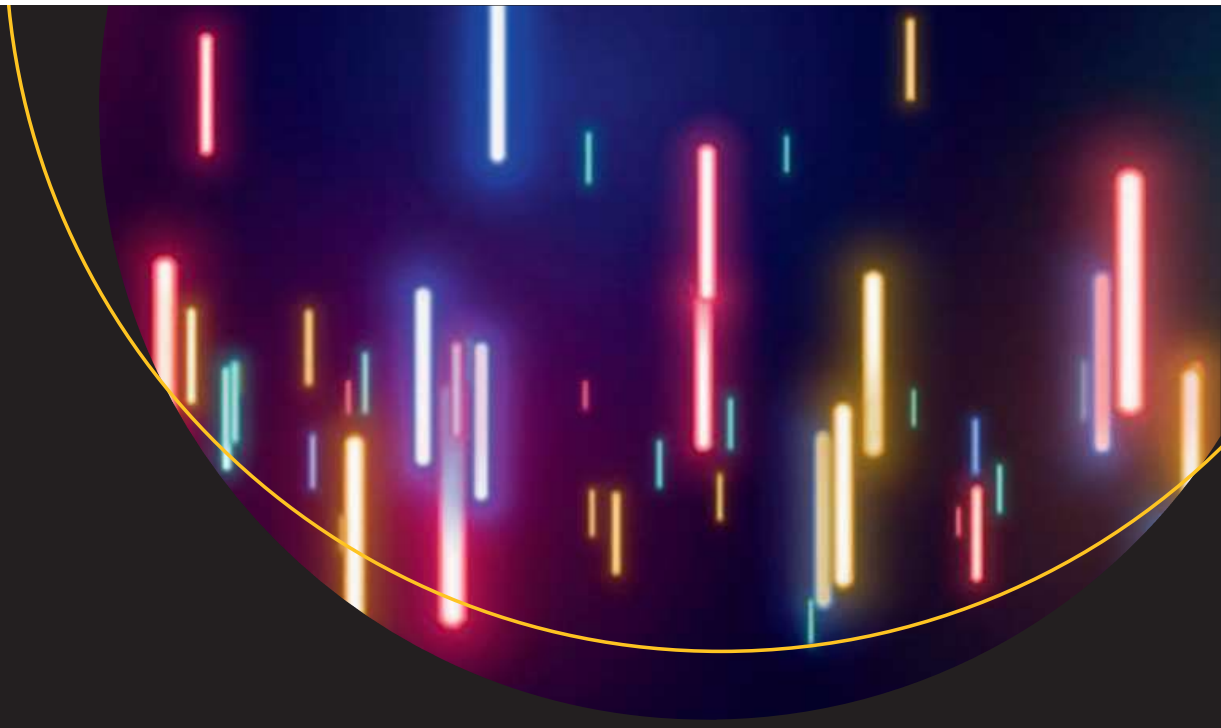

Profesjonalny kod T-SQL 2019

W stronę szybkości, skalowalności i standaryzacji rozwiązań dla SQL Server

—

Elizabeth Noble

Apress®

Apress®

Elisabeth Noble

Profesjonalny kod T-SQL 2019

W stronę szybkości, skalowalności i standaryzacji rozwiązań dla SQL Server

Przekład: Marek Włodarz

APN Promise, Warszawa 2020

Profesjonalny kod T-SQL 2019. W stronę szybkości, skalowalności i standaryzacji rozwiązań dla SQL Server

First published in English under the title
Pro T-SQL 2019: Toward Speed, Scalability, and Standardization for SQL Server Developers
by Elizabeth Noble

Polish language edition published by APN PROMISE S.A., Copyright © 2020

Autoryzowany przekład z wydania w języku angielskim, zatytułowanego: Pro T-SQL 2019: Toward Speed, Scalability, and Standardization for SQL Server Developers
by Elizabeth Noble, opublikowanego przez APress Media, LLC, oddział Springer Nature.

APN PROMISE SA, ul. Domaniewska 44a, 02-672 Warszawa
tel. +48 22 35 51 600, fax +48 22 35 51 699
e-mail: mspress@promise.pl

ISBN: 978-83-7541-431-8 (druk), 978-83-7541-438-7 (ebook)

Przekład: Marek Włodarz
Korekta: Ewa Swędrowska
Skład i łamanie: MAWart Marek Włodarz

Książkę tę dedykuję #SQLFamily. To wy pozwoliliście mi rozwijać się jako profesjonalistka danych. Daliście mi pewność siebie i odwagę dla realizacji marzeń.

Mam nadzieję, że książka ta pomoże innym tak bardzo, jak wy wszyscy pomagaliście mi.

Książkę dedykuję również mojej rodzinie i przyjaciołom w podzięce za ich miłość i wsparcie, szczególnie podczas jej pisania.

Eric i Danny, dacie radę wyzwaniom, które stoją przed wami.

Spis treści

O autorce

Elizabeth Noble zajmuje stanowisko dyrektora działu Database Development, albo „E" w firmie Pull-A-Part w rejonie metropolitalnym Atlanta. Od pierwszego zetknięcia się z bazami danych 10 lat temu była to miłość od pierwszego wejrzenia. Jej pasją jest pomaganie innym w poprawianiu jakości i szybkości wprowadzania zmian dzięki automatyzacji. Kiedy nie stara się zautomatyzować wszystkiego dookoła, można ją spotkać, gdy spędza czas ze swoimi psami, gra w disc golfa albo wędruje po górach. Często bierze udział jako wykładowca w konferencjach SQL Saturdays w całych Stanach Zjednoczonych, a także brała udział jako początkujący prezenter w konferencji PASS Summit w roku 2019.

O recenzencie technicznym

Warner Chaves ma tytuły SQL Server MCM, Data Platform MVP i pełni stanowisko Principal Consultant w firmie Pythian. Rozpoczął karierę od krótkiego epizodu jako programista .NET, co doprowadziło do pracy dla klientów korporacyjnych w organizacji ITO firmy Hewlett-Packard. Stamtąd przeszedł do bieżącego stanowiska w Pythian, gdzie buduje i zarządza rozwiązaniami bazodanowymi dla największych marek w wielu działach gospodarki. Warner jest częstym wykładowcą na spotkaniach SQLSaturday i konferencjach PASS Summit, największym wydarzeniu dedykowanym zagadnieniom Microsoft Data Platform.

Podziękowania

Chciałabym podziękować mojemu mężowi Caseyowi za to, że zawsze we mnie wierzy i zachęca do realizowania marzeń. Dziękuję też mojej Mamie za nauczenie mnie, że rozwój jest kluczem do dobrego życia. Dziękuję Tacie za przypominanie mi o tym, że powinnam nieco czasu poświęcać sobie samej. Muszę też podziękować Edowi, Jeffowi, Mike'owi, Philowi i Robowi za naukę i wprowadzenie mnie do #SQLFamily.

Na koniec chcę podziękować moim przyjaciółkom, Angeli, Joey, Jude, Lee, LeeAnn i Tammy za ich słowa zachęty podczas pisania tej książki.

Wstęp

Praca w języku T-SQL pozwala pisać kod i szybko widzieć jego wyniki. Istnieje też wielka elastyczność tworzenia wyrażeń T-SQL. W wielu przypadkach dostępny jest więcej niż jeden sposób osiągnięcia tego samego efektu. Książka ta jest przeznaczona dla deweloperów baz danych i profesjonalistów danych, którzy mają ogólną wiedzę na temat T-SQL, ale szukają sposobów poprawienia ogólnej jakości swojego kodu. Zakładam, że Czytelnik zna składnię T-SQL i już przed rozpoczęciem lektury wie, jak pisać wyrażenia SELECT, INSERT, UPDATE lub DELETE. *Profesjonalny kod T-SQL 2019* pomoże w opanowaniu sztuki pisania spójnego, czytelnego kodu o lepszej wydajności. Zaprezentuję również techniki ochrony naszego kodu przy użyciu systemów kontroli źródeł i usprawniania przebiegu wdrażania nowych rozwiązań. W ogólności celem tej książki jest przedstawienie ram pozwalających na pisanie lepszego kodu T-SQL. Jako profesjonaliści danych możemy znajdować się często w sytuacjach, gdy wymagania są bardzo wysokie, a terminy krótkie. Książka ta ma na celu przedstawić sposoby tworzenia kodu, które w przyszłości pozwolą oszczędzić czas i energię.

