DAX i Power BI w analizie danych

Tworzenie zaawansowanych i efektywnych analiz dla biznesu

Michiel Rozema
Henk Vlootman

Helion
Packt

Tytuł oryginału: Extreme DAX: Take your Power BI and Microsoft data analytics skills to the next level

Tłumaczenie: Anna Mizerska

ISBN: 978-83-283-9659-3

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/daxpow
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 4

Kontekst i filtrowanie

Najważniejszą koncepcją, którą należy rozumieć podczas pisania obliczeń DAX, jest **kontekst**. Kontekst jest tym, co odróżnia DAX, dynamiczny język analizy, od funkcji Excela lub zapytań SQL czy skryptów Power Query. Choć wszystkie one tylko zwracają różne wyniki, gdy dane się zmieniają (z kilkoma wyjątkami, wtedy gdy używamy parametrów), pojedyncza formuła DAX może dostarczyć wielu różnych wyników w zależności od tego, gdzie i jak ich użyjesz, czyli od kontekstu.

Kontekst DAX jest również kluczem do osiągniecia zaawansowanych wyników. Gdy już przestaniesz popełniać typowe błędy początkujących, między innymi będziesz już wiedział, której funkcji należy użyć, nie będziesz popełniał błędów składniowych lub zapominał nawiasów, coraz częściej będziesz miał problemy z kontekstem podczas pracy z DAX. Jesteśmy w stanie nawet zaryzykować stwierdzenie:

> *Każdy problem w DAX wywodzi się z kontekstu i każde rozwiązanie może być znalezione przez dokładne zbadanie kontekstu.*

To stwierdzenie jest rzadko negowane!

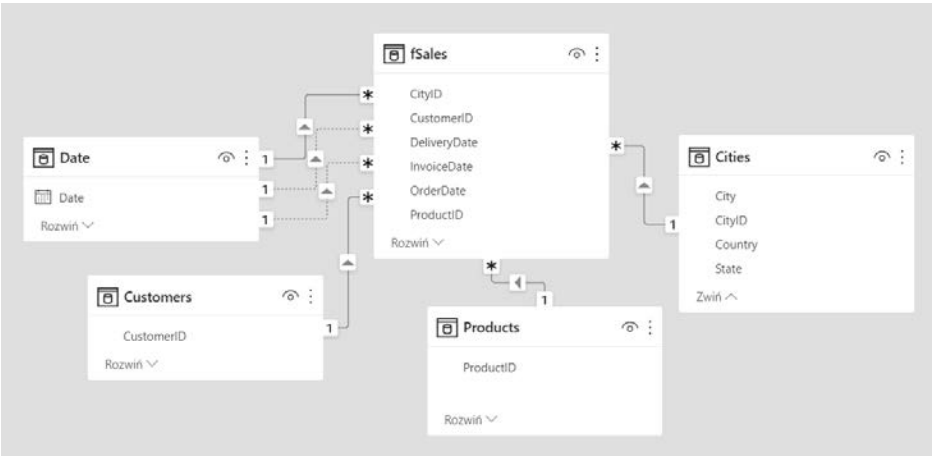
W tym rozdziale omówimy podstawowe tematy związane z kontekstem, które są niezbędne do tego, by zrozumieć treść części II niniejszej książki. Ten rozdział omawia następujące zagadnienia:

- Wprowadzenie do kontekstu DAX.
- Filtrowanie DAX — zastosowanie funkcji `CALCULATE`.
- Analiza czasowa.
- Zmiana działania relacji.
- Funkcje tablicowe w DAX.
- Filtrowanie za pomocą funkcji tablicowych.
- Zmienne w języku DAX.

W tym rozdziale nie znajdzie się wiele przykładów, gdyż w rozdziałach części II przedstawiono szczegółowo wszystkie aspekty kontekstu DAX.

Model Power BI

Przykłady w tym rozdziale zostały wzięte z prostego modelu Power BI. Model ten składa się z jednej tabeli faktów, `fSales`, i kilku tabel filtrów (rysunek 4.1).



Rysunek 4.1. Przykładowy model Power BI

> Plik *Kontekst i filtrowanie.pbix* z przykładami do tego rozdziału znajdziesz pod adresem *https://ftp.helion.pl/przyklady/daxpow.zip*.

Wprowadzenie do kontekstu DAX

Ogólnym określeniem kontekstu DAX jest **kontekst obliczeniowy**, czyli okoliczności, w jakich wykonywana jest formuła DAX dająca określone wyniki. Chcemy wyróżnić trzy typy kontekstu:

- kontekst wiersza,
- kontekst zapytania,
- kontekst filtra.

W większości dokumentacji i publikacji dotyczących Power BI wymieniane są tylko dwa rodzaje kontekstu: wiersza i filtra. Określenie *kontekst zapytania* było używane w związku z dodatkiem Power Pivot w programie Excel długo przed powstaniem Power BI (tak, mamy

już tyle lat) i wciąż go używamy. Z naszego doświadczenia, jako osób szkolących z DAX, wynika, że rozróżnienie między kontekstem zapytania i filtra pomaga zrozumieć trudniejsze scenariusze.

Przyjrzyjmy się bliżej każdemu typowi kontekstu.

Kontekst wiersza

Z kontekstem wiersza masz do czynienia podczas tworzenia kolumn obliczeniowych. Formuła DAX definiująca kolumnę obliczeniową jest wykonywana dla każdego wiersza w tabeli. Wynik obliczenia jest zwykle szczególny dla każdego wiersza. Wynika to z faktu, że wartości w innych kolumnach z tej samej tabeli mogą być częścią tych obliczeń, a te wartości mogą być różne w poszczególnych wierszach. Na przykład kolumna obliczeniowa w tabeli `fSales`, która oblicza marżę jako różnicę między kwotą sprzedaży a kosztami, wyglądałaby tak:

```
Margin = fSales[SalesAmount] - fSales[Costs]
```

Od razu możesz zauważyć, że ta formuła jest używana dla kolumny obliczeniowej przez bezpośrednie odniesienie się do kolumn. Tak można zrobić jedynie w kontekście wiersza i to odróżnia ten typ kontekstu od pozostałych. (W prostych formułach kolumny obliczeniowej przedrostek `fSales` jest często pomijany).

Pamiętaj jednak, że bezpośrednie odwołanie działa tylko dla wartości kolumn z bieżącego wiersza (czyli wiersza, dla którego w danym momencie wykonywane jest obliczenie). Do uzyskania wartości z innych wierszy będziesz musiał podejść zupełnie inaczej. To podstawowa różnica w porównaniu z obliczeniami w Excelu, w którym dość często pobiera się wartość z wiersza powyżej. Łatwo to zrozumieć, gdy zdasz sobie sprawę, że w modelu Power BI nie ma ścisłej kolejności wierszy.

Jest tylko kilka funkcji DAX stworzonych specjalnie z myślą o kontekście wiersza. Jeśli tabela zawierająca kolumnę obliczeniową jest powiązana z inną tabelą, w każdym wierszu możesz uzyskać odpowiadającą wartość z innej tabeli za pomocą funkcji `RELATED`. Pokazana niżej formuła wykorzystuje relację między kolumnami `fSales[OrderDate]` a `Date[Date]`, by uzyskać rok dla każdego wiersza.

```
Year = RELATED('Date'[Year])
```

Wynikiem tego wyrażenia jest kolumna `Year` w tabeli `fSales` (rysunek 4.2).

Jest tylko jeden warunek, by funkcja `RELATED` działała — relacja musi mieć unikatowe wartości w drugiej tabeli, w tym przypadku tabeli `Date`. Koniec końców formuła musi zwrócić jedną wartość.

Jeśli kardynalność relacji jest odwrócona, możesz użyć funkcji `RELATEDTABLE`. Na przykład jeśli chcesz dodać kolumnę obliczeniową do tabeli `Date` z liczbą transakcji sprzedaży każdego dnia, to sprawdzi się następująca formuła:

```
Number of Transactions = COUNTROWS(RELATEDTABLE(fSales))
```

OrderDate	ProductID	UnitAmount	SalesAmount	Year
21.12.2020	215	25	222,50	2020
21.12.2020	215	38	1 623,38	2020
21.12.2020	215	14	149,52	2020
21.12.2020	215	12	576,73	2020
21.12.2020	215	2	42,72	2020
21.12.2020	215	8	85,44	2020
21.12.2020	215	11	411,18	2020
21.12.2020	215	15	320,40	2020
21.12.2020	215	50	801,01	2020

Rysunek 4.2. Dodawanie kolumny obliczeniowej Year (niektóre kolumny zostały usunięte ze względu na czytelność tabeli)

Wynikiem powyżej przytoczonego przykładu użycia funkcji `RELATEDTABLE` dla każdego wiersza w tabeli `Date` jest zestaw wierszy w tabeli `fSales`, które są powiązane z danym wierszem. Jako że to jest tabela, nie może być bezpośrednio użyta jako wartości w kolumnie obliczeniowej, w tym przypadku użyliśmy funkcji `COUNTROWS`, by po prostu policzyć liczbę wierszy w tej tabeli.

Choć funkcja `RELATEDTABLE` jest przeznaczona do użycia z kontekstem wiersza, jest zasadniczo różna od funkcji `RELATED` pod tym względem, że w tle używa innego typu kontekstu.

> Jak już zostało powiedziane w rozdziale 3. „Zastosowanie języka DAX", odradzamy korzystania z kolumn obliczeniowych. To jednak nie oznacza, że nie będziesz musiał się zajmować kontekstem wiersza. Kontekst wiersza odgrywa ważną rolę w funkcjach tablicowych w języku DAX. Więcej o tym dowiesz się później w tym rozdziale.

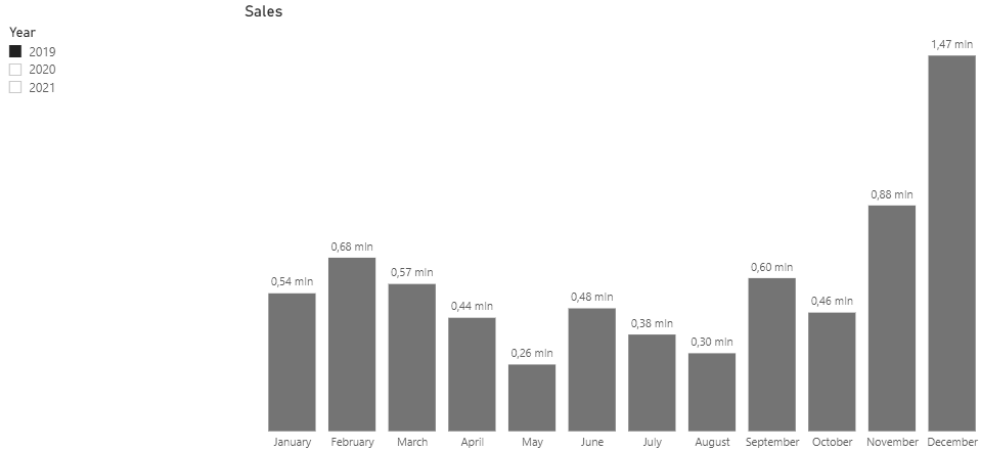
Teraz przejdziemy do pozostałych typów kontekstów. Z kontekstem wiersza łączą się pewne osobliwości, które można zrozumieć po omówieniu kontekstu zapytania i filtra.

Kontekst zapytania

Z kontekstem zapytania pracujesz wtedy, gdy używasz miar DAX. Podobnie do kontekstu wiersza kontekst zapytania jest tym, co sprawia, że zapytanie DAX zwraca określony wynik. Różnica oczywiście jest taka, że tutaj nie pracujemy tylko na jednej tabeli. Krótko mówiąc, kontekst zapytania odnosi się do zestawu wierszy w modelu Power BI, które są wybierane i używane do obliczeń DAX. To pomaga nam rozróżnić dwa osobne, choć blisko ze sobą powiązane, elementy w kontekście zapytania.

- **Wybór** odnosi się do zestawu wierszy w każdej tabeli w modelu, wybranych w określonym kontekście.
- **Filtry** powodują wybór wierszy.

W kontekście zapytania filtry pochodzą od elementów w raporcie (Power BI). Mamy różnego rodzaju filtry: fragmentatory (ang. *slicer*), filtry w panelu *Filtry*, etykiety w wizualizacji lub wybrane elementy w innej wizualizacji. Każdy z tych filtrów tworzy określoną regułę dla kolumny, na przykład na rysunku 4.3 fragmentator zakłada filtr na kolumnie `Year`: *Rok jest równy 2019*. Możemy mieć wiele filtrów na różnych kolumnach, a nawet na jednej kolumnie możemy mieć wiele filtrów. Razem filtry wyznaczają, które wiersze mają być wybrane w każdej tabeli, czyli wszystkie wiersze spełniające zasadę filtra.

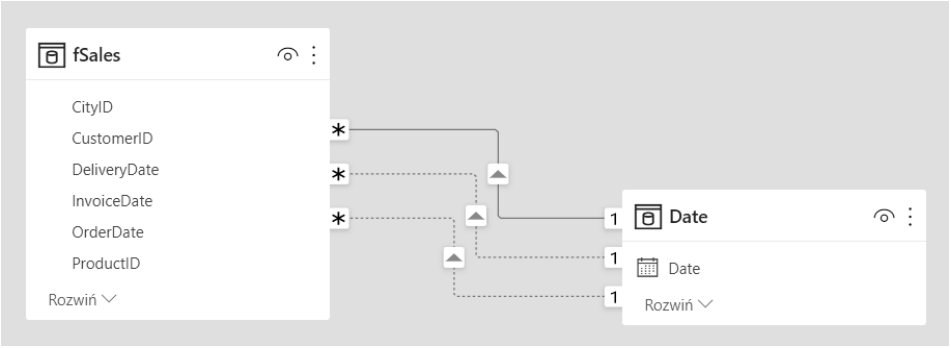


Rysunek 4.3. Prosty raport Power BI

> Filtry wizualizacji zostaną omówione szerzej w rozdziale 8. „Praca z AutoExist".

