Содержание

Об авторах

Марко Руссо и **Альберто Феррари** являются основателями сайта sqlbi.com, на котором регулярно публикуют статьи по Microsoft Power BI, Power Pivot, DAX и SQL Server Analysis Services. Они работают с DAX с момента появления первой бета-версии Power Pivot в 2009 году, и за это время сайт sqlbi.com стал одним из главных поставщиков статей и обучающих материалов по DAX. Их семинары, как очные, так и в удаленном режиме, являются основным источником вдохновения и обучения для энтузиастов DAX.

Марко и Альберто проводят консультации и обучение в области бизнес-аналитики (BI) с использованием технологий от Microsoft. За время своей практики они написали несколько книг и статей по Power BI, DAX и Analysis Services. Также они обеспечивают сообщество DAX постоянной поддержкой в виде новых материалов для сайтов daxpatterns.com, daxformatter.com и dax.guide.

Кроме того, Марко и Альберто регулярно выступают на крупнейших международных конференциях, включая Microsoft Ignite, PASS Summit и SQLBits. Связаться с Марко можно по электронной почте marco.russo@sqlbi.com, а с Альберто – alberto.ferrari@sqlbi.com.

Предисловие

Вы можете не знать наших имен. Мы проводим дни за написанием кода для программ, которые вы ежедневно используете в своей работе. Мы – часть команды разработчиков Power BI, SQL Server Analysis Services и… да, мы приложили руку к созданию языка DAX и движка VertiPaq.

Язык, который вы собираетесь изучать, читая эту книгу, является нашим детищем. Мы провели не один год, работая над ним, улучшая движок и находя способы для ускорения оптимизатора в попытке превратить DAX в простой и лаконичный язык, призванный значительно облегчить жизнь и повысить эффективность труда аналитиков данных.

Но позвольте, это ведь предисловие к книге, так что больше ни слова о нас! Почему же мы пишем вводное слово к изданию Марко и Альберто – парней из SQLBI? Хотя бы потому, что при поиске информации по DAX в сети новички постоянно выходят на их статьи. Они начинают читать их, увлекаются языком и в конечном счете, мы надеемся, проникаются уважением к результатам нашего тяжелого труда. Мы познакомились с Марко и Альберто довольно давно и сразу отметили их глубочайшие познания в области SQL Server Analysis Services. И они были в числе первопроходцев нового языка DAX, изучали его и старались применить на практике.

Их статьи, заметки и посты в блогах стали источником познания для многих тысяч людей. Мы пишем код, но не так много времени уделяем обучению разработчиков тому, как им пользоваться. А Марко и Альберто как раз из числа тех, кто распространяет знания о DAX по миру.

Книги этих парней являются мировыми бестселлерами в данной области, а написание подробного руководства по DAX ознаменовало собой историческую веху в популяризации языка, который мы сотворили и к которому питаем самые нежные чувства. Мы пишем код, они пишут книги, а вы изучаете DAX, привнося в свой бизнес невиданную аналитическую мощь. Вместе же мы делаем общее дело – извлекаем максимум аналитической информации из данных. И это здорово!

Мариус Думитру (Marius Dumitru),
руководитель отдела разработки Power BI

Кристиан Петкулеску (Cristian Petculescu),
главный разработчик Power BI

Джеффри Ванг (Jeffrey Wang),
управляющий отдела разработки ПО

Кристиан Уэйд (Christian Wade),
старший руководитель проекта

Благодарности

Написание второго издания этой книги заняло у нас целый год – на три месяца больше, чем первого. Это было долгое и увлекательное путешествие вместе с самыми разными людьми из разных широт и часовых поясов, результатом которого стала эта книга. Мы хотели бы поблагодарить за помощь в ее написании очень многих людей, но понимаем, что всех перечислить просто не сможем. Так что просто скажем спасибо всем, кто так или иначе способствовал выпуску книги – возможно, даже не подразумевая об этом. Комментарии в блогах, посты на форумах, обсуждения по почте, общение на технических конференциях, анализ различных сценариев – все это было для нас очень полезно, и многие из ваших идей нашли отражение в данной книге. Также мы выражаем огромную признательность всем студентам наших курсов: обучая вас, мы развивались сами!

И все же отдельных людей мы не можем не выделить особо за их заметный вклад в написание книги.

Начать список персональных благодарностей мы хотим с Эдуарда Меломеда. Именно он вдохновил нас на написание книги. Если бы не страстная дискуссия с ним несколько лет назад, итогом которой стало содержание нашей первой книги по Power Pivot, написанное на салфетке, мы могли бы вовсе не отправиться в путешествие по миру DAX.

Также мы очень признательны издательству Microsoft Press и его сотрудникам, внесшим весомый вклад в наш труд.

Написание книги отнимает немало времени, но еще больше времени уходит на подготовительные исследования. Люди, которых мы называем «инсайдерами SSAS (SQL Server Analysis Services)», очень помогли нам в подготовке к путешествию. Кроме того, стоит особо отметить нескольких людей из Microsoft, уделивших нам свое время при описании важных концепций, касающихся Power BI и DAX. Это Мариус Думитру (Marius Dumitru), Джеффри Ванг (Jeffrey Wang), Акшай Мирчандани (Akshai Mirchandani), Кристиан Саковски (Krystian Sakowski) и Кристиан Петкулеску (Cristian Petculescu). Ваша помощь была неоценима, парни!

Также мы хотим поблагодарить Амира Нетца (Amir Netz), Кристиана Уэйда (Christian Wade), Ашвини Шарма (Ashvini Sharma), Каспера Де Йонга (Kasper De Jonge) и Т. К. Ананда (T. K. Anand) за многочисленные дискуссии касательно этого проекта. Эти люди помогли нам при выборе стратегического направления как в этой книге, так и в карьере в целом.

Отдельные слова признательности хотелось бы сказать в адрес женщины, изрядно поработавшей над нашим английским. Клэр Коста (Claire Costa) тщательно вычитала исходный текст книги и привела его в порядок. Клэр, мы высоко ценим твою помощь! Спасибо!

Последнюю персональную благодарность мы адресуем нашему техническому рецензенту Даниилу Маслюку (Daniil Maslyuk), проверившему все без

исключения фрагменты кода, примеры и ссылки в книге. Он обнаружил все ошибки и опечатки, которые мы не заметили, а его комментарии всегда были по делу. Результат совместной работы превзошел все наши ожидания. И если в книге ошибок оказалось меньше, чем в исходном тексте, в этом заслуга Даниила. А оставшиеся опечатки – исключительно наша вина.

Спасибо, ребята!

Поддержка

Если вам требуется дополнительная помощь или информация, вы можете обратиться по адресу: https://MicrosoftPressStore.com/Support.

Отметим, что услуги по поддержке программного обеспечения Microsoft по этому адресу не оказываются. Если вам требуется помощь такого плана, перейдите на сайт http://support.microsoft.com.

Оставайтесь с нами

Давайте продолжим общение! Заходите на наш Twitter: @MicrosoftPress.

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте http://dmkpress.com/authors/publish_book/ или напишите в издательство: dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты автор-

ских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Предисловие ко второму изданию

Когда мы задумались о том, что пришло время обновить книгу, мы посчитали, что сделать это будет легко: в конце концов, в языке DAX за это время произошло не так много изменений, а теоретическая ценность первого издания не была утрачена. Мы полагали, что ограничимся лишь заменой рисунков с Excel на Power BI и добавим что-то по мелочи тут и там. Как же мы ошибались!

Приступив к обновлению первой главы, мы очень быстро поняли, что хотим переписать в ней почти все. И так на протяжении всей книги. Так что вы держите в руках не просто второе издание, а совершенно новую книгу.

И причина таких серьезных обновлений отнюдь не в том, что за это время как-то кардинально изменился язык или описываемые в книге инструменты. Скорее, мы как авторы и преподаватели изменились – надеемся, в лучшую сторону. Мы научили языку DAX тысячи людей по всему миру, неустанно работали со своими студентами и старались максимально доходчиво объяснять им самые сложные темы. В конечном счете мы нашли совершенно новый способ донесения до читателя информации о любимом нами языке.

