O QUE É DAX

DAX, que significa Data Analysis eXpressions, é a linguagem de programação do Microsoft Power BI, Microsoft Analysis Services e Microsoft Power Pivot para Excel. Foi criada em 2010, com o primeiro lançamento do PowerPivot para o Microsoft Excel 2010. Em 2010, PowerPivot era escrito sem espaço. O espaço foi introduzido no nome Power Pivot em 2013. Desde então, DAX ganhou popularidade, tanto na comunidade do Excel, que utiliza DAX para criar modelos de dados Power Pivot no Excel, quanto na comunidade de Business Intelligence (BI), que utiliza DAX para construir modelos com o Power BI e o Analysis Services. DAX está presente em muitas ferramentas diferentes, todas compartilhando o mesmo motor interno chamado Tabular. Por essa razão, muitas vezes nos referimos a modelos Tabulares, incluindo todas essas ferramentas diferentes em uma única palavra.

DAX é uma linguagem simples. Dito isso, DAX é diferente da maioria das linguagens de programação, então familiarizar-se com alguns de seus novos conceitos pode levar algum tempo. Em nossa experiência, tendo ensinado DAX para milhares de pessoas, aprender o básico do DAX é direto: você poderá começar a usá-lo em questão de horas. Quando se trata de entender conceitos avançados como contextos de avaliação, iterações e transições de contexto, tudo provavelmente parecerá complexo. Não desista! Seja paciente. Quando seu cérebro começar a assimilar esses conceitos, você descobrirá que DAX é, de fato, uma linguagem fácil. Só precisa de um pouco de prática.

Este primeiro capítulo começa com um resumo do que é um modelo de dados em termos de tabelas e relacionamentos. Recomendamos que leitores de todos os níveis de experiência leiam esta seção para se familiarizar com os termos usados ao longo do livro ao se referir a tabelas, modelos e diferentes tipos de relacionamentos.

Nas seções seguintes, oferecemos conselhos para leitores que têm experiência com linguagens de programação como o Microsoft Excel, SQL e MDX. Cada seção está focada em uma linguagem específica, para leitores interessados em comparar brevemente DAX com ela. Concentre-se nas linguagens que você conhece se uma comparação for útil para você; em seguida, leia a última seção, "DAX para usuários do Power BI", e passe para o próximo capítulo, onde nossa jornada na linguagem DAX realmente começa

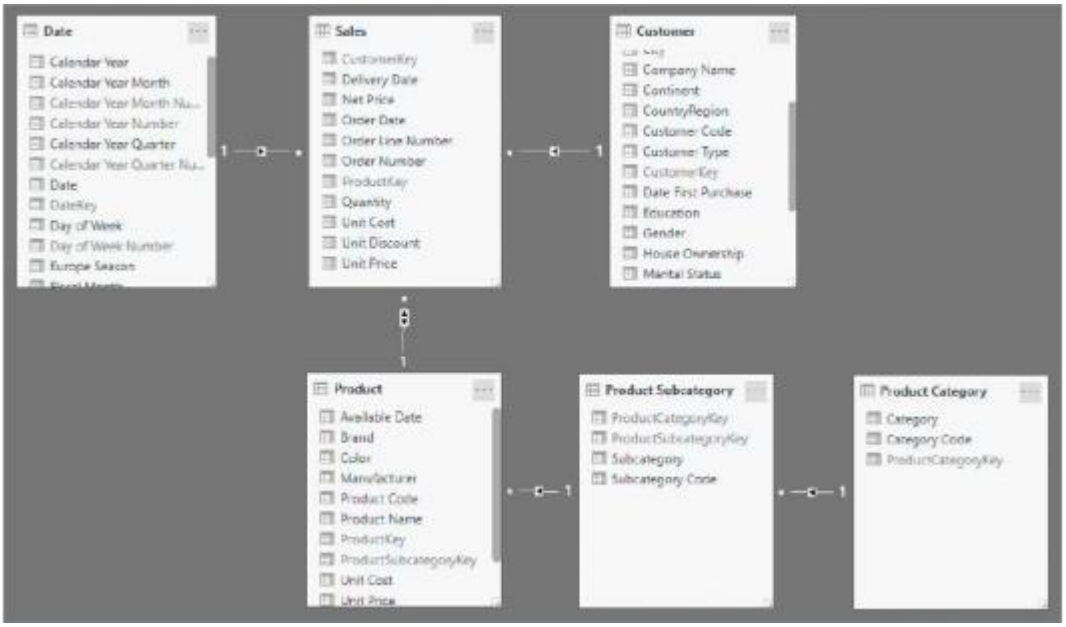
Compreendendo o modelo de dados

DAX é especificamente projetado para calcular fórmulas de negócios sobre um modelo de dados. Os leitores podem já saber o que é um modelo de dados. Se não, começamos com uma descrição de modelos de dados e relacionamentos para criar uma base sobre a qual construir seu conhecimento em DAX.

