Язык M Power Query. Типы – Число (Number)

В предыдущей заметке мы начали изучать типы значений (литералов) в языке M Power Query. И рассмотрели тип *текст* (text). Вы можете подумать, что работа с числами настолько простая, что хватит одного абзаца, чтобы ее описать. Однако есть одна тонкость, и если вы не будете осторожны, то арифметика может не дать ожидаемых результатов. Сначала мы рассмотрим синтаксис M для числовых литералов (значений), а потом остановимся на упомянутой тонкости. Кроме числовых литералов, числовой тип в M может быть назначен столбцам. Правильная настройка типа столбца может повысить производительность и сэкономить объем памяти, а также улучшить форматирование по умолчанию, используемое для значений столбца.[1]

Предыдущая заметка Следующая заметка

Синтаксис числовых литералов

Напомню, *литералом* в коде M называется фиксированное значение (подробнее см. определение в Википедии и в спецификации языка M). Числовые литералы могут быть представлены целыми числами: 0, 5, 17. Десятичными: 2.5, 0.5, .5. В коде M десятичный разделитель – точка. Перед десятичной точкой цифры являются необязательными, но после точки требуется хотя бы одна цифра. В экспоненциальной форме: 1.2e10, 1.2e-5, 1.2E2. Символ экспоненты *e* может быть строчным или прописным. В шестнадцатеричном виде: 0xFF, 0xff, 0XFF, 0xFF. Символы, используемые в шестнадцатеричных числах, также не чувствительны к регистру

Специальные числа

В дополнение к обычным поддерживаются специальные числа:

#infinity // возвращает результат деления 1/0

#nan // не число (not-a-number) возвращает результат деления 0/0

Отрицательные числа

Можно добавить знак минуса перед числом: -5, -5e-2, -#infinity (результат деления -1/0). Теоретически допустимо -#nan. Однако результат не меняется и знак минуса будет проигнорирован. #nan – это концепция без знака.

Только стиль

Различные стили, используемые для ввода чисел, – это только стили. Варианты предлагаются для удобства пользователей. Синтаксис каждого стиля создает выражение, которое возвращает числовое значение. Если два выражения возвращают одинаковые числа, их значения равны. Даже если выражения имеют разные синтаксические стили.

0x0A = 10 // выражение вернет *true*, так как оба литерала равны 10

Точность

Обсудив синтаксис числовых литералов, обратим внимание на потенциальное затруднение в работе с числами в M. Как арифметические (+, -, *, /), так и операторы равенства (=) *всегда* используют двойную точность. Двойная точность жертвует некоторой точностью ради эффективности. Десятичная точность – потенциально более медленная, но дающая более точные результаты – доступна, но должна быть явно запрошена. Обратимся к двум примерам…

Листинг 1[2]

0.1 + 0.2 // возвращает 0.30000000000000004

Листинг 2

10000000000000000 + 1 // возвращает 10000000000000000

Десятичная точность позволяет получить то, что вы ожидаете:[3]

---

[1] Заметка написана на основе статьи Ben Gribaudo. Power Query M Primer (Part 7): Types —Numbers. Если вы впервые сталкиваетесь с Power Query, рекомендую начать с Марк Мур. Power Query.

[2] Номер листинга соответствует номеру запроса в приложенном Excel файле.

[3] Проблема *0.1 + 0.2* характерна для большинства языков программирования, а не только для M. Есть даже сайт, на котором перечислены результаты на разных языках: https://0.30000000000000004.com/ Проблема связана с тем, что в двоичном исчислении, которое использует ПК, дроби 0,1 и 0,2 являются периодическими, и не могут быть вычислены точно.

Листинг 3
Value.Add(0.1, 0.2, Precision.Decimal) // возвращает 0.3

Листинг 4
Value.Add(1, 10000000000000000, Precision.Decimal) // возвращает 10000000000000001

Библиотечная функция Value.Add здесь используется для сложения с десятичной точностью. Её третий аргумент (необязательный) может принимать два значения: *Precision.Double* (значение по умолчанию) и *Precision.Decimal*. Существуют аналогичные функции для вычитания, умножения и деления. В каждой из них можно указать десятичную точность.

Различие между двойной и десятичной точностями может проявляться и при сравнении.