Мы расширили количество примеров в этом издании, чтобы показать, как работает на практике то, что вы сначала изучаете в теории. При этом мы старались максимально упростить примеры без ущерба для полноты описываемой ситуации. Мы боролись с редактором за возможность увеличить количество страниц в книге, чтобы она могла вместить все темы, которые мы собирались осветить. Но мы не изменили главный посыл книги, состоящий в том, что вам не нужно владеть языком DAX, чтобы ее читать, хотя она и не предназначена для тех, кому просто нужно решить пару задачек на DAX. Скорее, эта книга для людей, желающих в полной мере овладеть искусством программирования на DAX и познать весь его потенциал и сложность.

Если вы действительно хотите использовать всю мощь языка DAX, то должны приготовиться к длительному путешествию с чтением этой книги от корки до корки и возвращением к ней с целью отыскать то, что ускользнуло от вас при первом прочтении.

Предисловие к первому изданию

В нашем авторском активе немало материалов, посвященных языку DAX. Это и книги по Power Pivot и *табличной модели SQL Server Analysis Services* (SSAS Tabular), и посты в блогах, и статьи, и экспертные доклады, и, наконец, книга, посвященная *шаблонам* (patterns) в DAX. Так зачем нам было писать (а вам, надеемся, читать) еще одну книгу по DAX? Неужели об этом языке так много можно узнать? Мы, разумеется, считаем, что да.

Первое, что редактор стремится выведать у переводчика в момент начала работы над новой книгой, – это предполагаемое количество страниц. И это не праздный интерес – на объем книги завязана и цена, и весь производственный процесс, включая распределение ресурсов издательства, и прочее. Практически все, что связано с книгой, так или иначе зависит от количества страниц в ней. Нас как авторов это немало расстраивает. Всякий раз, когда мы садились писать книгу, мы должны были выделять приличное место для описания программных продуктов, будь то Power Pivot для Microsoft Excel или SSAS Tabular, и только затем переходить к самому языку DAX. И каждый раз мы оставались недовольны тем, что нам вновь не удалось рассказать о DAX в объеме, в котором планировали. В конце концов, не писать же книгу по Power Pivot объемом в тысячу страниц – такая книга на полке магазина напугает кого угодно.

Так что нам приходилось раз за разом писать о SSAS Tabular и Power Pivot, а проект книги по DAX продолжал пылиться в ящике стола. Но однажды мы открыли этот ящик и решили не думать о том, что включать в новую книгу, – она должна была быть посвящена DAX целиком и полностью. Результат этого решения вы держите в руках.

Здесь вы не прочитаете о том, как создать вычисляемый столбец или какое диалоговое окно использовать для установки того или иного свойства. Эта книга – не пошаговое руководство по Microsoft Visual Studio, Power BI или Power Pivot для Excel. В ней вы сможете с головой погрузиться в мир DAX – начиная с самых основ и заканчивая техническими нюансами, позволяющими оптимизировать код и модель.

В процессе написания мы полюбили каждую страницу нашей книги. Мы столько раз ее перечитывали, что буквально выучили наизусть. При этом мы добавляли новый контент всякий раз, когда считали это уместным, не боясь превысить лимит на объем книги, и ничего не сокращали только для того, чтобы остаться в рамках дозволенного. Одновременно мы все больше узнавали о DAX и наслаждались своими открытиями.

Но есть еще один вопрос: зачем вам вообще читать руководство по DAX?

Признайтесь, вы подумали так, впервые попробовав поработать в Power Pivot или Power BI! И вы не одиноки. В свой первый раз мы подумали точно так

же. DAX предельно прост! Он очень похож на Excel! Более того, обладая опытом работы с одним языком программирования или запросов, вы наверняка привыкли изучать другие языки, просто глядя на примеры и сопоставляя его синтаксис с уже знакомыми вам шаблонами. Мы сами допустили эту ошибку и не хотим, чтобы через это прошли и вы.

DAX – очень мощный язык, который используется во все большем количестве аналитических инструментов. Потенциал его велик, но некоторые его концепции непросто понять, идя в своих рассуждениях от частного к общему. Например, изучение контекста вычисления в DAX требует обратного подхода – от общего к частному. Вы начинаете с теории, а после этого обращаетесь к соответствующим практическим примерам. Именно такой подход, именуемый дедукцией, характерен для этой книги. Мы понимаем, что многим не по душе подобный метод обучения – они предпочитают идти от практики к теории, сначала разобравшись с конкретной задачей, а затем подводя под нее определенные теоретические выводы. Если вы сторонник такого подхода, эта книга не для вас. Мы уже писали практические книги по DAX, полные примеров и без описания того, как работает та или иная формула и почему тот или иной подход к коду будет более оптимальным. Их вполне можно использовать как справочник функций DAX. Цель написания данной книги была совершенно иной. Мы хотели, чтобы вы в полной мере овладели языком DAX. Все примеры в этой книге демонстрируют определенное поведение, а не решают конкретные проблемы. Если вы сможете воспользоваться формулами из этой книги в своей модели, что ж, отлично. Но помните, что это лишь приятное дополнение, но никак не основная цель написания примеров. И всегда читайте описание к примерам, чтобы не угодить в ловушку. С целью обучения мы часто приводим в них не самые оптимальные способы решения задач.

Мы искренне надеемся, что вам придется по душе наше совместное путешествие в мир DAX и во время чтения книги вы получите не меньшее удовольствие, чем мы – во время ее написания.

Для кого предназначена эта книга?

Если вы лишь время от времени используете DAX, эта книга, скорее всего, не для вас. Есть множество книг с простым введением в инструменты, использующие DAX, и в сам язык – начиная с самых основ и заканчивая базовыми понятиями программирования. Мы хорошо осведомлены об этом, поскольку и сами писали такие книги.

Если же вы настроены на освоение DAX очень серьезно и с далеко идущими намерениями, эта книга – ваш выбор! При этом вы можете ничего не знать об этом языке. В этом случае, правда, не надейтесь на усвоение сложных концепций с первого раза. Мы советуем прочитать книгу от корки до корки, а затем, по мере приобретения опыта, возвращаться к наиболее сложным главам для повторного прочтения. Вполне вероятно, что описанные в них техники откроются для вас по-новому.

Язык DAX может быть полезен для людей, занятых в самых разных областях: пользователям Power BI может понадобиться написать формулы на DAX в своих

моделях данных, специалистам по работе в Excel язык DAX может пригодиться в совместном использовании с надстройкой Power Pivot, а профессионалы в области *бизнес-аналитики* (business intelligence – BI) могут применять код на DAX в своих решениях вне зависимости от их масштаба. В этой книге мы попытались представить информацию, которая может оказаться полезной для всех перечисленных категорий специалистов. При этом некоторые главы (в особенности касающиеся оптимизации работы DAX) могут быть предназначены для профессионалов в области бизнес-аналитики, поскольку содержат сложную техническую информацию. Но мы считаем, что пользователям Power BI и Excel также может быть полезно узнать возможности оптимизации выражений DAX для достижения максимальной эффективности функционирования модели.

И наконец, мы хотели написать книгу не только для чтения, но и для обучения. Поначалу мы будем стараться все объяснять максимально простым языком – с самого нуля. Но с усложнением концепций мы будем постепенно уходить от простоты и приближаться к реальности. DAX – простой язык, но использовать его не так легко. Нам потребовалось несколько лет, чтобы в полной мере освоить все его премудрости. Не ожидайте, что вы все это усвоите за несколько дней беззаботного чтения. Эта книга потребует от вас максимальной концентрации внимания. Взамен мы предлагаем вам шанс освоить всю глубину DAX и стать настоящим экспертом в этой области.

Как мы представляем себе нашего читателя?

Мы предполагаем, что наш читатель обладает базовыми знаниями в области Power BI и имеет представление об анализе данных. Если у вас есть опыт использования языка DAX, тем лучше для вас – быстрее прочитаете первые главы. Но в целом для чтения книги навыки работы с этим языком не обязательны.

В книге встречаются фрагменты кода на MDX и SQL, но вам не нужно знать эти языки – они приводятся здесь лишь для сравнения способов написания выражений. Если вы не поймете, что написано в этих фрагментах кода, ничего страшного. Значит, вам это не нужно.

