Язык M Power Query. Метаданные

Power Query позволяет прикрепить к значению сведения о нём. Обычно эти описательные фрагменты невидимы. Их присутствие не меняет поведение движка M. Однако код (ваш или чужой, например, функция стандартной библиотеки) может считывать и задавать эти нотации. Они известны как *метаданные.* Их можно использовать для передачи дополнительной информации, относящейся к значению. Метаданные применяются в информационных или диагностических целях. Они также могут быть задействованы в выражениях, поскольку метаданные определяют поведение значений.[1]

Предыдущая заметка   Следующая заметка

Возьмем, например, параметры. Они обрабатываются особым образом в пользовательском интерфейсе Microsoft Excel (и Power BI), но в самом Power Query они хранятся как обычные значения.

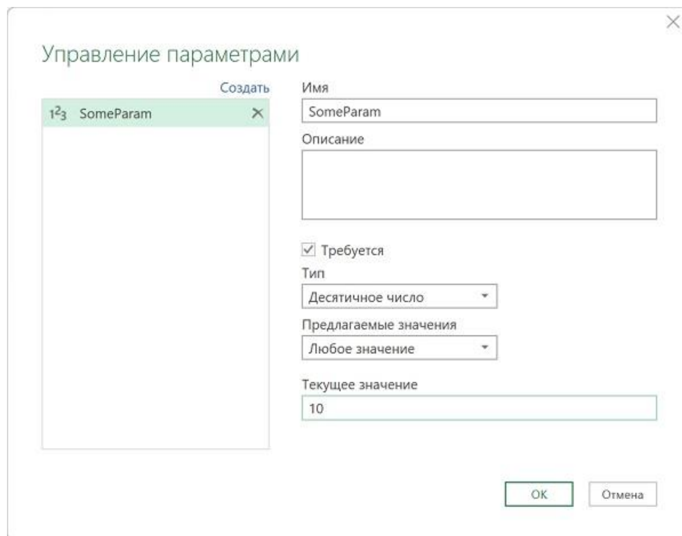

Рис. 1. Параметры в Microsoft Excel

Это хорошо. Вам не нужно делать ничего особенного, чтобы использовать параметр. Это – просто значение для вашего кода. Тем не менее, параметры являются особенными, но где скрывается эта «особенность» (например, их настройки)? С точки зрения PQ параметры хранятся в метаданных.

Начнем с основ....

Все значения имеют метаданные. Метаданные хранятся в обычной записи. По умолчанию запись метаданных пуста.

Вид метаданных

Чтобы увидеть метаданные, используйте Value.Metadata():

Запрос 1

```
let
   SomeValue = 1
in
   Value.Metadata(SomeValue) // возвращает пустую запись: []
```

Сначала мы определили переменную SomeValue, а затем считали метаданные. По умолчанию запись метаданных является пустой.

Замена (установка) метаданных

Эту неинтересную пустую запись можно заменить более интересной записью с помощью Value.ReplaceMetadata():

Запрос 2

```
let
```

[1] Заметка написана на основе статьи Ben Gribaudo. Power Query M Primer (Part 20): Metadata. Если вы впервые сталкиваетесь с Power Query, рекомендую начать с Марк Мур. Power Query.

```
    SomeValue = 1,
    MetadataReplaced = Value.ReplaceMetadata(SomeValue, [Exciting = true])
in
    Value.Metadata(MetadataReplaced) // [Exciting = true]
```

Технически, как мы знаем, значения в M неизменяемы, поэтому Value.ReplaceMetadata на самом деле не изменяет текущее значение для обновления его метаданных. Вместо этого функция возвращает новое значение, которое совпадает с предыдущим, только с удаленными предыдущими метаданными и новой записью, установленной в качестве метаданных значения SomeValue.

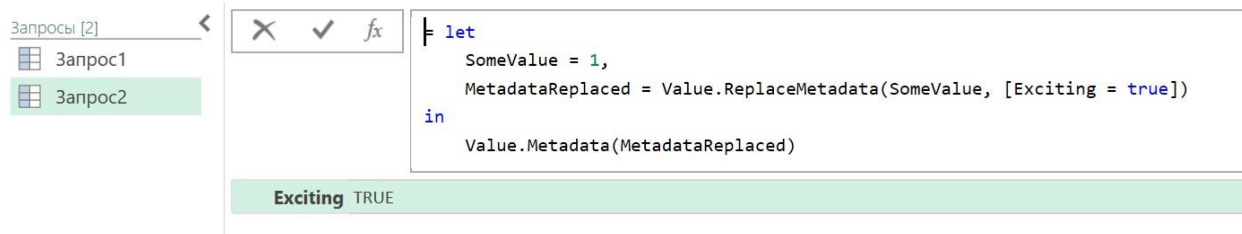

Рис. 2. Результат Запроса 2

Метаданные прикрепляются к значению, а не к переменной.