W kontekście zapytania relacje między tabelami odgrywają ważną rolę — **propagacji filtra**. To oznacza, że filtr na kolumnie w jednej tabeli jest przenoszony do drugiej tabeli przez relację, zgodnie z kierunkiem filtra krzyżowego (rysunek 4.4).

Dokładniej mówiąc, to nie filtr sam w sobie jest propagowany, ale raczej jego efekty. W powiązanej tabeli są wybierane tylko wiersze powiązane z wierszami spełniającymi regułę filtra. To oczywiście jest dokładnie takie działanie, jakiego potrzebujemy — gdy we fragmentatorze zostanie wybrany rok 2019, tak jak na rysunku 4.3, chcemy zobaczyć wyniki właśnie dla tego roku, co oznacza, że wszystkie obliczenia powinny być wykonywane na wierszach, których daty odnoszą się do roku 2019.



Rysunek 4.4. Propagacja filtra przez relację odbywa się zgodnie z kierunkiem strzałki

Z powodu natury kontekstu zapytania nie możesz używać kolumn bezpośrednio w formule, tak jak w przypadku kontekstu wiersza. Poniżej pokazana formuła, jako miara, nie zostanie przyjęta przez edytor.

```
Report Year = 'Date'[Year]
```

W rezultacie otrzymamy komunikat błędu: *Nie można określić pojedynczej wartości kolumny „Year" w tabeli „Date"*. Wynika to z faktu, że z założenia wybór może zawierać wiele wartości. To prawda, nawet gdy kolumna zawiera tylko jedną unikatową wartość lub gdy tabela ma tylko jeden wiersz.

Kontekst filtra

Kontekst filtra przypomina kontekst zapytania, z jednym istotnym wyjątkiem — kontekst filtra to kontekst zmieniany przez kod DAX, na przykład przez dodawanie lub zmianę filtra w kontekście zapytania. W tym celu używana jest funkcja DAX, `CALCULATE` (lub jej pochodna `CALCULATETABLE`). Później w tym rozdziale, w podrozdziale „Filtrowanie DAX — zastosowanie funkcji CALCULATE", omówimy te funkcje szerzej.

Trudność kontekstu filtra tkwi w braku możliwości określenia filtrów w kontekście wizualizacji w taki sposób, jak możesz to zrobić dla kontekstu zapytania. Praca z kontekstem filtra wymaga pewnej umiejętności myślenia abstrakcyjnego i skrupulatnego pilnowania, które filtry są aktywne w danej sytuacji. Z tego samego powodu miary, które tworzą kontekst filtra i używają go, mogą być mylące dla użytkowników raportu i dlatego powinny być stosowane z dużą dbałością, na przykład przez nadawanie znaczących nazw miarom.

Zmiana kontekstu daje mnóstwo możliwości, nieodstępnych w kontekście wiersza i kontekście zapytania. Pozwala nam uzyskać rezultaty niezgodne z kontekstem zapytania i takie, które mogą być użyte, by przeprowadzić zawansowane analizy, na przykład porównanie sprzedaży produktu ze sprzedażą wszystkich produktów, porównanie sprzedaży z tego roku z poprzednim rokiem, prognozowanie trendów. Tak naprawdę żadne scenariusze z części II tej książki nie są możliwe do zrealizowania bez kontekstu filtra.

Ogólne podejście do tworzenia zaawansowanych analiz z użyciem DAX może być ujęte w następujących krokach:

1. Przeanalizuj (możliwe) konteksty zapytania, w których będzie wykonywane Twoje obliczenie.
2. Określ, jaki kontekst filtra jest potrzebny, by zostały zwrócone wymagane wyniki.
3. Ustal, jak przejść z kontekstu zapytania do kontekstu filtra.

Aby opanować język DAX, powinieneś przyjąć taki sposób myślenia, który jest zupełnie inny od pobierania danych za pomocą SQL, programowania czy wykonywania obliczeń w programie Excel.

Wykrywanie filtrów

Filtry w kontekście obliczeń powodują wybór wierszy w tabelach modelu. To może przynieść bardzo wiele różnych efektów dla pojedynczej kolumny. Może stać się tak, że nie zostanie dokonany żaden wybór, gdy wszystkie wartości w kolumnie są w kontekście. Albo zostanie wybrany podzbiór wartości. To może być spowodowane filtrem na tej kolumnie i wówczas mówimy, że kolumna jest filtrowana **bezpośrednio**, lub może to być spowodowane filtrem na innej kolumnie w tej samej tabeli bądź filtrem w innej tabeli, który jest propagowany przez relację. Bez względu na to, skąd pochodzi filtr, mówimy wtedy, że kolumna jest filtrowana **pośrednio** lub filtrowana krzyżowo.

DAX zawiera kilka funkcji służących do wykrywania filtrów w kontekście i ich efektów. Każda z tych funkcji jako argument pobiera odwołanie do kolumny, powiedzmy A.

- Funkcja `ISFILTERED` sprawdza, czy jest filtr bezpośrednio na kolumnie A.
- Funkcja `ISCROSSFILTERED` sprawdza, czy jest filtr na jakiejś kolumnie w modelu, który powoduje wybór w kolumnie A.
- Funkcja `HASONEFILTER` sprawdza, czy (bezpośredni) filtr na kolumnie A wybiera *dokładnie jedną* wartość.
- Funkcja `HASONEVALUE` sprawdza, czy jest jakiś filtr na kolumnie w modelu, który powoduje wybór *dokładnie jednej* wartości w kolumnie A.
- Funkcja `ISINSCOPE` sprawdza, czy w kolumnie A jest wybrana *dokładnie jedna* wartość przez filtr na kolumnie A, który pochodzi z wizualizacji. Ta funkcja została stworzona, by wykrywać bieżący poziom szczegółowości wizualizacji, który umożliwia dalsze zagłębianie się w dane.

Te funkcje mogą być pomocne, jeśli będziesz chciał zobaczyć, jak wygląda kontekst. Mogą być również użyte, by zaimplementować określone zachowanie miary DAX, choć są pewne pułapki po drodze. Znajdziesz kilka przykładów w rozdziale 5. „Bezpieczeństwo z DAX”.

Porównanie kontekstów zapytania i filtra z kontekstem wiersza

Teraz, gdy wiemy już, czym jest kontekst zapytania i kontekst filtra, możemy spojrzeć na kontekst wiersza z innej perspektywy. Załóżmy na przykład, że tworzysz kolumnę obliczeniową w tabeli `fSales` za pomocą takiej formuły:

```
TotalTax = SUM(fSales[Tax])
```

Zobaczysz, że w otrzymanej w ten sposób kolumnie `TotalTax` każdy wiersz zawiera tę samą wartość. Funkcja `SUM` obliczyła sumę *wszystkich* wierszy w tabeli, nawet mimo tego, że jesteśmy w kontekście wiersza dla jednego wiersza. Dla początkujących to często jest zaskakujące odkrycie. Spójrzmy na inny przykład, tym razem na kolumnę obliczeniową w tabeli `Date`.

```
TotalShipping = SUM(fSales[ShippingCosts])
```

I znowu przekonasz się, że w każdym wierszu jest ten sam wynik, mimo to, że mamy relację między tabelami `fSales` a `Date`. Czy relacja nie powinna spowodować, że w wyniku obliczeń otrzymamy całkowite koszty dostawy dla każdego dnia z osobna?

Te przykłady pokazują nam naturę kontekstu wiersza. Przykład z `TotalShipping` sugeruje, że w kontekście wiersza relacje nie propagują wyboru. To przydatna zasada, którą można się kierować, jednak rzeczywistość jest nieco subtelniejsza. W kontekście wiersza DAX specjalnie pozwala na użycie wartości kolumny z tej samej tabeli, poza tym nic nie jest wybierane i filtrowane. W przypadku kolumny obliczeniowej nie ma żadnych filtrów na żadnej kolumnie w tej tabeli. W rezultacie relacje nie mają nic do propagowania. To oznacza, że kiedy odwołujesz się do innej tabeli, tak jak to robi obliczenie `TotalShipping`, pracujesz z całą tabelą. Nawet jeżeli odwołujesz się do tabeli, w której znajduje się ta kolumna obliczeniowa, tak jak w przypadku obliczenia `TotalTax`, patrzysz na wszystkie wiersze.

Dlatego jeśli jesteś w kontekście wiersza, ale potrzebujesz propagacji, musisz znaleźć sposób, by **zmienić kontekst wiersza na kontekst filtra**. A w tym celu musisz użyć funkcji `CALCULATE`.

Filtrowanie DAX — zastosowanie funkcji CALCULATE

Zmiana kontekstu za pomocą kodu DAX jest jedną z najistotniejszych funkcjonalności DAX. Funkcja `CALCULATE`, używana do **zmiany kontekstu**, jest zapewne najważniejszą funkcją DAX. Przez podanie wyrażeń filtra w funkcji `CALCULATE` możesz sterować zestawami wierszy, na których działa Twoja formuła. Można to zrobić przez dodanie lub zastąpienie filtrów, ale także przez usunięcie ich z kontekstu. Jako że relacje odgrywają ważną rolę w kontekście, gdyż propagują filtry, aktywowanie, dezaktywowanie relacji lub zmiana ustawień propagacji filtra to również formy zmiany kontekstu.

Zacznijmy od przykładu miary DAX.

```
SalesLargeUnitAmount =
CALCULATE(
    SUM(fSales[SalesAmount]),
    fSales[UnitAmount] > 25
)
```

Ta miara zwraca sprzedaż dla transakcji, w których zostało sprzedanych więcej niż 25 sztuk. Pierwszy argument funkcji `CALCULATE` to obliczenie, które ma zostać wykonane. W tym przypadku jest to suma kolumny `SalesAmount` w tabeli `fSales`. Wszystkie pozostałe argumenty, a może być ich wiele, to argumenty filtra.

> DAX pozwala napisać formułę bez jawnego użycia funkcji `CALCULATE`, gdy obliczenie jest w osobnej mierze. Na przykład w podstawowej mierze sprzedaży:
>
> ```
> Sales = SUM(fSales[SalesAmount])
> ```
>
> podana wcześniej przykładowa miara może być napisana w taki sposób:
>
> ```
> SalesLargeUnitAmount = [Sales] (fSales[UnitAmount] > 25)
> ```
>
> Nie polecamy takiej składni, gdyż formuły używające funkcji `CALCULATE` jawnie są czytelniejsze, szczególnie w bardziej złożonych formułach.

Aby zrozumieć, jak działa funkcja `CALCULATE`, powinieneś pamiętać, że wykonuje ona po kolei cztery podstawowe kroki:

1. Istniejący kontekst (kontekst wiersza, kontekst zapytania lub inny kontekst filtra) jest zamieniamy w kontekst filtra.
2. Istniejące filtry, jeśli jakieś są, na kolumnach (lub całych tabelach), do których odnosimy się w argumentach filtra, są usuwane.
3. Dodawane są nowe filtry.
4. Wyrażenie w pierwszym argumencie jest wykonywane w nowym kontekście filtra.

Aby zmienić to działanie, w argumencie filtra może być używanych kilka konkretnych funkcji DAX, ale najpierw skupmy się na tych krokach na przykładzie miary `SalesLargeUnitAmount`.

Krok 1. Ustawienie kontekstu filtra

Na potrzeby funkcji `CALCULATE` najpierw trzeba stworzyć środowisko, w którym filtry mogą być zmieniane. Gdy zaczynamy w kontekście zapytania lub filtra, mamy już to środowisko, więc nie musimy nic robić. Tak więc dla naszej miary `SalesLargeUnitAmount` ten krok jest bardzo łatwy. Jednak jeżeli zaczynamy w kontekście wiersza, różnica jest ogromna.

Przejście z kontekstu wiersza do kontekstu filtra odbywa się przez tworzenie filtra na każdej kolumnie tabeli, który określa, że wartość w tej kolumnie ma być wartością dla bieżącego wiersza (pamiętaj, że kontekst filtra zawsze dotyczy jednego wiersza). W ten sposób będziemy

mieć kontekst filtra z wybranym bieżącym wierszem. Co więcej, jeśli są relacje biegnące od tej tabeli do innych, to teraz będą mieć filtry do propagowania, a co za tym idzie, możemy spodziewać się, że również w innych tabelach będą wybrane podzbiory wierszy.

Na przykład możemy zmienić formułę dla kolumny obliczeniowej `TotalTax`, której używaliśmy wcześniej, i opakować ją w funkcję `CALCULATE` (zauważ, że tutaj używamy funkcji `CALCULATE` bez żadnych argumentów filtra).

```
TotalTax2 = CALCULATE(SUM(fSales[Tax]))
```

Ta formuła dla każdego wiesza zmienia kontekst wiersza w kontekst filtra. Teraz funkcja `SUM` działa na wybranych wierszach, czyli tylko na bieżącym. Innymi słowy, wynikiem jest tylko wartość kolumny `Tax` w tym wierszu, co widać na rysunku 4.5.

