Глава 10. Работа с датой, временем и длительностью

Это продолжение перевода книги Грегори Декер, Рик де Гроот, Мелисса де Корте. Полное руководство по языку M Power Query. Учитывая важность дат в отчетах, понимание того, как обрабатывать даты, время и длительности в Power Query, является критически важным навыком. Действительно, почти каждая модель данных Power BI имеет или должна иметь таблицу дат. Библиотека M включает более 100 функций для работы с датами, временем и длительностями. В этой главе мы рассмотрим многие из них, а также примеры, которые улучшат ваше понимание M.

Мои комментарии набраны с отступом.

Предыдущая глава Содержание Следующая глава

Рекомендуем выполнять примеры в редакторе Power Query. Так вы получите более глубокое их понимание. Исходные файлы включены в репозиторий GitHub.

Даты

Даты – это основа временн***о***го анализа. Существует около 60 функций для работы с датами. На самом деле даты – просто числа, представляющие количество дней до или после контрольной даты. Покажем это на примере. Создайте пустой запрос и в расширенном редакторе введите:

```
let
   Source = List.Generate(() => -10, each _ <= 10, each _ + 1)
in
   Source
```

Пройдите *Средства для списков –> Преобразование –> В таблицу*. В открывшемся окне кликните *OK*. Щелкните правой кнопкой мыши заголовок *Column1* и выберите *Создать дубликат столбца*. Щелкните правой кнопкой мыши заголовок столбца *Column1* и выберите *Изменить тип –> Дата*:

	Column1	Копия Column1
1	20.12.1899	-10
2	21.12.1899	-9
3	22.12.1899	-8
4	23.12.1899	-7
5	24.12.1899	-6
6	25.12.1899	-5
7	26.12.1899	-4
8	27.12.1899	-3
9	28.12.1899	-2
10	29.12.1899	-1
11	30.12.1899	0
12	31.12.1899	1
13	01.01.1900	2
14	02.01.1900	3
15	03.01.1900	4
16	04.01.1900	5
17	05.01.1900	6
18	06.01.1900	7
19	07.01.1900	8
20	08.01.1900	9
21	09.01.1900	10

Рис. 10.2. Преобразование столбца в дату

Число 0 соответствует контрольной дате 30.12.1899. Каждое положительное или отрицательное приращение на 1 увеличивает или уменьшает дату на один день. В M существуют ограничения на обработку. Невозможно представить даты до 1/1/1 и после 31/12/9999. Преобразование числа в дату имеет другие ограничения. Максимум тот же, 31/12/9999 (2 958 465), а вот минимум = -657 434, что соответствует 1/1/100. Эти ограничения кажутся произвольными. Они не соответствуют количеству байтов для хранения числа, и границы не являются симметричными относительно контрольной даты.

Для большинства практических задач ограничения не являются проблемой, но могут повлиять на расчет прошлых и будущих астрономических событий, например солнцестояния и равноденствия.

Тот факт, что даты представлены в виде чисел, возможно, не так важен в M, как, например, в DAX или Excel. То что даты являются числами, позволяет в DAX выполнять над ними арифметические операции. Однако в M попытка выполнить сложение дат приведет к ошибке:

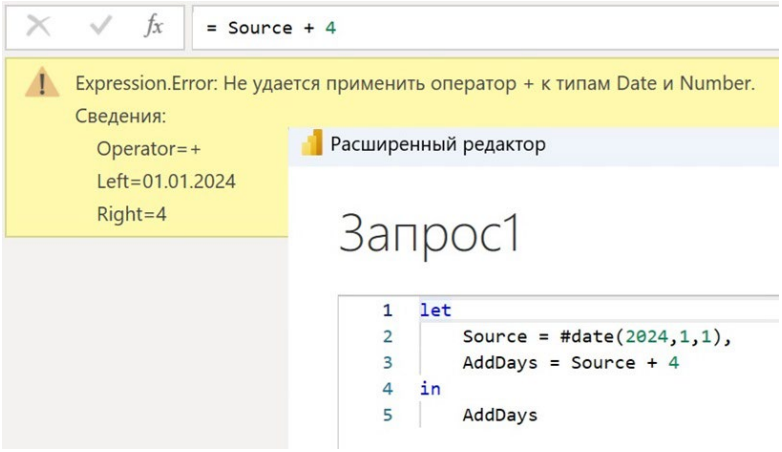

Рис. 10.3. Ошибка при добавлении числа к дате

Нужно сначала преобразовать дату в число, сложить два числа, и преобразовать число в дату:

```
let
   Source = #date(2024,1,1),
   AddDays = Number.From(Source) + 4,
   NewDate = Date.From(AddDays)
in
   NewDate
```

Этот код возвращает дату 5 января 2024 года. В окне предварительного просмотра мы увидим 05.01.2024 или 01.05.2024 в зависимости от региональных настроек.

Сначала мы задаем дату функцией *#date* с тремя аргументами: год, месяц и день. Функция *Number.From* приводит тип данных *date* к *number*, а затем функция *Date.From* выполняет обратное преобразование. Функция *Date.From* принимает значение *text*, *datetime*, *datetimezone* или *number* в качестве первого аргумента и пытается преобразовать это значение в дату. Если указан какой-либо другой тип данных или *Date.From* не может разобрать значение и определить дату, возвращается ошибка.

Необязательный второй параметр, *Culture*, позволяет указать региональные настройки. Например, если вы используете Power BI Desktop на компьютере с локальными настройками России (ru-RU), а получаете дату в виде текста из США (en-US) или Великобритании (en-GB), то следующие выражения позволят верно распознать даты:

Рис. 10.3а. Распознавание дат зависит от региональных настроек

С датами можно выполнять некоторые арифметические операции. Например…

```
let
   Date1 = #date(2024, 1, 1),
   Date2 = #date(2024, 1, 5),
   Diff = Date2 - Date1
in
   Diff
```

… вернет длительность *(Duration)* со значением 4.00:00:00, что соответствует четырем дням.