В наиболее сложных главах книги мы затронем вопросы параллелизма, доступа к памяти, использования центрального процессора и другие сложные темы, с которыми далеко не все должны быть знакомы. Опытные разработчики почувствуют себя в этих главах в своей тарелке, а пользователи Power BI и Excel могут быть немного напуганы. Но без этих технических нюансов просто не обойтись при описании темы оптимизации кода на DAX. И хотя эти сложные главы больше предназначены для опытных разработчиков в области бизнес-аналитики, чем для пользователей Power BI и Excel, мы уверены, что пользу от их чтения получат все без исключения.

Структура книги

Эта книга построена так, что темы в ней располагаются по нарастающей – от простых к сложным. В каждой следующей главе предполагается, что вы полно-

стью усвоили материал предыдущей – мы старались практически не повторять то, о чем уже писали ранее. Именно поэтому мы настоятельно советуем читать книгу от начала до конца, не прыгая от главы к главе.

Будучи прочитанной, книга может превратиться в полезный справочник по DAX. Например, если вы захотите вспомнить, как работает функция *ALLSELECTED*, то можете открыть конкретный раздел и освежить память. Но обращаться к главам без их предварительного чтения мы не советуем – вы просто рискуете не до конца понять описываемую концепцию.

Представляем вам описание глав этой книги:

- глава 1 содержит краткое введение в DAX с несколькими разделами, предназначенными для тех, кто уже знаком с другими языками, такими как SQL, MDX или язык формул Excel. В этой главе мы не представляем какие-то новые концепции, а описываем базовые отличия между DAX и другими языками программирования, которые может знать читатель;
- в главе 2 мы познакомим вас с языком DAX. Мы пройдемся по основным терминам вроде вычисляемых столбцов и мер, а также расскажем о функциях для перехвата ошибок в выражениях. Кроме того, здесь будут перечислены все основные функции языка;
- глава 3 будет посвящена основным табличным функциям. Многие функции DAX работают с таблицами и возвращают таблицы в качестве результата. Здесь мы опишем работу большинства табличных функций, а в главах 12 и 13 расскажем о более сложных функциях для работы с таблицами;
- в главе 4 мы впервые затронем тему контекстов вычисления. Данная концепция является основополагающей в DAX, так что эта глава и следующая, возможно, являются наиболее значимыми в этой книге;
- в главе 5 мы ограничимся всего двумя функциями – *CALCULATE* и *CALCULATETABLE*. Это наиболее важные функции в DAX, и к их изучению можно приступать только после усвоения концепции контекстов вычисления;
- глава 6 будет посвящена переменным. Мы используем переменные в примерах на протяжении всей книги, но именно в этой главе познакомим вас с их синтаксисом и объясним назначение. Вы сможете возвращаться к этой части книги, когда будете встречаться с переменными в последующих главах;
- в главе 7 мы обсудим сладкую парочку из итерационных функций и функции CALCULATE, союз которых поистине был заключен на небесах. Использование итерационных функций совместно с техникой преобразования контекста позволит вам извлечь максимум пользы из языка DAX. В этой главе мы продемонстрируем несколько примеров, позволяющих реализовать весь потенциал данной связки;
- в главе 8 мы подробно остановимся на функциях логики операций со временем. Нарастающие итоги с начала года и месяца, показатели предыдущих лет, недельные интервалы и нестандартные календари – все это будет рассмотрено в этой части книги;
- глава 9 будет посвящена относительно новой особенности языка DAX – группам вычислений. Это очень мощный инструмент моделирования данных. В данной главе мы рассмотрим создание и использование групп

вычислений, познакомим вас с базовыми концепциями и представим несколько примеров;

- в главе 10 мы более подробно поговорим об особенностях использования контекста фильтра, привязке данных и других полезных средствах для расчета сложных формул;
- в главе 11 вы научитесь проводить вычисления над иерархиями и работать со структурами родитель/потомок в DAX;
- главы 12 и 13 посвящены продвинутым табличным функциям, полезным как при написании запросов, так и при проведении сложных вычислений;
- прочитав главу 14, вы продвинетесь на шаг вперед в понимании контекстов вычисления, а заодно узнаете об использовании сложных функций *ALLSELECTED* и *KEEPFILTERS* и концепции *расширенных таблиц* (expanded tables). Это глава для опытных пользователей, раскрывающая секреты сложных выражений DAX;
- глава 15 посвящена управлению связями в DAX. Благодаря этому языку в модели данных можно создавать связи всех возможных типов. Здесь мы также приведем описание всех типов связей, которые допустимо использовать в моделях;
- в главе 16 мы приведем несколько примеров сложных расчетов с использованием DAX. Это будет последняя глава, посвященная непосредственно языку, и в ней мы расскажем о разных решениях и новых идеях;
- в главе 17 мы приведем детальное описание *движка* (engine) VertiPaq, являющегося самым распространенным *движком хранилища данных* (storage engine) в моделях с использованием DAX. Понимание особенностей движка позволит вам извлекать максимум потенциала из языка;
- в главе 18 мы воспользуемся знаниями, полученными в предыдущей главе, чтобы продемонстрировать возможные способы оптимизации на уровне модели данных. Вы узнаете, как снизить количество уникальных значений в столбце, выбрать столбцы для импорта и повысить эффективность системы за счет выбора правильных типов связей и снижения количества используемой памяти в DAX;
- в главе 19 вы научитесь читать *планы выполнения запросов* (query plan) и замерять производительность выражений на DAX при помощи DAX Studio и SQL Server Profiler;
- в главе 20 мы покажем вам несколько техник по оптимизации модели с использованием знаний, полученных в предыдущих главах. Мы продемонстрируем разные выражения на DAX, проанализируем их производительность, а затем представим оптимизированные варианты формул.

Условные обозначения

В этой книге приняты следующие условные обозначения:

- **жирным** помечен текст, который вводите вы;
- *курсив* используется для обозначения новых терминов, а также названия мер, вычисляемых столбцов, таблиц и баз данных;

- первые буквы в названиях диалоговых окон, их элементов, а также команд – прописные. Например, в диалоговом окне **Save As...** (Сохранить как...);
- названия вкладок на ленте даются ПРОПИСНЫМИ БУКВАМИ;
- комбинации нажимаемых клавиш на клавиатуре обозначаются знаком плюс (+) между названиями клавиш. Например, **Ctrl**+**Alt**+**Delete** означает, что вы должны одновременно нажать клавиши **Ctrl**, **Alt** и **Delete**.

Сопутствующий контент

Для развития ваших навыков и подкрепления их практикой мы снабдили книгу сопутствующим контентом, который можно скачать по ссылке: MicrosoftPressStore.com/DefinitiveGuideDAX/downloads.

Представленный архив содержит:

- бэкап базы данных Contoso Retail DW в формате SQL Server, который вы можете использовать для самостоятельной проверки примеров. Это стандартная демонстрационная база от Microsoft, которую мы расширили путем добавления нескольких *представлений* (view) для облегчения создания на ее основе модели данных;
- файлы в формате Power BI Desktop для всех примеров из этой книги. Каждому рисунку соответствует отдельный файл. Модель данных при этом практически не меняется, но вы можете использовать эти файлы для самостоятельного выполнения всех шагов, описанных в книге.

ГЛАВА 1

Что такое DAX?

DAX, или *выражения анализа данных* (Data Analysis eXpressions), – это язык программирования в средах Microsoft Power BI, Microsoft Analysis Services и Microsoft Power Pivot для Excel. Он был создан в 2010 году – с первым выходом надстройки PowerPivot для Microsoft Excel 2010. Да, тогда название PowerPivot писалось слитно, а пробел появился лишь через три года. С тех пор язык DAX постоянно набирал популярность как в среде пользователей Excel, применяющих его для создания моделей данных в Power Pivot, так и в сообществе бизнес-аналитики, где этот язык используется для проектирования моделей в Power BI и Analysis Services. DAX присутствует во многих инструментах, которые объединяет один *табличный движок* (Tabular). Именно поэтому мы будем часто говорить просто о табличных моделях, подразумевая все инструменты сразу.

