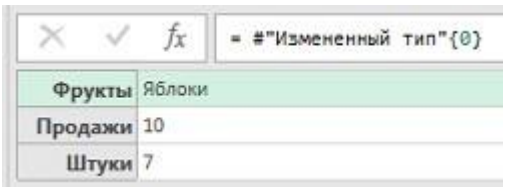


Рис. 32. Запись, представляющая собой первую строку таблицы

На ту же строку в таблице можно ссылаться используя выражение…

```
   FirstRow = #"Измененный тип"{[Фрукты="Яблоки"]}
```

При этом в таблице выполняется поиск строки, в которой столбец *Фрукты* содержит значение *Яблоки*. Если строка не найдена или найдено несколько строк, возвращается сообщение об ошибке. Например, выражение…

```
   FirstRow = #"Измененный тип"{[Фрукты="Виноград"]}
```

… возвращает ошибку:

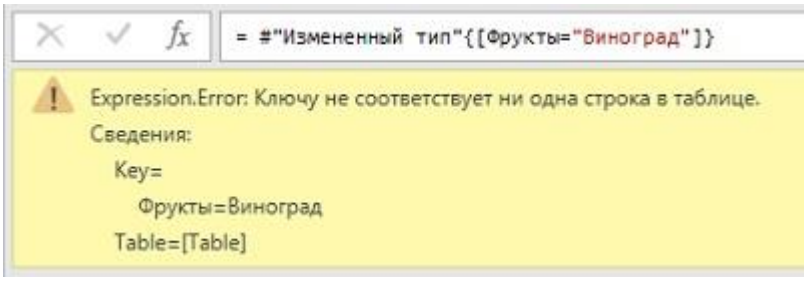


Рис. 33. Ошибка, при попытке извлечь строку с несуществующим ключом

При этом добавление ? оператора в конце выражения…

```
FirstRow = #"Измененный тип"{[Фрукты="Виноград"]}?
```

…возвращает значение null вместо ошибки.

Если искомый столбец является первичным ключом таблицы, то повторяющихся значений в нем нет, и, следовательно, нет риска возврата ошибки из-за дублей.

Ссылки на поля в записях

На значения полей в записи можно ссылаться с помощью имени поля. Дополним листинг 25, чтобы вернуть значение поля *Фрукты* из первой строки таблицы:

Листинг 26

```
let
   Источник = Excel.CurrentWorkbook(){[Name="Таблица6"]}[Content],
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Фрукты", type text},
         {"Продажи", Int64.Type},
         {"Штуки", Int64.Type}
      }
   ),
   FirstRow = #"Измененный тип"{0},
   FirstColumn = FirstRow[Фрукты]
in
   FirstColumn
```

Можно вернуть запись, содержащую часть полей более крупной записи. Для этого в строке запроса нужно перечислить имена возвращаемых столбцов через запятую, не забыв взять перечень имен в квадратные скобки:

Листинг 27

```
let
   Источник = Excel.CurrentWorkbook(){[Name="Таблица6"]}[Content],
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Фрукты", type text},
         {"Продажи", Int64.Type},
         {"Штуки", Int64.Type}
      }
   ),
   FirstRow = #"Измененный тип"{0},
   SeveralColumns = FirstRow[[Фрукты],[Продажи]]
in
   SeveralColumns
```

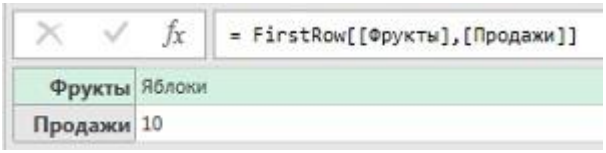


Рис. 34. Запись со значения из первых двух столбцов строки

Ссылки на значения в таблицах

Для ссылки на отдельное значение в таблице нужно указать номер строки и название столбца. При этом порядок, в котором они указаны не влияет на результат:

Листинг 28

```
let
   Источник = Excel.CurrentWorkbook(){[Name="Таблица6"]}[Content],
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Фрукты", type text},
         {"Продажи", Int64.Type},
         {"Штуки", Int64.Type}
      }
   ),
   OneValue = #"Измененный тип"{0}[Фрукты]
in
   OneValue
```

Общий шаблон для извлечения значения из таблицы…

TableName{RowIndex|KeyMatch}[ColumnName]

Обратите внимание, что этому шаблону также подчиняется первая строка в листинге 28. Шаг…

```
   Источник = Excel.CurrentWorkbook(){[Name="Таблица6"]}[Content],
```

…возвращает значение из столбца [Content] строки *Таблица6* таблицы, содержащей все таблицы текущего файла Excel:

