Wydanie II

Microsoft Power BI dla zaawansowanych

Eksperckie techniki tworzenia interaktywnych analiz w świecie biznesu

Greg Deckler
Brett Powell

Helion

packt

Tytuł oryginału: Mastering Microsoft Power BI: Expert techniques to create interactive insights for effective data analytics and business intelligence, 2nd Edition

Tłumaczenie: Zbigniew Waśko

ISBN: 978-83-8322-445-9

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/mipoz2
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 3
Łączenie się ze źródłami i przekształcanie danych za pomocą języka M

W tym rozdziale, po przygotowaniu środowiska i źródła danych opisanym w rozdziale 2. „Przygotowanie źródeł danych", przechodzimy do implementacji zapytań **Power Query** (**M**) w nowym pliku Power BI Desktop w celu pobrania wymaganych tabel faktów i wymiarów. Zapytania Power Query są zwykle pisane ręcznie w języku transformacji danych zwanym potocznie M, ale mogą też być generowane przy użyciu interfejsu o nazwie **Power Query Editor** (**Edytor Power Query**). Zapytania te uzyskują dostęp do źródeł danych i opcjonalnie stosują logikę transformacyjną, aby przygotować tabele dla modelu danych w Power BI.

Wyrażenia Power Query (M) stają się wszechobecne w całej microsoftowej platformie przetwarzania danych. Są używane z przepływami danych, które mogą być stosowane wielokrotnie przez liczne zestawy danych w Power BI. Wyrażenia są również obsługiwane przez **Azure Data Factory** (**ADF**), co oznacza, że procesy łączenia danych rozpoczęte w Power BI, mogą być w razie potrzeby skalowane. Wreszcie zapytania M stanowią podstawę przepływów danych w **Dataverse**, operacyjnym magazynie danych utrzymywanym przez firmę Microsoft.

Język M zawiera setki funkcji, a na temat niego samego i jego zastosowań napisano wiele książek. Głównym celem tego rozdziału jest ułatwienie zrozumienia zapytań M w kontekście korporacyjnego rozwiązania Power BI, które w głównej mierze wykorzystuje zarządzaną przez IT hurtownię danych.

W omawianych przykładach podstawowym źródłem danych dla zapytań Power Query (M) są system SQL Server i pliki Excela. Przez dobór odpowiednich parametrów i zmiennych uzyskujemy dostęp do zestawu widoków SQL odzwierciedlających tabele hurtowni

danych w systemie SQL Server oraz dane rocznego planu sprzedaży zawarte w skoroszycie Excela.

Za pomocą dodatkowych zapytań tworzymy relacje między planem sprzedaży i tabelami wymiarów, a także zwiększamy użyteczność zbioru danych i ułatwiamy zarządzanie nim. Przedstawiamy też przykłady przekształcania danych i logiki zapytań, np. przez utworzenie dynamicznej kolumny historii klienta. Na koniec omawiamy kwerendy wielokrotnego użytku zwane przepływami danych, a także narzędzia do edycji i zarządzania kwerendami, takie jak rozszerzenia dla **Visual Studio** i **Visual Studio Code**.

W tym rozdziale omawiamy następujące zagadnienia:

- rodzaje zapytań Power Query (M),
- tworzenie zapytań Power Query (M),
- przykłady zapytań Power Query (M),
- przepływy danych,
- narzędzia edycji zapytań Power Query (M).

Zacznijmy od przyjrzenia się różnym typom zapytań M.

Rodzaje zapytań Power Query M

W rozdziale 2. „Przygotowanie źródeł danych" utworzyliśmy widoki SQL, skonfigurowaliśmy źródła danych i ustawiliśmy opcje środowiska Power BI Desktop. Po wykonaniu tych zadań projektant zbioru danych może wreszcie przystąpić do opracowania zapytań pobierających dane i ustalenia parametrów zbioru danych. Kwerendy Power Query (M) są środkiem, za pomocą którego zbiory danych w Power BI łączą się ze źródłami i importują dane. Kwerendy M są niezbędne do łączenia się ze źródłami, takimi jak widoki SQL, a także mogą przekształcać dane zgodnie z wymaganiami.

Język M stosowany w Power Query jest językiem programowania nazywanym bardziej formalnie językiem formuł Power Query. M zawiera ponad 700 funkcji służących do łączenia się z danymi i wykonywania transformacji tych danych. Kod, który nawiązuje połączenie z danymi i przekształca je, nazywany jest zapytaniem lub kwerendą.

Istnieje wiele różnych typów zapytań, które służą różnym celom:

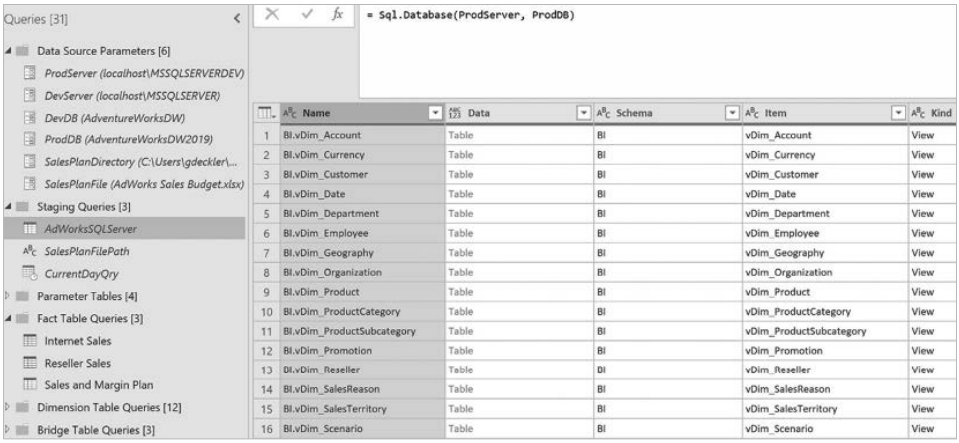
- zapytania o parametry źródła danych,
- kwerendy przejściowe,
- zapytania o fakty i wymiary,
- zapytania do tabeli parametrów,

- zapytania do tabeli zabezpieczeń,
- niestandardowe kwerendy funkcyjne.

W tym rozdziale szczegółowo omówimy wszystkie typy, jednak przed zagłębieniem się w te rozważania należałoby powiedzieć kilka słów na temat organizowania kwerend w celu uzyskania rozwiązań łatwiejszych do zrozumienia i możliwych do utrzymania przez dłuższy czas.

Organizacja zapytań

W **Edytorze Power Query** działającym w ramach aplikacji Power BI Desktop foldery grupowe służą do organizowania zapytań w określone kategorie, takie jak `Data Source Parameters` (parametry źródła danych), `Staging Queries` (kwerendy przejściowe), `Parameter Tables` (tabele parametrów), `Fact Table Queries` (zapytania tabeli faktów), `Dimension Table Queries` (zapytania tabeli wymiarów) i `Bridge Table Queries` (zapytania tabeli pomostowej), jak w przykładzie z rysunku 3.1.



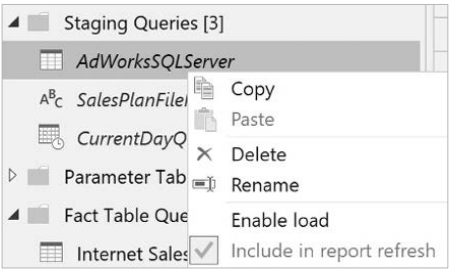
Rysunek 3.1. Edytor Power Query w Power BI Desktop z folderami grupowymi

Aby utworzyć nową grupę, należy kliknąć prawym przyciskiem myszy zapytanie w panelu *Queries* (*Zapytania*), a następnie wybrać kolejno polecenia *Move to Group* (*Przenieś do grupy*) i *New Group* (*Nowa grupa*). Po utworzeniu grup zapytania można przenosić pomiędzy nimi metodą przeciągania i upuszczania lub przez kliknięcie prawym przyciskiem myszy na zapytaniu, wybranie polecenia *Move to Group* (*Przenieś do grupy*), a następnie wskazanie grupy docelowej.

Parametry i zapytania zapisane kursywą są uwzględniane w procesie odświeżania zbioru danych, ale nie są ładowane do zbioru danych w ramach Power BI Desktop. Na przykład zapytanie `AdWorksSQLServer` zaznaczone na rysunku 3.1 jedynie eksponuje obiekty

bazy danych SQL Server jako wynik działania funkcji `Sql.Database()` M, aby inne zapytania mogły się do nich odwoływać. Zapytanie to, wraz z parametrami źródła danych, jest zapisane kursywą i służy do usprawnienia procesu pobierania danych w taki sposób, że pojedyncza zmiana może być uwzględniana w aktualizacji wielu zapytań zależnych.

Kliknij prawym przyciskiem myszy zapytanie lub parametr w panelu *Queries* (*Zapytania*), aby otworzyć menu z opcjami *Enable load* (*Włącz ładowanie*) i *Include in report refresh* (*Uwzględnij w odświeżeniu raportu*), jak na rysunku 3.2.



Rysunek 3.2. Włączenie ładowania i uwzględniania w odświeżaniu raportu

Dla wielu zbiorów danych jedynymi zapytaniami, które powinny zostać załadowane do modelu, są te dotyczące tabel wymiarów i faktów oraz niektóre dotyczące tabeli parametrów. Ponadto może się zdarzyć, że dane w systemie źródłowym będą statyczne czyli niezmienne. W takich przypadkach nie ma powodu, aby odświeżać dane w zbiorze po pierwszym załadowaniu, dlatego opcję *Include in report refresh* (*Uwzględnij w odświeżeniu raportu*) można wyłączyć, aby zaoszczędzić zasoby systemowe i skrócić czas odświeżania.

Przyjrzymy się teraz szczegółowo różnym typom zapytań M, zaczynając od parametrów źródła danych.

Parametry źródła danych

Na rysunku 3.1 w poprzednim punkcie pokazano grupę o nazwie `Data Source Parameters` zawierającą parametry źródła danych. Parametry to specjalne zapytania, które nie uzyskują dostępu do zewnętrznego źródła danych i zwracają jedynie konkretną wartość skalarną, taką jak data, liczba lub ciąg znaków tekstowych.

Podstawowym zastosowaniem parametrów jest zdefiniowanie jednego wspólnego elementu, takiego jak nazwa serwera lub bazy danych, a następnie odwołanie się do niego w wielu innych zapytaniach. Podobnie jak zmienne globalne, tak i parametry usprawniają zarządzanie dużymi zbiorami danych, ponieważ projektant takiego zbioru w razie potrzeby zmiany tego elementu może to po prostu zrobić w jednym miejscu (przez modyfikację stosownego parametru), a nie w każdym zapytaniu oddzielnie.

Poza tym twórcy zbiorów danych Power BI mogą używać parametrów, aby załadować do lokalnego pliku w Power BI Desktop tylko próbkę danych z tabeli źródłowej, którą w całości załadują później do opublikowanego już zbioru danych Power BI. Można na przykład utworzyć parametry daty początkowej i końcowej, a następnie wstawić je do wyrażenia ustalającego warunek filtrowania ładowanej tabeli.

Lokalny plik aplikacji Power BI Desktop może używać parametrów reprezentujących tylko pojedynczy miesiąc lub określony zakres lat, ale opublikowany zbiór danych może załadować wiele lat w oparciu o zmienione wartości parametrów daty początkowej i końcowej. Autor zbioru danych może manipulować wartościami parametrów ręcznie w usłudze Power BI lub za pomocą skryptu modyfikującego parametry za pośrednictwem Power BI REST API. Interfejs Power BI REST jest opisany w rozdziale 10. „Zarządzanie obszarami roboczymi i zawartością", w punkcie „Power BI REST API i moduł PowerShell".

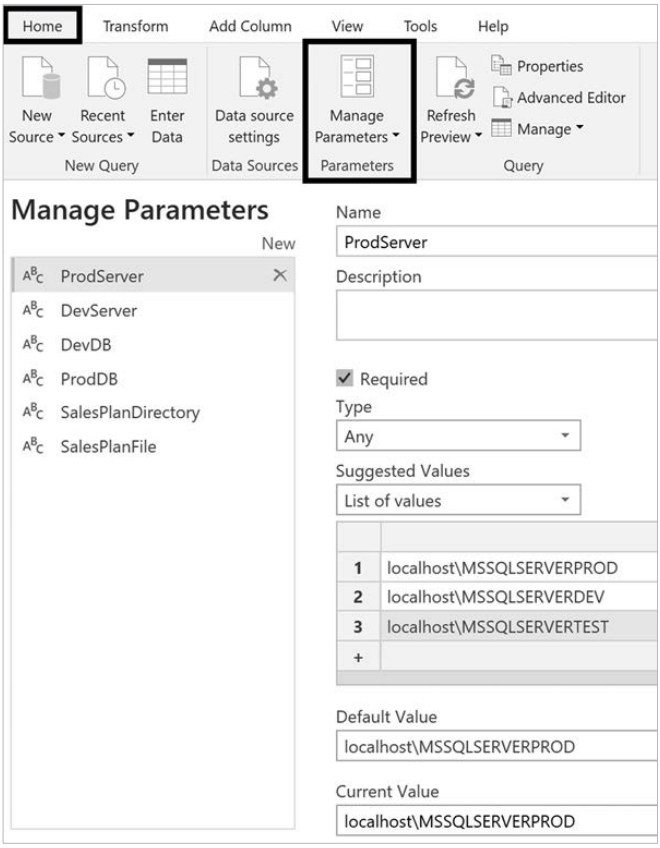
Parametry zapytań można tworzyć i modyfikować za pomocą okna dialogowego *Manage Parameters* (*Zarządzaj parametrami*) dostępnego na karcie *Home* (*Narzędzia główne*) **Edytora Power Query**. Na rysunku 3.3 przedstawiono to okno z sześcioma parametrami zdefiniowanymi dla bazy danych SQL Server i skoroszytu Microsoft Excel.