Książka składa się z czterech części. W pierwszej przedstawiam sposoby poprawienia czytelności naszego kodu T-SQL. Znajduje się tu przegląd różnych typów danych wraz ze wskazówkami, jak najlepiej używać tych typów. Wyjaśniam tam również zalety i wady różnych obiektów bazodanowych występujących w SQL Server. Kolejne rozdziały omawiają standaryzowanie i projektowanie kodu T-SQL. Druga część wyjaśnia, jak pisać wydajny kod T-SQL. Mamy tu omówienie stosowania projektowania opartego na zbiorach oraz wyjaśnienie zależności pomiędzy sprzętem a projektowaniem kodu T-SQL. Pokazuję tu również, jak używać planów wykonania i nowych funkcjonalności dostępnych w SQL Server 2019 w celu poprawy wydajności naszego kodu. Trzecia część poświęcona jest zarządzaniu kodem T-SQL. Rozdziały zawarte w tej części obejmują opracowywania standardów kodowania i wykorzystania systemów kontroli źródeł do przechowywania kodu. Aby móc dalej zarządzać kodem T-SQL, poznamy również pewne metody testowania i wdrażania kodu bazodanowego. Czwarta część poświęcona jest sposobom pisania kodu T-SQL, który będzie stabilnie działał z upływem czasu. Rozdziały te zawierają metody bezpiecznego dodawania nowych funkcjonalności,

rejestrowania zmian zachodzących w bazach danych i zarządzania rozrostem wielkości danych z upływem czasu.

Przykłady kodu

Wszystkie pliki skryptów przedstawione w książce dostępne są w witrynie wydawcy pod adresem:

https://github.com/Apress/pro-t-sql-2019

Część I

Budowanie zrozumiałego T-SQL

Rozdział 1

Typy danych

Typy danych są cegiełkami, z których powstaje fundament efektywnego i wydajnego kodu T-SQL. Choć większość typów danych to albo liczby, albo łańcuchy (ciągi) znaków, istnieje też kilka typów danych, które nie mieszczą się w żadnej z tych kategorii. Dla wybrania właściwego typu danych rzeczą istotną jest dobre zrozumienie tego, czym jest ten typ danych i kiedy należy go używać.

Numeryczne typy danych

Choć wszystkie liczby wydają się być bytami tego samego rodzaju, T-SQL rozróżnia wiele typów danych liczbowych. Typy te obejmują liczby całkowite oraz liczby zawierające część dziesiętną (ułamkową). Liczby można również kategoryzować jako dokładne lub przybliżone. Zrozumienie zachowania różnych typów danych numerycznych podczas wykonywania obliczeń matematycznych jest krytyczne dla zapewnienia właściwego, zgodnego z oczekiwaniem przetwarzania danych.

Wprawdzie najłatwiej byłoby wybierać najbardziej typowe typy danych z każdej kategorii, istnieją sytuacje, w których analizowanie danych będzie można najlepiej wykonywać pod warunkiem przechowywania ich w odpowiednim formacie. Przy wybieraniu typów danych trzeba rozważyć wiele czynników. Najważniejszym krokiem jest uświadomienie sobie, jakiego rodzaju dane chcemy przetwarzać. Kolejnym logicznym krokiem jest rozważenie tego, jak te dane będą używane i przechowywane. Dodatkowo trzeba wiedzieć, jak T-SQL obsługuje obliczenia, w których uczestniczą różne typy danych.

Dokładne numeryczne typy danych

Zazwyczaj mamy do czynienia z liczbami, których wartości są ustalone i znane. Takie typy danych liczbowych możemy określić mianem dokładnych. Do przykładów należą prawda albo fałsz (1 lub 0), liczba sprzedanych produktów, procent rabatu albo pieniądze (dolary i centy). Wybór właściwej kategorii typu danych dla określonego pola jest kluczowy dla tworzenia dobrego kodu T-SQL. W niektórych przypadkach kategorie te mogą obejmować więcej niż jeden dostępny typ danych.

Przy rozważaniu typu danych do wykorzystania trzeba zastanowić się nad przeznaczeniem tych danych. Zapewni to łatwiejsze ustalenie, jaki typ danych powinien zostać użyty. Będziemy chcieli rozważyć zarówno korzyści, jak i ograniczenia powiązane z każdym typem danych. Zastanowimy się też, czy w wyniku użycia tego typu danych SQL Server będzie musiał dokonywać niejawnych konwersji typów przy wykonywaniu obliczeń wykorzystujących inne typy danych. Końcowym elementem do rozważenia jest to, jak ten typ danych jest przechowywany w SQL Server.

BIT

Termin BIT wywodzi się z określenia *binary digit* (cyfra dwójkowa, binarna). Oznacza to, że typ danych BIT może przechowywać tylko jedną z dwóch możliwych wartości – zero lub jeden. Jednak w przypadku SQL Server dostępna jest trzecia opcja, odnosząca się do wartości nieznanych (niedostępnych). Opcja ta nosi nazwę NULL. Trzeba podkreślić, że NULL nie jest zerem (ani pustym ciągiem). Sygnalizuje *brak informacji* o wartości, jaką ma zmienna lub pole tabeli danych. Podsumowując, jedyne wartości dozwolone dla typu danych BIT to 0, 1 oraz NULL. W konsekwencji sprawia to, że typ danych BIT ma najmniejszy zbiór dostępnych wartości.

Typ BIT jest doskonałym wyborem typu danych dla sytuacji, gdy mamy do czynienia z wyborem „albo X, albo Y". Przechowywaną informacją może być prawda lub fałsz, włączony lub wyłączony, tak lub nie i tak dalej. W przypadku wartości logicznych (prawda lub fałsz) typ BIT może zostać użyty do oznaczenia, czy rekord danych został poprawnie przetworzony. Typowym użyciem sygnalizowania włączenia lub wyłączenia (on/off) przy użyciu typu danych BIT jest pokazywanie, czy określona funkcjonalność jest aktywna. Przykładem zastosowania wartości tak/nie może być rejestrowanie decyzji klienta o akceptowaniu (lub nie) informacji marketingowych.

Stosowanie typu danych BIT wiąże się z wyzwaniem dotyczącym użycia go w taki sposób, który promuje dobry projekt bazy danych. Oznacza to, że będziemy mieli do czynienia z sytuacjami, w których trzeba będzie rozważyć ogólne przeznaczenie

przy wybieraniu typu danych BIT. Na przykład mogłoby się wydawać, że oznaczenie tego, czy dany element posiada szczególną cechę, jest dobrym wyborem użycia typu BIT. Przykładem mogłaby być taka kolumna tabeli, jak IsVegetarian (jest wegetarianinem/ką). Jednak lepszym wyborem może okazać się przeprojektowanie bazy danych, aby przechowywać takie atrybuty w oddzielnej tabeli. Typu BIT moglibyśmy zechcieć użyć do oznaczenia powodzenia transakcji. Jednak często mamy potrzebę rejestrowania większej liczby statusów transakcji z upływem czasu. Jeśli rejestrowanie zmian statusów w czasie jest istotne, wykorzystanie typu BIT do oznaczenia powodzenia transakcji nie będzie najlepszym wyborem.

Zaletą typu BIT jest oszczędność miejsca wymaganego na przechowywanie tych wartości w bazie danych. Jako że każdy bajt zawiera osiem bitów, to samo dotyczy przechowywania rekordów w bazie danych. Dla pierwszych 8 kolumn typu BIT w tabeli ich wartości będą przechowywane jako pojedynczy bajt (dla każdego rekordu). Gdyby tabela zawierała od 9 do 16 kolumn typu BIT, wszystkie te wartości będą przechowywane w dwóch bajtach. Tak mała wielkość wymaganej pamięci oznacza, że tabela z 8 kolumnami BIT i milionem rekordów zużyje zaledwie 1 MB.

TINYINT, SMALLINT, INT, BIGINT

SQL Server pozwala na przechowywanie liczb całkowitych, czyli takich, które nie mają części dziesiętnej (ułamkowej). Liczby takie określane są zbiorczym terminem *integer*. Przykładem danych przechowywanych jako liczby całkowite mogą być ilości produktów w magazynie. SQL Server udostępnia kilka typów liczb całkowitych, różniących się zakresem i ilością zajmowanego przez nie miejsca. Pierwszym z nich jest TINYINT. Wartość TINYINT może być dowolną liczbą całkowitą od 0 do 255 (włącznie). Ze względu na ograniczony zakres wartości tego typu może być on przydatny do identyfikowania lokalizacji lub (ograniczonych z góry) rodzajów konfiguracji. Ten typ danych jest podobny do typu BIT, ale z nieco szerszym zakresem wartości. Może być użyteczny do przechowywania różnych statusów albo kategorii obiektów, o ile nie potrzebujemy przechować więcej niż 256 statusów. Typ TINYINT zajmuje w pamięci jeden bajt.

