

Глава 14. Проблемные паттерны данных

Это продолжение перевода книги Грегори Декер, Рик де Гроот, Мелисса де Кортс. Полное руководство по языку M Power Query. Изменение и подготовка данных к анализу часто представляют собой комбинацию искусства и науки, сочетая теорию и навыки с творческим решением задач. Существует широкий спектр проблемных шаблонов данных, от работы с данными в стеке до многострочных заголовков. Проблемы не ограничиваются только структурой данных, но часто включают несоответствия и другие сложности, которые могут возникнуть из-за уникального характера данных или конкретных бизнес-правил и требований.

Мои комментарии набраны с отступом.

Предыдущая глава [Содержание](#) Следующая глава

Когда мы думаем о решении проблем, важно помнить, что существует множество различных подходов и техник. В этой книге мы не утверждаем, что у нас есть ответы на все вопросы или что наши методы являются лучшими. Скорее, мы стремимся предложить коллекцию идей и разжечь воображение, чтобы вы начали свой путь к решению проблем, и были готовы к столкновению с любыми неприятными шаблонами данных, которые встретятся на вашем пути.

Примеры, приведенные в этой главе, являются отправной точкой. По мере того, как вы ее читаете, мы надеемся не только объединить теорию с практическими знаниями, но и вдохновить вас рассматривать решение проблем как творческий процесс, потому что, в конечном счете, лучшее решение – это то, которое вы разрабатываете для своей уникальной задачи.

Существует бесчисленное множество сценариев, которые хорошо подойдут для этой главы. Но место в книге ограничено, поэтому мы решили подробно рассмотреть две темы:

- извлечение фиксированных шаблонов,
- извлечение и объединение данных.

Учитывая повторяющиеся дискуссии на онлайн-форумах, ясно, что эти темы отражают типичные проблемы, с которыми пользователи сталкиваются в реальном мире. Мы начнем с выявления и извлечения значимых шаблонов из данных.

Сопоставление с шаблоном

Сопоставление с шаблоном (*Pattern matching*) – важный навык, часто связанный с регулярными выражениями (*regex*), который необходим для идентификации и манипуляции определенными шаблонами в текстовых данных. Хотя язык M не поддерживает регулярные выражения напрямую, он предоставляет множество функций, которые помогают справиться с такими задачами. В этом разделе мы рассмотрим сопоставление с шаблоном в рамках ограничений Power Query, выделяя ключевые техники и стратегии для решения этих задач.

Сопоставление с шаблоном невероятно полезно для поиска и извлечения последовательностей символов из данных. Этот процесс использует текстовые функции языка M. Мы рассмотрим некоторые из них.

Чувствительность к регистру

Язык M чувствителен к регистру, т.е. различает прописные и строчные буквы как отдельные символы. Но есть текстовые функции, которые могут выполнять операции без учета регистра, т.е. при сравнении двух строк не будет учитываться регистр букв. Эти функции оснащены дополнительным аргументом (функцией) сравнения.

В ситуациях, когда в функции отсутствует средство сравнения, хорошей стратегией может быть изменение регистра текста для достижения единообразия перед дальнейшей обработкой:

```
Text.Lower("Power Query") // вернет power query  
Text.Upper("Power Query") // вернет POWER QUERY  
Text.Proper("power query") // вернет Power Query
```

Содержит против точного соответствия

При поиске "Power Query" в строке "Power Query is awesome!" вторая строка содержит первую, но точного совпадения нет. Требования к сопоставлению с шаблоном могут различаться – иногда вам нужно подтвердить наличие подстроки, как в случае с типом "contains", в то время как в других

случаях необходимо точное совпадение. Функции *Text.Start*, *Text.End*, *Text.Middle* и *Text.Range*, могут возвращать подстроку для сопоставления.

Text.Contains предлагает простой метод для проверки наличия подстроки в строке. *Text.PositionOf* можно использовать для нахождения позиции подстроки в строке. Если функция возвращает число большее или равное 0, подстрока существует. Если возвращает -1, подстрока отсутствует.

Допустимые символы

Важным шагом является определение, какие символы имеют значение для вашего шаблона. Это могут быть прописные или строчные буквы, цифры, специальных символов или их комбинация. В языке M есть функции, которые могут сохранять или удалять символы:

```
Text.Select("Power Query", {"o", "e"}) // вернет oee  
Text.Remove("Power Query", {"o", "e"}) // вернет Pwr Qury
```

Работа с одним или несколькими элементами

Вы работаете с одним словом или с несколькими: строкой/предложением? Если с несколькими, нужно ли их разбить на более мелкие единицы? При работе с несколькими элементами вы будете прибегать к операциям со списками: *List.Select*, *List.Transform* и др. Для возврата одного значения часто используется *Text.Combine*. Эту функция принимает список текстовых значений и объединяет их в одну строку. При необходимости, добавляя разделитель между элементами.

Подстановочные знаки

Подстановочные знаки – символы, используемые для представления одного или нескольких символов в текстовой строке. Они часто используются в операциях поиска и сопоставления с шаблоном. Хотя Power Query изначально не поддерживает подстановочные знаки, это не означает, что подстановочные знаки нельзя имитировать в M. Стандартные библиотечные функции *Text.StartsWith* и *Text.EndsWith* упрощают поиск строк, которые начинаются или заканчиваются определенным текстом. Иное использование подстановочных знаков зачастую более сложно и потребует разработки особой логики для их воспроизведения.

Извлечение фиксированных шаблонов

Извлечение фиксированных шаблонов из текстовой строки – задача, которую многие пользователи считают сложной. Она может включать в себя извлечение из больших строк текста идентификаторов, артикулов, номеров документов или иных кодов. Сложность в том, что не бывает двух одинаковых сценариев. Мы рассмотрим несколько примеров, дав ноу-хау для успешной работы с фиксированными шаблонами, например, четыре буквы и пять цифр.

Пример 1, префикс

Задача – извлечь коды, которые имеют четкую схему: каждый из них начинается с букв DGPQ, за которыми следует пять цифр. Подразумевается, что в строке будет только один код:

```
let  
    Source = Table.FromColumns(  
        {"The code DGPQ33446 is important for this task.",  
        "Unique document identifier: DGPQ13295.",  
        "Please use the code DGPQ36006 to access the system.",  
        "For verification, enter DGPQ30881 on the website.",  
        "The transaction ID DGPQ78388 must be noted.",  
        "DGPQ10273 is the code for your appointment.",  
        "Reference code DGPQ36144 is included in the report.",  
        "To complete registration, use DGPQ90158 as your code.",  
        "Package with tracking number DGPQ52287 has been shipped."},  
        type table [String=text]  
    )  
in  
    Source
```

Подробнее о коде:

let

Выражение, начинающееся с *let* и заканчивающееся *in* позволяет создать одну или несколько переменных, каждой из которых присвоено выражение и которые разделены запятыми.

Source

Это имя шага или переменной в запросе. За ним следует знак равенства и присваивается выражение для хранения значения.

Table.FromColumns

Функция используется для создания таблицы из списка.

```
{{"String1", "String2", "String3"}}
```

Первый аргумент – это список, содержащий список значений столбцов (один список для каждого столбца). В нашем примере один список внутри списка для создания одного столбца.

```
type table [String=text]
```

Необязательному второму аргументу присвоен тип таблицы. В этом случае указывают имя столбца и тип значений.

in

Заключительной частью выражения *let* является *in*. То, что указано после него, будет возвращено как результат выражения *let*. В нашем случае возвращается значение переменной *Source*.

Выполните следующие действия, чтобы добавить столбец *Code* в таблицу. Процесс потребует нескольких функций. Обычной практикой является вложение этих функций изнутри наружу. Использование выражения записи позволяет легко просмотреть результаты всех промежуточных шагов, что отлично подходит для оптимизации и устранения неполадок в коде.

Пройдите *Добавление столбца* → *Настраиваемый столбец*. Введите *Code* в поле *Имя нового столбца*. Инициализируйте выражение записи, введя набор квадратных скобок [] в поле *Настраиваемая формула столбца*. Внутри записи можно создать последовательность пар имя – значение, разделенных запятыми.

Введи $n = \text{Text.PositionOf}([\text{String}], \text{"DGPQ"}, \text{Occurrence.First})$ внутри квадратных скобок. Здесь n – имя поля, а формула после знака равно – значение поля. Чтобы точно определить положение префикса *DGPQ*, мы использовали функцию *Text.PositionOf*. Если она вернет число, больше или равное 0, подстрока существует. При этом возвращается позиционный индекс первого символа в строке. Если подстрока не найдена, функция вернет -1.

Добавьте запятую и введите вторую пару имя значения в квадратных скобках $\text{result} = \text{Text.Range}([\text{String}], n, 9)$. Функция *Text.Range* вернет 9 символов, начиная с позиции n из строки текста *String*.

Чтобы вернуть значение *result*, примените доступ к полю, поместив курсор после закрывающей квадратной скобки записи, и в новом наборе квадратных скобок введите имя поля, из которого нужно получить значение *[result]*. Вот что должно получиться:

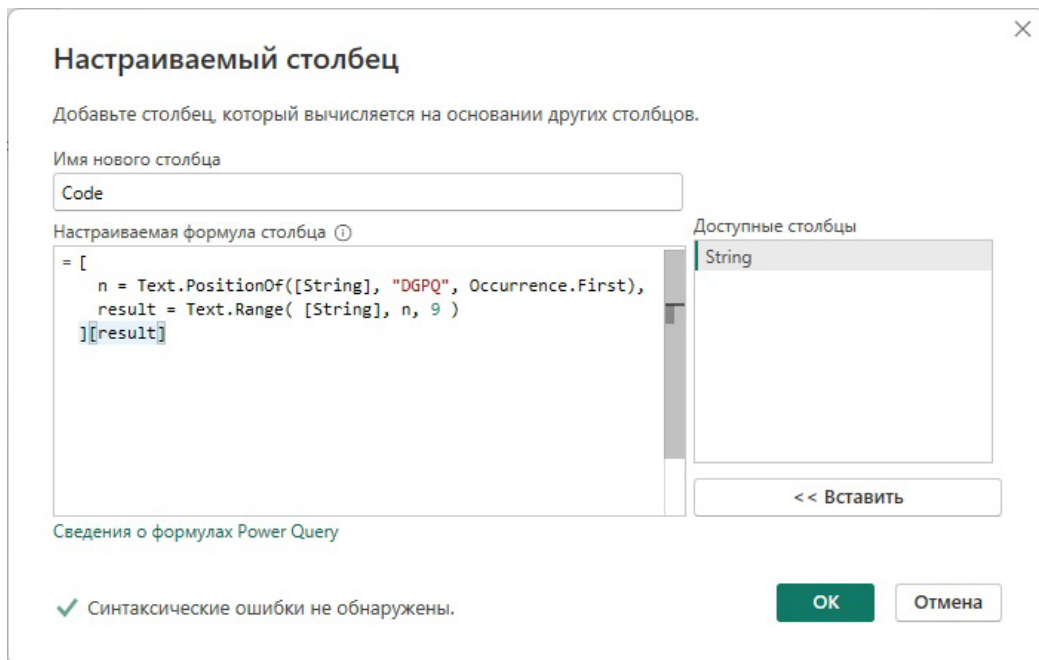


Рис. 14.0. Настройка столбца *Code*

Нажмите *OK*.

В строке формул мы увидим:

```
= Table.AddColumn(Source, "Code", each [
    n = Text.PositionOf([String], "DGPQ", Occurrence.First),
    result = Text.Range( [String], n, 9 )
][result])
```

Или, если вы предпочитаете вложенные выражения, то можно записать...

```
Table.AddColumn(Source, "Code", each
    Text.Range( [String],
        Text.PositionOf([String], "DGPQ", Occurrence.First
    ), 9
    )
)
```

	String	Code
1	The code DGPQ33446 is important for this task.	DGPQ33446
2	Unique document identifier: DGPQ13295.	DGPQ13295
3	Please use the code DGPQ36006 to access the system.	DGPQ36006
4	For verification, enter DGPQ30881 on the website.	DGPQ30881
5	The transaction ID DGPQ78388 must be noted.	DGPQ78388
6	DGPQ10273 is the code for your appointment.	DGPQ10273
7	Reference code DGPQ36144 is included in the report.	DGPQ36144
8	To complete registration, use DGPQ90158 as your code.	DGPQ90158
9	Package with tracking number DGPQ52287 has been shipped.	DGPQ52287

Рис. 14.1. Результат примера 1

Отлично! Хорошей практикой является пересмотр своего подхода и выявление ограничений и областей для улучшения. Предположим, что шаблон устойчив и не подвержен изменениям. Тогда основная проблема – отсутствие обработки ошибок. Обработка ошибок исправит ситуацию, когда подстрока *DGPQ* не найдена в тексте. В таких случаях функция *Text.PositionOf* возвращает -1. Если это не учитывать, возникнет ошибка: *Expression.Error: Аргумент "offset" выходит за пределы диапазона.* Добавим условный оператор...

```
= Table.AddColumn(Source, "Code", each [
```

```
n = Text.PositionOf([String], "DGPQ", Occurrence.First),
result = if n > -1 then Text.Range( [String], n, 9 ) else ""
][result])
```

... или (в учебных целях) немного перепишем код:

```
Table.AddColumn(Source, "Code", each
  let n = Text.PositionOf([String], "DGPQ", Occurrence.First) in
  if n > -1 then Text.Range( [String], n, 9 ) else ""
)
```

```
let n = Text.PositionOf([String], "DGPQ", Occurrence.First) in
```

Вложение выражения *let* позволяет присвоить результат выражения переменной. Мы создали переменную с именем *n*, которая содержит результат функции *Text.PositionOf*. Теперь мы можем ссылаться на результат этого выражения (*n*) несколько раз, без необходимости повторять формулу в коде.

```
if n > -1 then Text.Range( [String], n, 9 ) else ""
```

Выражение *let* всегда завершается предложением *in*. В *in* вы можете вернуть переменную или написать другое выражение. Здесь мы выбрали условный оператор, который выполняет логическую проверку. Если проверка завершилась значением *true* выполняется результирующее выражение. Если проверка вернула значение *false*, выполняется предложение *else*, и возвращается пустая текстовая строка. Эта проверка позволяет проверить, есть ли подстрока в строке перед выполнением функции *Text.Range*. Проверка предотвратит ошибку и обеспечит надлежащую работу кода в сценариях, где подстрока может отсутствовать во входной строке.