Dla tego zbioru danych parametry bazy danych środowiska programistycznego i produkcyjnego (na przykład `ProdServer` i `ProdDB`) mają skonfigurowaną listę możliwych wartości, co ułatwia pracę i pozwala uniknąć błędów przy przełączaniu źródeł danych. W tym samym celu zarówno nazwa skoroszytu Excela zawierającego roczny plan sprzedaży i marży, jak i katalog jego plików są również zapisane jako parametry.

Lista rozwijana *Suggested Values* (*Sugerowane wartości*) pozwala na ręczne wprowadzenie dowolnej wartości, wybór wartości z zakodowanej listy lub użycie zapytania zwracającego stosowną listę (typ wartości w języku M, taki jak tabela czy rekord). Ze względu na niewielką liczbę nazw serwerów w tym przykładzie oraz rzadkość zmiany nazw serwerów produkcyjnych i deweloperskich wprowadzono ręcznie trzy sugerowane wartości.

Parametry są często używane z plikami **szablonów Power BI**, aby umożliwić użytkownikom biznesowym modyfikowanie własnych raportów za pomocą predefiniowanych oraz wstępnie przefiltrowanych zapytań i miar. Na przykład użytkownik otwiera szablon i wybiera konkretny dział, a ten wybór jest wykorzystywany do filtrowania zapytań w zbiorze danych.

Poza tym parametry mogą być przydatne podczas definiowania wartości używanych w filtrach zapytań (np. daty początkowe i końcowe) oraz w logice obliczeniowej używanej do tworzenia niestandardowych kolumn w zapytaniach. Parametry są zwykle wykorzystywane tylko przez kwerendy i dlatego nie są ładowane (kursywa na rysunku 3.3),



Rysunek 3.3. Zarządzanie parametrami w Edytorze Power Query

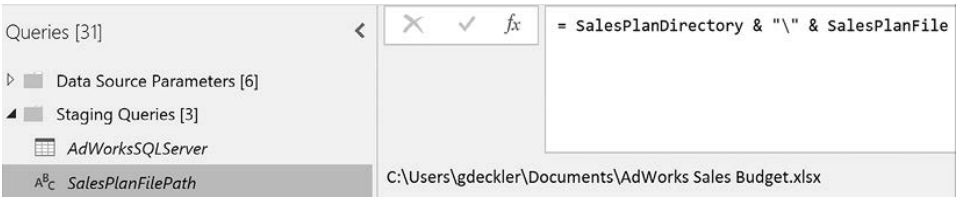
choć mogą być ładowane do modelu danych jako pojedyncze tabele z pojedynczą kolumną i pojedynczym wierszem. Po załadowaniu parametry mogą być dostępne dla wyrażeń DAX, tak jak inne tabele w modelu.

Teraz zajmiemy się kwerendami przejściowymi.

Kwerendy przejściowe

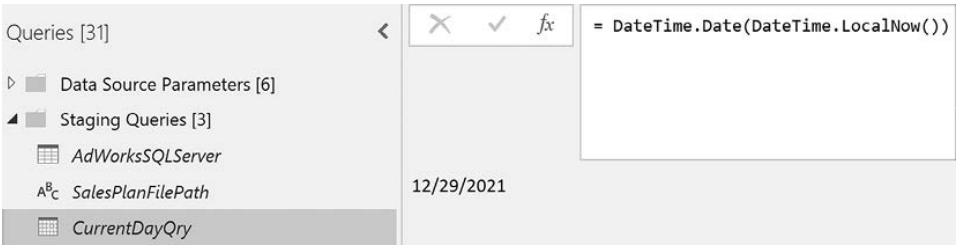
Po skonfigurowaniu parametrów źródła danych tworzy się zapytania przejściowe (ang. *staging queries*) w celu wyeksponowania źródła danych dla zapytań dotyczących tabel wymiarów i faktów w zbiorze danych. Na przykład zapytanie przejściowe `AdWorksSQLServer` przekazuje jedynie wartości parametrów określające serwer produkcyjny i produkcyjną bazę danych do funkcji `Sql.Database()`, tak jak na rysunku 3.1. Wynikiem tego zapytania jest tabela zawierająca schematy i obiekty przechowywane w bazie danych, w tym widoki SQL z tabelami faktów i wymiarów.

Kwerenda przejściowa `SalesPlanFilePath` używana w odniesieniu do źródła `Annual Sales Plan` (roczny plan sprzedaży) w Excelu jest bardzo podobna, gdyż tak samo odwołuje się jedynie do sparametryzowanych nazw pliku i katalogu w celu utworzenia pełnej ścieżki, tak jak na rysunku 3.4.



Rysunek 3.4. Kwerenda przejściowa rocznego planu sprzedaży — skoroszyt Excela

Trzecie i ostatnie zapytanie przejściowe, `CurrentDateQry`, po prostu wyznacza bieżącą datę, co pokazano na rysunku 3.5.



Rysunek 3.5. Kwerenda przejściowa bieżącej daty

Podobnie jak parametry, wyniki kwerend przejściowych, takich jak `CurrentDateQry`, mogą być przywoływane przez inne kwerendy, np. w warunku filtrowania tabeli faktów. W poniższym przykładzie funkcja `Table.SelectRows()` (wybierz wiersze) ma za zadanie pobrać tylko te wiersze z tabeli faktów, w których kolumna `Order Date` (data zamówienia) zawiera wartości mniejsze lub równe temu, co zwraca kwerenda `CurrentDateQry` (np. 12/29/2021).

```
let
    Source = AdWorksSQLServer,
    ISales = Source{[Schema = "BI", Item = "vFact_InternetSales"]}[Data],
    ↪CurrentDateFilter = Table.SelectRows(ISales, each   [Order Date]
    ↪<= CurrentDateQry)
in
    CurrentDateFilter
```

W tym prostym przypadku ten sam warunek filtrujący może być łatwo wbudowany w widok SQL (`vFact_ InternetSales`) zawierający tabelę faktów i takie podejście jest na ogół preferowane. Należy jednak zauważyć, że silnik M jest w stanie przekształcić

zmienną zapytania (`CurrentDateFilter`) wraz z odwołaniem do zapytania przejściowego (`CurrentDateQry`) w pojedynczą instrukcję SQL przez **składanie zapytań**.

W niektórych scenariuszach transformacji danych, szczególnie przy szybkich iteracjach i cyklach życia projektów w podejściu zwinnym, wskazane może być przynajmniej tymczasowe wykorzystanie wydajnych zapytań M w ramach zbioru danych Power BI (lub modelu Analysis Services) zamiast wprowadzania zmian w źródle danych (na przykład w tabelach lub widokach hurtowni danych).

Jak zostało to opisane w podrozdziale „Składanie zapytań" rozdziału 2. „Przygotowanie źródeł danych", jeśli jest konieczne użycie języka M do implementacji transformacji zapytań lub logiki, projektant zbioru danych powinien upewnić się, że ta logika jest składana w instrukcję SQL i w ten sposób wykonywana przez system źródłowy. Jest to szczególnie ważne w przypadku dużych zapytań pobierających miliony wierszy przy ograniczonych zasobach serwera bramy danych (jeśli jest stosowany) lub przydzielonych mocy przerobowych (sprzętowych) w Power BI Premium.

Kwerendy przejściowe mogą być również używane z zapytaniami DirectQuery.

Przejściowość w DirectQuery

Możliwe jest użycie zapytań przejściowych nawet podczas pracy w trybie DirectQuery. Przejściowe zapytanie bazodanowe dla zbioru danych w trybie DirectQuery jest nieco inne niż dla zbioru danych w trybie importu. W przypadku takiego zapytania do wyrażenia `let` dodawana jest nowa zmienna, jak pokazano w poniższym przykładzie:

```
let
    Source = Sql.Database(ProdServer, ProdDB),
    DummyVariable = null
in
    Source
```

Dodatkowa zmienna, `DummyVariable`, jest pomijana przez zapytanie, a sama funkcja, `Sql.Database()`, która odwołuje się do parametrów serwera i bazy danych ze zbioru danych w trybie importu, może być również użyta w odniesieniu do zbioru danych DirectQuery.

Po zdefiniowaniu parametrów i zapytań przejściowych możemy skupić się na głównych zapytaniach o dane dla naszych faktów i wymiarów.

Zapytania o fakty i wymiary

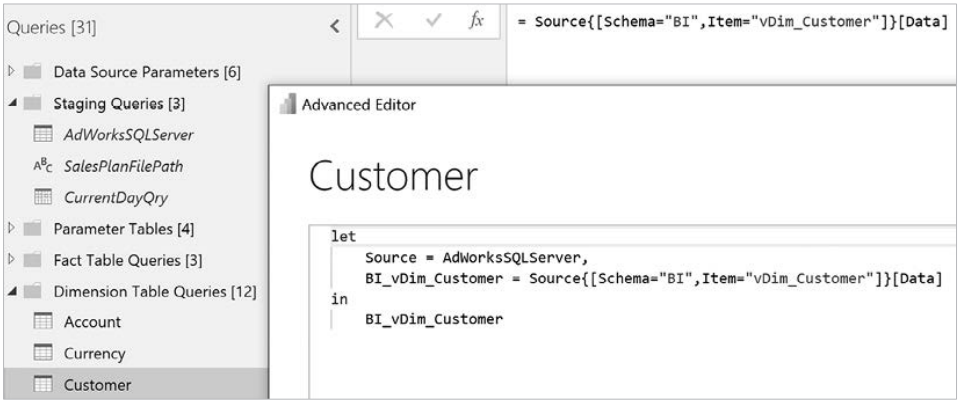
Cała dotychczasowa praca była tak naprawdę tylko przygotowaniem do opracowania zapytań łączących się ze źródłem i importujących interesujące nas dane dla naszych faktów i wymiarów.

Dla zbiorów danych w trybie importu zapytanie jest wykonywane podczas zaplanowanego odświeżenia, a wyniki są ładowane do skompresowanego formatu kolumnowego. Zbiory danych w trybie DirectQuery i zapytania importowe z wyłączoną właściwością *Enable load* (*Włącz ładowanie*) (zob. punkt „Organizacja zapytań") definiują jedynie instrukcję SQL reprezentującą określone tabele wymiarów lub faktów. Źródło danych DirectQuery wykorzystuje tego typu instrukcje do tworzenia zapytań SQL niezbędnych do wykonania zapytań raportowych, takich jak połączenie zapytania `Internet Sales` (sprzedaż internetowa) z zapytaniem `Product` (produkt).

Po odpowiednim przygotowaniu większość zapytań w języku M powinna po prostu uzyskać dostęp do pojedynczego widoku SQL, zastosować niewielkie lub żadne transformacje, a następnie przekazać wyniki do zbioru danych jako tabelę wymiarów lub faktów. Takie zapytania są określane jako zapytania źródłowe wyłącznie referencyjne (ang. *source reference-only queries*).

Tylko odniesienie do źródła

Zapytanie w języku M przedstawione na rysunku 3.6 odwołuje się do widoku SQL (`BI.vDim_Customer`) poprzez zapytanie przejściowe (`AdWorksSQLServer`) i nie wprowadza żadnych dalszych transformacji:



Rysunek 3.6. Zapytanie o wymiar klienta

Jak pokazano na rysunku 3.6, zapytanie `Customer` (klient) uzyskuje dostęp do unikatowego rekordu związanego ze schematem (`BI`) i widokiem SQL (`vDim_Customer`) tabeli wytworzonej przez zapytanie przejściowe (`AdWorksSQLServer`). Rekord ten zawiera wszystkie nazwy pól tabeli zapytania przejściowego, w tym pole `Data` (dane) przechowujące widok SQL. Odwołanie się do pola `Data` we wspomnianym rekordzie powoduje pobranie wyników widoku SQL.