Następnym po TINYINT typem danych całkowitoliczbowych jest SMALLINT. Zakres wartości typu SMALLINT obejmuje około 70 000 możliwych liczb. Mając taki zakres, możemy rozważyć użycie go dla informacji, które mogą mieć od 256 do 65 435 unikatowych wartości. Zakres wartości SMALLINT zaczyna się od –32 768 i kończy na 32 767. Wartości te przechowywane są w dwóch bajtach.

Ten typ danych nie będzie użyteczny dla tabeli, która rejestruje każdą pojedynczą aktywność. Wiele baz danych lub tabel danych łatwo przekroczy liczbę 70 000 transakcji lub rekordów w ciągu niewielu lat, a być może dni. Sprawia to, że ten typ nie nadaje się dla takich tabel. Niemniej jednak istnieją takie tabele, dla których typ danych SMALLINT będzie idealny jako identyfikator. Jeśli mamy tabelę danych, w której nie występuje intensywna aktywność transakcyjna, ale która będzie rosła przez jakiś czas, typ SMALLINT może okazać się bardzo korzystny. Dobra znajomość specyfiki biznesowej powinna pomóc w ustaleniu, czy SMALLINT jest właściwym typem danych dla przechowywanych danych.

Jeśli na przykład tworzymy tabelę rejestrującą dodawanie funkcjonalności do naszych aplikacji, będziemy chcieli w niej umieścić rekord wskazujący każdą nową funkcjonalność. Przykładem mogą być flagi funkcjonalności (*feature flag*). Nasza aplikacja zapewne będzie miała więcej niż 256 usprawnień w czasie swojego życia. Możemy również zechcieć przechować w tabeli wartości konfiguracyjne aplikacji. Przechowywanie takich wartości może być doskonałym zastosowaniem dla typu SMALLINT.

Kolejnym typem są „właściwe" liczby całkowite, czyli typ INT. Jest to najczęściej stosowany typ liczb całkowitych. Wiele baz danych używa wyłącznie tego typu danych dla dowolnego rodzaju liczb całkowitych (liczby rekordów, numeru porządkowego zdarzeń i tak dalej). Jednym z powodów jest to, że całkowity zakres wartości tego typu to około 4,3 miliarda wartości. Typ danych INT obejmuje zakres od –2 147 483 648 do 2 147 483 647. Niemniej jednak przy tworzeniu większości tabel danych ich kolumna identyfikatora jest zwykle inicjowana liczbą 1. Powoduje to, że wielkość tabeli jest ograniczona do około 2,15 miliarda unikatowych tożsamości. Typ INT zajmuje 4 bajty pamięci.

Jeśli spodziewamy się, że tabela może potrzebować więcej niż 2,15 miliarda unikatowych identyfikatorów (ale nie więcej, niż 4 miliardy), możemy rozpocząć wartość pola tożsamości od najmniejszej możliwej liczby, czyli –2 147 483 648. W przeciwnym razie musimy sięgnąć po typ BIGINT, który wykorzystuje 8 bajtów do przechowywania wartości i obejmuje ponad 18 trylionów wartości (od -2^{63} do $2^{63}-1$).

W tym miejscu często trzeba dokonać trochę obliczeń. Niektóre firmy przetwarzają kilka do kilkuset transakcji na sekundę. Inne sięgają poziomu 10 000 lub 20 000 transakcji na sekundę. W obu przypadkach trzeba się zastanowić, jakiego poziomu wzrostu można oczekiwać w tabelach przechowujących takie informacje transakcyjne. Jeśli aplikacja przetwarza kilkadziesiąt czy kilkaset transakcji na sekundę przez kilka lat, liczba zapisanych rekordów będzie znacznie mniejsza, niż w przypadku dziesiątków tysięcy transakcji na sekundę w tym samym okresie.

DECIMAL/NUMERIC

Po omówieniu różnych typów całkowitoliczbowych powinniśmy zastanowić się, jak mamy traktować liczby wymagające części dziesiętnej. Istnieje wiele przypadków, w których potrzebujemy przechować wartości ułamkowe. Jedne z najczęstszych przypadków to używanie walut, ale możemy też potrzebować więcej miejsc dziesiętnych dla precyzyjnych pomiarów. W takich przypadkach mamy do dyspozycji kilka opcji.

Pierwszą z nich jest typ DECIMAL, alternatywnie nazywany typem NUMERIC. Wartość ta nie zawiera żadnych informacji o walucie; tym niemniej rejestruje miejsca dziesiętne. Miejsca te (czyli dokładność) można wyspecyfikować, wskazując całkowitą liczbę cyfr oraz liczbę cyfr na prawo od znaku dziesiętnego. Jak się okazuje, typ danych DECIMAL/NUMERIC jest akceptowalny dla niemal dowolnych typów danych wykorzystujących liczby. Należą do nich liczby ogólnego stosowania, ułamki, wyniki pomiarów czy wartości monetarne.

Biorąc pod uwagę to, że typ DECIMAL może w istocie reprezentować wiele typów liczb, powinniśmy przyjrzeć mu się bliżej. Warto podkreślić, że nie ma żadnej różnicy pomiędzy typem DECIMAL i NUMERIC. W SQL Server są one tym samym. Istnieje kilka specjalnych pojęć dotyczących typu DECIMAL. Wartości budujące typ DECIMAL to dokładność (*precision*) oraz skala (*scale*). Dokładność odnosi się do całkowitej liczby cyfr zapisywanych w typie danych DECIMAL. Skala dotyczy liczby cyfr, które są przechowywane na prawo od punktu dziesiętnego.

SMALLMONEY, MONEY

Kolejne typy danych, które trzeba omówić, są MONEY i SMALLMONEY. Typy te są podobne do typu DECIMAL/NUMERIC, jednak są przeznaczone specjalnie do przechowywania wartości walutowych. SQL Server przechowa wartość numeryczną, oddzielając typ waluty powiązanej z zapisaną wartością.

Główna różnica pomiędzy typami MONEY i SMALLMONEY leży w ich zakresie oraz zajmowanej pamięci. Typ danych SMALLMONEY obejmuje zakres od –214 000 do 214 000 i zajmuje tylko 4 bajty danych, podczas gdy typ MONEY obejmuje zakres od –922 miliardów do 922 miliardów i zużywa 8 bajtów.

Typ MONEY przechowuje dokładną wartość do czterech miejsc dziesiętnych. Limit liczby miejsc dziesiętnych ogranicza dokładność do jednej dziesięciotysięcznej przechowywanej wartości monetarnej. Ustalona liczba miejsc dziesiętnych ma wpływ na zaokrąglenia powstające w obliczeniach, w których uczestniczą dane typu MONEY.

Przybliżone typy danych liczbowych

Przejdziemy teraz do omówienia różnic pomiędzy liczbami dokładnymi a przybliżonymi. Dokładne liczby występują w sytuacjach, gdy znamy dokładną wartość, na przykład ile produktów kupiliśmy w sklepie albo dokładną kwotę w dolarach i centach. Liczby przybliżone są użyteczne w scenariuszach, w których pomiary mogą nie być dokładne. Przybliżone liczby mogą służyć do przechowywania zarówno bardzo wielkich, jak i bardzo małych liczb. Możemy również napotkać sytuacje, w których aplikacja rejestruje pomiary, które z samej natury nie są dokładne. Możemy na przykład odcinać fragmenty tkaniny o długości bliskiej, ale nie dokładnie równej określonej wartości. Przykładowo długość ta może wynosić około 12 cali. Przechowanie wartości równej 12 będzie wówczas wartością przybliżoną. Ten pomiar może być tak bliski rzeczywistemu, że trudno byłoby stwierdzić, że długość ta nie była dokładnie równa 12 calom.

Przy posługiwaniu się liczbami przybliżonymi w grę wchodzą różne zagadnienia związane z zaokrągleniami. Wynika to stąd, że liczby takie z założenia nie są traktowane jako dokładne. W systemie SQL Server dostępny jest jeden tylko typ liczb przybliżonych. Nosi on nazwę FLOAT. Jeśli liczba zmiennoprzecinkowa zawiera 24 cyfry, synonimiczny typ danych nosi nazwę REAL.