Запрос 3

```
let
    SomeValue = 1,
    MetadataReplaced = Value.ReplaceMetadata(SomeValue, [Exciting = true]),
    YetAnotherVariable = MetadataReplaced
in
    Value.Metadata(YetAnotherVariable) // [Exciting = true]
    //Value.Metadata(MetadataReplaced) также возвращает [Exciting = true]
```

Value.Metadata(YetAnotherVariable) возвращает те же метаданные, что и Value.Metadata(MetadataReplaced). Обе переменные (YetAnotherVariable и MetadataReplaced) ссылаются на одно и то же значение, а метаданные находятся на значении, а не на переменной.

Слияние метаданных

Вместо того, чтобы заменять метаданные новой записью, можно объединить новую запись метаданных с существующими записями метаданных. Это можно сделать, получив текущую запись метаданных Value.Metadata(Initial), добавить к ней новую запись & [Important = true], а затем передав выходные данные слияния в Value.ReplaceMetadata:

Запрос 4

```
let
    Initial = Value.ReplaceMetadata(1, [Exciting = true]),
    Final = Value.ReplaceMetadata(Initial, Value.Metadata(Initial) & [Important = true])
in
    Value.Metadata(Final) // [Exciting = true, Important = true]
```

Тот же эффект может быть достигнут с более простым синтаксисом благодаря оператору *meta* Power Query:

Запрос 5

```
let
    Initial = Value.ReplaceMetadata(1, [Exciting = true]),
    Final = Initial meta [Important = true]
in
    Value.Metadata(Final) // [Exciting = true, Important = true]
```

При объединении двух записей, если в обеих записях присутствует одно и то же имя поля, значение этого поля из объединяемой записи переопределяет значение исходной записи. При

объединении [A = 1] и [A = 11] получается [A = 11], так как значение A из объединенной записи переопределяет значение A из исходной записи.

Оператор *meta* фактически выполняет слияние записей, поэтому то же самое относится и к нему. Ниже значение *Important* присутствует как в метаданных *Initial*, так и в новой записи, предоставленной оператору *meta* в переменной *Final*. Значение этого поля в последней записи переопределяет значение, указанное в поле *Initial*. Таким образом, слияние переключает *Important* с *true* на *false*.

Запрос 6

```
let
   Initial = Value.ReplaceMetadata(1, [Exciting = true, Important = true]),
   Final = Initial meta [Important = false]
in
   Value.Metadata(Final) // [Exciting = true, Important = false]
```

Удаление метаданных

В мире Power Query удаление метаданных приравнивается к замене их пустой записью. Вы можете сделать это вручную, используя формулу Value.ReplaceMetadata(SomeValue, []). Но вы можете использовать функцию Value.RemoveMetadata() с методом, созданным специально для этой цели:

Запрос 7

```
let
   Initial = Value.ReplaceMetadata(1, [Exciting = true]),
   Final = Value.RemoveMetadata(Initial)
in
   Value.Metadata(Final) // возвращает пустую запись []
```

Продолжительность жизни метаданных

Если значение с метаданными передается оператору в качестве аргумента, выходные данные получат метаданные по умолчанию, т.е., пустую запись. Допустим, вы складываете два значения, каждое из которых содержит метаданные. Сумма не будет иметь метаданных. Операторы M, за исключением оператора *meta*, не переносят метаданные с входа на выход.

Запрос 8

```
let
   Original = 1 meta [Important = true],
   Result = Original * .5
in
   Value.Metadata(Result) // возвращает пустую запись: []
```

Если вы хотите, чтобы входные метаданные были добавлены к выходным данным оператора, вам нужно написать логику. В этом есть смысл. Возьмем пример, подобный приведенному выше: как гибридный движок должен знать, делает ли входной сигнал *Important* вывод оператора умножения также *Important*? Это не так. Вы сами устанавливаете метаданные на выходе, основываясь на ваших правилах.

Встроенные специальные метаданные?

Возможно, вы ожидаете список имен полей метаданных, встроенных в язык, которые делают особые вещи — например, если вы устанавливаете метаданные значения в SecretCode = 123, а затем оцениваете выражение, движок готовит для вас чай или косит ваш газон — или, по крайней мере, выполняет в два раза быстрее или что-то в этом роде. На языковом уровне полей метаданных нет. Язык просто обеспечивает поддержку метаданных, но не придает им особого значения.

Это означает, что сам Power Query не изменяет поведение в зависимости от наличия или отсутствия метаданных. Любое специальное поведение метаданных исходит из логики, использующей возможности метаданных, предлагаемые языком, а не из самого языка. Это включает в себя логику в хост-среде (Microsoft Power BI, Microsoft Excel и т. д.) и в стандартной библиотеке, а также в написанных вами выражениях.