Um modelo de dados é um conjunto de tabelas, conectadas por relacionamentos.

Todos nós sabemos o que é uma tabela: um conjunto de linhas contendo dados, com cada linha dividida em colunas. Cada coluna tem um tipo de dados e contém uma única informação. Normalmente nos referimos a uma linha em uma tabela como um registro. Tabelas são uma forma conveniente de organizar dados. Uma tabela é um modelo de dados em si, embora em sua forma mais simples. Portanto, quando escrevemos nomes e números em uma planilha do Excel, estamos criando um modelo de dados.

Se um modelo de dados contém muitas tabelas, é provável que elas estejam vinculadas por relacionamentos. Um relacionamento é uma ligação entre duas tabelas. Quando duas tabelas estão conectadas por um relacionamento, dizemos que elas estão relacionadas. Graficamente, um relacionamento é representado por uma linha conectando as duas tabelas. A Figura 1-1 mostra um exemplo de um modelo de dados



Aqui estão alguns aspectos importantes sobre relacionamentos:

■ Duas tabelas em um relacionamento não têm o mesmo papel. Elas são chamadas de lado um e lado muitos do relacionamento, representadas respectivamente com um 1 e com um *. Na Figura 1-1, concentre-se no relacionamento entre Produto e Subcategoria de Produto. Uma única subcategoria contém muitos produtos, enquanto um único produto tem apenas uma subcategoria. Portanto, Subcategoria de Produto é o lado um do relacionamento, tendo uma subcategoria, enquanto Produto é o lado muitos, tendo muitos produtos.

■ Relacionamentos especiais incluem 1:1 e relacionamentos fracos. Em relacionamentos 1:1, ambas as tabelas são o lado um, enquanto em relacionamentos fracos, ambas as tabelas podem ser o lado muitos. Esses tipos especiais de relacionamentos são incomuns; discutimos detalhadamente no Capítulo 15, "Relacionamentos Avançados".

As colunas usadas para criar o relacionamento, que geralmente têm o mesmo nome em ambas as tabelas, são chamadas de chaves do relacionamento. No lado um do relacionamento, a coluna precisa ter um valor único para cada linha e não pode conter espaços em branco. No lado muitos, o mesmo valor pode se repetir em muitas linhas diferentes, e frequentemente o faz. Quando uma coluna tem um valor único para cada linha, é chamada de chave para a tabela.

■ Relacionamentos podem formar uma cadeia. Cada produto tem uma subcategoria e cada subcategoria tem uma categoria. Portanto, cada produto tem uma categoria. Para recuperar a categoria de um produto, é necessário percorrer uma cadeia de dois relacionamentos. A Figura 1-1 inclui um exemplo de uma cadeia formada por três relacionamentos, começando com Vendas e continuando com Categoria de Produto.

■ Em cada relacionamento, uma ou duas pequenas setas podem determinar a direção do filtro cruzado. A Figura 1-1 mostra duas setas no relacionamento entre Vendas e Produto, enquanto todos os outros relacionamentos têm uma única seta. A seta indica a direção do filtro cruzado automático do relacionamento (filtro cruzado). Porque determinar a direção correta dos filtros é uma das habilidades mais importantes a serem aprendidas, discutimos esse tópico com mais detalhes nos capítulos posteriores. Geralmente desencorajamos o uso de filtros bidirecionais, conforme descrito no Capítulo 15. Eles estão presentes neste modelo apenas para fins educacionais.

Entendendo a direção de um relacionamento

Cada relacionamento pode ter um filtro cruzado unidirecional ou bidirecional. A filtragem sempre ocorre do lado um do relacionamento para o lado muitos. Se o filtro cruzado for bidirecional - ou seja, se tiver duas setas - a filtragem também ocorre do lado muitos para o lado um. Um exemplo pode ajudar a entender esse comportamento. Se um

relatório for baseado no modelo de dados mostrado na Figura 1-1, com os anos nas linhas e Quantidade e Contagem de Nome do Produto na área de valores, ele produzirá o resultado mostrado na Figura 1-2.

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**

FIGURE 1-2 This report shows the effect of filtering across multiple tables.

Calendar Year é uma coluna que pertence à tabela Data. Porque Data está no lado um do relacionamento com Vendas, o mecanismo filtra Vendas com base no ano. É por isso que a quantidade mostrada é filtrada pelo ano.

Com Produtos, o cenário é um pouco diferente. A filtragem ocorre porque o relacionamento entre as tabelas Vendas e Produto é bidirecional. Quando colocamos a contagem de nomes de produtos no relatório, obtemos o número de produtos vendidos em cada ano porque o filtro no ano se propaga para Produto através de Vendas. Se o relacionamento entre Vendas e Produto fosse unidirecional, o resultado seria diferente, como explicamos nas seções seguintes.