Przy korzystaniu z liczb typów REAL lub FLOAT mogą występować problemy związane z konwertowaniem tego typu na inne typy danych. Przy konwertowaniu typu danych FLOAT na dowolny typ całkowitoliczbowy (np. INT) wszystkie cyfry części dziesiętnej zostaną odrzucone. Trzeba mieć świadomość, że używanie liczb przybliżonych może prowadzić do nieoczekiwanych wyników. Przykładem jest używanie ich w połączeniu z typami DECIMAL lub NUMERIC. Podczas konwertowania liczby typu FLOAT lub REAL na typ DECIMAL lub NUMERIC możemy zachować co najwyżej siedem miejsc dziesiętnych.

Konwertowanie numerycznych typów danych

Omówiliśmy już wszystkie dostępne typy danych liczbowych, ale oprócz ich przechowywania ważne jest również, jak się zachowują podczas interakcji ze sobą. Najpierw należy się zastanowić, co się stanie, gdy wykonujemy obliczenia na polach danych tego samego typu. W takim scenariuszu, jeśli wszystkie pola w obliczeniu mają ten sam typ danych, wynik również będzie miał ten sam typ. Tym samym, jeśli pomnożymy ilość przez cenę i obie wartości są przechowywane jako NUMERIC typu DECIMAL(5,2), wynik będzie miał ten sam typ danych DECIMAL. SQL Server ustali dokładność i skalę

wyniku na podstawie dokładności i skali danych wejściowych, ale również na podstawie typu wykonywanych obliczeń. Trzeba również uwzględnić to, jak dokładność i skala są przechowywane w aplikacji. Może ona oczekiwać wartości typu DECIMAL(5,2). Zależnie od wykonywanych obliczeń, zwracana wartość może wykraczać poza zakres wskazanego typu danych, co może prowadzić do przepełnienia numerycznego.

Sprawy stają się bardziej interesujące, gdy pracujemy z różnymi typami danych. Ponownie używając przykładu z mnożeniem ilości produktów przez cenę, możemy zbadać, co się stanie, gdy pomnożymy liczbę typu INT przez liczbę typu DECIMAL(5,2). SQL Server użyje wówczas *pierwszeństwa typów danych*. Na początek musimy poznać kolejność pierwszeństwa typów danych, które poznaliśmy do tej pory. Poniższa lista jest uporządkowana od najwyższego priorytetu do najniższego.

1. FLOAT
2. REAL
3. DECIMAL
4. MONEY
5. SMALLMONEY
6. MONEY
7. BIGINT
8. INT
9. SMALLINT
10. TINYINT
11. BIT

W omawianym przykładzie używamy typów INT i DECIMAL. W tym przypadku typ INT ma niższe pierwszeństwo. Zgodnie z porządkiem pierwszeństwa, SQL Server wewnętrznie przekształci typ danych INT na typ DECIMAL. Konwersja ta nie zmienia oryginalnej wartości danych, a jedynie sposób jej użycia jako części obliczeń. Po wykonaniu konwersji SQL Server przechodzi do wykonywania obliczeń. Działa to doskonale, o ile nie będziemy próbowali takich działań, jak złączanie (konkatenacja) liczby z łańcuchowymi typami danych.

Łańcuchowe typy danych (string)

Po poznaniu podstaw numerycznych typów danych poświęcimy teraz nieco czasu na różne typy danych łańcuchowych. Te rodzaje typów danych służą do przechowywania danych alfanumerycznych – liter, słów lub kombinacji słów i liter. Dodatkowo łańcuchowe typy danych służą do przechowywania danych nie-Unicode lub Unicode. Ostatnia kategoria danych łańcuchowych zawiera typy przeznaczone do przechowywania obrazów i innych wartości binarnych.

Łańcuchy znakowe

W każdej bazie danych znajdziemy informacje, które nie są bezpośrednio powiązane z liczbami. Danymi takimi mogą być nazwy, opisy, adresy lub inne dane znakowe. Ustalenie używanego typu zależy od tego, jakiego rodzaju informacje są przechowywane i jak dużo informacji musimy przechować.

CHAR oraz VARCHAR

Dwa podstawowe typy danych łańcuchów znakowych to CHAR i VARCHAR. Typy te są podobne i różnią się tylko pewnymi szczególnymi cechami. Pole danych można skonfigurować, aby określić sposób przechowywania danych. Kolumny tabel można skonfigurować, aby włączać lub wyłączać rozróżnialność wielkości liter (*case sensitivity*), liter akcentowanych (*accent*), rozróżnialność kany albo szerokość (liczbę bajtów używaną do zapisu jednego znaku). Kombinację tych rozróżnialności nazywamy *collation* (sortowanie*). Typ danych, który można przechować, zazwyczaj pasuje do ustawienia *collation* bazy danych. *Collation* kolumny jest takie samo, jak bazy danych, o ile nie zostanie jawnie zmienione. Obydwa typy danych służą do przechowywania danych tekstowych i znaki, które można przechować w tych polach, są tymi samymi znakami, które są dozwolone przez ustawienie *collation* kolumny.

Przy ustalaniu, którego typu danych użyć, trzeba wziąć pod uwagę, jakiego rodzaju dane będą przechowywane. Jeśli dane te mają podobną długość, jak w przypadku numerów telefonów lub kodów pocztowych, preferowanym typem może być typ CHAR. Jeśli jednak szerokość pól kolumny może się zmieniać znacząco, jak w przypadku adresów albo kolumn notatek, lepszą opcją będzie wybór typu VARCHAR. Warto ograniczyć stosowanie typu VARCHAR(MAX) tylko do sytuacji, w których spodziewamy się, że będziemy musieli przechować więcej niż 8000 znaków w jednym polu. Przy wyspecyfikowaniu typu kolumny jako VARCHAR(MAX) maksymalna wielkość wynosi 2 GB.

Istnieje jeszcze kilka innych uwarunkowań, które trzeba mieć na uwadze przy korzystaniu z typów CHAR i VARCHAR. Gdy używamy CHAR i VARCHAR do definiowania danych lub deklarowania zmiennych, trzeba pamiętać, że wartością domyślną jest jeden znak. Niemniej jednak przy korzystaniu z funkcji CAST lub CONVERT domyślną liczbą znaków będzie 30. Aby zminimalizować ryzyko obcinania danych, trzeba zadbać

---

* Termin ten występuje w polskiej dokumentacji SQL Server, jest jednak mylący, gdyż collation określa znacznie więcej cech, niż tylko sposób sortowania danych znakowych. Z tego względu w powszechnym użyciu jest termin angielski. (przyp. tłum.)

o to, aby zawsze jawnie specyfikować liczbę znaków przy korzystaniu z typów danych CHAR lub VARCHAR.

W przypadku *collation* używającego jednobajtowego kodowania znaków, takiego jak Latin, wielkość pamięci w bajtach dla typu CHAR jest równa zadeklarowanej liczbie znaków. W przypadku VARCHAR liczba przechowywanych bajtów jest równa rzeczywistej liczbie znaków plus 2 dodatkowe bajty. Możliwe jest również przechowywanie w typach CHAR i VARCHAR danych znakowych o kodowaniu wielobajtowym. W tej sytuacji w obu typach danych liczba znaków będzie mniejsza od całkowitej liczby przechowywanych bajtów.

Począwszy od wersji SQL Server 2019 możliwe jest przechowywanie wartości typu Unicode w polach typów CHAR lub VARCHAR. Jednak jest to możliwe tylko pod warunkiem włączenia kodowania UTF-8.

TEXT

Typ danych TEXT był dawniej używany do przechowywania bardzo długich łańcuchów znaków. Obecnie jednak ten typ danych jest uważany za przestarzały. Można się spodziewać, że jego obsługa zostanie wycofana w przyszłości w którejś wersji oprogramowania, a tym samym powinniśmy unikać stosowania typu danych TEXT w nowych projektach. Jeśli ktoś odczuwa potrzebę użycia typu danych TEXT, powinien rozważyć zastosowanie typu danych VARCHAR(MAX) w nowych rozwiązaniach. Zawsze jednak trzeba się zastanowić, czy w ogóle potrzebujemy tej funkcjonalności albo czy nie wystarczyłoby użycie typu VARCHAR z mniejszą liczbą znaków. Jedynym powodem stosowania typu TEXT powinno być zapewnie wstecznej kompatybilności dla istniejących aplikacji. Typ danych TEXT jest głównie używany w sytuacjach, w których występują bardzo długie łańcuchy tekstowe o zmiennej długości. Zazwyczaj będzie tak wtedy, gdy długość pojedynczego zapisywanego łańcucha przekracza 8000 znaków. Maksymalna liczba znaków, którą można zapisać w typie danych TEXT, wynosi 2 147 483 647 (2GB – 1). Zazwyczaj jednak będziemy mieli do czynienia z sytuacjami, gdy całkowita liczba znaków, które potrzebujemy zapisać, jest znacznie mniejsza od tej liczby.