Рассмотрим некоторые функции дат, доступные в M. А начнем с запроса, возвращающего список дат для 2024, 2025 и 2026 годов:

```
let
   Source = List.Generate(
      () => #date(2024, 1, 1),
      each _ <= #date(2026, 12, 31),
      each Date.AddDays( _, 1)
   ),
   Table = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
   RenameColumn1 = Table.RenameColumns(Table,{{"Column1", "Date"}})
in
   RenameColumn1
```

На шаге *Source* с помощью функции *List.Generate* мы создаем список от начального *#date(2024, 1, 1)* до конечного *#date(2026, 12, 31)* значения с шагом один день *Date.AddDays()*. В M также есть функции для добавления других стандартных временных периодов: *Date.AddWeeks*, *Date.AddMonths*, *Date.AddQuarters* и *Date.AddYears*.

Существуют функции для извлечения компонентов даты, и мы можем добавить их в наш код:

```
   AddYearColumn  = Table.AddColumn(RenameColumn1, "Year", each Date.Year([Date])),
   AddMonthColumn = Table.AddColumn(AddYearColumn, "Month", each Date.Month([Date])),
   AddDayColumn   = Table.AddColumn(AddMonthColumn, "Day", each Date.Day([Date]))
```

Еще один способ извлечения компонентов даты – функция *Date.ToRecord*. Следующее выражение возвращает запись, содержащую пару поле/значение для года, месяца и даты:

```
Date.ToRecord(#date(2024, 2, 14))
```

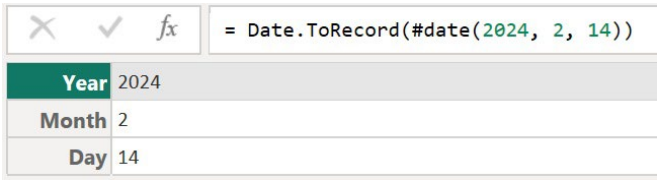

Рис. 10.3б. Извлечение компонентов даты в запись

Существуют также функции для вывода дополнительной информации для даты: квартал года, неделя года, неделя месяца, название дня и т. д.:

```
   AddQuarterColumn = Table.AddColumn( AddDayColumn, "Quarter", each
Date.QuarterOfYear([Date])),
   AddWeekOfYearColumn = Table.AddColumn( AddQuarterColumn, "Week of Year", each
Date.WeekOfYear([Date])),
   AddWeekOfMonthColumn = Table.AddColumn( AddWeekOfYearColumn, "Week of Month", each
Date.WeekOfMonth([Date])),
   AddDayOfWeekColumn = Table.AddColumn( AddWeekOfMonthColumn, "Day of Week", each
Date.DayOfWeek([Date])),
   AddDayOfWeekNameColumn = Table.AddColumn( AddDayOfWeekColumn, "Day of Week Name",
each Date.DayOfWeekName([Date])),
   AddMonthNameColumn = Table.AddColumn( AddDayOfWeekNameColumn, "Month Name", each
Date.MonthName([Date])),
   AddDaysInMonthColumn = Table.AddColumn( AddMonthNameColumn, "Days In Month", each
Date.DaysInMonth([Date])),
   AddLeapYearColumn = Table.AddColumn( AddDaysInMonthColumn, "Is Leap Year", each
Date.IsLeapYear([Date]))
```

В стандартной библиотеке есть также семейство функций *Date.StartOf* и *Date.EndOf* для каждого временного периода – день, неделя, месяц, квартал, год. Функции этого семейства возвращают значение даты и времени, указывающее начало или конец периода. Например, выражение…

```
Date.EndOfMonth(#date(2024, 2, 14))
```

… возвращает 29.02.2024, т.е., дату окончания месяца для 14 февраля 2024 года.

Для семейств функций *Date.StartOf* и *Date.EndOf* выходные данные зависят от типа входных данных. При подаче на вход типа *date*, *datetime* или *datetimezone* в качестве выходных данных создается соответствующий тип. Например, выражение …

Date.EndOfMonth(#datetime(2024, 2, 14, 0, 0, 0))

… вернет значение 2024-02-29T23:59:59.9999999 с типом данных *datetime*, а выражение …

Date.EndOfMonth(#datetimezone(2024, 2, 14, 0, 0 , 0, -5, 0))

… 2024-02-29T23:59:59.99999999-05:00 с типом данных *datetimezone* с часовым поясом Восточного стандартного времени (EST) или Всемирное координированное время (UTC) минус 5.

Для типов данных *date* существуют три дополнительных семейства функций: *IsInCurrent*, *IsInNext* и *IsInPrevious*. Каждое из этих семейств включает функции для стандартных временных периодов: дня, недели, месяца, квартала и года. Эти функции возвращают логические значения в зависимости от того, находится ли входная дата, в пределах текущего периода. Например, года для функции *IsInCurrentYear*. Расчеты основаны на текущей дате и времени системы, поэтому при запуске Power BI Desktop на локальном компьютере и в службе Power BI с клиентом, находящимся в другом часовом поясе, могут возвращаться разные результаты.

Семейства функций *IsInNextN* и *IsInPreviousN* также покрывают все временные периоды. Кроме первого параметра типа *date*, функции имеют второй параметр, указывающий количество временных единиц (положительных или отрицательных), чтобы определить, находится ли текущая дата в указанном диапазоне. Например, функция Date.IsInNextNWeeks(#date(2024, 2, 14), 2) определяет, находится ли дата в диапазоне двух следующих недель.

Дополнительно есть функция *IsIn* для текущего года – *Date.IsInYearToDate*. Она возвращает *true* или *false* в зависимости от того, находится ли переданная в функцию дата в пределах текущего года до текущей даты.

