Язык M Power Query. Функции

Это вторая заметка в серии, описывающей азы языка M Power Query. В предыдущей заметке говорилось, что основу языка M составляют выражения, которые возвращают значения. Простой оператор, такой как 1, – это выражение, которое создает значение. *let* – это также выражение, которое выдает значение. Так вот, функция – это выражение, которое тоже возвращает значение. В отличие от выражений, функция выдает это значение только при ее вызове.[1]

Предыдущая заметка Следующая заметка

Обычно параметры передаются функции при ее вызове. Функция может ссылаться на эти входные данные при вычислении результата.

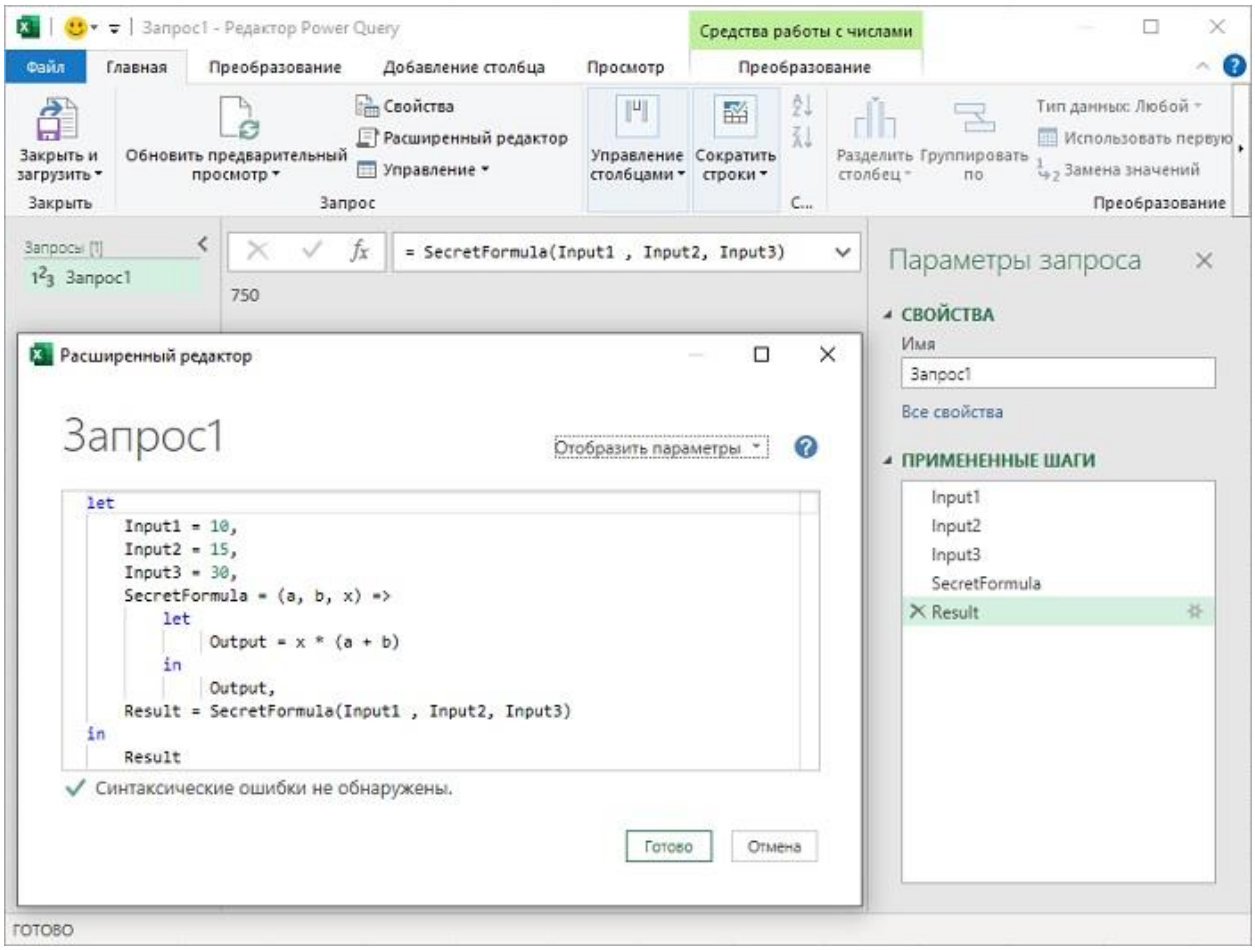

Рис. 1. Пример вложенной функции

Определение функции

Определение функции начинается со списка ожидаемых параметров внутри пары круглых скобок, за которым следует знак =>, и далее тело функции. Последнее определяет, как вычислить значение, возвращаемое функцией. Параметры, определенные для функции, становятся переменными внутри неё. В теле функции параметры используются для вычисления возвращаемого значения.

Следующая функция принимает два аргумента и умножает их в теле функции:

(x, y) => x * y

Поскольку тело функции – это выражение, которое выдает значение, то оно может содержать выражение *let*:

(a, b, x) =>

[1] Это сокращенный перевод статьи Ben Gribaudo. Power Query M Primer (part 2): Functions. Defining. Если вы впервые сталкиваетесь с Power Query, рекомендую начать с Марк Мур. Power Query.

```
let
   Result = x * (a + b)
in
   Result
```

Технически могут быть определены функции, которые не принимают никаких параметров. На практике это делается редко. Зачем использовать подобную функцию, если подойдет обычная переменная?!

```
() => "Привет, мир! Это супер-простая, почти бессмысленная функция"
```

Область определения функции

Как и другие выражения, функции могут быть **вложены** в другие выражения, а другие выражения могут быть вложены в функции. Функция также может быть вложена внутрь другой функции. Вот пример функции, определенной внутри выражения *let* (см. также рис. 1).