В этом примере мы показали решение с помощью вложенного *let* и выражения записи. Плюсы и минусы этих методов мы рассмотрели в [главе 8 Работа с вложенными структурами](#). Каждый разрабатывает свой стиль кодирования. Когда дело касается внедрения, устранения неполадок и ясности кода, мы рекомендуем использовать запись.

Пример 2, шаблон

В предыдущем сценарии все коды имели фиксированный буквенный префикс. Усложним шаблон. Теперь он начинается с четырех заглавных букв, за которыми следует последовательность из пяти цифр. Мы разобьем строку на последовательность символов, и извлечем все цепочки, соответствующие шаблону. Следующий набора данных содержит 6 *правильных* цепочек:

```
let
  Source = Table.FromColumns(
    {"The code CHLO33446 is important for this task.",
    "Unique document identifier: JXKE13295.",
    "Please use the code SBT36006 to access the system.",
    "For verification, enter ERKZ30881 on the website.",
    "The transaction ID YNZ78388 must be noted.",
    "IHCY10273 is the code for your appointment.",
    "Reference code SSK36144 is included in the report.",
    "To complete registration, use PRL90158 as your code.",
    "Package with tracking number MGA52287 has been shipped."},
    type table [String=text]
  )
in
  Source
```

	A ^B C String
1	The code CHLO33446 is important for this task.
2	Unique document identifier: JXKE13295.
3	Please use the code SBT36006 to access the system.
4	For verification, enter ERKZ30881 on the website.
5	The transaction ID YNZ78388 must be noted.
6	IHCY10273 is the code for your appointment.
7	Reference code SSK36144 is included in the report.
8	To complete registration, use PRLL90158 as your code.
9	Package with tracking number MGAk52287 has been shipped.

Рис. 14.1а. Исходный набор данных для примера 2

Добавим столбец *Code*. Процесс потребует множества функций, и мы используем выражение записи. Скопируйте приведенный выше код, откройте редактор Power Query, пройдите *Главная* → *Создать источник* → *Пустой запрос*. Откройте расширенный редактор, удалите заготовку и вставьте текст из буфера обмена. Пройдите *Добавление столбца* → *Настраиваемый столбец*. Введите *Code* в поле *Имя нового столбца*. Введите набор квадратных скобок [] в поле *Настраиваемая формула столбца*. Введите пары имя–значения, разделяя их запятыми:

```
w = Text.Select([String], {"A".."Z", "0".."9", " "})
```

Допустимые символы – это прописные буквы и цифры. Давайте включим пробел. Этот пробел можно использовать для разделения каждой строки на более мелкие единицы или слова. С помощью функции *Text.Select* мы отберем только эти символы из входной строки. Добавьте запятую в конце, чтобы создать новое поле в этой записи.

```
s = Text.Split( w, " ")
```

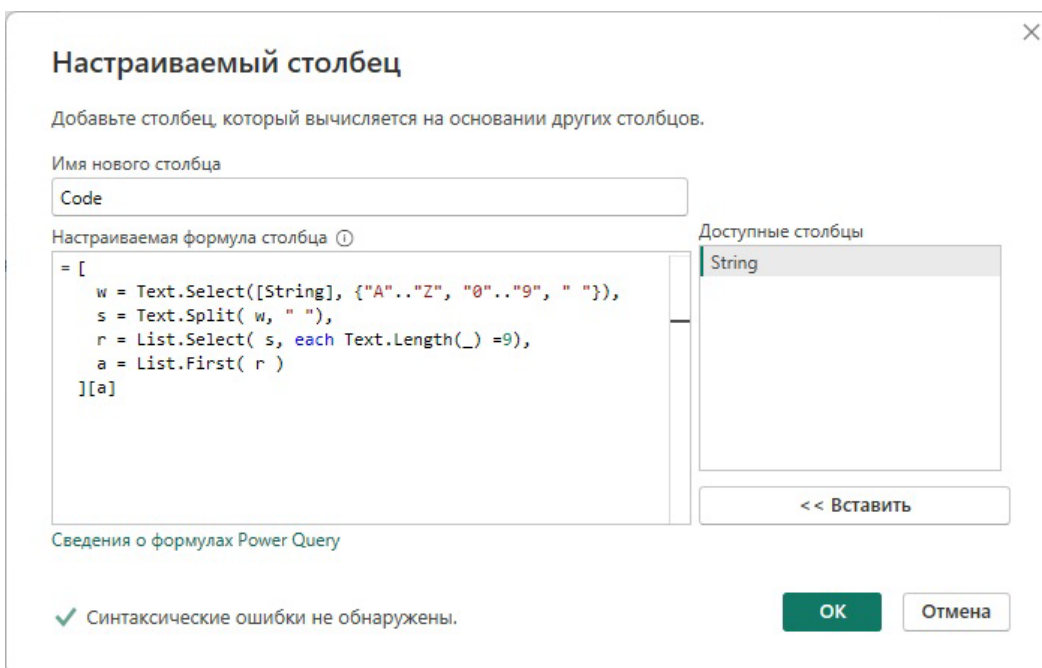
Разделите каждую строку по символу пробела, чтобы получить список с текстовыми значениями.

```
r = List.Select(s, each Text.Length(_) = 9)
```

Для начала мы выберем любой элемент списка с общей длиной в девять символов.

```
a = List.First(r)
```

Мы можем извлечь ответ с помощью функции *List.First*. Получите доступ к полю *a*, чтобы вернуть его значение после закрывающей скобки выражения записи. В итоге:



Настраиваемый столбец

Добавьте столбец, который вычисляется на основании других столбцов.

Имя нового столбца
Code

Настраиваемая формула столбца ⓘ

```
= [
  w = Text.Select([String], {"A".."Z", "0".."9", " "}),
  s = Text.Split( w, " "),
  r = List.Select( s, each Text.Length(_) =9),
  a = List.First( r )
][a]
```

Доступные столбцы
String

<< Вставить

Сведения о формулах Power Query

✓ Синтаксические ошибки не обнаружены.

OK Отмена

Рис. 14.1б. Настройка столбца *Code* в примере 2

Нажмите *OK*. В строке формул появится код для второго шага *Code*:

```
Table.AddColumn(Source, "Code", each [  
    w = Text.Select([String], {"A".."Z", "0".."9", " "}),  
    s = Text.Split(w, " "),  
    r = List.Select(s, each Text.Length(_) =9),  
    a = List.First(r )  
])[a]
```

В результате получим:

	String	Code
1	The code CHLO33446 is important for this task.	CHLO33446
2	Unique document identifier: JXKE13295.	JXKE13295
3	Please use the code SBT36006 to access the system.	null
4	For verification, enter ERKZ30881 on the website.	ERKZ30881
5	The transaction ID YNZ78388 must be noted.	null
6	IHCY10273 is the code for your appointment.	IHCY10273
7	Reference code SSK36144 is included in the report.	null
8	To complete registration, use PRL90158 as your code.	PRL90158
9	Package with tracking number MGA52287 has been shipped.	MGA52287

Рис. 14.2. Результат примера 2

Успех! Но, что мы упустили, и что можно улучшить? Вот о чем следует подумать:

- *Регистр букв.* Текущий метод учитывает только прописные буквы. Это автоматически исключает другие девятибуквенные слова, содержащие строчные буквы, включая любые коды с неверным регистром – они не будут распознаны.
- *Валидация шаблона.* Сам шаблон, состоящий из четырех букв, за которыми следуют пять цифр, не был проверен. Это может привести к неверным результатам.
- *Риск потери данных.* Существует риск потери данных, если в строке содержится более одного кода. Функция *List.Select* извлечет лишь первое прошедшее проверку значение.

Чтобы разработать надежное решение, которое устранил указанные проблемы, воспользуемся более коварным набором данных:

let

```
Source = Table.FromColumns(  
    {"The code CHLO33446 is IMPORTANT.",  
    "Unique document identifier: JXKE13295.",  
    "Please use the code SBT36006 to access the system.",  
    "For verification, enter ERKZ30881 and MdKZ85426.",  
    "The transaction ID YNZ78388 must be noted.",  
    "IHCY10273 is the code for your appointment.",  
    "Reference code SSK36144 is included in the report.",  
    "To complete registration, use PRL90158 as your code.",  
    "Package with tracking number MGA52287 has been shipped."},  
    type table [String=text]  
)
```

in

```
Source
```

Создайте новый запрос, поместите в него исходные данные, добавьте настраиваемый столбец. Иницилируйте запись и включите в нее следующие пары имя–значение:

```
w = Text.SplitAny( [String], "., :")
```

Разделим входную строку на более мелкие части. Разбиение происходит по любому символу, указанному во втором аргументе, возвращая список текстовых значений.

```
R = List.Select(  
    w,  
    each Text.Length(_) = 9
```

```
w,
each Text.Length(_)
= 9 and Text.Start(_, 4)
= Text.Select(_, {"A" .. "Z", "a" .. "z"}) and Text.Range(_, 4, 5)
= Text.Select(_, {"0" .. "9"})
)
```

Выберите любой элемент из этого списка $r = List.Select(w, each, \text{соответствующий шаблону})$. Т.е., имеющий общую длину в девять символов ($Text.Length(_) = 9$), где первые четыре символа – это буквы ($Text.Start(_, 4) = Text.Select(_, {"A" .. "Z", "a" .. "z"})$), а следующие пять – цифры ($Text.Range(_, 4, 5) = Text.Select(_, {"0" .. "9"})$). Закройте скобку в функции $List.Select$ и добавьте запятую.

```
a = Text.Combine(r, ", ")
```

Извлеките и объедините все результаты в одну строку. Примените доступ к полю, чтобы получить значение для a . В итоге получим код для шага *Code*:

```
Table.AddColumn(
Source,
"Code",
each [
w = Text.SplitAny([String], ".", ":"),
r = List.Select(
w,
each Text.Length(_)
= 9 and Text.Start(_, 4)
= Text.Select(_, {"A" .. "Z", "a" .. "z"}) and Text.Range(_, 4, 5)
= Text.Select(_, {"0" .. "9"})
),
a = Text.Combine(r, ", ")
][a]
)
```

	A ^B C String	ABC 123 Code
1	The code CHLO33446 is IMPORTANT.	CHLO33446
2	Unique document identifier: JXKE13295.	JXKE13295
3	Please use the code SBT36006 to access the system.	
4	For verification, enter ERKZ30881 and MdKZ85426.	ERKZ30881, MdKZ85426
5	The transaction ID YNZ78388 must be noted.	
6	IHCY10273 is the code for your appointment.	IHCY10273
7	Reference code SSK36144 is included in the report.	
8	To complete registration, use PRLL90158 as your code.	PRLL90158
9	Package with tracking number MGA5K2287 has been shipped.	

Рис. 14.3. Результат улучшенного кода для примера 2

Пример 3, разделители

Функции разделителей (*splitter*) в языке M особенно полезны для извлечения шаблонов. Эти функции были подробно рассмотрены [в главе 11 Сравнение, замена, соединение, разделение](#). Однако, поскольку форматы данных с фиксированной шириной очень распространены во многих устаревших системах, будет уместно рассмотреть этот конкретный сценарий здесь. В отличие от форматов, где поля разделяются символами, такими как запятые или табуляции, форматы с фиксированной шириной полагаются исключительно на позицию данных в строке. Это различие часто представляет собой вызов: поля встроены в большой блок текста без явных разделителей.

```
let
Source = Table.FromRows({
{"Hardware ZEQQNZE Nails - 2 inch 4SPYBBU8 WRPBOSGENTFD300 2024-01-10"},
{"Software CGUL2L Antivirus SW ZH1R987S2 SADBTO 150 2024-01-11"},
```

```

{"Hardware OY6IL4VFH21 Hammer - 5kg O08AUF8JG 80LC6OMO 75 2024-01-12"},
{"Software DLTQ80V7X Operating System NZ8DD797 AB9MTEI09L 200 2024-01-13"},
{"FurnitureWCPKVSZJX Office Chair FTHJK QQMR1QZMR71 50 2024-01-14"},
{"FurnitureZRE4CCR1 Office Desk OYOWTO7IOQMFSNGRQ5 40 2024-01-15"},
{"Hardware WJAA7DHJ Screwdriver Set EO65VPCJ IGC2FHI8G 120 2024-01-16"},
{"Software JX6URB9HI Database Software 143OD1ADFC4 3YD7QU07T8OF300 2024-01-17"},
{"Hardware 1ADXHN Electric Drill AK3M3MJMY6M XUBQZP7R 60 2024-01-18"},
{"FurnitureD20R63MNM6 Bookshelf 4G6GD3G 373NWC1I9JT55 2024-01-19"}
}, type table [Column1=text])