DAX – простой язык. При этом он существенно отличается от других языков программирования, так что на освоение его новых концепций у вас может уйти немало времени. По опыту преподавания DAX тысячам студентов мы можем заметить, что с основами языка проблем обычно не возникает – можно приступить к его использованию уже через несколько часов после начала обучения. Что касается продвинутых тем вроде контекста вычисления, итерационных функций и преобразования контекста, они могут вызвать серьезные затруднения. Но не сдавайтесь! Наберитесь терпения. Когда вы вникнете в эти концепции, вы поймете всю простоту языка DAX. К нему нужно просто привыкнуть.

В начале первой главы мы расскажем о том, что представляет из себя модель данных с таблицами и связями. Мы советуем прочитать эти страницы всем, независимо от опыта, чтобы понять, какую терминологию мы будем использовать на протяжении всей книги, описывая таблицы, модели и разные типы связей.

В следующих разделах мы дадим полезные советы читателям, имеющим определенные навыки работы с другими языками, такими как SQL, MDX и язык формул Microsoft Excel. Каждому из этих языков мы отведем отдельный раздел, чтобы читатели могли сравнить их с DAX. Если вам это поможет, попробуйте смотреть на DAX через призму этих языков. Прочитав заключительный раздел «DAX для пользователей Power BI», переходите к следующей главе, с которой, по сути, и начинается наше путешествие в мир DAX.

Введение в модель данных

Язык DAX предназначен для расчета бизнес-показателей посредством формул в модели данных. Некоторые читатели могут знать, что из себя представляет модель данных. Для остальных мы сделаем разъяснение.

Модель данных (data model) – это набор таблиц, объединенных связями.

Все мы знаем, что такое таблица. Это перечисление строк, содержащих информацию, при этом каждая строка поделена на столбцы. Столбец, в свою очередь, характеризуется определенным типом данных и содержит единый фрагмент информации. Обычно мы называем строку в таблице записью. Табличный способ хранения информации очень удобен в плане организации данных. По сути, таблица сама по себе является моделью данных, пусть и в своей простейшей форме. А значит, когда мы вводим на лист Excel текст и цифры, мы создаем модель данных.

Если модель состоит из нескольких таблиц, вполне вероятно, что вам захочется связать их. *Связь* (relationship) представляет собой объединение двух таблиц. Такие таблицы мы называем *связанными* (related). Графически связь двух таблиц обозначается линией между ними. На рис. 1.1 показан пример модели данных.

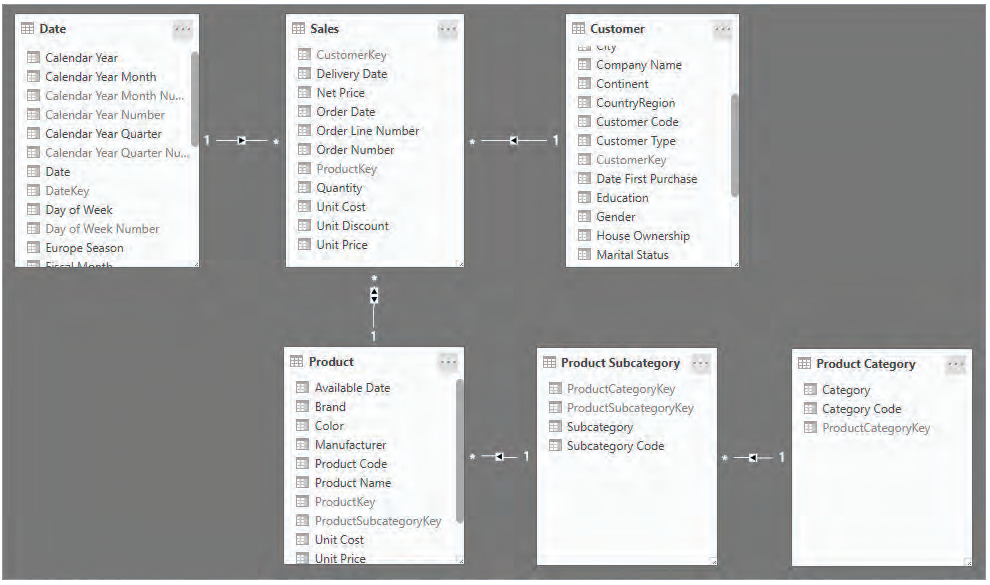


Рис. 1.1 Модель данных, состоящая из шести таблиц

Далее перечислим важные аспекты связей между таблицами:

- таблицы, объединенные связью, выполняют разные роли. Одна из них представляет сторону «один», а вторая – «многие», которые помечены на схеме данных символами «1» и «*» (звездочка) соответственно. Обратите внимание на связь между таблицами *Product* (Товары) и *Product Subcategory* (Подкатегории товаров) на рис. 1.1. Одной подкатегории может принадлежать несколько товаров, тогда как один товар может представлять только одну подкатегорию. Таким образом, таблица *Product Subcategory* являет собой сторону «один» в этой связи, а *Product* – сторону «многие»;
- существуют особые виды связей. Это связи «один к одному» (1:1) и *слабые связи* (weak relationships). В связи «один к одному» обе таблицы представ-

ляют собой сторону «один», тогда как в слабых связях они могут находиться на стороне «многие». Такие особые виды связей не слишком распространены, и мы подробно обсудим их в главе 15;

- столбцы, использующиеся для объединения таблиц и обычно имеющие одинаковые имена, называются *ключами* (keys) связи. При этом в ключевом столбце таблицы, представляющей сторону «один», должны находиться уникальные значения без пропусков. В то же время в таблице «многие» значения в ключевом столбце могут повторяться, и чаще всего это так и есть. Столбец, содержащий исключительно уникальные значения, называется *ключом* таблицы;
- связи могут образовывать цепочки. Каждый товар принадлежит какой-то подкатегории, которая, в свою очередь, представляет определенную категорию товаров. Следовательно, каждый товар можно отнести к конкретной категории. Но чтобы получить ее название, необходимо пройти к ней от товаров через цепочку из двух связей. В модели данных, представленной на рис. 1.1, присутствует цепочка связей, состоящая сразу из трех звеньев, – от таблицы *Sales* к *Product Category*;
- стрелкой посередине связи обозначается *направление перекрестной фильтрации* (cross filter direction). По рис. 1.1 видно, что связь между таблицами *Sales* и *Product* отмечена стрелками в обоих направлениях, тогда как остальные связи в модели – однонаправленные. Стрелкой обозначается направление распространения фильтра по этой связи. Поскольку выбор правильных направлений для фильтров является одним из важнейших навыков в работе с моделью данных, мы подробно обсудим эту тему в следующих главах. Обычно мы не советуем пользователям включать *двунаправленную фильтрацию* (bidirectional filtering) в связях, как сказано в главе 15. В этой модели такая связь присутствует исключительно в образовательных целях.

Введение в направление связи

Каждая связь может характеризоваться однонаправленной или двунаправленной перекрестной фильтрацией (кросс-фильтрацией). Фильтр всегда распространяется от стороны «один» к стороне «многие». Если же связь двунаправленная, то есть обозначена на схеме двумя разнонаправленными стрелками, фильтр по ней может распространяться и в обратном направлении.

Приведем пример, который поможет вам лучше разобраться в этом. Если построить отчет на основе модели данных, представленной на рис. 1.1, вынеся годы (*Calendar Year*) на строки, а количество проданных товаров (*Quantity*) и количество наименований товаров (*Count of Product Name*) – в область значений, мы увидим вывод, показанный на рис. 1.2.

Столбец *Calendar Year* принадлежит таблице дат (*Date*). А поскольку таблица *Date* представляет сторону «один» в связи с продажами (*Sales*), движок отфильтрует таблицу *Sales* по годам. Именно поэтому количество проданных товаров в отчете показано с разбивкой по годам.

С таблицей товаров (*Products*) дело обстоит несколько иначе. Фильтрация в этом случае работает корректно, поскольку связь, объединяющая таблицы

Sales и *Product*, является двунаправленной. Выводя в отчет количество наименований товаров, мы фактически получаем ежегодно продаваемый ассортимент посредством фильтра, распространенного от таблицы *Sales* к *Product*. Если бы связь между *Sales* и *Product* была однонаправленной, результат был бы иным, и мы расскажем об этом в следующих разделах.

