**Введение в язык формул M Power Query**

Это фрагмент книги [Гил Равив. Power Query в Excel и Power BI: сбор, объединение и преобразование данных](https://baguzin.ru/wp/?p=24451).

[Предыдущий раздел](https://baguzin.ru/wp/?p=24644) [К содержанию](https://baguzin.ru/wp/?p=24451#content)  Следующий раздел

Возможности языка M реализуются с помощью трех пользовательских интерфейсов:

* Строка формул — дает возможность изменить функцию M для настройки шагов преобразований в соответствии с вашими потребностями. Обычно мы так поступали, если в пользовательском интерфейсе отсутствовала конкретная возможность (например, по применению расширенного логического условия при фильтрации столбца).
* Окно *Настраиваемый столбец* — обеспечивает выполнение некоторых вычислений на уровне строк и сохранение результатов в новом столбце.
* Расширенный редактор — позволяет изменить несколько шагов в запросе, включить код из другого запроса или создать пользовательскую функцию.

Даже если вы не до конца разобрались с синтаксисом или с кодом, тот факт, что код генерится с помощью пользовательского интерфейса, помогает понять код. Лучший способ изучить возможности языка M — применить шаги преобразования в пользовательском интерфейсе, а затем постепенно пытаться освоить базовый код.

Можно выделить шесть этапов в изучении языка M. Каждый этап позволяет решать новые типы проблем с данными, но при этом он будет удерживать вас в определенной «зоне комфорта». Задайте себе вопрос: «Готовы ли вы приложить значительные усилия, чтобы перейти к следующему этапу?»



Рис. 1. Этапы зрелости в обучении языку М

#### Этап 1. Только интерфейс пользователя

Если вы впервые познакомились с языком М именно в этой книге и до сих пор прилежно выполняли упражнения, значит, вы уже овладели начальным уровнем подготовки в изучении М и можете считаться опытным пользователем Power Query. Как известно, язык M используется в качестве основного языка программирования, но у вас могут быть сомнения в необходимости изучать его, поскольку непонятно, стоит ли игра свеч. Возможно, вы также опасаетесь приступать к изучению языка программирования или же отдаете предпочтение другим языкам, которые отличаются от языка M. Имеется большое число языков, включая формулы Excel, VBA, DAX, Python и R, и вы можете освоить многие из них. Почему же следует изучать М, который значительно отличается от всего, с чем мы привыкли работать?

Катализатор для перехода к следующему этапу изучения этого языка заключается в понимании, что пользовательский интерфейс поможет устранить только около 40% ваших проблем. Если же задаться целью разрешить большее число проблем с данными с помощью Power Query, вдруг обнаруживается, что многие проблемы могут быть преодолены с помощью небольших изменений кода. Поскольку данная настройка не требует глубокого понимания синтаксиса, это естественный этап на пути изучения языка, даже если вы не очень глубоко разбираетесь в формулах Excel или мало осведомлены в вопросах программирования.

#### Этап 2. Простое редактирование в строке формул

На втором этапе зрелости в строке формул применяются небольшие модификации к М-коду. Можно не понимать формулы полностью, но уже связывать некоторые элементы в формуле с действиями, которые вы предприняли в пользовательском интерфейсе, и расширить его для удовлетворения своих потребностей. На первых двух этапах можно устранить около 60% проблем, с которыми вы сталкиваетесь, и существенно повлиять на возможности по обработке данных.

Желание перейти к следующему этапу приходит, когда появляется растущая потребность в более сложных вычислениях в отчетах и когда вы становитесь достаточно опытны в других областях, таких как DAX в Power BI или Power Pivot в Excel. Когда вы узнаете о возможностях вычисляемых столбцов в DAX, то можете подумать, что проще реализовать решение с помощью настраиваемого столбца на языке M.

#### Этап 3. Формулы M в настраиваемых столбцах

На третьем этапе рассматривается методика применения настраиваемых столбцов, что позволит вам создавать простые, но достаточно эффективные M-формулы в поле *Пользовательская формула* *столбца* окна *Настраиваемый столбец*. Вы узнаете об использовании логических условий в формулах, успешно примените в работе операторы if/then/else в сочетании с логическими операторами and, or и not. Окно *Настраиваемый столбец* удобнее, чем окно *Условный столбец*. На первых трех этапах можно устранить около 80% проблем, с которыми обычно сталкиваются пользователи.

И хотя некоторым пользователям вполне достаточно знаний, полученных на этом этапе, многие уже представляют спектр широких возможностей, предоставляемых М, приобретают уверенность в работе с языком M для решения новых задач. В этом случае появляется желание покинуть границы строки формул и приступить к изучению выражений M в расширенном редакторе, чтобы применять свои знания при создании формул M и расширять запросы для решения дополнительных задач.

#### Этап 4. Настраиваемые функции в окне расширенного редактора

На четвертом этапе осваивают формирование настраиваемых функций для повторного использования шагов преобразования. И хотя средство *Создать функцию* редактора Power Query трудно переоценить, можно без затруднений преобразовывать запросы в пользовательские функции с помощью расширенного редактора. Также можно применять расширенный редактор для масштабного изменения кода.

На этом этапе уже можно успешно разрешать до 95% проблем, с которыми обычно сталкивается пользователь. Ваша репутация агента по сбору данных уже выйдет за пределы команды, многие коллеги смогут обращаться к вам за помощью в решении их проблем с данными. На этом этапе вы уже можете успешно решить практически любую проблему с данными. Основным катализатором для перехода на следующий уровень является ваша личная любознательность, а также желание досконально овладеть возможностями языка М. Находясь на этом этапе, можно приступить к формированию библиотеки полезных пользовательских функций и искать новые способы эффективного решения проблем и повторного использования запросов в сложных ситуациях.

#### Этап 5. Расширенные вычисления (List.Accumulate, List Generate)

На пятом этапе можно переходить к сложным сценариям и создавать сложный М-код для их воплощения. Продолжая использовать Power Query Editor для создания кода, вы сами формируете существенные и сложные выражения кода в расширенном редакторе. Также вы сможете легко создавать вложенные выражения для блоков let/in и обобщать автоматически сгенерированный код для расширения его возможностей.

На этом этапе обнаруживается, что функции List.Accumulate и List.Generate чрезвычайно полезны для создания итераций преобразований. Эти функции пригодны для создания небольших фрагментов кода М, которые применяются как аргументы в других функциях М. Даже если число случаев, где это действительно нужно, невелико, тот факт, что вы создаете расширенные формулы M с помощью функций List.Accumulate и List.Generate, является свидетельством высокого уровня вашего мастерства при работе с языком M. Если вам известно, как задействовать эти функции, то можно полагать, что вы находитесь на этапе 5. В некоторых организациях вы уже заслуженно получите репутацию специалиста по сбору данных.

На этом этапе можно устранить 99% возникающих проблем. Оставшиеся 1% нерешенных проблем могут быть решены с помощью других языков программирования. Поэтому ваш переход к следующему этапу определяется исключительно интеллектом или опытом разработки программного обеспечения. Если вы не являетесь суперпрофессионалом в области разработки программного обеспечения, существенной причины для перехода к последнему этапу нет.

#### Этап 6. Кодирование в произвольной форме в окне расширенного редактора

На шестом этапе освоения языка M вы сможете создавать сложнейшие M-выражения «с нуля», не используя пользовательский интерфейс редактора Power Query. В этом случае кодирование в произвольной форме является естественным для вас, у вас должна быть исключительно хорошая память, поскольку расширенный редактор не предоставляет возможности автозаполнения или IntelliSense (по крайней мере, на момент написания этой книги). На этом этапе вам необходимо запоминать большое число имен функций и их аргументов.

