Introdução DAX

Neste capítulo, começamos a discutir a linguagem DAX. Aqui, você aprenderá a sintaxe da linguagem, a diferença entre uma coluna calculada e uma medida (também chamada de campo calculado, em certas versões antigas do Excel) e as funções mais comumente usadas no DAX.

Por ser um capítulo introdutório, não abordaremos muitas funções em detalhes. Em capítulos posteriores, as explicaremos de maneira mais aprofundada. Por enquanto, a introdução às funções e a análise geral da linguagem DAX são suficientes. Ao nos referirmos a recursos do modelo de dados no Power BI, Power Pivot ou Analysis Services, usamos o termo "Tabular", mesmo quando o recurso não está presente em todos os produtos.
Por exemplo, "DirectQuery em Tabular" se refere ao recurso de modo DirectQuery disponível no Power BI e no Analysis Services, mas não no Excel. Essa abordagem visa fornecer uma compreensão abrangente da linguagem DAX, independentemente do ambiente específico em que está sendo aplicada.

Compreendendo os cálculos DAX

Antes de trabalhar com fórmulas mais complexas, é necessário aprender os conceitos básicos do DAX. Isso inclui a sintaxe do DAX, os diferentes tipos de dados que o DAX pode manipular, os operadores básicos e como referenciar colunas e tabelas. Esses conceitos serão discutidos nas próximas seções.

No DAX, calculamos valores em colunas de tabelas. Podemos agregar, calcular e procurar por números, mas, no final, todos os cálculos envolvem tabelas e colunas. Portanto, a primeira sintaxe que você precisa aprender é como fazer referência a uma coluna em uma tabela.

O formato geral é escrever o nome da tabela entre aspas simples, seguido pelo nome da coluna entre colchetes, como no exemplo:

```
'Sales'[Quantity]
```

Podemos omitir as aspas simples se o nome da tabela não começar com um número, não conter espaços e não for uma palavra reservada (como Date ou Sum).

O nome da tabela também é opcional caso estejamos fazendo referência a uma coluna ou medida dentro da tabela onde definimos a fórmula. Assim, [Quantity] é uma referência de coluna válida, se escrita em uma coluna calculada ou em uma medida definida na tabela de Vendas. Apesar dessa opção estar disponível, desencorajamos fortemente omitir o nome da tabela. Neste ponto, não explicamos por que isso é tão importante, mas a razão ficará clara quando você ler o Capítulo 5, "Compreendendo CALCULATE e CALCULATETABLE". Mesmo assim, é de suma importância poder distinguir entre. medidas (discutidas mais tarde) e colunas ao ler o código DAX. O padrão de fato é sempre usar o nome da tabela em referências de colunas e sempre evitá-lo em referências de medidas. Quanto mais cedo você começar a adotar este padrão, mais fácil será sua vida com DAX. Portanto, você deve se acostumar
com esta forma de referenciar colunas e medidas:

```
Sales[Quantity] * 2 -- This is a column reference
[Sales Amount] * 2 -- This is a measure reference
```

Você aprenderá a lógica por trás desse padrão após aprender sobre a transição de contexto, que é abordada no Capítulo 5. Por enquanto, apenas confie em nós e adote este padrão.

Comentários em DAX

O exemplo de código anterior mostra os comentários em DAX pela primeira vez. DAX suporta comentários de linha única e comentários de várias linhas. Comentários de linha única começam com -- ou //, e a parte restante da linha é considerada um comentário.

```
= Sales[Quantity] * Sales[Net Price] -- Comentário de linha única
= Sales[Quantity] * Sales[Unit Cost] // Outro exemplo de comentário de linha única
```

Um comentário de várias linhas começa com /* e termina com */. O analisador DAX ignora tudo incluído entre esses

marcadores e os considera um comentário.

```
=
IF (
    Sales[Quantity] > 1,
    /* Primeiro exemplo de um comentário de várias linhas
Tudo pode ser escrito aqui e é ignorado pelo DAX
*/
    "Multi",
    /* Um caso comum de uso de comentários de várias linhas é comentar uma parte
do código existente
A próxima instrução IF é ignorada porque está dentro de um comentário de várias linhas
IF (
Sales[Quantity] = 1,
"Single",
"Nota especial"
)
*/
    "Single"
)
```

É melhor evitar comentários no final de uma expressão DAX em uma definição de medida, coluna calculada ou tabela calculada. Esses comentários podem não ser visíveis à primeira vista e podem não ser suportados por ferramentas como o DAX Formatter, que é discutido mais tarde neste capítulo.

Tipos de dados em DAX

DAX pode realizar cálculos com diferentes tipos numéricos, dos quais existem sete. Ao longo do tempo, a Microsoft introduziu diferentes nomes para os mesmos tipos de dados, criando algum tipo de confusão. A Tabela 2-1 fornece os diferentes nomes sob os quais você pode encontrar cada tipo de dados em DAX.

TABLE 2-1 Data Types

DAX Data Type	Power BI Data Type	Power Pivot and Analysis Services Data Type	Correspondent Conventional Data Type (e.g., SQL Server)	Tabular Object Model (TOM) Data Type
Integer	Whole Number	Whole Number	Integer / INT	int64
Decimal	Decimal Number	Decimal Number	Floating point / DOUBLE	double
Currency	Fixed Decimal Number	Currency	Currency / MONEY	decimal
DateTime	DateTime, Date, Time	Date	Date / DATETIME	dateTime
Boolean	True/False	True/False	Boolean / BIT	boolean
String	Text	Text	String / NVARCHAR(MAX)	string
Variant	-	-	-	variant
Binary	Binary	Binary	Blob / VARBINARY(MAX)	binary

Neste livro, usamos os nomes na primeira coluna da Tabela 2-1, aderindo aos padrões de facto na comunidade de banco de dados e Business Intelligence. Por exemplo, no Power BI, uma coluna contendo TRUE ou FALSE seria chamada TRUE/FALSE, enquanto no SQL Server seria chamada BIT. No entanto, o nome histórico e mais comum para esse tipo de valor é Booleano.

DAX vem com um sistema poderoso de manipulação de tipos para que não tenhamos que nos preocupar com os tipos de dados. Em uma expressão DAX, o tipo resultante é baseado no tipo do termo usado na expressão. Você precisa estar ciente disso no caso de o tipo retornado de uma expressão DAX não ser o tipo esperado; você teria que investigar o tipo de dados dos termos usados na própria expressão.

Por exemplo, se um dos termos de uma soma for uma data, o resultado também será uma data; da mesma forma, se o mesmo operador for usado com inteiros, o resultado será um inteiro. Esse comportamento é conhecido como

sobrecarga de operador, e um exemplo é mostrado na Figura 2-1, onde a coluna OrderDatePlusOneWeek é calculada adicionando 7 ao valor da coluna Order Date

```
Sales[OrderDatePlusOneWeek] = Sales[Order Date] + 7
```

O resultado é uma data.

Order Date	OrderDatePlusOneWeek
10/08/2008	10/15/2008
10/10/2008	10/17/2008
10/12/2008	10/19/2008
09/05/2008	09/12/2008
09/07/2008	09/14/2008
09/23/2008	09/30/2008
11/05/2008	11/12/2008
11/07/2008	11/14/2008
11/09/2008	11/16/2008
11/17/2008	11/24/2008

FIGURE 2-1 Adding an integer to a date results in a date increased by the corresponding number of days.

Além da sobrecarga de operadores, o DAX converte automaticamente strings em números e números em strings sempre que necessário pelo operador. Por exemplo, se usarmos o operador &, que concatena strings, o DAX

converte seus argumentos em strings. A fórmula a seguir retorna "54" como uma string:

```
= 5 & 4
```

Por outro lado, esta fórmula retorna um resultado inteiro com o valor 9:

```
= "5" + "4"
```

O valor resultante depende do operador e não das colunas de origem, que são convertidas conforme os requisitos do operador. Embora esse comportamento pareça conveniente, mais adiante neste capítulo você verá que tipos de erros podem ocorrer durante essas conversões automáticas. Além disso, nem todos os operadores seguem esse comportamento. Por exemplo, operadores de comparação não podem comparar strings com números. Consequentemente, você pode adicionar um número a uma string, mas não pode comparar um número com uma string. Você pode encontrar uma referência completa aqui: https://docs.microsoft.com/en-us/power-bi/desktopdata-types. Como as regras são tão complexas, sugerimos evitar conversões automáticas. Se uma conversão precisar ocorrer, recomendamos que você a controle e torne a conversão explícita. Para ser mais explícito, o exemplo anterior deveria ser escrito assim:

```
= VALUE ( "5" ) + VALUE ( "4" )
```

Pessoas acostumadas a trabalhar com Excel ou outras linguagens podem estar familiarizadas com os tipos de dados DAX. Alguns detalhes sobre os tipos de dados dependem do mecanismo e podem ser diferentes para Power BI, Power Pivot ou Analysis Services. Você pode encontrar informações mais detalhadas sobre os tipos de dados DAX do Analysis Services em https://learn.microsoft.com/pt-br/analysis-services/tabular-models/data-types-supported-ssastabular?view=asallproducts-allversions&redirectedfrom=MSDN, e informações sobre o Power BI estão disponíveis em https://learn.microsoft.com/pt-br/power-bi/connect-data/desktop-data-types. No entanto, é útil compartilhar algumas considerações sobre cada um desses tipos de dados.

Inteiro

DAX tem apenas um tipo de dado Inteiro que pode armazenar um valor de 64 bits. Todas as operações internas entre valores inteiros em DAX também usam um valor de 64 bits.

Decimal

Um número Decimal é sempre armazenado como um valor de ponto flutuante de dupla precisão. Não confunda este tipo de dado DAX com o tipo de dado decimal e numérico do Transact-SQL. O tipo de dado correspondente de um número decimal DAX no SQL é Float.

Currency

O tipo de dado Currency, também conhecido como Número Decimal Fixo no Power BI, armazena um número decimal fixo. Ele pode representar quatro casas decimais e é armazenado internamente como um valor inteiro de 64 bits dividido por 10.000. Somar ou subtrair tipos de dados Currency sempre ignora decimais além da quarta casa decimal, enquanto a multiplicação e divisão produzem um valor de ponto flutuante, aumentando assim a precisão do resultado. Em geral, se precisarmos de mais precisão do que as quatro casas decimais fornecidas, devemos usar o tipo de dado Decimal.
O formato padrão do tipo de dado Currency inclui o símbolo da moeda. Também podemos aplicar a formatação de moeda a números inteiros e decimais, e podemos usar um formato sem o símbolo da moeda para um tipo de dado Currency.

DateTime

O DAX armazena datas como um tipo de dados DateTime. Esse formato usa um número de ponto flutuante internamente, onde a parte inteira corresponde ao número de dias desde 30 de dezembro de 1899, e a parte decimal identifica a fração do dia. Horas, minutos e segundos são convertidos em frações decimais de um dia. Portanto, a seguinte expressão retorna a data atual mais um dia (exatamente 24 horas):

```
= TODAY () + 1
```

O resultado é a data de amanhã no momento da avaliação. Se você precisa apenas da parte da data de um DateTime, sempre lembre-se de usar o TRUNC para se livrar da parte decimal. O Power BI oferece dois tipos de dados adicionais: Data e Hora. Internamente, eles são uma variação simples de DateTime. Na verdade, Data e Hora armazenam apenas a parte inteira ou a parte decimal do DateTime, respectivamente.

O bug do ano bissexto

O Lotus 1-2-3, uma planilha popular lançada em 1983, apresentava um bug no tratamento do tipo de dados DateTime. Ele considerava 1900 como um ano bissexto, embora não fosse. O último ano de um século é um ano bissexto apenas se os dois primeiros dígitos puderem ser divididos por 4 sem deixar resto. Naquela época, a equipe de desenvolvimento da primeira versão do Excel deliberadamente replicou o bug para manter a compatibilidade com o Lotus 1-2-3. Desde então, cada nova versão do Excel manteve o bug para compatibilidade.
No momento da impressão em 2019, o bug ainda está presente no DAX, introduzido para compatibilidade retroativa com o Excel. A presença do bug (deveríamos chamá-lo de recurso?) pode levar a erros em períodos anteriores a 1º de março de 1900. Portanto, por design, a primeira data oficialmente suportada pelo DAX é 1º de março de 1900. Cálculos de datas executados em períodos anteriores a essa data podem levar a erros e devem ser considerados imprecisos.