Ponieważ nie są stosowane żadne transformacje, zapytanie odzwierciedla źródłowy widok SQL, a zmiany w tym widoku, takie jak usunięcie kolumny, są automatycznie przenoszone do zbioru danych Power BI przy następnym odświeżeniu. Relacja jeden do jednego pomiędzy widokiem SQL a zapytaniem jest jednym z głównych powodów, dla których warto zaimplementować lub przenieść logikę transformacji danych do źródła hurtowni danych, a nie do zbioru danych Power BI.

Połączenie z bazami hurtowni danych, takimi jak Azure SQL Database lub Azure SQL Managed Instance, powinno zazwyczaj skutkować prostymi wyrażeniami zapytań w języku M z niewielką liczbą wymaganych transformacji. Źródła częściowo ustrukturyzowane i nieustrukturyzowane, takie jak pliki JSON i Excela, wymagają oczywiście więcej transformacji w celu przygotowania danych do analizy. Następnie przyjrzymy się kwerendzie o większej złożoności spowodowanej nawiązaniem połączenia z nieustrukturyzowanym źródłem danych, jakim jest Microsoft Excel.

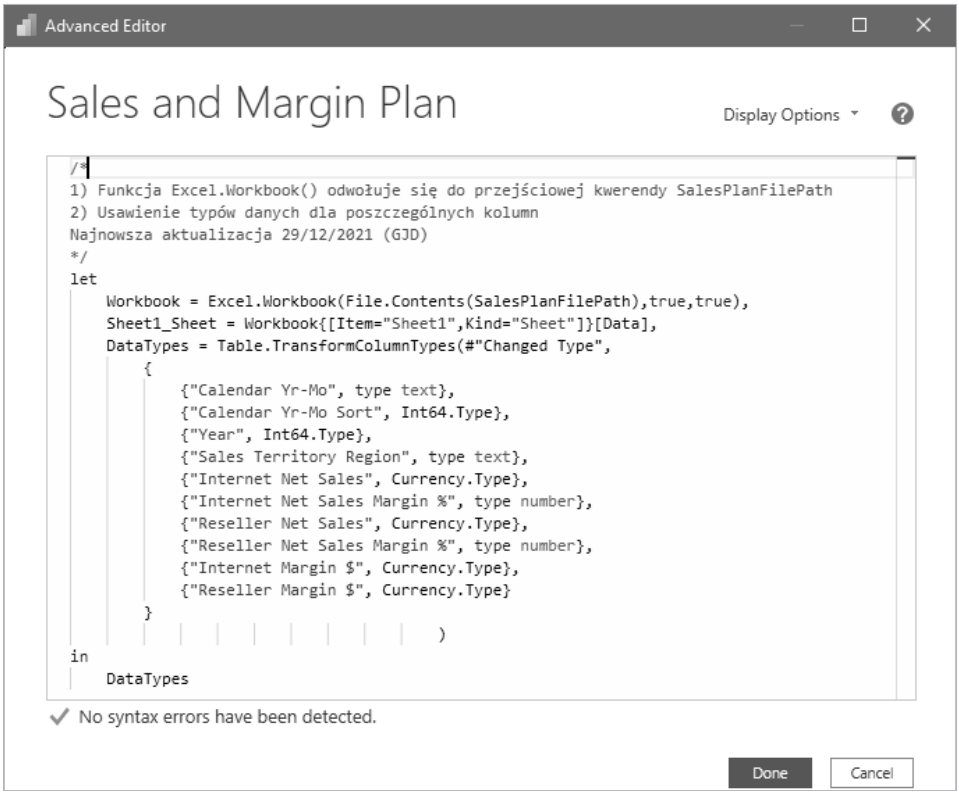
Skoroszyt Excela — roczny plan sprzedaży

Jak wykazaliśmy w poprzednim punkcie, zapytania o fakty i wymiary dla ustrukturyzowanych źródeł danych, takich jak bazy SQL Server, powinny zasadniczo mieć charakter wyłącznie referencyjny. Jednak w przypadku zbioru danych w trybie importu dane dotyczące rocznego planu sprzedaży i marży są pobierane z obiektu tabelarycznego należącego do skoroszytu Excela, czyli do nieustrukturyzowanego źródła danych.

W kwerendzie tabeli faktów pokazanej na rysunku 3.7, `Sales and Margin Plan` (plan sprzedaży i marży), funkcja udostępniająca dane, `Excel.Workbook()`, wywołuje zapytanie przejściowe `SalesPlanFilePath` (ścieżka pliku z planem sprzedaży).

Zgodnie z tym, co napisaliśmy w rozdziale 2. „Przygotowanie źródeł danych" (punkt „Opcje programu Power BI Desktop"), opcja automatycznego wykrywania typów danych dla źródeł nieustrukturyzowanych powinna być wyłączona. Źródła danych ustrukturyzowane, takie jak SQL Server, jawnie określają typy danych dla kolumn. W przypadku źródeł nieustrukturyzowanych funkcja automatycznego wykrywania typu danych próbuje określić odpowiedni typ dla każdej kolumny.

Przy wyłączonym automatycznym wykrywaniu typów danych konieczne jest jawne określenie odpowiedniego typu danych dla każdej kolumny tabeli Excela za pomocą funkcji `Table.TransformColumnTypes()`. Argumenty `Int64.Type`, `Currency.Type` i `type number` użyte w tej funkcji odpowiadają typom danych *Whole Number* (*Liczba całkowita*), *Fixed Decimal Number* (*Stałoprzecinkowa liczba dziesiętna*) i *Decimal Number* (*Liczba dziesiętna*).



Rysunek 3.7. Zapytanie Sales and Margin Plan dla skoroszytu Excela

W przypadku zbioru danych w trybie DirectQuery dane dotyczące planu sprzedaży i marży byłyby pobierane z widoku SQL w ramach tej samej bazy danych, z której pobierane są inne tabele faktów i wymiarów, jak pokazano na rysunku 3.8.

Rysunek 3.8. Zapytanie Sales and Margin Plan dla zbioru danych w trybie DirectQuery

Koszt i czas integracji danych z kwerendy `Sales and Margin Plan` z bazą hurtowni danych są jednym z powodów, dla których w tym projekcie wybrano dla zbioru danych domyślny trybu importu. Ograniczenie pojedynczej bazy danych w ramach jednego źródła danych

jest obecnie jednym z podstawowych czynników ograniczających zbiory danych w trybie DirectQuery. Na rysunku 3.9 widoczny jest błąd zgłoszony przy próbie połączenia się w trybie DirectQuery z dwoma bazami danych funkcjonującymi na tym samym serwerze.

Rysunek 3.9. Ograniczenie trybu DirectQuery — jedna baza danych

To ograniczenie DirectQuery można ominąć, używając złożonych modeli danych, ale wtedy wzrasta złożoność, która jest na ogół niepożądana.

A teraz przyjrzyjmy się kwerendom przeznaczonym do tworzenia tabel wspomagających budowanie relacji w obrębie modelu danych.

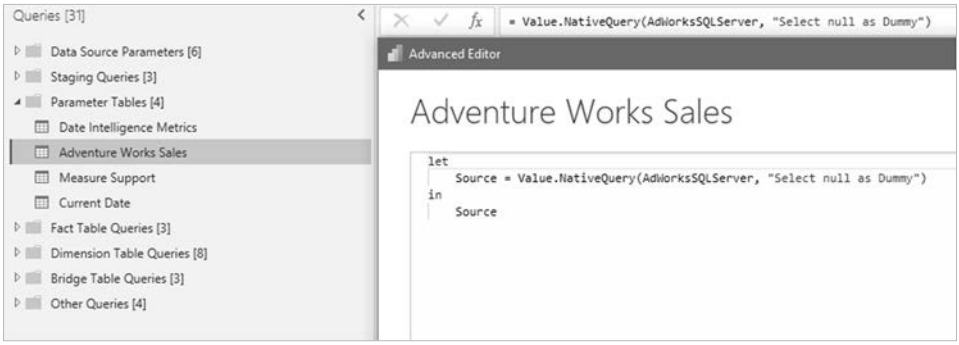
Kwerendy tabel parametrów

Zapytania tabel parametrów tworzy się dla celów użyteczności i zarządzania.

Z punktu widzenia użyteczności kwerendy `Data Intelligence Metrics` (metryki analizy danych) i `Adventure Works Sales` (sprzedaż w Adventure Works) służą do konsolidacji powiązanych miar na liście pól i udostępniania ich autorom raportów oraz analitykom, aby ci nie musieli przewijać całych tabel w poszukiwaniu potrzebnych miar. Kwerenda `Current Date` (bieżąca data) dostarcza raportom wiadomość tekstową informującą o dacie ostatniej operacji odświeżenia danych. Z punktu widzenia zarządzania zapytanie `Measure Support` (obsługa miar) służy do centralizacji pośrednich lub rozgałęzionych wyrażeń DAX, z których korzysta wiele miar DAX.

Jak widać na rysunku 3.10, w przypadku trzech z czterech zapytań zastosowano wyrażenie trywialne, ponieważ chodzi wyłącznie o przekazanie nazwy tabeli do modelu danych.

Kwerendy `Date Intelligence Metrics`, `Adventure Works Sales` i `Measure Support` pobierają pustą wartość, a właściwość *Include in report refresh* (*Uwzględnij w odświeżeniu raportu*) mają wyłączoną. W rozdziale 4. „Projektowanie modeli danych w trybach importu, DirectQuery i złożonym" pokazujemy, jak te puste tabele można wykorzystać w systemie metadanych modelu danych, a w rozdziale 5. „Tworzenie miar DAX i ról zabezpieczeń" omawiamy dodawanie miar DAX do zbioru danych.



Rysunek 3.10. Kwerenda Adventure Works Sales z grupy Parameter Tables

Zapytanie `Current Date` jest jedynym zapytaniem tabeli parametrów wykonywanym przy każdym odświeżeniu raportu. Poniższy skrypt M dla zapytania `Current Date` tworzy tabelę z jedną kolumną i jednym rekordem reprezentującym datę w momencie wykonania.

```
let
    RefreshDateTime = DateTime.LocalNow(),
    TimeZoneOffset = -5,
    RefreshDateTimeAdjusted = RefreshDateTime +
#duration(0,TimeZoneOffset,0,0),
    RefreshDateAdjusted = DateTime.Date(RefreshDateTimeAdjusted),
    TableCreate = Table.FromRecords({[CurrentDate = RefreshDateAdjusted]}),
    DateType = Table.TransformColumnTypes(TableCreate,{"CurrentDate", type
    ↪date})
in
    DateType
```

Wszystkie raportowane czasy w Microsoft Azure są wyrażone w **uniwersalnym czasie koordynowanym** (**UTC**). Aby więc mieć pewność, że ostatni odświeżony komunikat o dacie odzwierciedla lokalną strefę czasową, należy w zapytaniu zawrzeć stosowną logikę.

W powyższym przykładzie wartość zwracana przez funkcję `DateTime.LocalNow()` jest pomniejszana o pięć godzin, co odpowiada różnicy pomiędzy czasami wschodnioamerykańskim i UTC. Ze zmodyfikowanej w ten sposób wartości wyodrębniono datę i zapisano ją w utworzonej właśnie w tym celu tabeli.

Jak wynika z rysunku 3.11, zapytanie `Current Date` jest wykorzystywane przez miarę DAX do informowania o ostatnio zaktualizowanej dacie.

Obliczenie bieżącej daty i godziny w zapytaniu Power BI przechwytuje datę i godzinę w momencie odświeżenia i może być potencjalnie wykorzystane jako zapytanie pomocnicze dla innych zapytań w obrębie zbioru danych.

Rysunek 3.11. Tabele parametrów w liście Data (Dane) i komunikat o odświeżeniu danych

Wyrażenie DAX `Current Date` po prostu umieszcza te informacje w raporcie i jednocześnie wprowadza dodatkowy kontekst dla przeglądających raport.

Wyrażenie DAX obsługujące ostatni zaktualizowany komunikat jest następujące:

```
Last Refresh Msg =
   VAR __CurrentDateValue = MAX("Bieżąca data"[CurrentDate])
RETURN
   "Last Refreshed: " & __CurrentDateValue
```

Dodatkowy przykład wykorzystania języka DAX do zwracania wartości łańcuchowej na potrzeby tytułu lub etykiety znajduje się w rozdziale 6. „Planowanie raportów Power BI" (podrozdział „Strony uściślające raport").

W miarę jak zbiory danych stają się coraz większe i bardziej złożone, można starać się lepiej zorganizować zapytania M przez wprowadzanie nowych grup lub zmienianie nazw grup istniejących. Na przykład grupa czterech zapytań prezentowana w tym rozdziale spełnia trzy różne funkcje (lista pól danych, data ostatniego odświeżenia i centralizacja logiki DAX).

