Язык M Power Query. Область определения идентификатора и секции

Одно и то же имя идентификатора (например, имя переменной, поля и т.п.) в запросе Power Query может быть определено несколько раз. Если вы используете имя идентификатора, на какое из этих определений будет указывать ссылка? В этом посте мы узнаем, как M разбирается с этим. Мы также рассмотрим *разделы* – обычно скрытые «фреймы», лежащие в основе организации выражений, составляющих код Power Query.[1]

Предыдущая заметка Следующая заметка

Область идентификатора

Предположим, что код ниже — это все, что есть в мире Power Query. Нет ни стандартной библиотеки, ни других запросов, ни глобальной среды, с помощью движка оценивающей выражение. Что может «видеть» выражение, определяющее *a*? Другими словами, на какие переменные в коде можно ссылаться это выражение?

```
[
  a = 1,
  b = 2,
  c = 3
]
```

Выражение *a* может «видеть» (т.е. ссылаться) на переменные *b* и *c* (значения которых равны 2 и 3). Однако ссылка на *a* не допускается.

С технической точки зрения, ссылка на идентификатор в стиле *a* называется *эксклюзивной ссылкой на идентификатор*. «Эксклюзивный» подразумевает, что список идентификаторов, видимых для ссылки, исключает идентификатор переменной выражения let или поля записи, определяемого в данный момент. Приведенное выше выражение, определяющее *a*, не может содержать ссылку на *a*, так как *a* отсутствует в списке идентификаторов, видимых для ссылки, поскольку *a* является идентификатором, определяемым в данный момент.

Что произойдет, если вы попытаетесь сослаться на себя? M вернет ошибку, потому что он не видит имя, которое вы пытаетесь использовать.

```
[
   a = a * 10 /* вызывает ошибку "Expression.Error: Имя "a" не распознано.
      Убедитесь в том, что оно написано верно." */
]
```

Это ограничение против самоссылок – хорошая вещь. А не то мы бы получили бесконечную рекурсию.

Ссылки на инклюзивные идентификаторы

Хотя обычно самоссылки плохи, есть случай, когда они не только приемлемы, но и необходимы – рекурсивные функции. Для этого случая применяются *инклюзивные идентификаторы*. Просто добавьте к имени переменной символ @.

Ссылка на инклюзивный идентификатор работает так же, как и ссылка на эксклюзивный, за исключением того, что правило эксклюзивной ссылки «исключить текущую переменную let или поле записи» не применяется. Таким образом, набор идентификаторов, видимых для инклюзивной ссылки, включает определяемую в данный момент переменную или поле.

```
let
  SumConsecutive = (x) => if x <= 0 then 0 else x + @SumConsecutive(x - 1),
  Result = SumConsecutive(4)
in
  Result // ответ 10
```

[1] Заметка написана на основе статьи Ben Gribaudo. Power Query M Primer (Part 21): Identifier Scope & Sections. Если вы впервые сталкиваетесь с Power Query, рекомендую начать с Марк Мур. Power Query.

Нет никакого технического запрета на использование инклюзивных ссылок там, где они не нужны (как показано ниже), но это не имеет смысла.

```
let
  a = 10,
  b = 20
in
  @a * @b // знак @ здесь не нужен; запись a * b предпочтительнее
```

Ненужное использование этого стиля ссылок может даже вызвать путаницу, поскольку разработчик, читающий ваш код, может увидеть @, и потратить время на размышления о том, почему вы его использовали.

Если вам приходит идея использовать инклюзивную ссылку на идентификатор вне контекста рекурсивной функции, скажем, для кодирования выражения a = @a * 10, вы ступаете на тонкий лед. M предотвратит беду, возвращая ошибку Expression.Error: Циклическая ссылка была обнаружена во время оценки. Обратите внимание, что эта ошибка возникнет только в том случае, если движок обнаружит циклическую ссылку. Если это не так, M попытается вычислить выражение, и тогда у него могут закончиться ресурсы.

Родительский уровень

Усложним задачу – завернем первый пример во внешнюю запись. Что увидят внутренние идентификаторы?

```
[
  a = 10,
  inner = [
    a = 1, // видит a = 10 (из внешней записи), b = 2, c = 3
    b = 2, // видит a = 1 (из внутренней записи), c = 3
    c = 3 // видит a = 1 (из внутренней записи), b = 2
  ]
]
```

Внутреннее *a* теперь может видеть *a*, но это *a* не является самим собой, это *a* из внешней записи (a = 10). Ссылка на *a* в выражениях *b* и *c* указывает на внутреннее *a* (a = 1). Следующий код…

Запрос 1[2]

```
[
    a = 10,
    inner =
    [
        a = a,
        b = 2,
        c = 3,
        d = a + b + c
    ][d]
][inner]
```

… вернет 15.