Boolean

O tipo de dados Boolean é usado para expressar condições lógicas. Por exemplo, uma coluna calculada definida pela seguinte expressão é do tipo Boolean:
= Sales[Unit Price] > Sales[Unit Cost]
Você também verá tipos de dados Booleanos como números, onde VERDADEIRO é igual a 1 e FALSO é igual a 0. Essa notação às vezes é útil para fins de ordenação, porque VERDADEIRO > FALSO.

String

Cada string em DAX é armazenada como uma string Unicode, onde cada caractere é armazenado em 16 bits. Por

padrão, a comparação entre strings não diferencia maiúsculas de minúsculas, então as duas strings "Power BI" e "POWER BI" são consideradas iguais.

Variant

O tipo de dados Variant é usado para expressões que podem retornar diferentes tipos de dados, dependendo das condições. Por exemplo, a seguinte declaração pode retornar tanto um número inteiro quanto uma string, portanto, retorna um tipo variante:

```
IF ( [measure] > 0, 1, "N/A" )
```

O tipo de dados Variant não pode ser usado como tipo de dados para uma coluna em uma tabela regular. Uma medida DAX e, em geral, uma expressão DAX podem ser do tipo Variant.

Binary

O tipo de dados Binary é usado no modelo de dados para armazenar imagens ou outros tipos não estruturados de informações. Não está disponível no DAX. Era usado principalmente pelo Power View, mas pode não estar disponível em outras ferramentas, como o Power BI.

Operadores DAX

Agora que você viu a importância dos operadores em determinar o tipo de uma expressão, veja a Tabela 2-2, que fornece uma lista dos operadores disponíveis no DAX.

TABLE 2-2 Operators

Operator Type	Symbol	Use	Example
Parenthesis	( )	Precedence order and grouping of arguments	(5 + 2) * 3
Arithmetic	+ - * /	Addition Subtraction/negation Multiplication Division	4 + 2 5 – 3 4 * 2 4 / 2
Comparison	= <> > >= < <=	Equal to Not equal to Greater than Greater than or equal to Less than Less than or equal to	[CountryRegion] = "USA" [CountryRegion] <> "USA" [Quantity] > 0 [Quantity] >= 100 [Quantity] < 0 [Quantity] <= 100
Text concatenation	&	Concatenation of strings	"Value is" & [Amount]
Logical	&& || IN NOT	AND condition between two Boolean expressions OR condition between two Boolean expressions Inclusion of an element in a list Boolean negation	[CountryRegion] = "USA" && [Quantity]>0 [CountryRegion] = "USA" || [Quantity] > 0 [CountryRegion] IN {"USA", "Canada"} NOT [Quantity] > 0

Além disso, os operadores lógicos também estão disponíveis como funções DAX, com uma sintaxe semelhante à do Excel. Por exemplo, podemos escrever expressões como estas:

```
AND ( [CountryRegion] = "USA", [Quantity] > 0 )
OR ( [CountryRegion] = "USA", [Quantity] > 0 )
```

Esses exemplos são equivalentes, respectivamente, aos seguintes:

```
[CountryRegion] = "USA" && [Quantity] > 0
[CountryRegion] = "USA" || [Quantity] > 0
```

O uso de funções em vez de operadores para lógica booleana torna-se útil ao escrever condições complexas. Na verdade, quando se trata de formatar grandes seções de código, as funções são muito mais fáceis de formatar e ler do que os operadores. No entanto, uma grande desvantagem das funções é que só podemos passar dois parâmetros de cada vez. Portanto, precisamos aninhar funções se tivermos mais de duas condições para avaliar.

Constructores de Tabelas

No DAX, podemos definir tabelas anônimas diretamente no código. Se a tabela tiver uma única coluna, a sintaxe requer apenas uma lista de valores - um para cada linha - delimitados por chaves. Podemos delimitar várias linhas

por parênteses, que são opcionais se a tabela for composta por uma única coluna. As duas definições a seguir, por exemplo, são equivalentes:

```
{ "Vermelho", "Azul", "Branco" }
{ ( "Vermelho" ), ( "Azul" ), ( "Branco" ) }
```

Se a tabela tiver várias colunas, os parênteses são obrigatórios. Cada coluna deve ter o mesmo tipo de dados em todas as suas linhas; caso contrário, o DAX converterá automaticamente a coluna para um tipo de dados que pode acomodar todos os tipos de dados fornecidos em diferentes linhas para a mesma coluna.

```
{
( "A", 10, 1.5, DATE ( 2017, 1, 1 ), CURRENCY ( 199.99 ), TRUE ),
( "B", 20, 2.5, DATE ( 2017, 1, 2 ), CURRENCY ( 249.99 ), FALSE ),
( "C", 30, 3.5, DATE ( 2017, 1, 3 ), CURRENCY ( 299.99 ), FALSE )
}
```

O construtor de tabelas é comumente usado com o operador IN. Por exemplo, as seguintes são sintaxes possíveis e válidas em um predicado DAX:

```
'Produto'[Cor] IN { "Vermelho", "Azul", "Branco" }
( 'Data'[Ano], 'Data'[NúmeroMês] ) IN { ( 2017, 12 ), ( 2018, 1 ) }
```

Este segundo exemplo mostra a sintaxe necessária para comparar um conjunto de colunas (tupla) usando o operador IN. Tal sintaxe não pode ser usada com um operador de comparação. Em outras palavras, a seguinte sintaxe não é válida:

```
( 'Data'[Ano], 'Data'[NúmeroMês] ) = ( 2007, 12 )
```

No entanto, podemos reescrevê-lo usando o operador IN com um construtor de tabela que tem uma única linha, como no exemplo a seguir:

```
( 'Data'[Ano], 'Data'[NúmeroMês] ) IN { ( 2007, 12 ) }
```

Declarações Condicionais

No DAX, podemos escrever uma expressão condicional usando a função IF. Por exemplo, podemos escrever uma expressão que retorna MULTI ou SINGLE, dependendo se o valor da quantidade for maior que um ou não, respectivamente.

```
IF (
Vendas[Quantidade] > 1,
"MULTI",
"SINGLE"
)
```

A função IF tem três parâmetros, mas apenas os dois primeiros são obrigatórios. O terceiro é opcional e tem um valor padrão de BLANK. Considere o seguinte código:

```
IF (
Vendas[Quantidade] > 1,
Vendas[Quantidade]
)
```

Corresponde à seguinte versão explícita:

```
IF (
Vendas[Quantidade] > 1,
Vendas[Quantidade],
BLANK ()
```

)

Compreendendo colunas calculadas e medidas

Agora que você conhece os conceitos básicos da sintaxe DAX, precisa aprender um dos conceitos mais importantes do DAX: a diferença entre colunas calculadas e medidas. Embora colunas calculadas e medidas possam parecer similares à primeira vista porque você pode realizar certos cálculos usando ambas, elas são, na realidade, diferentes. Compreender a diferença é fundamental para desbloquear o poder do DAX.

Colunas Calculadas

Dependendo da ferramenta que você está usando, você pode criar uma coluna calculada de maneiras diferentes. Na verdade, o conceito permanece o mesmo: uma coluna calculada é uma nova coluna adicionada ao seu modelo, mas em vez de ser carregada de uma fonte de dados, ela é criada recorrendo a uma fórmula DAX.

Uma coluna calculada é como qualquer outra coluna em uma tabela, e podemos usá-la em linhas, colunas, filtros ou valores de uma matriz ou qualquer outro relatório. Também podemos usar uma coluna calculada para definir um relacionamento, se necessário. A expressão DAX definida para uma coluna calculada opera no contexto da linha atual da tabela à qual a coluna calculada pertence. Qualquer referência a uma coluna retorna o valor dessa coluna para a linha atual. Não podemos acessar diretamente os valores de outras linhas.

Se você estiver usando o Modo de Importação padrão do Tabular e não estiver usando o DirectQuery, um conceito importante a lembrar sobre colunas calculadas é que essas colunas são calculadas durante o processamento do banco de dados e depois armazenadas no modelo. Esse conceito pode parecer estranho se você estiver acostumado a colunas calculadas no SQL (não persistidas), que são avaliadas no momento da consulta e não usam memória. No Tabular, no entanto, todas as colunas calculadas ocupam espaço na memória e são calculadas durante o processamento da tabela.

Esse comportamento é útil sempre que criamos colunas calculadas complexas. O tempo necessário para calcular colunas calculadas complexas é sempre o tempo de processamento e não o tempo de consulta, resultando em uma melhor experiência do usuário. No entanto, esteja ciente de que uma coluna calculada usa RAM preciosa. Por exemplo, se tivermos uma fórmula complexa para uma coluna calculada, podemos sentir a tentação de separar as etapas do cálculo em colunas intermediárias diferentes. Embora essa técnica seja útil durante o desenvolvimento do projeto, é um hábito ruim na produção porque cada cálculo intermediário é armazenado na RAM e desperdiça espaço valioso.

Se um modelo for baseado em DirectQuery, o comportamento é muito diferente. No modo DirectQuery, as colunas calculadas são calculadas na hora em que o mecanismo Tabular consulta a fonte de dados. Isso pode resultar em consultas pesadas executadas pela fonte de dados, produzindo modelos lentos.

Calcular a duração de um pedido

Imagine que temos uma tabela de Vendas contendo tanto a data do pedido quanto a data de entrega. Usando essas duas colunas, podemos calcular o número de dias envolvidos na entrega do pedido. Como as datas são armazenadas como o número de dias após 30/12/1899, uma simples subtração calcula a diferença em dias entre duas datas:

```
Sales[DaysToDeliver] = Sales[Delivery Date] - Sales[Order Date]
```

No entanto, porque as duas colunas usadas para a subtração são datas, o resultado também é uma data. Para produzir um resultado numérico, converta o resultado para um número inteiro desta forma:

```
Sales[DaysToDeliver] = INT(Sales[Delivery Date] - Sales[Order Date])
```

O resultado é mostrado na Figura 2-2.

Order Date	Delivery Date	DaysToDeliver
01/02/2007	01/08/2007	6
01/02/2007	01/09/2007	7
01/02/2007	01/10/2007	8
01/02/2007	01/11/2007	9
01/02/2007	01/12/2007	10
01/02/2007	01/13/2007	11
01/02/2007	01/14/2007	12

FIGURE 2-2 By subtracting two dates and converting the result to an integer, DAX computes the number of days between the two dates.

Medidas

Colunas calculadas são úteis, mas você pode definir cálculos em um modelo DAX de outra maneira. Sempre que você não quiser calcular valores para cada linha, mas sim desejar agregar valores de muitas linhas em uma tabela, você encontrará esses cálculos úteis; eles são chamados de medidas.

```
Sales[SalesAmount] = Sales[Quantity] * Sales[Net Price]
Sales[TotalCost] = Sales[Quantity] * Sales[Unit Cost]
Sales[GrossMargin] = Sales[SalesAmount] - Sales[TotalCost]
```

O que acontece se você quiser mostrar a margem bruta como uma porcentagem do valor das vendas? Você poderia criar uma coluna calculada com a seguinte fórmula:

```
Sales[GrossMarginPct] = Sales[GrossMargin] / Sales[SalesAmount]
```

Esta fórmula calcula o valor correto no nível da linha - como você pode ver na Figura 2-3 - mas no nível do total geral, o resultado está claramente incorreto.