Se modificarmos o relatório colocando Cor nas linhas e adicionando Contagem de Data na área de valores, o resultado é diferente, como mostrado na Figura 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**

FIGURE 1-3 This report shows that if bidirectional filtering is not active, tables are not filtered.

O filtro nas linhas é a coluna Cor na tabela Produto. Como Produto está no lado um do relacionamento com Vendas, a Quantidade é filtrada corretamente. A Contagem de Nome do Produto é filtrada porque está calculando valores da tabela que está nas linhas, ou seja, Produto. O número inesperado é a Contagem de Data. De fato, ela sempre mostra o mesmo valor para todas as linhas - ou seja, o número total de linhas na tabela Data.

O filtro proveniente da coluna Cor não se propaga para Data porque o relacionamento entre Data e Vendas é unidirecional. Assim, embora Vendas tenha um filtro ativo, o filtro não pode se propagar para Data porque o tipo de relacionamento o impede.

Se alterarmos o relacionamento entre Data e Vendas para permitir filtro cruzado bidirecional, o resultado é como mostrado na Figura 1-4.
Os números agora refletem o número de dias em que pelo menos um produto da cor específica foi vendido. À primeira vista, pode parecer que todos os relacionamentos deveriam ser definidos como bidirecionais, para permitir

que o filtro se propague em qualquer direção e sempre retorne resultados que façam sentido. Como você aprenderá mais tarde neste livro, projetar um modelo de dados dessa maneira quase nunca é apropriado. Na verdade, dependendo do cenário com o qual você está trabalhando, você escolherá a propagação correta dos relacionamentos. Se seguir nossas sugestões, evitará o filtro bidirecional o máximo possível.

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**

FIGURE 1-4 If we enable bidirectional filtering, the *Date* table is filtered using the *Color* column.

Dax para usuários de excel

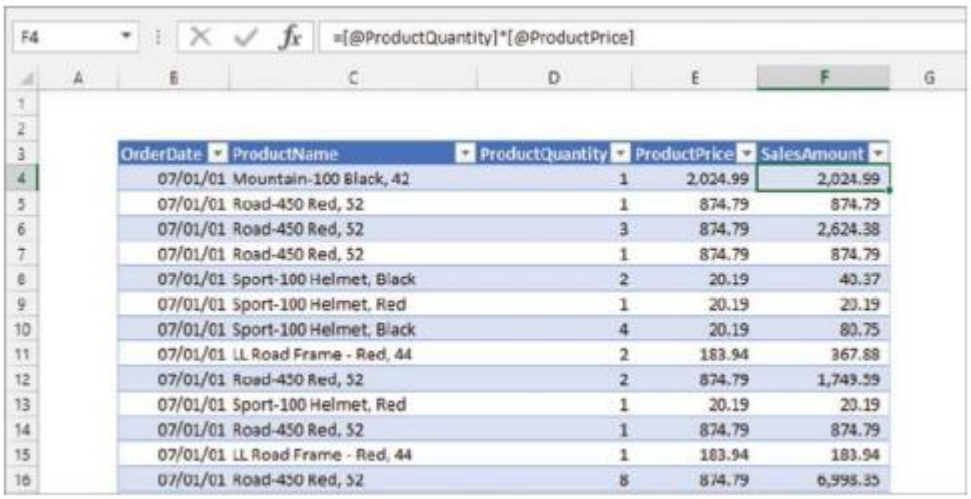
Chances are you already know the Excel formula language that DAX somewhat resembles. After all, the roots of DAX are in Power Pivot for Excel, and the development team tried to keep the two languages similar. This similarity makes the transition to this new language easier. However, there are some important differences.

Células vs Tabela

O Excel realiza cálculos sobre células. Uma célula é referenciada usando suas coordenadas. Assim, podemos escrever fórmulas da seguinte forma:

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

No DAX, o conceito de uma célula e suas coordenadas não existe. O DAX trabalha com tabelas e colunas, não com células. Como consequência, as expressões em DAX referem-se a tabelas e colunas, o que significa escrever código de maneira diferente. Os conceitos de tabelas e colunas não são novos no Excel. Na verdade, se definirmos um intervalo do Excel como uma tabela usando a função Formatar como Tabela, podemos escrever fórmulas no Excel que fazem referência a tabelas e colunas. Na Figura 1-5, a coluna SalesAmount avalia uma expressão que faz referência a colunas na mesma tabela em vez de células na planilha



F4 =[@ProductQuantity]*[@ProductPrice]

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

FIGURE 1-5 Excel can reference column names in tables.

Em Excel, nos referimos às colunas em uma tabela usando o formato [@NomeDaColuna]. NomeDaColuna é o nome da coluna a ser utilizada, e o símbolo @ significa 'pegar o valor da linha atual'. Embora a sintaxe não seja intuitiva, normalmente não escrevemos essas expressões. Elas aparecem quando clicamos em uma célula, e o Excel se encarrega de inserir o código correto para nós.