Обширный опыт работы с языками программирования поможет вам достичь этого этапа, и если вы находитесь на нем, то можно искать новые способы совершенствования. Можно написать код M для более широкой аудитории, поделиться функциями M с коллегами или со всем сообществом или, в качестве следующего этапа, приступить к разработке пользовательских коннекторов данных в Visual Studio. Описание настраиваемых коннекторов данных выходит за рамки этой книги. Чтобы узнать, как их разработать, обратитесь на сайт [Microsoft/DataConnectors](https://github.com/Microsoft/DataConnectors).

#### Ресурсы в Интернете

Официальный ресурс Microsoft – [Язык формул Power Query M](https://docs.microsoft.com/ru-ru/powerquery-m/). Довольно часто на официальном справочном сайте отсутствуют конкретные примеры, которые помогают лучше понять ту или иную функцию. Вам предлагается поискать в Интернете соответствующие статьи в блогах на эту тему. Вот некоторые из основных блогов, посвященные Power Query:

* [Chris Webb](https://blog.crossjoin.co.uk/category/power-query/)
* Ken Puls: <https://www.excelguru.ca/blog/> (на момент публикации заметки ссылка не открывалась)
* [Matt Allington](https://exceleratorbi.com.au/blog/power-query-2/)
* [Reza Rad](https://radacad.com/category/power-query)
* [Imke Feldmann](https://www.thebiccountant.com/)
* [DataChant](https://datachant.com/)

Power Query имеет встроенную справочную систему (автономную документацию). Просто введите знак равенства и имя функции в строке формул редактора Power Query. Не добавляйте скобки или аргументы. После нажатия клавиши Enter справка появится на панели предварительного просмотра:



Рис. 2. Введите имя функции в строку формул для просмотра справки по ней

#### Применение переменной #shared для изучения встроенных функций

Для нахождения новых функций и ознакомления с автономной документацией можно запустить редактор Power Query и в строке формул ввести строку:

= #shared

Эта переменная возвращает запись всех поддерживаемых функций в Power Query, включая ваши запросы. Создайте пустой запрос: пройдите *Данные* –> *Получить данные* –> *Из других источников* –> *Пустой запрос*. В окне Power Query в строке формул введите:

= #shared

На панели предварительного просмотра отобразится список. Выберите пустое пространство в любой из ячеек функции, чтобы просмотреть в нижней части окна документацию по этой функции. Или щелкните на ссылке *Function* для перехода к фактической функции.



Рис. 3. Некоторые функции и константы

Для нахождения всех функций с помощью пользовательского интерфейса Power Query преобразуйте выходную запись #shared в таблицу, пройдя *Средства для записей* –> *Преобразовать* –> *В таблицу*. Результат можно загрузить в таблицу на лист Excel.

Вернитесь в редактор PQ. Выберите элемент управления фильтра в столбце *Name* и выберите *Текстовые фильтры* –> *Начинается с* –> *List*. Щелкните Ok. Теперь можно изучить все функции, названия которых начинаются с List. Файл решения C09E01 - Solution.xlsx приложен.

#### Компоненты языка M

Центральный компонент языка М — выражение — блок кода, в результате вычисления которого получаем одно значение. Каждый шаг преобразования, созданный в редакторе Power Query, генерирует M-выражение. Панель предварительного просмотра всегда отображает значение, полученное в результате вычисления выражения.

В языке M каждый шаг преобразования представляет собой пару, состоящую из идентификатора и его выражения. Строка формул отображает элемент выражения на выбранном шаге, а на панели *Примененные шаги* можно увидеть идентификатор. В окне *Расширенный редактор* можно увидеть все выражение M-запроса. И хотя код в окне Расширенного редактора сложнее, чем одношаговое M-выражение в строке формул, в результате выполнения этого кода возвращается одно значение. Следующее упражнение демонстрирует это на простом выражении запроса, возвращающем текст Hello World.

Пройдите *Данные* –> *Получить данные* –> *Из других источников* –> *Пустой запрос*. На вкладке *Главная* кликните *Расширенный редактор.* Вставьте код:

let

Источник = "Hello",

Step2 = Источник & " ",

Step3 = Step2 & "World"

in

Step3

Щелкните *Готово*. На панели предварительного просмотра появится сообщение Hello World. Поздравляю. Вы написали первую программу на языке M. Рассмотрим код и выясним, как он связан с элементами пользовательского интерфейса. На рис. 4 приведены основные связи между элементами пользовательского интерфейса расширенного редактора, строкою формул и панелью *Примененные шаги*.



Рис. 4. Пользовательский интерфейс Power Query содержит несколько компонентов языка М: расширенный редактор, строка формул и панель *Примененные шаги*

На панели предварительного просмотра можно видеть значение Hello World. Обычно на панели просмотра отображаются таблицы, но в нашем случае показано текстовое значение. Выражение, включенное в окно расширенного редактора, возвращает единственное значение Hello World. В строке формул выводится такой код:

= Step2 & "World"

Этот код является выражением, которое присваивается идентификатору Step3 в окне расширенного редактора. Он объединяет значение Step2 с текстом World. И хотя выражение находится в строке формул, идентификатор Step3 представлен на панели *Примененные шаги*.

Если выбрать шаг Step2 на панели *Примененные шаги*, то в строке формул отобразится:

= Источник & " "

В окне расширенного редактора это выражение представлено с сопутствующим идентификатором Step2 следующим образом:

Step2 = Источник & " ",

Завершающая запятая в этом выражении не отображается в строке формул, но служит для разделения между парами идентификатор/выражение, которые являются частью выражения let.

И хотя большинство запросов транслируется в линейную последовательность из M-шагов, таких как Источник, Step2 и Step3, язык M позволяет описывать сложные потоки и применять условия, которые приведут к различным путям для исполнения.

#### Выражение let

Всякий раз при создании новых запросов с помощью редактора Power Query и импорте наборов данных в отчет после некоторой подготовки данных генерируется базовое выражение let. После открытия окна расширенного редактора отображается выражение M, которое начинается с ключевого слова let и обычно заканчивается in, за которым следует идентификатор последнего шага.

Выражения let позволяют создавать подмножества меньших выражений, которые объединяются в одно значение. Рассмотрим пример из предыдущего упражнения. В этом случае выражение let состоит из трех шагов. Step1 возвращает значение Hello. Step2 возвращает значение в Step1 и завершающий пробел, а Step3 возвращает значение в Step2 и завершающее значение World. Значение in определяет вторую фазу вычислений, которую возвращает выражение let. В этом базовом примере после in определяется только Step3. В результате возвращаемое значение является тем же значением в Step3: Hello World.

Ключевое слово let позволяет записывать выражения, которые выполняются и после ключевого слова in. Например, код:

let

Источник = "Hello",

Step2 = Источник & " ",

Step3 = Step2 & "World"

in

Step3 & "!!!"

…вернет значение Hello World!!!

Или так:

let

Источник = "Hello",

Step2 = Источник & " ",

Step3 = Step2 & "World",

ExclamationMarks = "!!!"

in

Step3 & ExclamationMarks

Возвращаемое выражение такое же: Hello World!!!

Выражение let может включать вложенное выражение let:

let

FirstWord =

let

Источник = "Hello"

in

Источник,

SecondWord =

let

Источник = "World"

in

Источник

in

FirstWord & " " & SecondWord & "!!!"