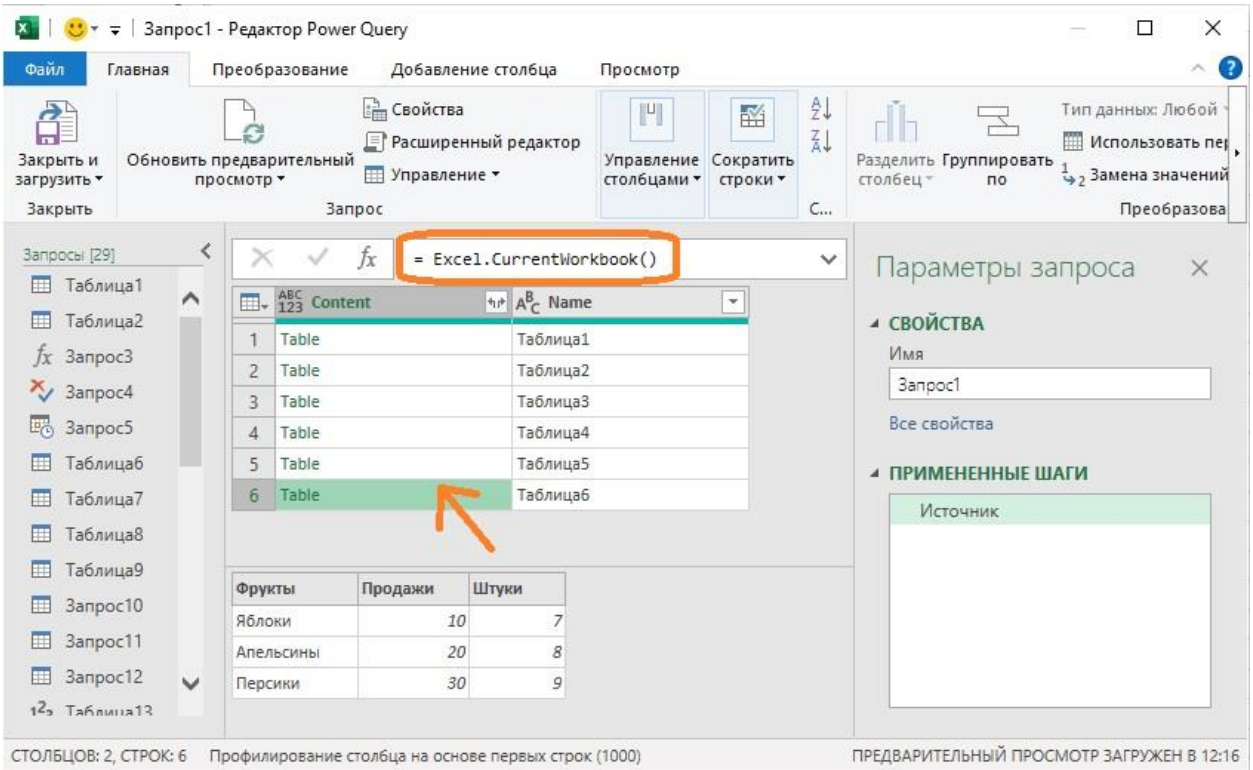


Рис. 35. Таблица в редакторе PQ всех умных таблиц текущего файла Excel

В пользовательском интерфейсе редактора PQ для выбора отдельного значения в таблице щелкните правой кнопкой мыши на ячейке, из которой хотите извлечь значение, и выдерите *Детализация*:

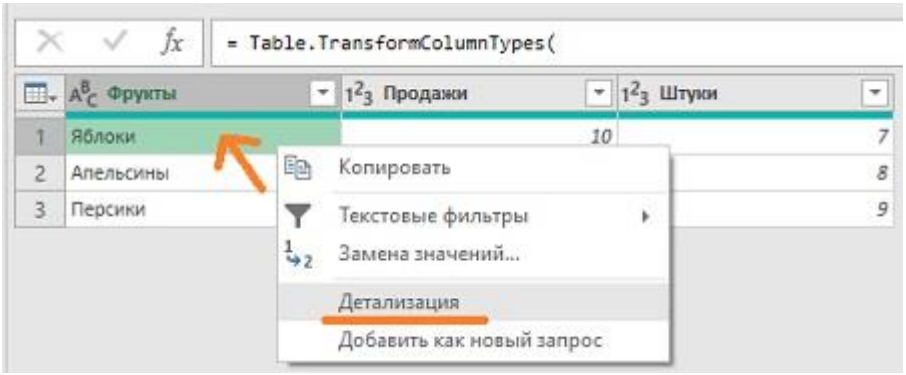


Рис. 36. Пункт *Детализации* в контекстном меню, доступном при щелчке правой кнопкой мыши на ячейке таблицы

Важно понимать, что ссылки, созданные с помощью этого метода, будут различаться в зависимости от того, определен ли первичный ключ для этой таблицы. Если первичный ключ не определен, то ссылка будет основана на индексе строки, например *Источник{0}[Фрукты]*. Если же первичный ключ определен, ссылка будет использовать значение из столбцов первичного ключа вместо индекса строки, например *Источник{[Фрукты="Яблоки"]}[Фрукты]*. Первое выражение возвращает значение из столбца *Фрукты* для первой строки таблицы, тогда как второе – значение из столбца *Фрукты* той строки, где столбец *Фрукты* содержит значение *Яблоки* (сравните запросы с именами *Таблица28* и *Таблица30* в приложенном Excel-файле).

Функции

Помимо встроенных функций M позволяет определять и использовать собственные функции. Эти функции могут быть определены внутри запроса или как отдельные запросы. В последнем случае они могут использоваться на различных шагах запроса или даже в нескольких запросах.

Определение функций внутри запроса

Функция может быть определена как шаг в запросе:

Листинг 31

```
let
   //Определение функции, которая умножает аргументы и прибавляет к произведению 1
   ExampleFunction = (x,y) => x * y + 1,
   //Вызов функции с аргументами 3 и 4
   QueryOutput = ExampleFunction(3,4)
in
   QueryOutput
```

Здесь первый шаг определяет функцию с именем *ExampleFunction*, которая принимает два параметра, x и y, и возвращает значение x * y + 1. На шаге *QueryOutput* происходит вызов функции *ExampleFunction* с параметрами 3 и 4. Функция *ExampleFunction* возвращает значение 3 * 4 + 1 = 13.