Calendar Year	Quantity	Count of Product Name
CY 2007	44,310	1258
CY 2008	40,226	1478
CY 2009	55,644	1513
Total	**140,180**	**2517**

Рис. 1.2 Отчет демонстрирует эффект фильтрации по нескольким таблицам

Если модифицировать отчет, вынеся на строки цвет товаров (*Color*) и добавив в область значений количество дат (*Count of Date*), результат также поменяется. Вывод этого отчета можно видеть на рис. 1.3.

Color	Quantity	Count of Product Name	Count of Date
Azure	546	14	2556
Black	33,618	602	2556
Blue	8,859	200	2556
Brown	2,570	77	2556
Gold	1,393	50	2556
Green	3,020	74	2556
Grey	11,900	283	2556
Orange	2,203	55	2556
Pink	4,921	84	2556
Purple	102	6	2556
Red	8,079	99	2556
Silver	27,551	417	2556
Silver Grey	959	14	2556
Transparent	1,251	1	2556
White	30,543	505	2556
Yellow	2,665	36	2556
Total	**140,180**	**2517**	**2556**

Рис. 1.3 В отчете показано, что в отсутствие двунаправленной связи фильтрация таблиц не выполняется

Столбец *Color*, вынесенный на строки отчета, принадлежит таблице *Product*. А поскольку *Product* представляет сторону «один» в связи с таблицей *Sales*, значения в столбце *Quantity* посчитались корректно. Поле *Count of Product Name* правильно отфильтровалось, поскольку его источником является таблица *Product*, вынесенная на строки. Неожиданные значения мы видим в столбце

Count of Date. Здесь для всех строк указано одно и то же число, представляющее общее количество строк в таблице *Date*.

Фильтр, идущий от столбца *Color*, не распространяется на *Date*, поскольку связь между таблицами *Date* и *Sales* – однонаправленная. Таким образом, несмотря на то что фильтр в таблице *Sales* активен, он не может распространиться на таблицу *Date* по причине однонаправленности связи.

Если сделать связь между таблицами *Date* и *Sales* двунаправленной, результат будет иным, что видно по рис. 1.4.

Color	Quantity	Count of Product Name	Count of Date
Azure	546	14	41
Black	33,618	602	811
Blue	8,859	200	408
Brown	2,570	77	169
Gold	1,393	50	106
Green	3,020	74	188
Grey	11,900	283	499
Orange	2,203	55	142
Pink	4,921	84	226
Purple	102	6	11
Red	8,079	99	286
Silver	27,551	417	722
Silver Grey	959	14	63
Transparent	1,251	1	14
White	30,543	505	750
Yellow	2,665	36	110
Total	**140,180**	**2517**	**2556**

Рис. 1.4 Если активировать двунаправленную фильтрацию, таблица *Date* будет отфильтрована по столбцу *Color*

Теперь в столбце отображается количество дней, когда был продан как минимум один товар выбранного цвета. На первый взгляд кажется, что стоит все связи в модели сделать двунаправленными, чтобы позволить фильтрам распространять свое действие во все стороны и доставать правильные данные. Как вы узнаете из этой книги, такой подход почти никогда не будет оправдан. Вы должны выбирать направление фильтрации для связей в зависимости от модели, с которой работаете. Если вы последуете нашим советам, то откажетесь от применения двунаправленной фильтрации там, где это возможно.

DAX для пользователей Excel

Велика вероятность, что вы знакомы с языком формул Excel, который немного напоминает DAX. В конце концов, корни DAX лежат в Power Pivot для Excel,

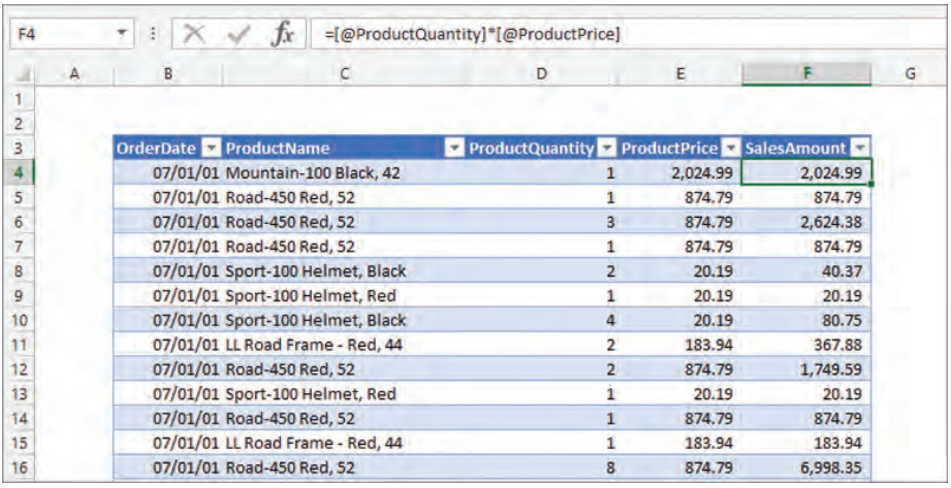
и разработчики сделали все, чтобы эти языки были похожими. Эти сходства облегчат вам переход на DAX. Но не стоит забывать и о различиях в этих языках.

Ячейки против таблиц

В Excel все вычисления производятся над ячейками, которые обладают координатами. Так что мы можем написать формулу вроде этой:

```
= (A1 * 1.25) - B2
```

В DAX концепция ячеек с координатами просто отсутствует. Этот язык работает с таблицами и столбцами, а не с отдельными ячейками. Как следствие выражения DAX обращаются именно к таблицам и столбцам, что сказывается на синтаксисе языка. Однако концепция таблиц и столбцов не нова для Excel. Если выделить диапазон и воспользоваться пунктом **Format as Table** (Форматировать как таблицу), можно писать формулы в Excel, обращающиеся непосредственно к таблицам и столбцам. На рис. 1.5 в столбце *SalesAmount* вычисляется выражение, ссылающееся на столбцы в той же таблице, а не на ячейки в рабочей книге.



OrderDate	ProductName	ProductQuantity	ProductPrice	SalesAmount
07/01/01	Mountain-100 Black, 42	1	2,024.99	2,024.99
07/01/01	Road-450 Red, 52	1	874.79	874.79
07/01/01	Road-450 Red, 52	3	874.79	2,624.38
07/01/01	Road-450 Red, 52	1	874.79	874.79
07/01/01	Sport-100 Helmet, Black	2	20.19	40.37
07/01/01	Sport-100 Helmet, Red	1	20.19	20.19
07/01/01	Sport-100 Helmet, Black	4	20.19	80.75
07/01/01	LL Road Frame - Red, 44	2	183.94	367.88
07/01/01	Road-450 Red, 52	2	874.79	1,749.59
07/01/01	Sport-100 Helmet, Red	1	20.19	20.19
07/01/01	Road-450 Red, 52	1	874.79	874.79
07/01/01	LL Road Frame - Red, 44	1	183.94	183.94
07/01/01	Road-450 Red, 52	8	874.79	6,998.35

Рис. 1.5 В формулах Excel можно ссылаться на столбцы таблицы

В Excel можно обращаться к столбцам, используя следующий формат: *[@ColumnName]*. Здесь ColumnName – название столбца, а символ @ говорит о том, что необходимо взять значение из текущей строки. Синтаксис получился не самым интуитивно понятным, но мы обычно и не пишем такие выражения вручную. Они появляются автоматически при нажатии на ячейку: Excel сам заботится о вставке нужного кода.

Таким образом, в Excel есть два разных вида вычислений. Можно использовать стандартное обращение к ячейкам – в этом случае формула для ячейки F4 будет выглядеть так: E4*D4. Или же применять ссылки на столбцы внутри таблицы. Это позволит использовать одинаковые выражения во всех ячейках

столбца, а Excel в своих расчетах будет брать значение из конкретной строки.