SalesKey	SalesAmount	TotalCost	GrossMargin	GrossMarginPct
20070104611301-0002	$72.19	$38.74	$33.45	46.34%
20070104611301-0003	$23.75	$11.50	$12.25	51.58%
20070104611320-0006	$216.57	$116.22	$100.35	46.34%
20070104611320-0007	$23.75	$11.50	$12.25	51.58%
20070104611506-0002	$72.19	$38.74	$33.45	46.34%
20070104611506-0003	$23.75	$11.50	$12.25	51.58%
20070104611914-0002	$64.59	$38.74	$25.85	40.02%
20070104611914-0003	$21.25	$11.50	$9.75	45.88%
20070104611952-0004	$64.59	$38.74	$25.85	40.02%
20070104611952-0005	$21.25	$11.50	$9.75	45.88%
20070104611998-0002	$64.59	$38.74	$25.85	40.02%
20070104611998-0003	$63.75	$34.50	$29.25	45.88%
Total	**$732.23**	**$401.92**	**$330.31**	**551.46%**

FIGURE 2-3 The *GrossMarginPct* column shows a correct value on each row, but the grand total is incorrect.

Ao computar o valor agregado de uma porcentagem, não podemos depender de colunas calculadas. Em vez disso, é necessário calcular a porcentagem com base na soma das colunas individuais. O valor agregado deve ser calculado como a soma da margem bruta dividida pela soma do valor das vendas. Neste caso, é preciso calcular a proporção nos agregados; não se pode utilizar uma agregação de colunas calculadas. Em outras palavras, a proporção das somas deve ser calculada, não a soma das proporções.

Seria igualmente incorreto simplesmente alterar a agregação da coluna GrossMarginPct para uma média e confiar no resultado, pois isso forneceria uma avaliação incorreta da porcentagem, sem considerar as diferenças entre os valores. O resultado desse valor médio é visível na Figura 2-4, e é possível verificar facilmente que (330,31 / 732,23) não é igual ao valor exibido, 45,96%; deveria ser 45,11% em vez disso.

SalesKey	SalesAmount	TotalCost	GrossMargin	Average of GrossMarginPct
20070104611301-0002	$72.19	$38.74	$33.45	46.34%
20070104611301-0003	$23.75	$11.50	$12.25	51.58%
20070104611320-0006	$216.57	$116.22	$100.35	46.34%
20070104611320-0007	$23.75	$11.50	$12.25	51.58%
20070104611506-0002	$72.19	$38.74	$33.45	46.34%
20070104611506-0003	$23.75	$11.50	$12.25	51.58%
20070104611914-0002	$64.59	$38.74	$25.85	40.02%
20070104611914-0003	$21.25	$11.50	$9.75	45.88%
20070104611952-0004	$64.59	$38.74	$25.85	40.02%
20070104611952-0005	$21.25	$11.50	$9.75	45.88%
20070104611998-0002	$64.59	$38.74	$25.85	40.02%
20070104611998-0003	$63.75	$34.50	$29.25	45.88%
Total	**$732.23**	**$401.92**	**$330.31**	**45.96%**

FIGURE 2-4 Changing the aggregation method to *AVERAGE* does not provide the correct result.

A implementação correta para GrossMarginPct é por meio de uma medida (measure):

```
GrossMarginPct := SUM ( Sales[GrossMargin] ) / SUM (Sales[SalesAmount] )
```

Como já mencionamos, o resultado correto não pode ser alcançado com uma coluna calculada. Se for necessário operar em valores agregados em vez de operar linha por linha, é necessário criar medidas. Você pode ter notado que usamos := para definir uma medida em vez do sinal de igual (=). Isso é um padrão que utilizamos ao longo do livro para facilitar a diferenciação entre medidas e colunas calculadas no código.

Após definir GrossMarginPct como uma medida, o resultado é correto, conforme mostrado na Figura 2-5.

SalesKey	SalesAmount	TotalCost	GrossMargin	GrossMarginPct
20070104611301-0002	$72.19	$38.74	$33.45	46.34%
20070104611301-0003	$23.75	$11.50	$12.25	51.58%
20070104611320-0006	$216.57	$116.22	$100.35	46.34%
20070104611320-0007	$23.75	$11.50	$12.25	51.58%
20070104611506-0002	$72.19	$38.74	$33.45	46.34%
20070104611506-0003	$23.75	$11.50	$12.25	51.58%
20070104611914-0002	$64.59	$38.74	$25.85	40.02%
20070104611914-0003	$21.25	$11.50	$9.75	45.88%
20070104611952-0004	$64.59	$38.74	$25.85	40.02%
20070104611952-0005	$21.25	$11.50	$9.75	45.88%
20070104611998-0002	$64.59	$38.74	$25.85	40.02%
20070104611998-0003	$63.75	$34.50	$29.25	45.88%
Total	**$732.23**	**$401.92**	**$330.31**	**45.11%**

FIGURE 2-5 *GrossMarginPct* defined as a measure shows the correct grand total.

Medidas (measures) e colunas calculadas (calculated columns) ambas utilizam expressões DAX; a diferença está no contexto de avaliação. Uma medida é avaliada no contexto de um elemento visual ou no contexto de uma consulta DAX. No entanto, uma coluna calculada é calculada no nível da linha da tabela à qual pertence. O contexto do elemento visual (mais tarde, no livro, você aprenderá que este é um contexto de filtro) depende das seleções do usuário no relatório ou no formato da consulta DAX. Portanto, ao usar SUM(Sales[SalesAmount]) em uma medida, queremos dizer a soma de todas as linhas que são agregadas sob uma visualização. No entanto, quando usamos Sales[SalesAmount] em uma coluna calculada, queremos dizer o valor da coluna SalesAmount na linha atual.

Uma medida precisa ser definida em uma tabela. Este é um dos requisitos da linguagem DAX. No entanto, a medida não pertence realmente à tabela. De fato, podemos mover uma medida de uma tabela para outra sem perder sua funcionalidade.

Diferenças entre colunas calculadas e medidas

Embora pareçam semelhantes, há uma grande diferença entre colunas calculadas e medidas. O valor de uma coluna calculada é calculado durante a atualização de dados e utiliza a linha atual como contexto. O resultado não depende da atividade do usuário no relatório. Uma medida opera em agregações de dados definidas pelo contexto atual. Em uma matriz ou em uma tabela dinâmica, por exemplo, as tabelas de origem são filtradas de acordo com as coordenadas das células, e os dados são agregados e calculados usando esses filtros. Em outras palavras, uma medida sempre opera em agregações de dados sob o contexto de avaliação. O contexto de avaliação é explicado mais detalhadamente no Capítulo 4, "Compreendendo os contextos de avaliação".

Escolhendo entre colunas calculadas e medidas

Agora que você viu a diferença entre colunas calculadas e medidas, é útil discutir quando usar uma em vez da outra. Às vezes, ambas são opções viáveis, mas na maioria das situações, os requisitos de cálculo determinam a escolha.

Como desenvolvedor, você deve definir uma coluna calculada sempre que desejar fazer o seguinte:

- Colocar os resultados calculados em um fatiador (slicer) ou ver resultados em linhas ou colunas em uma matriz ou em uma tabela dinâmica (ao contrário da área de Valores), ou usar a coluna calculada como uma condição de filtro em uma consulta DAX.
- Definir uma expressão estritamente vinculada à linha atual. Por exemplo, Preço * Quantidade não pode funcionar em uma média ou em uma soma dessas duas colunas.
- Categorizar texto ou números. Por exemplo, um intervalo de valores para uma medida, um intervalo de idades de clientes, como 0–18, 18–25, e assim por diante. Essas categorias são frequentemente usadas como filtros ou para segmentar valores.

No entanto, é obrigatório definir uma medida sempre que se deseja exibir valores de cálculo que reflitam seleções do usuário, e os valores precisam ser apresentados como agregados em um relatório, por exemplo:

- Calcular a porcentagem de lucro de uma seleção de relatório.
- Calcular proporções de um produto em comparação com todos os produtos, mas manter o filtro tanto por ano quanto por região.

Podemos expressar muitos cálculos tanto com colunas calculadas quanto com medidas, embora precisemos usar expressões DAX diferentes para cada caso. Por exemplo, alguém pode definir a Margem Bruta como uma coluna calculada:

```
Sales[GrossMargin] = Sales[SalesAmount] - Sales[TotalProductCost]
```

No entanto, também pode ser definida como uma medida:

```
GrossMargin = SUM ( Sales[SalesAmount] ) - SUM ( Sales[TotalProductCost] )
```

Sugerimos o uso de uma medida neste caso porque, ao ser avaliada no momento da consulta, ela não consome memória e espaço em disco. Como regra geral, sempre que puder expressar um cálculo de ambas as maneiras, as medidas são a opção preferida. Deve-se limitar o uso de colunas calculadas aos poucos casos em que são estritamente necessárias. Usuários com experiência no Excel geralmente preferem colunas calculadas às medidas, pois as colunas calculadas se assemelham de perto à maneira de realizar cálculos no Excel. No entanto, a melhor maneira de calcular um valor no DAX é por meio de uma medida.

Usando medidas em colunas calculadas

É óbvio que uma medida pode se referir a uma ou mais colunas calculadas. Embora menos intuitivo, o oposto também é verdadeiro. Uma coluna calculada pode se referir a uma medida. Dessa forma, a coluna calculada força o cálculo de uma medida para o contexto definido pela linha atual. Essa operação transforma e consolida o resultado de uma medida em uma coluna, que não será influenciada pelas ações do usuário. Obviamente, apenas certas operações podem produzir resultados significativos, pois uma medida normalmente realiza cálculos que dependem fortemente da seleção feita pelo usuário na visualização. Além disso, sempre que você, como desenvolvedor, usa medidas em uma coluna calculada, você está usando um recurso chamado transição de contexto, que é uma técnica avançada de cálculo no DAX. Antes de usar uma medida em uma coluna calculada, sugerimos fortemente que você

leia e compreenda o Capítulo 4, que explica detalhadamente os contextos de avaliação e as transições de contexto.

Introdução a variáveis

Ao escrever uma expressão DAX, é possível evitar a repetição da mesma expressão e aumentar significativamente a legibilidade do código usando variáveis. Por exemplo, observe a seguinte expressão:

```
VAR TotalSales = SUM ( Sales[SalesAmount] )
VAR TotalCosts = SUM ( Sales[TotalProductCost] )
VAR GrossMargin = TotalSales - TotalCosts
RETURN
 GrossMargin / TotalSales
```

As variáveis são definidas com a palavra-chave VAR. Após definir uma variável, é necessário fornecer uma seção RETURN que define o valor de resultado da expressão. Pode-se definir várias variáveis, e elas são locais à expressão em que são definidas.

Uma variável definida em uma expressão não pode ser usada fora da própria expressão. Não existe uma definição de variável global. Isso significa que não é possível definir variáveis usadas em todo o código DAX do modelo.

As variáveis são calculadas usando a avaliação tardia (lazy evaluation). Isso significa que se uma variável é definida e, por qualquer motivo, não é usada no código, a variável em si nunca será avaliada. Se precisar ser calculada, isso ocorre apenas uma vez. Usos posteriores da variável lerão o valor calculado anteriormente. Portanto, as variáveis também são úteis como uma técnica de otimização ao serem usadas várias vezes em uma expressão complexa.

As variáveis são uma ferramenta importante no DAX. Como você aprenderá no Capítulo 4, as variáveis são extremamente úteis porque usam o contexto de avaliação da definição em vez do contexto onde a variável é usada. No Capítulo 6, "Variáveis", abordaremos totalmente as variáveis e como usá-las. Também utilizaremos variáveis extensivamente ao longo do livro.

Lidando com erros em expressões DAX

Agora que você viu alguns dos conceitos básicos da sintaxe, é hora de aprender a lidar elegantemente com cálculos inválidos. Uma expressão DAX pode conter cálculos inválidos porque os dados aos quais ela faz referência não são válidos para a fórmula. Por exemplo, a fórmula pode conter uma divisão por zero ou referenciar um valor de coluna que não é um número ao ser usado em uma operação aritmética, como a multiplicação.