```
let
   Input1 = 10,
   Input2 = 15,
   Input3 = 30,
   SecretFormula = (a, b, x) =>
      let
         Output = x * (a + b)
      in
         Output,
   Result = SecretFormula(Input1 , Input2, Input3)
in
   Result
```

А вот пример функции, вложенной внутрь другой функции:

```
(OrderAmount, DiscountPercentage) =>
let
   NonDiscountedPercentage = (Rate) => 1 - Rate,
   Result = OrderAmount * NonDiscountedPercentage(DiscountPercentage)
in
   Result
```

В отличие от вложенной, **функция верхнего уровня** стоит сама по себе. На неё можно ссылаться из других выражений (включая другие функции), и она не определена внутри другого выражения. Она определена во всем файле Excel, и доступна для любых запросов в этом файле.

В редакторе запросов создание нового пустого запроса – это способ создания любого именованного выражения верхнего уровня, а не только запросов. Думайте о *новом пустом запросе* как о *новом пустом именованном выражении*. Редактор запросов посмотрит на заданное вами выражение, и сам сделает вывод, что это такое: запрос, таблица, список, функция… и автоматически внесет соответствующие коррективы. В редакторе запросов имя, присвоенное новому выражению верхнего уровня – это имя, которое можно использовать для ссылки на него из других частей вашего проекта.

Чтобы определить функцию верхнего уровня, в *Редакторе запросов* пройдите *Главная –> Создать источник –> Другие источники –> Пустой запрос*. Кликните *Главная –> Расширенный редактор*. Замените заготовку, которую предлагает редактор для ввода нового запроса, кодом функции. Проверьте, что нет синтаксических ошибок. Нажмите *Готово*. После выхода из расширенного редактора дайте подходящее имя новой функции. Теперь вы можете ссылаться на неё по имени из другого места проекта. Обратите внимание, как в списке запросов отображается функция. Перед ее именем нарисовался значок функции. Редактор запросов понял, что на самом деле это функция, а не запрос, и соответствующим образом изменил значок.