Tax	TotalTax	TotalTax2
318,13	4 358 171 557,25	318,13
277,09	4 358 171 557,25	277,09
59,01	4 358 171 557,25	59,01
318,13	4 358 171 557,25	318,13
333,53	4 358 171 557,25	333,53
333,53	4 358 171 557,25	333,53
207,81	4 358 171 557,25	207,81
307,88	4 358 171 557,25	307,88

Rysunek 4.5. Wynik działania funkcji CALCULATE w kolumnie obliczeniowej

W kolumnie obliczeniowej `TotalShipping` w tabeli `Date` dzieje się to samo.

```
TotalShipping = CALCULATE(SUM(fSales[ShippingCosts]))
```

Kontekst wiersza w tabeli `Date` jest przekształcony w kontekst filtra z filtrem na każdej kolumnie tabeli. Tym razem relacja między tabelami `fSales` a `Date` może propagować wybór, przez co podzbiór zostanie wybrany spośród wierszy tabeli `fSales`, który odpowiada bieżącemu wierszowi w tabeli `Date` (rysunek 4.6).

> Możesz się zastanawiać, czy w ogóle używa się funkcji `CALCULATE` bez argumentów filtra, ale tak się dzieje częściej, niż może Ci się wydawać. Za każdym razem, gdy wywołujesz miarę w formule, to funkcja `CALCULATE` ją niejawnie opakowuje. Na przykład:
>
> ```
> SalesByCustomer = DIVIDE([Sales], [Number of Customers])
> ```
>
> W tej formule zarówno obliczenie dla `Sales`, jak i `Number of Customers` jest wykonywane w kontekście filtra. Wywoływanie miary jest często stosowaną metodą, by przejść od kontekstu wiersza do kontekstu filtra, co często musisz zrobić.

Date	Year	Month	Month Name	Number of Transactions	TotalShipping
wtorek, 28 kwietnia 2020	2020	4	April	272	321 378,88
środa, 29 kwietnia 2020	2020	4	April	176	207 393,42
czwartek, 30 kwietnia 2020	2020	4	April	112	109 006,97
piątek, 1 maja 2020	2020	5	May	160	185 256,18
sobota, 2 maja 2020	2020	5	May	224	282 413,38
niedziela, 3 maja 2020	2020	5	May	272	361 922,66
poniedziałek, 4 maja 2020	2020	5	May	192	250 798,15
wtorek, 5 maja 2020	2020	5	May	256	292 352,95
środa, 6 maja 2020	2020	5	May	80	104 958,11
czwartek, 7 maja 2020	2020	5	May	176	178 446,53
piątek, 8 maja 2020	2020	5	May	176	200 070,26

Rysunek 4.6. Funkcja CALCULATE sprawia, że relacje propagują filtry

Efektem dodania funkcji `CALCULATE` jest poprawna kwota `TotalShipping` dla każdego dnia.

Teraz, gdy mamy już wstępny kontekst filtra, funkcja `CALCULATE` może przejść do następnego kroku.

Krok 2. Usuwanie istniejących filtrów

Drugim wykonywanym przez funkcję `CALCULATE` krokiem jest usunięcie filtrów z nowego kontekstu filtra. Proces jest prosty — sprawdzane są filtry w każdej kolumnie, do której odwołuje się jeden z argumentów filtra. Jeśli jest jakiś filtr na tej kolumnie, to jest on usuwany.

W naszym przykładzie miary `SalesLargeUnitAmount` mamy tylko jeden argument filtra, to jest:

```
fSales[UnitAmount] > 25
```

Ten argument filtra sprawia, że funkcja `CALCULATE` usuwa wszystkie istniejące filtry na kolumnie `fSales[UnitAmount]`.

Czasami zapomina się, że kolumny, które *nie* są wspomniane w argumentach filtra, zachowują swoje filtry (gdy takie są). A jako że nie masz pełnej kontroli nad tym, jak wygląda oryginalny kontekst, powinieneś dokładnie przemyśleć każdy scenariusz, w jakim Twoja miara może być użyta. Możliwe, że będziesz musiał usunąć więcej filtrów, niż początkowo zakładałeś. Po omówieniu wszystkich czterech kroków zobaczysz przykład, jaki wpływ mogą mieć inne filtry.

Działanie funkcji `CALCULATE` w kroku 2. może być zmienione za pomocą funkcji `KEEPFILTERS`. Powoduje ona, że funkcja `CALCULATE` pomija krok 2. dla argumentu filtra, dla którego zastosowałeś funkcję `KEEPFILTERS`. Na przykład:

```
SalesLargeUnitAmount KeepFilters =
CALCULATE(
    [Sales],
    KEEPFILTERS(fSales[UnitAmount] > 25)
)
```

W tej formule, jeśli jest jakiś filtr w kolumnie UnitAmount, zostanie on zachowany, a następnie pojawi się nowy filtr dodany w kroku 3.

Krok 3. Dodawanie nowych filtrów

Trzeci krok wykonywany przez funkcję CALCULATE to dodawanie nowych filtrów. Podobnie jak w kroku 2., funkcja przechodzi przez argumenty filtra i tworzy nowe filtry zgodnie z ich wytycznymi. W naszym przykładzie argument filtra:

```
fSales[UnitAmount] > 25
```

powoduje dodanie nowego filtra w kolumnie UnitAmount, przez co wybierane są tylko wartości większe od 25.

Aby wiedzieć, jak działa funkcja CALCULATE, warto zapamiętać, że kroki 2. i 3. są wykonywane w takiej kolejności. Sama kolejność argumentów filtra jest bez znaczenia. Oto prosty przykład:

```
Sales373_374 =
CALCULATE(
    [Sales],
    Products[ProductID] = 373,
    Products[ProductID] = 374
)
```

Wiele osób dopiero zaczynających swoją przygodę z DAX spodziewa się, że ta formuła zwróci sprzedaż dla produktu 374. Myślą, że najpierw filtr jest ustawiany na ProductID równy 373, a następnie na 374. W rzeczywistości taka miara zawsze będzie zwracać puste wartości, ponieważ są dodane dwa filtry na kolumnie ProductID, co wiąże się z tym, że wartość kolumny musiałaby być równa 373 i 374. Nie ma żadnych wierszy spełniających ten warunek i dlatego miara TotalSales zwróci pustą wartość (przy założeniu, że istnieje relacja propagująca filtry z tabeli Products do tabeli fSales).

Możesz pomyśleć, że to banalny przykład, jednak w nieco innej postaci może on wielu zmylić.

```
Sales373OrWhat =
CALCULATE(
    [Sales],
    Products[ProductID] = 373,
    ALL(Products)
)
```

Tak samo jak w poprzednim przykładzie, należy pamiętać, że najpierw są usuwane filtry dla każdego argumentu filtra, a następnie dodawane są nowe filtry. Wynikiem jest sprzedaż dla ProductID 373.

Krok 4. Wykonywanie obliczenia

Ostatni krok funkcji `CALCULATE` jest oczywisty. Po ustawieniu kontekstu filtra, usunięciu filtrów i dodaniu nowych wyrażenie podane jako pierwszy argument może być wykonane w nowym kontekście. To jest namacalny dowód, gdyż to tutaj tak naprawdę możemy zobaczyć efekt wszystkich zmian kontekstu.

Na rysunku 4.7 został pokazany przykład tego, jak manipulowanie filtrem może być podchwytliwe.

Group	Sales373	SalesRearWheel525	Sales
⊞ **Oil filters**			**190 215 158**
⊞ **Other**			**11 497 355 884**
⊟ **Rear wheel**	**735 209 424**	**735 209 424**	**2 391 842 110**
228	735 209 424		21 543 669
239	735 209 424		1 606 503 554
373	735 209 424	735 209 424	735 209 424
466	735 209 424		5 569 759
467	735 209 424		23 015 703
⊞ **Transport**			**415 907 579**
Suma	**735 209 424**	**735 209 424**	**14 495 320 732**

Rysunek 4.7. Dane wyjściowe dla przykładowych miar

W tej macierzy pokazujemy informacje o produktach za pomocą kolumn `Group` i `ProductID` jako etykiet. Skupiamy się szczególnie na produkcie `373` w grupie `Rear wheel`. Nazwa tego produktu w kolumnie `Product` to `REAR WHEEL STEEL #525`. Chcemy móc porównać sprzedaż każdego produktu ze sprzedażą produktu 373. Możesz wyobrazić sobie, że produkt 373 to najbardziej strategiczny produkt naszej firmy i chcemy wyrażać sprzedaż każdego produktu jako procent sprzedaży produktu 373.

Aby dokonać takiego porównania, potrzebujemy obliczenia, które zwraca sprzedaż produktu 373 w każdym wierszu wizualizacji. Próbujemy to osiągnąć za pomocą dwóch miar. Pierwsza z nich to:

```
Sales373 =
CALCULATE(
    [Sales],
    Products[ProductID] = 373
)
```

A druga to:

```
SalesRearWheel525 =
CALCULATE(
    [Sales],
    Products[Product] = "REAR WHEEL STEEL #525"
)
```

Miara `Sales` jest widoczna w wizualizacji, więc możesz lepiej zobaczyć, co się dzieje. W większości wierszy macierzy mamy dwa filtry w kontekście zapytania — jeden dla kolumny `Group`, a drugi dla kolumny `ProductID`. Wyjątkami są wiersze sum częściowych (gdzie mamy tylko filtr `Group`) i sumy (gdzie nie ma żadnych filtrów).

Widać wyraźnie, że te dwie miary z funkcją CALCULATE zwracają różne wyniki. Skąd ta różnica? Jako że miara Sales373 w argumencie filtra odwołuje się do kolumny ProductID, każdy istniejący na tej kolumnie filtr został usunięty (krok 2.) przed dodaniem nowego filtra (krok 3.). W wizualizacji na przykład w wierszu produktu 239 filtr *ProductID równa się 239* został usunięty, a dodany został filtr *ProductID równa się 373*. I to dlatego obliczenie zwraca sprzedaż dla produktu 373.

W przypadku miary SalesRearWheel525 dzieje się coś innego. Tutaj argument filtra odwołuje się do kolumny Product, więc każdy istniejący już tam filtr został usunięty (krok 2.). Potem został dodany nowy filtr (krok 3.). Gdy znowu popatrzymy na produkt 239, dostrzeżemy, że kontekst zapytania zawiera filtry na kolumnach Group i ProductID. Miara nie usuwa tych filtrów, ale dodaje nowy filtr do kolumny Product. W wyjściowym kontekście filtra wybrane są wszystkie wiersze tabeli Product, które spełniają warunki wszystkich trzech filtrów. Innymi słowy, żaden wiersz nie został wybrany, więc wynik jest pusty, chyba że stanie się tak, że te trzy filtry wskażą na ten sam produkt. Jest to prawda jedynie dla produktu 373 i to dlatego tylko w tym wierszu widać wynik.

Ten sam tok myślowy wyjaśnia, dlaczego miara Sales373 zwraca wyniki tylko w grupie Rear wheel. Gdy filtr na kolumnie Group wybiera inną grupę, w połączeniu z filtrem kolumny ProductID (nowo dodanym filtrem) prowadzi do pustego wyboru w tabeli Product.

Usuwanie filtrów za pomocą funkcji ALL

Obie miary z poprzedniego punktu mają ten sam problem, oczywiście w zależności od kontekstu. Aby stworzyć miarę, która zawsze zwraca wyniki dla produktu 373, bez względu na wybór produktów dokonany w kontekście zapytania, musimy pozbyć się wszystkich filtrów, które możemy spotkać po drodze.

Ważne, aby dokładnie kontrolować, które filtry są usuwane. Do tego służą funkcje DAX z kategorii ALL. Różnica między funkcjami z tej kategorii leży w usuwanych filtrach.

- Funkcja ALL może pobierać jako argumenty jedno odwołanie lub więcej odwołań do kolumn lub do tabeli. Usuwa filtry z podanych kolumn lub ze wszystkich kolumn podanej tabeli. Jeśli bardzo chcesz, możesz użyć funkcji ALL bez argumentów, by usunąć wszystkie filtry z całego modelu Power BI. Na przykład:

```
ALL(Cities[Country])
ALL(Cities[Country], Cities[State])
ALL(Cities)
ALL()
```

> Gdy używasz funkcji ALL na tabeli, filtry są usuwane również w powiązanych kolumnach. Na przykład wyrażenie ALL(fSales) usunie także filtry z tabeli Cities, gdy między tabelami fSales i Cities zachodzi relacja wiele do jednego. Zobacz również opis funkcji ALLCROSSFILTERED.