É bom aprender como esses erros são tratados por padrão e como interceptar essas condições para um tratamento especial.

Antes de discutir como lidar com erros, no entanto, descrevemos os diferentes tipos de erros que podem aparecer durante a avaliação de uma fórmula DAX. Eles são:

- Erros de conversão
- Erros em operações aritméticas
- Valores vazios ou ausentes

Erros de conversão

O primeiro tipo de erro é o erro de conversão. Como mostramos anteriormente neste capítulo, o DAX converte automaticamente valores entre strings e números sempre que o operador exige. Todos esses exemplos são expressões DAX válidas:

```
"10" + 32 = 42
"10" & 32 = "1032"
10 & 32 = "1032"
DATE (2010,3,25) = 3/25/2010
DATE (2010,3,25) + 14 = 4/8/2010
DATE (2010,3,25) & 14 = "3/25/201014"
```

Essas fórmulas são sempre corretas porque operam com valores constantes. No entanto, e quanto à seguinte fórmula se VatCode for uma string?

Sales[VatCode] + 100

Como o primeiro operando dessa soma é uma coluna do tipo de dados Texto, você, como desenvolvedor, deve ter confiança de que o DAX pode converter todos os valores dessa coluna em números. Se o DAX falhar ao converter parte do conteúdo para atender às necessidades do operador, ocorrerá um erro de conversão. Aqui estão algumas situações típicas:

"1 + 1" + 0 = Não é possível converter o valor '1 + 1' do tipo Texto para o tipo Número
DATEVALUE ("25/14/2010") = Incompatibilidade de tipo

Se você quiser evitar esses erros, é importante adicionar lógica de detecção de erros nas expressões DAX para interceptar condições de erro e retornar um resultado que faça sentido. Você pode obter o mesmo resultado interceptando o erro após ele ter ocorrido ou verificando os operandos quanto à situação de erro antecipadamente. No entanto, verificar proativamente a situação de erro é melhor do que permitir que o erro aconteça e depois captura-lo.

Erros em operações aritméticas

A segunda categoria de erros são operações aritméticas, como a divisão por zero ou a raiz quadrada de um número negativo. Esses não são erros relacionados à conversão: o DAX os gera sempre que tentamos chamar uma função ou usar um operador com valores inválidos.

A divisão por zero requer tratamento especial porque seu comportamento não é intuitivo (exceto, talvez, para os matemáticos). Quando se divide um número por zero, o DAX retorna o valor especial Infinito. Nos casos especiais de 0 dividido por 0 ou Infinito dividido por Infinito, o DAX retorna o valor especial NaN (não é um número).

Devido a esse comportamento incomum, ele é resumido na Tabela 2-3.

TABLE 2-3 Special Result Values for Division by Zero

Expression	Result
10 / 0	Infinity
7 / 0	Infinity
0 / 0	NaN
(10 / 0) / (7 / 0)	NaN

É importante observar que Infinidade (Infinity) e NaN (não é um número) não são erros, mas valores especiais no DAX. Na verdade, se você dividir um número por Infinidade, a expressão não gera um erro. Em vez disso, ela retorna 0:

```
9954 / (7 / 0) = 0
```

Além dessa situação especial, o DAX pode retornar erros aritméticos ao chamar uma função com um parâmetro incorreto, como a raiz quadrada de um número negativo:

```
SQRT(-1) = Um argumento da função 'SQRT' tem o tipo de dado errado ou o resultado é muito grande ou muito pequeno
```

Se o DAX detectar erros como este, ele bloqueia qualquer cálculo adicional da expressão e gera um erro. Pode-se usar a função ISERROR para verificar se uma expressão leva a um erro. Mostraremos esse cenário posteriormente neste capítulo.

Lembre-se de que valores especiais como NaN são exibidos na interface do usuário de várias ferramentas, como o Power BI, como valores regulares. No entanto, eles podem ser tratados como erros quando exibidos por outras ferramentas clientes, como uma tabela dinâmica do Excel. Por fim, esses valores especiais são detectados como erros pelas funções de detecção de erros.

Valores vazios ou ausentes

A terceira categoria que examinamos não é uma condição de erro específica, mas sim a presença de valores vazios. Valores vazios podem resultar em resultados inesperados ou erros de cálculo quando combinados com outros elementos em um cálculo.

O DAX trata valores ausentes, valores em branco ou células vazias da mesma maneira, usando o valor BLANK. BLANK não é um valor real, mas sim uma maneira especial de identificar essas condições. Podemos obter o valor BLANK em uma expressão DAX chamando a função BLANK, que é diferente de uma string vazia. Por exemplo, a seguinte expressão sempre retorna um valor em branco, que pode ser exibido como uma string vazia ou como "(vazio)" em diferentes ferramentas clientes:

```
= BLANK()
```

Por si só, esta expressão é inútil, mas a função BLANK em si se torna útil sempre que há a necessidade de retornar um valor vazio. Por exemplo, pode-se querer exibir um resultado vazio em vez de 0. A seguinte expressão calcula o desconto total para uma transação de venda, deixando o valor em branco se o desconto for 0:

```
=IF (
 Sales[DiscountPerc] = 0, -- Verifica se há desconto
 BLANK (), -- Retorna um valor em branco se nenhum desconto estiver presente
 Sales[DiscountPerc] * Sales[Amount] -- Calcula o desconto caso contrário
)
```

O BLANK, por si só, não é um erro; é apenas um valor vazio. Portanto, uma expressão contendo um BLANK pode retornar um valor ou um valor em branco, dependendo do cálculo necessário. Por exemplo, a seguinte expressão retorna BLANK sempre que Sales[Amount] for BLANK:

```
= 10 * Sales[Amount]
```

Em outras palavras, o resultado de um produto aritmético é BLANK sempre que um ou ambos os termos são BLANK. Isso cria um desafio quando é necessário verificar um valor em branco. Devido às conversões implícitas, é impossível distinguir se uma expressão é 0 (ou uma string vazia) ou BLANK usando um operador de igualdade. Na verdade, as seguintes condições lógicas são sempre verdadeiras:

```
BLANK() = 0 -- Sempre retorna VERDADEIRO
BLANK() = "" -- Sempre retorna VERDADEIRO
```

Portanto, se as colunas Sales[DiscountPerc] ou Sales[Clerk] estiverem em branco, as seguintes condições retornarão VERDADEIRO mesmo se o teste for contra 0 e string vazia, respectivamente:

```
Sales[DiscountPerc] = 0 -- Retorna VERDADEIRO se DiscountPerc for BLANK ou 0
Sales[Clerk] = "" -- Retorna VERDADEIRO se Clerk for BLANK ou ""
```

Nesses casos, pode-se usar a função ISBLANK para verificar se um valor está em branco ou não:

```
ISBLANK(Sales[DiscountPerc]) -- Retorna VERDADEIRO apenas se DiscountPerc estiver em branco
ISBLANK(Sales[Clerk]) -- Retorna VERDADEIRO apenas se Clerk estiver em branco
```

A propagação do BLANK em uma expressão DAX ocorre em várias outras operações aritméticas e lógicas, como mostrado nos seguintes exemplos:

```
BLANK() + BLANK() = BLANK()
10 * BLANK() = BLANK()
BLANK() / 3 = BLANK()
BLANK() / BLANK() = BLANK()
```

No entanto, a propagação do BLANK no resultado de uma expressão não ocorre para todas as fórmulas. Algumas operações não propagam o BLANK. Em vez disso, elas retornam um valor dependendo dos outros termos da fórmula. Exemplos disso são adição, subtração, divisão por BLANK e uma operação lógica que inclui um BLANK. As seguintes expressões mostram algumas dessas condições juntamente com seus resultados:

```
BLANK() − 10 = −10
18 + BLANK() = 18
```

```
4 / BLANK() = Infinito
0 / BLANK() = NaN
BLANK() || BLANK() = FALSO
BLANK() && BLANK() = FALSO
(BLANK() = BLANK()) = VERDADEIRO
(BLANK() = VERDADEIRO) = FALSO
(BLANK() = FALSO) = VERDADEIRO
(BLANK() = 0) = VERDADEIRO
(BLANK() = "") = VERDADEIRO
ISBLANK(BLANK()) = VERDADEIRO
FALSO || BLANK() = FALSO
FALSO && BLANK() = FALSO
VERDADEIRO || BLANK() = VERDADEIRO
VERDADEIRO && BLANK() = FALSO
```

Valores vazios no Excel e no SQL

O Excel possui uma maneira diferente de lidar com valores vazios. No Excel, todos os valores vazios são considerados como 0 sempre que são usados em uma soma ou multiplicação, mas podem retornar um erro se fizerem parte de uma divisão ou de uma expressão lógica.

No SQL, os valores nulos (NULL) são propagados em uma expressão de maneira diferente do que acontece com o BLANK no DAX. Como você pode ver nos exemplos anteriores, a presença de um BLANK em uma expressão DAX nem sempre resulta em um resultado BLANK, enquanto a presença de NULL no SQL frequentemente se avalia como NULL para toda a expressão. Essa diferença é relevante sempre que você usa o DirectQuery sobre um banco de dados relacional, pois alguns cálculos são executados no SQL e outros no DAX. As diferentes semânticas de BLANK nas duas engines podem resultar em comportamentos inesperados.

Compreender o comportamento de valores vazios ou ausentes em uma expressão DAX e usar o BLANK para retornar uma célula vazia em um cálculo são habilidades importantes para controlar os resultados de uma expressão DAX. Muitas vezes, pode-se usar o BLANK como resultado ao detectar valores incorretos ou outros erros, como demonstramos na próxima seção.

Interceptando erros

Agora que detalhamos os vários tipos de erros que podem ocorrer, ainda precisamos mostrar as técnicas para interceptar erros e corrigi-los ou, pelo menos, produzir uma mensagem de erro contendo informações significativas. A presença de erros em uma expressão DAX frequentemente depende do valor das colunas usadas na própria expressão. Portanto, pode-se querer controlar a presença dessas condições de erro e retornar uma mensagem de erro. A técnica padrão é verificar se uma expressão retorna um erro e, se sim, substituir o erro por uma mensagem específica ou um valor padrão. Existem algumas funções DAX para essa tarefa.

A primeira delas é a função IFERROR, que é semelhante à função IF, mas, em vez de avaliar uma condição booleana, verifica se uma expressão retorna um erro. Dois usos típicos da função IFERROR são os seguintes:

```
= IFERROR ( Sales[Quantity] * Sales[Price], BLANK() )
= IFERROR ( SQRT ( Test[Omega] ), BLANK() )
```

Na primeira expressão, se Sales[Quantity] ou Sales[Price] for uma string que não pode ser convertida em um número, a expressão retornará um valor vazio. Caso contrário, o produto de Quantity e Price é retornado.
Na segunda expressão, o resultado é uma célula vazia sempre que a coluna Test[Omega] contém um número negativo.

Usar IFERROR dessa maneira corresponde a um padrão mais geral que requer o uso de ISERROR e IF:

```
= IF (
    ISERROR ( Sales[Quantity] * Sales[Price] ),
    BLANK (),
    Sales[Quantity] * Sales[Price]
)
```

```
= IF (
   ISERROR ( SQRT ( Test[Omega] ) ),
   BLANK (),
   SQRT ( Test[Omega] )
)
```

Nesses casos, IFERROR é uma opção melhor. Pode-se usar IFERROR sempre que o resultado for a mesma expressão testada para um erro; não há necessidade de duplicar a expressão em dois lugares, e o código é mais seguro e legível. No entanto, um desenvolvedor deve usar IF quando deseja retornar o resultado de uma expressão diferente.