```

in

Source

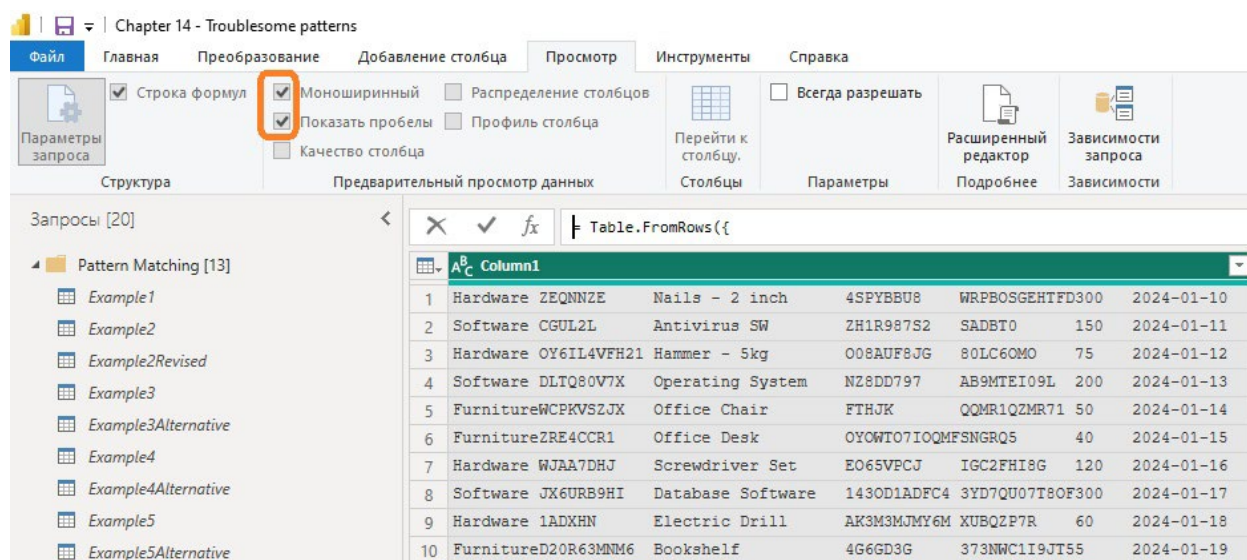


Рис. 14.3а. Пример 3; чтобы увидеть структуру данных включите опции на вкладке *Просмотр*

Для обработки данных фиксированной ширины используем *Splitter.SplitTextByPositions*. Этот разделитель принимает список позиционных индексных номеров. Каждый индекс должен быть больше или равен 0, а каждое следующее значение должно быть равно или больше предыдущего. Эти индексы указывают отсчитываемое от нуля место в тексте, где происходит разбиение, что идеально согласуется со структурой, присущей форматам данных фиксированной ширины. В следующей таблице показано, как определить эти позиции на основе известных длин полей устаревшей системы и, что очень важно, также указан порядок полей:

Field	Length	Position	Method	Order
Item type	10	0	Zero-based starting position = 0	0
Item code	12	9	Previous field length - 1	1
Description	20	21	Prev field len + Prev position	2
Supplier code	12	41	Prev field len + Prev position	3
Manufacturer code	12	53	Prev field len + Prev position	4
Quantity	6	65	Prev field len + Prev position	5
Date	10	71	Prev field len + Prev position	6

Рис. 14.3б. Определение позиций и порядка полей

Из каждой строки данных мы хотим извлечь код товара (*Item Code*), код поставщика (*Supplier Code*) и код производителя (*Manufacturer Code*). Мы продемонстрируем два метода, первый из которых использует *Splitter.SplitTextByPositions*. Зная, как определить каждую позицию, и имея порядок полей (рис. 14.3б), мы можем легко решить эту задачу.

Создайте новый запрос, поместите в него исходные данные, добавьте настраиваемый столбец. Иницилируйте запись и включите в нее следующие пары имя–значение:

```
s = Splitter.SplitTextByPositions({0, 9, 21, 41, 53, 65, 71})([Column1])
```

`Splitter.SplitTextByPositions` принимает один аргумент – список с позициями, и возвращает функцию. Чтобы вызвать эту функцию, добавьте еще один набор скобок, который является оператором вызова функции, и передайте имя столбца. Не забудьте про запятую в конце.

```
r = List.Transform( {1, 3, 4}, (x)=> Text.Trim( s{x} ))
```

В таблице 14.3б мы найдем порядок полей и отсчитываемый от нуля индекс каждого поля. Код товара (1), код поставщика (3) и код производителя (4) соответствуют позиции списка в `s`. Мы можем создать список с этими индексами для итерации, извлекая эти элементы и удаляя лишние пробелы за одну операцию.

Можно вместо `x` использовать нижнее подчеркивание в качестве единственной переменной:

```
r = List.Transform( {1, 3, 4}, (_) => Text.Trim( s{ _ } ))
```

```
a = Text.Combine( r, ", " )
```

Объедините все результаты в одну строку. Верните значение поля `a`, применив доступ к полю записи. Полный код шага:

```
AddCode
= Table.AddColumn(
    Source,
    "Code",
    each [
        s = Splitter.SplitTextByPositions({0, 9, 21, 41, 53, 65, 71})([Column1]),
        r = List.Transform({1, 3, 4}, (_) => Text.Trim(s{ _ })),
        a = Text.Combine(r, ", ")
    ][a]
)
```

На рис. 14.4 показаны выходные данные этого решения при включенных опциях *Моноширинный* и *Показать пробелы*, расположенных на вкладке *Просмотр*. Такое представление позволяет визуально убедиться в том, что данные имеют фиксированную ширину.

	Column1	Code
1	Hardware ZEQQNZE Nails - 2 inch 4SPYBBU8 WRPBOSGEHTFD300 2024-01-10	ZEQQNZE, 4SPYBBU8, WRPBOSGEHTFD
2	Software CGUL2L Antivirus SW ZH1R987S2 SADBT0 150 2024-01-11	CGUL2L, ZH1R987S2, SADBT0
3	Hardware OY6IL4VFH21 Hammer - 5kg O08AUF8JG 80LC6OMO 75 2024-01-12	OY6IL4VFH21, O08AUF8JG, 80LC6OMO
4	Software DLTQ80V7X Operating System NZ8DD797 AB9MTEI09L 200 2024-01-13	DLTQ80V7X, NZ8DD797, AB9MTEI09L
5	FurnitureWCPKVSZJX Office Chair FTHJK QQMR1QZMR71 50 2024-01-14	WCPKVSZJX, FTHJK, QQMR1QZMR71
6	FurnitureZRE4CCR1 Office Desk OYOWT07IOQMFSNGRQ5 40 2024-01-15	ZRE4CCR1, OYOWT07IOQMF, SNGRQ5
7	Hardware WJAA7DHJ Screwdriver Set E065VPCJ IGC2FHI8G 120 2024-01-16	WJAA7DHJ, E065VPCJ, IGC2FHI8G
8	Software JX6URB9HI Database Software 143OD1ADFC4 3YD7QU07T8OF300 2024-01-17	JX6URB9HI, 143OD1ADFC4, 3YD7QU07T8OF
9	Hardware 1ADXHN Electric Drill AK3M3MJMY6M XUBQZP7R 60 2024-01-18	1ADXHN, AK3M3MJMY6M, XUBQZP7R
10	FurnitureD20R63MNM6 Bookshelf 4G6GD3G 373NWC1I9JT55 2024-01-19	D20R63MNM6, 4G6GD3G, 373NWC1I9JT5

Рис. 14.4. Результат примера 3

Если в отчет устаревшей системы не вносятся изменения, такой метод практически не требует обслуживания и обработки ошибок. Если же исходные данные могут измениться нам пригодится альтернативный подход, который не полагается на функцию разделителя, но использует преимущества фиксированных позиций.

Важно помнить, что движок `M` генерирует код `M` во время работы с интерфейсом. Этот процесс основан на переменных, они же имена шагов. Однако если вы вставите пользовательский код, зависящий от переменных, дальнейшие взаимодействия с пользовательским интерфейсом могут нарушить эту красивую структуру. Чтобы избежать проблем, помните, что движок `M` безразличен к порядку шагов, потому что он следует цепочке зависимостей выражения внутри `let`. Разместив пользовательские переменные над шагом *Источник* (`Source`), вы минимизируете риск утраты возможности видеть шаги.

В [главе 8 Работа с вложенными структурами](#) мы показали, как функция `List.Zip` формирует новый список списков, связывая элементы из всех своих входных списков с соответствующими индексными позициями, где каждый вложенный список содержит элемент из `List 1` (позиции), за

которым следует элемент из *List 2* (длина поля). Для повышения эффективности, тем более что мы будем обращаться к этому списку неоднократно, мы сохраним его в памяти. Хотя этот конкретный код не ссылается напрямую на переменную из выражения *let*, мы рекомендуем придерживаться стиля и разместить его над шагом *Source*:

```
s = List.Buffer( List.Zip(  
  {  
    {0, 9, 21, 41, 53, 65, 71},  
    {10, 12, 20, 12, 12, 6, 10}  
  }  
)),
```

Используем исходные данные с рис. 14.3а. Сосредоточимся на логике формулы настраиваемого столбца. Инициализируем выражение записи и назовем первое поле *r*. Мы знаем порядок полей из устаревшей системы, индекс каждого поля, отсчитываемый от нуля, и стремимся извлечь код товара (1), код поставщика (3) и код производителя (4) из текстовой строки. Это можно сделать, поместив их позиции в список для итерации:

```
List.Transform( {1, 3, 4}, (x) => ... )
```

Мы извлечем всю длину каждого поля, включая конечные пробелы, и удалим последние:

```
Text.Trim( ... )
```

Внутри *Text.Trim* используем функцию [Text.Range](#) для извлечения трех полей, передавая строку из *Column1* в качестве первого аргумента:

```
Text.Range( [Column1], ... )
```

Обсудим подробнее второй аргумент. Переменная *s* представляет собой список списков, соответствующих каждому полю в строке. Мы можем получить соответствующий список с помощью *s{x}*, где *x* – текущее значение из списка, который мы итерируем внутри *List.Transform*. Поскольку результатом *s{x}* является список с двумя элементами, первый из которых является позицией, а второй – длиной поля, мы можем еще раз применить доступ к элементу для извлечения значения позиции, добавив {0}. Аргумент второго параметра *Text.Range* должен выглядеть следующим образом: *s{x}{0}*.

Третий аргумент *Text.Range* – количество возвращаемых символов. Но ведь это второй элемент в каждом из вложенных списков *s{x}*. Чтобы получить его значение, укажем: *s{x}{1}*.

Не забудьте про закрывающую скобку функции *Text.Range*.

Определите второе поле с именем *a* и задайте его значение следующим выражением для объединения всех результатов:

```
Text.Combine( r, " , " )
```

Наконец, примените доступ к полю к выражению записи, чтобы получить значение для *a*:

```
[  
  r = List.Transform({1, 3, 4}, (x) => Text.Trim(Text.Range([Column1], s{x}{0}, s{x}{1}))),  
  a = Text.Combine(r, " , ")  
][a]
```

Полный код альтернативного варианта примера 3:

```
let  
  s = List.Zip(  
    {  
      {0, 9, 21, 41, 53, 65, 71},  
      {10, 12, 20, 12, 12, 6, 10}  
    }  
  ),  
  Source = Table.FromRows({  
    {"Hardware ZEQQNZE Nails - 2 inch 4SPYBBU8 WRPBOSGEHTFD300 2024-01-10"},
```

```

{"Software CGUL2L Antivirus SW ZH1R987S2 SADBT0 150 2024-01-11"},
{"Hardware OY6IL4VFH21 Hammer - 5kg O08AUF8JG 80LC6OMO 75 2024-01-12"},
{"Software DLTQ80V7X Operating System NZ8DD797 AB9MTEI09L 200 2024-01-13"},
{"Furniture WCPKVSZJX Office Chair FTHJK QQMR1QZMR71 50 2024-01-14"},
{"Furniture ZRE4CCR1 Office Desk OYOWTO7IOQMFSNGRQ5 40 2024-01-15"},
{"Hardware WJAA7DHJ Screwdriver Set EO65VPCJ IGC2FHI8G 120 2024-01-16"},
{"Software JX6URB9HI Database Software 143OD1ADFC4 3YD7QU07T8OF300 2024-01-17"},
{"Hardware 1ADXHN Electric Drill AK3M3MJMY6M XUBQZP7R 60 2024-01-18"},
{"Furniture D20R63MNM6 Bookshelf 4G6GD3G 373NWC19JT55 2024-01-19"}
}, type table [Column1=text]),
AddCode = Table.AddColumn( Source, "Code", each
[
    r = List.Transform( {1, 3, 4}, (x)=>
        Text.Trim(
            Text.Range( [Column1], s{x}{0}, s{x}{1} )
        )
    ),
    a = Text.Combine( r, ", " )
][a]
)
in
AddCode

```

Пример 4, замена

Еще один подход для извлечения фиксированных шаблонов – это замена. Этот метод, возможно, самый сложный из всех. Он требует преобразования каждого допустимого символа в символ шаблона. Например, каждая буква преобразуется в каретку (^), каждая цифра — в хэштег (#), а каждый разделитель – в подчеркивание (_). Этот процесс приводит к созданию шаблона, что упрощает его идентификацию и извлечение.

Несмотря на эффективность подхода, он может быть излишне сложным. Особенно, учитывая количество более простых альтернатив. Но иногда замена может быть оправдана. Например, для такого набора данных:

```

let
Source = Table.FromColumns(
    {"The code CHLO-33-446 is IMPORTANT.",
    "Unique document identifier: JXKE-13-295.",
    "Please use the code SBT3-6006 to access the system.",
    "For verification, enter ERKZ-30-881 and MdKZ-85.426.",
    "The transaction ID YNZ-78-388 must be noted.",
    "IHC-Y10-273 is the code for your appointment.",
    "Reference code SSK-36-144 is included in the report.",
    "To complete registration, use PRL-90-158 as your code.",
    "Package with tracking number MGA5-K22.87 has been shipped."},
    type table [String=text]
)
in
Source

```

В этом сценарии допустимые коды определены следующим образом: они начинаются с четырех букв – заглавных, строчных или их комбинации, за которыми следует разделитель: дефис (-) или точка (.), затем две цифры, еще один разделитель и, наконец, набор из трех цифр. Общая длина шаблона постоянна и равна 11 символам. Целевые коды могут быть такими: ABCD-12-345, efgh-67.890 или ijkl.01.321.