- Funkcja ALLEXCEPT może być używana jako alternatywa do funkcji ALL z wieloma kolumnami jako argumentami. Zamiast wymieniać wszystkie kolumny, z których chcesz usunąć filtry, podajesz tylko tabelę i jej kolumny, w których filtry powinny być zachowane. Filtry z pozostałych kolumn tej tabeli zostaną usunięte. Na przykład:

```
ALLEXCEPT(Cities, Cities[Country])
```

- Funkcja ALLNOBLANKROW. Gdy używasz funkcji ALL, otrzymany kontekst zawiera wszystkie wartości z określonej kolumny. W tym mogą być zawarte również wartości z pustych wierszy dodanych do tabeli z powodu niepełnej relacji (zobacz rozdział 2. „Projektowanie modelu", te wartości są obowiązkowo puste). Jeśli nie chcesz, by te puste wartości zostały ujęte w Twoim kontekście, powinieneś użyć funkcji ALLNOBLANKROW zamiast ALL. Ta funkcja pobiera tylko jeden argument — kolumnę albo tabelę. Na przykład:

```
ALLNOBLANKROW(Cities[Country])
```

- Funkcja ALLSELECTED jest specjalną funkcją ALL, gdyż jest jedyną funkcją, która zna pochodzenie filtrów. Funkcja ALLSELECTED usuwa filtry, ale tylko te, które pochodzą z etykiet w wizualizacji, w której dana miara została użyta. Zewnętrzne filtry, jak te pochodzące z fragmentatorów, filtrów strony lub innych wizualizacji, pozostają nietknięte. Ta funkcja jest używana do tworzenia miar, które agregują wybrane elementy w wizualizacji, na przykład by obliczyć wartości procentowe, których suma w wizualizacji zawsze wynosi 100%. Omawiana funkcja jako argument przyjmuje tabelę, jedną lub więcej kolumn albo nic. Na przykład:

```
ALLSELECTED(Cities[Country])
```

- Funkcja ALLCROSSFILTERED została wprowadzona dla złożonych modeli Power BI, modeli zawierających zarówno tabele w trybie zapytań bezpośrednich, jak i trybie importu lub różnych połączeń zapytań bezpośrednich. Relacje między tabelami o różnym pochodzeniu w takich modelach są „słabe" i nie zapewniają standardowego działania. I tak wyrażenie ALL(fSales) nie usunie filtrów z tabeli Cities, gdy tabela fSales jest w trybie zapytań bezpośrednich, a tabela Cities w trybie importu. Funkcja ALLCROSSFILTERED pobiera jako argument odwołanie do tabeli i usunie filtry zarówno z tej tabeli, jak i z tabel powiązanych, nawet pomimo słabej relacji. W standardowym modelu w trybie importu nie musisz używać funkcji ALLCROSSFILTERED. Na przykład:

```
ALLCROSSFILTERED(fSales)
```

Istnieje jeszcze jedna funkcja DAX, której możesz użyć do usunięcia filtrów w wyrażeniu CALCULATE — REMOVEFILTERS. Pobiera ona jako argument jedną lub więcej kolumn albo całą tabelę. Na przykład:

```
CALCULATE(
    [Sales],
    REMOVEFILTERS(Cities)
)
```

Ta funkcja została wprowadzona jako łatwiejsza do zrozumienia alternatywa dla funkcji ALL. My wolimy ALL i nigdy nie używamy REMOVEFILTERS.

Jeśli odpowiednio wybierzesz jedną lub więcej funkcji ALL, możesz sprawić, by funkcja CALCULATE wykonywała dokładnie to, co chcesz. Przypominamy, że zamierzamy stworzyć miarę, która zawsze będzie zwracać sprzedaż produktu 373, czyli wiemy dokładnie, jak ma wyglądać kontekst filtra. Nie mamy kontroli nad tym, jakie filtry istnieją w kontekście zapytania, ale mamy wpływ na to, jakie filtry są usuwane. Spójrz na zaktualizowaną miarę:

```
SalesRearWheel525_ALL =
CALCULATE(
    [Sales],
    Products[Product] = "REAR WHEEL STEEL #525",
    ALL(Products)
)
```

Dzięki tej formule *wszystkie* istniejące filtry w tabeli Products zostaną usunięte przed dodaniem nowego filtra do kolumny Product. Różnicę widać wyraźnie, co możesz zaobserwować na rysunku 4.8.

Group	Sales373	SalesRearWheel525	SalesRearWheel525_ALL	Sales
⊞ **Oil filters**			**735 209 424**	**190 215 158**
⊞ **Other**			**735 209 424**	**11 497 355 884**
⊟ **Rear wheel**	**735 209 424**	**735 209 424**	**735 209 424**	**2 391 842 110**
228	735 209 424		735 209 424	21 543 669
239	735 209 424		735 209 424	1 606 503 554
373	735 209 424	735 209 424	735 209 424	735 209 424
466	735 209 424		735 209 424	5 569 759
467	735 209 424		735 209 424	23 015 703
⊞ **Transport**			**735 209 424**	**415 907 579**
Suma	**735 209 424**	**735 209 424**	**735 209 424**	**14 495 320 732**

Rysunek 4.8. Użycie funkcji ALL

Nie dość, że wyniki dla produktu 373, produktu REAR WHEEL STEEL #525 są zwracane dla innych produktów, to jeszcze wynik jest zwracany nawet, jeśli kontekst zapytania filtruje inne grupy produktów. Mamy teraz podstawowy element składowy do wyrażania sprzedaży dowolnego produktu w porównaniu ze sprzedażą produktu 373 i użyjemy tego elementu w prostym dzieleniu.

```
Sales% = DIVIDE([Sales], [SalesRearWheel525_ALL]
```

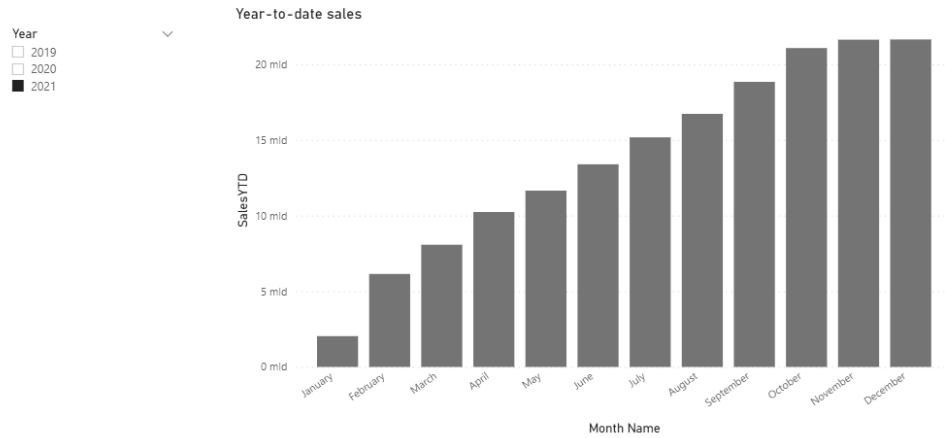
Przez połączenie argumentów filtra i funkcji ALL możemy stworzyć wiele wszechstronnych miar DAX. Jednak niektóre filtry trudniej stworzyć i określić, a wśród nich są filtry obsługujące kalendarz. To dlatego DAX zawiera specjalną kategorię służących do tego funkcji, które omówimy w następnym podrozdziale.

Analiza czasowa

Prawdopodobnie prawie w ogóle nie ma modeli niezawierających analizy na przestrzeni wybranego okresu. Chcemy na przykład porównać aktualne wyniki z tymi z zeszłego roku.

Mogą być wymagane również inne obserwacje związane z kalendarzem, między innymi wyniki od początku bieżącego roku do dnia dzisiejszego (ang. *year to date*), suma bieżąca lub wskaźniki wzrostu z dowolnego okresu w przeszłości. Trudność polega na tym, że kalendarz gregoriański jest dość zawiły, gdyż większość lat ma 365 dni, ale niektóre mają 366, a miesiące mają różną liczbę dni, od 28 do 31.

Niezależnie od tych zawiłości kalendarza analiza czasowa polega po prostu na filtrowaniu, by zmienić kontekst. Weźmy na przykład wykres sprzedaży od początku roku do danego dnia, widoczny na rysunku 4.9.



Rysunek 4.9. Wykres sprzedaży od początku roku do danego dnia

Z definicji okresu od początku roku do danego dnia wynika, że to, co widzisz w słupku `August` (sierpień), to całkowita sprzedaż od 1 stycznia 2021 roku do 31 sierpnia 2021 roku. Jednak kontekst zapytania dla tej kolumny zawiera filtr dla lat (2021) i miesięcy (sierpień), co powoduje wybór okresu od 1 sierpnia 2021 roku do 31 sierpnia 2021 roku. W wyraźny sposób widać, że kontekst musi być zmieniany, by móc zwracać sprzedaż całkowitą od początku roku do wybranego dnia.

Zatem w obliczeniu sprzedaży od początku roku do wybranego dnia użyjesz funkcji `CALCULATE` z kilkoma argumentami filtrów.

```
SalesYTD =
CALCULATE(
    [Sales],
    ... (kilka argumentów filtra)
)
```

W rzeczywistości funkcje analizy czasowej DAX implementują argumenty filtrów w funkcji `CALCULATE`, by obsługiwać złożoność kalendarza. Filtr okresu od początku roku do wybranego dnia jest zwracany przez funkcję `DATESYTD`.

```
SalesYTD =
CALCULATE(
    [Sales],
    DATESYTD('Date'[Date])
)
```

Funkcja DATESYTD działa w oparciu o kontekst zapytania w tabeli danych według następujących kroków:

1. Pobranie ostatniej daty kontekstu.
2. Określenie roku, z którego ta data pochodzi, i pierwszego dnia tego roku.
3. Dodanie na kolumnie Date[Date] filtra, który wybiera wszystko od pierwszego dnia roku do ostatniej daty w kontekście.

   Z tym nowym kontekstem funkcja CALCULATE może wykonać swoje obliczenia, w naszym przypadku w mierze Sales. Ale, poczekaj, argument filtra funkcji DATESYTD odwołuje się do kolumny Date, a na wykresie z rysunku 4.9 te filtry są na kolumnach Year i Month! To bardzo często spotykana sytuacja i dlatego funkcja DATESYTD wykonuje jeszcze jeden krok.
4. Dodanie niejawnego argumentu ALL('Date').

Ten ostatni krok jest wspólny dla wszystkich funkcji analizy czasowej i dzięki temu nie musimy za każdym razem dodawać sami funkcji ALL.

> Niejawne wyrażenie ALL('Date'[Date]) jest dodawane jedynie wtedy, gdy oficjalnie oznaczyłeś swoją tabelę jako tabelę dat modelu Power BI lub gdy relacje biegnące od tabeli faktów do Twojej tabeli dat są stworzone dla kolumn z danymi typu *Data*. Mimo że funkcje analizy czasowej mogą działać poprawnie bez oficjalnego oznaczania tabeli jako tabeli dat, radzimy, by mimo wszystko to robić.

Jak już zostało wspomniane, analiza czasowa jest bardzo często potrzebna. Dlatego wiele funkcji analizy czasowej ma swoje krótsze, łatwiejsze do użycia wersje. Funkcja DATESYTD używana w połączeniu z funkcją CALCULATE może być zastąpiona funkcją TOTALYTD.

```
SalesYTD_short =
TOTALYTD(
    [Sales],
    'Date'[Date]
)
```

Ta formuła jest dokładnie taka sama, lecz z punktu widzenia składni inna. Choć to może być zaleta, minus jest taki, że dla wielu osób zaczynających naukę DAX ta funkcja wygląda, jakby obliczała **sumę** od początku roku do wybranego dnia. A tak naprawdę funkcja TOTALYTD zmienia tylko kontekst. Może zwracać średnią od początku roku do danego dnia lub inne obliczenia, a to wszystko zależy od miary lub wyrażenia podanego w pierwszym argumencie.

> Dla uzupełnienia informacji należy wspomnieć, że możliwe jest użycie funkcji DATESYTD i TOTALYTD dla okresów fiskalnych, które nie zaczynają się 1 stycznia. DATESYTD może przyjmować drugi argument, z którego powinna być w stanie wydzielić dzień w roku, na przykład „8/31" lub „2020/9/30". Ten argument jest traktowany jako *ostatni* dzień roku. Chyba nigdy już tak naprawdę nie zrozumiemy, dlaczego w funkcji TOTALYTD to jest czwarty argument, co oznacza, że musisz podać trzeci argument. To jest dodatkowy filtr. Możesz bezpiecznie użyć wyrażenia ALL('Date'[Date]), które i tak jest dodawane niejawnie.

Obok DATESYTD najczęściej używane funkcje analizy czasowej to:

- Funkcja SAMEPERIODLASTYEAR pobiera bieżący kontekst i przesuwa go dokładnie o jeden rok wstecz. To jest oczywiście to, czego potrzebujesz, by obliczyć liczby zmieniające się rok do roku. O dziwo, ta funkcja nie ma skróconej wersji. Sprzedaż dla zeszłego roku może być obliczona w następujący sposób:

```
SalesLY =
CALCULATE(
    [Sales],
    SAMEPERIODLASTYEAR('Date'[Date])
)
```

- Funkcja DATESINPERIOD zwraca okres zaczynający się (lub kończący) na podanej dacie, trwający określoną liczbę dni, miesięcy, kwartałów lub lat. Ta funkcja jest przydatna w obliczaniu sumy bieżącej. Na przykład suma bieżąca sprzedaży dla ostatnich 12 miesięcy (czyli patrzymy 12 miesięcy wstecz) jest obliczana za pomocą pokazanej poniżej formuły. Tutaj wyrażenie MAX('Date'[Date]) jest używane, by uzyskać ostatni dzień w kontekście jako data odwołania.

```
SalesRollingTotal =
CALCULATE(
    [Sales],
    DATESINPERIOD(
          'Date'[Date],
          MAX('Date'[Date]),
          -12, MONTH
    )
)
```

Podczas pracy z funkcjami analizy czasowej należy pamiętać, że za każdym razem taka funkcja zmienia kontekst w tabeli dat, nic więcej. To oznacza, że żadna tabela w modelu, która nie jest powiązana z tabelą dat, nie będzie miała zmienionego kontekstu. Tak więc uzyskasz nieoczekiwane wyniki, gdy Twoja tabela dat jest „krótka". Na przykład jeśli Twoja tabela dat zaczyna się rokiem 2020 i używasz funkcji SAMEPERIODLASTYEAR w kontekście, który wybiera daty w lutym 2020, wynikiem będzie pusty kontekst kalendarza. Co za tym idzie, miara zwróci pusty wynik, nawet jeśli agregowana przez Ciebie tabela faktów ma daty z 2019 roku i z lat wcześniejszych.