И хотя это выражение допустимо, оно усложняет работу с редактором Power Query. При этом отдельные шаги не отображаются на панели *Примененные шаги*. Для того чтобы решить эту проблему и получить отдельные шаги на панели *Примененные шаги,* следует переписать код М, добавляя идентификатор Result, и переместить выражение, расположим его перед заключительным in:

let

FirstWord =

let

Источник = "Hello"

in

Источник,

SecondWord =

let

Источник = "World"

in

Источник,

Result = FirstWord & " " & SecondWord & "!!!"

in

Result



Рис. 5. Структура с несколькими let и отражением шагов на панели *Примененные шаги*

Можно заметить, что идентификатор *Источник* встречается дважды. Как правило это приводит к ошибке, но в последнем выражении эти идентификаторы содержатся внутри двух различных выражений let, вложенных во внешнее выражение let. Все подобные идентификаторы, определенные внутри выражения let, являются локальными, и к ним невозможен доступ из внешнего выражения let. Первый *Источник* имеет значение Hello, второй – World.

#### Объединение выражений из нескольких запросов и областей доступа

Представление об области доступа в языке M и понимание выражения let может быть весьма полезным. Часто возникает необходимость повторно использовать некоторые этапы преобразования из запроса и применить их в другом запросе. Чтобы правильно скопировать и вставить строки выражений из одного запроса в другой, обычно следует выполнить три корректирующих шага в объединенном выражении.

Представим, что у вас есть запрос с тремя шагами преобразования, как показано в этом псевдокоде:

let

Step1 =…,

Step2 =…,

Step3 =…,

in

Step3

Во втором запросе на шаге Step2 применялся полезный фильтр, который желательно использовать в первом запросе:

let

Step1 =…,

Step2 = Table.SelectRows(Step1, …),

in

Step2

Простое копирование строки кода…

let

Step1 =…,

Step2 =…,

Step3 =…,

Step2 = Table.SelectRows(Step1, …),

in

Step3

…конечно же приведет к ошибке. Нужно исправить три момента:

1. Step2 уже определен в исходном выражении
2. Первым аргументом функции Table.SelectRows в исходном варианте был Step1, но теперь необходимо применить данное преобразование к Step3.
3. После ключевого слова in сначала возвращается Step3 как значение этого выражения, теперь следует вернуть значение в последнем идентификаторе.

Вот правильный код:

let

Step1 =…,

Step2 =…,

Step3 =…,

Step4 = Table.SelectRows(Step3, …),

in

Step4

В некоторых случаях нужно скопировать множество строк кода из одного запроса в другой. В подобных ситуациях иногда довольно сложно изменять сразу все строки, особенно если в объединенном выражении много дублирующих идентификаторов. Для разрешения этой проблемы можно применить вложенное выражение let:

let

Step1 =…,

Step2 =…,

Step3 =…,

Step4 =

let

Step2 = Table.SelectRows(Step3, …),

In

Step2

in

Step4

Обратите внимание, что область во вложенном выражении let содержит идентификаторы Step2, который локально переопределяется в области вложенного выражения let и не воспринимается в дальнейшем как дублирующий идентификатор. Когда вычисляется Step4, внешнее значение Step2 не переписывается с помощью значения, которое возвращается внутренним Step2. Два идентификатора Step2 имеют различные диапазоны.

Наконец, пример, где внешний идентификатор используется во вложенном выражении let и затем переопределяется во внутреннем диапазоне вложенного let:

let

A = 1,

B =

let

A = A + 1

in

A

in

B

Это выражение возвращает число 2. Идентификатор A определяется дважды. Сначала A определяется в контексте верхнего выражения let и получает значение 1. Затем внутри вложенного let определяется новый идентификатор A с выражением, которое использует внешний идентификатор A и увеличивает его на единицу.

Однако внутренний идентификатор A отличается от внешнего идентификатора A и недоступен во внешнем выражении let. Рассмотрите, что происходит, когда используется то же выражение, но на сей раз идентификатор A возвращается последним выражением in:

let

A = 1,

B =

let

A = A + 1

in

A

in

А

Этот код возвращает число 1 вместо 2 даже в том случае, если A увеличен во внутреннем выражении let. Во внешней области A по-прежнему будет равен 1.

#### Типы, операторы и встроенные функции в языке M

Значение в языке M — это данные, полученные путем вычисления выражения. Каждый тип значения связан с определенным синтаксисом, набором значений подобного типа, набором операторов, функциями, которые применяются со значениями подобных типов, и определением типа, который наследуется.

Рассмотрим различные языковые элементы, зависящие от типа. Числа могут быть представлены в виде литералов, таких как 0, 100, -2,5 или 2,5e-7. Этими значениями можно манипулировать с помощью семейства встроенных M-функций, начинающихся с префикса Number, за которым следуют точка и имя функции. Например, Number.Abs — встроенная функция, которая принимает число в качестве аргумента и возвращает его абсолютное значение. Числа в языке M имеют встроенные операторы, такие как =, <, >, <=, >=, + и -.

Язык M позволяет объявлять типы в аргументах или возвращаемых значениях пользовательских функций и проверять, относится ли конкретное значение к определенному типу. Например, следующее выражение, введенное в строке формул, вернет TRUE, как показано на рис. 9.5.

= 1000 is number



Рис. 6. 1000 является числом, поэтому выражение в строке формул возвращает значение TRUE

Язык M чувствителен к регистру символов. Логические значения, отображаемые в верхнем регистре, TRUE и FALSE используются только для вывода информации. В выражениях M следует применять версию в нижнем регистре (true или false); в противном случае вы получите ошибку.

Типы также определяются во встроенных функциях. Например:

Number.Abs(number as nullable number) as nullable number

Данная строка, называемая объявлением функции, объясняет, что Number.Abs принимает один аргумент, который может быть числом или нулем. Возвращаемое значение этой функции — число или нуль. Таким образом…

= Number.Abs(-1000)

… вернет…

1000

…а

= Number.Abs(null)

…вернет значение…

null

Если попытаться выполнить функцию для других типов данных, вы получите сообщение об ошибке. Например…

= Number.Abs("hello")

…вернет

Expression.Error: Не удается преобразовать значение "hello" в тип Number.

Далее мы будем ссылаться на функции определенного типа, такого как Type.X. Например, Number.X (к числу которых относится Number.Abs).

#### Основные типы данных языка M

Представьте, что имеется функция Number.X, которую нельзя разыскать в пользовательском интерфейсе, допустим Number.Sqrt, которая преобразует числа в первом столбце в значения квадратного корня. Чтобы написать формулу воспользуемся интерфейсом с иной формулой. Например, после выбора типа данных Научный –> Абсолютное значение в строке формул появится выражение:

= Table.TransformColumns(Источник,{{"Column1", Number.Abs, type number}})

Можно просто заменить его на

= Table.TransformColumns(Источник,{{"Column1", Number. Sqrt, type number}})

Если функция Number.X требует больше одного аргумента, можно указать ключевое слово each, за которым следует имя функции и символ подчеркивания \_, что позволит представить числовое значение, затем можно добавить другие аргументы.

Например, формула возведения в степень чисел, находящихся в столбце Column1:

= Table.TransformColumns(Источник,{{"Column1", each Number.Power(\_,3), type number}})

#### Числовой тип данных

Числа в языке М могут находиться в диапазоне от 5,0\*10-324 до 1,7\*10308, с точностью 15 или 16 цифр. Также считаются числовыми значениями:

* Положительная бесконечность: +#infinity
* Отрицательная бесконечность: -#infinity
* Не-число, часто употребляется сокращение NaN: #nan

Бесконечные значения формируются в результате выполнения таких операций, как деление ненулевого числа на нуль. Например, 1/0 дает положительную бесконечность, а -1/0 — отрицательную. Значение NaN может быть получено с помощью недопустимых операций с плавающей точкой, таких как деление нуля на нуль.

Вы можете сравнивать два числа, применяя логические операторы >, > =, <, <=, = и <>. Основные арифметические операции могут быть выполнены с использованием +, -, \* и /.