Помимо основной замены значений, оставшиеся шаги схожи с альтернативным подходом с использованием *Splitters* из Примера 3. Создайте список замен. Он включает значение замены для

каждого допустимого символа и загружается в память. Хотя шаг *replacements* не связан с переменной из выражения *let*, придерживаясь лучших практик мы разместим его до шага *Source*:

```
Replacements = List.Buffer(  
  List.Zip({"A" .. "Z"}, List.Repeat({"^"}, 26))  
    & List.Zip({"0" .. "9"}, List.Repeat({"#"}, 10))  
    & List.Zip({"-", "."}, List.Repeat({"_"}, 2))  
)
```

Добавьте в таблицу пользовательский столбец и инициализируйте выражение записи []. Используйте *List.ReplaceMatchingItems* для замены всех символов в каждой строке. Для этого требуется список замен в формате {oldValue, newValue}, а также *Comparer.OrdinalIgnoreCase*, чтобы исключить влияние регистра букв. После проведения замен объедините все элементы списка, чтобы сформировать одну текстовую строку:

```
replVal = Text.Combine(  
  List.ReplaceMatchingItems(  
    Text.ToList([String]),  
    replacements,  
    Comparer.OrdinalIgnoreCase  
  )  
)
```

Теперь мы можем идентифицировать каждый экземпляр шаблона, который выглядит следующим образом : `^^^^_##_###` и собрать их позиции в списке:

```
lookUp = Text.PositionOf( replVal, "^^^^_##_###", Occurrence.All)
```

Используйте список поиска для итерации и извлечения результатов из входной строки:

```
getMatches = Text.Combine(  
  List.Transform(  
    lookUp,  
    (x)=> Text.Range([String], x, 11 )  
  ), ", "  
)
```

Поскольку мы использовали выражение записи для генерации всех значений, осталось вернуть конечный результат, хранящийся в поле *getMatches*, применив доступ к полю. Итоговый код шага *AddCode*:

```
Table.AddColumn(  
  Source,  
  "Code",  
  each [  
    replacements = List.Buffer(  
      List.Zip({"A" .. "Z"}, List.Repeat({"^"}, 26))  
        & List.Zip({"0" .. "9"}, List.Repeat({"#"}, 10))  
        & List.Zip({"-", "."}, List.Repeat({"_"}, 2))  
    ),  
    replVal = Text.Combine(  
      List.ReplaceMatchingItems(Text.ToList([String]), replacements, Comparer.OrdinalIgnoreCase)  
    ),  
    lookUp = Text.PositionOf(replVal, "^^^^_##_###", Occurrence.All),  
    getMatches = Text.Combine(List.Transform(lookUp, (x) => Text.Range([String], x, 11)), ", ")  
  ][getMatches]  
)
```

	String	Code
1	The code CHLO-33-446 is IMPORTANT.	CHLO-33-446
2	Unique document identifier: JKKE-13-295.	JKKE-13-295
3	Please use the code SBT3-6006 to access the...	
4	For verification, enter ERKZ-30-881 and MdK...	ERKZ-30-881, MdKZ-85.426
5	The transaction ID YNZ-78-388 must be noted.	
6	IHC-Y10-273 is the code for your appointmen...	
7	Reference code SSK-36-144 is included in th...	
8	To complete registration, use PRL-90-158 a...	PRL-90-158
9	Package with tracking number MGA5-K22.87 ha...	

Рис. 14.5. Результат примера 4

Мы отметили, что существуют более простые альтернативы, даже в тех случаях, когда вам необходимо проверять каждый отдельный символ. Чтобы продемонстрировать это, давайте используем тот же набор данных.

Добавьте пользовательский столбец *Code* и инициализируйте выражение записи: []. Разбейте каждую строку на слова:

```
w = Text.Split([String], " ")
```

Выберите элементы списка, соответствующие шаблону. Начните игнорировать слова, которые не соответствуют минимальному требованию к символам (менее 11 символов). Во всех остальных случаях проверьте значение, удалив все допустимые символы из каждого сегмента и проверив, равна ли оставшаяся длина текста нулю:

```
r
= List.Select(
  w,
  (x) =>
    if Text.Length(x) < 11 then
      false
    else
      Text.Length(
        Text.Remove(Text.Start(x, 4), {"A" .. "Z", "a" .. "z"})
        & Text.Remove(Text.Range(x, 4, 1) & Text.Range(x, 7, 1), {"-", "."})
        & Text.Remove(Text.Range(x, 5, 2) & Text.Range(x, 8, 3), {"0" .. "9"})
      )
      = 0
    )
)
```

Из оставшихся элементов списка извлеките первые 11 символов:

```
a = Text.Combine( List.Transform(r, (x) => Text.Start(x, 11)), " ")
```

Извлеките значение из поля *a*, применив доступ к полю – [a].

Итоговый код шага *AddCode*:

```
Table.AddColumn(
  Source,
  "Code",
  each [
    w = Text.Split([String], " "),
    r = List.Select(
      w,
      (x) =>
        if Text.Length(x) < 11 then
          false
        else
          Text.Length(
```

```

    Text.Remove(Text.Start(x, 4), {"A" .. "Z", "a" .. "z"})
    & Text.Remove(Text.Range(x, 4, 1) & Text.Range(x, 7, 1), {"-", "."})
    & Text.Remove(Text.Range(x, 5, 2) & Text.Range(x, 8, 3), {"0" .. "9"})
  )
  = 0
),
a = Text.Combine(List.Transform(r, (x) => Text.Start(x, 11)), ", ")
][a]
)

```

Пример 5, регулярное выражение

Power Query не поддерживает регулярные выражения, но их можно реализовать с помощью функции *Web.Page* для выполнения кода *JavaScript (JS)*. Производительность этого метода страдает, особенно при применении к большим наборам данных. Убедитесь, что преимущества метода перевешивают недостатки.

В следующем примере мы хотим извлечь почтовые индексы – 5 цифр подряд:

Source

```

= Table.FromColumns(
  {
    {
      "Boulevard des Écoles 73, 31000 Lyon",
      "6 Boulevard du Château, 69001 La Ville Rose Toulouse",
      "Rue Saint-Martin 65, 31000 Lyon",
      "Chemin Victor Hugo 143, 69001 Bordeaux",
      "Avenue des Vignes 7, 67000 Toulouse",
      "74 Quai de la République 69001 Latin Quarter Paris",
      "55 Boulevard de la Liberté, 67000 La Petite France Strasbourg",
      "82 Chemin des Jardins, 59000 Paris"
    }
  },
  type table [Address = text]
)
In
Source

```

Создадим пользовательскую функцию на основе *Web.Page* для запуска *JavaScript*. Нам потребуется текстовое значение с тегом `<script>` для вставки кода *JavaScript* в документ *HTML*. Все, что находится между `<script>` и `</script>` выполняется как *JavaScript*. Наша задача – создать JS-код, выполняющий операцию с регулярным выражением. Ему необходимо идентифицировать и извлечь последовательности из пяти цифр из входной текстовой строки, переданной пользовательской функции. После этого функция *Web.Page* выведет таблицу, показывающую содержимое HTML-документа, организованную по основным элементам. Затем можно получить доступ к этой таблице для получения результатов. Хотя мы не являемся экспертами в JS-коде, вот как это может выглядеть. Придерживаясь лучших практик, мы реализуем эту пользовательскую функцию над шагом *Source*:

```

fxRegex = (input as text) as text =>
  Web.Page(
    "<script>
      var a = "" & input & ""; // входной текст
      var b = a.match(/\d{5}/g); // регулярное выражение
      document.write(b); // выходной текст
    </script>"
  ){0}[Data]{0}[Children]{1}[Children]{0}[Text]

```

Подробнее о коде функции:

```
fxRegex = (input as text) as text =>
```

Объявляется функция с именем *fxRegex*, которая принимает один параметр *input* типа *text*. Функция вернет значение текстового типа.

Web.Page(...)

... создает веб-страницу в памяти, и внутри себя запускает JavaScript (текстовая строка в паре двойных кавычек), который поддерживает регулярные выражения. Все, что находится между тегами `<script>` и `</script>`, обрабатывается и выполняется как код JavaScript.

```
var a = "" & input & "";
```

Объявляет переменную JavaScript с именем *a*. Одинарная кавычка (') после знака равенства иницирует строку в JavaScript. Двойная кавычка (") сигнализирует об окончании текстовой строки в M, временно пропуская нотацию JS. Это позволяет конкатенировать входное значение параметра (& input) в коде M. После внедрения входного значения M конкатенирует новую строку (& ") и продолжает работу в JavaScript, закрывая строку одинарной кавычкой (').

```
var b = a.match(/\d{5}/g);
```

Объявляет переменную *b* в JavaScript и использует метод *match* JavaScript с шаблоном регулярного выражения, где `\d` соответствует любой цифре, `{5}` указывает ровно пять цифр подряд, а *g* – это глобальный флаг для поиска всех совпадений.

Подробнее о регулярных выражениях см. [Джеффри Фридл. Регулярные выражения.](#)

```
document.write(b);
```

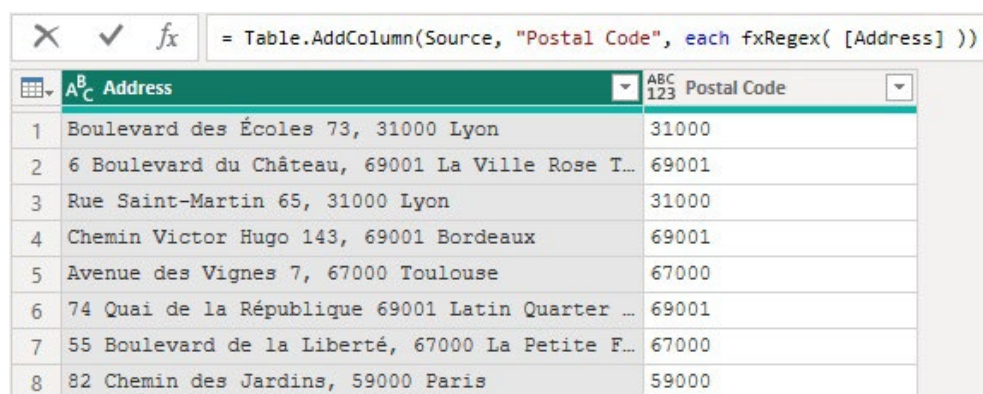
Записывает результат совпадения регулярного выражения в HTML-документ.

```
{0}[Data]{0}[Children]{1}[Children]{0}[Text]
```

Эта часть обращается к выходным данным, сгенерированным *Web.Page*, и перемещается по иерархии html-документа для получения текста, записанного с помощью *document.write(b)*. Несколько раз применяется доступ к элементам и полям для извлечения содержимого.

Логика функции *fxRegex* мы применяем к каждой строке в таблице, сохраняя результаты в столбце *Postal Code*. Поскольку пользовательская функция такая же, как и любая другая функция в M, мы добавим Пользовательский столбец и введем: *fxRegex([Address])*. Шаг *InvokedCF*:

```
Table.AddColumn(Source, "Postal Code", each fxRegex( [Address] ))
```



	Address	Postal Code
1	Boulevard des Écoles 73, 31000 Lyon	31000
2	6 Boulevard du Château, 69001 La Ville Rose T...	69001
3	Rue Saint-Martin 65, 31000 Lyon	31000
4	Chemin Victor Hugo 143, 69001 Bordeaux	69001
5	Avenue des Vignes 7, 67000 Toulouse	67000
6	74 Quai de la République 69001 Latin Quarter ...	69001
7	55 Boulevard de la Liberté, 67000 La Petite F...	67000
8	82 Chemin des Jardins, 59000 Paris	59000

Рис. 14.7. Результат вызова функции *fxRegex*

Метод JavaScript поддерживается в Power BI Desktop и Excel, но недоступен в службе Power BI. В службе Power BI регулярные выражения можно реализовать с помощью скриптов на Python или R.

Учитывая проблемы с запуском *JavaScript*, можно ли решить задачу с помощью стандартных функций M? Используем тот же набор данных.

Чтобы добавить столбец *Postal Code* в таблицу, пройдите *Добавление столбца* → *Настраиваемый столбец*. Введите *Postal Code* в качестве имени столбца. Инициализируйте выражение записи, введя набор квадратных скобок [] в поле *Настраиваемая формула столбца*. Введите три записи:

```
w = Text.SplitAny([Address], Text.Remove([Address], {"0".."9"})),
```

Разделит входную строку. В качестве разделителя используем функцию *Text.Remove* для включения всех символов из входной строки, за исключением цифр. Добавьте запятую в конце, чтобы создать новое поле в записи.

```
r = List.Select( w, each Text.Length(_)=5),
```

Выберите все элементы из списка, которые имеют общую длину 5 символов.

```
a = Text.Combine(r, ", ")
```

Объедините результаты в одну строку. Извлеките значение поля, применив доступ к полю к записи [...]a]. Итоговый код шага *Extracted2*:

```
Table.AddColumn(  
  Source,  
  "Postal Code",  
  each [  
    w = Text.SplitAny([Address], Text.Remove([Address], {"0" .. "9"})),  
    r = List.Select(w, each Text.Length(_)=5),  
    a = Text.Combine(r, ", ")  
  ]a  
)
```

Этот лаконичный код позволяет достичь результата и будет работать лучше, чем регулярное выражение на *JS*.

Переходя ко второй части этой главы, мы сместим акцент на объединение данных. Этот процесс позволяет добавлять информацию, например, из различных таблиц, листов или книг.