Можно задать тип аргументов и / или тип значения, возвращаемое функцией. Параметры можно пометить как необязательные, и в этом случае при вызове функции их можно опустить. Если параметры не передаются, то в теле функции им дается значение null. Необязательные параметры должны быть перечислены после всех обязательных параметров в списке аргументов функции.

Ниже представлен код, объявляющий функцию с тремя параметрами, для которых указан тип. Причем z является необязательным. Если z передается, то функция возвращает значение x * y + z; если z не передается – значение x * y + 1.

Листинг 32

```
let
   //Определение функции
   ExampleFunction = (x as number, y as number, optional z as number)
```

```
  as number =>
  x * y + (if z=null then 1 else z),
  //Вызов функции с параметрами 3 и 4
  QueryOutput = ExampleFunction(3,4)
in
  QueryOutput
```

Оператор each

each может быть использован для определения безымянной функции с одним параметром. *each* является сокращением для функции, которая принимает один параметр без объявления типа с именем _

Листинг 33

```
let
  //Определение функции
  ExampleFunction = each _*2,
  //Вызов функции с параметром 3
  QueryOutput = ExampleFunction(3)
in
  QueryOutput
```

Выражение *each _ *2* эквивалентно выражению (_) => _*2. На выходе получаем 6.

Однако чаще *each* используется в качестве аргумента другой функции. Многие функции принимают функции в качестве параметров, например, Table.TransformColumns()

```
Table.TransformColumns(
  table as table,
  transformOperations as list,
  optional defaultTransformation as nullable function,
  optional missingField as nullable number
) as table
```

Второй параметр – *transformOperations* – отвечает за преобразование таблицы *table*, указанной в первом параметре. При этом параметр *transformOperations* – это список { имя столбца, преобразование }. Ниже представлен пример того, как *each* может быть использован для умножения на два всех значений в столбце *Продажи* таблицы с рис. 31.

Листинг 34

```
let
  Источник = Excel.CurrentWorkbook(){[Name="Таблица6"]}[Content],
  #"Измененный тип" = Table.TransformColumnTypes(
    Источник,
    {
      {"Фрукты", type text},
      {"Продажи", Int64.Type},
      {"Штуки", Int64.Type}
    }
  ),
  //Умножим все значения в столбце Продажи на 2
  SalesTimesTwo = Table.TransformColumns(
    #"Измененный тип",
    {"Продажи", each _ * 2}
  )
in
  SalesTimesTwo
```

Здесь первый параметр функции Table.TransformColumns() – исходная таблица, полученная на шаге #"Измененный тип". Второй параметр – список, где первым элементом является имя столбца, значения которого следует изменить, а вторым – функция, применяемая к каждому

значению в столбце. Функция Table.TransformColumns() передает каждое значение в столбце *Продажи* параметру _ функции, определенной как *each* _ * 2. На выходе получим:

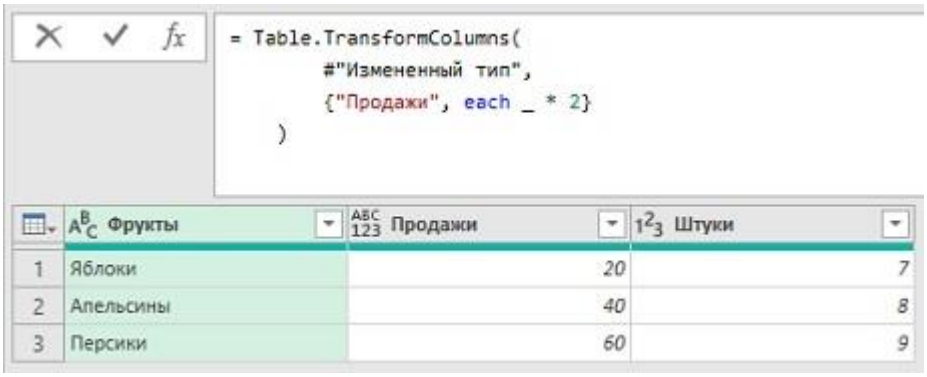


Рис. 37. Результат применения функции Table.TransformColumns (сравни с рис. 31)

Запросы как функции

Помимо определения функции как части тела запроса, функцию можно определить как отдельный запрос. Это позволит использовать функцию в любом другом запросе в книге Excel. Максимальный эффект будет получен, если предполагается неоднократное использование функции. Причина, по которой это работает, заключается в том, что функция является типом данных в Power Query, как и типы *table*, *number* или *text*. Запрос вернет значение типа *function*.

Чтобы определить запрос, возвращающий функцию, необходимо создать пустой запрос, а затем в расширенном редакторе написать код M. Следующий код аналогичен листингу 31. Но здесь мы ограничились лишь определением функции (без ее вызова).

Листинг 35

```
let
   MyFunction = (x,y) => x * y + 1
in
   MyFunction
```

После того, как вы введете этот код в расширенном редакторе и нажмёте Ok, вы увидите, что в области предварительного просмотра вам предложат ввести параметры функции, а на панели запросов в левой части поменяется значок перед именем запроса.