В отличие от Excel, DAX работает исключительно с таблицами. Все формулы должны ссылаться на столбцы внутри таблиц. Например, в DAX предыдущая формула будет выглядеть так:

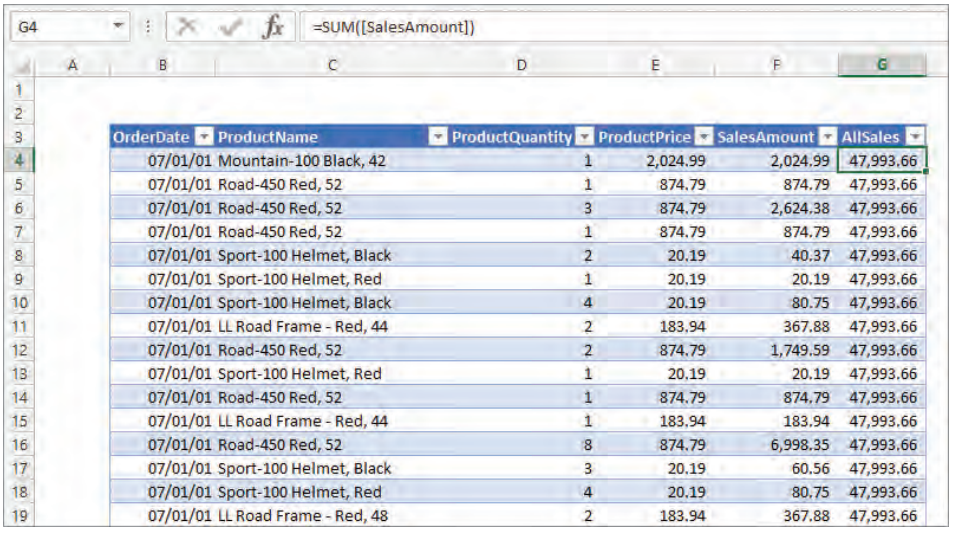
```
Sales[SalesAmount] = Sales[ProductPrice] * Sales[ProductQuantity]
```

Как видите, каждое название столбца предваряется наименованием соответствующей таблицы. В Excel мы не указываем названия таблиц, поскольку там формулы работают внутри одной таблицы. DAX же работает в модели данных, состоящей из нескольких таблиц. Как следствие мы просто обязаны конкретно указывать таблицы, ведь в разных таблицах могут находиться столбцы с одинаковыми названиями.

Многие функции DAX работают подобно аналогичным функциям в Excel. К примеру, функция *IF* в обоих языках применяется одинаково:

```
Excel ЕСЛИ ( [@SalesAmount] > 10; 1; 0)
DAX IF ( Sales[SalesAmount] > 10; 1; 0)
```

Единственным существенным отличием между Excel и DAX является способ обращения к целому столбцу. В Excel, как мы уже говорили, символ @ в выражении *[@ProductQuantity]* означает, что необходимо взять значение из текущей строки. В DAX нет необходимости указывать этот факт явно, поскольку такое поведение является для языка обычным. В Excel мы можем обратиться ко всем строкам в столбце, убрав из формулы символ @. Это можно видеть на рис. 1.6.



	A	B	C	D	E	F	G
1							
2							
3		OrderDate	ProductName	ProductQuantity	ProductPrice	SalesAmount	AllSales
4		07/01/01	Mountain-100 Black, 42	1	2,024.99	2,024.99	47,993.66
5		07/01/01	Road-450 Red, 52	1	874.79	874.79	47,993.66
6		07/01/01	Road-450 Red, 52	3	874.79	2,624.38	47,993.66
7		07/01/01	Road-450 Red, 52	1	874.79	874.79	47,993.66
8		07/01/01	Sport-100 Helmet, Black	2	20.19	40.37	47,993.66
9		07/01/01	Sport-100 Helmet, Red	1	20.19	20.19	47,993.66
10		07/01/01	Sport-100 Helmet, Black	4	20.19	80.75	47,993.66
11		07/01/01	LL Road Frame - Red, 44	2	183.94	367.88	47,993.66
12		07/01/01	Road-450 Red, 52	2	874.79	1,749.59	47,993.66
13		07/01/01	Sport-100 Helmet, Red	1	20.19	20.19	47,993.66
14		07/01/01	Road-450 Red, 52	1	874.79	874.79	47,993.66
15		07/01/01	LL Road Frame - Red, 44	1	183.94	183.94	47,993.66
16		07/01/01	Road-450 Red, 52	8	874.79	6,998.35	47,993.66
17		07/01/01	Sport-100 Helmet, Black	3	20.19	60.56	47,993.66
18		07/01/01	Sport-100 Helmet, Red	4	20.19	80.75	47,993.66
19		07/01/01	LL Road Frame - Red, 48	2	183.94	367.88	47,993.66

Рис. 1.6 В Excel можно сослаться на весь столбец, опустив символ @ в формуле

Значение столбца *AllSales* одинаковое для всех строк и равно общему итогу по столбцу *SalesAmount*. Иными словами, в Excel существует четкое синтаксическое разграничение между обращением к ячейке в конкретной строке и к столбцу в целом.

DAX ведет себя иначе. В этом языке для вычисления столбца *AllSales* из рис. 1.6 можно было бы использовать следующую формулу:

```
AllSales := SUM ( Sales[SalesAmount] )
```

И здесь нет никаких отличий между извлечением значения из текущей строки или из всего столбца. DAX понимает, что мы хотим просуммировать все значения из столбца, поскольку его название передается в качестве аргумента в агрегирующую функцию (здесь это функция *SUM*). Таким образом, если Excel требует явного указания, какие данные извлекать из столбца, DAX решает эту неоднозначность автоматически. Такая разница в подходах к вычислениям может приводить в замешательство – по крайней мере, поначалу.

Excel и DAX: два функциональных языка

В чем язык формул Excel и DAX похожи, так это в том, что оба они являются функциональными языками программирования. Функциональные языки состоят из выражений, в основе которых лежат вызовы функций. В Excel и DAX не реализованы концепции операторов, циклов и переходов, характерные для большинства языков программирования. В DAX буквально все является выражениями. Это бывает непросто понять тем, кто приходит из других языков программирования, а для пользователей Excel, наоборот, должно быть привычно.

Итерационные функции в DAX

Концепция, которая может оказаться для вас в новинку, – это итерационные функции, или просто *итераторы* (iterators). В Excel все расчеты выполняются последовательно, по одному за раз. В предыдущем примере вы видели, что для того, чтобы рассчитать итог по продажам, мы создали столбец, в котором цена умножалась на количество. На втором шаге мы подсчитывали сумму по этой колонке. Получившийся результат впоследствии можно использовать в качестве знаменателя при подсчете, например, доли продаж по каждому товару.

В DAX все это можно сделать за один шаг с помощью итерационных функций. Итератор делает ровно то, что и должен, исходя из названия, – проходит по таблице и производит вычисления в каждой строке, одновременно агрегируя запрошенное значение.

Таким образом, вычисления из предыдущего примера можно произвести при помощи одной итерационной функции *SUMX*:

```
AllSales :=
SUMX (
    Sales;
    Sales[ProductQuantity] * Sales[ProductPrice]
)
```

Такой подход имеет как достоинства, так и недостатки. К достоинствам можно отнести то, что мы можем производить множество вычислений за один шаг, не беспокоясь о создании вспомогательных столбцов, функциональность которых ограничивается лишь промежуточными формулами. Недостатком же

является то, что программирование на DAX менее визуально по сравнению с формулами Excel. Мы ведь даже не видим столбца с результатом умножения цены на количество – он существует только во время вычисления.

Как вы узнаете позже, у вас есть возможность создания вычисляемых столбцов для хранения подобных промежуточных вычислений. Но делать это не рекомендуется, поскольку тогда будут задействованы дополнительные ресурсы памяти и может пострадать производительность, если вы не используете режим DirectQuery совместно с агрегациями, о чем мы поговорим в главе 18.

DAX требует изучения теории

Будем откровенны: DAX является не единственным языком программирования, для использования которого вам понадобится обширная теоретическая база. Разница лишь в подходе. Признайтесь, вы ведь частенько ищете в интернете сложные формулы и шаблоны, которые помогут вам в решении вашего собственного сценария. И шансы на то, что вы найдете подходящую формулу для Excel, достаточно высоки – вам останется лишь адаптировать ее под свои нужды.

Но в DAX дела обстоят иначе. Вам придется досконально изучить этот язык и понять, как работает контекст вычисления, чтобы написать работающий код. Без должной теоретической базы вам может показаться, что DAX производит свои вычисления каким-то магическим образом или что он выдает цифры, не имеющие с реальностью ничего общего. Проблема не в DAX, а в том, что вы не понимаете всех тонкостей его работы.