Você pode pensar no Excel como tendo duas maneiras diferentes de realizar cálculos. Podemos usar referências padrão de células, onde a fórmula para F4 seria E4*D4, ou podemos usar referências de colunas dentro de uma tabela. O uso de referências de colunas oferece a vantagem de que podemos usar a mesma expressão em todas as células de uma coluna, e o Excel calculará a fórmula com um valor diferente para cada linha.

Ao contrário do Excel, o DAX funciona apenas em tabelas. Todas as fórmulas devem fazer referência a colunas dentro de tabelas. Por exemplo, no DAX, escrevemos a multiplicação anterior da seguinte forma:

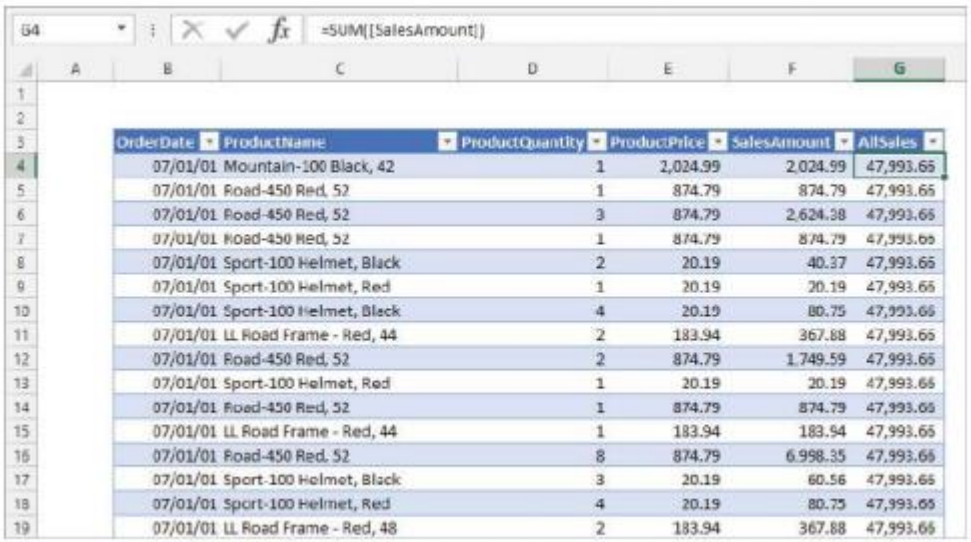
```
Sales[SalesAmount] = Sales[ProductPrice] * Sales[ProductQuantity]
```

Como você pode ver, cada coluna é prefixada com o nome de sua tabela. No Excel, não fornecemos o nome da tabela porque as fórmulas do Excel funcionam dentro de uma única tabela. No entanto, o DAX trabalha em um modelo de dados contendo muitas tabelas. Como consequência, devemos especificar o nome da tabela porque duas colunas em tabelas diferentes podem ter o mesmo nome.

Muitas funções no DAX funcionam da mesma forma que a função equivalente no Excel. Por exemplo, a função SE lê da mesma forma no DAX e no Excel

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

Um aspecto importante em que a sintaxe do Excel e do DAX é diferente é na maneira de referenciar a coluna inteira. Na verdade, em [@ProductQuantity], o @ significa 'o valor na linha atual'. No DAX, não é necessário especificar que um valor deve ser da linha atual, porque esse é o comportamento padrão da linguagem. No Excel, podemos fazer referência à coluna inteira, ou seja, todas as linhas dessa coluna, removendo o símbolo @. Você pode ver isso na Figura 1-6



G4 =SUM([SalesAmount])

OrderDate	ProductName	ProductQuantity	ProductPrice	SalesAmount	AllSales
07/01/01	Mountain-100 Black, 42	1	2,024.99	2,024.99	47,993.65
07/01/01	Road-450 Red, 52	1	874.79	874.79	47,993.65
07/01/01	Road-450 Red, 52	3	874.79	2,624.38	47,993.65
07/01/01	Road-450 Red, 52	1	874.79	874.79	47,993.65
07/01/01	Sport-100 Helmet, Black	2	20.19	40.37	47,993.65
07/01/01	Sport-100 Helmet, Red	1	20.19	20.19	47,993.65
07/01/01	Sport-100 Helmet, Black	4	20.19	80.75	47,993.65
07/01/01	LL Road Frame - Red, 44	2	183.94	367.88	47,993.65
07/01/01	Road-450 Red, 52	2	874.79	1,749.59	47,993.65
07/01/01	Sport-100 Helmet, Red	1	20.19	20.19	47,993.65
07/01/01	Road-450 Red, 52	1	874.79	874.79	47,993.65
07/01/01	LL Road Frame - Red, 44	1	183.94	183.94	47,993.65
07/01/01	Road-450 Red, 52	8	874.79	6,998.35	47,993.65
07/01/01	Sport-100 Helmet, Black	3	20.19	60.56	47,993.65
07/01/01	Sport-100 Helmet, Red	4	20.19	80.75	47,993.65
07/01/01	LL Road Frame - Red, 48	2	183.94	367.88	47,993.65

FIGURE 1-6 In Excel, you can reference an entire column by omitting the @ symbol before the column name.