= Comparer.Equals(Comparer.Ordinal, 0.10000000000000001, 0.1) // возвращает true, так как осуществляет сравнение на основе двойной точности

= Value.Compare(0.10000000000000001, 0.1) /* возвращает 0;
третий аргумент опущен, а по умолчанию он равен *Precision.Double*;
функция Value.Compare возвращает значение -1, 0 или 1,
если первое значение соответственно меньше второго, равно ему или больше него */

= Value.Compare(0.10000000000000001, 0.1, Precision.Decimal) // возвращает 1

= 10000000000000000 = 10000000000000001 // возвращает true

= Value.Compare(10000000000000000, 10000000000000001, Precision.Decimal) // возвращает -1

Что происходит!?
Давайте сделаем шаг назад и обсудим, как в принципе компьютеры работают с точностью. Почему они требуют выбрать точность вместо того, чтобы всегда использовать максимально возможную точность.

Помните, еще в начальной школе вы узнали, что 1/3 не может быть точно представлена в виде десятичного числа. 1/3 = 0,333333333... Поскольку мы не можем записать бесконечное число троек, если мы хотим работать с дробью 1/3 в десятичном формате, мы должны выбрать уровень точности, который достаточно хорош для нашей ситуации. Например, если можно купить три одинаковых товара за доллар, мы могли бы представить цену как $0,33.

Вы заметили потерю точности? Одна штука стоит 1/3 доллара. Однако, если вы умножите $0,33 на 3, результат не будет равен $1. Вместо этого вы получите 0,99 доллара. Потери $0,01 можно было бы избежать, если бы мы работали с простыми дробями (1/3 доллара * 3 получается ровно 1 доллар). Тем не менее, удобнее работать с десятичным форматом. Настолько удобнее, что мы готовы немного потерять в точности.

Обратите внимание, потеря точности связана не с арифметикой. Неточность возникла из-за того, что используемая нами система счисления (десятичная) не могла точно представить дробь 1/3. Чтобы обойти это ограничение, было использовано приближение (0,33). И хотя арифметика было верной, результат был неточным, потому что он был основан на приближении.

Нечто подобное происходит и с компьютерами. Хотя компьютеры мыслят иначе, чем человек, но применяется тот же компромисс: удобство или точность. Поскольку запросы Power Query выполняются компьютерами, то в языке M мы встречаем ту же дилемму. В M по умолчанию работает двойная точность (соответствует стандарту IEEE 754-2008 для арифметики с плавающей запятой). Десятичная точность выше, но… расточительнее.

Что выбрать?
Если вы сравниваете расстояние между звездами, есть ли потребность в десятичной точности? В данном контексте небольшая потеря точности для повышения эффективности разумна. С другой стороны, вы, вероятно, не захотите, чтобы банк пренебрег точностью для повышения эффективности при расчете вашего банковского счета. В этом контексте точность имеет первостепенное значение, независимо от затрат на эффективность.

В M повышенная точность достигается за счет использования специальных функций, представленных выше. Нужно понимать, что стандартные операторы арифметики и сравнения

работают с двойной точностью, даже если исходные данные хранятся с десятичной точностью. Если вам нужна десятичная точность для операций, единственный способ получить ее – применить соответствующую функцию и явно указать параметр *Precision.Decimal*. В M десятки функций снабжены этим параметром.

Третья альтернатива

Если проблема связана с неточностью арифметики малых чисел, можно применить округление. Допустим в примере сложения 0.1 + 0.2 речь идет о долларах. Тогда значение имеют только две цифры после запятой. Округлим результат:

0.1 + 0.2 // возвращает 0.30000000000000004

= Number.Round(0.1 + 0.2, 2) // возвращает 0.30

Здесь второй параметр функции Number.Round (число 2) требует округления до двух знаков после запятой. К сожалению, округление поможет не всегда. В примере с простой дробью 1/3 можно повысить точность округления до трех знаков – 0,333. Тогда три предмета по $0,333 составят $0,999, которые при округлении до центов, составят $1,00 – ровно ту сумму, которую мы искали.

А теперь подсчитаем стоимость 10 предметов по 0,333 доллара = $3,33 доллара. Добавим еще десять предметов, а затем еще 10. Итого $3,33 + $3,33 + $3,33 = $9,99. Опять мы потеряли $0,01. Округление не выручило. Округление – это возможность, про которую следует помнить! Но не думайте, что оно решает все проблемы с математикой двойной точности.