Typy danych łańcuchowych Unicode

Przed wprowadzeniem wersji SQL Server 2019 dowolne dane tekstowe Unicode musiały być przechowywane jako specjalny typ danych. Nadal jest to prawdą, jeśli w instancji serwera nie jest włączona obsługa kodowania UTF-8.

NCHAR i NVARCHAR

Dostępnych jest kilka opcji używania wartości tekstowych zapisanych jako Unicode. Obejmują one przechowywanie łańcuchów o ustalonej lub zmiennej długości. Aby uniknąć nieoczekiwanych skutków, trzeba wiedzieć, jak działają te typy danych, jeśli nie zostanie wyspecyfikowana liczba znaków lub *collation*.

Po stwierdzeniu, że potrzebujemy użyć typu danych NCHAR lub NVARCHAR, wybór pomiędzy nimi jest prosty. Jeśli przechowywane dane będą miały w ogólności podobną długość, właściwym wyborem będzie typ danych NCHAR. Jeśli jednak zapisywane wartości będą się znacząco zmieniać, lepszym wyborem będzie typ NVARCHAR. Ponadto jeśli liczba znaków do przechowania przekracza 4000, zalecane jest użycie typu NVARCHAR(MAX).

W typowej sytuacji najlepszą praktyką jest wyspecyfikowanie liczby znaków przy deklarowaniu typu danych NCHAR lub NVARCHAR. Domyślna liczba znaków dla definicji danych lub zmiennej wynosi jeden znak dla obu typów NCHAR oraz NVARCHAR. Jednak przy korzystaniu z funkcji CAST lub CONVERT domyślna liczba znaków wynosi 30, jeśli nie zostanie jawnie wyspecyfikowana. Jeśli nie określimy *collation* dla typu NCHAR lub NVARCHAR, zostanie użyte domyślne ustawienie bazy danych.

Znajomość wielkości miejsca wymaganego do przechowywania tego typu danych również pomaga w podejmowaniu lepszych decyzji o tym, czy jest to właściwy typ danych. Przechowanie danych NCHAR wymaga dwukrotnie większej liczby bajtów, niż zadeklarowana długość łańcucha znakowego, podczas gdy w przypadku NVARCHAR liczba zapisywanych bajtów to podwojenie rzeczywistej długości łańcucha plus 2 bajty.

NTEXT

Do niedawna przechowywanie obszernych danych Unicode o zmiennej długości było realizowane przy użyciu typu danych NTEXT. Jeśli ten typ nadal jest w użyciu w naszych systemach, możemy oczekiwać, że pomieści on do 1 073 741 823 znaków. Jednak ze względu na zmienną wielkość całkowita przechowywana długość może być mniejsza. Podobnie jak w przypadku typu TEXT, obecnie zalecane jest stosowanie w takich sytuacji typu danych NVARCHAR(MAX).

Binarne typy danych łańcuchowych

W pewnym momencie może się okazać, że potrzebujemy przechować dane, które nie są ani liczbami, ani tekstem. W takich przypadkach odpowiednie będzie użycie łańcucha

binarnego. Podobnie jak w przypadku łańcuchów tekstowych, mamy do dyspozycji kilka alternatywnych typów.

BINARY i VARBINARY

Dostępne opcje przechowywania binarnych danych łańcuchowych obejmują zapisywanie łańcuchów o ustalonej lub zmiennej długości. Podobnie jak w przypadku innych typów łańcuchowych, stosowanie tych typów wiąże się z kilkoma uwarunkowaniami.

Łańcuchy binarne przeznaczone są do przechowywania elementów danych, które są łańcuchami, ale nie są tekstem. Należą do nich pliki audio, wideo, obrazy i inne podobne obiekty. Podstawowe typy danych tego rodzaju to BINARY i VARBINARY. Najlepszym wyborem dla przechowywania łańcuchów binarnych o zbliżonej długości jest typ danych BINARY. I odwrotnie, jeśli zamierzamy przechowywać łańcuchy o znacząco zmieniającej się długości, lepszym wyborem będzie typ danych VARBINARY. Jeśli oczekujemy, że całkowita długość binarnego łańcucha przekroczy 8000 bajtów, zalecane jest użycie typu VARBINARY(MAX).

Typ danych BINARY i VARBINARY przy definiowaniu kolumny danych lub zmiennej domyślnie przyjmuje długość równą jeden, jeśli długość nie zostanie wyspecyfikowana. Przy konwertowaniu typu BINARY na VARBINARY przy użyciu funkcji CAST lub CONVERT domyślna długość wynosi 30. Należy zachować ostrożność przy konwertowaniu danych o zmiennej długości na typ BINARY lub VARBINARY, gdyż SQL Server może dopełnić lub obciąć dane binarne w razie potrzeby.

Typ BINARY przechowuje tę samą liczbę bajtów, co wielkość zapisywanych danych, podczas gdy typ VARBINARY używa dodatkowo 2 bajtów określających bieżącą długość danych. W obu typach długość może wynosić do 8000 bajtów. W przypadku typu VARBINARY(MAX) maksymalna wielkość przechowywanego obiektu to 2 GB.

IMAGE

Jednym z rodzajów obiektów, które można przechowywać jako łańcuch binarny, jest obraz. Przy korzystaniu z obrazu ważne jest uwzględnienie tego, jak dane powinny być przechowywane i jak ten typ danych powinien być używany.

Typ danych IMAGE służy do przechowywania wielkich danych binarnych o zmiennej długości. Choć ten typ danych może mieć długość do 2 147 483 647 bajtów, istnieją sytuacje, w których dopuszczalna długość będzie mniejsza. Warto zauważyć, że typ danych IMAGE jest uważany za przestarzały i w przyszłości należy zamiast niego stosować typ VARBINARY(MAX).

Typ danych daty i czasu

Każda transakcja bazodanowa następuje w określonym punkcie w czasie. Często potrzebujemy odniesienia do tego, kiedy transakcja nastąpiła. Nasza aplikacja może też rejestrować daty ważne dla użytkowników, takie jak urodziny lub rocznice. Daty i czas mogą być używane również do określania cen lub dostępnych funkcjonalności. Używając dat i godzin można określić, kiedy pewna funkcjonalność powinna być włączona lub wyłączona. Daty i czas pokazują również, kiedy konto użytkownika jest nieaktywne lub kiedy możliwy jest dostęp do określonego systemu. Stawki używane w cenach i opłatach mogą odnosić się do wielu różnych zakresów danych. Gdy jeden zestaw cen staje się nieaktywny, uaktywniany jest inny zestaw. Zgodnie z wymogami prawnymi firma może potrzebować rejestrować ceny przez dłuższy czas. Obejmuje to wskazanie, kiedy określona cena zaczęła i kiedy przestała być stosowana. Zależnie od celów śledzenia takich informacji, możemy potrzebować znać jedynie datę albo jedynie czas (godzinę) transakcji. Istnieją też inne sytuacje, w których musimy znać zarówno datę, jak i dokładny czas powiązany z określonym działaniem.

DATE

Przy obsługiwaniu transakcji z zasady zachodzi potrzeba rejestrowania, kiedy coś się wydarzyło. W wielu przypadkach istotne jest tylko to, którego dnia wystąpiła dana transakcja. Typ danych DATE może również posłużyć do przechowywania danych zagregowanych dla określonego dnia. Podczas rejestrowania daty zdarzenia dostępne są również opcje dotyczące tego, jak te dane są wyświetlane. Podczas decydowania, czy typ danych DATE jest odpowiedni, trzeba rozważyć nie tylko to, jak wiele informacji jest przechowywanych w tym typie danych, ale również możliwe ograniczenia zapisywania tych danych. W niektórych przypadkach łatwiej będzie myśleć o tym, kiedy typ danych DATE nie jest preferowany. Dla dowolnych działań, w których potrzebujemy znać dokładny czas zdarzenia, typ danych DATE nie jest dobrym wyborem. Jeśli jednak musimy tylko wiedzieć, którego dnia wystąpiło dane działanie, typ DATE będzie doskonały.