Параметры приложения

Вернемся к рис. 1. Excel использует метаданные для настройки параметров. Для Power Query (язык, включая движок) эти элементы являются просто произвольными метаданными. Их особое значение возникает только потому, что приложение придает им это значение.

В Excel пройдите *Данные –> Получить данные –> Из других источников –> Пустой запрос*. В редакторе Power Query пройдите *Главная –> Управление параметрами –> Создать параметр*. Создайте параметр с именем *SomeParam* и присвойте ему значение 10. Изучите метаданные на нем:

Запрос 10

Value.Metadata(SomeParam)

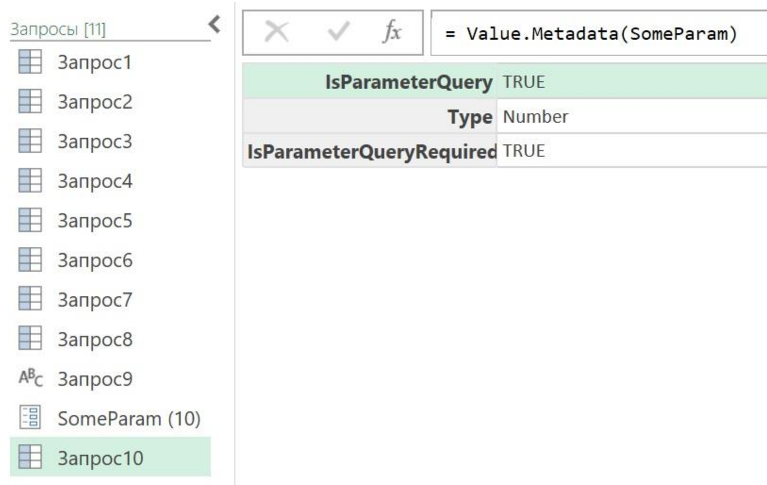

Рис. 3. Метаданные параметра SomeParam

Обратите внимание, как PQ сохранил метаданные вместе со значением 10. Для движка эти метаданные непрозрачны: поля и значения не имеют внутреннего значения, которое их попросили сохранить. Для Excel эти сведения сообщают, что значение является параметром, а также говорят, как этот параметр настроен.

Рассмотрим метаданные параметров подробнее. В редакторе запросов выберите SomeParam и откройте расширенный редактор. Теперь вы можете увидеть, где определены метаданные параметра. (При желании вы даже можете отредактировать их вручную.)

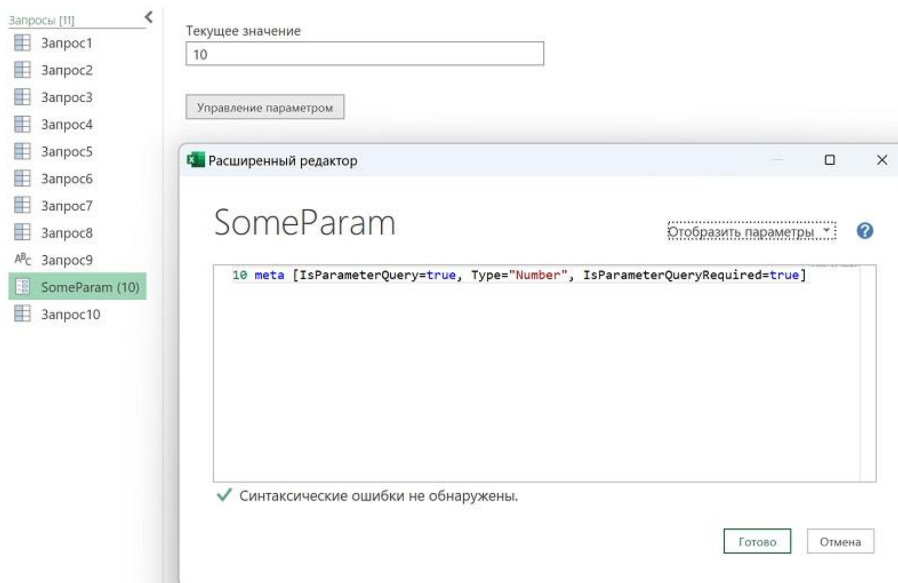

Рис. 4. Код, сохраняющий метаданные параметра

Создайте дубль запроса *SomeParam (10)*, скопируйте фрагмент *meta [...]* в буфер обмена, удалите фрагмент из выражения, нажмите *Готово*.

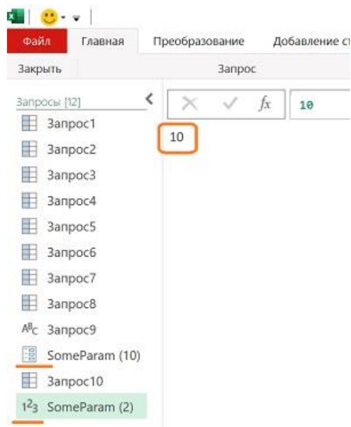

Рис. 5. Без метаданных параметр стал обычным значением (числом)

SomeParam больше не ведет себя и не выглядит как параметр в пользовательском интерфейсе (например, значок параметра больше не отображается рядом с его именем). Он больше не является параметром, так как метаданные, которые предписывали Excel обрабатывать его как таковой, исчезли. Это по-прежнему значение, но не значение, которое рассматривается приложением как параметр.