Язык M предоставляет богатый набор встроенных функций Number.X. Есть несколько постоянных. Например, Number.Pi возвращает значение числа пи. Это позволяет использовать постоянные в преобразованиях. Функции преобразования, такие как Number.From, могут преобразовывать значения в числа. Например, следующая функция преобразует текст "6" в число 6:

= Number.From("6")

Многие математические операции и тригонометрические функции Number.X также доступны в редакторе Power Query, например Number.Power и Number.Sin, на вкладках *Преобразование* и *Добавить столбец*. Перечень основных функций Number.X см. <https://docs.microsoft.com/ru-ru/powerquery-m/number-functions>

#### Тип времени

В языке М значение времени представлено как число 100-наносекундных тактов, прошедших с полуночи. Максимальное число тактов с полуночи соответствует 23:59:59.9999999 часов. Значения времени могут быть созданы с использованием выражения #time, синтаксис которого следующий: #time(чac, минута, секунда). Например, 2:53 после полудня можно записать в строке формул:

= #time(14, 53, 0)

что приведет к такому результату:

12:53:00

Значения времени можно обрабатывать с помощью логических операторов =, <>, >=, >, < и <=, причем эти значения могут быть левым или правым операндом в выражениях с операторами +, - и & со значениями времени и продолжительности. Например, добавить 7 минут ко времени 2:53 вечера можно следующим образом:

= #time(14, 53, 0) + #duration(0, 0, 7, 0)

Что приведет к такому результату:

15:00:00

Функции Time.X позволяют извлекать элементы времени из значения времени или вычислять начало или конец часа. Например, применение функции Time.Hour к временному значению

= Time.Hour(#time(14, 53, 0))

вернет значение часа в виде числа:

14

Больше о функциях времени можно узнать на вкладках *Преобразование* и *Добавить столбец*. См. также <https://docs.microsoft.com/ru-ru/powerquery-m/time-functions>

#### Тип даты

Значение даты представляется в виде числа дней, прошедших с начала эпохи (1 января 0001 г., общее время, прошедшее с начала летоисчисления по григорианскому календарю). Максимальное число дней с начала эпохи составляет 3 652 058, что соответствует 31 декабря 9999 года. Значения даты можно получить с помощью выражения #date, используя следующий синтаксис: #date(year, month, day). Например, дату 2 апреля 2018 года можно записать:

= #date(2018, 4, 2)

Тип даты поддерживает логические операторы =, <>, >=, >, < и <= и может быть левым или правым операндом в выражениях с операторами +, - и & со значениями длительности.

В языке М имеется большое число функций Date.X. Например, Date.AddDays получает дату в качестве первого аргумента и число дней для добавления в качестве второго аргумента. В следующем выражении добавляются 5 дней к дате 2 апреля 2018 г.:

= Date.AddDays(#date(2018, 4, 2), 5)

Полный список встроенных функций Date.X см. <https://docs.microsoft.com/ru-ru/powerquery-m/date-functions>

Язык M также поддерживает значения DateTime и DateTimeZone. Типами для них являются datetime и datetimezone, их можно автоматически генерировать в редакторе Power Query в случае применения соответствующих преобразований, доступных на вкладках *Преобразование* и *Добавить столбец*. Имеется большое число функций DataTime.X и DateTimeZone.X, включая DateTime.LocalNow, которые позволяют вычислить текущее время и дату, рассчитать прошедшее время транзакций или применить фильтры относительного времени.

#### Тип длительности

Значение длительности хранит числовое представление интервала между двумя точками на временной шкале, измеренное в 100-наносекундных тактах. Величина длительности может быть как положительной, так и отрицательной, с положительными значениями, обозначающими увеличение времени, и отрицательными значениями, обозначающими уменьшение времени. Минимальные и максимальные значения, которые можно сохранить в продолжительности, составляют 10,7 млн. дней назад или вперед во времени.

Значения длительности строятся на основе выражения #duration. Например, продолжительность 5 дней и 10 минут:

= #duration(5, 0, 10, 0)

Чаще всего значения длительности применяют к двоичным операторам, таким как + и - в сочетании со значениями даты, времени, datetime и datetimezone. Например, можно добавить 5 дней к дате 2 апреля 2018:

= #date(2018, 4, 2) + #duration(5, 0, 0, 0)

Вычитание двух значений даты или двух значений времени также приводит к значению длительности. Например, выражение

= #date(2018, 08, 28) - #date(2018, 01, 01)

…даст результат:

239.00:00:00

С помощью встроенных функций Duration.X можно вычислить значения длительности, что позволит извлечь дни, часы, минуты или секунды для компонентов длительности. Функции Duration.TotalX, такие как Duration.TotalSeconds, позволяют организовать подсчет количества секунд, которые составляют определенную продолжительность. Например, следующее выражение вычисляет общее количество секунд в 5 днях:

= Duration.TotalSeconds(#duration(5, 0, 0, 0))

…и выводит такой результат:

432000

Полный перечень встроенных функций Duration.X см. <https://docs.microsoft.com/ru-ru/powerquery-m/duration-functions>

#### Текстовый тип данных

Текстовые типы данных могут содержать произвольные символы Юникода. Можно объединять два текстовых значения с помощью оператора &. Например, с помощью функции Text.From можно преобразовать число в текст. В следующем примере выполняется объединение текста и числа, но сначала число преобразуется в текст:

= "Testing, Testing " & Text.From(123)

и выводится следующее значение:

Testing, Testing 123

В отличие от сложных типов данных, в языке M для доступа к определенному символу в тексте нельзя применять операторы, такие как [] или {}. Для получения доступа к первому символу можно использовать функцию Text.At с текстом в качестве первого аргумента и нулевым базовым индексом в качестве второго аргумента:

= Text.At("ABCD", 0)

Это выражение дает следующий результат:

A

Функция:

= Text.At("ABCD", 1)

дает следующий результат:

B

Полный перечень текстовых функций см. <https://docs.microsoft.com/ru-ru/powerquery-m/text-functions>

#### Тип данных Null

Тип данных null представляет отсутствие значения. Можно применять логические или арифметические операторы как нулевые, и возвращаемый результат также будет нулевым. Например, в следующих выражениях, если A равно нулю, то результаты будут равны нулю:

= A < 5 = A >= 5 = A + 1 = A \* 5

Особый случай — применение операторов равенства (=) и неравенства (<>) к нулевому значению. В следующих выражениях результат равен true или false, если A равно нулю:

= A = null

на панели предварительного просмотра будет выведено:

TRUE

Выражение

= A <> 5

приводит к следующему результату, отображаемому на панели предварительного просмотра:

FALSE

Выражение

= A <> null

приводит к следующему результату, отображаемому на панели предварительного просмотра:

FALSE

= A is null

приводит к следующему результату на панели предварительного просмотра:

TRUE

Обратите внимание, что null является как типом, так и значением. Таким образом, можно применять либо соответствующий оператор, либо знак равенства, чтобы проверить, является ли определенное значение нулевым.

#### Логический тип данных

Логическое значение используется для логических операций и может быть true или false. Доступны следующие операторы: >, >= , <, <=, =, <> (не равно), or (или), and (и), not (не). Можно проверить, является ли определенное значение логическим, используя выражение is logical. Например, если A содержит значение false, то следующее выражение вернет значение true:

= A is logical

На панели предварительного просмотра отображается результат TRUE.

Логические операторы обычно присутствуют в выражениях if.

#### Сложные типы данных

К сложным типам относятся: списки, записи, таблицы. Базовые типы определяют одно или атомарное значение, сложные типы имеют структурированный формат, который позволяет включить несколько базовых типов или другие сложные типы. При импорте данных в Power Query обычно работают с таблицами. Но таблица может содержать вложенные записи, списки и таблицы (при работе с неструктурированными данными, такими как файлы JSON или XML). Поэтому можно преобразовать ее в список (при детализации до столбца) и даже — в список записей.