Dostępnych jest kilka możliwości wyświetlania wartości typu DATE. Domyślnym formatem daty jest YYYY-MM-DD. W tym przypadku YYYY reprezentuje czterocyfrowy rok w zakresie od 0001 do 9999. MM oznacza numer miesiąca od 01 do 12, zaś DD numer dnia, od 01 do 31 (oczywiście odpowiednio dla liczby dni w konkretnym miesiącu). Jednak typ DATE można wyświetlać w rozmaitych formatach numerycznych i alfabetycznych. Warto jednak zauważyć, że format YDM (rok, dzień, miesiąc) nie jest obsługiwany.

Typ DATE może przechowywać wartości od 0001-01-01 do 9999-12-31. Typ DATE ma długość 10 znaków, zajmuje do 3 bajtów i jest wewnętrznie przechowywany jako liczba całkowita (INT).

Daty mogą być konwertowane na typy DATETIME, SMALLDATETIME, DATETIME2 lub DATETIMEOFFSET. W takim przypadku część odpowiadająca za czas zostanie ustawiona na północ (0:00:00). Nie jest natomiast możliwa konwersja typu DATE na typ TIME i dowolna próba takiej konwersji (jawnej lub niejawnej) zakończy się niepowodzeniem i zgłoszeniem błędu. Dodatkowo daty nie zawierają informacji o przesunięciu strefy czasowej ani o zmianach na czas letni/zimowy.

TIME

Kolejnym typem danych związanym z momentem, w którym nastąpiło działanie, jest typ TIME. Ważne jest zrozumienie, jak czas jest przechowywany i jego formatowanie przy wyświetlaniu. Istotne są również implikacje konwertowania tego typu danych na inne typy dat i czasu. Istnieją również pewne ograniczenia związane ze stosowaniem typu TIME.

Typ TIME służy do rejestrowania dokładnego czasu (momentu), w którym nastąpiła transakcja lub inne działanie. W tym przypadku czas jest rejestrowany niezależnie od daty i ustalenie dnia może być niewykonalne. Jednym ze sposobów obejścia tego problemu może być przechowanie daty niezależnie od czasu. Dokładność typu TIME wynosi 100 nanosekund, zaś domyślna wartość to 00:00:00.

Domyślny format dla typu TIME to hh:mm:ss[.nnnnnnn]. W tym formacie hh oznacza dwucyfrową godzinę od 00 do 23, mm to dwucyfrowa wartość minut od 00 do 59, zaś ss do dwucyfrowa wartość sekund, ponownie od 00 do 59. Typ danych TIME pozwala na zmienną dokładność i jeśli zostanie określona, używanych jest do siedmiu cyfr dziesiętnych dla ułamków sekundy, reprezentowanych jako nnnnnnn.*

Przy korzystaniu z zapisu dwunastogodzinnego z sufiksami AM lub PM dla odróżnienia poranka i popołudnia trzeba uwzględnić dodatkowe uwarunkowania. Jeśli sufiks AM lub PM nie został podany i wartość godziny należy do przedziału 00 do 11, czas zostanie zarejestrowany jako AM. Dla godzin od 12 do 23 czas będzie zapisany jako PM. Jeśli jako typ TIME wprowadzona zostanie wartość 12 AM, zostanie ona przekonwertowana na godzinę 0.

---

* W języku SQL znakiem dziesiętnym zawsze jest kropka, niezależnie od ustawień językowych używanego systemu operacyjnego – z tego względu wszystkie liczby ułamkowe występujące w książce są zapisywane z kropką jako znakiem dziesiętnym.

Zakres wartości typu TIME to 00:00:00.0000000 do 23:59:59.9999999. Długość w znakach może się zmieniać od 8 do 16 cyfr, zależnie od wyspecyfikowanej dokładności. W każdym przypadku jednak typ TIME jest zapisywany w stałej długości 5 bajtów. Jeśli typ TIME zostanie przekonwertowany na dowolny typ zawierający datę i czas, wartość dnia będzie reprezentowana jako 1900-01-01. Jeśli dokładność ułamkowa typu TIME jest większa, niż nowego (docelowego) typu danych, wartość ułamkowa zostanie obcięta. Dowolna próba konwersji typu TIME na typ DATE zakończy się niepowodzeniem. Podobnie jak typ DATE, typ TIME nie uwzględnia ani strefy czasowej, ani czasu letniego/zimowego.

SMALLDATETIME, DATETIME, DATETIME2, DATETIMEOFFSET

Często mamy do czynienia z sytuacją, w której przechowanie samej daty lub samego czasu jest niewystarczające. W takich scenariuszach najlepsze może być połączenie razem wartości daty i czasu. Niekiedy takie wartości mogą być w jakiś sposób prostsze, zapewniać większą dokładność albo uwzględniać strefy czasowe.

Najprostszym typem tego rodzaju jest SMALLDATETIME. Typ ten rejestruje zarówno czas, jak i datę. Domyślna wartość to 1900-01-01 00:00:00. Choć dokładność tego typu danych jest równa jednej sekundzie, trzeba pamiętać, że wartość sekund zostanie zawsze zapisana w bazie danych jako 00. Inaczej mówiąc, choć można wprowadzać dane zawierające sekundy, a nawet wykonywać na nich obliczenia, w chwili zapisu w bazie danych zostaną one zaokrąglone do najbliższej pełnej minuty.

Podobnie jak w przypadku typu DATE, typ SMALLDATETIME może być wyświetlany w różnych formatach alfanumerycznych. Zakres wartości typu SMALLDATETIME jest dość ograniczony, jeśli porównać go z innymi typami dat i czasu. Część odpowiadająca za dzień rozciąga się od 1900-01-01 do 2079-06-06. Choć jak wspomnieliśmy, wprowadzana wartość czasu może zmieniać się od 00:00:00 do 23:59:59, w bazie danych zostanie zapisana wartość od 00:00:00 do 23:59:00. Całkowita długość typu SMALLDATETIME wynosi do 19 znaków, zaś wielkość wymaganej pamięci to 4 bajty (stała).

Przy konwertowaniu typu SMALLDATETIME na inne typy DATETIME trzeba pamiętać, że dodatkowe cyfry dokładności zostaną wypełnione zerami. Trzeba też podkreślić, że typ SMALLDATETIME nie jest zgodny ze standardem ANSI. Podobnie jak typy DATE i TIME, typ SMALLDATETIME nie zawiera informacji o strefie czasowej ani czasie letnim/zimowym.

Najczęściej wybieranym typem kombinowanym jest typ DATETIME, zapewniający większą dokładność, niż wcześniej wspomniane typy danych. Istnieje jednak kilka ważnych uwarunkowań stosowania tego typu danych.

Choć typ danych DATETIME może rejestrować datę i czas, nie jest on w pełni zgodny ze standardem ANSI SQL. Jednym z głównych problemów związanych z tym typem danych jest ograniczona dokładność. Typ danych DATETIME może rejestrować jedynie trzy miejsca dziesiętne jako ułamkowa część sekundy; trzecie miejsce dziesiętne jest zawsze zaokrąglane do wartości kończącej się na 0, 3 lub 7. Inaczej mówiąc, dokładność tego typu danych wynosi około 1/300 sekundy.

Domyślna wartość typu DATETIME to 1900-01-01 00:00:00. Jak poprzednio, typ ten udostępnia możliwość wyświetlania danych w wielu formatach alfanumerycznych. Zakres dat dla typu DATETIME rozciąga się od 1753-01-01 do 2999-12-31, zaś część czasowa może się zmieniać od 00:00:00.000 do 23:59:59.997. Wielkość tego typu danych w pamięci (i w bazie danych) to 8 bajtów, podczas gdy wielkość wyświetlana może mieć długość od 19 do 26 znaków.

Choć możliwa jest konwersja innych typów danych dat i czasu na typ DATETIME, nie jest to zalecane, jako że ten typ danych nie jest zgodny ze standardem SQL określonym w normie ANSI. Jak wspomnieliśmy wcześniej, DATETIME ma również dokładność ograniczoną do 1/300 sekundy. Ten typ danych nie uwzględnia także informacji o strefie czasowej ani czasie letnim/zimowym.