Dla doświadczonych programistów Power BI i Analysis Services Tabular tabela parametrów jest po prostu niestandardową tabelą wartości parametrycznych załadowaną do modelu i dostępną dla interfejsu raportowania. Miary DAX powinny być tak zaprojektowane, aby wykrywały, która wartość (parametr) została wybrana przez użytkownika (na przykład 10% wzrostu, 20% wzrostu), i dynamicznie obliczały wynik. W takim zbiorze danych koncepcja tabeli parametrów ulega rozszerzeniu na każde zapytanie załadowane do modelu danych, ale niezwiązane z żadną inną tabelą w tym modelu.

Większość dużych zbiorów danych Power BI zawiera parametry źródła danych, kwerendy przejściowe, kwerendy faktów i wymiarów, kwerendy tabel relacji oraz kwerendy tabel parametrów. Teraz omówimy dwa opcjonalne typy kwerend: kwerendy tabel zabezpieczeń i kwerendy funkcji niestandardowych.

Kwerendy tabel zabezpieczeń

Kwerendy tabel bezpieczeństwa ułatwiają wdrażanie **zabezpieczeń na poziomie wierszy** (**RLS**) dla zbioru danych. Takie zapytania mogą być wymagane w sytuacji, gdy każdy użytkownik powinien widzieć wyłącznie swoje dane. Wówczas tabela zabezpieczeń może importować **nazwy główne użytkowników** (**UPN**) z dostępem do raportów tworzonych na podstawie zbioru danych. Zazwyczaj UPN-y pokrywają się z adresami e-mailowymi użytkowników i za pomocą funkcji DAX `USERPRINCIPALNAME()` można je pobierać w celu ustanowienia zabezpieczeń lub wprowadzenia innych filtrów w usłudze Power BI.

W przypadku projektu opisanego w rozdziale 1. „Planowanie projektów Power BI" zabezpieczenie danych przez ustawienie roli RLS nie wymaga pobierania żadnych tabel.

Zgodnie z tym, co powiedzieliśmy w rozdziale 1. „Planowanie projektów Power BI" (w punkcie „Przykładowy szablon projektu Power BI"), kierownicy sprzedaży i ich współpracownicy powinni mieć dostęp tylko do swoich grup terytorialnych (`Sales Territory`), natomiast wiceprezesi powinni mieć dostęp globalny.

Przy takich prostych wymaganiach można utworzyć grupy bezpieczeństwa użytkowników (na przykład Ameryka Północna, Europa, region Pacyfiku) i przypisać je do odpowiednich ról RLS ustanowionych w modelu danych. Szczegółowe informacje na temat implementacji tych ról znajdują się w rozdziale 5. „Tworzenie miar DAX i ról zabezpieczeń".

W projektach o bardziej złożonych lub drobiazgowych wymaganiach bezpieczeństwa często konieczne jest załadowanie do modelu danych dodatkowych tabel, takich jak `Users` (użytkownicy) i `Permissions` (uprawnienia). Na przykład, jeśli na użytkowników miałyby być nałożone ograniczenia dotyczące kodów pocztowych, a nie terytorialnych grup sprzedaży, lepsze byłoby podejście dynamiczne oparte na tabelach i filtrach uwzględniających użytkownika żądającego raportu niż tworzenie (i utrzymywanie) dużej ilości różnych ról RLS i grup bezpieczeństwa.

Ze względu na znaczenie zabezpieczeń dynamicznych (opartych na użytkownikach), szczególnie w przypadku dużych zbiorów danych, w rozdziale 5. „Tworzenie miar DAX i ról zabezpieczeń" omówimy szczegółowe przykłady implementacji takich zabezpieczeń zarówno dla zbiorów danych importowanych, jak i w trybie DirectQuery.

Teraz przyjrzymy się kolejnemu opcjonalnemu typowi zapytań, czyli zapytaniom funkcyjnym.

Niestandardowe zapytania funkcyjne

Ponieważ język zapytań w Power Query (M) jest funkcyjnym językiem programowania, możliwe jest tworzenie niestandardowych funkcji w formie zapytań. Kwerendy te umożliwiają tworzenie kodu wielokrotnego użytku, który można stosować przy wykonywaniu powtarzających się transformacji danych, takich jak niestandardowa analiza składniowa lub obliczenia. Oto prosty przykład niestandardowej funkcji do obliczania wieku klientów w tabeli `Customer` (klient):

```
let CalculateAge = (BirthDate as date) =>
        Date.Year(CurrentDayQuery) - Date.Year(BirthDate)
in CalculateAge
```

Ta niestandardowa funkcja przyjmuje jeden parametr, `BirthDate` (data urodzenia), który musi być typu data. Funkcja `Date.Year` (data.rok) jest używana zarówno w odniesieniu do zapytania `CurrentDayQuery` (data bieżąca), jak i parametru `BirthDate`, a różnica tych dwóch wyników zwraca liczbę lat. Całość można zapisać jako zapytanie o nazwie `GetAge` (oblicz wiek) i używać go w ramach niestandardowej formuły dla kolumny w tabeli `Customer` w następujący sposób:

```
= GetAge([BirthDate])
```

Dodatkowe przykłady i zastosowania funkcji niestandardowych można znaleźć w oficjalnej dokumentacji firmy Microsoft pod adresem *https://bit.ly/33VJfRz*.

Na tym kończymy omawianie rodzajów zapytań Power Query (M), które można tworzyć podczas opracowywania zbioru danych. Teraz zajmiemy się bardziej szczegółowo samymi zapytaniami pisanymi w języku M.

Tworzenie zapytań w języku M

Jak wspomniano, język M jest funkcyjnym językiem programowania, który zawiera ponad 700 funkcji. Podobnie jak inne języki programowania, M ma swoją specyficzną składnię, strukturę, operatory i typy danych, których należy używać podczas kodowania. Doświadczeni programiści Power Query (M) bardzo dobrze znają listy, rekordy i tabele oraz funkcje dostosowane do pracy z tymi specyficznymi typami.

Pełny opis języka M wykracza poza zakres tej książki, ale w kolejnych punktach poruszymy wiele ważnych tematów dotyczących tego języka, jak również podamy przykłady bardziej złożonych zapytań. Czytelników zainteresowanych lepszym poznaniem języka M odsyłamy do jego oficjalnej dokumentacji, którą można znaleźć pod adresem *https://bit.ly/3vmFSyr*.

Na początek przyjrzyjmy się numerycznym typom danych w M.

Numeryczne typy danych

W przypadku strukturalnych źródeł danych, takich jak SQL Server, typy danych kolumn źródłowych określają typy danych stosowane w Power BI. Na przykład typ danych waluty lub pieniędzy w SQL Server skutkuje typem danych *Fixed Decimal Number* (*Stałoprzecinkowa liczba dziesiętna*) w Power BI. Podobnie liczby całkowite w SQL Server wymuszają typ danych *Whole Number* (*Liczba całkowita*), a liczby dziesiętne w SQL Server przekładają się na typ *Decimal Number* (*Liczba dziesiętna*) w Power BI.

Gdy zapytanie jest ładowane do modelu danych w zbiorze Power BI, typ numeryczny w SQL Server (np. 19,4) staje się typem *Fixed Decimal Number* (*Stałoprzecinkowa liczba dziesiętna*). Dzięki czterem cyfrom po przecinku typ ten pozwala uniknąć błędów zaokrąglania. Typ *Decimal Number* (*Liczba dziesiętna*) jest odpowiednikiem zmiennoprzecinkowego lub przybliżonego typu danych z ograniczeniem do 15 cyfr znaczących.

Ze względu na możliwość wystąpienia błędów zaokrąglenia w przypadku danych typu *Decimal Number* (*Liczba dziesiętna*) oraz przewagę wydajnościową typu *Fixed Decimal Number* (*Stałoprzecinkowa liczba dziesiętna*), zaleca się przechowywanie liczb ułamkowych jako *Fixed Decimal Number* (*Stałoprzecinkowa liczba dziesiętna*), jeśli tylko wystarczająca jest precyzja czterocyfrowa. Wszystkie kolumny liczbowe typu całkowitego powinny być przechowywane w Power BI jako typy *Whole Number* (*Liczba całkowita*).

Kolumnom numerycznym w zapytaniach M można przypisywać typ danych *Whole Number* (*Liczba całkowita*), *Fixed Decimal Number* (*Stałoprzecinkowa liczba dziesiętna*) lub *Decimal Number* (*Liczba dziesiętna*) za pomocą następujących wyrażeń, odpowiednio: `Int64.Type`, `Currency.Type` i `type number`. Do konwersji typów danych służy funkcja `Table.TransformColumnTypes()` — w poniższym przykładzie dotyczy to kolumn `Discount Amount` (wielkość rabatu), `Sales Amount` (wielkość sprzedaży) i `Extended Amount` (wartość):

```
let
    Source = AdWorksSQLServer,
    Sales = Source{[Schema = "BI", Item = "vFact_InternetSales"]}[Data],
    TypeChanges = Table.TransformColumnTypes(Sales,
            {
                {"Discount Amount", Int64.Type}, // liczba całkowita
                {"Sales Amount", Currency.Type}, // stałoprzecinkowa liczba
                                                 // dziesiętna
                {"Extended Amount", type number} // liczba dziesiętna
            })
in
    TypeChanges
```

Ponieważ język M rozróżnia wielkość liter, wyrażenia określające typ danych muszą być wprowadzane z zachowaniem dokładnej pisowni, np. `type number`, a nie `Type Number`. Zauważ, że w zapytaniach M mogą być zawarte komentarze jedno- i wielowierszowe.

Więcej szczegółów na ten temat podajemy w podrozdziale „Przykłady zapytań w języku Power Query M".

Ze względu na wydajność i możliwość wystąpienia błędów zaokrąglenia ważne jest sprawdzenie, czy poszczególnym kolumnom numerycznym zostały przypisane właściwe typy danych. Dodatkowe szczegóły dotyczące typów danych znajdują się w rozdziale 4. „Projektowanie modeli danych w trybach importu, DirectQuery i złożonym".

Teraz przyjrzymy się metodom dostępu do elementów (rekordów) przy użyciu języka M.

Dostęp do poszczególnych danych w języku M

Dostęp do rekordów w tabelach, elementów w listach i wartości w rekordach ma fundamentalne znaczenie w formułowaniu zapytań kierowanych do źródeł danych. W poniższym przykładzie wyniki widoku SQL `BI.vDim_Account` są przekazywane do Power BI przy użyciu nieco innej składni kwerendy M niż w przypadku zapytania o wymiar `Customer` z poprzedniego podrozdziału.

```
let
    Source = AdWorksSQLServer,
    AccountRecord = Source{[Name = "BI.vDim_Account"]},
    Account = AccountRecord[Data]
in
    Account
```

W tym zapytaniu rekord jest pobierany z zapytania przejściowego `AdWorksSQLServer` tylko na podstawie kolumny `Name` (nazwa). Pole `Data` (dane) tego rekordu jest następnie przenoszone do osobnej zmiennej (`Account`) i za jej pomocą przekazywane do Power BI jako rezultat wykonania widoku `BI.vDim_Account`. Projektanci zbiorów danych mogą wybrać metodę dostępu do elementów zawartych w zapytaniu przejściowym.

Poniższy przykładowy kod pobiera łańcuch `"Cherry"` z listy zdefiniowanej za pomocą instrukcji języka M:

```
let
    Source = {"Apple","Banana","Cherry","Dates"},
    ItemFromList = Source{2}
in
    ItemFromList
```

W języku M numeracje rozpoczynają się od zera, więc wyrażenie `Source{0}` zwróciłoby wartość `"Apple"`, a `Source{4}` zwróciłoby błąd, ponieważ na liście są tylko cztery pozycje. To samo dotyczy również wyodrębniania znaków z łańcucha tekstowego. Na przykład wyrażenie `Text.Range("Brett",2,2)` zwraca znaki `et`.

Listą w języku M jest uporządkowana sekwencja wartości. Istnieje wiele funkcji służących do analizowania i przekształcania list, np. `List.Count()` (liczebność listy) czy `List.Distinct()` (bez duplikatów). Funkcje agregujące wartości zawarte w listach, takie

jak `List.Average()` (wartość średnia), są często używane w zapytaniach grupujących, które wywołują funkcję `Table.Group()` (grupowanie wierszy tabeli). Pełną listę funkcji języka M czytelnik znajdzie w artykule „Dokumentacja funkcji języka Power Query M" opublikowanym pod adresem *https://learn.microsoft.com/pl-pl/powerquery-m/power-query-m-function-reference*.

Następnie przyjrzymy się być może jednemu z najważniejszych aspektów języka M, jakim jest składnia zapytań.

Przykłady zapytań w języku Power Query M

Jak wynika z dotychczasowych przykładów, połączenie dojrzałej hurtowni danych i warstwy widoków SQL w ramach jednego źródła może wyeliminować potrzebę dalszego przekształcania danych. Projektanci zbiorów danych Power BI powinni mimo wszystko znać podstawy formułowania zapytań w języku M i najczęstsze przypadki ich stosowania, ponieważ często konieczne jest dalsze rozszerzanie i wzbogacanie danych źródłowych.