Список включает последовательность значений. Список может содержать любые значения, в том числе списки, записи, таблицы или любые комбинации различных типов. Списки в выражении M формируются с использованием синтаксиса инициализации, в котором фигурные скобки инкапсулируют значения, разделенные запятыми. Например, данный список включает числа, логические элементы и текст:

= {1, 3, true, false, "Hello World"}

Следующее выражение инициализирует список чисел 1, 2 и 3:

= {1, 2, 3}



Рис. 7. Можно инициализировать список в строке формул, используя фигурные скобки

Если необходимо создать список увеличивающихся чисел от 1 до 100, можно применить следующее выражение инициализации:

= {1:100}

Для создания списка чисел от 1 до N, где N предоставляется другим запросом (например, на основе другого источника данных), можно применить следующее выражение:

= {1:N}

Достичь того же результата позволяет функция List.Numbers:

= List.Numbers(1, N)

Оператор двух точек (..) дает возможность определить числовой диапазон в списке. Например, следующее выражение возвращает список чисел 1, 4, 5, 6, 7, 8, 9, 10, 15:

= {1, 4..10, 15 }

Количество элементов в списке можно определить с помощью функции List.Count:

= List.Count({1..10})

Эта функция возвращает следующий результат:

10

Также можно определить пустой список, используя фигурные скобки, без каких-либо элементов между ними.

Применение функции List.Count для пустого списка

= List.Count({})

приведет к выводу нуля.

#### Операторы списка

Для объединения двух списков предусмотрен оператор объединения (&). Вот пример, возвращающий список {1, 2, 3, 4, 5}:

= {1, 2} & {3, 4, 5}

Чтобы определить, равны ли два списка, можно использовать операторы равенства (=) и неравенства (< >).

Следующее выражение, введенное в строке формул, возвращает значение TRUE:

= {1, 2} = {1, 2}

Следующее выражение, введенное в строке формул, тоже возвращает TRUE:

= {2, 1} <> {1, 2}

Два списка с одинаковыми элементами и разным порядком их следования не равны. Для получения доступа к элементу в списке по его местоположению можно задать нулевой индекс в фигурных скобках. Например, выражение

= {"A", "B", "C"}{0}

возвращает значение:

A

Выражение

= {"A", "B", "C"}{2}

возвращает значение:

C

При попытке доступа к элементу, расположенному за пределами списка, получаем ошибку выражения. Например, попытка доступа к четвертому элементу (индекс 3) в списке из трех элементов

= {"A", "B", "C"}{3}

приводит к ошибке:

Expression.Error: Элементов в перечислении было недостаточно для выполнения операции.

Получать доступ к первым или последним элементам в списке позволяют встроенные функции List.First и List.Last:

= List.First({"A", "B", "C"})

получим следующий результат:

A

Выражение

= List.Last({"A", "B", "C"})

возвращает следующий результат:

C

#### Функции List.X

Знание функций списков позволяет выполнять различные операции с содержимым столбца. Например, можно определить среднее значение в числовом столбце, а затем применить к столбцу фильтр на основе среднего. Для того чтобы начать работу со значениями в столбце, можно щелкнуть правой кнопкой мыши на заголовок столбца и выбрать *Детализация*. Результатом будет список. Теперь можно применить одну из встроенных функций. На рис. 6 показаны опции, доступные на вкладке *Преобразование*. Язык M предоставляет гораздо больше функций, чем показано на вкладке.

Язык M включает богатую библиотеку встроенных функций для списков. Вот некоторые из наиболее распространенных функций, разбитые на категории:

Информационные функции: описанная выше List.Count, List.IsEmpty, которая возвращает true, если список пуст, и false — в противном случае.

Функции выбора — возвращают выбранный элемент из списка или подсписка. Например, функция List.Select может применяться для фильтрации элементов в списке с соблюдением определенных критериев. Функции List.FirstN и List.LastN возвращают подсписок первых или последних N элементов исходного списка.

Функции преобразования — управляют списками. Например,

List.Transform(list as list, **transform** as function) as list

…позволяет изменять список с помощью функции *transform*. Можно увеличить каждое значение списка на единицу:

List.Transform({1, 2}, each \_ + 1)

List.Combine действует как оператор конкатенации (&).

Функции принадлежности — применяют логические операции к элементам в списке и возвращают true или false, если выполняются определенные условия. Например, функция List.Contains возвращает значение true, если в списке найдено определенное значение.

Операции над множествами — применяют операции над множествами для двух или более списков. List.Intersect позволяет найти общие элементы в двух списках, а List.Union дает возможность добавлять списки и удалять дубликаты. List.Difference возвращает все элементы в списке, недоступные во втором списке. Она применяется в именах столбцов, что позволит ссылаться на другие имена столбцов, помимо упомянутых.

Функции упорядочения — в дополнение к List.Sort, которая пригодна для чисел или аргументов любого типа, где можно определить функцию упорядочения, также позволяют извлекать элементы по внутреннему порядку значений, а не только по их порядку в списке. Например, List.Min и List.Max извлекают самые малые или самые большие значения, а List.MinN и List.MaxN извлекают n-е самые малые или самые большие значения.

Арифметические функции — в эту категорию входят функции, выполняющие усреднение, сложение и другие числовые операции. Например, List.Average и List.StandardDeviation могут работать с элементами чисел, датами и временем в списках одного типа. List.Sum оперирует с числами и значениями длительности, возвращая общую сумму значений в списке.

Функции генератора — в этой категории имеются функции, которые генерируют списки. Существует несколько основных функций, таких как List.Numbers, описанных ранее в этом разделе, и List.Dates, которая возвращает последовательный список дат от даты начала, количество дат и длительность между датами. В этой категории также присутствует расширенная функция List.Generate, которая может формировать список из исходного состояния.

Дополнительные сведения о функциях для операций со списками см. <https://docs.microsoft.com/ru-ru/powerquery-m/list-functions>

#### Записи

В Power Query обработка значений записей выполняется в двух распространенных сценариях: при загрузке неструктурированных данных, таких как файлы JSON, и при манипулировании данными в строках таблицы. Запись — это набор пар "ключ/значение". Ключи (их также называют именами полей) являются уникальными текстовыми значениями, а значения (их также называют значениями полей) могут принимать значения любого типа, базовые или сложные.

Инициировать новую запись можно, поместив разделенные запятыми пары между операторами открывающей и закрывающей скобок. При этом употребляется синтаксис ключ=значение. Например, запись

= [ID=123, Имя="Алиса", Город="Страна чудес"]

…вернет:



Рис. 8. Запись в строке формул можно инициализировать с помощью квадратных скобок []

При работе с таблицами можно перейти к просмотру записи, выбирая или указывая определенную строку с помощью фигурных скобок. Представьте, что имеется таблица сотрудников, определяемая идентификатором Employees. Четвертая строка в таблице Employees содержит информацию об Алисе, чей идентификатор — 123, а город — Страна чудес. Получить доступ к четвертой строке можно, указав в следующем выражении значение 3 в качестве индекса:

= Employees{3}

Если вы не уверены, в какой строке находится Алиса, можно выполнить поиск строки, имя сотрудника которой Алиса, или ID, равный 123.

= Employees{[Имя="Алиса"]}

= Employees{[ID=123]}

Извлечение строки в виде записи из таблицы завершится неудачей, если связанный столбец содержит несколько значений. Например, если имеются два сотрудника с именем Алиса или две строки с идентификатором 123.