Указание конкретных подтипов (также известных как фасеты)

При работе с таблицей в пользовательском интерфейсе Power Query вы, возможно, заметили, что для каждого столбца можно изменить тип. В редакторе PQ кликните правой кнопкой мыши на заголовке столбца и выберите *Изменить тип*.

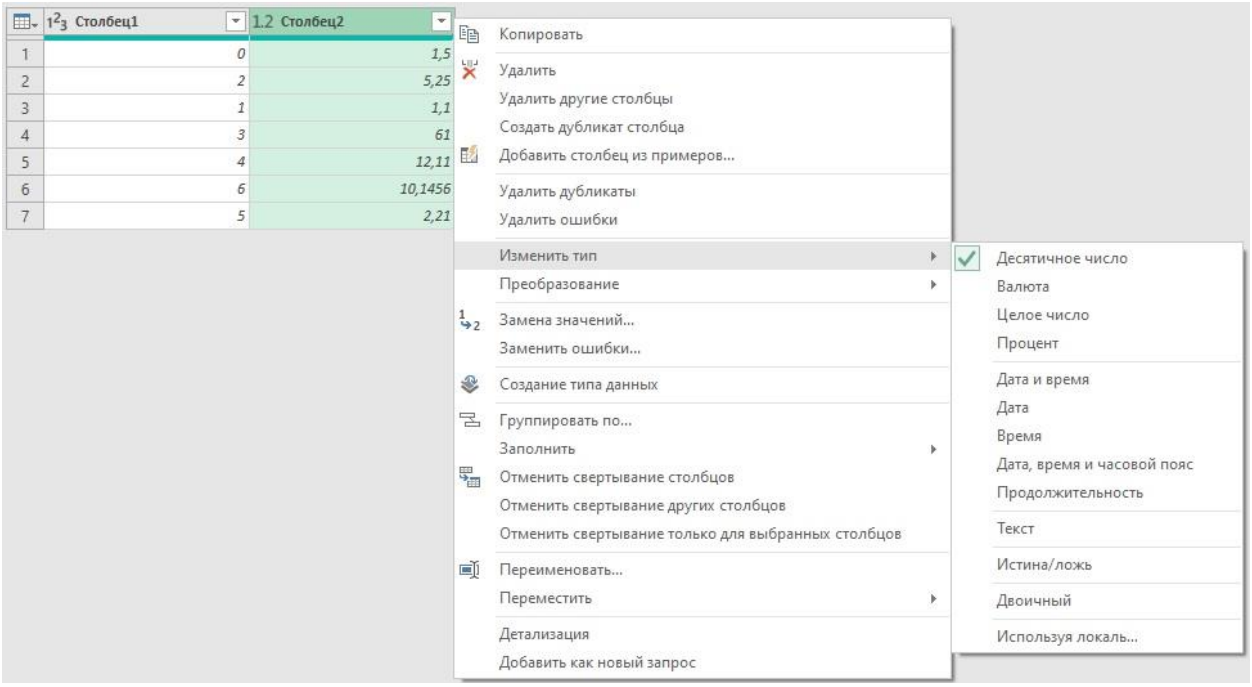

Рис. 1. Изменение типа столбца в редакторе Power Query

Для чисел допустимы четыре варианта:

- Десятичное число (с плавающей запятой)
- Валюта (отражает два знака после запятой, но можно задать точность до 4 цифр после запятой)
- Целое число (нет десятичной части)
- Процент (число, умноженное на 100 с добавлением знака процента)

Десятичное число соответствует базовому типу числа M; остальные являются подтипами (или фасетами) типа number. Выбор одного из этих параметров помечает столбец как содержащий значения определенного подтипа и преобразует данные в этом столбце в этот подтип. Если вы

укажете, что столбец представляет собой целое число, столбец будет помечен как содержащий целые числа, и любые числа с десятичной частью будут округлены. Например, 5,25 преобразуется в 5. Важно! Изменятся числа, хранящиеся в ячейках, а не только их отображение в редакторе.

Это определение подтипа может создать преимущества за пределами Power Query. Когда результирующие данные запроса будут помещены в Excel, последний использует информацию о типе столбца для обработки значений. Это повышает эффективность хранения и вычислений, а также создает удобный формат чисел. Например, если в PQ столбец получил подтип валюты, Excel может добавить знак доллара к числам (на моем ПК региональная настройка ru-RU и такое преобразование не работает).