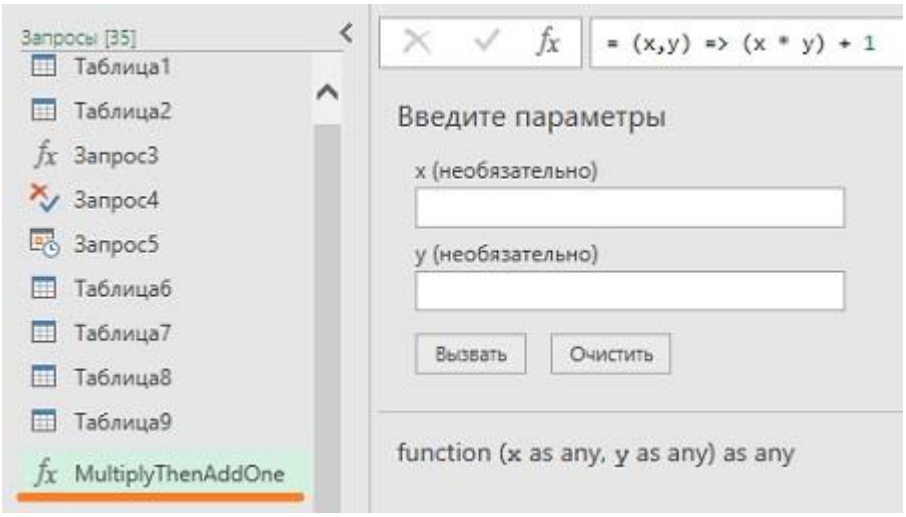


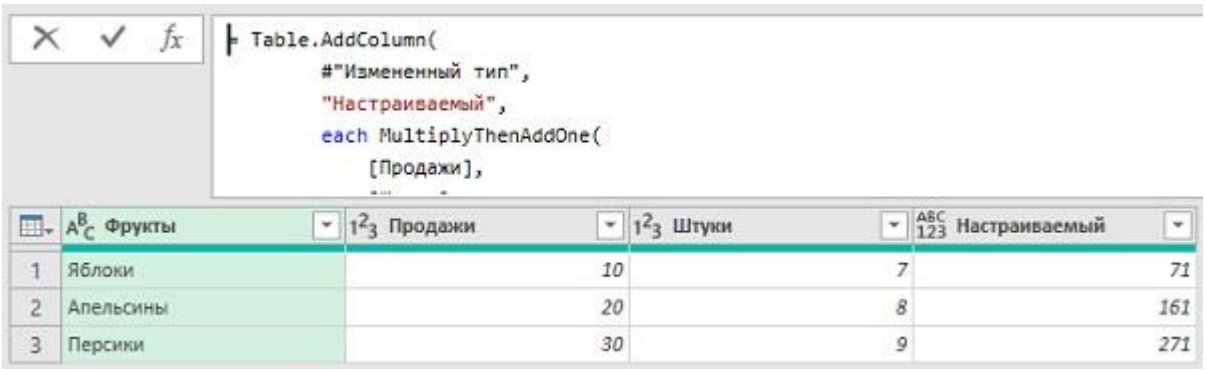
Рис. 38. Функция на панели предварительного просмотра в редакторе Power Query

Имя, которое вы дадите этому запросу, будет именем созданной функции. Я переименовал запрос в *MultiplyThenAddOne*. Если на вкладке *Главная* попытаться выбрать *Закрыть и загрузить в…*, вы увидите, что опция недоступна. Это связано с тем, что функция не возвращает данных до тех пор, пока она не будет вызвана с какими-либо параметрами.

Чтобы воспользоваться функцией, создадим новый запрос. Воспользуемся данными с рис. 31, и используем функцию *MultiplyThenAddOne*() внутри пользовательского столбца.

Листинг 36

```
let
   Источник = Excel.CurrentWorkbook(){[Name="Таблица6"]}[Content],
   #"Измененный тип" = Table.TransformColumnTypes(
      Источник,
      {
         {"Фрукты", type text},
         {"Продажи", Int64.Type},
         {"Штуки", Int64.Type}
      }
   ),
   #"Добавлен пользовательский объект" = Table.AddColumn(
      #"Измененный тип",
      "Настраиваемый",
      each MultiplyThenAddOne(
         [Продажи],
         [Штуки]
      )
   )
in
   #"Добавлен пользовательский объект"
```



	Фрукты	Продажи	Штуки	Настраиваемый
1	Яблоки	10	7	71
2	Апельсины	20	8	161
3	Персики	30	9	271

Рис. 39. Пользовательский столбец, основанный на функции *MultiplyThenAddOne*()

Функцию также можно вызвать в Excel с панели *Запросы и подключения*, щелкнув по ней правой кнопкой мыши и выбрав *Вызвать*. Появится диалоговое окно с предложением ввести параметры функции. Если ввести параметры и нажать Ok, откроется окно редактора Power Query и будет создан новый запрос, который вызовет функции с этими значениями и вернет результат:

Листинг 37

```
let
   Источник = MultiplyThenAddOne(2, 4)
in
   Источник
```

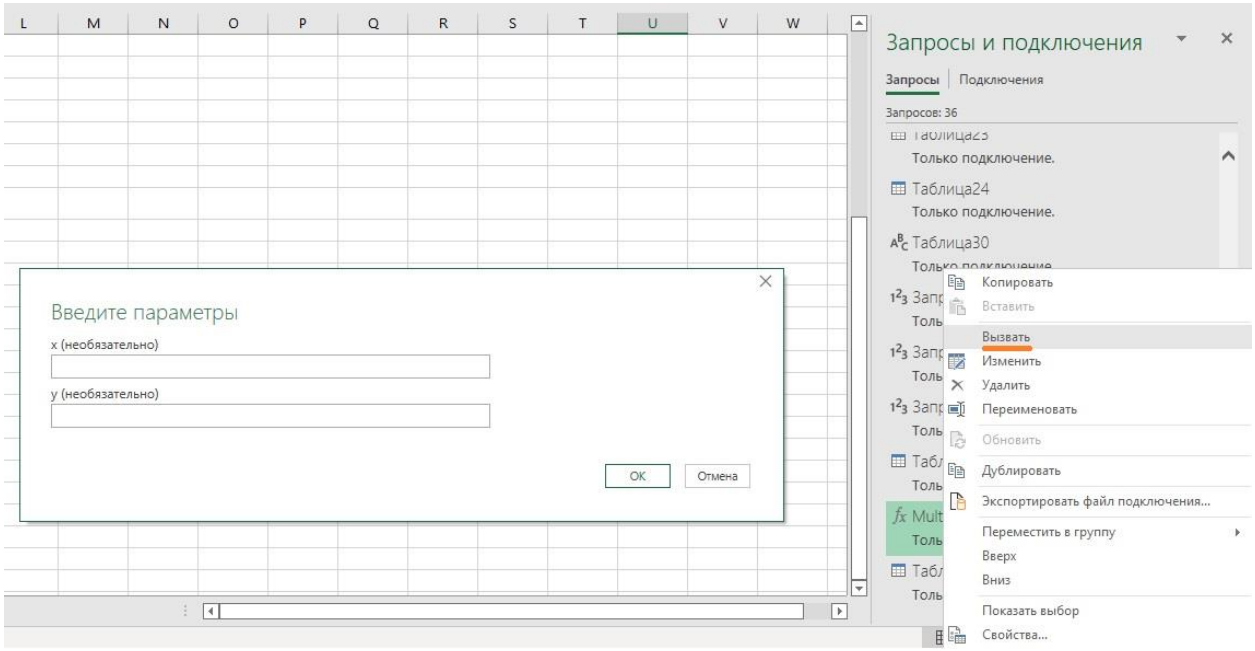


Рис. 40. Вызов функции из области *Запросы и подключения* в Excel

Оператор let в определениях функций

Для определения сложных функций потребуется более одной строки кода. В этом случае необходимо использовать оператор *let* внутри определения функции. Ниже приведен код функции из листинга 35, переписанный с использованием двух строк.

Листинг 38

```
let
   //Определение функции с использованием выражения let
   MyFunction = (x,y) =>
      let
         //Умножение x на y
         Step1 = x * y,
         //Добавление единицы к результату
         Step2 = Step1 + 1
      in
         Step2
in
   MyFunction
```

Помните, что выражение *let* – это просто способ объединить несколько шагов и вернуть значение, и что это значение может быть функцией. Если в определении функции используется выражение *let*, редактор запросов сможет отобразить только один шаг для всего определения функции, и вы не увидите весь код M для него в строке формул. Редактирование функций в строке формул часто недоступно. Но по-прежнему код функций можно редактировать в расширенном редакторе.