Чтобы получить доступ к одному полю в записи, заключите имя поля в квадратные скобки:

= [ID=123, Имя="Алиса", Город="Страна чудес"][Имя]

Если применить фильтр к определенному столбцу, то сгенерится следующий код:

= Table.SelectRows(Employees, each ([Имя] = "Алиса"))

Данный код ссылается не на столбец в таблице, а на поле в записи. Каждая строка в таблице Employees представлена как запись при вызове функции Table.SelectRows. Ключевое слово each позволяет сократить выражение функции и сделать его понятнее. В данном случае функция получает запись в качестве аргумента и возвращает true, если имя сотрудника — Алиса.

#### Операторы записи

Для объединения двух записей или возвращения записи с обновленным значением в поле записи можно использовать оператор &:

= [ID=123, Имя="Алиса"] & [Город="Страна чудес"]

Возвращение новой записи с измененным значением в имеющееся поле данной записи является одной из наиболее распространенных операций. Для её реализации этого можно использовать оператор &. Следующее выражение изменяет ID Алисы с 123 на 456:

= [ID=123, Имя="Алиса"] & [ID=456]

Обратите внимание, что язык M не позволяет изменять имеющееся значение; можно изменять лишь копию значения в новом выражении. Например, следующее выражение, пытающееся изменить идентификатор Алисы с 123 на 456, вернет ошибку:

let

Источник = [ID=123, Имя="Алиса", Город="Страна чудес"],

Источник[ID]=456

in

Custom1



Рис. 9. Вы не можете изменить запись, получив доступ к ее полю

Для смены ID Алисы можно применить следующее выражение:

let

Источник = [ID=123, Имя="Алиса", Город="Страна чудес"],

Result = Источник & [ID=456]

in

Result

Для проверки равенства двух записей можно использовать операторы равенства (=) и неравенства (< >). Следующее выражение, введенное в строке формул, возвращает TRUE:

= [ID=123, Имя="Алиса"] = [Имя="Алиса", ID=123]

Порядок полей в записи не важен.

#### Функции Record.X

Возможно, вы обратили внимание, что на рис. 6 редактор Power Query предлагает только одну команду для преобразования записи на ленте – *В таблицу*. Эта команда эквивалентна функции *Record.ToTable*. Хотя лента не предлагает иных команд, в языке М можно найти несколько полезных встроенных функций Record.X.

Record.FieldCount возвращает число полей в записи, Record.HasFields возвращает true, если поля имеются, и false, если запись пуста. Пустая запись может быть представлена двумя скобками [].

Можно добавлять пары полей "ключ/значение" с помощью функции Record.AddField и обновлять значение поля в записи, возвращая новую запись, которая объединяет две записи с помощью функции Record.Combine:

= Record.Combine({[ID=123, Имя="Алиса"], [ID=456]})

Дополнительные сведения о функциях Record.X см. <https://docs.microsoft.com/ru-ru/powerquery-m/record-functions>

#### Табличный тип

Табличное значение – наиболее распространенный тип в Power Query. Большинство преобразований, которые формируются с помощью команд ленты, транслируются во встроенные функции Table.X. Можно получать доступ к столбцу таблицы, используя имя столбца, заключенное в квадратные скобки. Например, следующее выражение возвращает Column1 (как список) в таблице MyTable:

= MyTable[Column1]

Получить доступ к первой строке таблицы MyTable можно с помощью выражения:

= MyTable{0}

Получить доступ к значению в строке X и в столбце Y из таблицы MyTable можно с помощью одного из выражений:

= MyTable{X-1}[ColumnY]

= MyTable[ColumnY]{X-1}

Чтобы инициализировать таблицу, можно применить выражение #table:

= #table({"ID", "Имя", "Город"}, {{123, "Алиса",

"Страна чудес"}, {234, "Боб", "Страна чудес"}})



Рис. 10. Можно инициализировать таблицу в M, используя выражение #table.

Это удобный синтаксис для создания небольших таблиц, которые не требуют внешних наборов данных

В языке M имеется большое число встроенных табличных функций. Наиболее распространенные из них представлены на лентах Power Query. Подробнее см. <https://docs.microsoft.com/ru-ru/powerquery-m/table-functions>

Вот краткий перечень табличных функций, а также сценарии, в которых они обычно встречаются:

Функции Table.RowCount, Table.ColumnCount и Table.IsEmpty полезны для проверки размера и ширины таблицы, а также для обработки краевых случаев пустых таблиц. Эти функции удобно применять в сочетании с выражениями if.

Table.Firstvaiue – возвращает значение в ячейке первого столбца и первой строки.

Table.Profile – очень полезная функция для исследования данных, которая возвращает новую таблицу со статистическими атрибутами числовых столбцов: Min, Max, Average, Standard Deviation, Count, Null Count и Distinct Count) (минимальное, максимальное, среднее, стандартное отклонение, количество, нулевое число и различное количество).

Table.ColumnNames – возвращает имена столбцов таблицы в виде списка. Это чрезвычайно полезная функция, она понадобится нам для создания динамической ссылки на столбцы и предотвращения сбоев обновления при изменении имен столбцов.

Table.FromColumns, Table.FromList, Table.FromRecords и Table.FromRows – это расширенные функции, которые служат для создания новых таблиц из сложных структур данных, таких как список вложенных списков или список вложенных записей.

Table.ToColumns, Table.ToList, Table.ToRecords и Table.ToRows – преобразуют таблицы в сложные структуры данных.

#### Условия и выражения if

Одно из применений условия – фильтры в столбцах. Если пользовательский интерфейс кажется вам недостаточно гибким, используйте условия в строке формул. Вот базовое выражение, которое было создано в редакторе Power Query для фильтрации строк с именем Алиса в столбце Имя:

= Table.SelectRows(Employees, each ([Имя] = "Алиса"))

А вот варианты выражений для фильтрации всех строк, кроме, содержащих имя Боб:

= Table.SelectRows(Employees, each ([Имя] <> "Боб"))

= Table.SelectRows(Employees, each not ([Имя] = "Боб"))

Вот пример более сложной фильтрации строк, с именем Алиса и ID равным 123 или 456:

= Table.SelectRows(Employees, each ([Имя] = "Алиса" and ([ID]=123 or [ID]=456)))

Обратите внимание, что в выражении присутствует новая пара скобок для вкладывания условия [ID] = 123 and [ID] = 456. Если не использовать внутренние скобки, как показано в следующем выражении, то в конечных результатах вы сохраните других сотрудников с идентификатором 456:

= Table.SelectRows(Employees, each ([Имя] = "Алиса" and ([ID]=123 or [ID]=456)))

Другой сценарий, который может потребовать изменить код, созданный интерфейсом, – это использование условных столбцов.