Наконец, есть функция, которая преобразует тип *date* в *text* – *Date.ToText* с необязательными аргументами *Format* и *Culture*. Рик де Гроот ведет список кодов, которые могут быть использованы в параметре *Format*:

```
= Table.AddColumn(#"Добавлен пользовательский объект", "1 февраля 2003", each
    Date.ToText( #date(2023, 2, 1), [Format], "ru-RU"))
```

	Format	31 декабря 2023	1 февраля 2003
1	%d	31	1
2	dd	31	01
3	ddd	Вс	Ср
4	dddd	воскресенье	среда
5	%M	12	2
6	MM	12	02
7	MMM	дек	фев
8	MMMM	Декабрь	Февраль
9	%y	23	23
10	yy	23	23
11	yyy	2023	2023
12	yyyy	2023	2023
13	yyyyy	02023	02023
14	M	31 декабря	1 февраля
15	Y	Декабрь 2023	Февраль 2023
16	d	31.12.2023	01.02.2023
17	D	31 декабря 2023 г.	1 февраля 2023 г.
18	gg	наша эра	наша эра

Рис. 10.3в. Коды аргумента *Format* для типов данных *date*

Таким образом, код…

Date.ToText(#date(2024, 2, 14), "dddd, MMMM %d yyyy gg")

… возвращает "Wednesday, February 14, 2024 A.D.". Этот результат может варьироваться в зависимости от настроек культуры. В коде для отображения рис. 10.3в я добавил "ru-RU", чтобы вернуть значения для региональных настроек.

Функция *Date.FromText* наоборот преобразует текстовое представление даты в тип данных *date*, и также поддерживает параметры *Format* и *Culture*.

Таблица календаря

Как упоминалось ранее, почти все модели данных Power BI имеют или должны иметь таблицу дат. Часто они берутся из хранилища данных. В случае отсутствия последнего можно использовать знаменитую функцию Extended Date Table Мелиссы де Корте. Код функции слишком длинный, чтобы включать его в книгу, но функция *Melissa de Korte's Date Table* есть в файле для этой главы.

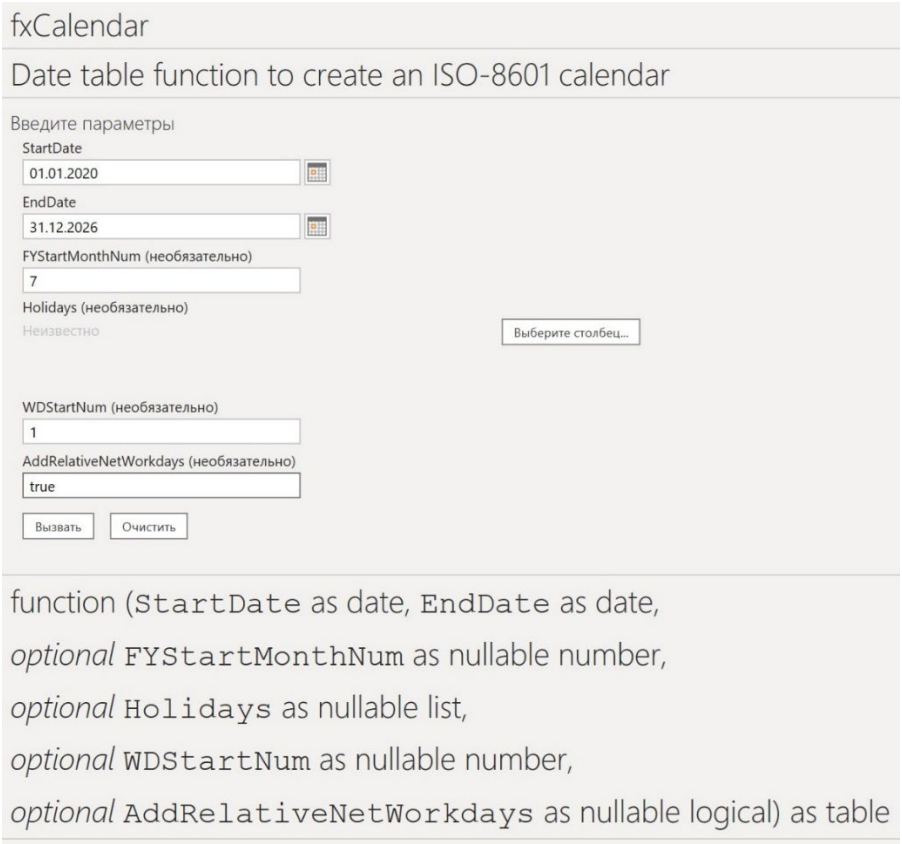

Рис. 10.4. Функция расширенной таблицы дат *Melissa de Korte's Date Table Function*

Для работы функции требуются два параметра, *StartDate* и *EndDate*. Также есть четыре необязательных параметра:

- *FYStartMonthNum*: номер месяца, с которого начинается финансовый год; 1, если опущено.
- *Holidays*: выберите запрос (и столбец), содержащий список праздничных дат.
- *WDStartNum*: переключение нумерации дней недели с 0-6 (по умолчанию) на 1-7, введя 1.
- *AddRelativeNetWorkdays*: если значение равно *true*, в таблицу дат добавляется столбец *Relative Networkdays*

На рис. 10.4 показана конфигурация для создания таблицы дат между 2020 и 2026 годами, где финансовый год начинается в июле, дни недели пронумерованы от 1 до 7, а столбец *Relative Networkdays* включен в таблицу дат. Праздники не указаны. При нажатии кнопки *Вызвать* создается запрос *Вызванная функция* с кодом:

```
let
   Источник = #"Melissa de Korte's Date Table Function"(
      #date(2020, 1, 1), #date(2026, 12, 31), 7, null, 1, true)
in
   Источник
```

Запрос можно переименовать в *Календарь*. Для дат с 01.01.2020 по 31.12.2026 был создан 61 столбец. Обратите внимание:

- Столбец *Fiscal Week* начинается с понедельника и может содержать менее 7 дней в первой и/или последней неделе финансового года.
- В столбце *IsWeekDay* не учитываются праздничные даты.
- В столбце *IsBusinessDay* учитываются необязательные даты праздников.
- Столбцы *IsPYTD* и *IsPFYTD* сравнивают предыдущий номер дня года с текущим номером дня года, поэтому даты не совпадают в високосные годы.
- Столбцы *Fiscal* не будут добавлены, если параметр *FYStartMonthNum* опущен.