Что происходит? Ссылки сначала ищут идентификаторы локально. Если совпадение не найдено, проверяется родительский уровень. Поскольку вложенностей может быть несколько, проверка родителя идет по восходящей вплоть до глобальной среды, пока не будет найден соответствующий идентификатор.

Запрос 1 иллюстрирует идею для записи, вложенной в другую запись, но такое поведение является универсальным, и применимо ко всем идентификаторам в M (к переменным выражения let, ссылкам между запросами и т.п.).

---

[2] Номер соответствует запросу в приложенном Excel-файле

Ссылки на перекрестные запросы

Говоря о ссылках между запросами, рассмотрим книгу с двумя запросами:

```
// Data
10

// Query1
let
   Data = 100,
   Result = Data /* видит Data из выражения let (Data = 100),
      а не значение из запроса с именем Data (Data = 10) */
in
   Result // возвращает 100
```

Result в Query1 ссылается на идентификатор Data. В файле есть два идентификатора Data. На какой из них указывает ссылка Result? Данные, определенные в выражении let (значение = 100). Почему? Когда два идентификатора имеют одно и то же имя, локальный побеждает родительский.

Секции

Концепция ссылок между запросами наводит на интересную мысль. В редакторе запросов каждый запрос отображается в пользовательском интерфейсе как отдельная сущность. Тем не менее, как показывает последний пример, эти визуально разделенные запросы могут ссылаться друг на друга. Чтобы сделать это возможным, должна существовать какая-то родительская область — например, какое-то скрытое супер-выражение или запись, содержащая все запросы файла.

Родительский «контейнер», содержащий запросы, называется *секцией* и определяется внутри документа раздела (section document).

Хорошо это или плохо, но документы разделов в настоящее время не видны в инструментах редактирования запросов Microsoft ориентированных на потребителя (например, в редакторе запросов). Это не значит, что их там нет — просто вы (обычно) их не видите. Вместо этого, за кулисами, движок автоматически управляют ими за вас (например, когда вы создаете запрос, движок добавляет его в соответствующий раздел; когда вы обновляете запрос, он обновляет соответствующее место в разделе и т.д.).

Тем не менее, разделы (хотя они и скрыты) влияют на то, как работает движок, поэтому изучение их поможет вам понять язык M. Кроме того, кто знает, может быть, однажды вы окажетесь в одной из тех сложных ситуаций, когда вам нужно будет напрямую редактировать документ раздела (например, при создании пользовательского соединителя данных с помощью пакета SDK для Power Query).

Вот как выглядит документ раздела для предыдущего примера:

```
section Section1;

shared Data = 10;

shared Query1 = let
  Data = 100,
  Result = Data
in
  Result // returns 100
;
```

Вы можете убедиться в этом сами, создав два запроса в Microsoft Excel. Сохраните и закройте файл. Откройте его в Data Mashup Explorer и пройдите по меню *Package Parts –> Formulas –> Section1.m*. Раздел начинается с ключевого слова *Section*, за которым следует название раздела и точка с запятой. Каждое имя раздела должно быть уникальным в глобальной среде.

Члены раздела (также известные как запросы)

Пользовательский интерфейс редактора запросов называет определенные выражения верхнего уровня «запросами». Однако этот термин не всегда является точным дескриптором, поскольку то, что вы можете определить, не ограничивается запросами: вы также можете создавать параметры, функции и т. д. К счастью, в разделах используется более широко применимый термин для эквивалентной концепции.

В разделе каждый *запрос* из пользовательского интерфейса приравнивается к члену раздела. Глядя на приведенный выше пример, обратите внимание, что он содержит два элемента раздела. Каждый идентифицируется по имени, затем есть знак равенства, затем выражение для члена раздела, затем точка с запятой.

О *shared* в начале каждого члена раздела мы поговорим чуть позже.

Разрешение ссылок на идентификатор

Что касается разрешения ссылок на идентификаторы, раздел является родительской областью для выражений членов этого раздела.