Além disso, pode-se evitar totalmente a ocorrência do erro testando os parâmetros antes de usá-los. Por exemplo, pode-se verificar se o argumento para SQRT é positivo, retornando BLANK para valores negativos:

```
= IF (
   Test[Omega] >= 0,
   SQRT ( Test[Omega] ),
   BLANK ()
)
```

Considerando que o terceiro argumento de uma instrução IF padrão é BLANK, também é possível escrever a mesma expressão de forma mais concisa:

```
= IF (
   Test[Omega] >= 0,
   SQRT ( Test[Omega] )
)
```

Um cenário frequente é testar contra valores vazios. ISBLANK detecta valores vazios, retornando TRUE se seu argumento for BLANK. Essa capacidade é importante, especialmente quando a falta de um valor não implica que seja 0. O exemplo a seguir calcula o custo do envio para uma transação de venda, usando um custo de envio padrão para o produto se a transação em si não especificar um peso:

```
= IF (
   ISBLANK ( Sales[Weight] ), -- Se o peso estiver ausente
   Sales[DefaultShippingCost], -- então retorne o custo padrão
   Sales[Weight] * Sales[ShippingPrice] -- caso contrário, multiplique o peso pelo preço do envio
)
```

Se simplesmente multiplicarmos o peso do produto pelo preço do envio, obtemos um custo vazio para todas as transações de venda sem dados de peso devido à propagação de BLANK em multiplicações.

Ao usar variáveis, os erros devem ser verificados no momento da definição da variável, em vez de onde as usamos. Na verdade, a primeira fórmula no código a seguir retorna zero, a segunda fórmula sempre gera um erro, e a última produz resultados diferentes dependendo da versão do produto que utiliza o DAX (a versão mais recente também gera um erro):

```
IFERROR ( SQRT ( -1 ), 0 ) -- Isso retorna 0
```

```
VAR WrongValue = SQRT ( -1 ) -- O erro acontece aqui, então o resultado é sempre um erro

RETURN -- Este trecho nunca é executado
   IFERROR ( WrongValue, 0 ) -- Esta linha nunca é executada
```

```
IFERROR (
   -- Resultados diferentes dependendo das versões
   VAR WrongValue = SQRT ( -1 ) -- IFERROR gera um erro nas versões de 2017
   RETURN -- IFERROR retorna 0 nas versões até 2016
      WrongValue, 0
```

```
)
```

O erro ocorre quando WrongValue é avaliado. Assim, a engine nunca executará a função IFERROR no segundo exemplo, enquanto o resultado do terceiro exemplo depende das versões do produto. Se precisar verificar erros, tome precauções extras ao usar variáveis. Evite usar funções de tratamento de erros.

Evite usar funções de tratamento de erros

Embora abordemos otimizações mais adiante no livro, é importante saber que funções de tratamento de erros podem criar problemas de desempenho significativos em seu código. Não é que sejam lentas por si mesmas. O problema é que a engine DAX não pode usar caminhos otimizados em seu código quando ocorrem erros. Na maioria dos casos, verificar operandos para possíveis erros é mais eficiente do que usar a engine de tratamento de erros. Por exemplo, em vez de escrever isto:

```
IFERROR ( SQRT ( Test[Omega] ), BLANK () )
```

É muito melhor escrever isto:

```
IF ( Test[Omega] >= 0, SQRT ( Test[Omega] ), BLANK () )
```

Esta segunda expressão não precisa detectar o erro e é mais rápida do que a expressão anterior. Isso, é claro, é uma regra geral. Para uma explicação detalhada, consulte o Capítulo 19, "Otimizando DAX". Outra razão para evitar IFERROR é que ela não pode interceptar erros que ocorrem em um nível mais profundo da execução. Por exemplo, o código a seguir intercepta qualquer erro que ocorre na conversão da coluna Table[Amount], considerando um valor em branco no caso de Amount não conter um número. Como discutido anteriormente, essa execução é cara porque é avaliada para cada linha na tabela.

```
SUMX (
   Table,
   IFERROR ( VALUE ( Table[Amount] ), BLANK () )
)
```

Esteja ciente de que, devido às otimizações na engine DAX, o código a seguir não intercepta os mesmos erros interceptados pelo exemplo anterior. Se Table[Amount] contiver uma string que não seja um número em apenas uma linha, a expressão inteira gera um erro que não é interceptado pelo IFERROR.

```
IFERROR (
   SUMX (
      Table,
      VALUE ( Table[Amount] )
   ),
   BLANK ()
)
```

ISERROR tem o mesmo comportamento que IFERROR. Certifique-se de usá-los com cuidado e apenas para interceptar erros gerados diretamente pela expressão avaliada dentro de IFERROR/ISERROR e não em cálculos aninhados.

Gerando erros

Às vezes, um erro é simplesmente um erro, e a fórmula não deve retornar um valor padrão em caso de erro. De fato, retornar um valor padrão acabaria produzindo um resultado real que seria incorreto. Por exemplo, uma tabela de configuração que contém dados inconsistentes deve produzir um relatório inválido em vez de números que não são confiáveis, embora possa ser considerado correto. Além disso, em vez de um erro genérico, pode-se querer produzir uma mensagem de erro mais significativa para os usuários. Essa mensagem ajudaria os usuários a identificar onde está o problema.

Considere um cenário que requer o cálculo da raiz quadrada da temperatura absoluta medida em Kelvin, para ajustar aproximadamente a velocidade do som em um cálculo científico complexo. Obviamente, não esperamos que a temperatura seja um número negativo. Se isso acontecer devido a um problema na medição, precisamos gerar um erro e interromper o cálculo.

Nesse caso, este código é perigoso porque esconde o problema:

```
= IFERROR (
   SQRT ( Test[Temperature] ),
   0
)
```

Em vez disso, para proteger os cálculos, deve-se escrever a fórmula assim:

```
= IF (
   Test[Temperature] >= 0,
   SQRT ( Test[Temperature] ),
   ERROR ( "A temperatura não pode ser um número negativo. Cálculo interrompido." )
)
```

Formatação do código DAX

Antes de continuarmos explicando a linguagem DAX, gostaríamos de abordar um aspecto importante do DAX - ou seja, a formatação do código. O DAX é uma linguagem funcional, o que significa que, não importando quão complexo seja, uma expressão DAX é como uma única chamada de função. A complexidade do código se traduz na complexidade das expressões que usamos como parâmetros para a função mais externa.

Por esse motivo, é normal ver expressões que se estendem por mais de 10 linhas ou até mais. Ver uma expressão DAX de 20 linhas é comum, então você se acostumará com isso. No entanto, à medida que as fórmulas começam a crescer em comprimento e complexidade, é extremamente importante formatar o código para torná-lo legível para os humanos.

Não há um padrão "oficial" para formatar o código DAX, mas acreditamos que é importante descrever o padrão que usamos em nosso código. Provavelmente, não é o padrão perfeito, e você pode preferir algo diferente. Não temos problema com isso: encontre seu padrão ideal e use-o. A única coisa que você precisa lembrar é: formate seu código e nunca escreva tudo em uma única linha; caso contrário, você terá problemas mais cedo do que espera.

Para entender por que a formatação é importante, veja uma fórmula que calcula uma operação de inteligência de tempo. Esta fórmula um tanto complexa ainda não é a mais complexa que você escreverá. Veja como a expressão se parece se você não a formatar de alguma forma:

```
IF(CALCULATE(NOT ISEMPTY(Balances), ALLEXCEPT (Balances, BalanceDate)),SUMX (ALL(Balances
[Account]), CALCULATE(SUM (Balances[Balance]),LASTNONBLANK(DATESBETWEEN(BalanceDate[Date],
BLANK(),MAX(BalanceDate[Date])),CALCULATE(COUNTROWS(Balances))))),BLANK())
```

Tentar entender o que esta fórmula calcula em sua forma atual é praticamente impossível. Não há indício de qual é a função mais externa e como o DAX avalia os diferentes parâmetros para criar o fluxo de execução completo. Já vimos muitos exemplos de fórmulas escritas dessa maneira por alunos que, em algum momento, pedem ajuda para entender por que a fórmula retorna resultados incorretos. Adivinhe o que fazemos primeiro? Formatamos a expressão; somente depois começamos a trabalhar nela.

A mesma expressão, devidamente formatada, parece assim:

```
IF (
   CALCULATE (
      NOT ISEMPTY ( Balances ),
      ALLEXCEPT ( Balances, BalanceDate )
   ),
   SUMX (
      ALL ( Balances[Account] ),
      CALCULATE (
         SUM ( Balances[Balance] ),
         LASTNONBLANK (
```

```
            DATESBETWEEN (
               BalanceDate[Date],
               BLANK (),
               MAX ( BalanceDate[Date] )
            ),
            CALCULATE ( COUNTROWS ( Balances ) )
         )
      )
   ),
   BLANK ()
)
```

O código é o mesmo, mas desta vez é muito mais fácil ver os três parâmetros do IF. Mais importante, é mais fácil seguir os blocos que surgem naturalmente da indentação das linhas e como eles compõem o fluxo de execução completo. O código ainda é difícil de ler, mas agora o problema é o DAX, não a formatação inadequada. Uma sintaxe mais detalhada usando variáveis pode ajudá-lo a ler o código, mas mesmo nesse caso, a formatação é importante para proporcionar uma compreensão correta do escopo de cada variável:

```
IF (
   CALCULATE (
      NOT ISEMPTY ( Balances ),
      ALLEXCEPT ( Balances, BalanceDate )
   ),
   SUMX (
      ALL ( Balances[Account] ),
      VAR PreviousDates =
         DATESBETWEEN (
            BalanceDate[Date],
            BLANK (),
            MAX ( BalanceDate[Date] )
         )

         VAR LastDateWithBalance =
               LASTNONBLANK (
                     PreviousDates,
                     CALCULATE (
                           COUNTROWS ( Balances )
                     )
               )
         RETURN
               CALCULATE (
                     SUM ( Balances[Balance] ),
                     LastDateWithBalance
               )
   ),
   BLANK ()
)
```

DAXFormatter.com

Foi criado um site dedicado à formatação de códigos DAX. Desenvolvemos esta ferramenta para uso próprio, pois a formatação de código é uma tarefa demorada, e não queríamos gastar nosso tempo fazendo isso para cada fórmula que escrevemos. Após a ferramenta estar funcionando, decidimos disponibilizá-la para o domínio público, para que os usuários possam formatar seus próprios códigos DAX (aliás, conseguimos promover nossas regras de formatação dessa maneira).

Você pode encontrar o site em www.daxformatter.com. A interface do usuário é simples: basta copiar o código DAX, clicar em FORMATAR, e a página será atualizada exibindo uma versão bem formatada do seu código, que você pode então copiar e colar na janela original.

Aqui estão as regras que usamos para formatar o DAX:

- Sempre separe os nomes das funções, como IF, SUMX e CALCULATE, de qualquer outro termo usando um espaço e sempre escreva-os em maiúsculas.
- Escreva todas as referências de colunas no formato TableName[ColumnName], sem espaço entre o nome da tabela e o colchete de abertura. Sempre inclua o nome da tabela.
- Escreva todas as referências de medidas no formato [MeasureName], sem nenhum nome de tabela.
- Sempre use um espaço após as vírgulas e nunca as preceda com um espaço.
- Se a fórmula couber em uma única linha, não aplique nenhuma outra regra.
- Se a fórmula não couber em uma única linha, então
  - Coloque o nome da função em uma linha por si só, com o parêntese de abertura.
  - Mantenha todos os parâmetros em linhas separadas, com recuo de quatro espaços e com a vírgula no final da expressão, exceto para o último parâmetro.
  - Alinhe o parêntese de fechamento com a chamada da função, para que o parêntese de fechamento fique em sua própria linha.

Essas são as regras básicas que usamos. Uma lista mais detalhada dessas regras está disponível em http://sql.bi/daxrules. Se você encontrar uma maneira de expressar fórmulas que se ajuste melhor ao seu método de leitura, use-a. O objetivo da formatação é tornar a fórmula mais fácil de ler, então use a técnica que funciona melhor para você. O ponto mais importante a lembrar ao definir seu conjunto pessoal de regras de formatação é que você sempre precisa ser capaz de ver os erros o mais rápido possível. Se, no código não formatado mostrado anteriormente, o DAX reclamasse de um parêntese de fechamento ausente, seria difícil identificar onde está o erro. Na fórmula formatada, é muito mais fácil ver como cada parêntese de fechamento corresponde à chamada de função de abertura.