Объединение данных

В реальном мире данные часто поступают из различных источников и управляются несколькими пользователями, что усложняет поддержание согласованности между файлами. В этом разделе мы рассмотрим работу с несколькими файлами Excel.

Основы объединения данных

Процесс ETL (Extract, Transform and Load; извлечение, преобразование и загрузка) начинается с подключения к источнику данных. Вот некоторые моменты, о которых следует помнить, особенно при работе с несколькими файлами:

Расположение данных. Это уникальный адрес. Например, путь к локальному / сетевому диску или URL-адрес. Местоположение и имена файлов не статичны. Они могут меняться со временем по разным причинам, например, миграция систем, реорганизация данных и др. Здесь вступают в игру параметры. Они действуют как переменные и позволяют хранить и управлять значением, которое можно использовать в запросе. Параметры позволяют обновлять и изменять значение один раз, а не в каждом запросе, в котором используется это значение. Подробнее см. [глава 9 Параметры и пользовательские функции](#).

Источник данных. Для считывания данных используется коннектор. Необходимо определить, доступен ли коннектор для источника. Если коннектор по умолчанию недоступен, можно разработать собственный. Этот процесс описан в главе 16 Включение расширений.

Выбор данных. Данные могут храниться в папке и распределены по нескольким файлам. Важно знать, какие файлы следует рассмотреть, и если требуется отбор, то какие критерии применять. Например, диапазон дат, расширение файла или другой атрибут.

Организация данных. Как данные организованы в файлах? Вы имеете дело с одним или несколькими листами и таблицами в рабочей книге? Возможны ли файлы без данных? Представляют ли они интерес, и как отражать отсутствие данных?

Структура данных. Поскольку мы говорим о файлах Excel, изучите, как данные структурированы в каждом файле. Они расположены на листах или в таблицах? Унифицированы ли названия таблиц для удобства отбора? Для данных на листах, как определять, где находятся данные?

Согласованность данных. Проверка на единообразии в разных файлах, например совпадение имен столбцов, согласование типов значений в столбцах с одинаковыми именами и т. д. Могут иметься соглашения, которые делают имена столбцов уникальными. Например, *1-1-2024 Stock On Hand*. Столбцы с такими именами будут уникальными. Но вы можете извлечь дату, и унифицировать названия.

Сбор данных. Если данные в выбранных файлах одинаковые, их можно объединить в общую таблицу. Если нет, то вероятно, это различные наборы данных. Помните, что для каждой таблицы должен быть разработан отдельный запрос. Представьте себе файлы из финансового отдела, в каждом из которых есть баланс, бюджет, отчет о прибылях и убытках, таблица расходов и др. Очевидно, что они описывают разное. Какие данные необходимо собрать для анализа?

Преобразование данных. Нужно ли преобразовать данные перед объединением? Есть ли закономерности или проблемы в данных, которые нужно устранить? Можно ли применить динамические преобразования? Нужен отчет об ошибках? Простая и эффективная стратегия оптимизации модели данных – использовать только то, что нужно. Ранний отбор столбцов и строк может дать дополнительное преимущество в виде свертывания запросов. Подробнее см. в главе 15 Оптимизация производительности.

Время и усилия, затраченные на разработку хорошо структурированных запросов, окупаются удобством обслуживания, надежностью и качеством сведений, которые вы можете получить на основе анализа. Перейдем к практическим примерам.

Извлечение, преобразование и объединение

Здесь мы обсудим работу с несколькими файлами Excel, хранящимися в одной папке. Рекомендую работать с примерами. Скачайте файлы с репозитория [GitHub](#). Существует множество методов для достижения желаемого результата. Мы проведем вас через одно конкретное решение. Скриншоты взяты из Power BI Desktop версии 2.124.1805.0. Если вы используете другую версию, могут быть некоторые отличия.

Мои скриншоты взяты из Power BI Desktop версии 2.132.908.0 (август 2024).

Получение и проверка данных

Данные содержатся в девяти файлах. Три первых мы объединим, чтобы получить результат:

Имя	Дата изменения	Тип	Размер
Daily_1.xlsx	29.04.2024 20:08	Лист Microsoft Ex...	24 КБ
Daily_2.xlsx	29.04.2024 20:08	Лист Microsoft Ex...	24 КБ
Daily_3.xlsx	29.04.2024 20:08	Лист Microsoft Ex...	24 КБ
ExpectedCombined.xlsx	29.04.2024 20:08	Лист Microsoft Ex...	15 КБ

Рис. 14.9. Файлы для объединения и файл ожидаемого результата

Откройте эти три файла. Все они имеют одинаковую структуру:

Daily by Date Performance Data - Preliminary															
My Property: Business Name1 Comp Set: Location1 #111111, Location2 #2222222, Location 3 #333333, Location 4 #444444.															
Job Number: 37165321 Staff: User1 Created: May 12, 2023 Currency: EUR - Euros															
Date	DOW	Occupancy						ADR				This Year			
		My Prop	Comp Set	My Prop	Comp Set	Index (MPI)	Rank	My Prop	Comp Set	My Prop	Comp Set	Index (ARI)	Rank	My Prop	Comp S
05/22/2023	Mon	89.4	93.1	69.5	73.6	96.0	5 of 6	262.66	170.67	334.9	97.6	153.9	1 of 6	234.80	158
05/23/2023	Tue	100.0	98.6	87.5	77.2	101.4	1 of 6	201.63	188.13	325.4	67.3	107.2	2 of 6	201.63	185
05/24/2023	Wed	70.7	84.4	28.2	48.1	83.8	6 of 6	161.59	151.22	163.3	64.5	106.9	2 of 6	114.25	127
05/25/2023	Thu	61.6	77.6	9.3	12.4	79.4	5 of 6	134.60	143.95	85.8	47.5	93.5	4 of 6	82.94	111
05/26/2023	Fri	71.7	83.1	10.6	-12.5	86.3	5 of 6	122.46	166.62	20.7	21.5	73.5	6 of 6	87.82	138
05/27/2023	Sat	93.9	97.7	40.9	2.8	96.1	5 of 6	171.25	205.50	43.1	19.7	83.3	5 of 6	160.87	200
05/28/2023	Sun	78.8	95.3	20.4	4.4	82.6	6 of 6	120.65	142.41	7.9	-18.7	84.7	6 of 6	95.06	135
Partial May 2023		80.9	90.0	36.6	21.9	89.9	6 of 6	172.63	168.09	104.1	27.3	102.7	2 of 6	139.62	151
Period		80.9	90.0	36.6	21.9	89.9	6 of 6	172.63	168.09	104.1	27.3	102.7	2 of 6	139.62	151

Рис. 14.10. Фрагмент одного из файлов для объединения

А вот, что мы хотим получить:

	A	B	C	D		E		F		G	
1	Name	Date	DOW	Occupancy This Year My Prop	Occupancy This Year Comp Set	Occupancy % Chg My Prop	Occupancy % Chg Comp Set	Occupancy % Chg My Prop	Occupancy % Chg Comp Set	Occupancy % Chg My Prop	Occupancy % Chg Comp Set
2	Daily_1.xlsx	5/22/2023	Mon	89.393939393939391	93.126385809312623	69.540229885057471	73.553719008264466				95.99206
3	Daily_1.xlsx	5/23/2023	Tue	100	98.59571322985957	87.5	77.15803452852463				101.4242
4	Daily_1.xlsx	5/24/2023	Wed	70.707070707070713	84.405025868440504	28.205128205128204	48.119325551232166				83.77116
5	Daily_1.xlsx	5/25/2023	Thu	61.616161616161612	77.605321507760536	9.3189964157706093	12.419700214132762				79.39682
6	Daily_1.xlsx	5/26/2023	Fri	71.717171717171723	83.074648928307454	10.59190031152648	-12.52918287937743				86.32858
7	Daily_1.xlsx	5/27/2023	Sat	93.939393939393938	97.708795269770874	40.909090909090914	2.7993779160186625				96.14220
8	Daily_1.xlsx	5/28/2023	Sun	78.787878787878796	95.343680709534368	20.370370370370367	4.3689320388349522				82.63565
9	Daily_2.xlsx	5/22/2023	Mon	96.84210526315789	74.096754439681561	40.359760159893405	53.310164556783604				130.6968
10	Daily_2.xlsx	5/23/2023	Tue	97.89473684210526	78.199632578077157	35.866028708133967	31.6618163489374				125.1856

Рис. 14.11. Фрагмент ожидаемого результата

Метод, который мы собираемся рассмотреть, демонстрирует, как создать удобное в управлении решение, аналогичное тому, что предоставляет интерфейс пользователя при выборе *Объединить и преобразовать данные*. Наш метод будет эффективным, настраиваемым и модульным, что позволит передавать работу другим, даже с учетом того, что код содержит сложные концепции.

Принимая во внимание, что расположение файлов может меняться со временем, начнем разработку с создания параметра.

Параметр местоположения

Допустим мы храним загруженные файлы в папке *Sample files*. Вместо того чтобы жестко зафиксировать этот путь в запросе, создадим параметр. Значение параметра можно обновить без необходимости изменять код M и даже не открывая редактор Power Query.

Чтобы создать параметр в редакторе Power Query в области *Запросы* щелкните правой кнопкой мыши на свободном месте и выберите *Создать параметр*, или пройдите по меню *Главная* → *Управление параметрами* → *Создать параметр*:

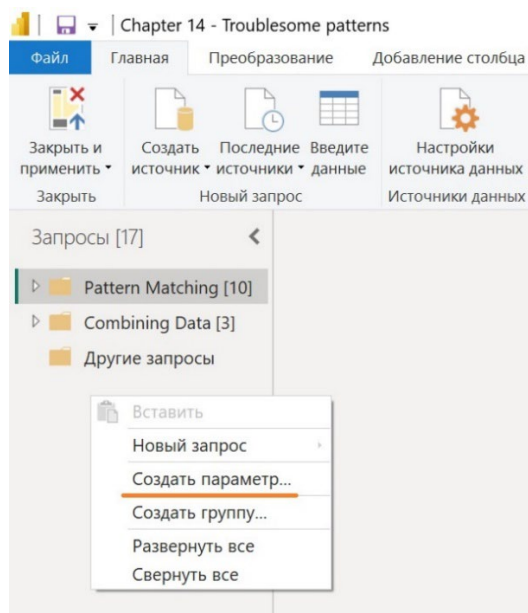


Рис. 14.12. Быстрое создание нового параметра в области *Запросы*

Настройте окно *Управление параметрами*. Убедитесь, что установлен флажок *Требуется*. В поле *Тип* выберите *Текст* и укажите *Текущее значение*. Каждый параметр хранится и отображается как отдельный запрос в области *Запросы*. В M имя запроса считается идентификатором. Когда имя запроса содержит пробелы, его нужно заключить в кавычки. Чтобы избежать этого, выберите имя без пробела, например *FolderLocation*.

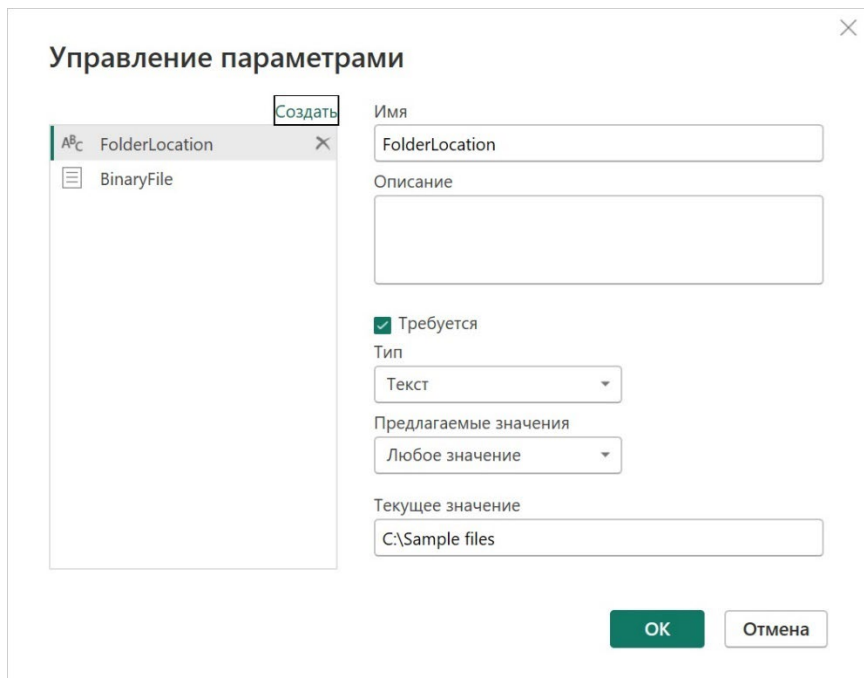


Рис. 14.13. Диалоговое окно *Управление параметрами*

Подключение к данным

В редакторе Power Query перейдите по меню *Главная* → *Создать источник* → *Дополнительно* → *Папка*. В окне *Папка* переключите тип ввода с *Текст* (обозначается значком ABC) на *Параметр*, выберите *FolderLocation* в качестве значения. Нажмите *OK*, отобразится содержимое папки:

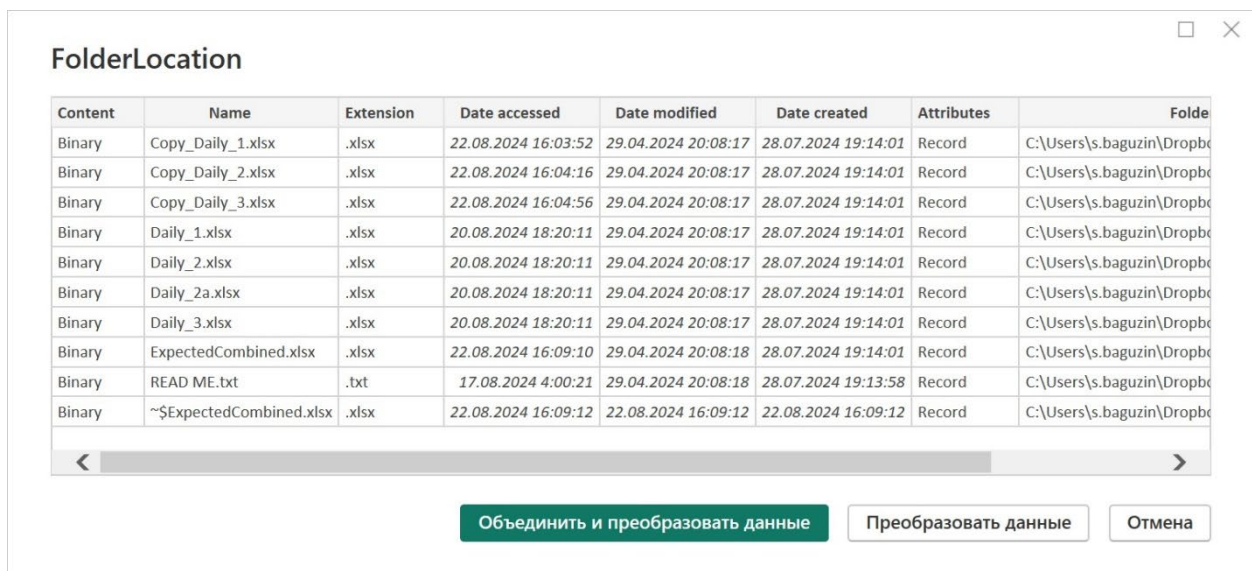


Рис. 14.14. Содержимое папки и доступные кнопки управления

Кнопки управления предлагают следующую функциональность:

Объединить и преобразовать данные. Эту кнопку выбирайте, если все содержимое папок, включая вложенные папки, должно обрабатываться одинаковым образом. В этом случае нет возможности отобрать файлы из папки. В будущем при изменении формата или структуры файлов объединение данных не будет выполнено. Например, если для *Файла примера* в интерфейсе была выбрана таблица *myData*, все файлы будут рассматриваться как файлы Excel, содержащие таблицы *myData*.

Преобразовать данные. Эту кнопку следует выбрать для ручного управления отбором и преобразованием файлов в редакторе Power Query перед объединением данных.

Отмена. Прервет операцию подключения к данным.

Поскольку наша цель – создать гибкое решение, позволяющее отбирать файлы для объединения, жмем *Преобразовать данные*. Создастся новый запрос с именем *Запрос1*, если это имя еще не существует. Он содержит данные, показанные на рис. 14.14.

После настройки подключения пришло время выбрать файлы для анализа.

Фильтрация файлов

Единственное преобразование, которое нам нужно применить к *Запросу1* – фильтр для выбора трех верхних файлов, изображенных на рис. 14.9. Начнем с того, что в интерфейсе по столбцу *Extension* выберем *Текстовые фильтры* → *Начинается с...* и введем *.xls*. Нажмите *OK*. Создастся шаг запроса с кодом:

```
Table.SelectRows(Source, each Text.StartsWith([Extension], ".xls"))
```

Используя код созданный интерфейсом в качестве базы, дополним его двумя опциями. Чтобы сравнение выполнялось без учета регистра, передадим в функцию *Text.StartsWith* третий параметр *Comparer.OrdinalIgnoreCase*. Дополним фильтр отбором файлов, название которых начинается с *Daily_*:

```
Table.SelectRows(
  Source,
  each Text.StartsWith([Extension], ".xls", Comparer.OrdinalIgnoreCase)
  and Text.StartsWith([Name], "Daily_", Comparer.OrdinalIgnoreCase)
)
```

Общая стратегия

Коннектор папок Power Query использует выбранный пользователем файл-образец для разработки логики преобразования, которая будет одинаковой для всех файлов в папке. Но этому процессу не хватает гибкости для выбора конкретных файлов или извлечения определенных наборов данных (таблиц или листов). Мы создадим решение, похожее на *Объединить и преобразовать данные*, но привнесем в него гибкость. Шаги нашего решения будут включать:

- удаление пустых строк и столбцов;
- извлечение названия компании из заголовка документа;
- извлечение основной части данных;
- обработку многострочных заголовков.

Чтобы начать этот процесс, нам нужно выбрать подходящий образец. Предположим, что в будущем мы захотим переключиться на другой файл примера. Поскольку это еще один вход, мы можем сохранить его в параметре.

Выберите образец файла

Значение *Binary* в столбце *Content* (рис. 14.14) содержит данные файла Excel. Думайте о *Binary* как о значении, которое нужно преобразовать. Изучив файлы, мы нашли *Daily_2.xlsx* подходящим для образца. Теперь:

- щелкните правой кнопкой мыши пробел рядом с двоичным значением для *Daily_2.xlsx*;
- выберите *Добавить как новый запрос*;
- переименуйте этот новый запрос в *BinarySample*.

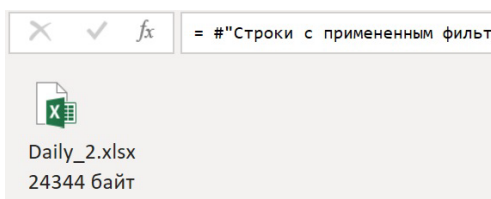


Рис. 14.14а. Бинарное содержимое запроса *BinarySample*

Параметр файла

Двоичное содержимое является входным значением. Разрабатывая структурированный и модульный подход, создадим для входного значения параметр:

- пройдите *Главная* → *Управление параметрами* → *Создать параметр*;
- дайте параметру говорящее имя *BinaryFile*;

- поставьте галку *Требуется*, выберите *Тип = Двоичный*;
- в полях *Значение по умолчанию* и *Текущее значение* установите *BinarySample*.

Шаблон преобразования

Мы продолжим настройку запроса, который принимает параметр *BinaryFile* в качестве входного значения:

- щелкните правой кнопкой мыши запрос *BinaryFile*;
- выберите *Ссылка*; будет добавлен новый запрос и в области предварительного просмотра отобразится двоичное содержимое файла *Daily_2.xlsx*, как на рис. 14.14а.
- дважды щелкните файл, чтобы вызвать функцию *Excel.Workbook*;
- переименуйте этот запрос в *TransformData*.

Код запроса...

```
let
    Source = Excel.Workbook(BinaryFile, null, true)
in
    Source
```

... а в окне предварительного просмотра появится таблица:

	Name	Data	Item	Kind	Hidden
1	Daily	Table	Daily	Sheet	FALSE

Рис. 14.16. Содержимое файла образца

Это еще одна важная точка в процессе разработки. Таблица может содержать более одной строки. Т.е., файл включает нескольких наборов данных. Для каждой таблицы придется создать новый запрос. Можно продублировать исходный запрос, дать ему подходящее имя и повторить процесс, описанный ниже, для каждой таблицы.

Чтобы справиться с возможными несоответствиями между файлами, которые затрудняют идентификацию данных, внедрим проактивные меры. Вместо того чтобы сразу раскрыть столбец *Data*, используем защитный механизм, устраняющий некоторые несоответствия, например регистр букв. Мы применим фильтры к строкам, *Kind = Sheet* и *Name = Daily*, и в первом случае выполним сравнение нечувствительное к регистру:

```
Table.SelectRows( Source,
    each Text.Lower([Name]) = "daily" and [Kind] = "Sheet"
)
```

Такое выражение фильтра никогда не вернет более одной строки, так как имена листов должны быть уникальными в пределах книги, а регистр букв не учитывается. Это позволяет безопасно детализировать столбец *Data* путем применения комбинации доступа к необязательному элементу и полю.

Внутри строки формул, после закрывающей скобки, примените доступ к необязательному элементу, *{0}?*, чтобы получить первую запись из таблицы. Далее укажите необязательный доступ к полю, *[Data]?*, чтобы вернуть значение из запрашиваемого поля. На панели *Примененные шаги* переименуйте этот шаг в *RAW*. В результате возвращается содержимое листа Excel:

	Column1	Column2	Column3	Column4	Column5	Column6
1	null	null	Daily by Date Performance D...	null	null	null
2	null	null	My Property: Business Name:	null	null	null
3	null	null	Comp Set: Location4 #444446...	null	null	null
4	null	null		null	null	null
5	null	null		null	null	null
6	null	null	Job Number: 37165322 Starr...	null	null	null
7	null	null		null	null	null
8	null	null		null	null	Occupancy
9	null	null	Date	DOW	null	This Year
10	null	null		null	null	My Prop
11	null	null	05/22/2023	Mon	null	96 84210526

Рис. 14.17. Предварительный просмотр данных

Удаление пустых строк

В данных довольно много пустых столбцов и строк. Логично удалить их, и в интерфейсе есть такая функция для строк. Пройдите *Главная* → *Удалить строки* → *Удалить пустые строки*. Переименуйте шаг в *NoEmptyRows*. Команда сгенерит код:

```
Table.SelectRows(RAW,  
  each not List.IsEmpty(List.RemoveMatchingItems(Record.FieldValues(_), {"", null}))  
)
```

Удаление пустых столбцов

Функции для удаления пустых столбцов в интерфейсе нет. Используем логику шага *NoEmptyRows*. Скопируйте часть кода М от слова *each* до предпоследней закрывающей скобки:

```
each not List.IsEmpty(List.RemoveMatchingItems(Record.FieldValues(_), {"", null}))
```

Нажмите f_x перед строкой формул, чтобы вставить шаг вручную. В строке формул вы увидите название предыдущего шага. Поместите курсор перед ним и введите *Table.SelectColumns*(

Переместите курсор в конец, поставьте запятую. Второй аргумент ожидает списка с именами столбцов. Мы предоставим его выражением, создающим список с именами столбцов на основе условия. Введите *List.Select*(. Передайте функции имена столбцов текущей таблицы: *Table.ColumnNames*(*NoEmptyRows*). Поставьте запятую и вставьте скопированную логику в качестве условия. Эта операция будет выполняться не построчно, а столбец за столбцом, поэтому *Record.FieldValues*(_) замените на *Table.Column*(*NoEmptyRows*, _), создав список со значениями столбцов. Добавьте нужное число закрывающих скобок. Переименуйте шаг в *NoEmptyCols*. Если вы нигде не ошиблись, у вас должно получиться:

```
Table.SelectColumns(  
  NoEmptyRows,  
  List.Select(  
    Table.ColumnNames(NoEmptyRows),  
    each not List.IsEmpty(  
      List.RemoveMatchingItems(  
        Table.Column(NoEmptyRows, _),  
        {"", null}  
      )  
    )  
  )  
)
```

В результате все столбцы, содержащие только пробелы и/или *null*, были удалены. Учитывая, что задача удаления пустых столбцов при работе с файлами Excel является типичной, полезно преобразовать этот код в пользовательскую функцию.

Пользовательская функция

В коде есть три ссылки на таблицу шага *NoEmptyRows*. Логично передать эту ссылку в качестве параметра функции. Чтобы создать пользовательскую функцию откроем запрос в расширенном редакторе. Разместим код функции над шагом *Source*:

- Скопируйте выражение, назначенное шагу *NoEmptyCols*.
- Добавьте пустую строку над *Source* и введите: $fxNoEmptyCols =$.
- Прежде чем мы вставить логику, мы хотим превратить ее в пользовательскую функцию. Инициуем функцию, введя круглые скобки и знак перехода: $() =>$. Для функции нужен один параметр – входное значение табличного типа. Назовем его *tbl* и укажем тип: *tbl as table*. Функция возвращает также значение типа *table*: $(tbl as table) as table =>$. Теперь можно вставить скопированный код. Замените все ссылки на таблицу параметром *tbl*.

Если вы все сделали верно, код функции:

```
fxNoEmptyCols = (tbl as table) as table =>  
  Table.SelectColumns(  
    NoEmptyRows,  
    List.Select(  
      Table.ColumnNames(NoEmptyRows),  
      each not List.IsEmpty(  
        List.RemoveMatchingItems(  
          Table.Column(NoEmptyRows, _),  
          {"", null}  
        )  
      )  
    )  
  )
```

```
tbl,
List.Select(
  Table.ColumnNames(tbl),
  each not List.IsEmpty(List.RemoveMatchingItems(Table.Column(tbl, _), {null, ""}))
)
)
```

Теперь, когда у нас есть пользовательская функция, нужно обновить код шага *NoEmptyCols*:

```
NoEmptyCols = fxNoEmptyCols( NoEmptyRows ),
```

Извлечение данных из заголовков

Мы планируем удалить все строки заголовка документа (рис. 14.18). Но перед этим необходимо извлечь ключевую информацию. Последнее слово во второй строке, *Name2*, является идентификатором свойства, который мы хотим поместить в новый столбец в итоговом выводе:

	A ^B _C Column3	A ^B _C Column4
1	Daily by Date Performance Data - Preliminary	<i>null</i>
2	My Property: Business Name2	<i>null</i>
3	Comp Set: Location4 #444444, Location5 #555555, Location 6 #66666...	<i>null</i>
4	Job Number: 37165322 Staff: User1 Created: May 15, 2023 Curr...	<i>null</i>

Рис. 14.18. Строки заголовка документа

При разработке стратегии извлечения этого идентификатора важно избегать ссылок на поля, поскольку согласованность между файлами не гарантируется. Использование жестко заданной ссылки может привести к ошибкам или возврату неправильных значений. Получение идентификатора включает два шага: извлечение всей строки из заголовка и извлечение последнего слова строки.