O valor da coluna AllSales é o mesmo em todas as linhas porque é o total geral da coluna SalesAmount. Em outras palavras, há uma diferença sintática entre o valor de uma coluna na linha atual e o valor da coluna como um todo.

O DAX é diferente. No DAX, é assim que você escreve a expressão AllSales da Figura 1-6:
AllSales := SOMA (Sales[SalesAmount])

Não há diferença sintática entre obter o valor de uma coluna para uma linha específica e usar a coluna como um todo. O DAX entende que queremos somar todos os valores da coluna porque usamos o nome da coluna dentro de um agregador (neste caso, a função SOMA), que requer que um nome de coluna seja passado como parâmetro. Assim, embora o Excel exija uma sintaxe explícita para diferenciar entre os dois tipos de dados a serem recuperados, o DAX faz a dissociação automaticamente.

Essa distinção pode ser confusa, pelo menos no início.

Excel e DAX: Duas linguagens funcionais

Um aspecto em que as duas linguagens são similares é que tanto o Excel quanto o DAX são linguagens funcionais. Uma linguagem funcional é composta por expressões que são basicamente chamadas de função. No Excel e no DAX, os conceitos de instruções, loops e saltos não existem, embora sejam comuns a muitas linguagens de programação. No DAX, tudo é uma expressão. Este aspecto da linguagem muitas vezes é um desafio para programadores que vêm de diferentes linguagens, mas não deveria ser uma surpresa para os usuários do Excel.

Iteradores em DAX

Um conceito que pode ser novo para você é o conceito de iteradores. Ao trabalhar no Excel, você realiza cálculos passo a passo. O exemplo anterior mostrou que, para calcular o total de vendas, criamos uma coluna contendo o preço multiplicado pela quantidade. Em seguida, como segundo passo, somamos para calcular o total de vendas. Esse número é então útil como denominador para calcular a porcentagem de vendas de cada produto, por exemplo.

Usando o DAX, você pode realizar a mesma operação em uma única etapa usando iteradores. Um iterador faz exatamente o que seu nome sugere: ele itera sobre uma tabela e realiza um cálculo em cada linha da tabela, agregando o resultado para produzir o valor único solicitado.

Usando o exemplo anterior, agora podemos calcular a soma de todas as vendas usando o iterador SUMX:

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

Essa abordagem destaca tanto uma vantagem quanto uma desvantagem. A vantagem é que podemos realizar muitos cálculos complexos em uma única etapa sem nos preocuparmos em adicionar colunas que acabariam sendo úteis apenas para fórmulas específicas. A desvantagem é que programar com DAX é menos visual do que programar com o Excel. De fato, você não vê a coluna que calcula o preço multiplicado pela quantidade; ela existe apenas durante o cálculo.

Como explicaremos mais tarde, podemos criar uma coluna calculada que calcula a multiplicação do preço pela quantidade. No entanto, fazer isso raramente é uma boa prática porque consome memória e pode retardar os cálculos, a menos que você use DirectQuery e Aggregations, conforme explicamos no Capítulo 18, 'Otimizando VertiPaq.'"

DAX requer teorias

Vamos ser claros: o fato de que o DAX exige que se estude a teoria primeiro não é uma diferença entre linguagens de programação. Isso é uma diferença de mentalidade. Você provavelmente está acostumado a procurar na web por fórmulas complexas e padrões de solução para os cenários que está tentando resolver. Quando você está usando o Excel, é provável que encontre uma fórmula que quase faz o que você precisa. Você pode copiar a fórmula, personalizá-la para se adequar às suas necessidades e, em seguida, usá-la sem se preocupar muito com como ela funciona.

Essa abordagem, que funciona no Excel, não funciona com o DAX, no entanto. Você precisa estudar a teoria do DAX e entender completamente como os contextos de avaliação funcionam antes de poder escrever um código DAX eficiente. Se você não tiver uma base teórica adequada, perceberá que o DAX ou calcula valores como mágica ou calcula números estranhos que não fazem sentido. O problema não é o DAX, mas o fato de você ainda não ter entendido exatamente como o DAX funciona.

Felizmente, a teoria por trás do DAX se limita a alguns conceitos importantes, que explicamos no Capítulo 4, "Compreendendo os contextos de avaliação". Quando você chegar a esse capítulo, esteja preparado para uma aprendizagem intensa. Depois de dominar esse conteúdo, o DAX não terá mais segredos para você, e aprender DAX será principalmente uma questão de ganhar experiência. Lembre-se: saber é metade da batalha. Portanto, não tente avançar até que você esteja um pouco proficiente com os contextos de avaliação.