Одной из замечательных особенностей этой таблицы дат является реализация смещений в столбцах *CurrYearOffset*, *CurrQuarterOffset*, *CurrMonthOffset*, *CurrWeekOffset* и *CurrDayOffset*. Эти столбцы значительно упрощают логику вычислений с датами и временем в DAX, как показано в статье Брайана Джулиуса Time Intelligence In DAX: How To Dynamically Select Starting Period.

Другие форматы дат

Даты, описанные здесь, используются в большинстве стран мира и основаны на так называемом григорианском календаре. Григорианский календарь был введен в 1582 году папой Григорием XIII в качестве замены юлианского календаря. Тем не менее, существуют и другие календари и форматы дат, используемые в различных уголках мира и в некоторых научных областях.

Юлианские дни

Номер юлианского дня (*Julian Day Number, JDN*) – это порядковый номер дня в непрерывной системе отсчета, начинающейся с 1 января 4713 года до н. э. JDN используется в различных научных и вычислительных областях, в том числе:

- *Измерение времени*: Юлианские дни предоставляют непрерывную и равномерную систему измерения времени, которая широко используется в астрономии и астрофизике для указания даты и времени наблюдений, вычислений и событий. Это упрощает вычисления интервалов между двумя датами.
- *Эфемериды*: Астрономические эфемериды, которые предоставляют положения небесных объектов во времени, часто используют юлианские дни для своей временной шкалы.
- *Универсальная система датирования*: Юлианские дни предлагают единую и непрерывную систему датирования, полезную в исторической хронологии. Это позволяет легко рассчитывать временные интервалы и особенно ценно для исторических событий, охватывающих длительные периоды.
- *Данные с временной меткой*: В геофизике и науках о Земле данные, собираемые с течением времени, такие как сейсмические события или экологические наблюдения, часто записываются с использованием *JDN*. Это позволяет упорядочить события хронологически.
- *Вычислительная эффективность*: Юлианские дни удобны для компьютерных алгоритмов и программирования, так как они предоставляют одно число, упрощающее вычисления дат и времени. Это особенно полезно при работе с большими наборами данных или выполнении вычислений, связанных с временными интервалами.
- *Навигационные расчеты*: В навигации юлианские дни используются для расчета положений небесных тел и определения времени небесных событий. Они предоставляют стандартизированный способ представления времени для навигационных целей.
- *Расчеты високосных дней и годов*: Юлианские дни упрощают расчеты, связанные с високосными днями и годами, поскольку они представляют собой непрерывный счет дней, не зависящий от различий в длине месяцев или годов.

В файле этой главы есть функция *fnJulianDay* для преобразования григорианских дат в *JDN*. Следующий запрос возвращает таблицу за 1987 год, содержащую столбец для юлианских дней:

```
let
 Source = List.Generate(
  () => #date(1987, 1, 1),
  each _ <= #date(1987, 12, 31),
  each Date.AddDays(_, 1)
 ),
 Table = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
 RenameColumns = Table.RenameColumns(Table, {{"Column1", "Date"}}),
 AddJDNColumn = Table.AddColumn(RenameColumns, "Julian Day", each fnJulianDay([Date]))
```

```
in
 AddJDNColumn
```

	Date	Julian Day
1	01.01.1987	2446796,5
2	02.01.1987	2446797,5
3	03.01.1987	2446798,5
4	04.01.1987	2446799,5
5	05.01.1987	2446800,5
6	06.01.1987	2446801,5
7	07.01.1987	2446802,5
8	08.01.1987	2446803,5
9	09.01.1987	2446804,5
10	10.01.1987	2446805,5
11	11.01.1987	2446806,5
12	12.01.1987	2446807,5
13	13.01.1987	2446808,5
14	14.01.1987	2446809,5
15	15.01.1987	2446810,5
16	16.01.1987	2446811,5

Рис. 10.4а. Григорианские даты и юлианские дни

В файле также есть обратная функцию *fnGregorianDate* для преобразования юлианских дней в даты, эквивалентные григорианскому календарю.

Форматы дат ERP Oracle

Некоторые системы хранят даты в разных форматах. В качестве примера можно привести несколько версий популярной программы J. D. Edwards Enterprise Resource Planning (ERP), которая в настоящее время принадлежит Oracle. Эта система хранит даты с помощью флага столетия в следующем формате: cyyddd. Здесь c – флаг века, где 0 соответствует 1900, а 1 – 2000. Год хранится в виде двузначного числа от 0 до 99. День хранится как номер дня года. Для преобразования этого формата даты в тип данных *date* воспользуйтесь функцией *fnJDEToDate*.

Пользовательские функции даты

Создание пользовательских функций для работы с датами может сэкономить массу времени. Мы обсудим несколько таких функций.

Будни

Существует множество способов учитывать течение времени. Некоторые организации отслеживают ход времени в рабочих днях. Пользовательская функция *CategorizeDay* классифицирует дни на рабочие и нерабочие:

```
(date as date) => if Date.DayOfWeek(date, Day.Monday) <= 4 then "Workday" else "Weekend"
```

Она принимает один аргумент – дату. Функция *Date.DayOfWeek* возвращает число от 0 до 6, где 0 – понедельник. Необязательный параметр, перечисление *Day.Monday*, переключает выходные данные со значения по умолчанию 0 для воскресенья на 0 для понедельника.