Кодируя логику, используем выражение записи, так как оно позволяет заглянуть внутрь и увидеть промежуточные результаты во время разработки и при устранении неполадок. Откройте расширенный редактор, создайте новую переменную за выражением *let*, назовите ее *GetPropertyID*. Инициализируйте выражение записи: []. Первому полю *PropertyString* присвоим выражение для получения второй строки таблицы в виде записи:

```
NoEmptyCols{1}
```

Преобразуем запись в список, обернув это выражение функцией *Record.ToList(...)*. Извлечем первый элемент списка, обернув функцией *List.First(...)*. Итого *PropertyString* вернет текст. Добавьте запятую в конце, чтобы создать новое поле для записи *GetPropertyID*. Назовем второе поле *PropertyID* и присвоим ему выражение для разделения строки:

```
Text.Split( PropertyString, " ") // вернет список
```

Извлечем последний элемент списка, обернув в *List.Last(...)*. Вернем значение через доступ к полю. Код записи *GetPropertyID*:

```
GetPropertyID
= [
  PropertyString = List.First(Record.ToList(NoEmptyCols{1})),
  PropertyID = List.Last(Text.Split(PropertyString, " "))
][PropertyID]
```

Значение *PropertyID* мы используем позднее в коде.

Удаление строк

После удаления пустых строк и столбцов данные, содержащиеся в верхнем и нижнем колонтитулах размещаются в первом столбце таблицы, а все ячейки, расположенные справа, пусты:

	ABC Column3	ABC Column4	ABC Column6	ABC Column7	ABC Column8
1	Daily by Date Performance Data - Preli...	null	null	null	null
2	My Property: Business Name2	null	null	null	null
3	Comp Set: Location4 #444444, Locatio...	null	null	null	null
4	Job Number: 37165322 Staff: User1 ...	null	null	null	null
5		null	Occupancy		null
6	Date	DOW	This Year		% Chg
7		null	My Prop	Comp Set	My Prop
8	05/22/2023	Mon	96,84210526	74,09675444	40,35976016
9	05/23/2023	Tue	97,89473684	78,19963258	35,86602871
10	05/24/2023	Wed	83,50877193	68,76913656	7,435442539
11	05/25/2023	Thu	83,15789474	70,9736681	2,93598862
12	05/26/2023	Fri	91,57894737	54,82529118	4,857894737
13	05/27/2023	Sat	98,59649123	73,96006656	11,77523015
14	05/28/2023	Sun	96,84210526	62,72878536	12,57280256
15	Partial May 2023		92,63157895	69,07904782	15,55519148
16	Period		92,63157895	69,07904782	15,55519148
17	2023 © Company X	null	null	null	null
18	Submit Data = Data is missing for the s...	null	null	null	null
19	DND = Data Not Displayed. Subject pr...	null	null	null	null
20	Subtotal / Total calculations inclusive ...	null	null	null	null
21	Report Selection: My Property vs. Pri...	null	null	null	null

Рис. 14.18а. Идентификация колонтитулов

Мы можем использовать знакомый шаблон для удаления этих строк, так как этот процесс схож с процессом *NoEmptyRows*, но мы оставим этот шаг без изменений для того, чтобы извлечь *PropertyID*. Скопируйте значение второго аргумента из шага *NoEmptyRows*:

```
each not List.IsEmpty(List.RemoveMatchingItems(Record.FieldValues(_), {"", null}))
```

На панели *Примененные шаги* перейдите к последнему шагу *NoEmptyCols*, и отфильтруйте по любому столбцу. Это сгенерирует новый шаг и базовый код. Замените второй аргумент на этом шаге кодом, скопированным из *NoEmptyRows*. Внутренний метод *Record.FieldValues()* возвращает список значений полей в том же порядке, что и порядок столбцов в таблице. Чтобы опустить значения из первого столбца, оберните это выражение функцией *List.Skip(...)*. Переименуйте шаг в *RemoveRows*. Код этого шага:

```
Table.SelectRows(
  NoEmptyCols,
  each not List.IsEmpty(List.RemoveMatchingItems(List.Skip(Record.FieldValues(_), {"", null}))
)
```

В самом низу останутся две лишние строки. Пройдите *Главная* → *Удалить строки* → *Удалить нижние строки* → 2, чтобы убрать их, и переименуйте шаг в *GetDataRange*.

При объединении файлов для обеспечения точного размещения данных в столбцах необходимо, чтобы у них были совпадающие имена.

Новые заголовки

Первые три строки промежуточной таблицы содержат заголовки:

	ABC Column3	ABC Column4	ABC Column6	ABC Column7	ABC Column8	ABC Column9
1	null	null	Occupancy	null	null	
2	Date	DOW	This Year		% Chg	
3	null	null	My Prop	Comp Set	My Prop	Comp Set
4	05/22/2023	Mon	96,84210526	74,09675444	40,35976016	53,31016
5	05/23/2023	Tue	97,89473684	78,19963258	35,86602871	31,66181
6	05/24/2023	Wed	83,50877193	68,76913656	7,435442539	10,38537

Рис. 14.19. Верхняя часть промежуточной таблицы

В Excel заголовки часто размещают в объединенных ячейках. При импорте в Power Query одна ячейка будет содержать значение, а другие будут пустыми. Существует пятишаговый шаблон для решения этой проблемы: транспонировать таблицу, заполнить вниз, объединить столбцы, транспонировать обратно, повисить заголовки. Все эти шаги можно выполнить через интерфейс без написания кода.

Мы представим альтернативный метод. Для начала необходимо разделить таблицу и преобразовать строки заголовка отдельно от строк данных. В строках заголовка мы заполним столбцы вправо с помощью трюка, описанного в [главе 8 Работа с вложенными структурами](#).

Пройдите Главная → Сохранить строки → Сохранить верхние строки → 3. В строке формул отразится код:

```
= Table.FirstN(GetDataRange,3)
```

Поместите курсор после знака равно и введите `Table.TransformRows(`. Поместите курсор в конец, введите запятую, и второй аргумент `each Record.ToTable(_)`. Добавьте закрывающую скобку, чтобы посмотреть на промежуточные результаты:

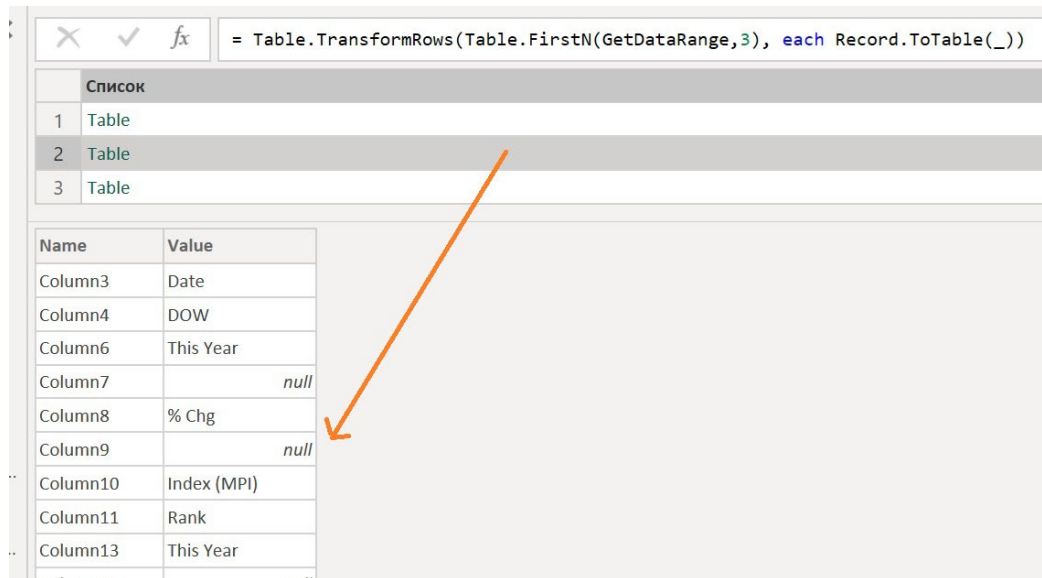


Рис. 14.19а. Трансформация трех строк в таблицы с двумя столбцами *Name* и *Value*

Чтобы заполнить значения в столбце *Value*, поместите выражение `Record.ToTable(_)` внутрь функции `Table.FillDown(..., {"Value"})`. Извлеките столбец *Value* в виде списка, добавив `[Value]` после закрывающей скобки функции `Table.FillDown` и перед закрывающей скобкой функции `Table.TransformRows`. При этом возвращается список с вложенным списком для каждой строки:

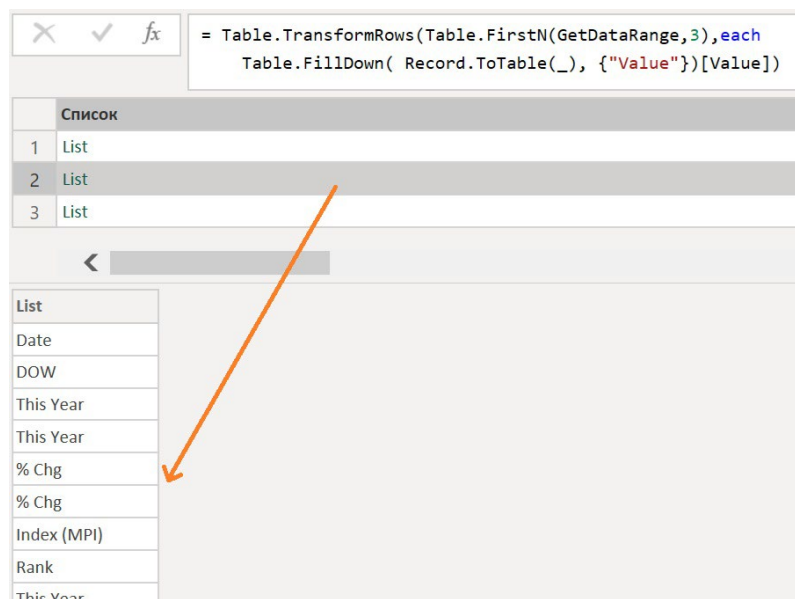


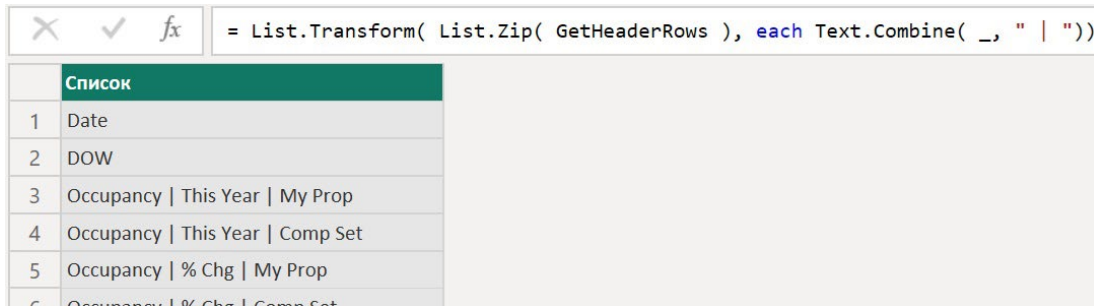
Рис. 14.20. `GetHeaderRows` создаст список списков

Переименуйте шаг в `GetHeaderRows`. Поместите шаг первым, сразу после `let`. Его код:

```
Table.TransformRows(
  Table.FirstN(GetDataRange, 3),
  each Table.FillDown(Record.ToTable(_), {"Value"})[Value]
```

)

Сохраняя модульный подход, сейчас самое время выделить логику в новый шаг. Вставьте ручной шаг, нажав f_x . Значения нужно комбинировать по их позиции: `List.Zip(GetHeaderRows)`. Затем все элементы списка нужно преобразовать в одно текстовое значение для каждого столбца: `List.Transform(..., each Text.Combine(_, " | ")`):



Список	
1	Date
2	DOW
3	Occupancy This Year My Prop
4	Occupancy This Year Comp Set
5	Occupancy % Chg My Prop
6	Occupancy % Chg Comp Set

Рис. 14.21. Список заголовков, показывающий первые пять элементов

Переименуйте шаг в *Headers*:

```
List.Transform(  
  List.Zip(GetHeaderRows),  
  each Text.Combine( _, " | " )  
)
```

Пришло время получить основные данные и назначить столбцам извлеченные заголовки. Перейдите к последнему шагу запроса – *GetDataRange*. Нажмите f_x . Пропустите три верхние строки: `Table.Skip(GetDataRange, 3)`. Переименуйте столбцы, используя `List.Zip` для создания списков имен:

```
Table.RenameColumns( ..., List.Zip({Table.ColumnNames(GetDataRange), Headers} ) )
```

Переименуйте шаг в *GetTable*. Код шага:

```
Table.RenameColumns(  
  Table.Skip(GetDataRange, 3),  
  List.Zip({Table.ColumnNames(GetDataRange), Headers})  
)
```

Типы данных

Рекомендуется задавать типы столбцов как можно позже. Установка типа данных столбца с помощью интерфейса преобразует в тип, допускающий значение *null*, и вызывает функцию `Table.TransformColumnTypes`.

Для примера зададим тип столбца *Date*. Нажмите ABC123 слева от названия столбца, выберите *Используя локаль*. В открывшемся окне выберите *Тип данных = Дата* и *Языковой стандарт = Английский (США)*. Это создает следующий синтаксис:

```
Table.TransformColumnTypes(GetTable, {"Date", type date}, "en-US")
```

Однако согласно спецификации ожидаемого файла результата, все столбцы должны быть преобразованы в текст. Хотя можно спорить о целесообразности такого типа данных, стоит отметить, что текстовые значения легко преобразовать в любой примитивный тип данных – это безопасный выбор. Выберите все столбцы и преобразуйте их в *Текст*.

Изучите выражение в строке формул. Важно понимать, что мы имеем дело с вложенной структурой – список списков, формат которого { `columnName, typeName` }. Для каждого столбца, для которого необходимо установить тип, должен быть свой список. Этот шаблон можно сделать динамичным и более кратким:

- в строке формул удалите шаги, связанные с изменением типов;
- нажмите f_x , введите `List.Transform(`.
- передайте функции список *Headers* в качестве первого аргумента (этот список содержит все имена столбцов);

- преобразуйте каждое имя столбца в формат { columnName, typeName }, например, так: each {_, type text}; не забудьте ввести закрывающую скобку функции.