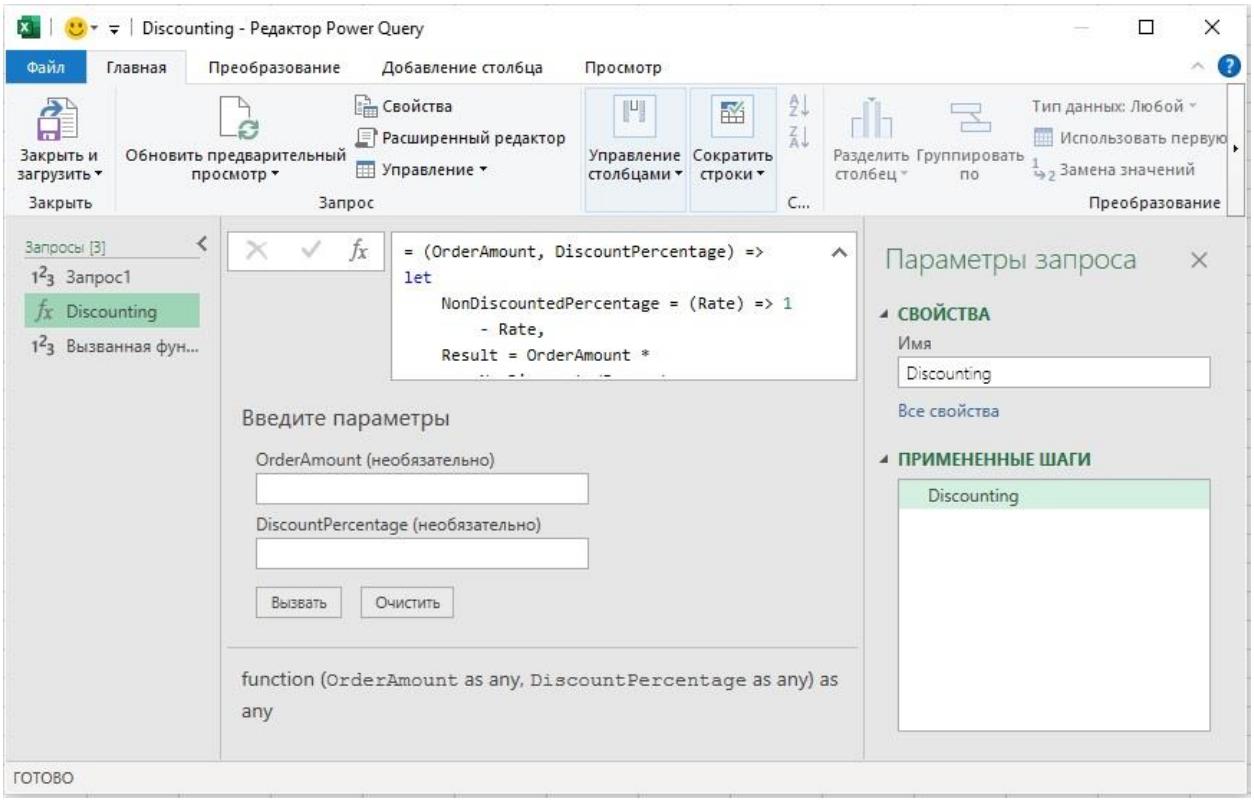

Рис. 2. Функция верхнего уровня Discounting

К сожалению, для функций в панели ПРИМЕНЕННЫЕ ШАГИ отображается одна строчка. С этой панели редактирование невозможно. Для изменения кода функции нужно перейти в расширенный редактор.

Необязательные параметры функции

По умолчанию при вызове функции для каждого параметра должно быть указано значение. Если вы хотите, чтобы некоторые аргументы были необязательными, укажите на это при определении параметров функции:

```
(LastName, FirstName, optional MiddleName) =>
   Text.Combine({FirstName, MiddleName, LastName}, " ")
```

Вызов этой функции требует указания по крайней мере двух обязательных аргументов и, необязательно, третьего. Обязательные аргументы перечисляются первыми.

Если назвать эту функцию StringifyName, то она может быть вызвана так…

```
StringifyName("Smith", "Joe")
```

… или так…

```
StringifyName("Brown", "Robert", "James")
```

Пользовательский интерфейс может сбивать с толку

Посмотрите на рис. 2. Хотя вы определили оба аргумента функции как обязательные, редактор Power Query показывает, что они необязательные. К сожалению, редактор запросов определяет необязательность аргументов не так, как язык M Power Query. Для редактора запросов необязательный означает, что либо параметр является необязательным, либо он является обязательным, но может иметь значение *null*.