Пользовательская функция *WorkingDays* определяет количество рабочих дней между двумя датами:

```
(startDate as date, endDate as date, optional holidays as list) =>
 let
   holidayDates = if holidays = null then {} else holidays,
   days = Number.From(Duration.From(endDate - startDate)) + 1,
   allDates = List.Dates(startDate, days, #duration(1, 0, 0, 0)),
   workingDays = List.Select(
    allDates,
    each Date.DayOfWeek(_, Day.Monday) <= 4 and not List.Contains(holidayDates, _)
   )
 in
```

```
  List.Count(workingDays)
```

Функция принимает три параметра: дату начала, дату окончания и необязательный список праздничных дней. Сначала мы проверяем наличие необязательного параметра, и, если он отсутствует, создаем пустой список. Затем мы вычисляем количество дней, которые нужно включить в наш диапазон дат от *startDate* до *endDate*, вычитая две даты, преобразуя полученную длительность в число и добавляя 1. Затем мы используем функцию *List.Dates* для создания списка дат, включенных в наш диапазон (также могли бы использовать *List.Generate*). Следующее выражение выбирает сгенерированные элементы списка, используя ту же логику, что и наша функция *CategorizeDay*. Даты, которые есть в списке *holidayDates* не выбираются. Наконец, мы подсчитываем элементы в отфильтрованном списке.

> Учитывая ранее подсмотренный прием, код можно немного упростить: изменить *List.Contains(holidayDates, _)* на *List.Contains(holidays ?? {}, _)* и удалить шаг *holidayDates*.

Скользящее среднее

Распространенным сценарием в аналитике данных является расчет скользящего среднего для определенной метрики. Это можно сделать как с помощью DAX, так и на стороне Power Query. Пользовательская функция *MovingAvg* добавляет столбец со скользящим средним в таблицу:

```
(sourceTable as table, valueColumn as text, windowSize as number) =>
 let
  BufferedValues = List.Buffer(Table.ToColumns(Table.SelectColumns(sourceTable, valueColumn)){0}),
  addIndex = Table.AddIndexColumn(sourceTable, "Index", 1, 1, Int64.Type),
  movingAvg = Table.AddColumn(
   addIndex,
   "MovingAvg",
   each
    if [Index] >= windowSize then
     List.Average(List.Range(BufferedValues, _[Index] - windowSize, windowSize))
    else
     null,
   type number
  ),
  removeIndex = Table.RemoveColumns(movingAvg, {"Index"})
 in
  removeIndex
```

Функция принимает три параметра: таблицу, имя столбца для которого вычисляется среднее и размер окна скользящего среднего.

> Chat GPT. В контексте вычисления скользящего среднего *размер окна* (*window size*) определяет количество последовательных данных, которые будут учитываться при вычислении среднего значения.

Функция помещает в буфер переданный в аргументе столбец, добавляет столбец индекса, отсчитываемый от 1. Затем добавляет столбец скользящего среднего *MovingAvg*, с параметром *windowSize*, и возвращает исходную таблицу вместе со столбцом *MovingAvg*.

Предположим, что функция используется в таблице фактов, содержащей ежедневную информацию о продажах с размером окна 30, вызов этой функции вернет таблицу со скользящим средним, рассчитанным для каждой строки за последние 30 дней. Функцию можно адаптировать к скользящей сумме или иному вычислению, заменив *List.Average* на *List.Sum* или иной агрегатор.

Перейдем к обсуждению времени.

Время

Время используется в отчетах не так часто, как даты. Тем не менее, время играет важную роль при оценке производительности сотрудников или скорости операций в цепочках поставок. Время – число долей суток. Чтобы убедиться в этом, создайте запрос...

```
let
```

```
    Source = List.Generate(() => 0, each _ <= 24, each _ + 1),
    Table = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
    AddMultiplicationColumn = Table.AddColumn(
        Table,
        "Multiplication",
        each [Column1] / 24,
        type number
    ),
    DuplicateColumn = Table.DuplicateColumn(
        AddMultiplicationColumn,
        "Multiplication",
        "Multiplication - Copy"
    ),
    ChangeType = Table.TransformColumnTypes(DuplicateColumn, {{"Multiplication - Copy", type time}}),
    RenameColumns = Table.RenameColumns(ChangeType, {{"Multiplication - Copy", "Time"}})
in
    RenameColumns
```

… который возвращает таблицу:

	Column1	Multiplication	Time
1	0	0	0:00:00
2	1	0,041666667	1:00:00
3	2	0,083333333	2:00:00
4	3	0,125	3:00:00
5	4	0,166666667	4:00:00
6	5	0,208333333	5:00:00
7	6	0,25	6:00:00
8	7	0,291666667	7:00:00
9	8	0,333333333	8:00:00
10	9	0,375	9:00:00
11	10	0,416666667	10:00:00
12	11	0,458333333	11:00:00
13	12	0,5	12:00:00
14	13	0,541666667	13:00:00
15	14	0,583333333	14:00:00
16	15	0,625	15:00:00
17	16	0,666666667	16:00:00
18	17	0,708333333	17:00:00
19	18	0,75	18:00:00
20	19	0,791666667	19:00:00
21	20	0,833333333	20:00:00
22	21	0,875	21:00:00
23	22	0,916666667	22:00:00
24	23	0,958333333	23:00:00
25	24	1	Error

Рис. 10.5. Таблица часов

Число 0 в столбце *Column1* соответствует полуночи. В столбце *Multiplication* мы разделили *Column1* на 24, чтобы получить долю дня. Эти значение при преобразовании в тип *time* возвращают часы. Обратите внимание, что значение 1 возвращает ошибку.

Тип *time* имеет ограничения для арифметических операций. Вы можете складывать или вычитать значения времени, но не можете делать это без предварительного преобразования значений в тип *number*.