Zmiana działania relacji

W rozdziale 2. „Projektowanie modelu" dowiedziałeś się, że między tabelami może istnieć wiele relacji, ale tylko jedna z nich może być aktywna. To samo dotyczy ścieżek relacji między tabelami, model Power BI pozwala tylko na jedną aktywną ścieżkę między dwiema dowolnymi tabelami w modelu. To oczywiście przydaje się tylko wtedy, gdy w razie potrzeby możemy aktywować nieaktywną relację. Możesz to zrobić za pomocą funkcji `USERELATIONSHIP`.

Funkcja `USERELATIONSHIP` jest używana jako argument filtra w funkcji `CALCULATE`. To może wydać się zaskakujące, ale naprawdę ma sens. Relacja między tabelą faktów a tabelą filtra daje nam pewność, że wybór w tabeli filtrów przejdzie do tabeli faktów. Aktywacja innej relacji oznacza, że teraz ta relacja propaguje wybór do tabeli faktów, przez co inne wiersze są wybierane w tabeli faktów. Innymi słowy, aktywowanie innej relacji oznacza zmianę kontekstu dla obliczenia. A funkcja `CALCULATE` służy właśnie do zmiany kontekstu.

Funkcja `USERELATIONSHIP` pobiera dwa argumenty, które są odwołaniem do kolumn, między którymi zachodzi relacja do aktywacji. Na przykład jeśli aktywna relacja między tabelami `fSales` a `Date` jest dla kolumny `fSales[OrderDate]`, ale Ty chcesz użyć relacji dla kolumny `fSales[InvoiceDate]`, to możesz zastosować następującą formułę:

```
TotalInvoiced =
CALCULATE(
    [Sales],
    USERELATIONSHIP(fSales[InvoiceDate], 'Date'[Date])
)
```

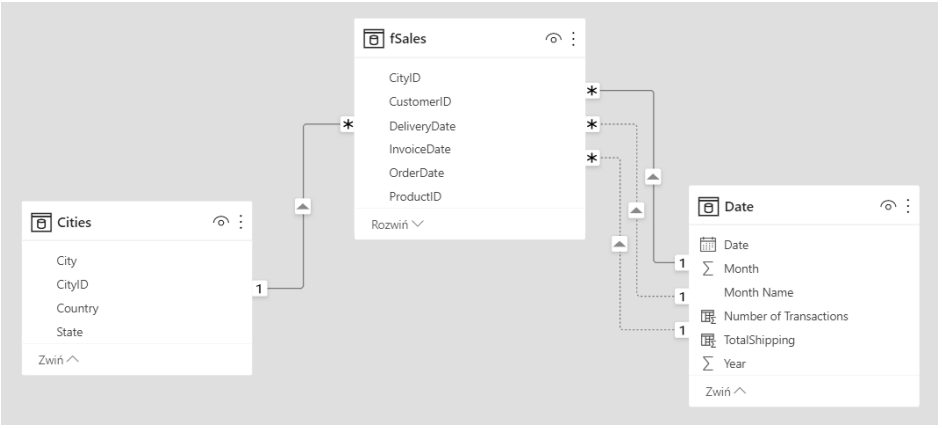
Powinno być jasne, że znaczenie otrzymanych wyników zmienia się, gdy używasz innej relacji. Podczas gdy miara `Sales` zwraca kwotę zamówień, miara `TotalInvoiced` kwotę, na jaką zostały wystawione faktury. Tej pierwszej miary użyto by do analizy dochodu, a ta druga mogłaby pomóc w analizie przepływu gotówki (w której obliczenia na bieżących płatnościach byłyby kluczowe). To oczywiście zależy od tego, co przez sprzedaż rozumieją przedstawiciele firmy.

Innym sposobem zmiany działania relacji jest zmiana działania propagacji filtra w aktywnej relacji. Do tego służy funkcja DAX o nazwie `CROSSFILTER`, która również jest używana jako argument filtra w funkcji `CALCULATE`.

Podobnie jak funkcja `USERELATIONSHIP`, `CROSSFILTER` jako argumenty przyjmuje dwie kolumny biorące udział w relacji. Za pomocą trzeciego argumentu możesz ustawić kierunek propagacji filtra lub rodzaj filtra krzyżowego tej relacji. Do wyboru jest pięć typów filtra:

- `OneWay` propaguje filtr w domyślnym kierunku, od tabeli z kluczami podstawowymi (unikatowymi) do tabeli z kluczami obcymi (nieunikatowymi).
- `Both` propaguje filtr w dwie strony.
- `None` nie propaguje w ogóle filtrów.
- `OneWay_LeftFiltersRight` propaguje filtry w jednym kierunku, od kolumny podanej jako pierwszy argument do kolumny z drugiego argumentu.
- `OneWay_RightFiltersLeft` propaguje filtry w jednym kierunku, od kolumny podanej jako drugi argument do kolumny z pierwszego argumentu.

Na potrzeby przykładu zastosowania funkcji `CROSSFILTER` załóżmy, że chcesz wiedzieć, w jak wielu stanach dany produkt został sprzedany. Model zawiera relacje między tabelami `fSales`, `Cities` a `Date` (rysunek 4.10).



Rysunek 4.10. Schemat modelu dla sprzedaży według miast

Na schemacie modelu z rysunku 4.10 zauważ, że moglibyśmy wybrać miesiąc, a wszystkie transakcje sprzedaży z tego miesiąca zostałyby wybrane przez aktywną relację. Nie możemy bezpośrednio policzyć liczby stanów, jednak relacja między tabelami `Cities` a `fSales` propaguje filtry tylko z tabeli `Cities` do tabeli `fSales`, a my potrzebujemy propagacji filtra w przeciwnym kierunku. W takim przypadku wybranie wierszy w tabeli `fSales` spowoduje jednocześnie wybranie odpowiadających im wierszy z tabeli `Cities`, a wtedy moglibyśmy policzyć, w ilu stanach dany produkt został sprzedany.

Jest oczywiste, że kierunek propagacji filtra musi zostać zmieniony, a pomoże nam w tym następująca formuła:

```
StatesSoldTo =
CALCULATE(
    DISTINCTCOUNT(Cities[State]),
    CROSSFILTER(fSales[CityID], Cities[CityID], Both)
)
```

Za pomocą funkcji `DISTINCTCOUNT` możemy policzyć unikatowe wartości w kolumnie `State`. Ale najpierw funkcja `CROSSFILTER` powoduje wybranie wierszy z tabeli `Cities` na podstawie wierszy wybranych z tabeli `fSales`.

Funkcje tablicowe w DAX

Można wiele zrobić z podstawowymi funkcjami agregacji, takimi jak `SUM` (suma) czy `AVERAGE` (średnia), w połączeniu z filtrowaniem DAX za pomocą funkcji `CALCULATE`. Ale język DAX idzie jeszcze dalej. Ten podrozdział traktuje o funkcjach tablicowych, które otwierają worek z bardziej

zaawansowanymi obliczeniami w DAX. W części II tej książki przekonasz się, że w wielu scenariuszach biznesowych użycie funkcji tablicowych DAX jest bardzo pomocne.

Agregacje tabeli

Na początku spójrzmy na prostą agregację w DAX.

```
Sales1 = SUM(fSales[SalesAmount])
```

W tej formule funkcja `SUM` przechodzi przez tabelę `fSales` i pobiera wartości z kolumny `SalesAmount`, z każdego wiersza. Wszystkie te wartości są sumowane i zwracany jest wynik końcowy.

> Z powodu szczególnego sposobu, w jaki Power BI szyfruje i zapisuje dane (zobacz rozdział 2. „Projektowanie modelu"), przebieg działania tej funkcji może być nieco inny z technicznego punktu widzenia. Jednak logicznie rzecz biorąc, to właśnie to, co robi funkcja `SUM`, jest dokładnie tym, czego potrzebujemy.

Teraz załóżmy, że kolumna `SalesAmount` jest kolumną obliczeniową, utworzoną za pomocą następującej formuły:

```
SalesAmount = fSales[UnitAmount] * fSales[SalesPrice]
```

Jako że zarówno kolumna `UnitAmount`, jak i `SalesPrice` znajdują się w tabeli `fSales`, warto zadać sobie pytanie — czy możemy obliczyć sprzedaż bez użycia kolumny `SalesAmount`? W końcu chcemy za wszelką cenę unikać kolumn obliczeniowych w modelu. Obliczenie, którego szukamy, powinno jeszcze raz przejść przez tabelę `fSales`, ale zamiast pobierać wartości z kolumny `SalesAmount`, powinno wziąć wartości z kolumn `UnitAmount` oraz `SalesPrice` i pomnożyć je przez siebie. Funkcją DAX, która nam w tym pomoże, jest `SUMX`.

```
Sales2 =
SUMX(
    fSales,
    fSales[UnitAmount] * fSales[SalesPrice]
)
```

Podczas gdy funkcja `SUM` przyjmuje jako argument tylko odwołanie do kolumny, do funkcji `SUMX` musimy przekazać tabelę (w naszym przykładzie tabelę `fSales`), a jako drugi argument wyrażenie, które ma być wykonane dla każdego wiersza tej tabeli. Funkcję `SUMX` nazywamy **funkcją agregującą tabelę**. Możesz się również spotkać z nazwą **funkcja iteracyjna**, gdyż funkcja `SUMX` przechodzi przez wszystkie wiersze tabeli podanej jako pierwszy argument.

Większość podstawowych funkcji agregujących ma swój odpowiednik tablicowy, między innymi `SUMX`, `AVERAGEX`, `MINX`, `MAXX`, `COUNTX`, `COUNTAX`, `PRODUCTX` i `CONCATENATEX`. Jak wyraźnie widać na podanej właśnie liście, funkcje agregujące tabele można rozpoznać po literze `X` dodanej na końcu nazwy funkcji. Mniej znane funkcje agregujące tabele to funkcje statystyczne, takie jak `MEDIANX`, `PERCENTILEX` i `STDEVX` (ostatnich dwóch nie będziemy tutaj omawiać).

Prostą funkcją agregującą tabelę jest COUNTROWS, która zwraca liczbę wierszy w tabeli i nie ma swojego odpowiednika tablicowego. Funkcja RANKX jest tablicowym odpowiednikiem funkcji RANK.EQ.

Podany wyżej przykład przydaje się, gdy chcesz unikać kolumn obliczeniowych. Ale prawdziwa siła funkcji agregujących tabele polega na tym, że jako pierwszy argument możesz podać dowolną tabelę. Załóżmy, że chcesz stworzyć miarę, która zwraca średnią sprzedaż w poszczególnych miastach. W tym celu obliczenie powinno przejść nie przez wiersze tabeli fSales, która zawiera pojedyncze transakcje sprzedaży, ale przez tabelę Cities.

```
SalesPerCity =
AVERAGEX(
    Cities,
    [Sales]
)
```

Omówimy dokładnie, co dzieje się w tym obliczeniu, ale najpierw powiedzmy więcej o tabelach, które mogą zostać użyte. Nie tylko dowolna tabela w modelu może zostać użyta z funkcją agregującą tabelę, ale możesz stworzyć nawet własną specjalną tabelę i przekazać ją do tej funkcji. Tego typu tabele noszą nazwę **tabel wirtualnych** (gdyż nazwa „tabele obliczeniowe" jest już zarezerwowana przez model Power BI).

Zastosowanie tabel wirtualnych

W poprzednim punkcie widzieliśmy formułę obliczającą średnią sprzedaż w poszczególnych miastach. Teraz załóżmy, że chcemy obliczyć średnią sprzedaż w poszczególnych stanach. Stosując to samo podejście jak w przypadku miary SalesPerCity, potrzebujemy tabeli tylko z jednym wierszem na każdy stan, przez którą przejdzie nasza funkcja agregująca. Taka tabela nie jest dostępna w modelu, gdyż stany są wymienione w kolumnie States w tabeli Cities. Dlatego musimy stworzyć sami taką tabelę i choć w tym prostym przypadku moglibyśmy po prostu dodać tabelę State do modelu (na przykład jako tabelę obliczeniową), lepiej stworzyć **tabelę wirtualna**. Tabela ta będzie istnieć tylko podczas obliczeń wykonywanych przez tę miarę.

W języku DAX mamy szeroki wachlarz dostępnych funkcji służących do tworzenia tabel wirtualnych. Złożoność tych funkcji polega ogólnie na tym, że zwracają tabelę. To oznacza, że nie ma standardowego mechanizmu, by podglądnąć uzyskany rezultat, tak jak to możesz zrobić w przypadku miary DAX, którą możesz zobaczyć za pomocą wizualizacji Power BI. Tworzenie tabeli obliczeniowej w modelu Power BI za pomocą tej samej funkcji mogłoby tutaj pomóc, ale praca z funkcjami tablicowymi i tak wymaga umiejętności myślenia abstrakcyjnego.