#### Выражение if-then-else

Вот синтаксис выражения:

= if (условие) then A else B

Для облегчения просмотра кода рекомендуется разбивать выражение на отдельные строки:

= if (condition) then

A

else

B

Можно вкладывать дополнительные операторы if:

= if (условие\_1) then

A

else if (условие\_2) then

B

else

C

#### Выражение if внутри выражения let

Весьма распространена следующая ошибка. Допустим, проверяя запрос Query1, вы хотите вернуть null, если запрос пуст; в противном случае хотите вернуть первую строку Query1 (как запись):

let

if (Table.IsEmpty(Query1) then

Result = null

else

Result = Query1{0}

in

Result

Это выражение не будет работать. Внутри выражения let всегда нужно указывать идентификатор, за которым следует знак равенства, и только затем записывать выражение. Правильный код:

let

Result =

if (Table.IsEmpty(Query1) then

null

else

Query1{0}

in

Result

Выражения if можно использовать внутри функций, например:

(employeeName) =>

if employeeName = "Алиса" then

"Страна чудес"

else

"Реальный мир "

Если преобразования применяются к столбцам, можно включать выражения if в выражения M, изменяя код в строке формул. Представьте, что вы импортируете источник данных, который содержит столбец *Страна*, причем имеет место несоответствие регистра символов (для некоторых стран применяются строчные буквы, для других — прописные). На вкладке *Преобразование* вы задаете *Каждое Слово С Прописной*. В строке формул отразится:

= Table.TransformColumns(Источник,{{"Страна", Text.Proper, type text}})

Выясняется, что для некоторых стран изменения неудачны (например, сокращения US и UK были заменены на Us и Uk). Решить проблемы можно с помощью оператора if:

= Table.TransformColumns(Источник,{{"Country",

each if \_ = "US" or \_ = "UK" then \_

else

Text.Proper(\_), type text}})

#### Пользовательские функции

При работе с языком M помимо применения пользовательского интерфейса *Создать функцию* можно создавать пользовательские функции, используя пустой запрос или методом встраивания пользовательской функции в существующее выражение M в расширенном редакторе. Вот пример простой функции, которая увеличивает число на единицу:

(X) => X + 1

Результат X+1 является копией числа, поскольку в языке M все значения неизменны.

Внутри функции можно выполнить несколько шагов, используя выражение let. В следующем искусственном примере, при вызове функции с помощью ввода X, возвращается X+2, после выполнения двух последовательных шагов по увеличению значения на единицу:

(X) =>

let

Step1 = X + 1,

Step2 = Step1 + 1

in

Step2

Часто, при написании подобных выражений в расширенном редакторе, обнаруживается, что Power Query изменяет выражение и обертывает код в выражение let…in. Следующее выражение, измененное Power Query, эквивалентно предыдущему:

let

Источник =

(X) =>

let

Step1 = X + 1,

Step2 = Step1 + 1

in

Step2

in

Источник

Можно создать функцию, которая получает несколько аргументов. Вот пример функции, которая складывает два числа:

(X, Y) => X + Y

При создании пользовательской функции можно явно определять типы аргументов и тип возвращаемого значения:

(X as number, Y as number) as number => X + Y

Конечно, пользовательские функции могут принимать различные типы аргументов. В следующем примере текст и число принимаются в качестве аргументов, а возвращаемый результат представляет объединенный текст:

(X as text, Y as number) as text => X & Text.From(Y)

Функции могут определяться с помощью необязательных аргументов. В следующем выражении Y объявляется как необязательный аргумент. Когда функция будет вызвана без необязательного аргумента, Y получит нулевое значение:

(X, optional Y) => if (Y = null) then X else X + Y

Обратите внимание, что следующая реализация возвращает null вместо X, если Y не вызывается в качестве аргумента, потому что значение выражения X + null будет null:

(X, optional Y) => X + Y

Наконец, можно объявить функцию, которая не получает аргументов. Такие функции, например List.Generate, встречаются в расширенных сценариях, один из которых представлен ниже. Например, функция, которая не принимает аргументов и возвращает нуль:

() => 0

#### Вызов функции

Функции могут определяться внутри области выражения let или в виде отдельного запроса. Можно вызывать их в виде встроенных функций, задавая имя функции, за которым следует открывающая скобка, значения аргумента и закрывающая скобка. В примере пользовательская функция func, определенная в первой строке внутри выражения let, вызывается в третьей строке:

let

func = (X) => X+1,

Number1 = 1000,

Result = func(Number1)

in

Result

Функции могут быть переданы в качестве аргументов других функций. Когда пользовательская функция с одним аргументом передается в качестве аргумента другой функции, можно вызывать ее в качестве аргумента, указывая имя функции без скобок и аргумента. В примере запрос загружает таблицу цен из запроса *PricesTable* и значение в столбец *Цена* на 0,98 (скидка 2%), если цена больше 10:

let

FnGetPriceAfterDiscount =

(price) =>

if price > 10 then

price \* 0.98

else

price,

Источник = PricesTable,

DiscountedPrices = Table.TransformColumns(

Источник,

{{"Цена", FnGetPriceAfterDiscount, type number}}

)

in

DiscountedPrices

В этом выражении пользовательская функция FnGetPriceAfterDiscount вызывается внутри Table.TransformColumns. Поскольку пользовательской функции необходим один числовой аргумент, можно вызвать ее внутри Table.TransformColumns без указания скобок или точной ссылки на число в качестве аргумента. Например, функция:

= Table.TransformColumns(Источник,{{"Цена", FnGetPriceAfterDiscount, type number}})

…эквивалентна такому синтаксису:

= Table.TransformColumns(Источник,{{"Цена",
each FnGetPriceAfterDiscount(\_), type number}})

#### Выражение each

Выражение each разработано для того, чтобы позволить вам определять простые функции в удобной для пользователя форме.

Например, следующее выражение получает число и увеличивает его на единицу:

(X) => X + 1

Аналогично:

(\_) => \_ + 1

Это выражение использует подчеркивание в качестве имени аргумента. Или так:

each \_ + 1

В некоторых случаях можно даже не указывать подчеркивание в каждом выражении, если в аргументе имеются поля (например, таблица или запись). Следующие выражения фильтруют значения в столбце:

Table.SelectRows(Источник, ( )=> [Price] > 5)

или:

Table.SelectRows(Источник, each \_[Price] > 5)

Но поскольку вы ссылаетесь на поле в квадратных скобках, можно избежать подчеркивания:

Table.SelectRows(Источник, each [Price] > 5)

Последнее выражение наиболее удобно для восприятия.

#### Дополнительные темы

Не забывайте, что эта заметка лишь введение в язык М. Существует масса элементов языка, которые здесь не описаны. В последнем разделе мы затронем часть сложных тем, которые могут быть полезны, особенно если в дальнейшем вы решите освоить язык M как программист.

#### Обработка ошибок

Power Query располагает встроенными функциями, которые позволяют просматривать и устранять ошибки с помощью пользовательского интерфейса. Также можно заменить значения ошибок или удалить их. В дополнение к существующим пользовательским интерфейсам язык M позволяет программно обрабатывать случаи ошибок с помощью логики преобразования.

В языке M ошибка является признаком того, что вычисление выражения не может привести к значению. Если выражение приводит к ошибке, то запрос возвращает запись Expression.Error. Обычно, если не применяются явные обработчики ошибок, ваши запросы отменяются, выдавая сообщения об ошибках. Чтобы перехватить ошибки, не вызывая сбоя запроса, можно применить выражение try/else:

let

Result = try Источник[Column1] otherwise null

in

Result

В этом выражении try/otherwise используется для того, чтобы перейти к столбцу *Column1* в таблице *Источник*. Но если в таблице столбец *Column1* недоступен, вернется значение null. Если этого не сделать запрос прервется с ошибкой. Это выражение можно упростить:

let

Result = try Источник[Column1]

in

Result

Это выражение возвращает запись об ошибке:



Рис. 11. Конструкцию try можно использовать для возврата записи об ошибке, когда выражение некорректно

Возвращенная запись об ошибке может обрабатываться, как и любой шаг преобразования. При успешной реализации выражения try в результат входит запись, в которой HasError имеет значение false, а поле Value содержит значение успешного выражения. Следующая формула демонстрирует методику программной обработки ошибки без применения ключевого слова otherwise:

let

Attempt = try Источник[Column1],

Result =

if Attempt[HasError] then

null

else

Attempt[Value]

in

Result

Выражение возвращает null, если в таблице *Источник* *Column1* отсутствует, и эквивалентно выражению:

let

Result = try Источник[Column1] otherwise null

in

Result

Язык M позволяет изменить встроенное сообщение об ошибке. В следующем примере показано, как можно перезаписать значение *Message* записи об ошибке:

= try Источник[Column1] otherwise error "Column1 is missing from AdventureWorks database"

Дополнительную информацию об обработке ошибок см. <https://docs.microsoft.com/ru-ru/powerquery-m/error-handling>

#### Отложенные и немедленные вычисления

Внутри let язык M вычисляет выражения не в последовательном порядке, а использует отложенный подход. Каждый идентификатор вычисляется только по необходимости. Например, в следующем выражении *Step2* пропускается:

let

Step1 = 1,

Step2 = 10000000,

Step3 = Step1 + 1

in

Step3

Выражения списка и записи также обрабатываются с помощью отложенного вычисления. Остальные выражения вычисляются немедленно, и результат появляется, как только выражение встречается в процессе вычисления.