W następnych punktach zademonstrujemy trzy typowe scenariusze przekształcania danych, które można zrealizować za pomocą języka M. Zapytania M nie tylko pobierają poprawne wyniki, ale też generują instrukcje SQL przeznaczone do wykonania przez system źródłowy (mechanizm składania zapytań), a komentarze są dołączane dla celów długoterminowej konserwacji.

Jeśli jeszcze nie tworzyłeś zapytań w języku M, możesz zacząć od utworzenia zapytania pustego w kategorii *Other* (*Inne*) łączników ze źródłami danych dostępnych w oknie dialogowym *Get Data* (*Pobierz dane*).

Można też powielić istniejące zapytanie za pomocą polecenia wybranego w menu kontekstowym po kliknięciu prawym przyciskiem jednego z istniejących zapytań w **Edytorze Power Query**. Potem można zmienić nazwę duplikatu i zmodyfikować jego treść.

Filtrowanie poprzednich trzech lat

Celem tego przykładu jest pobranie dat z trzech lat poprzedzających rok bieżący i wszystkich dat z roku bieżącego aż do dnia, w którym pytanie jest formułowane. Na przykład zapytanie sformułowane 30 grudnia 2021 roku powinno pobrać daty od 1 stycznia 2018 roku do 30 grudnia 2021 roku. Wymóg ten zapewni, że trzy pełne lata danych historycznych plus rok bieżący będą zawsze dostępne dla opracowywanych raportów.

Daty początkowa i końcowa dla warunku filtra są obliczane za pomocą funkcji języka M `Date` i `DateTime`, a następnie są przypisywane do zmiennych, odpowiednio: `StartDate` (data początkowa) i `CurrentDate` (data bieżąca). Ponieważ data początkowa wypada zawsze 1 stycznia, potrzebne jest tylko wyznaczenie roku początkowego i przekazanie tej wartości do konstruktora `#date`.

Na koniec dwie wspomniane zmienne są przekazywane do funkcji `Table.SelectRows()` (wybierz wiersze) w celu nałożenia filtra na widok tabeli faktów `Reseller Sales` (sprzedaż przez pośredników):

```
let
// Daty z trzech poprzednich lat
    CurrentDate = DateTime.Date(DateTime.LocalNow(),
    StartYear = Date.Year(CurrentDate)-3,
    StartDate = #date(StartYear,1,1),
// Widok tabeli faktów Reseller Sales
    Source = AdWorksSQLServer,
    ResellerSales = Source{[Schema = "BI", Item = "vFact_ResellerSales"]}
    ↪[Data],
// Filtrowanie danych z trzech poprzednich lat
    FilterResellerSales = Table.SelectRows(ResellerSales,
    each Date.From([OrderDate]) >= StartDate and Date.From([OrderDate])
    ↪<= CurrentDate)
in
    FilterResellerSales
```

Jak widać w oknie dialogowym polecenia *View Native Query* (*Wyświetl zapytanie natywne*) dostępnym w menu kontekstowym panelu *Applied Steps* (*Zastosowane kroki*) **Edytora Power Query**, niestandardowy warunek filtra jest tłumaczony na instrukcję w języku T-SQL do wykonania w źródłowej bazie danych SQL Server:

```
from [BI].[vFact_ResellerSales] as [_]
where [_].[OrderDate] >= convert(datetime2, '2018-01-01 00:00:00') and
[_].[OrderDate] < convert(datetime2, '2021-12-31 00:00:00')
```

Warto zauważyć, że kolejność zmiennych w wyrażeniu nie ma wpływu na końcowe zapytanie. Na przykład dwie zmienne widoku `Reseller Sales` mogą być określone przed trzema zmiennymi daty, a końcowa zmienna `FilterResellerSales` nadal będzie generować to samo zapytanie SQL. Trzeba też pamiętać, że język M rozróżnia wielkość liter. Na przykład odwołanie się do zmiennej zdefiniowanej jako `StartDate` za pomocą nazwy `Startdate` powoduje błędne działanie kwerendy.

Komentarze jednowierszowe mogą być wprowadzane w zapytaniach M po dwóch ukośnikach (//), jak w przykładzie z trzema poprzednimi latami. Komentarze wielowierszowe zaczynają się od znaków (/*) i kończą znakami (*/), tak jak zapytania T-SQL dla bazy SQL Server.

Gdyby wymaganie dotyczyło jedynie pobierania danych z trzech lat wstecz względem bieżącej daty (na przykład od 30 grudnia 2018 do 30 grudnia 2021), zmienną `StartDate` można by obliczyć za pomocą funkcji `Date.AddYears()` w następujący sposób:

```
// Trzy poprzednie lata (np. od 18 października 2018 r. do 18 października 2021 r.)
  CurrentDate = DateTime.Date(DateTime.LocalNow()),
  StartDate = Date.AddYears(CurrentDate,-3)
```

Na koniec należy zauważyć, że standardowa baza danych `AdventureWorksDW` ma dane na temat wielkości sprzedaży w sieci pośredników tylko do roku 2013, więc użycie funkcji `Date.AddYears()` w celu odjęcia lat od obliczonej zmiennej `CurrentDate` jest w tym przypadku konieczne.

W następnym przykładzie użyjemy ponownie tego zapytania o trzy poprzednie lata, ale rozszerzamy je tak, aby w środowiskach testowych i produkcyjnych ładowane były wszystkie lata.

Łączenie zapytań

Wiele zapytań można łączyć (scalać) za pomocą funkcji `Table.Combine`. Funkcja ta może być niezwykle przydatna w przypadku zapytań typu `Folder`, gdy w ramach modelu danych trzeba wiele plików o tym samym formacie złączyć w jedną tabelę.

W kolejnym przykładzie utworzono parametr o nazwie `Mode` (tryb) z listą dostępnych wartości: `Dev`, `Test` i `Prod`. Przytoczone niżej zapytanie sprawdza wartość tego parametru. Jeśli parametr ma wartość `Dev`, to za pomocą zapytania z poprzedniego przykładu zwracane są tylko trzy poprzednie lata. W przeciwnym razie zapytanie z poprzedniego przykładu jest łączone z wyrażeniem tabelarycznym, które pobiera wszystkie dodatkowe lata. Dwa wyrażenia tabelaryczne są łączone ze sobą za pomocą funkcji `Table.Combine`:

```
let
// Daty wyznaczające trzy poprzednie lata
    CurrentDate = DateTime.Date(DateTime.LocalNow()),
    StartYear = Date.Year(CurrentDate)-3,
    StartDate = #date(StartYear,1,1),
    Results =
        if Mode = "Dev"
            then Trailing3Years
            else
                Table.Combine(
                    {
                        Trailing3Years,
                        Table.SelectRows(
                            AdWorksSQLServer{[Schema = "BI", Item =
                            ↪"vFact_ResellerSales"]}[Data],
                            each Date.From([OrderDate]) < StartDate)
                    }
```

```
                )
in
    Results
```

W tym przykładzie zapytanie `Trailing3Years` (poprzednie 3 lata) jest zwykle ustawiane tak, aby nie ładowało się do modelu danych, a całe powyższe zapytanie służy jako główna tabela faktów dla sprzedaży przez pośredników. Dzięki takiemu podejściu programiści mogą pracować z dużo mniejszym lokalnym zbiorem danych i łatwo dołączyć wszystkie wymagane dane podczas przenoszenia zbioru ze środowiska deweloperskiego do testowego i potem do produkcyjnego. Takie etapowe wdrażanie projektu omawiamy dokładniej w rozdziale 10. „Zarządzanie obszarami roboczymi i zawartością" (w podrozdziale „Wdrożenia etapowe").

Mamy tu również pokazane zastosowanie instrukcji `if`, której ogólny format przedstawia się następująco:

```
if <wyrażenie prawdziwe lub fałszywe> then <wyrażenie> else <wyrażenie>
```

W tym przykładzie użycie instrukcji `if` zapobiega włączeniu do zapytania wszystkich wierszy z tabeli `vFact_ResellerSales` (sprzedaże przez pośredników), gdy tryb jest ustawiony na `Dev` (tryb deweloperski), zapewniając mniejszy ogólny rozmiar zbioru danych i szybsze jego ładowanie w środowisku programistycznym.

Następny przykład również dotyczy tabel faktów i wiąże się z odświeżaniem przyrostowym.

Odświeżanie przyrostowe dla tabel faktów

Odświeżanie przyrostowe jest funkcją Power BI, która może znacznie zmniejszyć czas odświeżania bardzo dużych tabel faktów.

Odświeżanie przyrostowe pozwala na odświeżenie tylko części danych (nowych i zmienionych) w obrębie tabeli, w przeciwieństwie do przeładowywania wszystkich wierszy podczas każdego cyklu odświeżania, co jest domyślnym zachowaniem Power BI.

Korzystanie z odświeżania przyrostowego wymaga użycia dwóch parametrów o zarezerwowanych nazwach: `RangeStart` (początek zakresu) i `RangeEnd` (koniec zakresu). Parametry te muszą mieć przypisany typ daty lub czasu. W poniższym przykładzie zapytanie o sprzedaż internetową zostało zmodyfikowane przez dodanie filtrowania specyficznego dla implementacji odświeżania przyrostowego.

```
let
    Source = AdWorksSQLServer,
    InternetSales = Source{[Schema="BI",Item="vFact_InternetSales"]} [Data],
    FilterRows = Table.SelectRows(InternetSales, each [OrderDateKey] >
    ↪ConvertDateKey(RangeStart) and [OrderDateKey] <= ConvertDateKey(RangeEnd))
in
    FilterRows
```

Kod w kroku `FilterRows` (wiersze filtrowane) wykorzystuje funkcję `Table.SelectRows()` w połączeniu z parametrami `RangeStart` i `RangeEnd` oraz niestandardową funkcją `ConvertDateKey` (konwersja klucza daty). Funkcja `ConvertDateKey` jest konieczna, ponieważ w zapytaniu odwołano się do kolumny klucza zastępczego `OrderDateKey` (klucz daty zamówienia). Kolumna ta jest kluczem zastępczym, ponieważ przechowuje datę jako wartość całkowitą w formacie `YYYYMMMDD`, a nie jako wartość typu `date` lub `datetime`.

Kod niestandardowej funkcji `ConvertDateKey` wygląda następująco:

```
let
ConvertDateKey = (DateTime as datetime) => Date.Year(DateTime) * 10000
↪+ Date.Month(DateTime) * 100 + Date.Day(DateTime)
in
    ConvertDateKey
```

Podczas filtrowania tabeli z użyciem parametrów `RangeStart` i `RangeEnd` ważne jest, aby tylko jeden z warunków porównywania z tymi parametrami zawierał opcję równości (`=`). W przeciwnym razie może dojść do powielania danych, ponieważ niektóre wiersze danych mogą spełniać warunek końca jednego cyklu odświeżania i warunek początku następnego cyklu odświeżania.

Początkowy cykl odświeżania dla tego zbioru danych ładuje wszystkie wiersze w ramach zbioru, a parametr `RangeStart` jest ustawiany automatycznie przez usługę. Kolejny cykl odświeżania ustawia parametr `RangeEnd` na bieżącą datę i godzinę, dzięki czemu do tabeli danych dodawane są tylko nowe i zaktualizowane dane.

Teraz skierujemy uwagę na przykłady zapytań M dla wymiarów i zaczniemy od zapytania na temat klienta.

Kolumna historii klienta

W tym przykładzie celem jest dodanie kolumny do tabeli wymiaru `Customer` w celu pogrupowania klientów w czterech kategoriach na podstawie daty ich pierwszego zakupu. W szczególności nowa kolumna musi wykorzystać istniejącą kolumnę dat pierwszego zakupu i przypisać wiersze klientów do jednej z następujących czterech kategorii: `First Year Customer` (klient roczny), `Second Year Customer` (klient dwuletni), `Third Year Customer` (klient trzyletni) i `Legacy Customer` (klient starszy).

Kolumna jest obliczana codziennie przy każdym zaplanowanym odświeżeniu danych i korzystają z niej zespoły sprzedaży i marketingu, aby koncentrować swoje wysiłki na segmentach nowych i starszych klientów.