Теперь создайте новый пустой запрос, назовите его *SomeValue* и введите единственную строку со значением 10. Откройте Расширенный редактор и вставьте фрагмент *meta [...]* из буфера обмена. Нажмите *Готово*. Обратите внимание, пользовательский интерфейс редактора Power Query обрабатывает *SomeValue* как параметр! Имейте в виду, что параметры — это концепция приложения — для Power Query *SomeValue* по-прежнему является обычным значением, с некоторыми непрозрачными метаданными.

Игра с метаданными параметров может быть забавным развлечением. Хотя и не особенно полезным. Большинство людей будут использовать интерфейс для создания параметров (вместо того, чтобы вручную создавать их в коде), и, вероятно, есть несколько практических применений, которые вы найдете для программного чтения метаданных параметров. На самом деле, даже если вы его найдете, вам нужно быть осторожным, так как метаданные параметров — это внутренняя деталь реализации Excel (и Power BI), а она может измениться в будущем без предварительного уведомления. Но теперь, по крайней мере, вы видели реальный пример того, где и как используются метаданные.

Функциональная документация

Вы когда-нибудь замечали, как по-разному редактор Power Query отображает документацию стандартных библиотечных и пользовательских функций? Давайте определим функцию, а затем сошлёмся на неё. Не вызовем, а просто укажем её имя без скобок.

Определим *SomeFunction*:

```
let
   DoSomething = (input as number) as number => input * 10
in
   DoSomething
```

Затем в новом запросе сошлёмся на неё:

Запрос 11

```
SomeFunction
```

= SomeFunction

Введите параметр

input

Пример: 123

Вызвать Очистить

function (input as number) as number

Рис. 6. Лаконичная сигнатура пользовательской функции

Редактор запросов отображает сигнатуру функции и пользовательский интерфейс, позволяющий указать значение параметра и вызвать функцию. Коротко и по делу))

Для контраста посмотрим на функцию из стандартной библиотеки, скажем, Number.Sqrt:

= Number.Sqrt

Number.Sqrt

Возвращает квадратный корень числа number. Если number имеет значение NULL, Number.Sqrt возвращает NULL. Если это отрицательное значение, возвращается значение Number.NaN (нечисловое).

Введите параметр

number (необязательно)

Пример: 123

Вызвать Очистить

function (number as nullable number) as nullable number

Пример: Найти квадратный корень числа 625.

Использование:

```
Number.Sqrt(625)
```

Выходные данные:

```
25
```

Пример: Найти квадратный корень числа 85.

Использование:

```
Number.Sqrt(85)
```

Выходные данные:

```
9.2195444572928871
```

Рис. 7. Описание стандартной библиотечной функции

Какая разница! Помимо описания функции, приведено несколько примеров использования. Как Microsoft удалось это сделать? Документация по функциям определяется с использованием метаданных. Но ведь это означает, что можно использовать метаданные для документирования пользовательских функций!

На языковом уровне — для движка — элементы метаданных документации, которые мы собираемся рассмотреть, являются просто непрозрачными значениями метаданных, не имеющими внутреннего смысла. Это редактор запросов (приложение) придает им смысл. Существует набор задокументированных имен полей метаданных, которые, если они присутствуют в функции, будут использоваться редактором запросов, чтобы влиять на то, что отображает его пользовательский интерфейс.

То, что мы собираемся изучить, является одной из основных мотиваций для ручного определения и приписывания пользовательских типов.

Давайте определим и припишем тип функции *SomeFunction*. Этот тип должен совпадать с типом функции (иметь одинаковую сигнатуру функции). Мы также свяжем несколько полей метаданных с новым типом.

Запрос 13

```
let
   DoSomething = (input as number) as number => input * 10,
   Type = type function (input as number) as number
      meta [
         Documentation.Name = "SomeFunction",
         Documentation.LongDescription = "Some very long and elegant description.",
         Documentation.Examples = {
            [Description = "Positive number as input", Code = "SomeFuction(1)", Result = "10"],
            [Description = "Negative number as input", Code = "SomeFuction(-2)", Result = "-20"]
         }
      ],
   RetypedFunction = Value.ReplaceType(DoSomething, Type)
in
   RetypedFunction
```

Теперь взгляните на то, как редактор запросов отображает нашу функцию в своем пользовательском интерфейсе. Намного интереснее!

fx = Value.ReplaceType(DoSomething, Type)

SomeFunction

Some very long and elegant description.