К счастью, теоретическая база языка DAX ограничивается всего несколькими концепциями, которые мы опишем в главе 4. Приготовьтесь много учиться. После освоения этой главы DAX перестанет быть для вас тайной, а мастерство его использования будет зависеть исключительно от приобретенного опыта. Помните: знание – всего лишь полдела. И не пытайтесь двигаться дальше, пока досконально не освоите концепцию контекстов вычисления.

DAX для разработчиков SQL

Если вы знакомы с языком SQL, значит, у вас уже есть опыт работы со множеством таблиц и связей. В этом плане вы почувствуете себя в DAX как дома. По сути, вычисления здесь базируются на выполнении запросов к нескольким таблицам, объединенным связями, и агрегировании значений.

Работа со связями

Первые отличия между SQL и DAX заметны в области организации связей в модели данных. В SQL можно настроить внешние ключи в таблицах для определения связей, но движок никогда не будет использовать эти связи без явного указания. Например, если у нас есть таблицы *Customers* и *Sales*, и столбец *CustomerKey* является первичным ключом в *Customers* и внешним – в *Sales*, можно написать следующий запрос:

```
SELECT
    Customers.CustomerName,
    SUM ( Sales.SalesAmount ) AS SumOfSales
FROM
    Sales
    INNER JOIN Customers
     ON Sales.CustomerKey = Customers.CustomerKey
GROUP BY
    Customers.CustomerName
```

Хотя мы определили в модели внешние ключи для осуществления связей, нам все равно необходимо всякий раз явно указывать в запросе условия для выполнения соединений. Это приводит к увеличению объема запросов, зато можно каждый раз использовать разные условия для связей, что дает максимум свободы в извлечении данных.

В DAX связи являются составной частью модели данных, и все они – *LEFT OUTER JOIN*. А раз так, вам нет необходимости каждый раз указывать их в запросе, DAX автоматически будет использовать связи при задействовании объединенных таблиц. Так что на DAX можно переписать предыдущий запрос SQL следующим образом:

```
EVALUATE
SUMMARIZECOLUMNS (
    Customers[CustomerName];
    "SumOfSales", SUM ( Sales[SalesAmount] )
)
```

Поскольку движок знает о созданной связи между таблицами *Sales* и *Customers*, объединение таблиц в запросе происходит автоматически. После этого функции *SUMMARIZECOLUMNS* останется выполнить группировку по столбцу *Customers[CustomerName]*, причем для этого нет определенного ключевого слова: функция *SUMMARIZECOLUMNS* автоматически группирует данные по выбранным столбцам.

DAX как функциональный язык

SQL – декларативный язык. Вы определяете набор данных, который желаете извлечь, посредством оператора *SELECT*, при этом не беспокоясь о том, как именно движок это сделает.

DAX, напротив, является функциональным языком. В нем каждое выражение является вызовом функции. При этом параметры функции, в свою очередь, также могут быть вызовами функций. Анализ всех этих параметров приводит к созданию сложного плана выполнения запроса, который и вычисляется движком DAX с целью получить результат.

Например, если нам понадобится получить информацию о покупателях, живущих в Европе, мы можем написать следующий запрос на SQL:

```
SELECT
    Customers.CustomerName,
    SUM ( Sales.SalesAmount ) AS SumOfSales
```

```
FROM
    Sales
    INNER JOIN Customers
     ON Sales.CustomerKey = Customers.CustomerKey
WHERE
    Customers.Continent = 'Europe'
GROUP BY
    Customers.CustomerName
```

В языке DAX мы не объявляем условие в операторе WHERE. Вместо этого мы используем специальную функцию *FILTER* для осуществления фильтрации, как показано ниже:

```
EVALUATE
SUMMARIZECOLUMNS (
    Customers[CustomerName];
    FILTER (
        Customers;
        Customers[Continent] = "Europe"
    );
    "SumOfSales"; SUM ( Sales[SalesAmount] )
)
```

Вы видите, как работает функция *FILTER*: она возвращает только покупателей, проживающих в Европе, как мы и хотели. Порядок, в котором мы встраиваем функции в код, и виды функций, которые используем, очень важны как с точки зрения получения результата, так и в плане производительности запросов. В языке SQL это тоже важно, хотя там мы больше надеемся на *оптимизатор запросов* (query optimizer) при построении наилучшего плана выполнения. В DAX оптимизатор также занят своими прямыми обязанностями, но на вас как на разработчике лежит большая ответственность за написание быстро работающего кода.

DAX как язык программирования и язык запросов

В SQL существует четкое разделение между языком запросов и языком программирования, то есть набором инструкций, используемых для создания *хранимых процедур* (stored procedures), *представлений* (views) и других объектов в базе данных. В каждом диалекте SQL присутствуют свои операторы, призванные обогатить язык. Но в DAX не делается четких разграничений между языком запросов и языком программирования. Множество функций работают с таблицами и возвращают таблицы в качестве результата. Функция *FILTER* из предыдущего кода – лишь один из примеров.

В этом отношении DAX, пожалуй, проще SQL. Изучая его как язык программирования – а им он изначально и является, – вы узнаете все необходимое для использования его и в качестве языка запросов.

Подзапросы и условия в DAX и SQL

Одной из мощнейших особенностей языка запросов SQL является возможность использования подзапросов. В DAX применяется похожая концепция, но с учетом функциональной направленности языка.

Например, чтобы извлечь информацию о покупателях, сделавших покупки на сумму более $100, можно написать следующий запрос SQL:

```
SELECT
    CustomerName,
    SumOfSales
FROM (
    SELECT
     Customers.CustomerName,
     SUM ( Sales.SalesAmount ) AS SumOfSales
    FROM
     Sales
     INNER JOIN Customers
      ON Sales.CustomerKey = Customers.CustomerKey
    GROUP BY
     Customers.CustomerName
    ) AS SubQuery
WHERE
    SubQuery.SumOfSales > 100
```

В DAX можно добиться похожего эффекта с использованием вложенных функций, как показано ниже:

```
EVALUATE
FILTER (
    SUMMARIZECOLUMNS (
        Customers[CustomerName];
        "SumOfSales", SUM ( Sales[SalesAmount] )
    );
    [SumOfSales] > 100
)
```

В этом коде результаты подзапроса, извлекающего *CustomerName* и *SumOfSales*, прогоняются через функцию *FILTER*, которая оставляет в результирующем наборе только строки со значениями *SumOfSales*, превышающими 100. В данный момент вы можете не понимать, что делает этот код. Но, постепенно постигая все премудрости DAX, вы обнаружите, что в этом языке использовать подзапросы намного легче, чем в SQL, и код получается более естественным по причине функциональной природы DAX.

DAX для разработчиков MDX

Многие специалисты в области бизнес-аналитики переключаются на DAX как на новый язык табличного движка Tabular. В прошлом они использовали язык *MDX* для построения и обращения к *многомерным моделям данных* (Multidimensional models) Analysis Services. Если вы из их числа, приготовьтесь изучать абсолютно новый язык, поскольку у DAX и MDX не так много общего. Более того, некоторые концепции в DAX будут сильно напоминать вам MDX, но смысл их будет совершенно иным.

По опыту можем сказать, что путь от MDX к DAX наиболее тернист. Чтобы изучить DAX, вам придется забыть все, что вы знаете о MDX. Выкиньте из головы *многомерные пространства* (multidimensional spaces) и приготовьтесь к приобретению новых знаний с нуля.

Многомерность против табличности

MDX работает в многомерном пространстве, определенном моделью данных. Его форма зависит от *измерений* (dimensions) и *иерархий* (hierarchies), присутствующих в модели, и в свою очередь определяет систему координат многомерного пространства. Пересечения наборов элементов в разных измерениях определяют точки в многомерном пространстве. Может понадобиться немало времени, чтобы понять, что элемент *[All]* любой иерархии атрибута – это не более чем точка в многомерном пространстве.