Переименуйте шаг в *SetColTypes*. Должно получиться:

```
Table.TransformColumnTypes(
  GetTable,
  List.Transform( Headers, each {_, type text})
)
```

Теперь каждому столбцу присвоен тип данных *Текст*, что отображается значком *ABC* перед именем.

Добавление поля

На последнем шаге преобразования добавим отдельным столбцом идентификатор свойства, *GetPropertyID*. Пройдите *Добавление столбца* → *Настраиваемый столбец*. Введите имя столбца *Property ID*. В разделе формул введите *=GetPropertyID*. Нажмите *OK*. Переименуйте шаг в *InsertPropertyID*. В строке формул присвойте тип этому столбцу, поставив запятую перед закрывающей скобкой, за которой введите *type text*. Код шага:

```
Table.AddColumn(SetColTypes, "Property ID", each GetPropertyID, type text)
```

Пришло время подумать, стоит ли и как бороться с пустыми файлами и ошибками.

Пустой файл

Если не предусмотреть обработку пустых файлов, запрос завершится ошибкой. Когда критерии идентификации данных на шаге *RAW* не выполняются, возвращается пустая таблица. Это можно проверить и изменить вывод запроса. Вот как это сделать. Нажмите *f_x*. Введите выражение:

```
if Table.IsEmpty( RAW ) then "File is empty." Else InsertPropertyID
```

Переименуйте шаг в *Result*.

Условный оператор действует как переключатель, определяя, следует ли выводить текстовую строку или табличное значение. Сохранение этого шага в качестве последнего шага запроса гарантирует, что последовательность операций преобразования останется видимой и доступной в области *Примененные шаги*.

Нет никаких технических требований для сохранения переменной *Result* в качестве последнего шага запроса. Это не влияет на работу кода. Тем не менее, такое размещение делает взаимодействие с интерфейсом прозрачным, облегчая доработку запроса и исправление ошибок. Рекомендуем отразить это в комментарии. Щелкните правой кнопкой мыши на имени шага *Result*, выберите *Свойства* и введите Описание. Нажмите *OK*. После этого в разделе *Примененные шаги* рядом с шагом *Result* появится знак с информацией.

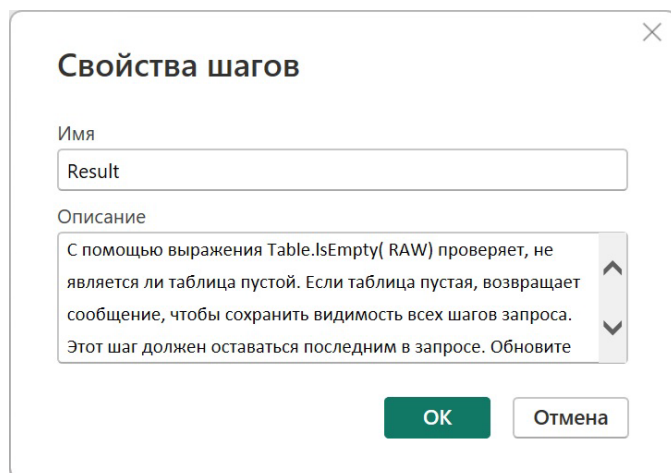


Рис. 14.23. Вставка комментария к шагу

Запрос *TransformData* содержит всю логику преобразования данных. Он был разработан и применен к файлу образца. Чтобы обработать все файлы, его нужно преобразовать в функцию, а затем вызвать для всех выбранных файлов в папке.

Преобразования запроса в функцию

Запрос в функцию можно преобразовать с помощью интерфейса. Важно иметь модульную структуру, так как интерфейс запросит параметр для каждого аргумента, передаваемого функции. Наша функция будет с одним параметром, в качестве которого уже настроен *BinaryFile*.

В области Запросы щелкните правой кнопкой мыши *TransformData* и выберите *Создать функцию*. В диалоговом окне введите имя функции *CollectData*. Нажмите *OK*.

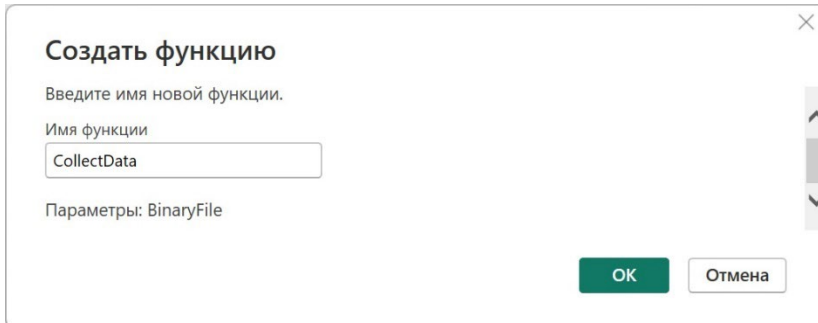


Рис. 14.24. Диалоговое окно Создать функцию

Это действие сгенерирует запрос *fxCollectData* и поместит параметр *BinaryFile*, запрос *TransformData* и функцию *fxCollectData* в папку с именем *fxCollectData*.

Организируйте запросы, переместив *FolderLocation* и *BinarySample* в папку *fxCollectData*. Это можно сделать, перетащив их или щелкнув правой кнопкой мыши и выбрав *Переместить в группу*.

Вернитесь к *Запрос1*, переименуйте его в *CombinedData*. Сохраните столбцы *Name* и *Content*. Это можно сделать через интерфейс. Выберите столбцы *Name* и *Content* и пройдите *Главная* → *Удалить столбцы* → *Удалить другие столбцы*. Или, находясь на шаге *SelectFiles* применить проекцию, добавив *[[Name], [Content]]* сразу за закрывающей скобкой. Для определенности мы будем считать, что использовали проекцию.

Существует несколько методов преобразования структурированных значений в таблице. Они были рассмотрены в [главе 8 Работа с вложенными структурами](#). Мы используем функцию *Table.TransformColumns*. Нажмите *fx*. Введите функцию между знаком равенства и именем переменной: *Table.TransformColumns{*. В качестве второго аргумента функция ожидает список операций преобразования. Введите *{"Content", each fxCollectData(_)}*. Добавьте закрывающую скобку и переименуйте шаг в *InvokedFunction*:

`Table.TransformColumns(SelectFiles, {"Content", each fxCollectData(_)})`

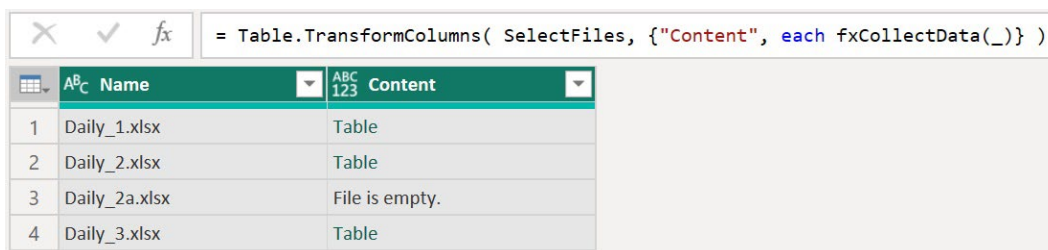


Рис. 14.25. Содержимое столбца *Content* после вызова *fxCollectData*

После преобразования значений в столбце *Content* обнаруживается пустой файл. Нужно реализовать процесс мониторинга, когда запрос *TransformData* обращается к пустому файлу.

Настройка мониторинга

Настроить мониторинг для получения уведомлений о пустых файлах в сервисе Power BI довольно просто. Создайте дубль запроса *CombinedData* и переименуйте его в *EmptyFiles*. Затем примените фильтр в столбце *Content*, чтобы отсечь строки, содержащие сообщение "File is empty.":

`Table.SelectRows(InvokedFunction, each [Content] = "File is empty.")`

Так как запрос *EmptyFiles* предназначен для облегчения мониторинга, полезно включить путь к файлу в результат. Выберите шаг *Source*. Можно увидеть столбец, содержащий путь – *Folder Path*. Теперь выберите шаг *SelectFiles*. Здесь мы применили проекцию для ограничения столбцов. Чтобы включить столбец *Folder Path* и поместить его первым в результирующей таблице, добавьте его в проекцию:

```
[[Folder Path], [Name], [Content]]
```

Запрос *EmptyFiles* можно загрузить в модель данных как отдельную скрытую таблицу. Для включения уведомления создайте меру DAX, например:

```
EmptyFileCount_ModelName = COUNTROWS(EmptyFiles)
```

Эту меру можно поместить в визуальный элемент, например карточку, чтобы установить пороговое значение в сервисе Power BI. Как только пороговое значение будет превышено, вы получите автоматическое уведомление для принятия решения.

Тонкая настройка

Вернитесь к запросу *CombinedData*. Пока переключатель пустого файла применяется к запросу *TransformData*, строки, соответствующие этому критерию, должны быть исключены. Независимо от того, удовлетворяют ли какие-либо строки этому условию в данный момент, назовите этот шаг *NoEmptyFiles*:

```
Table.SelectRows(InvokedFunction, each [Content] <> "File is empty.")
```

Если шаг *NoEmptyFiles* будет опущен, при обнаружении пустого файла возникнет ошибка. Кроме того, при пакетной обработке файлов со временем могут возникать другие проблемы, которые также могут привести к ошибке. Превентивное внедрение стратегии для решения этой ситуации может быть оправдано. В [главе 12 Обработка ошибок и отладка](#) описан метод захвата ошибок для отчетности, который позволяет реализовать аналогичный, но отдельный рабочий процесс для управления пустыми файлами.

Когда будут выявлены файлы, вызывающие сбои, вероятно, потребуется дополнительная логика преобразования для устранения основной проблемы. Вот как будет выглядеть процесс модификации или обновления кода пользовательской функции *fxCollectData*:

- Начните с определения файла, который вызывает ошибку во время обработки.
- Перейдите к запросу *BinarySample* и выберите шаг *SelectFiles*.
- Разверните столбец *Content* для выявленного файла.
- Удалите шаг *Imported Excel workbook*, если он был сгенерирован автоматически, чтобы отобразить выбранный файл как файл в панели предварительного просмотра.
- Перейдите к запросу *CollectData* и определите шаг, на котором возникают ошибки.
- Внедрите дополнительную логику или обновите код, чтобы устранить проблему.
- Выберите запрос *CombinedData* и убедитесь, что проблем больше нет. Если ошибки продолжают возникать, повторите эти шаги, пока все проблемы не будут устранены.

Ни при каких обстоятельствах не следует напрямую модифицировать запрос *fxCollectData*; этот запрос связан и отражает любые изменения, внесенных в *CollectData*. Однако эта связь будет разорвана, как только *fxCollectData* будет открыт в расширенном редакторе. К счастью, пользователь получит предупреждение и сможет отменить вход в расширенный редактор. Если же предупреждение не появится, что указывает на уже потерянную связь, любые изменения в *CollectData* не будут синхронизированы с *fxCollectData*. В таких случаях, чтобы восстановить связь функции с запросом, необходимо создать новую функцию на основе запроса *CollectData* и обновить шаг *InvokedFunction*.

Без включения отчетности об ошибках вы можете оставить ошибки, чтобы они вызывали сбои, сигнализируя о наличии проблемы, которую нужно решить. Сейчас мы предполагаем, что отчетность об ошибках уже реализована, и можно безопасно исключить все строки с ошибками. Пройдите *Главная* → *Удалить строки* → *Удалить ошибки*, переименуйте шаг в *RemoveErrors*.

Теперь можно объединить содержимое всех файлов.

Объединение файлов

Нажмите *развернуть столбец* в заголовке столбца *Content*. Отключите опцию *Использовать исходное имя столбца как префикс* внизу окна. Нажмите *ОК*. Это добавит новый шаг в запрос. В строке формул вы увидите, что функция *Table.ExpandTableColumn* получила два жестко закодированных списка. Первый список содержит *columnNames* для развертывания из каждой вложенной таблицы, а второй список, *newColumnNames*, назначает имена новым столбцам. Это помогает избежать конфликтов между именами существующих и новых столбцов. Однако риск того, что файл неожиданно будет включать поля с именами *Name* или *Content*, крайне низок, поэтому финальный список можно удалить. Что касается первого списка, его можно заменить функцией, которая динамически предоставляет все имена столбцов из запроса *CollectData*. После этих изменений выражение в строке формул будет выглядеть так:

```
Table.ExpandTableColumn(RemoveErrors, "Content", Table.ColumnNames(CollectData))
```

Переименуйте шаг в *ExpandAllColumns*.

Успешно объединив данные из всех трех файлов, каждый из которых соответствует неделе данных, мы получили одну полную таблицу, состоящую из 21 строки и 22 столбцов. Все значения отформатированы как текст, аналогично файлу *ExpectedCombined.xlsx*. Скомпилировав и структурировав все необходимые данные в соответствии с заданными спецификациями, мы готовы перейти к фазе преобразований для аналитики.

Резюме

В этой главе мы осветили практические аспекты работы с данными и их подготовки, сосредоточившись на двух совершенно разных темах: извлечении фиксированных шаблонов и объединении данных из нескольких файлов Excel. Несмотря на то, что эти навыки кажутся разными, они необходимы для преобразования необработанных данных в полезную информацию.

Разбивая процессы на управляемые части, подчеркивая важность методичного подхода, а также творческого решения проблем, мы стремились предложить коллекцию идей, подпитывающих ваше воображение для решения любых задач. Следующая глава посвящена еще одной важной области – оптимизации производительности. Вы узнаете о факторах, влияющих на производительность и методах ее улучшения.