Na początku naszego wyzwania ze stanami funkcja VALUES pobiera jako argument odwołanie do kolumny i zwraca tabelę z unikatowymi wartościami z tej kolumny. Na przykład:

```
VALUES(Cities[State])
```

To wyrażenie zwraca tabelę z unikatowymi wartościami State. Odpowiednikiem funkcji VALUES jest funkcja DISTINCT, która również zwraca unikatowe wartości z kolumny, z tą różnicą, że funkcja DISTINCT nie ujmuje pustych wartości w pustych wierszach pochodzących z niepełnej relacji (patrz rysunek 2.5, rozdział 2. „Projektowanie modelu"). To do Ciebie należy decyzja, czy chcesz mieć puste wartości, czy nie.

Oto formuła dla sprzedaży dla poszczególnych stanów:

```
SalesPerState =
AVERAGEX(
    VALUES(Cities[State]),
    [Sales]
)
```

Istnieje cała grupa funkcji DAX zwracających tabele, ale nie będziemy ich tutaj wszystkich wymieniać. Najczęściej używane funkcje to:

- Funkcja SUMMARIZE, choć jest bardzo wszechstronna i złożona, a co więcej, jest w stanie wygenerować wyniki w formie podobnej do tabeli przestawnej, to nie tak jej używamy w miarach DAX. Ta funkcja jest bardzo przydatna jako rozwinięcie funkcji VALUES. Podczas gdy VALUES zwraca unikatowe wartości z jednej kolumny, funkcja SUMMARIZE może zwrócić unikatowe zestawienia wartości z więcej niż jednej kolumny. Na przykład unikatowe połączenie wartości z kolumn CityID i ProductID z tabeli fSales może być uzyskane w następujący sposób (zauważ, że w pierwszym argumencie musisz podać samą tabelę):

  ```
  SUMMARIZE(fSales, fSales[CityID], fSales[ProductID])
  ```

- Funkcja FILTER pobiera dwa argumenty: pierwszy to tabela (istniejąca albo taka, która jest zwracana przez inną funkcję tablicową), a drugi to wyrażenie, które ma być wykonane dla każdego wiersza z tej tabeli. Wynikiem wyrażenia powinno być prawda albo fałsz, a funkcja FILTER uwzględni tylko te wiersze, dla których zwrócona wartość równa się prawda. Na przykład takie wyrażenie zwraca miasta w Niemczech:

  ```
  FILTER(Cities, Cities[Country] = "Germany")
  ```

- Funkcja TOPN, podobnie jak funkcja FILTER, zwraca część wierszy z tabeli. Zwracane są najwyższe albo najniższe wartości z wierszy tabeli na podstawie podanych kryteriów. Musisz podać liczbę wierszy, tabelę, z której mają zostać wzięte te wiersze, wartości, z którą porównywać każdy wiersz, i informację, czy chcesz by wyniki były posortowane od najwyższego do najniższego, czy na odwrót. Na przykład aby stworzyć tabelę z 15 klientami, którzy kupują najwięcej, można użyć takiego wyrażenia:

  ```
  TOPN(15, Customers, [Sales], DESC)
  ```

- Funkcja CROSSJOIN tworzy jedną tabelę z dwóch tabel podanych na wejściu. Funkcja ta zwraca tabelę, która po prostu zawiera wiersz dla każdego połączenia wierszy z tabel wejściowych. Pokazany poniżej przykład zwraca tabelę z wszystkimi zestawieniami produktów i miast, z wszystkimi kolumnami z tabel Cities i Products.

  ```
  CROSSJOIN(Cities, Products)
  ```

- Funkcja GENERATE, podobnie jak funkcja CROSSJOIN, zwraca tabelę, będącą połączeniem tabel podanych na wejściu. W tym przypadku drugim argumentem jest wyrażenie tabeli, które jest wykonywane dla każdego wiersza tabeli podanej jako pierwszy argument. Jeśli okaże się, że wyrażenie zwraca pustą tabelę dla określonego wiersza, ten wiersz nie zostanie uwzględniony w wyniku. Gdybyś jednak chciał, aby tak się stało, to możesz użyć w zamian funkcji GENERATEALL. Na przykład poniżej pokazane wyrażenie zwraca tabelę z miastami i produktami, które zostały sprzedane do klientów w tych miastach, z uwzględnieniem wszystkich kolumn z tabel Cities i Products.

```
GENERATE(Cities, FILTER(Products, [Sales] > 0))
```

W części II tej książki poznasz jeszcze kilka innych funkcji tablicowych, które omówimy przed ich użyciem.

Kontekst w funkcjach tablicowych

Pokazany powyżej przykład z funkcją GENERATE może być dla Ciebie jasny, jednak przyjrzyjmy mu się nieco bliżej, gdyż tak naprawdę to dość skomplikowany przykład. Aby zrozumieć, co robi takie wyrażenie, trzeba wiedzieć, jak działa kontekst DAX w funkcjach tablicowych. Posłużmy się przykładem funkcji GENERATE w formule miary:

```
AvgUnitAmount1 =
AVERAGEX(
    GENERATE(
        Cities,
        FILTER(
            Products,
            [Sales] > 10000
        )
    ),
    AVERAGE(fSales[UnitAmount])
)
```

Zadaniem tej miary jest obliczenie średniej liczby produktów sprzedanych w jednej transakcji (UnitAmount) dla wszystkich transakcji sprzedaży produktów w miastach, w których te produkty sprzedały się w ilości większej niż 10 000. Czy potrafisz znaleźć błędy w tej formule?

Gdy użyjemy tej miary w wizualizacji Power BI, to obliczenie zostanie wykonane w kontekście zapytania. Ten kontekst może być wszystkim i może zawierać jeden lub więcej filtrów na kolumnach w modelu Power BI.

Funkcja AVERAGEX pobiera dwa argumenty, a każdy z nich jest obliczany w różnych kontekstach.

- Pierwszy argument, wyrażenie tablicowe, jest wykonywany w tym samym kontekście co sama funkcja AVERAGEX.
- Drugi argument, wyrażenie skalarne, jest wykonywany w **kontekście wiersza** dla każdego wiersza tabeli z pierwszego argumentu.

Możliwe, że już zauważyłeś to w omawianej wcześniej mierze Sales2, która używa bezpośredniego odwołania do kolumn w drugim argumencie SUMX. Tutaj kontekst wiersza daje możliwość bezpośredniego użycia kolumn w tabeli. Tak naprawdę kontekst wiersza jest nad kontekstem zapytania. Gdy w kontekście wiersza w kolumnie obliczeniowej nie ma w ogóle filtrów, filtry z kontekstu zapytania wciąż są obecne.

Często popełniane błędy podczas pracy z wirtualnymi tabelami są związane z kontekstem wiersza w funkcjach agregujących tabelę. Oto prosty przykład:

```
ThisDoesntWork =
SUMX(
    VALUES(fSales[UnitAmount]),
    fSales[Tax]
)
```

Mimo że ta formuła używa dwóch kolumn z tabeli fSales, nie możemy się odwołać do kolumny Tax bezpośrednio, gdyż w tabeli fSales nie ma kontekstu wiersza, jest on w rezultacie zwróconym przez funkcję VALUES. Powyżej pokazana formuła zwraca tabelę tylko z jedną kolumną, fSales[UnitAmount]. Tylko ta kolumna może być użyta bezpośrednio.

Omówione wcześniej funkcje FILTER, TOPN oraz GENERATE działają w ten sam sposób — argument tablicowy jest wykonywany w kontekście, w którym jest wywoływana funkcja, a drugi argument — w kontekście wiersza. W przypadku funkcji GENERATE to oznacza, że mamy wyrażenie tablicowe wykonywane w kontekście wiersza.

Dla podanej wcześniej formuły AvgUnitAmount1 mamy cały stos kontekstów. Przeanalizujmy je krok po kroku.

1. Funkcja AVERAGEX jest wykonywana w kontekście zapytania.
2. Funkcja GENEARTE jest wykonywana w tym samym kontekście co funkcja AVERAGEX.
3. Tabela Cities wciąż jest obliczana w tym samym kontekście.
4. Funkcja FILTER jest wykonywana w kontekście wiersza w tabeli Cities.
5. Tabela Products jest obliczana w tym samym kontekście co funkcja FILTER.
6. Miara [Sales] — jako że jest odwołanie do innej miary, niejawna funkcja CALCULATE tworzy kontekst filtra. Przy każdym wywołaniu jesteśmy w jednym wierszu tabeli Cities i w jednym wierszu tabeli Products. W kontekście filtra dodawane są filtry na każdej kolumnie w tych dwóch tabelach. I tak w rezultacie otrzymujemy sprzedaż danego produktu w danym mieście.
7. Funkcja AVERAGE jest wykonywana w kontekście wiersza w tabeli zwróconej przez funkcję GENERATE, która jest podzbiorem zestawień miast i produktów.

Gdzie więc jest błąd w tej formule?

Jest w ostatnim kroku. Mimo że ten krok jest wykonywany dla poprawnych zestawień miast i produktów, jest wykonywany w kontekście wiersza. To oznacza, że tylko filtry już istniejące w kontekście zapytania mają wpływ na obliczenia funkcji `AVERAGE`. Bieżące miasto i produkt nie mają wpływu na obliczenia, gdyż nie ma żadnych (dodatkowych) filtrów w tabelach `Cities` i `Products`, by wybrać bieżące miasto i produkt. Aby rozwiązać ten problem, należy zmienić kontekst wiersza w kontekst filtra, dokładnie tak samo jak w kroku 6. Możemy to zrobić przez dodanie funkcji `CALCULATE`.

```
AvgUnitAmount2 =
AVERAGEX(
    GENERATE(
        Cities,
        FILTER(
            Products,
            [Sales] > 10000
        )
    ),
    CALCULATE(AVERAGE(fSales[UnitAmount]))
)
```

> W większości przypadków byłoby lepiej umieścić obliczenie średniej liczby sprzedanych sztuk w osobnej mierze, gdyż prawdopodobnie ta informacja będzie potrzebna jeszcze w innych miejscach. Wtedy musisz napisać logikę obliczenia tylko raz, a wywołanie tej miary spowoduje automatyczną zmianę kontekstu wiersza.

Podczas projektowania bardziej złożonych miar w języku DAX kluczowe jest staranne śledzenie kontekstu i jego zmian.

Jest jeszcze jeden błąd matematyczny w tej formule. Obliczamy średnią dla zestawień miasto/produkt ze średniej liczby sprzedanych sztuk. To niekoniecznie jest równe średniej liczbie sprzedanych sztuk ze wszystkich transakcji sprzedaży dla zestawień miast z produktami. Aby rozwiązać ten problem, musimy podejść do tego inaczej, ale najpierw skupmy się na wydajności.

Wydajność a funkcje tablicowe

Jest kilka kwestii, o których musimy pamiętać, gdy używamy tabel wirtualnych w DAX. Naszym ogólnym celem jest zapewnienie wyników tak szybko, jak to tylko możliwe, gdyż wydajność powinna być zawsze brana pod uwagę.

Przede wszystkim należy sobie zdać sprawę, że tabele wirtualne muszą być stworzone w pamięci przez silnik DAX. To oznacza, że im większa tabela wirtualna, tym więcej pamięci jest potrzebne i występuje większe ryzyko słabszej wydajności. Możesz nawet spotkać się z błędami wywołanymi przez **brak wystarczającej pamięci, by przeprowadzić dane obliczenia**. Z tego powodu powinieneś zadać sobie pytanie, czy używana przez Ciebie tabela może być mniejsza i czy potrzebujesz z niej wszystkich kolumn.

W pokazanej wyżej mierze AvgUnitAmount2 wyraźnie widać, że w tym tkwi problem. Tabela wirtualna stworzona przez funkcję GENERATE zawiera wszystkie kolumny zarówno z tabeli Cities, jak i Products. Ale na potrzeby obliczeń tylko unikatowe klucze są tak naprawdę potrzebne, gdyż określają, które wiersze w tabeli fSales są wybrane, i tym samym określają wartość miary Sales. Zoptymalizowana wersja miary AvgUnitAmount2 wygląda tak:

```
AvgUnitAmount3 =
AVERAGEX(
    GENERATE(
        VALUES(Cities[CityID]),
        FILTER(
            VALUES(Products[ProductID]),
            [Sales] > 10000
        )
    ),
    CALCULATE(AVERAGE(fSales[UnitAmount]))
)
```

Drugą kwestią, którą należy wziąć pod uwagę, jest liczba wierszy w tabeli wirtualnej. Oczywiście to również jest czynnik, który wpływa na całkowity rozmiar tabeli, a poza tym jest ściśle związany z liczbą iteracji w tabeli agregacji.

Na przykład jeśli cena zakupu produktu jest przechowywana w tabeli Products, to możesz obliczyć całkowitą kwotę zakupu na podstawie tabeli fSales.

```
TotalPurchased1 =
SUMX(
    fSales,
    fSales[UnitAmount] * RELATED(Products[PurchasePrice])
)
```

W zamian za to mógłbyś zoptymalizować liczbę iteracji i pozwolić funkcji SUMX przejść przez tabelę Products zamiast przez tabelę fSales. Albo nawet lepiej, funkcja SUMX mogłaby przejść tylko przez unikatowe wartości kolumny PurchasePrice.

```
TotalPurchased2 =
SUMX(
    VALUES(Products[PurchasePrice]),
    Products[PurchasePrice] * CALCULATE(SUM(fSales[UnitAmount]))
)
```

W tej wersji wszystkie transakcje odpowiadające wartości PurchasePrice są agregowane za jednym razem. Zwróć tutaj uwagę na funkcję CALCULATE, która zamienia kontekst wiersza na kontekst filtra i filtruje odpowiednie transakcje sprzedaży.