Допустим, один из этих членов определен как выражение let, которое включает ссылку на Query1. Чтобы разрешить эту ссылку, M сначала будет искать идентификатор с таким именем в выражении let. Если он не найдет соответствия, он проверит раздел, содержащий выражение let (так как оно является родительским выражением let), на наличие идентификатора с именем Query1.

```
section Section1;

shared Query1 = …;

shared Query2 = let
  Result = Table.FirstN(Query1, 10)
in
  Result /* поскольку в самом выражении нет Query1, ссылка перенаправит на Query1,
находящийся в разделе */
;
```

Вездесущий Section1

Каждый из рассмотренных нами разделов содержит один раздел под названием *Section1*. *Section1* — это раздел, который средства Power Query, ориентированные на потребителя, используют для хранения написанных вами запросов. С точки зрения языка, это имя не имеет особого значения; это просто то, что Microsoft решила использовать.

Хранение недопустимого кода в мире, требующем валидности

Выражение члена секции должно быть синтаксически корректным и должно быть, ну… выражением. Поскольку документ раздела не является выражением, член раздела не может содержать вложенный документ раздела.

Говоря о допустимом синтаксисе, в редакторе запросов вы можете установить выражение запроса на что-то, что содержит недопустимый синтаксис (представьте, что вы находитесь в процессе кодирования, а затем вам нужно выйти на день, чтобы сохранить неполное выражение). Как хранится этот ошибочный код, если выражения-члены раздела должны содержать только точный синтаксис?

За кулисами ваш редактор запросов экранирует любые двойные кавычки в неправильно сформированном коде, а затем оборачивает его в литерал #!" ...". Вот неполное выражение:

```
let "hi!"
```

В документе раздела этот код преобразуется в следующее выражение – член раздела:

```
#!"let ""hi!""";
```

Литеральные атрибуты

Как разделы, так и члены разделов могут быть украшены так называемыми литеральными атрибутами. Это набор литеральных (жестко запрограммированных, а не вычисляемых) значений,

хранящихся в конструкции, похожей на запись, которые можно использовать для присоединения дополнительного уровня метаданных к разделу в целом или к отдельным членам раздела.

Спецификация языка M не придает особого значения какому-либо конкретному литеральному атрибуту. Она лишь определяет синтаксис для задания этих атрибутов. Однако инструментарий может использовать эти атрибуты, чтобы, например, реализовать специальное поведение в средстве приложения.

Например, редактор запросов использует литеральный атрибут *Description* для хранения описания, связанного с членом раздела:

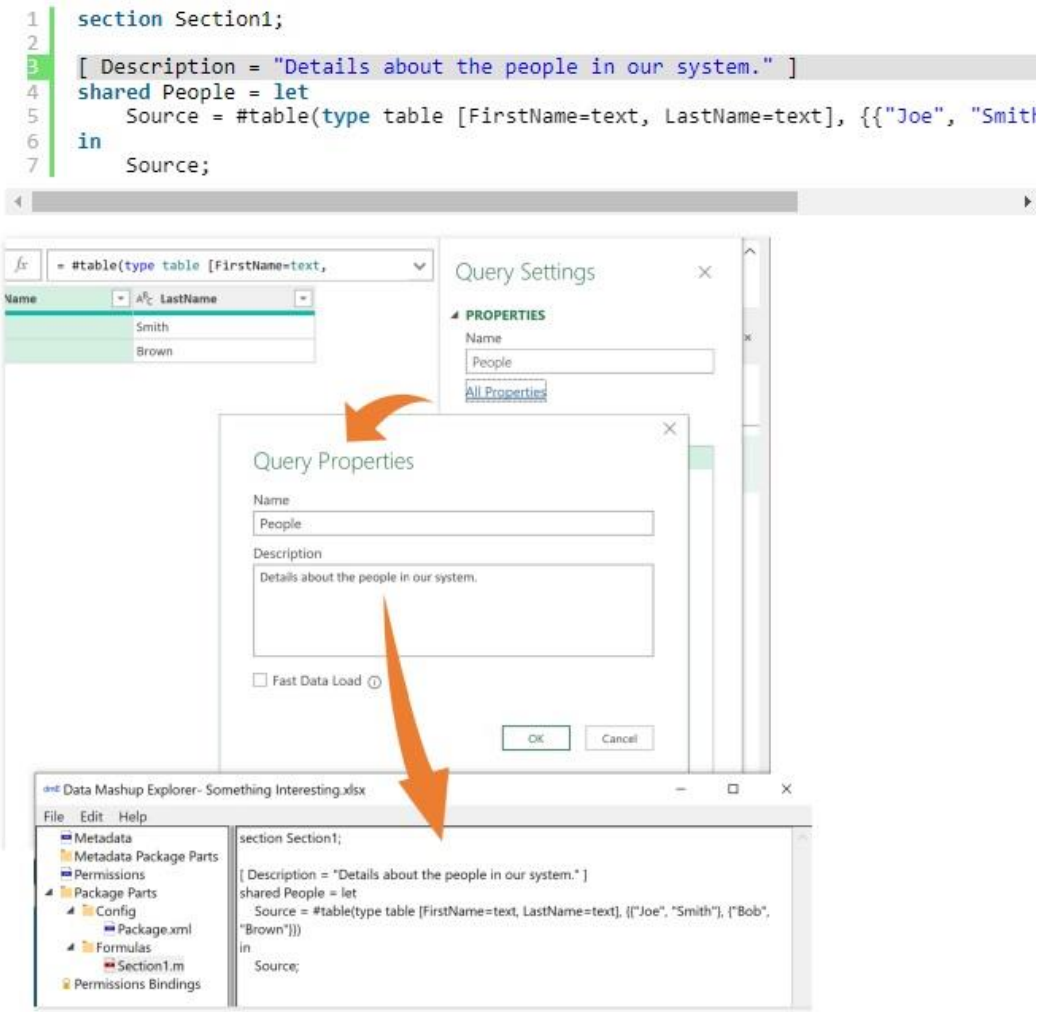

Рис. 1. Описание члена раздела, хранящимся в литеральном атрибуте *Description*

Из того, что я видел, эти атрибуты используются лишь в нескольких местах. Например, когда с помощью пользовательской функции импортируются все файлы из папки для связывания вместе примера и функции. Или во время разработки пользовательского коннектора данных.

Ключевым отличием между литеральными атрибутами и метаданными является буквальность первых. Литеральные атрибуты можно использовать в тех случаях, когда динамически вычисляемые значения были бы дорогостоящими или неуместными. Чтение литеральных атрибутов является быстрым, безопасным и предсказуемым, без возможности рекурсивных ссылок, вызовов внешних систем и т. д. С чем можно столкнуться, используя метаданные. Например, такое выражение для метаданных является допустимым:

value meta [IsImportant = CheckWhetherShouldBeImportantToday()]

Что касается синтаксиса литеральных атрибутов, подумайте о записи, значения полей которой ограничены только строками, числами, нулями и логическими значениями; жестко запрограммированные списки, состоящие из одного и того же набора литеральных значений, и вложенные записи, поля которых также ограничены.

Организационные, неисполнимые

До того, как мы столкнулись с разделами, весь M-код состоял только из выражений. Выражения могут выполняться и либо возвращать результаты, либо вызывать ошибки. Это не относится к разделам. Раздел – это организационная структура для языка M. Разделы сами по себе не являются исполняемыми (и поэтому не создают значений); они содержат именованные выражения, которые при желании могут быть выполнены. Вы не можете выполнить раздел, но вы можете попросить движок выполнить элемент раздела.

#sections

Как увидеть разделы, которые в настоящее время присутствуют в вашей среде? M определяет специальное ключевое слово *#sections*, которое позволяет увидеть все существующие разделы. За исключением некоторых расширенных сценариев (например, при использовании Embedded.Value или при разработке пользовательских коннекторов), будет присутствовать только один раздел: *Section1*.

Запрос 2

#sections

Рис. 2. *#sections* возвращает запись, содержащую одно поле

#sections возвращает запись, содержащую по одному полю для каждого раздела. Значение каждого также является записью, содержащей поле для каждого из членов раздела. Эти поля можно использовать для доступа к значениям этих элементов.

Запрос 3

#sections[Section1]

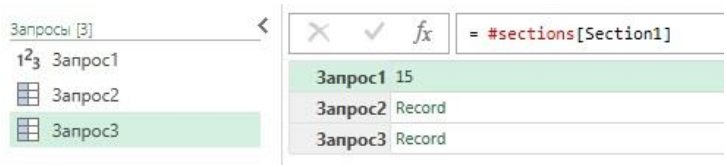

Рис. 3. Результат запроса *#sections[Section1]*: запись с одним полем для каждого члена раздела *Section1*

Вспомните один из предыдущих примеров. Если вы хотите, чтобы ссылка Query1 на Data указывала на запрос (также известный как член раздела) с именем Data, а не на переменную Date внутри let, вы можете изменить ссылку с Data на #sections[Section1][Data].

```
// Data
10


// Query1
let
  Data = 100,
  Result = #sections[Section1][Data] // видит запрос с именем Data (значение = 10)
in
  Result // возвращает 10
```

Это работает, хотя есть более предпочтительный вариант синтаксиса, который достигает того же эффекта...