Typ danych DATETIME2 ma kilka zalet względem wcześniej przedstawionych typów. Podczas gdy tamte typy danych miały ustalony rozmiar, ten typ ma rozmiar zależny od wybranej dokładności. Przyjrzymy się również opcjom przechowywania i formatowania dostępnym dla tego typu danych.

Typ DATETIME2 pozwala na rejestrowanie daty i czasu z dokładnością do 100 nanosekund. Domyślna wartość typu DATETIME2 to 1900-01-01 00:00:00. Ze względu na dostępną dokładność jest to doskonały typ danych dla scenariuszy, w których musimy znać czas z dokładnością do ułamków sekund. Jako że DATETIME2 nie wykonuje zaokrągleń podobnych do występujących w typie DATETIME, łatwiejsze jest posługiwanie się tym typem przy pisaniu kodu.

DATETIME2 można taktować jako połączenie typów DATE i TIME. Zakres dat dla tego typu rozciąga się od 0001-01-01 do 9999-12-31, a zakres czasu od 00:00:00 do 23:59:59.9999999 (przy najwyższym poziomie dokładności). Dostępnych jest kilka poziomów dokładności, przez co długość typu zmienia się od 19 znaków dla precyzji wynoszącej 1 sekundę (zero miejsc dziesiętnych) aż po 27 znaków dla dokładności wynoszącej 0.0000001 sekundy.

Zmiany dokładności wpływają również na wielkość pamięci zajmowanej przez typ DATETIME2. Jeden bajt służy do przechowania dokładności typu DATETIME2 i jest uzupełniony bajtami niezbędnymi do zapisania wartości. Jeśli dokładność jest mniejsza

niż trzy miejsca dziesiętne sekund, potrzeba 6 bajtów danych do przechowania wartości typu DATETIME2 (łącznie siedem bajtów). Jeśli dokładność wynosi 3 lub 4 miejsca dziesiętne, wartość przechowywana jest w 7 bajtach, co razem daje 8 bajtów. Wszystkie wyższe dokładności (od 5 do 7 miejsc dziesiętnych) wymagają 8 bajtów, co łącznie daje 9 zajmowanej pamięci. Domyślna dokładność to 7 cyfr dziesiętnych.

Ze względu na wysoką dokładność konwertowanie różnych wartości na typ danych DATETIME2 jest bardzo prawdopodobne. Przy konwertowaniu na ten typ samej daty (typu DATE) składnik czasowy jest rejestrowany jako 00:00:00. W przypadku konwersji typu TIME na typ DATETIME2 dzień zostanie zapisany jako 1900-01-01. W przypadku konwersji typu SMALLDATETIME na typ DATETIME2 wartości daty i czas są po prostu kopiowane, zaś dodatkowe cyfry wynikające z wybranej dokładności są wypełniane zerami. Przy przechodzeniu z typu DATETIME do DATETIME2 trzeba zadbać o wykonanie jawnej konwersji, aby uniknąć nieoczekiwanych wyników. Głównym ograniczeniem typu DATETIME2 jest to, że typ ten nie zawiera informacji o strefie czasowej i czasie letnim. Przejście od typu DATETIMEOFFSET do DATETIME2 powoduje odrzucenie informacji o strefie czasowej.

Ostatnim typem danych dla dat i czasu jest DATETIMEOFFSET. Typ ten zawiera pewne dodatkowe funkcjonalności, które nie występują w innych typach danych. Typ ten rejestruje datę i czas z wysoką dokładnością. Jedną z głównych zalet tego typu jest przechowywanie informacji o przesunięcie (offsecie) czasu, dzięki czemu bazy danych w różnych lokalizacjach geograficznych mogą być świadome nie tylko tego, kiedy coś nastąpiło względem ich lokalnego czasu, ale również względem czasu w innej lokalizacji.

Dokładność i zakres wartości typu DATETIMEOFFSET jest identyczny, jak typu DATETIME2. Domyślny format typu DATETIMEOFFSET to YYYY-MM-DD hh:mm:ss.nnnnnnn +|– hh:mm. Część +|– hh:mm tego formatu odpowiada offsetowi (przesunięciu czasu) względem UTC, czyli czasu uniwersalnego. Offset może mieć wartość od +14 do –14 i określa liczbę godzin (i potencjalnie minut – tak, istnieją na świecie strefy czasowe, których przesunięcie względem UTC nie stanowi pełnej liczby godzin), o który przesunięta jest dana strefa czasowa. Podobnie jak inne typy daty i czasu, wartość daty może być wyświetlana w wielu formatach alfanumerycznych. Przy dokładności równej sekundzie (zero miejsc dziesiętnych) długość typu wynosi 26 znaków. Rośnie ona do 34 znaków, gdy dokładność wynosi 7 znaków dziesiętnych. Wielkość pamięci zajmowanej przez ten typ jest o 1 bajt większa, niż odpowiedniego typu DATETIME2 i maksymalnie wynosi 10 bajtów.

Inne typy danych

Oprócz omówionych dotąd SQL Server udostępnia szereg innych typów danych. Niektóre z nich mogą być używane w definicjach tabel i mają specjalne przeznaczenie, podczas gdy inne są użyteczne tylko jako zmienne lub wewnątrz procedur składowanych.

UNIQUEIDENTIFIER

Ten typ danych może być kolumną w tabeli albo zostać użyty jako zmienna. UNIQUEIDENTIFIER zajmuje 16 bajtów, a maksymalna liczba znaków, która może w nim zostać zapisana, wynosi 36. Choć łańcuchy tekstowe nie-Unicode można przekonwertować na typ UNIQUEIDENTIFIER, zostaną obcięte, jeśli całkowita liczba znaków przekracza 36.

Ten typ danych to GUID, czyli *Globally Unique Identifier* (globalnie unikatowy identyfikator). Koncepcja opiera się na tym, że są to unikatowe wartości, które mogą być użyte tylko raz. Tym niemniej zgłaszane były przypadki, w których stwierdzenie to okazało się nieprawdziwe. Tak czy inaczej, UNIQUEIDENTIFIER może być wypełniany jednym z wielu sposobów. Należą do nich funkcje NEWID() oraz NEWSEQUENTIALID(). W przeciwnym razie wartości te mogą być wypełniane ręcznie, o ile ogólny format GUID jest poprawny i używane są właściwe cyfry szesnastkowe, czyli 0–9 oraz a–f.

Choć typu UNIQUEIDENTIFIER można używać jako IDENTITY, zalecany jest tylko w scenariuszach, w których jest to bezwzględnie konieczne. Nie tylko zajmuje znacząco więcej miejsca, niż INT lub BIGINT, ale UNIQUEIDENTIFIER ma limitowaną stosowalność ograniczeń, których można używać względem tego typu. Kolumna UNIQUEIDENTIFER może być kolumną IDENTITY, ale inne ograniczenia tabeli nie będą dozwolone.

XML

Rozmaite systemy i aplikacje wysyłają, używają lub przechowują dane w postaci XML. Choć istnieją możliwości analizowania tych danych i przechowywania ich w tabelach, występują sytuacje, w których konieczne jest przechowanie nietkniętych danych XML. Przy przechowywaniu danych XML występują inne uwarunkowania dotyczące tego, jakie dane znajdują się w tym XML.

Dla typu XML dane muszą mieć poprawny format. Istnieje wiele warunków, które muszą zostać spełnione, aby format był poprawny. Należy do nich wymóg, aby wszystkie otwierające znaczniki miały pasujące do nich znaczniki końcowe. Elementy

zagnieżdżone muszą się zaczynać i kończyć w ramach tego samego elementu nadrzędnego. Ponadto elementy XML nie mogą zawierać więcej niż jednego atrybutu, a symbole znaczników muszą być właściwie wyspecyfikowane. Jeśli dane XML spełniają te wszystkie wymagania, są uważane za dobrze uformowane.

Całkowita wielkość przechowywanych danych XML jest ograniczona do 2 GB. Dane te mogą zawierać znaki nie-Unicode albo Unicode. Niekiedy dane XML stosują się do wskazówek teorii zbiorów i mają określone typy danych. W takim scenariuszu dane XML mogą mieć zdefiniowany schemat XML. W przypadku danych XML posiadających schemat są one uznawane za typowane. Często okazuje się, że typowane dane XML zajmują mniej miejsca i dostępne mogą być dodatkowe funkcjonalności dotyczące przechowywanych danych. Jednak ograniczeniem typowanego XML jest to, że takie dane XML muszą przejść walidację. Jeśli wybierzemy nietypowane dane XML, nie ma konieczności ich walidacji i mogą one nie mieć przypisanego schematu.