Введите параметр

input

Пример: 123

Вызвать Очистить

function (input as number) as number

Пример: Positive number as input

Использование:

SomeFuction(1)

Выходные данные:

10

Пример: Negative number as input

Использование:

SomeFuction(-2)

Выходные данные:

-20

Рис. 8. Отображение в редакторе запросов документации по функции, созданной на основе метаданных

Documentation.Name – это значение, которое должно совпадать с именем идентификатора функции при ее определении. Почему? Потребители вашей функции могут присвоить ее другим идентификаторам (например, ReallyCoolFunction = SomeFunction). Если Documentation.Name совпадет с именем, которое вы изначально ему дали, это поможет им понять, с какой функцией они в действительности работают. Например, когда кто-то просматривает документацию по ReallyCoolFunction, он сможет увидеть, что он работает с SomeFunction.

Если вы пропустите указание *Documentation.Name*, пользовательский интерфейс редактора запросов будет игнорировать другие элементы метаданных документации на уровне функций. Если, например, задать только *Documentation.LongDescription*, выходные данные документации редактора запросов не будут отображаться, так как *Documentation.Name* отсутствует.

Почему мы просто не добавили метаданные в функцию, когда определили ее изначально. Вместо этого мы сначала определили функцию, затем определили пользовательский тип для функции, а затем приписали этот тип функции. Почему бы просто не применить метаданные к функции и не делать лишнюю работу!? Почему мы не сделали что-то типа:

((input as number) as number => input * 10) meta [… documentation goes here …]

Это не работает. Метаданные документации должны быть привязаны к типу функции, а не к самой функции.

Документация по параметрам

Приведенные выше метаданные документации применяются на уровне функций. Редактор запросов также поддерживает метаданные документации для отдельных параметров.

Запрос 14

```
let
  DoSomething = (input as number) as number => input * 10,
  Type =
    type function (
      input as (type number meta
        [
          Documentation.FieldCaption = "Number to multiply",
          Documentation.FieldDescription = "Will be multiplied by 10",
          Documentation.SampleValues = { 1, 0, -10 }
        ]
      )
    ) as number,
  RetypedFunction = Value.ReplaceType(DoSomething, Type)
in
  RetypedFunction
```

Интересно, что в редакторе запросов, хотя заголовок поля отображается, его описание не отображается, и отображается только первое из приведенных значений.

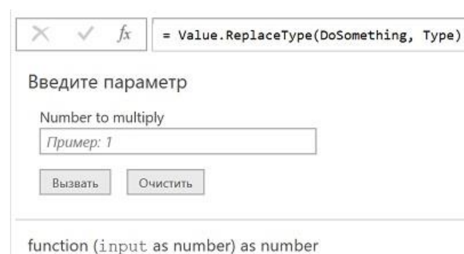

Рис. 9. Документация по параметру

Существует еще несколько опций метаданных для использования в документации на уровне параметров:

- Documentation.AllowedValues — ограничивает пользовательский интерфейс вызова функции входными данными из раскрывающегося списка *AllowedValues*; при вызове

функции из другого запроса, можно передать в неё любой параметр, не обязательно из списка *AllowedValues.*

Запрос 15

```
let
  DoSomething = (input as number) as number => input * 10,
  Type =
    type function (
      input as (type number meta
        [
          Documentation.FieldCaption = "Number to multiply",
          Documentation.FieldDescription = "Will be multiplied by 10",
          Documentation.AllowedValues = { 1, 0, -10 }
        ]
      )
    ) as number,
  RetypedFunction = Value.ReplaceType(DoSomething, Type)
in
  RetypedFunction
```

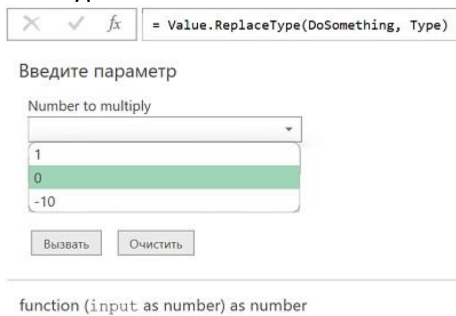

Рис. 10. Ограничение на ввод входных данных

- Documentation.FieldDescription — интересно, что значение этого поля, похоже, не отображается в пользовательском интерфейсе редактора запросов.
- Formatting.IsMultiLine — переключает элемент управления вводом параметра в многострочное текстовое поле.
- Formatting.IsCode — предполагается, что элемент управления вводом параметра будет отформатирован как код, но на момент написания этой статьи он также ни на что не влияет в редакторе запросов.

Параметры записи

Для параметров записи ожидаемая «форма» (то есть обязательные и разрешенные поля) может быть описана путем присвоения параметру соответствующего пользовательского типа записи в типе, приписываемом функции. Элементы метаданных Documentation.FieldCaption, Documentation.AllowedValues и Documentation.SampleValues можно дополнительно применить к полям типа записи, чтобы повлиять на то, как редактор запросов отображает их в пользовательском интерфейсе вызова функции.