#### Циклы

Иногда задаются вопросом, почему в языке M отсутствуют циклы. Ответ не прост. Язык M разработан для выполнения последовательностей преобразований, и итерация уже встроена в структуру данных. Например, для замены значения в столбце не нужно писать код, который будет пересматривать каждую строку в таблице. Язык M выполняет итерацию за вас, когда применяются какие-либо функции из встроенных в таблицу.

Тем не менее иногда вам могут понадобиться явная логика и выполнение итерации последовательности для агрегирования данных. Один из наиболее распространенных сценариев — импорт данных с веб-сайтов, причем требуется номер страницы или смещение для импорта небольшой части данных. Чтобы разбивать на страницы по всему набору данных, нужно перебирать диапазон чисел.

Для выполнения нумерации страниц можно начать с пользовательской функции, которая импортирует определенную страницу по ее номеру. Например, созданная вами пользовательская функция FnLoadPageByNumber может получать постраничные результаты поиска с веб-сайта. Затем можно сформировать список чисел:

= {1..10}

Затем вы конвертируете этот список в таблицу, как показано на рис. 6. После переименования *Column1* в *PageNumber* следует преобразовать тип столбца *PageNumber* в *Текст*, а затем на вкладке *Добавить столбец* можно выбрать *Вызвать настраиваемую функцию* и вызвать *FnLoadPageByNumber* для столбца *PageNumber*. Наконец, можно пазвернуть новый столбец, и все объекты таблицы страниц будут добавлены вместе.

#### Рекурсия

Другой способ повторить последовательность шагов преобразования – задать рекурсию. Классический пример — числа Фибоначчи. Последовательность Фибоначчи характеризуется тем, что каждое число, кроме 0 и 1, является суммой двух предыдущих чисел в последовательности. Последовательность Фибоначчи 0, 1, 1, 2, 3, 5, 8, 13 и т. д.

В языке M можно реализовать рекурсию, используя знак @ для повторного вызова основной функции внутри функции, избегая ограничений внутренней области видимости:

let

Fibonacci =

(n)=>

if (n = 0) then

0

else if (n = 1) then

1

else

@Fibonacci(n-1) + @Fibonacci(n-2)

in

Fibonacci

Однако применение рекурсии в языке M не рекомендуется. Выполнение рекурсии обременительно и потребует много памяти на любом языке программирования, поскольку требуется, чтобы стек памяти операционной системы содержал все внутренние состояния рекурсии. В результате при запуске рекурсии для больших наборов данных или при наличии большого значения для вычисления числа Фибоначчи вы быстро получите переполнение стека памяти.

#### Функция List.Generate

Функция List.Generate позволяет реализовывать бесконечные типы итераций, включая вложенные циклы. Эта функция генерирует список значений из начального состояния и набора функций:

List.Generate(начальное значение функции, условие функции,
следующее выражение функции, необязательный селектор n,
обнуляемая функция) список

Например, вернем последовательность Фибоначчи, не превышающую 100:

= List.Generate(

()=>[Previous = 0, Current = 1],

each [Previous] + [Current] < 100,

each [

Current = [Previous] + [Current],

Previous = [Current]

],

each [Previous] + [Current]

)

List.Generate начинается с начальной функции, которая не получает входных данных и возвращает числовую запись Previous и Current. Previous присвоено значение 0, а Current – значение 1. Второй аргумент функции List.Generate, условная функция each [Previous] + [Current] < 100, гарантирует, что список генерится, пока условие выполняется. Когда Previous и Current достигают числа, превышающего или равного 100, генерация списка завершается.

Третьим аргументом служит функция, которая определяет следующее состояние и перемещает значения из полей Current и Previous в запись. Итак, в начальном состоянии значение Previous было 0, а Current – 1. На следующем этапе значение Current будет [Previous] + [Current], т.е. 1. Previous присваивается значение Current, которое было равно 1. На следующем шаге Current присваивается 2, а Previous – 1. Затем Current присваивается 3, а Previous – 2, и т.д.

Четвертый аргумент назначает элемент для списка вывода во время каждой итерации, что приводит к последовательности Фибоначчи по завершении итерации.

Функция List.Generate позволяет решать сложные задачи и обходиться без циклов. Тем не менее для реализации логики итерации следует сначала попытаться обойтись встроенными табличными функциями, поскольку List.Generate при выполнении отложенных вычислений может выполняться очень медленно и потреблять больше памяти. И все же для небольших наборов данных эта функция удобна. Для просмотра других примеров использования функции List.Generate, таких как вложенные циклы, разбиения на страницы курсора и умножение матриц см. <https://datachant.com/tag/list-generate/>

#### Функция List.Accumulate

Эта функция накапливает результат из списка ввода. Начиная с начального значения, функция применяет функцию аккумулятор и возвращает конечный агрегированный результат. Эта функция подойдет для перебора элементов в списке и постепенного объединения результатов для получения желаемого результата. Синтаксис:

List.Accumulate (список как список,
начальное значение любое, аккумулятор функция) любой тип

Следующее выражение просматривает список чисел и суммирует их:

= List.Accumulate({1, 2, 3, 4}, 0, (state, current) => state + current)

Функция начинается со списка ввода {1, 2, 3, 4} и 0 в качестве начального значения. На первом этапе функция получает нулевое значение и значение 1 как текущее (так как первый элемент в списке равен единице) и суммирует их. Результат, значение 1, присваивается новому состоянию. На втором этапе функция аккумулятора получает значение 1 в качестве состояния и значение 2 в качестве текущего (поскольку второй элемент в списке равен 2) и суммирует их. Значение 3 (1 + 2 = 3) присваивается новому состоянию. И т.д. Итерации завершаются после того, как функция запускается на последнем элементе в списке. Последнее значение функции выводится как результат выполнения функции. Результат = 10.

Рассмотрим последовательность Фибоначчи, возвращающей число 89. Для List.Accumulate нельзя определить условие динамического разрыва. Функция перебирает все элементы в списке. Вы инициализируете список числами от 0 до 9, но действительные значения списка не участвуют в подсчете:

let

Fibonacci =

List.Accumulate(

{0..9},

[PreviousNum = 0, CurrentNum = 1],

(state, current) =>

[

PreviousNum = state[CurrentNum],

CurrentNum = state[CurrentNum] + state[PreviousNum]

]

),

Result = Fibonacci[CurrentNum]

in

Result

В данном примере аргумент *начальное значение* определяется как запись из двух числовых полей: *PreviousNum* и *CurrentNum*. Функция возвращает обновленную версию этой записи, передавая значение *CurrentNum* в *PreviousNum* и суммируя *CurrentNum* и *PreviousNum* в новом значении *CurrentNum*.

В главе 11 функция List.Accumulate применяется для выполнения итеративной замены текста в столбце. Информация о дополнительных возможностях функции List.Accumulate см. <https://datachant.com/2016/06/02/power-query-list-accumulate-unleashed/>