Лучшие практики рекомендуют для всех столбцов таблицы, содержащих числа, устанавливать в PQ подтип (фасет), наиболее полно раскрывающий суть чисел.

Число десятичных знаков

К сожалению, задать желаемое число десятичных знаков (как в Excel) в PQ невозможно. Сравните формат чисел в *Столбец2* на рис. 1 и 2.

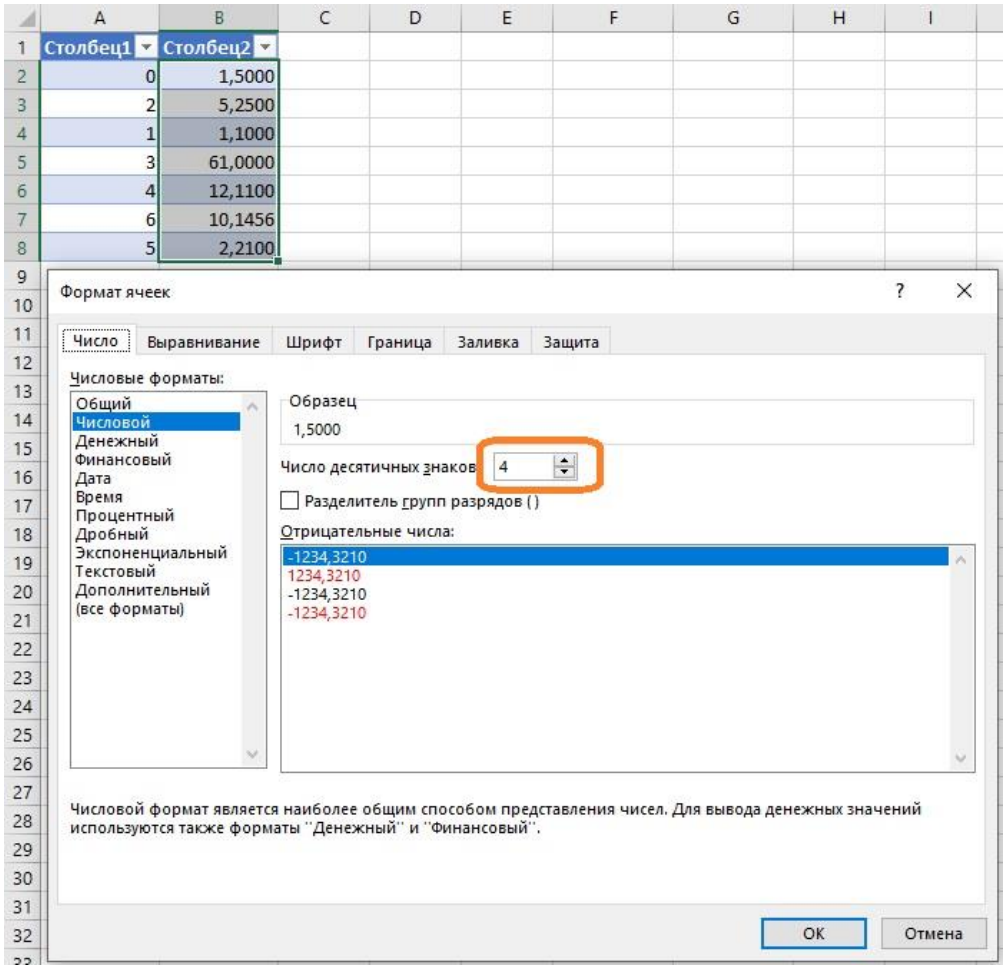

Рис. 2. Число десятичных знаков в Excel

PQ не удерживает нули после значащих цифр при использовании числового типа. Конечные нули фактически являются предпочтительным форматом, а не типом данных. PQ работает только с определенными типами данных. И типа *десятичное число с четырьмя знаками после запятой* в PQ нет.

Заключение

Раскрытие числовых литералов было быстрым и простым. Дискуссия о точности была долгой, но важной. Надеюсь, вы получили знания в этой области, чтобы принимать сознательные решения. И не забывайте всем числовым столбцам таблиц устанавливать подходящий подтип.

В следующий раз мы продолжим разговор о типах и остановимся на дате, времени и продолжительности.