Аналогично датам, есть несколько функций для извлечения компонентов времени:

```
AddHourColumn = Table.AddColumn(RenameColumns, "Hour", each Time.Hour([Time])),
AddMinuteColumn = Table.AddColumn(AddHourColumn, "Minute", each Time.Minute([Time])),
AddSecondColumn = Table.AddColumn(AddMinuteColumn, "Second", each Time.Second([Time]))
```

Также существует функция *Time.Record*, которая извлекает компоненты времени в запись. Тип данных времени имеет конструктор *#time*, принимающую три параметра: час, минута и секунда. Можно предположить, что добавление времени к дате приведет к получению типа данных *datetime*, но это не так. Выражение…

#date(2024, 2, 14) + #time(17, 0, 0)

… приводит к ошибке *Expression.Error: Не удается применить оператор + к типам Date и Time*.

А вот оператор конкатенации работает. Выражение…

#date(2024, 2, 14) & #time(17, 0, 0)

… вернет 14.02.2024 17:00:00.

Функции *Time.StartOfHour* и *Time.EndOfHour* возвращают начало и конец часа. Тип результата зависит от типа входных данных – *time*, *datetime* или *datetimezone*. Например:

Time.EndOfHour(#time(13,10,5)) // возвращает 13:59:59.9999999

Функции *Time.From* и *Time.FromText* преобразуют входное значение в тип данных *time*. Функция *Time.From* принимает значения *text*, *datetime*, *datetimezone* или *number* (≥ 0 и < 1). Обе функции включают необязательный параметр *Culture*.

Функция *Time.ToText* преобразует тип данных *time* в текст. Представлением управляют необязательные параметры *Format* и *Culture*. Более подробную информацию о различных вариантах форматирования можно найти на веб-сайте Рика де Гроота Gorilla BI.

Создание таблицы времени

Для создания отчетов могут потребоваться измерения времени с минутной или секундной детализацией. В приложенном файле есть запрос *Time Table Minute*:

```
let
   Source = List.Generate( () => 0, each _ < 24 * 60, each _ + 1 ),
   Table = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
   Divide = Table.TransformColumns(Table, {{"Column1", each _ / 24 / 60, type number}}),
   ChangeType = Table.TransformColumnTypes(Divide,{{"Column1", type time}}),
   RenameColumns = Table.RenameColumns(ChangeType,{{"Column1", "Time"}}),
   AddHourColumn = Table.AddColumn(RenameColumns, "Hour", each Time.Hour([Time])),
   AddMinuteColumn = Table.AddColumn(AddHourColumn, "Minute", each Time.Minute([Time])),
   AddSecondColumn = Table.AddColumn(AddMinuteColumn, "Second", each Time.Second([Time])),
   AddStartOfHourColumn = Table.AddColumn(
      AddSecondColumn, "Start of Hour", each Time.StartOfHour([Time])
   ),
   AddEndOfHourColumn = Table.AddColumn(
      AddStartOfHourColumn, "End of Hour", each Time.EndOfHour([Time])
   )
in
   AddEndOfHourColumn
```

Запрос возвращает таблицу с минутной детализацией:

	Time	Hour	Minute	Second	Start of Hour	End of Hour
1	0:00:00	0	0	0	0:00:00	1:00:00
2	0:01:00	0	1	0	0:00:00	1:00:00
3	0:02:00	0	2	0	0:00:00	1:00:00
4	0:03:00	0	3	0	0:00:00	1:00:00
5	0:04:00	0	4	0	0:00:00	1:00:00
6	0:05:00	0	5	0	0:00:00	1:00:00
7	0:06:00	0	6	0	0:00:00	1:00:00
8	0:07:00	0	7	0	0:00:00	1:00:00
9	0:08:00	0	8	0	0:00:00	1:00:00
10	0:09:00	0	9	0	0:00:00	1:00:00
11	0:10:00	0	10	0	0:00:00	1:00:00

Рис. 10.5а. Таблица времени с минутной детализацией

Если нужна детализация на секундном уровне, изучите код запроса *Time Table Second*.

Классификация смен

Можно создать функцию для классификации времени на первую, вторую и третью смены. Предполагая, что первая смена длится с 8 утра до 4 вечера, вторая – с 4 вечера до полуночи, а третья – с полуночи до 8 утра, следующая функция классифицирует время, переданное в качестве параметра, как попадающее в определенную смену (запрос *ShiftClassification*):

```
( actualTime as time ) =>
let
   hourEndShift3 = Time.Hour(#time(8, 0, 0)),
   hourEndShift1 = Time.Hour(#time(16, 0, 0)),
   hour = Time.Hour(Time.StartOfHour( actualTime )),
   shift = if hour < hourEndShift3 then "Third" else if hour < hourEndShift1 then "First" else "Second"
in
   shift
```

Дата и время

Поскольку даты – это целые числа, основанные на эталонной дате, а время – десятичные числа, выражающие доли дня, тип данных *datetime* может быть представлен в виде десятичного числа, где целая часть представляет дату, а десятичная – время. Например, следующее выражение…

```
DateTime.From(45336.5)
```

… возвращает 14.02.2024 12:00:00

Отдельные компоненты даты и времени можно извлечь из типа данных *datetime* с помощью функций *DateTime.Date* и *DateTime.Time*. Функция *DateTime.ToRecord* возвращает запись с парами поле-значение для года, месяца, дня, часа, минуты и секунды.

Для *datetime* доступен конструктор *#datetime*. Следующее выражение…

```
#datetime(2024, 2, 14, 17, 0, 0)
```

… вернет 14.02.2024 17:00:00.

Функции, *DateTime.LocalNow* и *DateTime.FixedLocalNow*, возвращают текущую дату и время системы. *DateTime.LocalNow* возвращает разные значения при каждом обращении к функции в пределах одного запроса. Несколько вызовов *DateTime.FixedLocalNow* – возвращают одно и то же значение.