Запрос 16

```
let
  Func = (input as record) => ...,
  NewType = type function (
    input as
     [
      Server = (
       type text meta
       [
         Documentation.SampleValues = { "localhost"}
       ]
      ),
      optional Timeout = (
```

```
        type number meta
        [
          Documentation.FieldCaption = "Time Out [in seconds]",
          Documentation.AllowedValues = { 10, 60, 360 }
        ]
      ),
      optional CutOff = date
    ]
  ) as any,
  Ascribed = Value.ReplaceType(Func, NewType)
in
  Ascribed
```

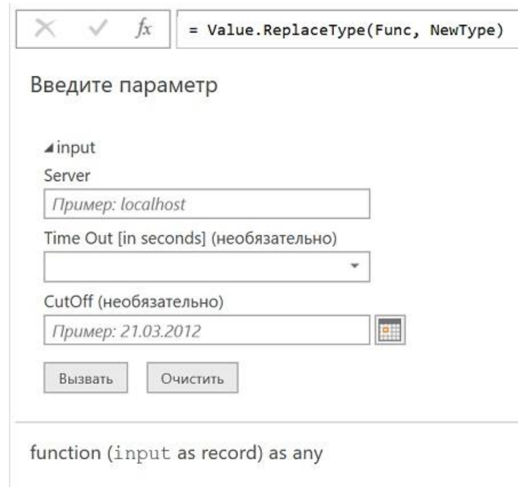

Рис. 11. Диалоговое окно вызова функции, отображающее параметр записи с подробными сведениями о поле, дополненными метаданными

Где всё же могут пригодиться метаданные!?

Мы рассмотрели, как Excel (Power BI) использует метаданные для параметров (интересно, но не слишком практично) и документации (настройка полей метаданных, которые ищет пользовательский интерфейс редактора запросов). Тем не менее, оба эти применения связаны с приложением, которое обеспечивает особое поведение. Есть ли практическое применение для самостоятельного определения и обработки «пользовательских» метаданных?

По общему признанию, пользовательское использование метаданных встречается редко. Обычно значения, с которыми мы работаем, достаточны сами по себе. Они не требуют метаданных. Всё же можно найти возможность для пользовательской логики, которая обрабатывает метаданные.

Допустим, вы хотите извлечь данные из веб-API. Эта конкретная конечная точка API возвращает не более 100 записей за вызов. Если запрашиваемый набор данных превышает 100 элементов, результаты разбиваются на страницы, при этом каждая страница (кроме последней) сопровождается ссылкой, которую можно использовать для получения следующей страницы данных:

```
// Ответ JSON на первый запрос возвращает первые 100 строк
{
  "Data": [ …rows 1-100…],
  "Next": "https://somewhere/someApi?nextPageToken=1905617808"
}

// Ответ JSON на второй запрос возвращает вторые 100 строк
{
  "Data": [ …rows 101-200…],
  "Next": "https://somewhere/someApi?nextPageToken=899780178"
}
```

```
// Ответ JSON на третий запрос -- возвращает последние 20 строк
{
  "Data": [ …rows 201-220…],
  "Next": null
}
```

Вы думаете начать с написания функции, которая передает URL-адрес и использует его для получения страницы данных. Другая логика, которую вы напишете, будет вызывать эту функцию столько раз, сколько необходимо, чтобы получить все страницы, а затем сшить их вместе в одну таблицу.

Ваша функция «получить страницу» должна возвращать две вещи: таблицу строк страницы и значение следующей ссылки. Вернуть таблицу достаточно просто, но как вы можете сопроводить ее следующей ссылкой? Метаданные — отличный способ сделать это:

```
GetPage = (url as text) as table =>
   let
      Response = Json.Document(Web.Contents(url)),
      Table = Table.FromRecords(Response[Data])
   in
      Table meta [NextLink = Response[Next]]
```

Обратите внимание, что следующая ссылка предназначена для тех, кому она нужна. Если информация о существовании следующей ссылки не интересна, можно просто рассматривать выходные данные функции как обычную таблицу.

Есть и другие способы, которыми это можно было бы осуществить, например, чтобы функция возвращала запись с полем для таблицы и одним для следующей ссылки. Если вы создаете новую структуру данных (например, запись) просто для того, чтобы сопроводить интересующее значение (в данном случае таблицу) дополнительной информацией (следующей ссылкой), отдайте предпочтение метаданным — это встроенный способ передачи данных о значении вместе со значением.

Ленивые вычисления и метаданные

Если метаданные не используются в коде, они не будут оцениваться. Например,

```
let
   CallFunction = (x) => x + 3,
   A = "somevalue" meta [ ts = CallFunction(5) ]
in
   A
```

Поскольку на выходе возвращается значение *A*, функция *CallFunction* никогда не будет вызываться.