Как редактор запросов обрабатывает аргументы, которые, как он утверждает, являются необязательными, если вы их не указываете? Он вернет выражение *null*:

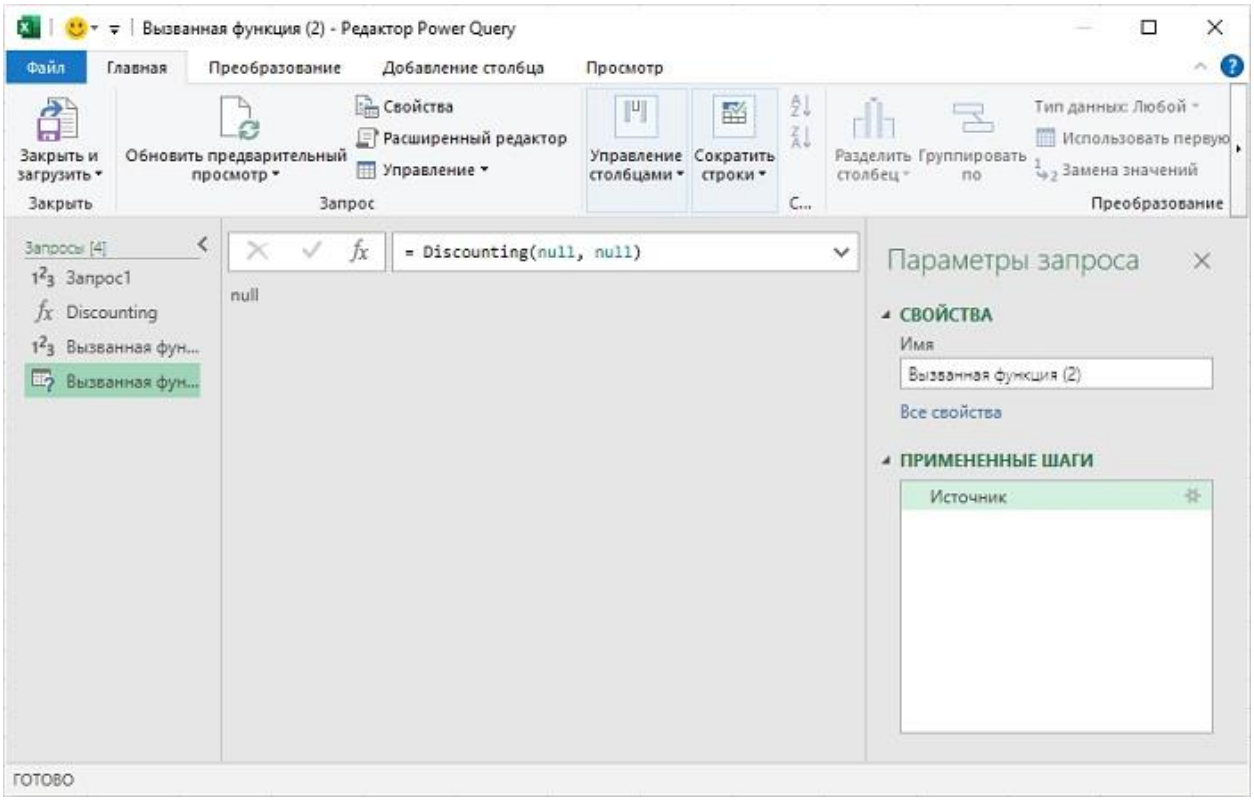

Рис. 3. Значения должны были быть указаны, поскольку аргументы действительно требуются, несмотря на то, что утверждает пользовательский интерфейс редактора

Если вы находитесь в редакторе запросов и хотите определить, какие аргументы действительно являются необязательными, посмотрите на описание функции внизу экрана. Там реальные необязательные параметры обозначаются ключевым словом *optional*.