Выражения доступа к разделам

Вместо #sections[Section1][Data] можно использовать выражение доступа к разделу в качестве более короткого способа указать ту же ссылку:

```
Section1!Query1
```

Идентификатор раздела + "!" + идентификатор члена раздела.

Общие участники секции

Для чего используется ключевое слово shared перед именем члена раздела?

Представим, что однажды экосистема Power Query позволит обычным пользователям создавать несколько разделов. Вы определяете один раздел для своей личной «стандартной библиотеки» удобных вспомогательных функций, которые вы копируете и вставляете во все файлы отчетов на основе Power Query. Затем в этих файлах вы используете обычный редактор запросов для определения запросов (членов раздела), которые пользовательский интерфейс по умолчанию сохраняет в Section1.

Допустим, что в одном из этих файлов документ раздела выглядит примерно так:

```
section Section1; // управляется графическим редактором запросов
shared CustomerData = …;
shared OrderData = …;

section MyHelpers; // copy-and-paste из вашего файла
shared CleanupColumnNames = (input) => …;
shared ConvertFromUnixTimestamp = (input) => …;
```

Как один из ваших запросов (членов раздела) в *Section1* будет ссылаться на запрос в MyHelpers? Предположим, что CustomerData (в *Section1*) хочет использовать CleanupColumnNames в MyHelper. Тогда код…

```
let
  Source = Sql.Database(…),
  CleanColumns = MyHelpers!CleanupColumnNames(Source),
  …
in
  Result
```

…будет работать.

Более того, использование идентификации другого раздела (фрагмент MyHelpers!) можно опустить, если расшарить члена раздела, на который идет ссылка. Расшаренный член раздела добавляется в глобальную среду, которая является финальным местом, где ищется идентификатор.

В нашем примере член раздела CleanupColumnNames в разделе MyHelper уже расшарен, и мы можем просто сократить ссылку на CustomerData до…

```
let
  Source = Sql.Database(…),
  CleanColumns = CleanupColumnNames(Source),
  …
in
  Result
```

Как работает поиск идентификатора для ссылки на CleanupColumnNames? Сначала M локально проверяет идентификатор с указанным именем. Не найдя ни одного, он проверяет раздел, содержащий CustomerData (это будет Section1). Поскольку CleanupColumnNames там также не определен, M переходит к проверке глобальной среды. Здесь, наконец, он находит идентификатор с именем CleanupColumnNames, который существует, потому что он был расшарен.

Если это так удобно, почему бы не делать всех членов разделов общими всегда? Если вы не расшарите члена раздела, вы избежите ненужного загрязнения глобальной окружающей среды и риска возникновения конфликта названий.

Предположим, что в разделе MyHelpers у вас есть член с именем ToUpper, который предназначен только для внутреннего использования другим кодом в MyHelpers. Отсутствие *shared* передает это намерение. Несмотря на то, что доступ к идентификатору по-прежнему можно получить из других разделов (с помощью #sections[Section1] или Section1!Query1), мы надеемся, что дополнительная

работа отпугнет потенциальных потребителей, подсказывая им, что вы не намеревались использовать этот член раздела в других разделах.

Глобальный конфликт имен. Предположим, что два раздела содержат расшаренный элемент с именем DoSomething. Вычисляется выражение, которое ссылается на DoSomething. Эта ссылка не находит идентификатора на локальном уровне. Когда M проверяет глобальную область, он находит два DoSomething. Не имея возможности выбрать, какой из них использовать, движок вызовет ошибку. Эта двусмысленность и возникающая в результате ошибка не возникнут, если будет только один из двух DoSomethings будет расшарен.

Не стесняйтесь делиться членами раздела, но в то же время не делитесь без необходимости. Для

Так как же не поделиться участником секции? Просто не используйте ключевое слово shared перед именем члена раздела:

```
section MyHelpers;

// not shared
ToUpper = (input) => …;

// shared
shared CleanupColumnNames = (input) => …;
```

Глобальная окружающая среда

Глобальная среда — это вершина иерархии, где M ищет идентификаторы для запрошенных ссылок. Она состоит из всех разделов, всех расшаренных членов разделов, а также любых идентификаторов, непосредственно введенных в глобальную область.

Внедрение в глобальную область!? Доступно ли это пользователям? Обычно нет, но Microsoft так делает. В частности, стандартная библиотека добавлена в глобальную среду. Обычная глобальная среда состоит из всех общих членов + стандартной библиотеки.