Ajuda na formatação do DAX

A formatação do DAX não é uma tarefa fácil, porque frequentemente escrevemos usando uma fonte pequena em uma caixa de texto. Dependendo da versão, Power BI, Excel e Visual Studio fornecem diferentes editores de texto para o DAX. No entanto, algumas dicas podem ajudar na escrita do código DAX:

Para aumentar o tamanho da fonte, mantenha pressionado Ctrl enquanto gira a roda do mouse, facilitando a visualização do código.
Para adicionar uma nova linha à fórmula, pressione Shift + Enter.
Se editar na caixa de texto não for adequado para você, copie o código para outro editor, como o Notepad ou o DAX Studio, e depois copie e cole a fórmula de volta na caixa de texto.
Quando você olha para uma expressão DAX, à primeira vista, pode ser difícil entender se é uma coluna calculada ou uma medida. Assim, em nossos livros e artigos, usamos um sinal de igual (=) sempre que definimos uma coluna calculada e o operador de atribuição (:=) para definir medidas:

```
CalcCol = SUM ( Sales[SalesAmount] ) -- é uma coluna calculada
Store[CalcCol] = SUM ( Sales[SalesAmount] ) -- é uma coluna calculada na tabela Store
CalcMsr := SUM ( Sales[SalesAmount] ) -- é uma medida
```

Finalmente, ao usar colunas e medidas no código, recomendamos sempre colocar um nome de tabela antes de uma coluna e nunca antes de uma medida, como fazemos em todos os exemplos.

Introduzindo agregadores e iteradores

Quase todo modelo de dados precisa operar em dados agregados. O DAX oferece um conjunto de funções que agregam os valores de uma coluna em uma tabela e retornam um único valor. Chamamos esse grupo de funções de funções de agregação. Por exemplo, a medida a seguir calcula a soma de todos os números na coluna SalesAmount da tabela Sales:

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

A função SUM agrega todas as linhas da tabela se for usada em uma coluna calculada. Sempre que é usada em uma medida, ela considera apenas as linhas que estão sendo filtradas por segmentadores, linhas, colunas e condições de filtro no relatório.

Existem muitas funções de agregação (SUM, AVERAGE, MIN, MAX e STDEV), e seu comportamento muda apenas na forma como agregam valores: SUM adiciona valores, enquanto MIN retorna o valor mínimo. Quase todas essas funções operam apenas em valores numéricos ou em datas. Apenas MIN e MAX podem operar também em valores de texto. Além disso, o DAX nunca considera células vazias ao realizar a agregação, e esse comportamento é diferente de seus equivalentes no Excel (mais sobre isso posteriormente neste capítulo).

NOTA MIN e MAX oferecem outro comportamento: se usados com dois parâmetros, eles retornam o mínimo ou máximo dos dois parâmetros. Assim, MIN (1, 2) retorna 1 e MAX (1, 2) retorna 2. Essa funcionalidade é útil quando é necessário calcular o mínimo ou máximo de expressões complexas, pois evita ter que escrever a mesma expressão várias vezes em instruções IF.

Todas as funções de agregação que descrevemos até agora trabalham em colunas. Portanto, agregam valores de uma única coluna. Algumas funções de agregação podem agregar uma expressão em vez de uma única coluna. Devido à forma como funcionam, são conhecidas como iteradores. Esse conjunto de funções é útil, especialmente quando você precisa fazer cálculos usando colunas de diferentes tabelas relacionadas ou quando precisa reduzir o número de colunas calculadas.

Iteradores sempre aceitam pelo menos dois parâmetros: o primeiro é uma tabela que eles escaneiam; o segundo é tipicamente uma expressão que é avaliada para cada linha da tabela. Após concluírem o escaneamento da tabela e avaliarem a expressão linha por linha, os iteradores agregam os resultados parciais de acordo com sua semântica.

Por exemplo, se calcularmos o número de dias necessários para entregar um pedido em uma coluna calculada chamada DiasParaEntregar e criarmos um relatório com base nisso, obtemos o relatório mostrado na Figura 2-6. Observe que o total geral mostra a soma de todos os dias, o que não é útil para essa métrica:

```
Sales[DaysToDeliver] = INT ( Sales[Delivery Date] - Sales[Order Date] )
```

SalesKey	Order Date	Delivery Date	DaysToDeliver
200701022CS425-0013	01/02/2007	01/08/2007	6
200701022CS425-0014	01/02/2007	01/09/2007	7
200701022CS425-0015	01/02/2007	01/10/2007	8
200701022CS425-0016	01/02/2007	01/11/2007	9
200701022CS425-0017	01/02/2007	01/12/2007	10
200701022CS425-0018	01/02/2007	01/13/2007	11
200701023CS425-0202	01/02/2007	01/08/2007	6
200701023CS425-0203	01/02/2007	01/09/2007	7
200701023CS425-0204	01/02/2007	01/10/2007	8
200701023CS425-0205	01/02/2007	01/11/2007	9
Total			**848075**

FIGURE 2-6 The grand total is shown as a sum, when you might want an average instead.

A soma total geral que podemos realmente usar requer uma medida chamada AvgDelivery, que mostra o tempo de entrega para cada pedido e a média de todas as durações no nível total geral:

```
AvgDelivery := AVERAGE ( Sales[DaysToDeliver] )
```

O resultado desta nova medida é visível no relatório mostrado na Figura 2-7.

SalesKey	Order Date	Delivery Date	DaysToDeliver	AvgDelivery
200701022CS425-0013	01/02/2007	01/08/2007	6	6.00
200701022CS425-0014	01/02/2007	01/09/2007	7	7.00
200701022CS425-0015	01/02/2007	01/10/2007	8	8.00
200701022CS425-0016	01/02/2007	01/11/2007	9	9.00
200701022CS425-0017	01/02/2007	01/12/2007	10	10.00
200701022CS425-0018	01/02/2007	01/13/2007	11	11.00
200701023CS425-0202	01/02/2007	01/08/2007	6	6.00
200701023CS425-0203	01/02/2007	01/09/2007	7	7.00
200701023CS425-0204	01/02/2007	01/10/2007	8	8.00
200701023CS425-0205	01/02/2007	01/11/2007	9	9.00
Total			**848075**	**8.46**

FIGURE 2-7 The measure aggregating by average shows the average delivery days at the grand total level.

A medida calcula o valor médio ao fazer a média de uma coluna calculada. Poderíamos remover a coluna calculada, economizando espaço no modelo, usando um iterador. Embora seja verdade que a função AVERAGE não pode calcular a média de uma expressão, sua contraparte AVERAGEX pode iterar sobre a tabela Sales e calcular os dias de entrega linha por linha, média dos resultados no final. Este código realiza o mesmo resultado que a definição anterior:

```
AvgDelivery :=
AVERAGEX (
 Sales,
 INT ( Sales[Delivery Date] - Sales[Order Date] )
)
```

A maior vantagem desta última expressão é que ela não depende da presença de uma coluna calculada. Portanto, podemos construir todo o relatório sem criar colunas calculadas caras.

A maioria dos iteradores tem o mesmo nome que seu equivalente não iterativo. Por exemplo, SUM tem um correspondente SUMX, e MIN tem um correspondente MINX. No entanto, tenha em mente que alguns iteradores não correspondem a nenhum agregador. Mais adiante neste livro, você aprenderá sobre FILTER, ADDCOLUMNS, GENERATE e outras funções que são iteradores mesmo que não agreguem seus resultados.

Ao aprender DAX pela primeira vez, você pode pensar que os iteradores são inerentemente lentos. O conceito de realizar cálculos linha por linha parece ser uma operação intensiva de CPU. Na verdade, os iteradores são rápidos, e não há penalidade de desempenho ao usar iteradores em vez de agregadores padrão. Os agregadores são apenas uma versão de sintaxe açucarada dos iteradores.

Na verdade, as funções básicas de agregação são uma versão abreviada da função correspondente com sufixo X. Por exemplo, considere a seguinte expressão:

```
SUM ( Sales[Quantity] )
```

Internamente traduzida para esta versão correspondente do mesmo código:

```
SUMX ( Sales, Sales[Quantity] )
```

A única vantagem em usar SUM é uma sintaxe mais curta. No entanto, não há diferenças de desempenho entre SUM e SUMX ao agregar uma única coluna. Elas são, em todos os aspectos, a mesma função.

Abordaremos mais detalhes sobre esse comportamento no Capítulo 4. Lá, introduzimos o conceito de contextos de avaliação para descrever adequadamente como os iteradores funcionam.

Usando funções DAX comuns

Agora que você viu os fundamentos do DAX e como lidar com condições de erro, o que se segue é um breve passeio pelas funções e expressões mais comumente usadas no DAX.

Funções de agregação

Nas seções anteriores, descrevemos os agregadores básicos como SUM, AVERAGE, MIN e MAX. Você aprendeu que SUM e AVERAGE, por exemplo, funcionam apenas em colunas numéricas.

O DAX também oferece uma sintaxe alternativa para funções de agregação herdadas do Excel, que adiciona o sufixo A ao nome da função, apenas para obter o mesmo nome e comportamento que o Excel. No entanto, essas funções são úteis apenas para colunas que contêm valores booleanos, pois TRUE é avaliado como 1 e FALSE como 0. Colunas de texto são sempre consideradas 0. Portanto, não importa o que está no conteúdo de uma coluna, se usar MAXA em uma coluna de texto, o resultado será sempre 0. Além disso, o DAX nunca considera células vazias ao realizar a agregação. Embora essas funções possam ser usadas em colunas não numéricas sem retornar um erro, seus resultados não são úteis porque não há conversão automática para números em colunas de texto. Essas funções são chamadas AVERAGEA, COUNTA, MINA e MAXA.

Sugerimos que você não use essas funções, cujo comportamento será mantido inalterado no futuro devido à compatibilidade com código existente que pode depender do comportamento atual.

Nota: Apesar dos nomes serem idênticos às funções estatísticas, eles são usados de maneira diferente no DAX e no Excel, porque no DAX uma coluna tem um tipo de dados, e seu tipo de dados determina o comportamento das funções de agregação. O Excel lida com um tipo de dados diferente para cada célula, enquanto o DAX lida com um único tipo de dados para a coluna inteira. O DAX lida com dados em forma tabular com tipos bem definidos para cada coluna, enquanto as fórmulas do Excel funcionam em valores de células heterogêneos sem tipos bem definidos. Se uma coluna no Power BI tiver um tipo de dados numérico, todos os valores podem ser apenas números ou células vazias. Se uma coluna for do tipo de texto, ela será sempre 0 para essas funções (exceto COUNTA), mesmo que o texto possa ser convertido em um número, enquanto no Excel o valor é considerado um número em uma base de célula por célula. Por esses motivos, essas funções não são muito úteis para colunas de texto. Apenas MIN e MAX também suportam valores de texto no DAX.

As funções que você aprendeu anteriormente são úteis para realizar a agregação de valores. Às vezes, você pode não estar interessado em agregar valores, mas apenas em contá-los. O DAX oferece um conjunto de funções que são úteis para contar linhas ou valores:

- COUNT opera em qualquer tipo de dado, exceto Booleano.
- COUNTA opera em qualquer tipo de coluna.
- COUNTBLANK retorna o número de células vazias (em branco ou strings vazias) em uma coluna.
- COUNTROWS retorna o número de linhas em uma tabela.
- DISTINCTCOUNT retorna o número de valores distintos de uma coluna, incluindo o valor em branco se presente.
- DISTINCTCOUNTNOBLANK retorna o número de valores distintos de uma coluna, excluindo o valor em branco.