DAX para desenvolvedores SQL

Se você está acostumado com a linguagem SQL, já trabalhou com muitas tabelas e criou junções entre colunas para estabelecer relacionamentos. Sob esse ponto de vista, é provável que você se sinta em casa no mundo do DAX. De fato, realizar cálculos no DAX se resume a consultar um conjunto de tabelas unidas por relacionamentos e agregar valores.

Relationship handling

A primeira diferença entre SQL e DAX está na forma como os relacionamentos funcionam no modelo. No SQL, podemos definir chaves estrangeiras entre tabelas para declarar relacionamentos, mas o mecanismo nunca usa essas chaves estrangeiras em consultas, a menos que sejamos explícitos sobre elas. Por exemplo, se tivermos uma tabela Customers e uma tabela Sales, onde CustomerKey é uma chave primária em Customers e uma chave estrangeira em Sales, podemos escrever a seguinte consulta:

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

Mesmo que declaremos o relacionamento no modelo usando chaves estrangeiras, ainda precisamos ser explícitos e indicar a condição de junção na consulta. Embora essa abordagem torne as consultas mais verbosas, ela é útil porque permite usar diferentes condições de junção em consultas diferentes, proporcionando muita liberdade na maneira como você expressa consultas.

No DAX, os relacionamentos fazem parte do modelo e são todos LEFT OUTER JOINs. Quando são definidos no modelo, você não precisa mais especificar o tipo de junção na consulta: o DAX usa automaticamente um LEFT OUTER JOIN na consulta sempre que você utiliza colunas relacionadas à tabela principal. Assim, em DAX, você escreveria a consulta SQL anterior da seguinte forma

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

Como o DAX conhece o relacionamento existente entre Sales e Customers, ele realiza a junção automaticamente, seguindo o modelo. Finalmente, a função SUMMARIZECOLUMNS precisa realizar um agrupamento por Customers[CustomerName], mas não temos uma palavra-chave para isso: o SUMMARIZECOLUMNS agrupa automaticamente os dados pelas colunas selecionadas.

Dax é uma linguagem funcional

O SQL é uma linguagem declarativa. Você define o que precisa declarando o conjunto de dados que deseja recuperar usando instruções SELECT, sem se preocupar com como o mecanismo realmente obtém as informações.

O DAX, por outro lado, é uma linguagem funcional. No DAX, cada expressão é uma chamada de função. Os parâmetros da função podem, por sua vez, ser outras chamadas de função. A avaliação de parâmetros pode levar a planos de consulta complexos que o DAX executa para calcular o resultado.

Por exemplo, se quisermos recuperar apenas clientes que moram na Europa, podemos escrever esta consulta em 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
```

Usando DAX, não declaramos a condição WHERE na consulta. Em vez disso, usamos uma função específica (FILTER) para filtrar o resultado:

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

Você pode ver que FILTER é uma função: ela retorna apenas os clientes que moram na Europa, produzindo o resultado esperado. A ordem em que aninhamos as funções e os tipos de funções que usamos têm um forte impacto tanto no resultado quanto no desempenho do mecanismo. Isso também acontece no SQL, embora no SQL confiemos no otimizador de consultas para encontrar o plano de consulta ideal. No DAX, embora o otimizador de consultas faça um ótimo trabalho, você, como programador, tem mais responsabilidade em escrever um código eficiente.

DAX como uma linguagem de programação e consulta

No SQL, existe uma clara distinção entre a linguagem de consulta e a linguagem de programação, ou seja, o conjunto de instruções usado para criar stored procedures, views e outras partes do código no banco de dados. Cada dialeto SQL tem suas próprias instruções para permitir que os programadores enriqueçam o modelo de dados com código. No entanto, o DAX virtualmente não faz distinção entre consulta e programação. Um conjunto rico de funções manipula tabelas e pode, por sua vez, retornar tabelas. A função FILTER na consulta anterior é um bom exemplo disso.

Nesse aspecto, parece que o DAX é mais simples que o SQL. Quando você o aprende como uma linguagem de programação, seu uso original, você saberá tudo o que é necessário para também usá-lo como uma linguagem de consulta.

Uma das características mais poderosas do SQL como uma linguagem de consulta é a opção de usar subconsultas. O DAX possui conceitos semelhantes. No caso de subconsultas em DAX, no entanto, elas derivam da natureza funcional da linguagem.

Por exemplo, para recuperar clientes e vendas totais especificamente para os clientes que compraram mais de US$100, podemos escrever esta consulta em 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
```

Podemos obter o mesmo resultado em DAX aninhando chamadas de função

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

Neste código, a subconsulta que recupera CustomerName e SumOfSales é posteriormente alimentada em uma função FILTER que retém apenas as linhas em que SumOfSales é maior que 100. Neste momento, esse código pode parecer ilegível para você. No entanto, assim que você começar a aprender DAX, descobrirá que usar subconsultas é muito mais fácil do que no SQL, e flui naturalmente porque o DAX é uma linguagem funcional.