В DAX все намного проще. Тут нет измерений, элементов и точек в многомерном пространстве. Да и самого многомерного пространства тоже нет. Есть иерархии, которые мы можем определять в модели данных, но они существенно отличаются от иерархий в MDX. Пространство DAX построено на таблицах, столбцах и связях. Таблицы в модели Tabular не являются ни *группами мер* (measure group), ни измерениями. Это просто таблицы, для проведения вычислений в которых вы можете сканировать их, фильтровать и суммировать значения. Все базируется на двух основных концепциях: таблицах и связях.

Скоро вы узнаете, что с точки зрения моделирования данных табличный движок предоставляет меньше возможностей по сравнению с многомерным. Но в данном случае это не означает, что в вашем распоряжении будет меньший аналитический потенциал, поскольку вы всегда можете использовать DAX в качестве языка программирования, чтобы обогатить модель данных. Истинный потенциал движка Tabular заключается в потрясающей скорости DAX. Обычно разработчики стараются не злоупотреблять языком MDX без необходимости, поскольку оптимизировать такие запросы бывает непросто. DAX, напротив, славится своим впечатляющим быстродействием. Так что большинство сложных вычислений вы будете производить не в модели данных, а в формулах DAX.

DAX как язык программирования и язык запросов

И DAX, и MDX являются одновременно и языками программирования, и языками запросов. В MDX это разделение обусловлено наличием скриптов, в которых помимо базового языка MDX можно использовать специальные операторы вроде *SCOPE*, применимые исключительно в скриптах. В запросах на извлечение данных в MDX вы пользуетесь оператором *SELECT*. В DAX все несколько иначе. Вы можете использовать его как язык программирования для определения вычисляемых столбцов, вычисляемых таблиц и мер. И если концепция вычисляемых столбцов и таблиц является новинкой в DAX, то меры очень напоминают вычисляемые элементы в MDX. Можно также использовать DAX в качестве языка запросов – например, для извлечения информации из модели Tabular при помощи *Службы отчетов* (Reporting Services). При этом в функциях DAX нет четкого разграничения в плане использования – все они

могут быть применены как в запросах, так и при вычислении выражений. Более того, в модели Tabular можно также использовать запросы, написанные на языке MDX. Таким образом, хотя MDX и может использоваться с табличной моделью данных в качестве языка запросов, когда речь идет о программировании в среде Tabular, единственным вариантом является DAX.

Иерархии

Производя большинство вычислений с помощью языка MDX, вы полагаетесь на иерархии. Если вам необходимо получить сумму продаж по предыдущему году, вам придется извлечь *PrevMember* из *CurrentMember* иерархии *Year* и использовать это выражение для переопределения фильтра в MDX. Например, вы можете написать такую формулу для осуществления расчетов по предыдущему году на MDX:

```
CREATE MEMBER CURRENTCUBE.[Measures].[SamePeriodPreviousYearSales] AS
(
    [Measures].[Sales Amount],
    ParallelPeriod (
        [Date].[Calendar].[Calendar Year],
        1,
        [Date].[Calendar].CurrentMember
    )
);
```

В мере используется функция *ParallelPeriod*, возвращающая соседний элемент относительно *CurrentMember* на иерархии *Calendar*. Таким образом, это вычисление базируется на иерархиях, определенных в модели. В DAX мы бы для этого использовали контекст фильтра и стандартные функции для работы с датой и временем, как показано ниже:

```
SamePeriodPreviousYearSales :=
CALCULATE (
    SUM ( Sales[Sales Amount] );
    SAMEPERIODLASTYEAR ( 'Date'[Date] )
)
```

Можно произвести это вычисление разными способами, в том числе при помощи функции *FILTER*, но идея остается прежней: вместо использования иерархий мы применяем фильтрацию таблиц. Это очень существенное различие, и вам, вероятно, будет не хватать иерархий в DAX, пока не привыкнете к новой для себя концепции.

Еще одним весомым отличием между этими языками является то, что в MDX вы ссылаетесь на *[Measures].[Sales Amount]*, тогда как функция агрегации, которая вам нужна, уже определена в модели. В DAX предопределенные агрегации не используются. Фактически, как вы заметили, вычисляемое выражение в приведенном выше примере следующее: `SUM(Sales[Sales Amount])`. Никаких предопределенных агрегаций в модели нет. Мы определяем их тогда, когда нам нужно. Всегда можно создать меру, вычисляющую сумму продаж, но эта тема выходит за рамки этого раздела и будет описана позже в данной книге.

Более существенным отличием DAX от MDX является то, что в MDX очень активно используется инструкция *SCOPE* для реализации бизнес-логики (опять же с использованием иерархий), тогда как в DAX применяется совсем другой подход. Вообще, работы с иерархиями не хватает этому языку.

Например, если нам нужно очистить меру на уровне *Year*, в MDX мы могли бы написать следующее выражение:

```
SCOPE ( [Measures].[SamePeriodPreviousYearSales], [Date].[Month].[All] )
    THIS = NULL;
END SCOPE;
```

В DAX нет функций, похожих на SCOPE, и для получения аналогичного результата придется выполнить проверку контекста фильтра, как показано ниже:

```
SamePeriodPreviousYearSales :=
IF (
    ISINSCOPE ( 'Date'[Month] );
    CALCULATE (
        SUM ( Sales[Sales Amount] );
        SAMEPERIODLASTYEAR ( 'Date'[Date] )
    );
    BLANK ()
)
```

Из кода функции понятно, что она возвратит результат, только если пользователь находится в календарной иерархии на уровне месяца или ниже. В противном случае функция вернет пустое значение (*BLANK*). Позже вы узнаете, как работает эта функция. Стоит отметить, что это выражение более уязвимо к ошибкам, чем код на MDX. Да, честно говоря, языку DAX очень не хватает функций для работы с иерархиями.

Вычисления на конечном уровне

Используя язык MDX, вы, возможно, привыкли избегать проведения расчетов на *конечном уровне* (leaf-level) элементов. Это настолько медленная операция, что всегда будет предпочтительнее предварительно рассчитывать значения и использовать агрегацию для возврата результата. В DAX вычисления на конечном уровне работают невероятно быстро, а предварительные агрегации служат другим целям и используются только в работе с большими наборами данных. Вам придется несколько изменить подход к проектированию моделей данных. В большинстве случаев модели, идеально подходящие для многомерной среды SQL Server Analysis Services, будут не лучшим образом показывать себя в движке Tabular, и наоборот.

DAX для пользователей Power BI

Если вы пропустили предыдущие разделы и сразу оказались здесь, что ж, приветствуем! Язык DAX является родным для Power BI. И если у вас нет опыта работы с Excel, SQL или MDX, Power BI станет для вас первой средой, в которой вы

сможете изучать DAX. В отсутствие навыков построения моделей данных при помощи других инструментов вам будет приятно узнать, что Power BI является мощнейшим средством анализа и моделирования, а DAX во всем ему помогает.

Возможно, вы не так давно начали работать с Power BI, а сейчас хотите сделать очередной качественный шаг вперед. Если это так, приготовьтесь к увлекательному путешествию в мир DAX.

Вот вам наш совет: не ожидайте, что уже через пару дней вы сможете писать сложный код на DAX. Этот язык потребует от вас полной концентрации и внимания, а на его освоение может уйти немало времени, включая практическую работу. По опыту можем сказать, что после проведения первых простых вычислений на DAX вы будете просто восхищены. Но восхищение пропадет, когда вы дойдете до изучения контекстов вычислений и функции *CALCULATE* – наиболее сложных составляющих языка. В этот момент вам все покажется очень сложным. Но не отчаивайтесь! Большинство разработчиков DAX проходили через это. На этой стадии вы уже так много всего изучите, что бросать все будет просто жалко. Читайте и практикуйтесь снова и снова, и вы увидите, что озарение придет раньше, чем вы ожидаете. И тогда вы очень быстро завершите чтение этой книги – уже в статусе гуру по DAX.

Контексты вычислений – это сердце языка DAX. Освоение их может занять много времени. Мы не знаем никого, кому удалось бы узнать все о DAX за пару дней. Но, как и с любым сложным делом, со временем вы научитесь получать наслаждение от мелочей. А когда решите, что знаете уже все, перечитайте книгу заново. Уверяем, вы найдете для себя массу полезных нюансов, которые при первом прочтении казались не такими важными, но с приобретением опыта смогут заиграть новыми красками.

Насладитесь остатком этой книги!