Существуют семейства функций *IsInCurrent*, *IsInNext* и *IsInPrevious*. Каждое из семейств включает отдельные функции для компонентов времени (часа, минуты и секунды). Семейства *IsInNext* и *IsInPrevious* включают подсемейства *IsInNextN* и *IsInPreviousN*. Все эти функции возвращают логическое значение в зависимости от того, соответствует ли значение *datetime* условиям.

Функция *DateTime.FromFileTime* преобразует время файла Windows в стандартный формат *datetime*. Файловое время Windows записывается как общее количество 100-наносекундных интервалов с 12:00 1 января 1601 г. н.э. по UTC. Эта отправная точка используется в качестве эталона для согласованности записей времени файлов в системах Windows. Таким образом, выражение…

```
DateTime.FromFileTime(133524036000000000)
```

… возвращает 14.02.2024 20:00:00, если вы находитесь в Москве.

Функция *DateTime.AddTimeZone* добавляет сведения о часовом поясе к типу данных *datetime*, возвращая тип данных *datetimezone*. Продолжая предыдущий пример, следующее выражение…

```
DateTime.AddZone(DateTime.FromFileTime(133524036000000000), 3, 0)
```

… возвращает тип данных *datetimezone*, который добавляет информацию о часовом поясе для Москвы 14.02.2024 20:00:00 +03:00.

Часовые пояса

Несмотря на то, что часовые пояса в высшей степени практичны и относительно просты для понимания, они вызывали проблемы с программированием с самого начала развития вычислительной техники. К счастью, язык M включает в себя тип данных *datetimezone* с множеством функций, которые облегчают работу с датой, временем и часовым поясом.

В отличие от типов данных *date*, *time* и *datetime*, типы данных *datetimezone* не могут быть представлены только в виде десятичных чисел. Фактически, создание значения *datetimezone* с помощью *DateTimeZone.From(45336.5)* добавляет дополнительные сведения о часовом поясе локальной системы, возвращая 14.02.2024 12:00:00 +03:00 для ПК, работающего в Москве.

Для типа данных *datetimezone* существует два дополнительных элемента: *ZoneHours* и *ZoneMinutes*. Их можно извлечь с помощью *DateTimeZone.ZoneHours* и *DateTimeZone.ZoneMinutes*. Функция *DateTimeZone.ToRecord* возвращает запись:

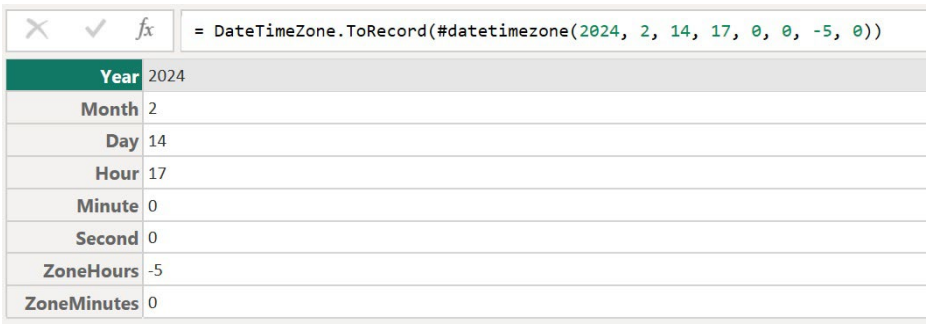

Рис. 10.5б. Запись, возвращаемая функцией *DateTimeZone.ToRecord*

Функция *#datetimezone* принимает восемь обязательных параметров.

Функции *DateTimeZone.LocalNow* и *DateTimeZone.FixedLocalNow* возвращают текущее локальное системное время, а функции *DateTimeZone.UtcNow* и *DateTimeZone.FixedUtcNow* – текущее время системы по UTC. Функция *DateTimeZone.ToUtc* преобразует значение *datetimezone* во время UTC. Например, выражение…

DateTimeZone.ToUtc(DateTimeZone.From(45336.5))

… возвращает 14.02.2024 9:00:00 +00:00 в Москве. Здесь исходное значение полудня 14 февраля 2024 года переносится на 3 часа назад. Это имеет смысл, учитывая, что часовой пояс Москвы равен +3, т.е. что часовой пояс Москвы на 3 часа опережает время UTC.

Функция *DateTimeZone.ToLocal* преобразует значение *datetimezone* в локальное время компьютера, вычисляющего выражение. Например, выражение…

DateTimeZone.ToLocal(DateTimeZone.From("14.02.2024 17:00:00 +00:00", "ru-RU"))

… возвращает 14.02.2024 20:00:00 +03:00 при вычислении на компьютере, работающем в Москве.

Функция *DateTimeZone.RemoveZone* удаляет сведения о часовом поясе, возвращая тип данных *datetime*. Функция *DateTimeZone.SwitchZone* заменяет сведения о часовом поясе часовым поясом, заданным вторым и третьим параметрами этой функции. Обратите внимание, что функция *SwitchZone* не просто заменяет часовой пояс, а преобразует значение *datetimezone* в новый часовой пояс. Таким образом, выражение…

DateTimeZone.SwitchZone(DateTimeZone.From(45336.5), 0, 0)

… эквивалентно примеру с использованием функции *ToUtc* и возвращает точно такое же значение, 14.02.2024 9:00:00 +00:00. Таким образом, функцию *SwitchZone* можно рассматривать как более универсальную функцию, чем более специфические функции *ToUtc* и *ToLocal*.

Функция *DateTimeZone.FromFileTime* работает аналогично функции *DateTime*, но добавляет сведения о часовом поясе, извлеченные из локальной системы, вычисляющей выражение.