COUNT e COUNTA são funções quase idênticas no DAX. Elas retornam o número de valores da coluna que não estão vazias, independentemente do tipo de dados. Elas são herdadas do Excel, onde COUNTA aceita qualquer tipo de dados, incluindo strings, enquanto COUNT aceita apenas colunas numéricas. Se quisermos contar todos os valores em uma coluna que contêm um valor vazio, podemos usar a função COUNTBLANK. Ambos os valores em branco e valores vazios são considerados valores vazios pelo COUNTBLANK. Por fim, se quisermos contar o número de linhas de uma tabela, podemos usar a função COUNTROWS. Cuidado, pois COUNTROWS requer uma tabela como parâmetro, não uma coluna.

As duas últimas funções, DISTINCTCOUNT e DISTINCTCOUNTNOBLANK, são úteis porque fazem exatamente o que seus nomes sugerem: contam os valores distintos de uma coluna, que é seu único parâmetro. DISTINCTCOUNT conta o valor em branco como um dos valores possíveis, enquanto DISTINCTCOUNTNOBLANK ignora o valor em branco.

Observação: DISTINCTCOUNT é uma função introduzida na versão 2012 do DAX. As versões anteriores do DAX não incluíam o DISTINCTCOUNT; para calcular o número de valores distintos de uma coluna, tínhamos que usar COUNTROWS (DISTINCT (tabela[coluna])). Os dois padrões retornam o mesmo resultado, embora DISTINCTCOUNT seja mais fácil de ler, exigindo apenas uma única chamada de função. DISTINCTCOUNTNOBLANK é uma função introduzida em 2019 e fornece a mesma semântica de uma operação COUNT DISTINCT no SQL sem precisar escrever uma expressão mais longa no DAX.

Funções lógicas

Às vezes, queremos construir uma condição lógica em uma expressão, por exemplo, para implementar cálculos diferentes dependendo do valor de uma coluna ou para interceptar uma condição de erro. Nestes casos, podemos usar uma das funções lógicas no DAX. A seção anterior intitulada "Lidando com erros em expressões DAX" descreveu as duas funções mais importantes deste grupo: IF e IFERROR. Descrevemos a função IF na seção "Declarações condicionais", no início deste capítulo.

As funções lógicas são bastante simples e fazem o que seus nomes sugerem. Elas são AND, FALSE, IF, IFERROR, NOT, TRUE e OR. Por exemplo, se quisermos calcular o valor como quantidade multiplicada pelo preço apenas quando a coluna Price contiver um valor numérico, podemos usar o seguinte padrão:

```
Sales[Amount] = IFERROR ( Sales[Quantity] * Sales[Price], BLANK ( ) )
```

Se não usássemos o IFERROR e se a coluna Price contivesse um número inválido, o resultado para a coluna calculada seria um erro porque se uma única linha gera um erro de cálculo, o erro se propaga para toda a coluna. O uso do IFERROR, no entanto, intercepta o erro e o substitui por um valor em branco.

Outra função interessante nesta categoria é SWITCH, que é útil quando temos uma coluna contendo um número baixo de valores distintos e desejamos obter comportamentos diferentes dependendo de seu valor. Por exemplo, a coluna Size na tabela

Product contém S, M, L, XL, e podemos querer decodificar esse valor em uma coluna mais explícita. Podemos obter o resultado usando chamadas aninhadas de IF:

```
'Product'[SizeDesc] =
IF (
   'Product'[Size] = "S",
   "Small",
   IF (
      'Product'[Size] = "M",
      "Medium",
      IF (
         'Product'[Size] = "L",
         "Large",
         IF ( 'Product'[Size] = "XL", "Extra Large", "Other" )
      )
   )
)
```

Uma maneira mais conveniente de expressar a mesma fórmula, usando SWITCH, é assim:

```
'Product'[SizeDesc] =
SWITCH (
   'Product'[Size],
   "S", "Small",
   "M", "Medium",
   "L", "Large",
   "XL", "Extra Large",
   "Other"
)
```

O código nesta última expressão é mais legível, embora não mais rápido, porque internamente o DAX traduz as instruções SWITCH em um conjunto de funções IF aninhadas.

Nota: o SWITCH é frequentemente usado para verificar o valor de um parâmetro e definir o resultado de uma medida. Por exemplo, alguém pode criar uma tabela de parâmetros contendo YTD, MTD, QTD como três linhas e permitir que o usuário escolha entre os três disponíveis qual agregação usar em uma medida. Este era um cenário comum antes de 2019. Agora não é mais necessário graças à introdução de grupos de cálculo, abordados no Capítulo 9, "Grupos de cálculo". Grupos de cálculo são a maneira preferida de calcular valores que o usuário pode parametrizar.

Dica: Aqui está uma maneira interessante de usar a função SWITCH para verificar várias condições na mesma expressão. Como o SWITCH é convertido em um conjunto de funções IF aninhadas, onde o primeiro que corresponder ganha, você pode testar várias condições usando este padrão:

```
SWITCH (
   TRUE (),
   Product[Size] = "XL"
      && Product[Color] = "Red", "Red and XL",
   Product[Size] = "XL"
      && Product[Color] = "Blue", "Blue and XL",
   Product[Size] = "L"
      && Product[Color] = "Green", "Green and L"
)
```

Usar TRUE como o primeiro parâmetro significa "Retornar o primeiro resultado em que a condição é verdadeira."

Funções de informação

Sempre que houver a necessidade de analisar o tipo de uma expressão, você pode usar uma das funções de informação. Todas essas funções retornam um valor booleano e podem ser usadas em qualquer expressão lógica. Elas são ISBLANK, ISERROR, ISLOGICAL, ISNONTEXT, ISNUMBER e ISTEXT.

É importante observar que quando uma coluna é passada como parâmetro em vez de uma expressão, as funções ISNUMBER, ISTEXT e ISNONTEXT sempre retornam TRUE ou FALSE, dependendo do tipo de dados da coluna e da condição de vazio de cada célula. Isso torna essas funções quase inúteis no DAX; elas foram herdadas do Excel na primeira versão do DAX.

Você pode estar se perguntando se pode usar ISNUMBER com uma coluna de texto apenas para verificar se uma conversão para número é possível. Infelizmente, essa abordagem não é possível. Se você deseja testar se um valor de texto é conversível em número, você deve tentar a conversão e lidar com o erro se ela falhar. Por exemplo, para testar se a coluna Price (que é do tipo string) contém um número válido, é necessário escrever:

```
Sales[IsPriceCorrect] = NOT ISERROR ( VALUE ( Sales[Price] ) )
```

O DAX tenta converter de um valor de string para um número. Se tiver sucesso, retorna TRUE (porque ISERROR retorna FALSE); caso contrário, retorna FALSE (porque ISERROR retorna TRUE). Por exemplo, a conversão falha se algumas das linhas tiverem um valor de string "N/A" para preço.

No entanto, se tentarmos usar ISNUMBER, como na seguinte expressão, sempre recebemos FALSE como resultado:

```
Sales[IsPriceCorrect] = ISNUMBER ( Sales[Price] )
```

Neste caso, ISNUMBER sempre retorna FALSE porque, com base na definição no modelo, a coluna Price não é um número, mas uma string, independentemente do conteúdo de cada linha.

Funções matemáticas

O conjunto de funções matemáticas disponíveis no DAX é semelhante ao conjunto disponível no Excel, com a mesma sintaxe e comportamento. As funções matemáticas de uso comum são ABS, EXP, FACT, LN, LOG, LOG10, MOD, PI, POWER, QUOTIENT, SIGN e SQRT. As funções aleatórias são RAND e RANDBETWEEN. Usando EVEN e ODD, você pode testar números. GCD e LCM são úteis para calcular o maior denominador comum e o menor múltiplo comum de dois números. QUOTIENT retorna a divisão inteira de dois números.

Finalmente, várias funções de arredondamento merecem um exemplo; na verdade, podemos usar várias abordagens para obter o mesmo resultado. Considere estas colunas calculadas, juntamente com seus resultados na Figura 2-8:

FLOOR = FLOOR (Tests[Value], 0.01) -- Arredonda para baixo o valor de Tests[Value] para o múltiplo mais próximo de 0.01.

TRUNC = TRUNC (Tests[Value], 2) -- Trunca Tests[Value] para o número especificado de casas decimais (neste caso, 2).

ROUNDDOWN = ROUNDDOWN (Tests[Value], 2) -- Arredonda para baixo Tests[Value] para o número especificado de casas decimais (neste caso, 2)

MROUND = MROUND (Tests[Value], 0.01) -- Arredonda Tests[Value] para o múltiplo mais próximo de 0.01.

ROUND = ROUND (Tests[Value], 2) -- Arredonda Tests[Value] para o número especificado de casas decimais (neste caso, 2). Utiliza o arredondamento padrão.

CEILING = CEILING (Tests[Value], 0.01) -- Arredonda para cima o valor de Tests[Value] para o múltiplo mais próximo de 0.01.

ISO.CEILING = ISO.CEILING (Tests[Value], 0.01) -- Arredonda para cima o valor de Tests[Value] para o múltiplo mais próximo de 0.01, seguindo as regras do padrão ISO.

ROUNDUP = ROUNDUP (Tests[Value], 2) -- Arredonda para cima Tests[Value] para o número especificado de casas decimais (neste caso, 2).

INT = INT (Tests[Value]) -- Retorna a parte inteira de Tests[Value], removendo a parte decimal.

FIXED = FIXED (Tests[Value], 2, TRUE) -- Formata Tests[Value] como texto com 2 casas decimais, incluindo zeros à direita se necessário.

Estas funções realizam diferentes tipos de arredondamentos ou truncamentos nos valores da coluna "Tests[Value]" com diferentes precisões especificadas.

Test	Value	FLOOR	TRUNC	ROUNDDOWN	MROUND	ROUND	CEILING	ISO.CEILING	ROUNDUP	INT	FIXED
A	1.123450	1.12	1.12	1.12	1.12	1.12	1.13	1.13	1.13	1	1.12
B	1.265000	1.26	1.26	1.26	1.26	1.27	1.27	1.27	1.27	1	1.27
C	1.265001	1.26	1.26	1.26	1.27	1.27	1.27	1.27	1.27	1	1.27
D	1.499999	1.49	1.49	1.49	1.50	1.50	1.50	1.50	1.50	1	1.50
E	1.511110	1.51	1.51	1.51	1.51	1.51	1.52	1.52	1.52	1	1.51
F	1.000001	1.00	1.00	1.00	1.00	1.00	1.01	1.01	1.01	1	1.00
G	1.999999	1.99	1.99	1.99	2.00	2.00	2.00	2.00	2.00	1	2.00

FIGURE 2-8 This summary shows the results of using different rounding functions.

FLOOR, TRUNC e ROUNDDOWN são semelhantes, exceto na forma como podemos especificar o número de dígitos para arredondar. Na direção oposta, CEILING e ROUNDUP são semelhantes em seus resultados. Você pode ver algumas diferenças na forma como o arredondamento é feito entre a função MROUND e a função ROUND.

Funções trigonométricas

O DAX oferece um conjunto rico de funções trigonométricas que são úteis para certos cálculos: COS, COSH, COT, COTH, SIN, SINH, TAN e TANH. Prefixando-os com A, calcula a versão arc (arcoseno, arcocosseno, etc.). Não vamos entrar em detalhes sobre essas funções porque o uso delas é direto.

DEGREES e RADIANS realizam conversões para graus e radianos, respectivamente, e SQRTPI calcula a raiz quadrada de seu parâmetro após multiplicá-lo por pi.

Funções de texto

A maioria das funções de texto disponíveis no DAX são semelhantes às disponíveis no Excel, com apenas algumas exceções. As funções de texto são CONCATENATE, CONCATENATEX, EXACT, FIND, FIXED, FORMAT, LEFT, LEN, LOWER, MID, REPLACE, REPT, RIGHT, SEARCH, SUBSTITUTE, TRIM, UPPER e VALUE. Essas funções são úteis para manipular texto e extrair dados de strings que contêm múltiplos valores.