DAX para desenvolvedores MDX

Muitos profissionais de Business Intelligence começam a aprender DAX porque é a nova linguagem do Tabular. No passado, eles usavam a linguagem MDX para construir e consultar modelos Analysis Services Multidimensionais. Se você está entre eles, esteja preparado para aprender uma linguagem completamente nova: DAX e MDX não têm muita coisa em comum. Pior ainda, alguns conceitos em DAX vão lembrá-lo de conceitos semelhantes em MDX, embora sejam diferentes.

Em nossa experiência, descobrimos que aprender DAX após aprender MDX é a opção mais desafiadora. Para aprender DAX, você precisa liberar sua mente do MDX. Tente esquecer tudo o que você sabe sobre espaços multidimensionais e esteja preparado para aprender essa nova linguagem com uma mente clara.

Multidimensional versus Tabular

O MDX trabalha no espaço multidimensional definido por um modelo. A forma do espaço multidimensional é baseada na arquitetura de dimensões e hierarquias definidas no modelo, e isso, por sua vez, define o conjunto de coordenadas do espaço multidimensional. As interseções de conjuntos de membros em diferentes dimensões definem pontos no espaço multidimensional. Você pode ter levado algum tempo para perceber que o membro [All] de qualquer hierarquia de atributos é, de fato, um ponto no espaço multidimensional.

O DAX funciona de uma maneira muito mais simples. Não há dimensões, membros ou pontos no espaço multidimensional. Em outras palavras, não há espaço multidimensional algum. Existem hierarquias, que podemos definir no modelo, mas são diferentes das hierarquias em MDX. O espaço DAX é construído em cima de tabelas, colunas e relacionamentos. Cada tabela em um modelo Tabular não é nem um grupo de medidas nem uma dimensão: é apenas uma tabela, e para calcular valores, você a percorre, a filtra ou soma valores dentro dela. Tudo é baseado nos dois conceitos simples de tabelas e relacionamentos.

Você logo descobrirá que, do ponto de vista da modelagem, o Tabular oferece menos opções do que o Multidimensional. Neste caso, ter menos opções não significa ser menos poderoso, porque você pode usar o DAX como uma linguagem de programação para enriquecer o modelo. O verdadeiro poder de modelagem do Tabular está na velocidade tremenda do DAX. De fato, você provavelmente tenta evitar o uso excessivo de MDX em seu modelo, porque otimizar a velocidade do MDX muitas vezes é um desafio. O DAX, por outro lado, é surpreendentemente rápido. Assim, a maior parte da complexidade dos cálculos não está no modelo, mas nas fórmulas DAX.

DAX como linguagem de programação e consulta

DAX e MDX são ambas linguagens de programação e linguagens de consulta. No MDX, a diferença é evidenciada pela presença do script MDX. Você usa o MDX no script MDX, juntamente com várias instruções especiais que podem ser usadas apenas no script, como instruções SCOPE. Você usa o MDX em consultas ao escrever instruções SELECT que recuperam dados. No DAX, isso é um pouco diferente. Você usa o DAX como uma linguagem de programação para definir colunas calculadas, tabelas calculadas e medidas. O conceito de colunas calculadas e tabelas calculadas é novo para o DAX e não existe no MDX; medidas são semelhantes aos membros calculados no MDX. Você também pode usar o DAX como linguagem de consulta, por exemplo, para recuperar dados de um modelo Tabular usando o Reporting Services. No entanto, as funções DAX não têm um papel específico e podem ser usadas tanto em consultas quanto em expressões de cálculo. Além disso, você também pode consultar um modelo Tabular usando MDX. Assim, a parte de consulta do MDX funciona com modelos Tabulares, enquanto o DAX é a única opção quando se trata de programar um modelo Tabular.

Hierarquias

No MDX, você depende de hierarquias para realizar a maioria dos cálculos. Se você quisesse calcular as vendas no ano anterior, teria que recuperar o PrevMember do CurrentMember na hierarquia Year e usá-lo para substituir o filtro MDX. Por exemplo, você pode escrever a fórmula desta maneira para definir um cálculo de ano anterior no MDX:

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

A medida utiliza a função ParallelPeriod, que retorna o elemento correspondente do CurrentMember na hierarquia Calendar. Assim, ela é baseada nas hierarquias definidas no modelo. Escreveríamos o mesmo cálculo em DAX utilizando contextos de filtro e funções padrão de inteligência de tempo:

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

Podemos escrever o mesmo cálculo de muitas outras maneiras usando FILTER e outras funções DAX, mas a ideia permanece a mesma: em vez de usar hierarquias, filtramos tabelas. Essa diferença é enorme, e você provavelmente sentirá falta de cálculos de hierarquia até se acostumar com o DAX.