Исправление времени обновления данных

В отчеты часто включают время последнего обновления данных. Служба Power BI записывает время обновления в часовом поясе региона клиента Power BI. Это может не совпадать с часовым

поясом страны, в которой пользователи просматривают отчет. Проблему можно устранить, применяя пользовательскую функцию для вычисления времени обновления, сохранения даты и времени в виде таблицы в модели данных, а затем отображения этой даты и времени на страницах отчета:

```
(Summer_GMT_Offset as number, Winter_GMT_Offset as number) =>
let
   UTC_DateTimeZone = DateTimeZone.UtcNow(),
   UTC_Date = Date.From( UTC_DateTimeZone ),
   StartSummerTime = Date.StartOfWeek( #date( Date.Year( UTC_Date ) , 3 , 31 ), Day.Sunday ),
   StartWinterTime = Date.StartOfWeek( #date( Date.Year( UTC_Date ) , 10, 31 ), Day.Sunday ),
   UTC_Offset = if UTC_Date >= StartSummerTime and UTC_Date < StartWinterTime then
Summer_GMT_Offset else Winter_GMT_Offset,
   CET_Timezone = DateTimeZone.SwitchZone( UTC_DateTimeZone, UTC_Offset)
in
   CET_Timezone
```

Функцию разработал Рик де Гроот. Она принимает два параметра: летнее и зимнее смещение от UTC. Во-первых, *DateTimeZone.UtcNow* получает дату, время и часовой пояс системы. Функция *Date.From* извлекает дату. Предполагая, что летнее время начинается в последнее воскресенье марта, а зимнее – в последнее воскресенье октября, следующие две строки кода определяют даты начала летнего и зимнего времени. Смещение вычисляется на основе текущей даты в сравнении с датами перехода на летнее и зимнее время. Затем время настраивается с помощью функции *DateTimeZone.SwitchZone*.

Длительность

В отличие от модели данных службы Analysis Services, язык M включает тип данных для длительности. Отсутствие типа данных *duration* в службах Analysis Services означает, что вычисления, включающие длительность лучше обрабатывать в Power Query, а не в столбцах или мерах DAX. Однако также важно понимать, что столбцы типа данных *duration* в Power Query преобразуются в десятичные числа при загрузке в модель данных. Выражение…

```
let
   Source = Duration.From("0.01:00:00"),
   #"Converted to Table" = #table(1, {{Source}}),
   #"Changed Type" = Table.TransformColumnTypes(#"Converted to Table",{{"Column1", type
duration}})
in
   #"Changed Type"
```

… возвращает таблицу с одной строкой и одним столбцом с типом данных *duration* со значением 1 час. В модель данных загрузится значение 0,04166666666666667, т.е. 1 час/24 часа. Чтобы отобразить формат длительности в отчетах, необходимо выполнить преобразование данных, которое либо окончательно превратит значение в текст с помощью функции FORMAT DAX, либо применить пользовательский формат, чтобы сохранить значение в виде числа.

Тип данных *duration* имеет конструктор – *#duration*, принимающий четыре параметра: дни, часы, минуты и секунды. Следующее выражение…

```
#duration(5, 2, 15, 33)
```

… возвращает длительность 5.02:15:33.

Тип данных *duration* можно добавить к типу данных *date*, *time*, *datetime* или *datetimezone* с помощью оператора сложения (+). Таким образом, следующее выражение…

```
#date(2024, 2, 9) + #duration(5, 2, 15, 33)
```

… возвращает дату 14.02.2024.

А выражение…

```
#time( 17, 0, 0) + #duration(5, 2, 15, 33)
```

… возвращает время 19:15:33.

Компоненты длительности можно извлечь с помощью функций *Duration.Days*, *Duration.Hours*, *Duration.Minutes* и *Duration.Seconds*. Функция *Duration.ToRecord* возвращает запись с 4 полями: *Days*, *Hours*, *Minutes* и *Seconds*.

Имеется семейство функций *Duration.Total*, приводящее длительность к дробному числу дней, часов, минут или секунд. Например, выражение…

```
Duration.TotalSeconds(Duration.From("1.2:10:12"))
```

… возвращает значение 94 212.

Поскольку движок службы Analysis Services не поддерживает тип данных длительности, рекомендуется выполнить вычисления длительности в Power Query, а затем использовать одну из функций семейства *Duration.Total* для преобразования столбца *duration* в число (например, секунд) перед загрузкой в модель данных. После сохранения в секундах существует множество стандартных методов DAX для преобразования значения в длительность. См., например, заметку Chelsie Eiden's Duration в коллекции быстрых мер сайта сообщества Power BI.

Продолжительность работы

Аналогично вычислению рабочих дней между двумя датами, можно создать пользовательскую функцию для вычисления продолжительности работы между двумя значениями времени:

```
( startTime as time, endTime as time, workStartTime as time, workEndTime as time) =>
let
   actualStartTime = if startTime < workStartTime then workStartTime else startTime,
   actualEndTime = if endTime > workEndTime then workEndTime else endTime,
   workDuration = actualEndTime - actualStartTime
in
   workDuration
```

Функция принимает четыре параметра: фактическое время начала и окончания работы, а также время начала и окончания рабочего дня. Операторы *if* сравнивают фактическое время с нормативным. Продолжительность равна разности измененного времени начала и окончания.

Для преобразования количества дней и рабочих часов в день в длительность можно использовать функцию:

```
(days as number, hoursPerDay as number) =>
   days * hoursPerDay * #duration(0, 1, 0, 0)
```

Саммари

В этой главе мы рассмотрели дату, время, дату и время, часовые пояса и длительность. Мы привели несколько практических примеров, включая знаменитую расширенную таблицу дат Мелиссы де Корте. Мы также показали примеры расчета количества рабочих дней между двумя датами и как рассчитать скользящее среднее. Мы продемонстрировали создание таблицы измерения времени на минутном и секундном уровнях детализации.

В следующей главе мы продолжим наше исследование M, и подробно обсудим сравнение, замену, объединение и разделение значений.