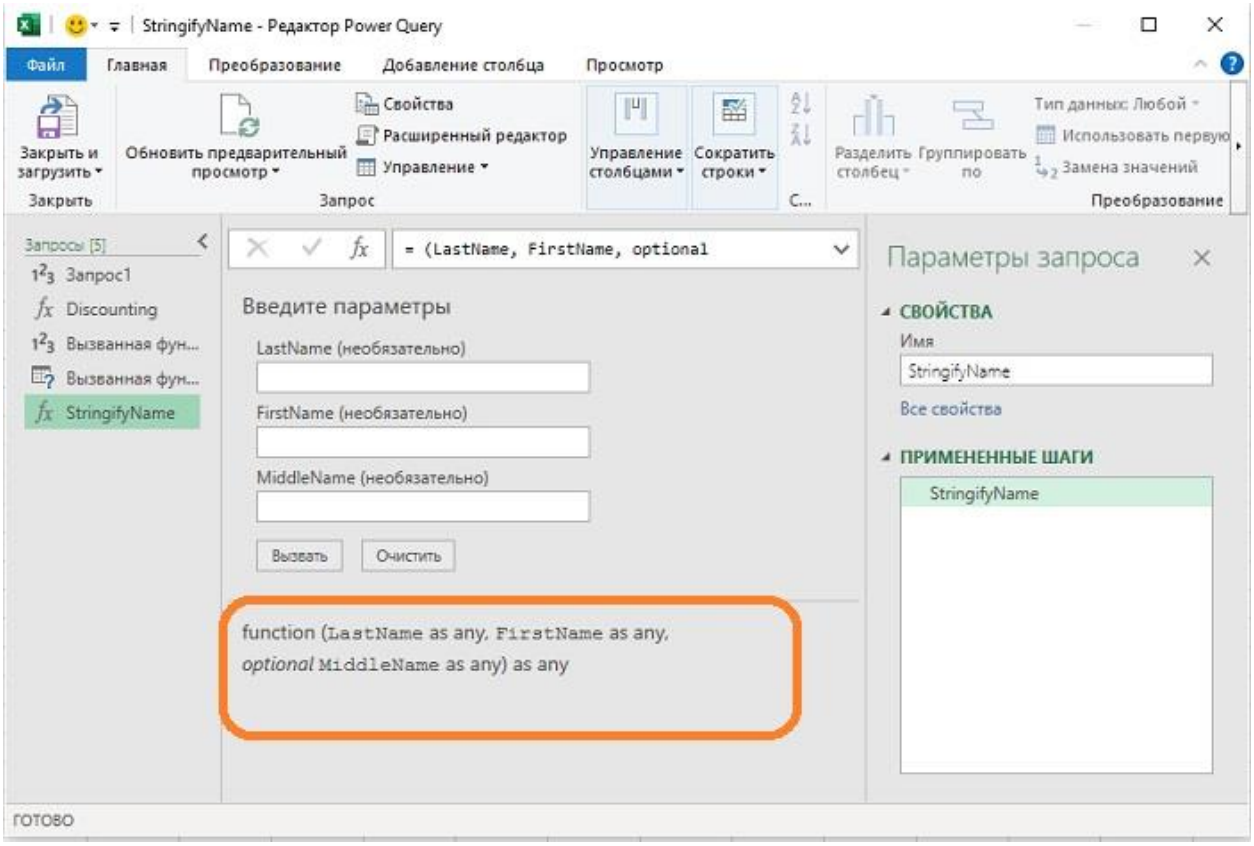

Рис. 4. В описании функции необязательные параметры сопровождаются ключевым словом *optional*

Типы

В синтаксисе функции также могут быть указаны типы параметров и тип возвращаемого значения. По умолчанию все они имеют тип *any*. Можно указать конкретный тип:

```
(PriceEach as number, Quantity as number) as text =>
  "Total Cost: " & Number.ToText(PriceEach * Quantity)
```

Эта функция определяет типы параметров (оба являются числами) и возвращаемого значения (текст).

Вы можете указать типы лишь для некоторых аргументов. Независимо от того, указываете вы типы параметров или нет, вы вольны указать тип результата функции или оставить его по умолчанию. Подробнее о типах в следующем посте.

Еще о пользовательском интерфейсе

Не все типы допускают *null*. Когда параметр указан как ненулевой тип, редактор запросов не может по умолчанию присвоить ему значение *null* и поэтому не будет отображаться слово *(необязательно)* рядом с именем параметра, если только параметр не был определен в коде как необязательный.