Outra diferença importante é que no MDX você se refere a [Measures].[Sales Amount], e a função de agregação que você precisa usar já está definida no modelo. No DAX, não há agregação predefinida. Na verdade, como você pode ter notado, a expressão para calcular é SUM(Sales[Sales Amount]). A agregação predefinida não está mais no modelo. Precisamos defini-la sempre que quisermos usá-la. Podemos sempre criar uma medida que calcule a soma das vendas, mas isso estaria além do escopo desta seção e é explicado posteriormente no livro.

Mais uma diferença importante entre DAX e MDX é que o MDX faz uso intensivo da instrução SCOPE para implementar lógica de negócios (novamente, usando hierarquias), enquanto o DAX precisa de uma abordagem completamente diferente. De fato, o manuseio de hierarquias está ausente na linguagem como um todo.

Por exemplo, se quisermos limpar uma medida no nível do ano, em MDX escreveríamos esta instrução:

```
SCOPE ( [Measures].[SamePeriodPreviousYearSales], [Date].[Month].[All] )
 THIS = NULL;
END SCOPE;
DAX does not have something like a SCOPE statement. To obtain the same result, we need to check
for the presence of fi lters in the fi lter context, and the scenario is much more complex:
SamePeriodPreviousYearSales :=
IF (
 ISINSCOPE ( 'Date'[Month] ),
 CALCULATE (
 SUM ( Sales[Sales Amount] ),
 SAMEPERIODLASTYEAR ( 'Date'[Date] )
 ),
 BLANK ()
)
```

De forma intuitiva, esta fórmula retorna um valor apenas se o usuário estiver navegando pela hierarquia do calendário no nível do mês ou abaixo. Caso contrário, retorna um BLANK. Você aprenderá mais tarde o que esta fórmula realmente calcula em detalhes. É muito mais propenso a erros do que o código MDX equivalente. Para ser honesto, o manuseio de hierarquias é uma das características que realmente falta no DAX.

Cálculos em nível de folha

Por fim, ao usar o MDX, você provavelmente se acostumou a evitar cálculos no nível das folhas. Realizar cálculos no nível das folhas em MDX acaba sendo tão lento que você sempre deve preferir pré-calcular valores e aproveitar as agregações para obter resultados. No DAX, os cálculos no nível das folhas funcionam incrivelmente rápido, e as agregações têm um propósito diferente, sendo úteis apenas para conjuntos de dados grandes. Isso requer uma mudança de mentalidade na hora de construir os modelos de dados. Na maioria dos casos, um modelo de dados que se encaixa perfeitamente no SSAS Multidimensional não é a escolha certa para o Tabular e vice-versa.

DAX para usuários de Power BI

Se você pulou as seções anteriores e veio diretamente aqui, bem-vindo! DAX é a linguagem nativa do Power BI, e se você não tem experiência em Excel, SQL ou MDX, o Power BI será o primeiro lugar onde você aprenderá DAX. Se você não tem experiência anterior na construção de modelos com outras ferramentas, descobrirá que o Power BI é uma ferramenta analítica e de modelagem poderosa, com o DAX como o companheiro perfeito.

Se você começou a usar o Power BI há algum tempo e agora deseja ir para o próximo nível, esteja preparado para uma jornada maravilhosa com o DAX. Aqui está nosso conselho para você: não espere poder escrever código DAX complexo em questão de alguns dias. O DAX requer seu tempo e dedicação, e dominá-lo exige alguma prática. Com base em nossa experiência, você ficará empolgado no início quando for recompensado com alguns cálculos simples.

A empolgação diminui assim que você começa a aprender sobre os contextos de avaliação e o CALCULATE, os tópicos mais complexos da linguagem. Nesse ponto, tudo parece complexo. Não desista; a maioria dos desenvolvedores de DAX teve que superar esse nível. Quando estiver lá, você estará tão perto de alcançar uma compreensão completa que seria uma pena real parar. Leia e pratique repetidamente porque uma lâmpada vai acender muito mais rápido do que você espera. Você será capaz de terminar o livro rapidamente, alcançando o status de guru do DAX.

Os contextos de avaliação estão no cerne da linguagem. Dominá-los leva tempo. Não conhecemos ninguém que tenha conseguido aprender tudo sobre DAX em apenas alguns dias. Além disso, como acontece com qualquer tópico complexo, você aprenderá a apreciar muitos detalhes ao longo do tempo. Quando você acha que aprendeu tudo, dê uma segunda leitura no livro. Você descobrirá muitos detalhes que pareciam menos importantes à primeira vista, mas, com uma mentalidade mais treinada, fazem uma grande diferença.