> Zużycie pamięci i liczba iteracji są powodem, dla którego funkcja CROSSJOIN nie jest często najlepszą funkcją do użycia w miarach DAX. Funkcja ta może zwrócić ogromną liczbę wierszy, co może szybko doprowadzić do problemów z pamięcią. Gdy dodajesz funkcję CROSSJOIN w tabeli agregacji, na przykład (A i B to dwa dowolne wyrażenia tablicowe):

```
SUMX(
    CROSSJOIN(A, B),
    [Sales]
)
```

to często wydajniej jest w ogóle nie używać tej funkcji:

```
SUMX(
    A,
    SUMX(
        B,
        [Sales]
    )
)
```

Jeszcze wydajniejszym podejściem jest unikanie iteracji i w zamian za to zastosowanie tabeli wirtualnych na potrzeby filtrowania. To jest temat następnego podrozdziału.

Filtrowanie za pomocą funkcji tablicowych

Jednym z najważniejszych aspektów pracy z DAX jest ścisłe połączenie między tabelami i filtrami. W tym podrozdziale dowiesz się, co to za połączenie i jak je wykorzystać.

Zastosowanie funkcji CALCULATETABLE

Jak już wspomnieliśmy wcześniej w tym rozdziale, wyrażenia tablicowe używane w funkcjach agregujących tabele, na przykład `SUMX`, są wykonywane w tym samym kontekście co sama funkcja agregująca. Nie zawsze potrzebujesz takiego działania, czasami wymagany jest inny kontekst. W języku DAX istnieje funkcja `CALCULATETABLE`, która służy dokładnie do takich celów.

Podobnie do funkcji `CALCULATE`, funkcja `CALCULATETABLE` zmienia kontekst przed obliczeniem wyrażenia. W funkcji `CALCULATE` to wyrażenie musi zwracać wartość skalarną, w przypadku funkcji `CALCULATETABLE` to musi być wyrażenie tablicowe. Poza tym obie funkcje wykonują te same cztery kroki:

1. Ustawienie kontekstu filtra.
2. Usunięcie istniejących filtrów z kolumn lub tabel, do których odwołują się argumenty filtra.
3. Dodanie nowych filtrów podanych w argumentach filtra.
4. Wykonanie wyrażenia tablicowego podanego w pierwszym argumencie.

Często funkcje `CALCULATE` i `CALCULATETABLE` mogą być używane zamiennie. Weźmy na przykład następującą formułę:

```
AveragePerCity_Canada1 =
AVERAGEX(
    CALCULATETABLE(
        VALUES(Cities[CityID]),
        Cities[Country] = "Canada"
    ),
    [Sales]
)
```

Możesz napisać tę miarę w postaci prostej formuły CALCULATE.

```
AveragePerCity_Canada2 =
CALCULATE(
    AVERAGEX(
        VALUES(Cities[CityID]),
        [Sales]
    ),
    Cities[Country] = "Canada"
)
```

Obie formuły zwracają średnią sprzedaż w poszczególnych miastach Kanady (oczywiście w zależności od kontekstu zapytania). Ale musisz wiedzieć, że między tymi dwiema formułami są techniczne różnice. W pierwszej formule filtr `Cities[Country] = "Canada"` jest stosowany do obliczeń tabeli `Cities`, natomiast w drugiej formule filtr jest stosowany zarówno do obliczeń tabeli `Cities`, jak i miary `Sales`. Choć w tym przypadku te różnice nie wpływają na wynik końcowy miary `Sales`, możesz kiedyś używać bardziej zaawansowanych miar, w których te różnice będą miały znaczenie.

> Podobnie do funkcji `CALCULATE`, funkcja `CALCULATETABLE` tworzy kontekst filtra. Gdy jest używana w kolumnie obliczeniowej, w każdym wierszu są dodawane nowe filtry, by wybrać ten wiersz. Gdy wykonywane są obliczenia w powiązanej tabeli w nowym kontekście, relacja ma filtry do propagowania, powiązana tabela zostanie przefiltrowana i będą brane pod uwagę tylko te wiersze, które są powiązane z bieżącym wierszem. To dlatego funkcja `RELATEDTABLE`, używana do pobrania powiązanej części z innej tabeli, jest po prostu funkcją `CALCULATE` bez argumentów filtra.

Za pomocą funkcji `CALCULATETABLE` możemy dodać filtry do obliczeń tabeli. Co ciekawe, możesz użyć również tabel, by dodawać filtry.

Filtry i tabele

Po omówieniu funkcji tablicowych nadszedł czas na to, by wrócić do filtrów i spojrzeć na nie na nowo. Wcześniej wprowadziliśmy filtry w kontekście zapytania i w kontekście filtra jako „reguły" dla kolumn w modelu Power BI, na przykład kolumna `Cities[Country]` musi być równa `France` lub `Germany`. Możesz patrzeć na tę regułę jako na dostarczanie wartości, które powinna zawierać kolumna `Country`, lub jeszcze z innej perspektywy — jak na tabelę z jedną kolumną i z dwoma wierszami: `France` i `Germany`.

W rzeczywistości właśnie tak działają filtry i funkcja CALCULATE, czyli przez dodanie tabel określających wybrane wartości w kolumnach i zastąpienie istniejących tabel, które wprowadzały filtr. Podstawowa zasada brzmi tak:

Każdy filtr to tabela, a każda tabela może być użyta jako filtr.

Ta zasada oznacza, że każdy prosty argument filtra w funkcji CALCUALTE może być napisany w postaci tabeli. Spójrzmy na przykład na następującą formułę:

```
SalesFranceGermany =
CALCULATE(
    [Sales],
    Cities[Country] IN {"France", "Germany"}
)
```

Argument filtra jest odpowiednikiem takiego wyrażenia tablicowego:

```
FILTER(
    ALL(Cities[Country]),
    Cities[Country] IN {"France", "Germany"}
)
```

Zatem dosłowne znaczenie tego filtra to: z wszystkich wartości kolumny Country wybierz tylko France i Germany.

To tłumaczy, dlaczego działa taka formuła jak ta poniżej.

```
CALCULATE(
    [Sales],
    Cities[Country] = "France"
    || Cities[Country] = "Germany"
)
```

Choć nie jest to najprostszy argument filtra, jest to po prostu odpowiednik następującej formuły:

```
FILTER(
    ALL(Cities[Country]),
    Cities[Country] = "France"
    || Cities[Country] = "Germany"
)
```

Większość funkcji DAX przedstawionych tutaj jako funkcje filtra jest tak naprawdę funkcjami tablicowymi, jak już pewnie to zauważyłeś w wyżej pokazanym wyrażeniu FILTER, które używa funkcji ALL jako funkcji tablicowej. Rzeczywiście funkcje ALL są funkcjami tablicowymi: ALL(Cities[Country]) to jednokolumnowa tabela zawierająca wszystkie unikatowe kraje, a ALL(Cities[Country], Cities[State]) to dwukolumnowa tabela zawierająca wszystkie unikatowe zestawienia krajów i stanów znalezionych w tabeli Cities.

> Nie wszystkie funkcje używane jako argument filtra są funkcjami tablicowymi. Funkcje `USERELATIONSHIP` i `CROSSFILTER` zmieniają sposób działania relacji, ale nie tworzą tabel. Funkcja `KEEPFILTERS` zmienia sposób działania funkcji `CALCULATE`, ale nie może zostać użyta do utworzenia tabeli. Funkcja `REMOVEFILTERS`, która działa jak funkcja `ALL` w argumencie filtra, również nie zwraca tabeli.

Nawet funkcje analizy czasowej są funkcjami tablicowymi, z wyłączeniem skróconych wersji, takich jak `TOTALYTD`. Każda z nich tworzy jednokolumnową tabelę zawierającą daty z danego okresu. To oznacza, że mógłbyś użyć tych funkcji w funkcjach agregujących tabele, na przykład w celu obliczenia średniej dziennej sprzedaży od początku roku do wybranego dnia.

```
AverageSalesPerDay_YTD =
AVERAGEX(
    DATESYTD('Date'[Date]),
    [Sales]
)
```

Ciekawszym zastosowaniem zasady *filtr jest tabelą* jest możliwość użycia dowolnej (wirtualnej) tabeli jako filtra w funkcji `CALCULATE`. Weźmy prosty przykład i załóżmy, że chcesz mieć miarę, która zwraca całkowitą sprzedaż w kraju lub krajach wybranych z tabeli `Cities`. Jeśli jesteś pewien, że w kontekście zapytania dla tego obliczenia kolumna `Country` jest przefiltrowana, to formuła nie jest trudna.

```
SalesWholeCountry1 =
CALCULATE(
    [Sales],
    ALLEXCEPT(Cities, Cities[Country])
)
```

To obliczenie usuwa wszystkie filtry z tabeli `Cities`, z wyjątkiem tego z kolumny `Country`. Zatem jeśli kontekst zapytania zawiera filtry *miasto to Atlanta* i *kraj to Stany Zjednoczone*, otrzymany w rezultacie kontekst filtra będzie miał tylko pozostawiony filtr *kraj to Stany Zjednoczone*.

Jednak jeśli kontekst zapytania ma na początku tylko filtr *miasto to Atlanta*, to mamy problem, gdyż usunięcie tego filtra daje nam cały świat! Aby to rozwiązać, musimy ponownie „włożyć" filtr *kraj to Stany Zjednoczone* do kontekstu — powinien on wywodzić się z filtra kolumny `City`. Można to zrobić za pomocą filtra w postaci tabeli:

```
SalesWholeCountry2 =
CALCULATE(
    [Sales],
    ALL(Cities),
    VALUES(Cities[Country])
)
```

Aby zobaczyć, co się tutaj dzieje, musimy zdać sobie sprawę, że *argumenty filtra są wykonywane w tym samym kontekście, co sama funkcja* `CALCULATE`. W kontekście zapytania, w którym tylko jedno miasto jest wybrane, wyrażenie `VALUES(Cities[Country])` zwraca jednokolumnową

tabelę z krajem, gdzie to miasto leży. Dokładnie takiego filtra potrzebujemy dla naszego obliczenia.

Kolejnym przykładem jest formuła pokazana poniżej, która oblicza kwotę sprzedaży dla 10 000 największych klientów.

```
SalesLargestCustomers1 =
CALCULATE(
    [Sales],
    TOPN(10000, ALL(Customers), [Sales], DESC)
)
```

Zauważ, że używamy tutaj `ALL(Customers)`, by upewnić się, że nie będziemy mieć filtrów w kontekście zapytania ograniczającego klientów, których chcemy wziąć pod uwagę. Funkcja `TOPN` zwraca 10 000 największych klientów (jako podzbiór tabeli `Customers`, wraz z wszystkimi kolumnami — możesz pominąć te, których nie potrzebujesz!). Następnie ta tabela jest używana jako filtr.

Ta formuła pokazuje wyraźnie, dlaczego stosowanie tabel jako filtrów powinno być wybierane zamiast agregacji tabel. Spójrz na alternatywę dla tej miary w postaci agregacji tabeli.

```
SalesLargestCustomers2 =
SUMX(
    TOPN(10000, ALL(Customers), [Sales], DESC),
    [Sales]
)
```

Teraz zapytaj siebie, ile razy miara `Sales` jest wywoływana w tym obliczeniu. To oczywiście zależy od liczby klientów w tabeli `Customers`. Załóżmy, że mamy 60 000 klientów. Funkcja `TOPN` musi wywołać miarę `Sales` dla każdego klienta, by ustalić, czy należy on do grupy największych, czy nie. Potem miara `SalesLargestCustomers1` musi wywołać jeszcze jeden raz miarę `Sales`. Tabela zwrócona przez funkcję `TOPN` jest zastosowana jako filtr i tworzy w ten sposób kontekst dla największych klientów. Jednak miara `SalesLargestCustomers2` przechodzi przez tabelę zwróconą przez funkcję `TOPN` i wywołuje miarę `Sales` dla każdego wiersza. Innymi słowy, ta miara wywołuje miarę `Sales` w sumie `70000` razy, podczas gdy miara `SalesLargestCustomers1` wywołuje ją tylko `60001` razy! Silnik DAX może zoptymalizować działanie tych miar, ale różnica będzie znacząca.

Stosowanie tabel jako filtrów daje jeszcze więcej możliwości, gdy zdasz sobie sprawę, że w przeciwieństwie do filtrów w kontekście zapytania tabele jako filtry mogą mieć wiele kolumn. To oznacza, że teraz mamy rozwiązanie dla naszej problematycznej miary `AvgUnitAmount3`, omawianej wcześniej w tym rozdziale.

```
AvgUnitAmount4 =
CALCULATE(
    AVERAGE(fSales[UnitAmount]),
    GENERATE(
        VALUES(Cities[CityID]),
        FILTER(
            VALUES(Products[ProductID]),
            [Sales] > 10000
```

```
        )
    )
)
```

Zamiast średniej średnich, która była efektem agregacji tabeli z wyrażeniem `GENERATE`, tym razem używamy funkcji `GENERATE`, by zapewnić filtr wybierający zestawienia miast i produktów, które sprzedały się w więcej niż 10 000 sztuk. Następnie obliczamy średnią liczbę sprzedanych produktów w jednej transakcji dla wszystkich transakcji sprzedaży, które są powiązane z wybranym zestawieniem miasto-produkt. Teraz nie tylko mamy poprawne obliczenie średniej, ale również wyeliminowaliśmy iterację tabeli `GENERATE`, co pomoże nam w dalszej poprawie wydajności tej miary.