W celu utworzenia nowej kolumny zastosowano funkcję `Table.AddColumn()` z generatorem kolumn łączącym funkcje operujące datami z logiką warunkową (`if..then..else`):

```
let
// Przedziały dat w historii klientów
   CurrentDate = DateTime.Date(DateTime.LocalNow()),
   OneYearAgo = Date.AddYears(CurrentDate,-1),
   TwoYearsAgo = Date.AddYears(CurrentDate,-2),
   ThreeYearsAgo = Date.AddYears(CurrentDate,-3),
// Wymiar Customer (klient)
    Source = AdWorksSQLServer,
    Customer = Source{[Schema = "BI", Item = "vDim_Customer"]}[Data],
    CustomerHistoryColumn = Table.AddColumn(Customer, "Customer History
Segment",
    each
    if [DateFirstPurchase] >= OneYearAgo then "First Year Customer"
    else if [DateFirstPurchase] >= TwoYearsAgo and [Customer First Purchase
    ↪Date] < OneYearAgo then "Second Year Customer"
    else if [DateFirstPurchase] >= ThreeYearsAgo and [Customer First
    ↪Purchase Date] < TwoYearsAgo then "Third Year Customer"
    else "Legacy Customer", type text)
in
    CustomerHistoryColumn
```

Jak widać na rysunku 3.12, kolumna `Customer History Segment` (segment historii klientów) generuje jedną z czterech wartości tekstowych na podstawie kolumny `Date ↪FirstPurchase` (data pierwszego zakupu).

	CustomerKey	DateFirstPurchase	Customer History Segment
382	11381	6/12/2011	Legacy Customer
383	11382	11/3/2012	Third Year Customer
384	11383	1/1/2014	First Year Customer
385	11384	8/2/2013	Second Year Customer

Rysunek 3.12. Kolumna segmentu historii klientów w Edytorze Power Query

Podobnie jak we wcześniejszym przykładzie zapytania M z filtrowaniem trzech poprzednich lat, logika warunkowa dla nowej kolumny klienta jest tłumaczona na język T-SQL przez mechanizm składania zapytań:

```
[_].[DateFirstPurchase] as [DateFirstPurchase],
[_].[CommuteDistance] as [CommuteDistance],
case
    when [_].[DateFirstPurchase] >= convert(date, '2013-12-30')
    then 'First Year Customer'
    when [_].[DateFirstPurchase] >= convert(date, '2012-12-30')
    ↪and [_].[DateFirstPurchase] < convert(date, '2013-12-30')
    then 'Second Year Customer'
    when [_].[DateFirstPurchase] >= convert(date, '2011-12-30')
    ↪and [_].[DateFirstPurchase] < convert(date, '2012-12-30')
    then 'Third Year Customer'
```

```
        else 'Legacy Customer'
    end as [Customer History Segment]
from [BI].[vDim_Customer] as [_]
```

Dwie dynamiczne kolumny, `Calendar Year Status` i `Calendar Month Status`, zawarte w widoku SQL wymiaru daty, o których była mowa wcześniej w tym rozdziale, mogły być również obliczone za pomocą funkcji M.

Przedstawimy teraz dodatkowe szczegóły dotyczące ostatniego parametru funkcji `Table.AddColumn()`, czyli `type text` (typ tekstowy).

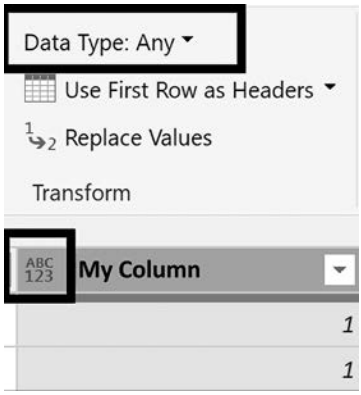
Typy danych kolumny pochodnej

Ostatni parametr funkcji `Table.AddColumn()` jest opcjonalny, ale powinien zostać podany w celu określenia typu danych nowej kolumny.

W przykładzie z segmentem historii klientów nowa kolumna ma przypisany tekstowy typ danych. Gdyby tworzono kolumnę z liczbami całkowitymi, przypisano by jej typ `Int64.Type`, jak w poniższym przykładzie:

```
MyNewColumn = Table.AddColumn(Product, "My Column", each 5, Int64.Type)
```

Jeśli nie określimy typu danych tworzonej kolumny w funkcji `Table.AddColumn()` lub później w zapytaniu przy użyciu funkcji `Table.TransformColumnTypes()`, to nowa kolumna przyjmie typ danych `Any` (dowolny), co pokazano na rysunku 3.13.



Rysunek 3.13. Typ danych Any

Kolumny o typie danych `Any` są ładowane do modelu jako dane tekstowe. Projektanci zbiorów danych powinni zadbać, aby każda kolumna w każdym zapytaniu miała określony typ danych. Innymi słowy, typ danych `Any` (czyli tak naprawdę nieznany) nie powinien być dopuszczany w zapytaniach M.

Przejdziemy teraz do przedstawienia przykładowego zapytania M dla wymiaru `Product`.

Integracja wymiaru produktu

Widok SQL dla wymiaru produktu przywołany w rozdziale 2. „Przygotowanie źródeł danych" zawierał następujące cztery operacje:

1. Połączenie tabel wymiarów `Product`, `ProductSubcategory` i `ProductCategory` w jedno zapytanie.
2. Utworzenie niestandardowej kolumny grup kategorii produktów (na przykład rowery i nierowery).
3. Zastosowanie wygodnych w raportowaniu nazw kolumn ze spacjami i właściwą wielkością liter.
4. Zastąpienie wszelkich pustych wartości w kolumnach `Product Subcategory` i `Product Category` wartością `Undefined` (nieokreślona).

Prawie wszystkie operacje dostępne dla zapytań SQL typu `SELECT` mają swoje odpowiedniki w funkcjach języka M. Jeśli w ramach źródła danych nie można utworzyć widoku SQL dla wymiaru produktu, to taki sam rezultat można uzyskać za pomocą następującej kwerendy M:

```
let
    Source = AdWorksSQLServer,
// Widoki tabeli wymiaru Product
    Product = Source{[Schema = "BI", Item = "vDim_Product"]}[Data],
    ProductSubCat = Source{[Schema = "BI", Item =
    ↪"vDim_ProductSubcategory"]}[Data],
    ProductCat = Source{[Schema = "BI", Item = "vDim_ProductCategory"]}[Data],
// Złączenia zewnętrzne tabeli Product
    ProductJoinSubCat = Table.NestedJoin(Product,"ProductSubcategoryKey",
    ↪ProductSubCat,"ProductSubcategoryKey","ProductSubCatTableCol",
    ↪JoinKind.LeftOuter),
    ProductJoinSubCatCol = Table.ExpandTableColumn(ProductJoinSubCat,
    ↪"ProductSubCatTableCol",{"EnglishProductSubcategoryName",
    ↪"ProductCategoryKey"},{"Product Subcategory", "ProductCategoryKey"}),

    ProductJoinCat = Table.NestedJoin(ProductJoinSubCatCol,
    ↪"ProductCategoryKey",ProductCat,"ProductCategoryKey",
    ↪"ProductCatTableCol",JoinKind.LeftOuter),
    ProductJoinCatCol = Table.ExpandTableColumn(ProductJoinCat,
    ↪"ProductCatTableCol", {"EnglishProductCategoryName"},{"Product Category"}),
// Zmienianie nazw wybranych kolumn
    ProductDimCols = Table.SelectColumns(ProductJoinCatCol,{"ProductKey",
    ↪"ProductAlternateKey","EnglishProductName","Product Subcategory",
    ↪"Product Category"}),
    ProductDimRenameCols = Table.RenameColumns(ProductDimCols,
    ↪{{"ProductKey", "Product Key"},{"ProductAlternateKey","Product
    ↪Alternate Key"},{"EnglishProductName","Product Name"}
    }),
```

```
    // Kolumna grupująca kategorie produktu
        ProductCatGroupCol = Table.AddColumn(ProductDimRenameCols,"Product
        ↪Category Group",
            each
            if [Product Category] = "Bikes" then "Bikes"
            else if [Product Category] = null then "Undefined"
            else "Non-Bikes",
            type text),
    // Usuwanie wszystkich wartości Null
        UndefinedCatAndSubcat = Table.ReplaceValue(ProductCatGroupCol,null,
        ↪"Undefined",Replacer.ReplaceValue,{"Product Subcategory","Product
        ↪Category"})
in
        UndefinedCatAndSubcat
```

Trzy tabele wymiaru produktu w schemacie `dbo` hurtowni danych są przywoływane z zapytania przejściowego `AdWorksSQLServer` opisanego wcześniej w tym rozdziale.

Funkcja `Table.NestedJoin()` (złączenie zagnieżdżone) działa analogicznie jak klauzula `LEFT JOIN` (złączenie lewostronne) w SQL, a funkcja `Table.ExpandTableColumn()` (rozwinięcie kolumny tabel) ekstrahuje potrzebne kolumny `Product Subcategory` i `Product Category` oraz zmienia ich nazwy.

Po wybraniu i zmianie nazwy kolumn wyrażenie warunkowe w funkcji `Table.Add↪Column()` tworzy kolumnę grupową o nazwie `Product Category`. Na koniec funkcja `Table.ReplaceValue()` zastępuje w kolumnach `Product Category` i `Product Subcategory` wszelkie wartości `null` ciągiem tekstowym `"Undefined"`. Edytor Power Query umożliwia podgląd wyników, co pokazano na rysunku 3.14.

Product Key	Product Alternate Key	Product Name	Product Subcategory	Product Category	Product Category Group
308	FR-M94S-38	HL Mountain Frame - Silver, 38	Mountain Frames	Components	Non-Bikes
309	FR-M94S-38	HL Mountain Frame - Silver, 38	Mountain Frames	Components	Non-Bikes
310	BK-R93R-62	Road-150 Red, 62	Road Bikes	Bikes	Bikes
311	BK-R93R-44	Road-150 Red, 44	Road Bikes	Bikes	Bikes

Rysunek 3.14. Podgląd wyniku zapytania Product wyświetlany w Edytorze Power Query

Pomimo dodatkowych kroków i złożoności tego zapytania w stosunku do poprzednich przykładów zapytań w języku M (filtr trzech poprzednich lat, kolumna `Customer History Segment`) całe zapytanie jest tłumaczone na pojedynczą instrukcję SQL i wykonane przez źródłową bazę danych w systemie SQL Server. Polecenie *View Native Query* (*Wyświetl zapytanie natywne*) dostępne w menu kontekstowym panelu *Applied Steps* (*Zastosowane kroki*) **Edytora Power Query** ujawnia specyficzną składnię zapytania SQL wygenerowanego przez złożenie zapytania:

```
select [_].[ProductKey] as [Product Key],
    [_].[ProductAlternateKey] as [Product Alternate Key],
    [_].[EnglishProductName] as [Product Name],
    case
        when [_].[EnglishProductSubcategoryName] is null
        then 'Undefined'
        else [_].[EnglishProductSubcategoryName]
    end as [Product Subcategory],
    case
        when [_].[EnglishProductCategoryName] is null
        then 'Undefined'
        else [_].[EnglishProductCategoryName]
    end as [Product Category],
    case
        when [_].[EnglishProductCategoryName] = 'Bikes' and [_].
[EnglishProductCategoryName] is not null
        then 'Bikes'
        when [_].[EnglishProductCategoryName] is null
        then 'Undefined'
        else 'Non-Bikes'
    end as [Product Category Group]
from
```

Zauważ, że dedykowany obiekt widoku SQL w schemacie BI (na przykład `BI.vDim_ProductSubcategory`) jest dostępny dla każdej z trzech tabel wymiaru produktu. Zgodnie z tym, co napisaliśmy w rozdziale 2. „Przygotowanie źródeł danych" (w podrozdziale „Widoki SQL"), powinno się zawsze uzyskiwać dostęp do widoków SQL, gdyż jest to równoznaczne z dostępem do tabel źródłowych.

Zauważ, że ze względu na wymóg zewnętrznego złączenia lewostronnego i obecność wspólnych nazw kolumn nie można tu użyć funkcji `Table.Join()`. Próba takiego złączenia tabel o wspólnych nazwach kolumn, takich jak `ProductSubcategoryKey` czy `ProductCategoryKey`, spowodowałaby błąd.

Mimo że złączenie zewnętrzne lewostronne jest domyślnym ustawieniem dla funkcji `Table.NestedJoin()`, zaleca się jawne określenie rodzaju złączenia — na przykład `JoinKind.Inner` (wewnętrzne), `JoinKind.LeftOuter` (zewnętrzne lewostronne) lub `JoinKind.LeftAnti` (antyzłączenie lewostronne) — zgodnie z wartościami zmiennych `ProductJoinSubCat` i `ProductJoinCat`. W celu odświeżenia sobie wiadomości na temat typów złączeń dostępnych w Power Query proszę zapoznać się z artykułem zamieszczonym pod adresem *https://learn.microsoft.com/pl-pl/power-query/merge-queries-overview*.

Jeśli tylko użycie funkcji `Table.Join` jest możliwe, to na ogół należy dać jej pierwszeństwo przed `Table.NestedJoin`, a to dlatego, że ta druga wykonuje operację łączenia tabel w oparciu o zasoby lokalne, podczas gdy pierwszą można przerzucić do systemu źródłowego, co zazwyczaj zwiększa wydajność kwerendy. Warto jednak zauważyć, że

operacja **scalania kwerend** dostępna w **graficznym interfejsie (GUI) Edytora Power Query** domyślnie stosuje funkcję `Table.NestedJoin`.