#shared

Чтобы просмотреть идентификаторы в глобальной среде, введи #shared в строку кода редактора PQ. Вернется запись с полем для каждого глобального идентификатора. Значения этого поля можно использовать для доступа к значениям, связанным с этими идентификаторами.

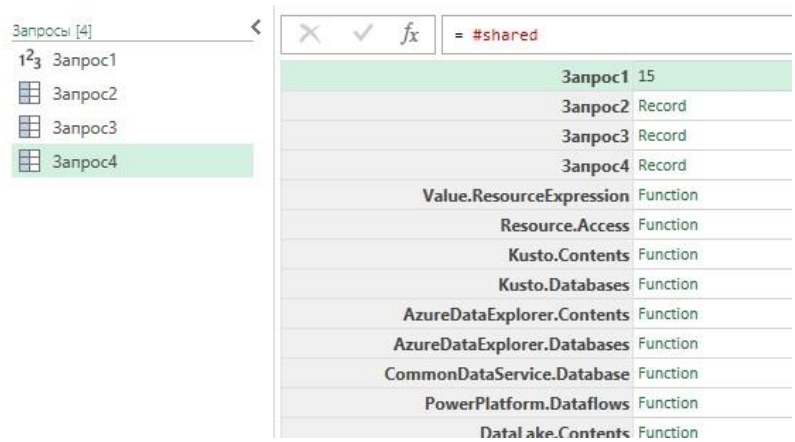

Рис. 4. Запрос #shared выводит запись, содержащую все элементы стандартной библиотеки M

Есть несколько интересных применений для #shared. Например, вы можете увидеть, какие стандартные библиотечные функции включает ваша конкретная установка Power Query.

Запрос 5

```
let
   SharedAsTable = Record.ToTable(#shared),
   FilteredToStandardLibraryFunctions = Table.SelectRows(
      SharedAsTable,
      each Text.Contains([Name], ".") and Type.Is(
         Value.Type([Value]),
```

```
        type function
    )
  )
in
  FilteredToStandardLibraryFunctions
```

Вы даже можете пойти дальше и извлечь документацию по этим функциям из метаданных в таблицу...

Запрос 6

```
let
  SharedAsTable = Record.ToTable(#shared),
  FilteredToStandardLibraryFunctions = Table.SelectRows(
    SharedAsTable,
    each Text.Contains([Name], ".") and Type.Is(Value.Type([Value]), type function)
  ),
  ExtractTypeMetadata = Table.AddColumn(FilteredToStandardLibraryFunctions, "Docs", each
Value.Metadata(Value.Type([Value]))),
  DocumentationToClumns = Table.ExpandRecordColumn(
    ExtractTypeMetadata,
    "Docs",
    {
      "Documentation.Name",
      "Documentation.Description",
      "Documentation.LongDescription",
      "Documentation.Category"
    }
  ),
  RemoveValueColumn = Table.RemoveColumns(DocumentationToClumns,{"Value"})
in
  RemoveValueColumn
```

Стандартные библиотечные специальные имена

Может ли у вас когда-нибудь возникнуть конфликт имен между расшаренным членом раздела и именем из стандартной библиотеки? С одной стороны, ни одно правило в спецификации языка M этому не препятствует. С другой, редактор Power Query не даст использовать точку в названии запроса (т.е. в имени члена раздела). Напротив, все идентификаторы, определенные стандартной библиотекой, имеют точку в своем имени (например, Table.SelectRows, List.Count).

Имейте в виду, что имена идентификаторов, содержащие точки, используемые стандартной библиотекой, — это просто имена. Если вы имеете опыт объектно-ориентированного программирования, этот стиль именования может заставить вас задаться вопросом, идентифицирует ли имя перед точкой объект, а имя после точки — метод в этом классе. Например, является ли Text.Upper ссылкой на метод *Upper* класса *Text*.

Нет! В M Text.Upper – это просто имя с точкой. Практика использования точек в названиях стандартной библиотеки является соглашением исключительно для удобства разработчиков.

В следующий раз

Вы узнали, что глобальная среда состоит из всех общих членов и стандартной библиотеки, и что обычно мы не можем вводить элементы в глобальную область. В следующий раз мы узнаем о случае, когда это возможно, т.е., когда у нас есть полный контроль над глобальной средой. Кроме того, есть одна ситуация с идентификаторами, которая дает возможность кодировать элементы, похожими на объекты.