Запрос метаданных (документации) библиотечной функции

Чтобы вывести документацию по библиотечной функции можно использовать код:

Запрос 17

```
let
   Источник = Value.Metadata(
      Value.Type(Table.SelectColumns)
   )
in
   Источник
```

```
= Value.Metadata(
        Value.Type(Table.SelectColumns)
    )
```

Documentation.Name	Table.SelectColumns
Documentation.Description	Возвращает таблицу, содержащую только определенные столбцы.
Documentation.LongDescriptio	Возвращает <code>table</code> только с указанными <code>columns</code>. <code>table</code>: предоставленная таблица. <code>columns</code>: список столбцов из таблицы <code>table</code>, которые должны быть <code>missingField</code>: <i>(необязательно)</i> что делать, если столбец не существует. При
Documentation.Category	Table.Column operations
Documentation.Examples	List

Рис. 12. Документация по функции Table.SelectColumns в окне предварительного просмотра

Детализация записи *Documentation.Examples* позволит вывести на экран примеры использования функции. Затем запрос можно загрузить на лист Excel.

Пример использования метаданных

Можно использовать несколько параметров с одинаковым именем для фильтра. Следующий код задает метаданные параметра *Country*:

```
null meta [
    IsParameterQuery=true,
    List={"Germany", "Poland", "Belgium", "Hungary", "Italy"},
    DefaultValue=...,
    Type="Text",
    IsParameterQueryRequired=false
],
```

Тогда код…

```
List.Contains(Value.Metadata(Country)[List],SearchTerm)
```

… позволяет добиться желаемого результата.

Использование метаданных для ссылки на промежуточный шаг другого запроса

Когда вы ссылаетесь на другой запрос в Power Query, вы автоматически получаете результаты последнего шага. Но что, если вы хотите сослаться на иной шаг в запросе!?[2]

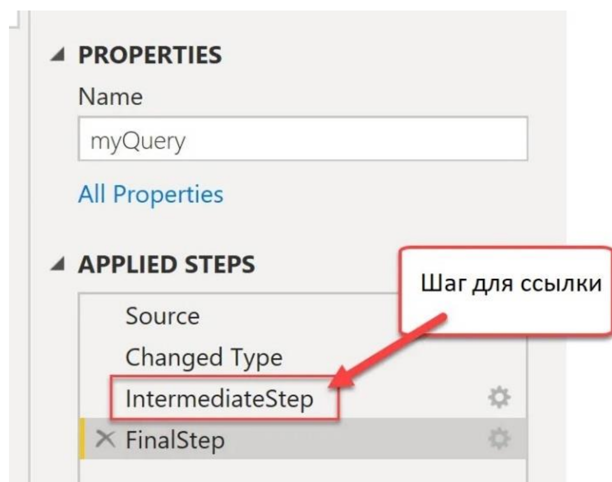

Рис. 13. Ссылка на промежуточный шаг другого запроса

---

[2] Этот раздел написан на основе заметки Имке Фельдманн (Imke Feldmann) [Reference an intermediate step from a different query in Power Query](#).

Конечно можно разделить запрос на два и ссылаться на конечный результат двух запросов. Но это может привести к увеличению времени запроса, так как данные могут извлекаться из источника дважды (по одному разу для каждого запроса). Альтернативой является добавление имени промежуточного шага в качестве метаданных последнего шага запроса:

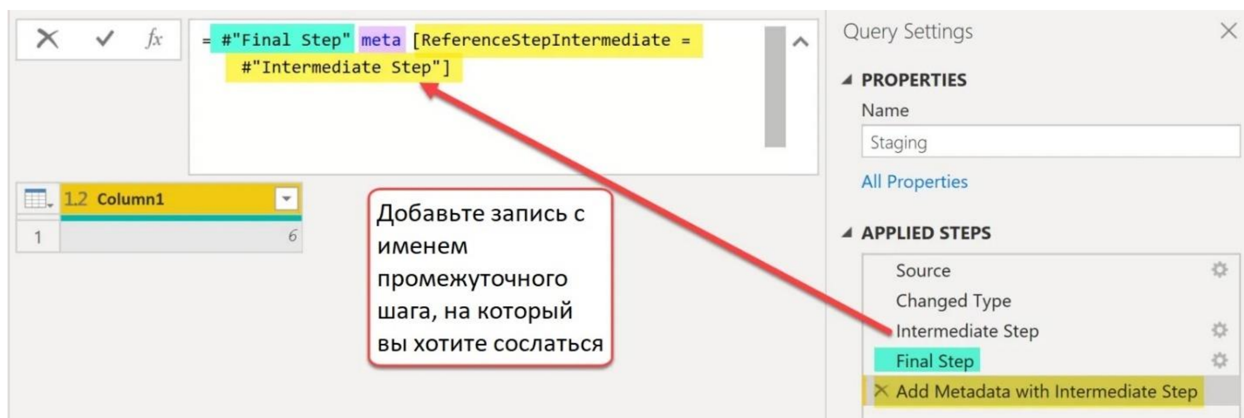

Рис. 14. Добавление метаданных в последний шаг запроса

Использование выражения *meta* позволяет добавить запись с упоминанием шага, на который хотите сослаться. Имя поля записи *ReferenceStepIntermediate* позволит обратиться к значению этого поля #"Intermediate Step" из другого запроса:

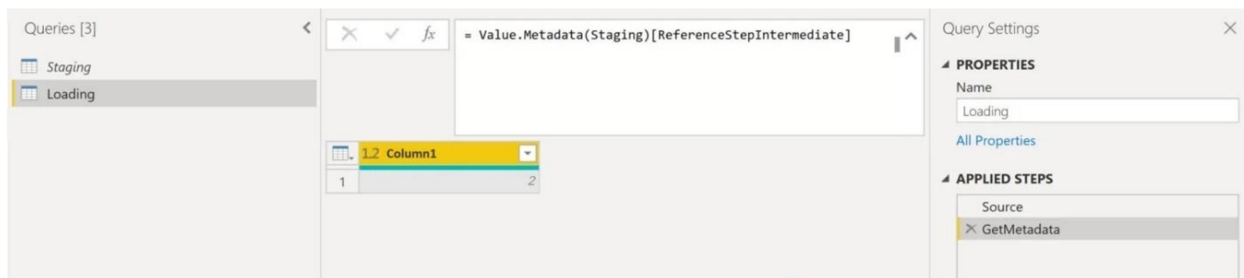

Рис. 15. Ссылка на промежуточный шаг из другого запроса

Это отличный способ сделать промежуточные шаги доступными для других запросов, не нарушая логику исходного запроса.

Давайте проясним этот подход на примере. Я продолжил Запрос 17, выполнив детализацию, и довел результат до записи с примерами использования функции *Table.SelectColumns*. А затем добавил в последний шаг метаданные:

Запрос 17

```
let
  Источник = Value.Metadata(
    Value.Type(Table.SelectColumns)
  ),
  #"Documentation Examples" = Источник[Documentation.Examples],
  #"Преобразовано в таблицу" = Table.FromList(
    #"Documentation Examples",
    Splitter.SplitByNothing(),
    null,
    null,
    ExtraValues.Error
  ),
  #"Развернутый элемент Column1" = Table.ExpandRecordColumn(
    #"Преобразовано в таблицу",
    "Column1",
    {"Description", "Code", "Result"},
    {"Description", "Code", "Result"}
  ) meta [ReferenceStepIntermediate = Источник]
```

```
in
    #"Развернутый элемент Column1"
```

	Description	Code	Result
1	Включить только столбец [Name].	Table.SelectColumns(Table.FromRecords({ [CustomerID = 1, Name = "E [CustomerID = 2, Name = "J [CustomerID = 3, Name = "F	Table.FromRecords({ [Name = "Bob"], [Name = "Jim"], [Name = "Paul"], [Name = "Ringo"]
2	Включить только столбцы [CustomerID] и [Name].	Table.SelectColumns(Table.FromRecords({[Custome {"CustomerID", "Name"})	Table.FromRecords({[CustomerID = 1, Name = "Bob"]})
3	Если включенный столбец не существует, результат по умолчанию...	Table.SelectColumns(Table.FromRecords({[Custome "NewColumn")	[Expression.Error] The field 'NewColumn' of the record wasn't found.
4	Если включенный столбец не существует, параметр <code>Missing...	Table.SelectColumns(Table.FromRecords({[Custome {"CustomerID", "NewColumn' MissingField.UseNull)	Table.FromRecords({[CustomerID = 1, NewColumn = null]})

Расширенный редактор

Запрос17

Отобразить параметры

```
let
    Источник = Value.Metadata(
        Value.Type(Table.SelectColumns)
    ),
    #"Documentation Examples" = Источник[Documentation.Examples],
    #"Преобразовано в таблицу" = Table.FromList(#"Documentation Examples", Splitter.SplitByNothing(), null, null,
         ExtraValues.Error),
    #"Развернутый элемент Column1" = Table.ExpandRecordColumn(#"Преобразовано в таблицу", "Column1",
        {"Description", "Code", "Result"}, {"Description", "Code", "Result"}) meta [ReferenceStepIntermediate =
        Источник]
in
    #"Развернутый элемент Column1"
```

✓ Синтаксические ошибки не обнаружены.

Готово Отмена

Рис. 16. Запрос 17 с метаданными

Далее создал…

Запрос 18

```
let
    Источник = Value.Metadata(Запрос17)[ReferenceStepIntermediate]
in
    Источник
```

… и получил картинку, как на рис. 12. Т.е., смог обратиться к промежуточной записи.