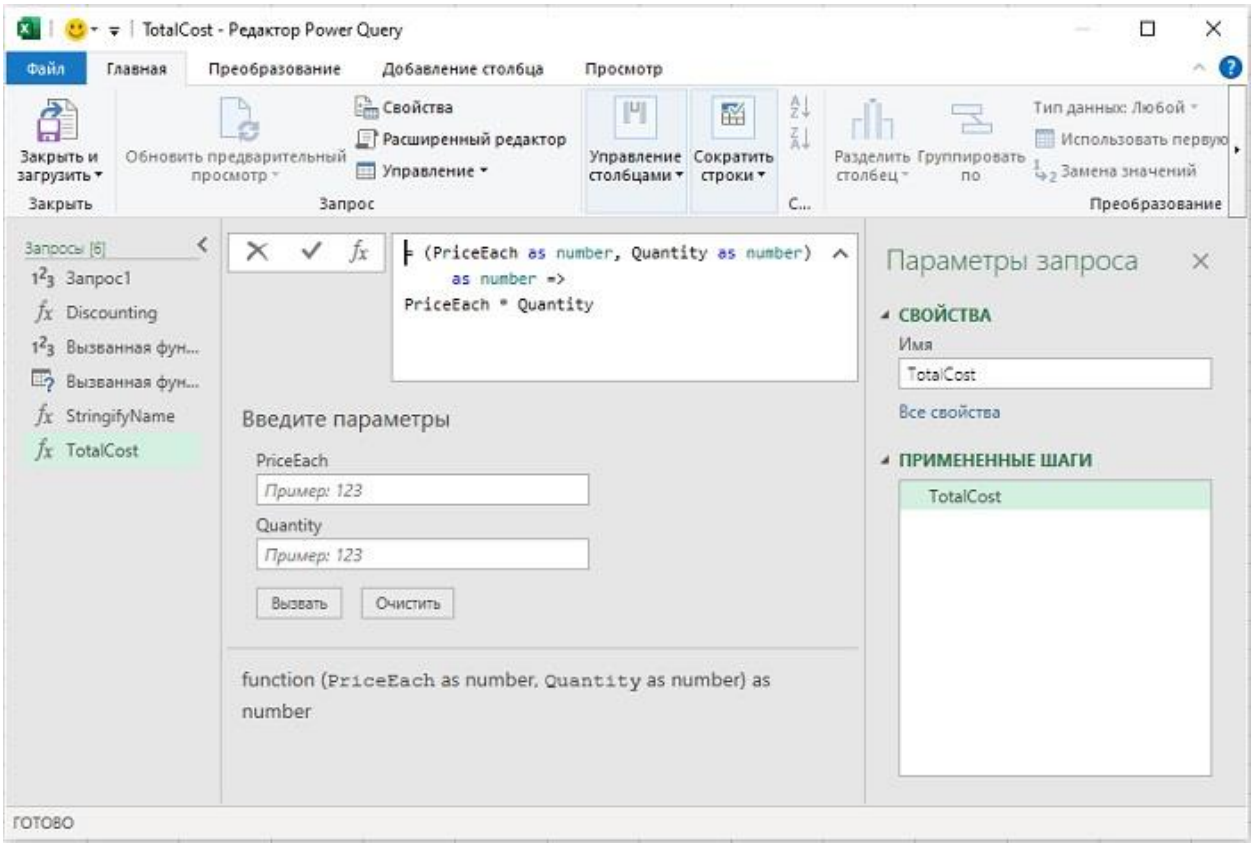

Рис. 5. Параметры функции *TotalCost*

Обратите внимание, что рядом с именами параметров *PriceEach* и *Quantity* не отображается слово *(необязательно)*, поскольку оба параметра не могут иметь значения *null*, т.е. являются обязательными.

Опять же, чтобы определить, действительно ли требуется параметр, посмотрите описание функции внизу окна.

> Существует способ представить описание функции иначе. Однако для этого требуется использовать метаданные. Мы рассмотрим эту технику позже.