Gdy źródłem danych dla zbioru Power BI ma być jakiekolwiek źródło nieustrukturyzowane lub należące do użytkownika biznesowego, zazwyczaj zaleca się zaimplementowanie w ramach zapytania M dodatkowej logiki jakości danych i obsługi błędów.

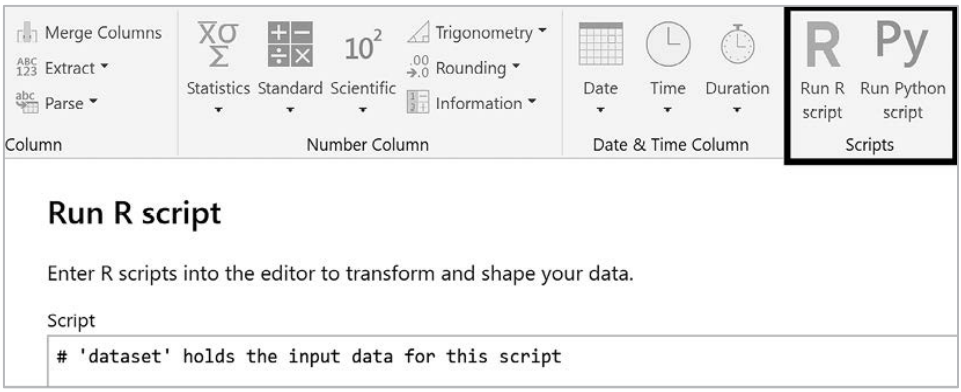
Na przykład do zapytania `Sales and Margin Plan`, które pobiera dane ze skoroszytu Excela, można dodać krok wywołujący funkcję `Table.Distinct()`, aby usunąć wszelkie zduplikowane wiersze. Podobnie trzeci parametr funkcji `Table.SelectColumns()` (wybierz kolumny), np. `MissingField.UseNull` (brakujące pole zastąp wartością `Null`), można wykorzystać do obsługi przypadków braku kolumny źródłowej lub zmiany jej nazwy.

Wprawdzie język M sam zapewnia szerokie możliwości transformacji danych, ale niektórzy analitycy mogą lepiej znać innymi języki, takie jak R czy Python. Właśnie taką sytuację omówimy w kolejnym przykładzie.

Przekształcanie danych za pomocą skryptów R i Pythona

W ramach zapytania M jako część procesu transformacji danych można wykorzystywać skrypty R lub Pythona.

Jak pokazano na rysunku 3.15, na karcie *Transform* (*Przekształć*) **Edytora Power Query** w Power BI Desktop są dostępne polecenia *Run R script* (*Uruchom skrypt języka R*) i *Run Python script* (*Uruchom skrypt języka Python*).



Rysunek 3.15. Polecenia Run R script (Uruchom skrypt języka R) i Run Python script (Uruchom skrypt języka Python) w Edytorze Power Query

Żeby aplikacja Power BI Desktop mogła wykonywać skrypty napisane w R i Pythonie, języki te muszą być zainstalowane na lokalnym komputerze wraz ze wszystkimi potrzebnymi pakietami. Jeśli używana jest **brama danych**, to R, Python i odpowiednie

pakiety muszą być zainstalowane również na serwerze uruchamiającym tę bramę. I rzecz najważniejsza, aby skrypty działały poprawnie, gdy zbiór danych jest publikowany w usłudze Power BI, poziomy prywatności dla źródeł danych muszą być ustawione na *Public* (*Publiczne*).

Dla większości organizacji to ograniczenie wyklucza stosowanie skryptów R i Pythona w Power BI. Poza tym obecność takich skryptów tworzy kolejną warstwę złożoności dla rozwiązania zawierającego już kwerendy w językach SQL, M i DAX.

Przyjrzyjmy się jeszcze innemu sposobowi tworzenia zapytań w języku M, a mianowicie z użyciem **przepływów danych**.

Przepływy danych

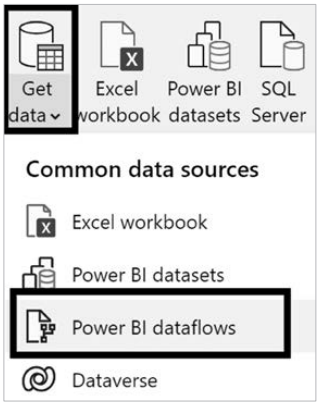
Mówiąc najprościej, **przepływy danych** to zapytania Power Query M utworzone w usłudze Power BI. W przypadku obszarów roboczych innych niż *My Workspace* (*Mój obszar roboczy*) usługa Power BI udostępnia interfejs niemal identyczny jak **Edytor Power Query** w Power BI Desktop do tworzenia i edycji zapytań Power Query.

Przepływy danych uzyskują dostęp do systemów źródłowych w taki sam sposób jak zapytania Power Query utworzone w **Edytorze Power Query** dostępnym w Power BI Desktop. Jednakże dane pobierane z tych zapytań dla przepływów danych w trybie importu są przechowywane w folderach zgodnych z Dataverse w ramach usługi Azure Data Lake Storage Gen2. Przepływy danych mogą być również tworzone z dostępem bezpośrednim (w trybie DirectQuery) do systemów źródłowych.

Istnieje kilka kluczowych zalet stosowania przepływów danych w analityce firmowej. Główną z nich jest możliwość wielokrotnego wykorzystania. Raz utworzony przepływ danych może być użyty jako źródło danych w wielu różnych plikach aplikacji Power BI Desktop podczas projektowania zbioru danych. Przepływy danych w Power BI są opcją przy korzystaniu z polecenia *Get data* (*Pobierz dane*) zarówno w Power BI Desktop, jak i w **Edytorze Power Query**, co pokazano na rysunku 3.16.

Oznacza to, że można raz opracować logikę transformacji danych w ramach zapytania i potem wykorzystywać ją w wielu różnych zestawach danych, dzięki czemu tworzenie tych ostatnich staje się wydajniejsze, bardziej ustandaryzowane i mniej podatne na błędy.

Drugą kluczową zaletą przepływów danych jako źródeł dla zestawów danych w trybie importu jest to, że przepływ danych izoluje systemy źródłowe od zbiorów danych Power BI. Oznacza to, że podczas odświeżania zbiorów danych Power BI pobierane są dane z instancji Azure Data Lake Storage Gen2 zamiast z samych systemów źródłowych, dzięki czemu proces odświeżania w żaden sposób nie wpływa na te systemy.



Rysunek 3.16. Przepływy danych w Power BI jako źródło danych

Przepływ danych może być odświeżony raz, a potem odświeżenia we wszystkich zbiorach danych Power BI wykorzystujących ten przepływ danych wpływają tylko na wysoce skalowalną instancję Azure Data Lake Storage Gen2, a nie na systemy źródłowe. Nabiera to jeszcze większego znaczenia, gdy źródłowy system danych, do którego ma dostęp przepływ danych, jest lokalny i wymaga zastosowania bramy danych.

Ponieważ dane pobierane przez zbiory danych Power BI Desktop znajdują się w instancji Azure Data Lake Storage Gen2, do ich odświeżania nie jest wymagana brama danych. Brama jest konieczna tylko podczas odświeżania samego przepływu danych.

Izolacja systemów źródłowych od zbiorów danych Power BI jest korzystna również z punktu widzenia bezpieczeństwa. Zamiast przekazywać twórcom zbiorów danych poświadczenia dostępu do systemów źródłowych, firmowy zespół BI może po prostu przekazać im poświadczenia dostępu do samego przepływu danych.

Idealnym stanem dla BI w przedsiębiorstwie byłaby jedna hurtownia danych i odpowiadający jej jeden zbiór danych Power BI do celów raportowania, jednak często coś takiego jest niemożliwe. Na przykład lista produktów dostosowanych do potrzeb klienta może być przechowywana przez zespół marketingowy w witrynie SharePoint, a jej włączenie do korporacyjnej hurtowni danych nie jest na razie planowane. Jednak ta lista może być przydatna w wielu zbiorach danych Power BI używanych przez działy sprzedaży, marketingu i dostaw.

W takiej sytuacji można by stworzyć jeden przepływ danych, który łączyłby się z tym plikiem źródłowym i stosował niezbędne transformacje. Zbiory danych dotyczące sprzedaży, marketingu i łańcucha dostaw mogłyby być połączone z tym jednym przepływem, tak że wszelkie aktualizacje tego centralnego zasobu w naturalny sposób przepływałyby do wszystkich zależnych zbiorów danych.

Jak wspominaliśmy, przepływy danych zapewniają kilka istotnych korzyści, a w przypadku zastosowania ich w Power BI Premium użytkownik uzyskuje dostęp do dodatkowych funkcji związanych z ich obsługą.

Funkcje przepływu danych w Power BI Premium

Power BI Premium obsługuje dodatkowe funkcje przepływu danych, w tym *Enhanced compute engine* (ulepszony aparat obliczeniowy), *DirectQuery* (zapytanie bezpośrednie), *Computed entities* (jednostki obliczone), *Linked entities* (jednostki połączone), *Incremental refresh* (odświeżanie przyrostowe) oraz *AutoML* (zautomatyzowane uczenie maszynowe).

Usprawniony silnik obliczeniowy może wyraźnie zwiększyć prędkość odświeżania dla złożonych transformacji, takich jak złączenia, grupowanie, filtrowanie i operacje rozróżniania. Jak wcześniej wspomniano, tworzenie przepływów danych dla źródeł DirectQuery jest obsługiwane, ale tylko w wersji Premium. Należy zauważyć, że modele złożone, które mają źródła w trybach zarówno importu, jak i DirectQuery, obecnie nie dają możliwości włączenia przepływów danych jako źródeł.

Encje obliczane pozwalają na wykonywanie w magazynie obliczeń łączących dane z wielu przepływów w nową, zespoloną encję lub wzbogacających istniejącą. Na przykład przepływ danych o nazwie `Produkt` może zostać wzbogacony o informacje z przepływów `Kategoria produktu` i `Podkategoria produktu`.

Podobnie jak w przypadku encji obliczanych, encje łączone pozwalają na odwołanie się do innych przepływów danych w celu wykonania obliczeń (encje obliczane) lub ustanowienia tabeli służącej jako jedyne źródło prawdy dla innych przepływów danych.

I na koniec funkcja AutoML automatyzuje tworzenie modeli **uczenia maszynowego** (**ML**), umożliwiając automatyczne rozpoznawanie wzorców, analizę nastrojów itp.

Na zakończenie rozdziału omówimy różne narzędzia edycyjne używane do tworzenia zapytań w języku M.

Narzędzia edycyjne Power Query M

Podobnie jak w przypadku innych języków i typów projektów, dostępne są różne narzędzia do edycji kodu, które wspierają tworzenie i dokumentowanie wersji zapytań w języku M oraz zarządzanie nimi.

Projektanci zbiorów danych mogą tworzyć zapytania M dla Power BI i innych projektów Microsoft za pomocą zaawansowanego edytora w Power BI Desktop i interfejsu edycji

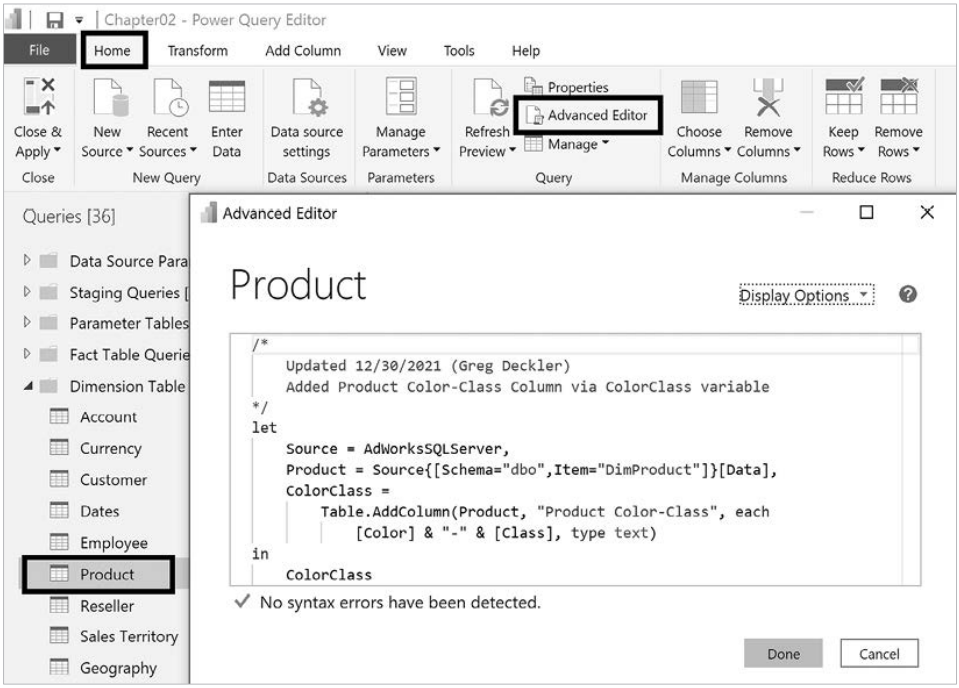
przepływów danych w usłudze Power BI, ale także przy użyciu aplikacji Visual Studio lub Visual Studio Code. Wszystkie te narzędzia zawierają podobne funkcje programistyczne, takie jak IntelliSense, podświetlanie składni czy zintegrowana kontrola źródeł.