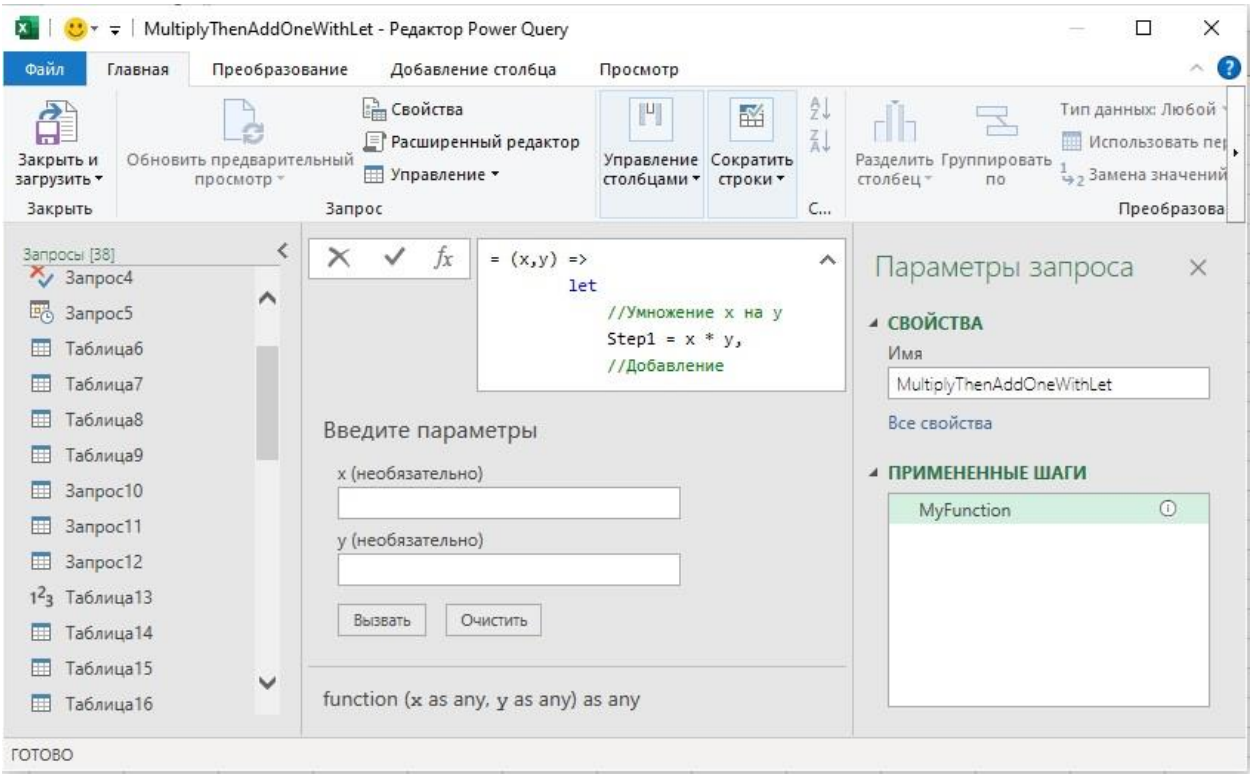


Рис. 41. Для функций редактор запросов отражает только один шаг

Рекурсивные функции

Рекурсивной называется функция, которая вызывает себя в теле своего собственного определения. Чтобы избежать попадания в бесконечный цикл, нужно задать условия выхода из цикла. Ниже приведен пример функции, которая принимает один параметр, и если он больше или равен 100, то функция возвращает значение параметра, а если он меньше 100, функция вызывается рекурсивно с исходным значением параметра, умноженным на 2.

Листинг 39

```
let
   //Определение рекурсивной функции с одним параметром
   DoubleToOneHundred = (x) =>
      //Если параметр более 100, возвращается значение параметра
      if x > 100
      then x
      //Если параметр не более 100, он умножается на два и функция вызывается повторно
      else @DoubleToOneHundred(x*2)
in
   DoubleToOneHundred
```

Для вызова рекурсивно используется оператор @ перед именем функции.

Функции, импортированные из источников данных

Некоторые типы объектов во внешних источниках данных, такие как определяемые пользователем функции в базе данных SQL Server и операции службы OData, при импорте в Power Query рассматриваются как функции. После импорта они ведут себя как любая другая функция Power Query, хотя, конечно, нет определения M, которое вы можете увидеть и отредактировать.

Работа с веб-службами

Одним из важных типов источников данных, для работы с которыми необходимо использовать M, являются веб-службы. Power Query с помощью функции Web.Contents() может вызывать веб-службы RESTful, т.е. веб-службы, к которым можно обращаться, используя только информацию, передаваемую по URL-адресу. Функции Web.Contents() принимает два параметра. Первый – URL-адрес веб-службы, второй (необязательный) принимает запись, содержащую дополнительные свойства. Напомню, что запись – это упорядоченная последовательность полей, где каждое поле имеет имя и одно значение (любого типа). Рассмотрим воображаемую веб-службу,

расположенную по адреса http://www.mywebservice.com/GetData. Для ее вызова используйте код:

```
Web.Contents("http://www.mywebservice.com/GetData")
```

Обычно веб-служба возвращает данные в виде документа XML или JSON, и Power Query автоматически распознает используемый формат и открывает его, чтобы вы могли видеть содержимое. Многие веб-службы также требуют добавления параметров в URL-адрес, что-то типа:

```
http://www.mywebservice.com/GetData?search=somedata&results=10
```

Вы можете объединить параметр с URL-адресом, как показано выше, но использование второго параметра придает запросу гибкость.

```
Web.Contents(
   "http://www.mywebservice.com/GetData",
   [Query=[#"search"="somedata", #"results"="10"] ]
)
```

Если вам нужно передать пользовательские заголовки HTTP, вы можете сделать это аналогичным образом, добавив поле *Headers*:

```
Web.Contents(
   "http://www.mywebservice.com/GetData",
   [
      Query=[#"search"="somedata", #"results"="10"],
      Headers=[#"HeaderName"="ValueToPass"]
   ]
)
```

В некоторых случаях веб-служба потребует передачи ключа или маркера через пользовательский http-заголовок. Если это сделать в коде, вы жестко запрограммируете значение ключа или маркера внутри кода запроса. Это не безопасно. Вместо этого можно использовать поле ApiKeyName для указания имени пользовательского заголовка HTTP, содержащего ключ или маркер, например:

```
Web.Contents(
   "http://www.mywebservice.com/GetData",
   [
      Query=[#"search"="somedata", #"results"="10"],
      Headers=[#"HeaderName"="ValueToPass"],
      ApiKeyName="APIToken"
   ]
)
```

При первом выполнении запроса появится диалоговое окно *Учетные данные Power Query* с предложением ввести значение ключа или маркера. После этого ключ или маркер будет сохранен в безопасном хранилище учетных данных Power Query.

Функция Web.Contents() по умолчанию создает запрос GET на чтение данных с сайта. С помощью кода M также можно создать запрос POST для передачи на сайт заполненных форм (подробнее см. Чем отличаются HTTP-методы GET и POST).

```
Web.Contents(
   "http://www.mywebservice.com/SendData",
   [Content=Text.ToBinary("Text content to send in POST request"]
)
```

В этом случае значение поля Content содержит двоичные данные, которые будут использоваться в качестве содержимого запроса POST; функция Text.ToBinary() должна использоваться для преобразования отправляемого текстового значения в значение типа *binary*.