Zastosowanie TREATAS

Jest jedno istotne ograniczenie zastosowania tabel jako filtrów — tabele muszą skutecznie filtrować tabele, na których wykonywane są obliczenia. Zwykłe dodanie jako filtra tabeli, która nie ma nic wspólnego z resztą modelu, nic nie da.

Na przykład ta formuła nie zwraca sprzedaży dla Wielkiej Brytanii:

```
UKSales_wrong =
CALCULATE(
    [Sales],
    ROW("Country", "United Kingdom")
)
```

Dlaczego to nie działa? Model Power BI nie ma możliwości określenia sprzedaży w Wielkiej Brytanii z utworzoną za pomocą funkcji `ROW` tabelą, która ma kolumnę o nazwie `Country`, gdyż powinna być przefiltrowana tabela `Cities` zawierająca kolumnę `Country`. Mógłbyś pomyśleć, że skoro nazwa jest taka sama, silnik DAX mógłby założyć, że ta formuła właśnie to ma zrobić, ale prowadziłoby to do nieprzewidywalnych rezultatów w modelach z tą samą nazwą kolumny w wielu różnych tabelach.

Aby można użyć takiej tabeli jako filtra, silnik DAX powinien być w stanie określić, że wirtualna tabela jest połączona z tabelą lub jakimiś kolumnami z modelu. To połączenie nazywamy **pochodzeniem** (ang. *lineage*) i w skrócie oznacza, że gdy tworzymy wirtualną tabelę, DAX zapisuje, jakie były oryginalne kolumny i skąd pochodziły. Spójrzmy ponownie na wyrażenie `GENERATE`.

```
GENERATE(
    VALUES(Cities[CityID]),
    FILTER(
        VALUES(Products[ProductID]),
        [Sales] > 10000
    )
)
```

Wyraźnie widać, że kolumna `Cities[CityID]` jest w modelu, a funkcja `VALUES` pobiera z niej unikatowe wartości. Tabela zwracana przez wyrażenie `VALUES(Cities[CityID])` pochodzi od `Cities[CityID]`. W ten sam sposób tabela zwracana przez wyrażenie `VALUES(Products[ProductID])`

pochodzi od ProductID z tabeli Products. Funkcja GENERATE tworzy tabelę zawierającą zestawienia wartości zwróconych przez dwa wyrażenia VALUES, więc każda kolumna w tabeli wyjściowej pochodzi od odpowiadającej jej kolumny z modelu.

Większość funkcji tablicowych zachowuje pochodzenie kolumn. Niektóre z nich jednak pozwalają na tworzenie tabel w dziwny sposób, który może stanowić problem, jeśli chodzi o pochodzenie. Na przykład funkcja UNION pozwala na łączenie tabel przez wzięcie wierszy z dwóch różnych tabel źródłowych, które mogą mieć niezgodne ze sobą pochodzenie. W takim przypadku kolumny w tabeli wynikowej nie będą mieć powiązania z żadną z istniejących kolumn w modelu.

Niekiedy chcesz, by tabela wirtualna miała pochodzenie inne niż domyślne. Język DAX oferuje rozwiązanie w postaci funkcji TREATAS, która zmusza model, by wziął pod uwagę szczególne pochodzenie kolumn w tabeli.

TREATAS jest kolejnym przykładem funkcji, która jest używana jedynie w funkcji CALCULATE lub CALCULATETABLE jako argumenty filtra. Poniżej pokazana formuła poprawnie oblicza sprzedaż w Wielkiej Brytanii (choć są łatwiejsze sposoby, aby to osiągnąć).

```
UKSales_correct =
CALCULATE(
    [Sales],
    TREATAS(
          ROW("Country", "United Kingdom"),
          Cities[Country]
    )
)
```

Zauważ, że TREATAS nie wymaga, aby nazwa kolumny była taka sama. Możemy nazwać kolumnę dowolnie w wyrażeniu ROW. Funkcja TREATAS działa również z tabelami, które mają wiele kolumn. W takim przypadku musisz podać kolumnę modelu dla każdej kolumny w tabeli. W rozdziale 9. „Interesy między różnymi oddziałami jednej firmy" zobaczysz przykłady, które pomogą Ci to lepiej zrozumieć.

Zmienne w języku DAX

Dzięki funkcjom tablicowym DAX i filtrowaniu możemy wykonywać bardzo złożone obliczenia. Jednak jest i druga strona medalu, gdyż formuły mogą się rozrastać. Co ważniejsze, gdy w grę wchodzą różne konteksty, otrzymanie poprawnych rezultatów może stanowić problem.

Zmienne DAX sprawiają, że pisanie zaawansowanego kodu DAX jest łatwiejsze. Sama nazwa jest dość dziwna, gdyż przeznaczeniem zmiennych DAX jest obliczenie czegoś raz, a następnie użycie tej wartości w innych okolicznościach (zwykle w innym kontekście) bez konieczności ponownego obliczania zmiennej. Innymi słowy, zmienna DAX jest używana jak stała!

Zmienną deklarujemy za pomocą słowa kluczowego VAR. Możemy zadeklarować wiele zmiennych, a deklaracja jednej zmiennej może korzystać z wartości innej, wcześniej zadeklarowanej zmiennej. Deklaracje zmiennych są zakończone słowem kluczowym RETURN.

```
VAR ThisValue = 5
RETURN
...
```

Warto wiedzieć, że zmienne DAX mogą być używane w dowolnym wyrażeniu w formule DAX. Zmienna może zawierać wartość skalarną, ale może być też tabelą. Taka (dość absurdalna) formuła jest poprawnym kodem DAX:

```
VariableTest =
VAR Variable1 = 3
VAR Variable2 = Variable1 + 5
RETURN
CALCULATE(
    VAR Variable3 = MAX(fSales[UnitAmount])
    RETURN
    SUM(fSales[Tax]) + Variable2 + Variable3,
    VAR Variable4 = 4
    VAR TableVariable =
          FILTER(ALL(fSales[UnitAmount]),fSales[UnitAmount] = Variable4)
    RETURN
    TableVariable
)
```

Miejsce deklaracji zmiennej w formule wyznacza kontekst, w którym zmienna jest obliczana. Na przykład tutaj zmienna Variable3 jest obliczana w kontekście filtra utworzonego przez zastosowanie filtra w postaci tabeli TableVariable w pierwotnym kontekście zapytania dla tej miary. Gdyby zmienna Variable3 była zadeklarowana zaraz po Variable2, byłaby obliczana w kontekście zapytania. Zauważ, że zmienne Variable4 i TableVariable są używane w argumencie filtra funkcji CALCULATE i obie są obliczane w pierwotnym kontekście zapytania.

Każda zmienna ma swój własny zasięg, co oznacza, że nie może być użyta poza wyrażeniem, w którym została zadeklarowana. W podanej wyżej formule zmienna Variable3 nie może być użyta w pierwszym argumencie funkcji CALCULATE. Zmienne Variable1 i Variable2 są częścią wyrażenia całej formuły, więc mogą być wszędzie użyte.

Zmienne DAX mogą uprościć przepływ obliczeń oraz poprawić czytelność Twojej formuły po prostu przez użycie jasnych nazw zmiennych. Spójrzmy jeszcze raz na miarę AvgUnitAmount4.

```
AvgUnitAmount4 =
CALCULATE(
    AVERAGE(fSales[UnitAmount]),
    GENERATE(
        VALUES(Cities[CityID]),
        FILTER(
            VALUES(Products[ProductID]),
            [Sales] > 10000
        )
    )
)
```

Tę formułę można uprościć za pomocą zmiennych.

```
AvgUnitAmount5 =
VAR LargeCityProductCombinations =
    GENERATE(
        VALUES(Cities[CityID]),
        FILTER(
            VALUES(Products[ProductID]),
            [Sales] > 10000
        )
    )
RETURN

CALCULATE(
    AVERAGE(fSales[UnitAmount]),
    LargeCityProductCombinations
)
```

Pamiętaj że zmienne DAX to stałe! Taka wersja tej formuły nie jest poprawna:

```
AvgUnitAmount_wrong =
VAR LargeProducts =
    FILTER(
            VALUES(Products[ProductID]),
            [Sales] > 10000
    )
VAR LargeCityProductCombinations =
    GENERATE(
        VALUES(Cities[CityID]),
        LargeProducts
    )
RETURN

CALCULATE(
    AVERAGE(fSales[UnitAmount]),
    LargeCityProductCombinations
)
```

Przez umieszczenie wyrażenia `FILTER` w zmiennej zamieniliśmy je w stałą tabelę. Jednak w funkcji `GENERATE` chcemy, by lista produktów była ustalana na nowo dla każdego miasta.

Podsumowanie

W tym rozdziale poznałeś kontekst wiersza, zapytania oraz filtra i dowiedziałeś się, jaką odgrywają rolę w formułach DAX. Powiedzieliśmy, jak można zmienić kontekst za pomocą funkcji `CALCULATE`, która usuwa filtry i dodaje nowe do istniejącego kontekstu. Ponadto omówiliśmy funkcje analizy czasowej, które zapewniają filtry specjalnie zaprojektowane pod kalendarz gregoriański.

Następnie skupiliśmy się na funkcjach tablicowych DAX, które dają nam możliwość agregacji wartości tabel i użycia specjalnie stworzonych tabel wirtualnych w formułach DAX.

Zastosowanie tabel wirtualnych daje nam całe morze możliwości poza tym, co oferują nam „standardowe" funkcje DAX i filtrowanie. Mówiliśmy o ścisłym połączeniu między tabelami i filtrami, co pozwala nam na użycie tabeli jako filtra. I w końcu, omówiliśmy zmienne DAX, które ułatwiają implementację złożonej logiki w DAX i sprawiają, że kod jest dużo bardziej czytelny.

Potrzebujesz tych wszystkich podstawowych koncepcji, by móc odkryć bardziej zaawansowaną analizę z DAX. Po tym ważnym rozdziale przejdziemy do drugiej części tej książki, która skupia się na zastosowaniu w praktyce wszystkich omówionych wcześniej koncepcji w oparciu o prawdziwe przykłady z życia. Chcielibyśmy, abyś po przejściu przez te wszystkie przykłady jeszcze bardziej docenił i zrozumiał moc języka DAX i abyś próbował odpowiadać na pytania ze strony przedstawicieli firmy, posiłkując się obliczeniami DAX.

Rozdział 5. „Bezpieczeństwo z DAX" jest poświęcony bezpieczeństwu modelu Power BI. Jak sam się przekonasz, znajomość języka DAX, kontekstu i filtrowania ma wiele zastosowań, gdy przychodzi do projektowania zabezpieczeń.

Poznaj prawdziwy potencjał języka DAX w analizie danych!

Microsoft Power BI jest doskonałym narzędziem do profesjonalnej analizy danych. Jeśli jednak chcesz uzyskać za jego pomocą naprawdę spektakularne efekty, musisz się biegle posługiwać językiem DAX (Data Analysis Expressions). Pozwala on na wykonywanie zaawansowanych obliczeń i zapytań dotyczących danych w powiązanych tabelach i kolumnach w tabelarycznych modelach danych.

To książka przeznaczona dla analityków biznesowych, którzy już poznali język DAX, chcą jednak skorzystać z pełnego potencjału formuł tego języka i modeli Power BI, by tworzyć wydajne i zaawansowane analizy danych. Opisano w niej zasady analizy biznesowej i reguły projektowania dobrych modeli. Zaprezentowano też praktyczne przykłady użycia języka DAX w rzeczywistych sytuacjach biznesowych. Pokazano niuanse pracy z modelami Power BI, a także z funkcjami DAX, filtrami i miarami. Nie zabrakło bardzo przydatnych wskazówek dotyczących błędów popełnianych często podczas tworzenia zaawansowanych agregacji danych. Do książki zostały dołączone materiały do pobrania (pliki PBIX), które ułatwią pełne zrozumienie prezentowanych treści i ich stosowanie we własnej praktyce zawodowej.

Najciekawsze zagadnienia:

- koncepcje modelowania danych i struktur
- modele Power BI a modele systemów zarządzania relacyjnymi bazami danych
- bezpieczne poziomy agregacji, atrybuty i hierarchie
- koncepcja kontekstu i jej stosowanie
- standardowa analiza czasowa
- inteligentna ocena inwestycji za pomocą finansowych funkcji DAX

Michiel Rozema

jest jednym z najlepszych znawców Power BI na świecie. Przez osiem lat, jako pracownik firmy Microsoft, zajmował się wprowadzaniem Power BI w Holandii. Poświęcił mu wiele wystąpień na branżowych konferencjach. W 2019 roku otrzymał tytuł Microsoft MVP.

Henk Vlootman

jest doświadczonym konsultantem do spraw Power Platform, Power BI i Excela. W 2019 roku otrzymał tytuł Microsoft MVP. Jest współzałożycielem niderlandzkiej grupy użytkowników Power BI.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-9659-3

9 788328 396593

Cena: 89,00 zł

Packt›