W kolejnych punktach omówimy te narzędzia nieco dokładniej.

Edytor zaawansowany

W Power BI Desktop i dla przepływów danych w usłudze Power BI kod M każdego zapytania jest dostępny w oknie *Advanced Editor* (*Edytor zaawansowany*) będącym częścią aplikacji **Edytor Power Query**.

Aby otworzyć to okno, w Edytorze Power Query wybierz z listy zapytań po lewej stronie interesującą Cię pozycję i na karcie *Home* (*Narzędzia główne*) kliknij ikonę *Advanced Editor* (*Edytor zaawansowany*). Wygląd okna edytora zaawansowanego jest pokazany na rysunku 3.17.



Rysunek 3.17. Edytor zaawansowany dostępny w Power BI Desktop

Doświadczeni autorzy zapytań M często używają ikon transformacji danych dostępnych w Edytorze Power Query, aby szybko stworzyć wstępną wersję jednego kodu zapytania. Potem przechodzą do okna *Advanced Editor* (*Edytor zaawansowany*) lub do

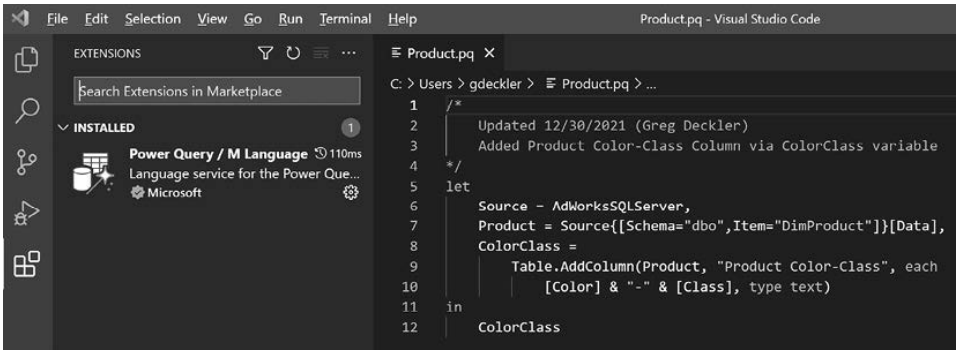
jakiegoś zewnętrznego narzędzia i tam analizują, a w razie potrzeby również modyfikują ten kod, np. przez zmianę nazw zmiennych lub wykorzystanie opcjonalnych parametrów niektórych funkcji języka M.

W przypadku popularnych i prostych zadań transformacji danych, takich jak filtrowanie wierszy na podstawie jednej wartości kolumny (np. `State = "Kansas"`), kod M wygenerowany przez **Edytor Power Query** zwykle wymaga tylko niewielkiej korekty. Dla bardziej złożonych zapytań z niestandardowymi lub mniej powszechnymi warunkami interfejs graficzny **Edytora Power Query** okazuje się niewystarczający i trzeba sięgać po narzędzia umożliwiające kodowanie w sposób bezpośredni.

Edytor zaawansowany jest wygodny, ale brakuje mu odpowiedniej integracji z kontrolą źródła. Mankament ten można ominąć przez zastosowanie alternatywnych narzędzi, takich jak Visual Studio Code.

Visual Studio Code

Visual Studio Code to darmowe, lekkie narzędzie edytorskie firmy Microsoft, które jest dostępne na wszystkich platformach (Windows, Mac i Linux). Po zainstalowaniu rozszerzenia o nazwie *Power Query / M Language* zapewnia obsługę edycji kodu dla języka Power Query M, co pokazano na rysunku 3.18.



Rysunek 3.18. Zapytanie M w Visual Studio Code

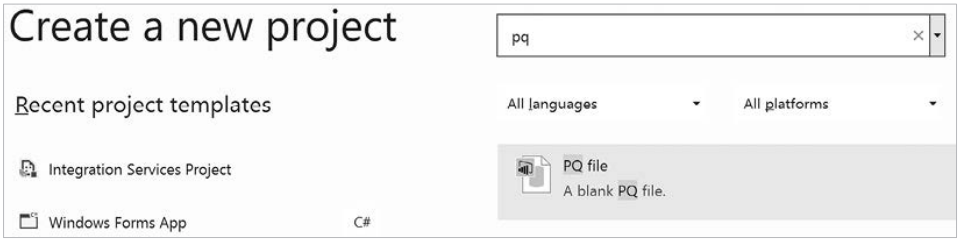
W prezentowanym przykładzie to samo zapytanie o nazwie `Product` przeglądane w Edytorze zaawansowanym będącym częścią programu Power BI Desktop zostało skopiowane do pliku z rozszerzeniem *.pq* obsługiwanym przez Visual Studio Code. Zmiana rozszerzenia pozwoliła na uaktywnienie funkcji edycyjnych, takich jak kolorowanie słów kluczowych, automatyczne uzupełnianie kodu i wyświetlanie informacji o wpisywanym kodzie. Pliki zapytań w języku M mogą być otwierane i zapisywane z następującymi czterema rozszerzeniami: *.m*, *.M*, *.pq* i *.PQ*.

Należy zauważyć, że kod M opracowany w Visual Studio Code nie może być wykorzystany bezpośrednio przez Power BI. Należy go skopiować z Visual Studio Code lub repozytorium kontroli źródeł i wkleić do Edytora zaawansowanego. Doświadczone zespoły BI w firmach przywiązują jednak dużą wagę do właściwej kontroli wersji, zwłaszcza jeśli chodzi o zmiany w bazowych tabelach danych i transformacjach danych.

Ponieważ rozszerzenie *.pq* jest używane przez Power Query SDK dla Visual Studio, zaleca się stosowanie go do przechowywania zapytań M w Visual Studio Code, a także w omawianym poniżej Visual Studio.

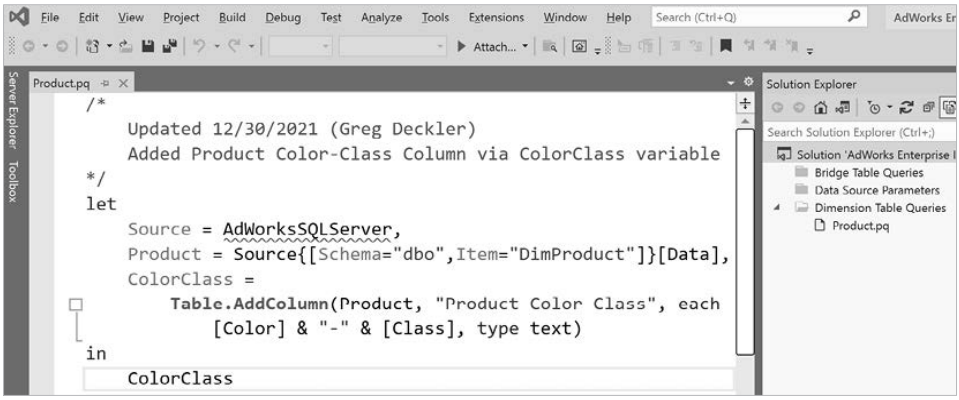
Visual Studio

Visual Studio jest rozbudowaną wersją lżejszego Visual Studio Code i jest **zintegrowanym środowiskiem programistycznym** (**IDE**) szeroko stosowanym w całej branży informatycznej. W Visual Studio 2015 i nowszych wersjach Power Query SDK może służyć do tworzenia łączników danych i zapytań M, co pokazano na rysunku 3.19.



Rysunek 3.19. Typy projektów Power Query w Visual Studio

Dzięki nowo opracowanemu formatowi plików PQ można do projektu realizowanego w Visual Studio dodać zapytania M jako oddzielne pliki (*.pq*), tak jak na rysunku 3.20.



Rysunek 3.20. Projekt Power Query w Visual Studio 2019

W przeciwieństwie do Visual Studio Code rozszerzenie pliku dla projektów Power Query jest tylko jedno (*.pq*). Co najważniejsze, w pełni obsługiwane są funkcje IntelliSense dla języka M, co znacznie ułatwia znalezienie funkcji M istotnych dla konkretnych operacji przekształcania danych. IntelliSense jest ogólnym terminem dla funkcji wspomagających pisanie kodu, takich jak automatyczne uzupełnianie, podpowiadanie treści (informacje o parametrach) i sygnalizowanie błędów składniowych.

Ponadto w przeciwieństwie do Visual Studio Code zapytania M mogą być wykonywane z poziomu Visual Studio za pośrednictwem Power Query SDK. Aby wykonać zapytanie M w Visual Studio, takie jak w poprzednim przykładzie, należy kliknąć przycisk *Start* na pasku narzędzi (zielona ikona *play*) lub nacisnąć klawisz *F5*.

Można również kliknąć prawym przyciskiem myszy projekt Power Query (np. `AdWorks Enterprise Import`), aby skonfigurować właściwości opracowywanego projektu, takie jak maksymalna liczba wierszy wyjściowych i możliwość wykonywania zapytań natywnych.

Aby zainstalować Power Query SDK dla Visual Studio, należy otworzyć witrynę *Visual Studio Marketplace* (*Extensions/Manage Extensions*) i wyszukać rozszerzenie o nazwie *Power Query SDK*.

Power Query SDK dla Visual Studio umożliwia typową integrację z narzędziami do kontroli źródeł i zarządzania projektami, takimi jak usługi Azure DevOps (dawniej Visual Studio Team Services).

To kończy naszą eksplorację metod łączenia się ze źródłami i przekształcania danych za pomocą języka M.

Podsumowanie

W tym rozdziale omówiliśmy wiele składników procesu pobierania danych ze źródła i dostarczania ich do zbioru danych w Power BI zgodnie z tym, co napisaliśmy w rozdziale 1. „Planowanie projektów Power BI”. Opisaliśmy konstruowanie warstwy dostępu do danych i procesu pobierania danych z użyciem zapytań M służących do definiowania i ładowania tabel wymiarów oraz faktów.

W następnym rozdziale wykorzystamy opisane tu kwerendy i techniki ich projektowania do tworzenia modeli danych w trybach importu i DirectQuery.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Ź

Power BI: Twój sposób na ekspercką analizę dla biznesu!

Profesjonalna analiza danych przekłada się na sukces rynkowy i zyski. Liderem w dziedzinie analityki biznesowej jest Microsoft Power BI, pakiet obejmujący usługi w chmurze, aplikacje mobilne, aplikacje do modelowania danych i tworzenia raportów, a także inne narzędzia, takie jak lokalne bramy danych. Zdobycie zaawansowanych umiejętności korzystania z Power BI umożliwi tworzenie interaktywnych i pełnych treści analiz, które ułatwią podejmowanie trafnych decyzji.

To drugie, w pełni zaktualizowane wydanie podręcznika dla profesjonalistów. Zarówno osoby zawodowo zajmujące się tworzeniem rozwiązań w Power BI, jak i zarządzający czy administrujący wdrożeniami takich rozwiązań znajdą tu coś dla siebie. Książka zawiera rozbudowaną analizę narzędzi i funkcji Power BI, dzięki którym nauczysz się kształtować i rozbudowywać dane źródłowe, jak również tworzyć modele analityczne. Dowiesz się, jak stosować niestandardowe wizualizacje, implementować nowe polecenia DAX, zarządzać obszarami roboczymi i metadanymi. Poznasz najlepsze praktyki projektowania raportów i interaktywnych pulpitów nawigacyjnych. Dowiesz się, jakie są nowe możliwości aplikacji mobilnych Power BI i techniki samoobsługowej analizy danych. Zapoznasz się z metodami zarządzania systemem, w tym z zarządzaniem cyklem życia aplikacji z użyciem potoków Power BI.

Najciekawsze zagadnienia:

- język Power Query M i przepływy danych
- skalowalne modele danych w trybach DirectQuery
- proste i zaawansowane miary DAX
- korzystanie z map ArcGIS
- skalowanie rozwiązań za pomocą Power BI Premium

Greg Deckler posiada kilka tytułów Microsoft MVP w obszarze platformy danych. Jest ekspertem w dziedzinie Power BI, a także autorem licznych rozwiązań i narzędzi dla Power BI.

Brett Powell jest konsultantem Microsoftu i autorem wielu książek technicznych. Ma bogate doświadczenie w analityce biznesowej, jest uznawany za świetnego programistę, architekta i administratora.

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 978-83-8322-445-9
9 788383 224459
Cena: 149,00 zł
packt