Свертывание запросов

Что не очевидно в Power Query, так это то, где на самом деле происходит вся тяжелая работа. Когда запрос подключается к источнику данных, Power Query пытается выполнить как можно больше работы в самом источнике данных, где (или так предполагается) это можно сделать более эффективно. Т.е., PQ не будет грузить данные в свое ядро, и выполнять преобразования по шагам, а попытается сгруппировать шаги запроса, и отправить этот пул в источник данных. Такое поведение называется свертыванием запросов.

Сможет ли Power Query сделать это, зависит от типа используемого источника данных и типа преобразований в запросе. В реляционной базе данных, такой как SQL Server, Power Query попытается преобразовать всю логику запроса в инструкцию SQL SELECT, а с помощью источника данных OData попытается преобразовать всю логику в один URL-адрес OData. Однако для источников данных, таких как текстовые файлы, Power Query не имеет другого выбора, кроме как загрузить все данные в свое ядро и выполнить преобразования внутри.

То, что Power Query может отправить обработку в источник данных, здорово повышает производительность. SQL Server агрегирует данные большой таблицы гораздо эффективнее, чем Power Query, который сначала загрузит всю таблицу в свое ядро, и лишь затем выполнит агрегирование. Раз свертывание запросов эффективно, вы захотите, чтобы этот механизм использовался в ваших запросах.

Можно ли так строить свои запросы, чтобы обеспечить их свертывание? К сожалению, простых ответов нет. На момент написания книги свертывание запроса ограничено четырьмя типами источников данных: реляционными базами данных, источниками данных OData, Exchange и Active Directory. Однако это может измениться в будущем.[3] Кроме того, существует множество других факторов, которые определяют, может ли происходить свертывание запросов, таких как определенные типы преобразования, и опять же эти факторы почти наверняка изменятся в будущем. В результате можно дать только очень общие рекомендации о том, как обеспечить свертывание запросов. Если производительность является проблемой, то следует экспериментировать, чтобы увидеть, как влияет код Power Query на скорость выполнения запросов.

Наблюдение за свертыванием запросов в SQL Server

К сожалению, в пользовательском интерфейсе Power Query нет указаний на то, происходит свертывание или нет. Для мониторинга связи между Power Query и источником данных необходимо использовать другие средства. При использовании SQL Server в качестве источника данных можно увидеть работу запроса, выполнив трассировку приложения SQL Server Profiler во время обновления запроса.

Запрос, приведенный ниже подключается к таблице DimProductCategory в базе данных Adventure Works DW в SQL Server, фильтрует строки, в которых значение в столбце *ProductCategoryKey* меньше 4, а затем подсчитывает количество оставшихся строк.

Листинг 41

```
let
  //connect to SQL Server
  Source = Sql.Database("localhost", "Adventure Works DW"),
  //connect to the DimProductCategory table
  dbo_DimProductCategory = Source{[Schema="dbo",Item="DimProductCategory"]}[Data],
  //filter the table where ProductCategoryKey is less than 4
  FilteredRows = Table.SelectRows(dbo_DimProductCategory, each [ProductCategoryKey] < 4),
  //count the number of rows in the resulting table
  GroupedRows = Table.Group(
    FilteredRows,
    {},
    {{"Count", each Table.RowCount(_), type number}}
```

[3] Это не так. По состоянию на май 2022 г. Microsoft поддерживает свёртывание запросов всё в тех же 4 типах источников данных.

```
  )
in
  GroupedRows
```

Если открыть приложение SQL Server Profiler и запустить трассировку, выполняемую в базе данных Adventure Works DW с помощью шаблона Standard при обновлении запроса Power Query, вы увидите SQL, показанный в листинге 42, в событии SQL:BatchCompleted, как показано на рис. 42. Этот SQL-запрос содержит всю логику запроса Power Query и возвращает таблицу, содержащую одно значение — ту же таблицу, которую возвращает сам запрос Power Query . В этом случае явно имело место свёртывание запроса.

Листинг 42

```
select count(1) as [Count]
from
(select [_].[ProductCategoryKey],
      [_].[ProductCategoryAlternateKey],
      [_].[EnglishProductCategoryName],
      [_].[SpanishProductCategoryName],
      [_].[FrenchProductCategoryName]
  from [dbo].[DimProductCategory] as [_]
  where [_].[ProductCategoryKey] < 4
) as [rows]
```