Por exemplo, a Figura 2-9 mostra um exemplo da extração de nomes e sobrenomes de uma string que contém esses valores separados por vírgulas, com o título no meio que queremos remover.

Name	Comma1	Comma2	FirstLastName	SimpleConversion
Ferrari, Alberto	8		Alberto Ferrari	Ferrari, Alberto Ferrari
Ferrari, Mr., Alberto	8	13	Alberto Ferrari	Alberto Ferrari
Russo, Mr., Marco	6	11	Marco Russo	Marco Russo

FIGURE 2-9 This example shows first and last names extracted using text functions.

Para obter esse resultado, começamos calculando a posição das duas vírgulas. Em seguida, usamos esses números para extrair a parte certa do texto. A coluna SimpleConversion implementa uma fórmula que pode retornar valores imprecisos se houver menos de duas vírgulas na string e gera um erro se não houver vírgulas. A coluna FirstLastName implementa uma expressão mais complexa que não falha em caso de vírgulas ausentes:

```
People[Comma1] =
IFERROR ( FIND ( ",", People[Name] ), BLANK () )
People[Comma2] =
IFERROR ( FIND ( " ,", People[Name], People[Comma1] + 1 ), BLANK () )
People[SimpleConversion] =
MID ( People[Name], People[Comma2] + 1, LEN ( People[Name] ) ) & " "
   & LEFT ( People[Name], People[Comma1] - 1 )
People[FirstLastName] =
TRIM (
   MID (
      People[Name],
```

```
    IF ( ISNUMBER ( People[Comma2] ), People[Comma2], People[Comma1] ) + 1,
    LEN ( People[Name] )
  )
)
  & IF (
    ISNUMBER ( People[Comma1] ),
    " "
      & LEFT ( People[Name], People[Comma1] - 1 ),
    ""
  )
```

Como você pode ver, a coluna FirstLastName é definida por uma expressão DAX longa, mas é necessário usá-la para evitar possíveis erros que se propagariam para toda a coluna se até mesmo um único valor gerar um erro.

Funções de conversão

Você aprendeu anteriormente que o DAX realiza conversões automáticas de tipos de dados para ajustá-los às necessidades do operador. Embora a conversão ocorra automaticamente, ainda existe um conjunto de funções que podem realizar conversões explícitas de tipo de dados.

CURRENCY pode transformar uma expressão em um tipo de moeda, enquanto INT transforma uma expressão em um número inteiro. DATE e TIME recebem as partes de data e hora como parâmetros e retornam um DateTime correto. VALUE transforma uma string em um formato numérico, enquanto FORMAT obtém um valor numérico como seu primeiro parâmetro e um formato de string como seu segundo parâmetro, podendo transformar valores numéricos em strings. FORMAT é comumente usado com DateTime. Por exemplo, a seguinte expressão retorna "2019 Jan 12":

```
= FORMAT(DATE(2019, 01, 12), "yyyy mmm dd")
```

A operação oposta, ou seja, a conversão de strings em valores DateTime, é realizada usando a função DATEVALUE.

DATEVALUE com datas em formatos diferentes

DATEVALUE exibe um comportamento especial em relação a datas em formatos diferentes. No padrão europeu, as datas são escritas no formato "dd/mm/aa", enquanto os americanos preferem usar "mm/dd/aa". Por exemplo, 28 de fevereiro tem representações de string diferentes nas duas culturas. Se você fornecer ao DATEVALUE uma data que não pode ser convertida usando a configuração regional padrão, em vez de gerar imediatamente um erro, ele tenta uma segunda conversão trocando meses e dias. DATEVALUE também suporta o formato inequívoco "aaaa-mm-dd". Como exemplo, as três expressões a seguir avaliam para 28 de fevereiro, independentemente das configurações regionais que você tem:

DATEVALUE("28/02/2018") -- Isso é 28 de fevereiro no formato europeu
DATEVALUE("02/28/2018") -- Isso é 28 de fevereiro no formato americano
DATEVALUE("2018-02-28") -- Isso é 28 de fevereiro (o formato não é ambíguo)
Às vezes, o DATEVALUE não gera erros quando você esperaria que eles ocorressem. No entanto, esse é o comportamento da função por design.

Funções de data e hora

Em quase todos os tipos de análise de dados, lidar com datas e horas é uma parte importante do trabalho. Muitas funções DAX operam em datas e horas. Algumas delas correspondem a funções semelhantes no Excel e fazem transformações simples para e a partir de um tipo de dados DateTime. As funções de data e hora são DATE, DATEVALUE, DAY, EDATE, EOMONTH, HOUR, MINUTE, MONTH, NOW, SECOND, TIME, TIMEVALUE, TODAY, WEEKDAY, WEEKNUM, YEAR e YEARFRAC.

Essas funções são úteis para calcular valores com base em datas, mas não são usadas para realizar cálculos típicos de inteligência temporal, como comparar valores agregados de um ano para outro ou calcular o valor acumulado do ano de uma medida. Para realizar cálculos de inteligência temporal, você usa outro conjunto de funções chamado funções de inteligência temporal, que descrevemos no Capítulo 8, "Cálculos de inteligência temporal".

Como mencionamos anteriormente neste capítulo, um tipo de dados DateTime internamente usa um número de ponto flutuante em que a parte inteira corresponde ao número de dias após 30 de dezembro de 1899, e a parte decimal indica a fração do dia no tempo. Horas, minutos e segundos são convertidos em frações decimais do dia. Assim, adicionar um número inteiro a um valor DateTime incrementa o valor em um número correspondente de dias. No entanto, você provavelmente achará mais conveniente usar as funções de conversão para extrair o dia, mês e ano de uma data. As expressões a seguir, usadas na Figura 2-10, mostram como extrair essas informações de uma tabela contendo uma lista de datas:

```
'Date'[Day] = DAY(Calendar[Date])
'Date'[Month] = FORMAT(Calendar[Date], "mmmm")
'Date'[MonthNumber] = MONTH(Calendar[Date])
'Date'[Year] = YEAR(Calendar[Date])
```

Date	Day	Month	Year
1/1/2010	1	January	2010
1/2/2010	2	January	2010
1/3/2010	3	January	2010
1/4/2010	4	January	2010
1/5/2010	5	January	2010
1/6/2010	6	January	2010
1/7/2010	7	January	2010
1/8/2010	8	January	2010
1/9/2010	9	January	2010

FIGURE 2-10 This example shows how to extract date information using date and time functions.

Funções relacionais

Duas funções úteis que você pode usar para navegar por relacionamentos dentro de uma fórmula DAX são RELATED e RELATEDTABLE.

Você já sabe que uma coluna calculada pode fazer referência a valores de colunas da tabela na qual ela está definida. Assim, uma coluna calculada definida em Vendas pode fazer referência a qualquer coluna de Vendas. No entanto, e se for necessário fazer referência a uma coluna em outra tabela? Em geral, não é possível usar colunas em outras tabelas a menos que um relacionamento seja definido no modelo entre as duas tabelas. Se as duas tabelas compartilharem um relacionamento, você pode usar a função RELATED para acessar colunas na tabela relacionada.

Por exemplo, pode-se querer calcular uma coluna calculada na tabela Vendas que verifica se o produto vendido está na categoria "Celulares" e, nesse caso, aplicar um fator de redução ao custo padrão. Para calcular tal coluna, é necessário usar uma condição que verifique o valor da categoria de produto, que não está na tabela de Vendas. No entanto, uma cadeia de relacionamentos parte de Vendas, alcançando a Categoria de Produto por meio de Produto e Subcategoria de Produto, como mostrado na Figura 2-11.

FIGURE 2-11 Sales has a chained relationship with *Product Category*.

Não importa quantas etapas são necessárias para viajar da tabela original para a tabela relacionada, o DAX segue a cadeia completa de relacionamentos e retorna o valor da coluna relacionada. Assim, a fórmula para a coluna AdjustedCost pode parecer assim:

```
Sales[AdjustedCost] =
IF (
   RELATED ( 'Product Category'[Category] ) = "Cell Phone",
   Sales[Unit Cost] * 0.95,
```

```
    Sales[Unit Cost]
)
```

Em um relacionamento um-para-muitos, a função RELATED pode acessar o lado "um" do lado "muitos" porque, nesse caso, apenas uma linha na tabela relacionada existe, se houver alguma. Se nenhuma linha existir, o RELACIONADO retorna BLANK.

Se uma expressão estiver no lado "um" do relacionamento e precisar acessar o lado "muitos", RELATED não será útil porque muitas linhas do outro lado podem estar disponíveis para uma única linha. Nesse caso, podemos usar o RELATEDTABLE. O RELATEDTABLE retorna uma tabela contendo todas as linhas relacionadas à linha atual. Por exemplo, se quisermos saber quantos produtos existem em cada categoria, podemos criar uma coluna em 'Product Category' com esta fórmula:

```
'Product Category'[NumOfProducts] = COUNTROWS ( RELATEDTABLE ( Product ) )
```

Para cada categoria de produto, esta coluna calculada mostra o número de produtos relacionados, como mostrado na Figura 2-12.

Category	NumOfProducts
Audio	115
Cameras and camcorders	372
Cell phones	285
Computers	606
Games and Toys	166
Home Appliances	661
Music, Movies and Audio Books	90
TV and Video	222

FIGURE 2-12 You can count the number of products by using *RELATEDTABLE*.

Assim como é o caso para RELATED, RELATEDTABLE pode seguir uma cadeia de relacionamentos sempre começando pelo lado "um" e indo em direção ao lado "muitos". RELATEDTABLE é frequentemente usadA em conjunto com iteradores. Por exemplo, se quisermos calcular a soma da quantidade multiplicada pelo preço líquido para cada categoria, podemos escrever uma nova coluna calculada da seguinte forma:

```
'Product Category'[CategorySales] =
SUMX (
 RELATEDTABLE ( Sales ),
 Sales[Quantity] * Sales[Net Price]
)
```

O resultado desta coluna calculada é mostrado na Figura 2-13.

Category	CategorySales
Audio	$384,518.16
Cameras and camcorders	$7,192,581.95
Cell phones	$1,604,610.26
Computers	$6,741,548.73
Games and Toys	$360,652.81
Home Appliances	$9,600,457.04
Music, Movies and Audio Books	$314,206.74
TV and Video	$4,392,768.29

FIGURE 2-13 Using *RELATEDTABLE* and iterators, we can compute the amount of sales per category.

Devido à coluna ser calculada, este resultado é consolidado na tabela, e não muda de acordo com a seleção do usuário no relatório, como aconteceria se fosse escrito como uma medida.

Conclusão

Neste capítulo, você aprendeu muitas novas funções e começou a examinar alguns códigos DAX. Pode ser que você não se lembre de todas as funções imediatamente, mas quanto mais você as usar, mais familiar se tornarão. Os tópicos mais cruciais que você aprendeu neste capítulo são:

Colunas calculadas são colunas em uma tabela que são calculadas com uma expressão DAX. Colunas calculadas são calculadas no momento da atualização dos dados e não alteram seu valor dependendo da seleção do usuário.

Medidas são cálculos expressos em DAX. Em vez de serem calculadas no momento da atualização, como as colunas calculadas, as medidas são calculadas no momento da consulta. Consequentemente, o valor de uma medida depende da seleção do usuário no relatório.

Erros podem ocorrer a qualquer momento em uma expressão DAX; é preferível detectar a condição de erro antecipadamente em vez de deixar o erro acontecer e interceptá-lo depois.

Agregadores como SUM são úteis para agregar colunas, enquanto para agregar expressões, você precisa usar iteradores. Iteradores funcionam digitalizando uma tabela e avaliando uma expressão linha por linha. No final da iteração, os iteradores agregam um resultado de acordo com sua semântica.

No próximo capítulo, você continuará sua jornada de aprendizado estudando as funções de tabela mais importantes disponíveis no DAX.