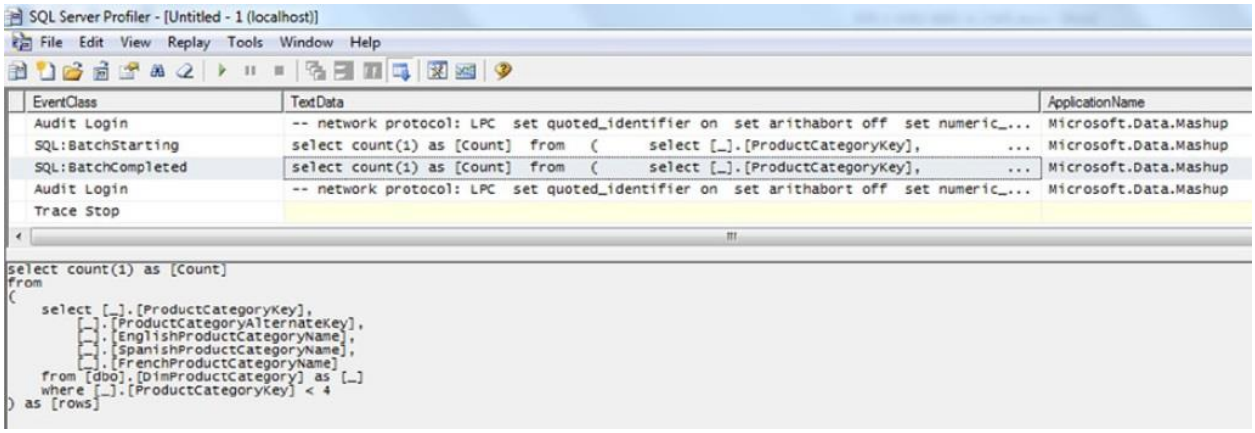


Рис. 42. Трассировка приложения SQL Server Profiler, показывающая свертывание запроса в действии

Предотвращение свертывания запросов в коде

Есть две функции, которые можно использовать, если вы хотите предотвратить свертывание запроса: List.Buffer() и Table.Buffer(). Обе функции работают одинаково, принимая список или таблицу и загружая все их данные в Power Query, а затем возвращая список или таблицу без изменений.

Ниже показано, как можно изменить запрос, приведенный в листинге 41, чтобы отключить свертывание. Трассировка при обновлении этого запроса отобразит в инструкции SQL SELECT только извлечение всех данных из таблицы DimProductCategory. В предложении Where нет фильтрации и нет Group By.

Листинг 43

```
let
  //connect to SQL Server
  Source = Sql.Database("localhost", "Adventure Works DW"),
  //connect to the DimProductCategory table
  dbo_DimProductCategory = Source{[Schema="dbo",Item="DimProductCategory"]}[Data],
  //buffer table
  BufferedTable = Table.Buffer(dbo_DimProductCategory),
  //filter the table where ProductCategoryKey is less than 4
  FilteredRows = Table.SelectRows(BufferedTable, each [ProductCategoryKey] < 4),
```

```
  //count the number of rows in the resulting table
  GroupedRows = Table.Group(FilteredRows, {}, {{"Count", each Table.RowCount(_), type number}})
in
  GroupedRows
```

Другие операции, которые могут препятствовать свертыванию запросов

Ниже приведен ряд сценариев, в которых свертывание запросов запрещено в версии Power Query, используемой для написания этой книги. Не думайте, что это будет применимо к последующим версиям Power Query. Следующий список будет полезен как показатель того, на какие операции следует обращать внимание.

Пользовательские инструкции SQL

Пользовательская инструкция SQL с любым источником данных реляционной базы данных (т.е. сценарий, в котором в качестве отправной точки запроса вводится собственный SQL-запрос вместо выбора таблицы на панели *Навигатор* и последующей фильтрации ее в редакторе запросов) автоматически предотвращает свертывание запроса. Если вам нужен пользовательский SQL и вы беспокоитесь о производительности, вы можете, либо создать представление с помощью пользовательского SQL и указать Power Query на представление; либо попытайтесь выполнить как можно меньше работы в Power Query и как можно больше работы в самом SQL-запросе.

Удаление строк с ошибками

Команда редактора PQ *Удалить строки –> Удалить ошибки* или функция Table.RemoveRowsWithErrors() предотвращает свертывание запроса на том шаге, где это используется. Если нужно удалить строки из таблицы, содержащие значения ошибок, это лучше сделать ближе к концу сценария. Пример ниже содержит шаг, который возвращает ошибку. Она фильтруется с помощью Table.RemoveRowsWithErrors() на следующем шаге. Поэтому шаг *FilteredRows* не может быть свернут вместе с предыдущими и передан в SQL, как единый запрос.

Листинг 44

```
let
  //connect to SQL Server
  Source = Sql.Database("localhost", "Adventure Works DW"),
  //connect to the DimProductCategory table
  dbo_DimProductCategory = Source{[Schema="dbo",Item="DimProductCategory"]}[Data],
  //insert custom column that returns an error in one row
  InsertedCustom = Table.AddColumn(
    dbo_DimProductCategory,
    "DivideByZero",
    each if [ProductCategoryKey]=1
      then xyz
      else null
  ),
  //remove error row
  RemovedErrors = Table.RemoveRowsWithErrors(InsertedCustom, {"DivideByZero"}),
  //filter table by ProductCategoryKey < 4
  FilteredRows = Table.SelectRows(RemovedErrors, each [ProductCategoryKey] < 4)
in
  FilteredRows
```

Измените порядок двух последних шагов: сначала *FilteredRows*, а затем *RemovedErrors.* Это включит в свертывание запроса шаг *FilteredRows*.

Комплексные операции

Многие другие функции M также могут предотвращать свертывание. Например, Table.Pivot() и Table.UnPivot(). Если вы определяете и используете функцию внутри запроса (даже если функция тривиальна), это также предотвращает свертывание. Старайтесь на первых шагах применять простые фильтры и агрегаторы, т.е. то, что можно выполнены лишь с помощью пользовательского интерфейса. Оставьте более сложные операции на потом.