Typ GEOMETRY

Podczas pracy z danymi może zajść potrzeba przechowywania w bazie danych rozmaitych kształtów. Choć nie jest to zbyt powszechna potrzeba, istnieje odpowiedni typ danych, który pozwala na przechowywanie kształtów lub rysunków opartych na płaszczyźnie euklidesowej.

Ten typ danych może obsługiwać wiele formatów obiektów, w tym punkty, linie, łuki kołowe, krzywe, wielokąty, zbiory punktów, zbiory linii, zbiory wielokątów oraz kolekcje dowolnych spośród tych obiektów.

Typ GEOGRAPHY

Podczas gdy typ GEOMETRY pozwala przechowywać w SQL Server kształty tworzące płaskie mapy, istnieją sytuacje, w których konieczne jest przechowanie informacji opartych na kształcie Ziemi. W takich przypadkach preferowane będzie użycie typu GEOGRAPHY. Typ ten pozwala przechowywać kraje, drogi lub inne elementy map, w których istotne jest uwzględnienie długości i szerokości geograficznej.

Podobnie jak typ GEOMETRY, typ danych GEOGRAPHY również obsługuje wiele opcji. Obejmują one wszystkie rodzaje kształtów występujące w typie GEOMETRY, czyli punkty, linie, krzywe, wielokąty, wielokąty krzywoliniowe oraz kolekcje dowolnych spośród tych kształtów. Jednak typ GEOGRAPHY wspiera również wystąpienie całego globu.

SQL_VARIANT

Możemy się zetknąć z sytuacją, gdy będziemy chcieli w jednej kolumnie tabeli przechować więcej niż jeden typ danych. Wprawdzie w ogólności nie jest to najlepszą praktyką, jednak warto wiedzieć, jak ten typ danych przechowuje informacje i jakiego rodzaju ograniczenia go dotyczą.

Typ danych SQL_VARIANT umożliwia przechowywanie danych różnych typów w tej samej kolumnie. Dane zapisywane w tej kolumnie mają maksymalną wielkość 8016 bajtów. 16 z tych bajtów służy do przechowania informacji o typie danych zawartych w konkretnym rekordzie. Pozostawia to 8000 bajtów dostępnych dla rzeczywistych danych zapisanych w kolumnie.

Dane mogą być wstawiane bezpośrednio do kolumny albo rzutowane jako określony typ. Jeśli nie określimy typu w momencie wstawiania do kolumny typu SQL_VARIANT, SQL Server będzie próbował ustalić właściwy typ. Może to spowodować, że dane zostaną przechowane inaczej, niż oczekiwano. Podczas gdy SQL_VARIANT, jak się wydaje, radzi sobie dobrze z liczbami, możemy napotkać sytuacje, gdy daty zostaną zapisane jako typ VARCHAR(8000), o ile nie określimy jawnie typu danych.

W kolumnie typu SQL_VARIANT można przechowywać dane wszystkich typów z wyjątkiem poniższych:

- VARCHAR(MAX)
- NVARCHAR(MAX)
- TEXT
- IMAGE
- SQL_VARIANT
- HIERARCHYID
- VARBINARY(MAX)
- XML
- NTEXT
- ROWVERSION
- GEOGRAPHY
- GEOMETRY
- DATETIMEOFFSET
- Typy zdefiniowane przez użytkownika

Sposób sortowania danych w kolumnach SQL_VARIANT również różni się od innych typów danych. SQL_VARIANT grupuje dane według podobnych typów, nazywanych

rodzinami typów danych. Każda rodzina typów ma swój własny porządek i wartości o typach należących do wyższych rodzin będą uważane za większe od tych z niższych rodzin. Jeśli porównywane dane należą do tej samej rodziny, SQL_VARIANT niejawnie przekonwertuje typ danych o niższym priorytecie na typ pasujący do wyższego priorytetu i dopiero wtedy wykona porównanie.

ROWVERSION

Zdarzają się sytuacje, w których będziemy chcieli wiedzieć, kiedy w tabeli nastąpiło jakieś działanie. Choć istnieje wiele metod śledzenia zmian w bazie danych, istnieje również specjalny typ danych, którego można użyć do rejestrowania zdarzeń aktualizacji rekordu.

Typ danych ROWVERSION pozwala uzyskać względne pojęcie o tym, kiedy określony rekord lub zbiór rekordów został zaktualizowany. Wartość ta nie zawiera komponentu daty ani czasu, ale jest wartością binarną. Wartość ROWVERSION można porównać z innymi wartościami tego samego typu lub z bieżącą wartością rowversion dla bazy danych. Dla przykładu możemy utworzyć tabelę zawierającą kolumnę o typie ROWVERSION. W momencie wstawiania rekordu kolumna ROWVERSION uzyska na przykład wartość 0x00000000000007D1. Jeśli później zmienię jedną z kolumn tego samego wiersza, kolumna ROWVERSION zostanie uaktualniona, uzyskując wartość 0x00000000000007D2. Będziemy zatem mogli stwierdzić, że wystąpiła jakaś zmiana, choć nie będzie można określić, kiedy to nastąpiło ani czego dotyczyła zmiana.

Przechowywanie wartości rowversion wymaga 8 bajtów pamięci. Możemy określić, czy kolumna typu ROWVERSION ma dopuszczać wartości NULL (*nullable*), czy nie. Jeśli kolumna nie jest *nullable*, zachowuje się podobnie, jak BINARY(8). W przeciwnym razie kolumna będzie się zachowywać podobnie do VARBINARY(8). Do każdej tabeli można dodać tylko jedną kolumnę typu ROWVERSION. Kolumna ta będzie systematycznie aktualizowana za każdym razem, gdy jakiś wiersz lub wiersze są wstawiane lub aktualizowane.

HIERARCHYID

Niekiedy dane są powiązane z innymi danymi w tej samej kolumnie. Często mamy z tym do czynienia przy strukturach hierarchicznych, złożonych z elementów, które są nadrzędne lub podrzędne względem innych. Przykładem mogą być dane lokalizacji, takich jak kraj–prowincja–miasto albo informacje produktowe, takich jak producent, model i numer seryjny (lub data produkcji).

W takich sytuacjach użyteczny może być typ HIERARCHYID, gdyż pomaga kategoryzować wzajemne zależności pomiędzy danymi. Typ danych HIERARCHYID jest ograniczony do 892 bajtów. Jednak jest to typ systemowy o zmiennej długości. Trzeba jednak pamiętać, że choć jest to systemowy typ danych, to za ustalenie właściwej hierarchii, którą należy zapisać, odpowiedzialna jest używająca go aplikacja.

TABLE

Niektóre typy danych nie mogą być stosowane jako kolumny w tabelach. Jednym z nich jest typ TABLE, definiujący zmienną tablicową. Jeśli w trakcie programowania pojawi się potrzeba chwilowego przechowania danych do późniejszego użycia, może to być dobry kandydat do rozważenia. Istnieje jednak kilka pułapek i ograniczeń, które trzeba mieć na uwadze, gdy chcemy używać tablicowego typu danych.

Zazwyczaj preferowane jest ograniczenie użycia zmiennych tablicowych do scenariuszy, w których nie spodziewamy się dużych rozmiarów zwracanych danych. Ujmując historycznie, SQL Server miał problemy z szacowaniem całkowitej liczby wierszy w zmiennych tablicowych. Jednak począwszy od wersji SQL Server 2019 silnik bazy danych powinien podawać lepsze oszacowania przy przetwarzaniu zmiennych tablicowych.

Choć zmienne tablicowe mogą być używane w procedurach składowanych, wsadach lub funkcjach, trzeba pamiętać, że zmienna istnieje tylko tak długo, jak długo istnieje wywołujący ją obiekt. W przypadku funkcji i procedur składowanych zmienna tablicowa przestaje istnieć, gdy funkcja lub procedura zakończy działanie. W przypadku wsadów tabela będzie istnieć przez cały czas działania wsadu. Jako że zmienna istnieje tylko przez czas aktualizowania rekordów, jej użycie może zmniejszyć zasięg i czas istnienia blokad wymaganych w ramach tej aktualizacji.

Innym ograniczeniem tabel jako typów danych jest brak statystyk generowanych dla zmiennych tablicowych. Oznacza to również, że bardzo ograniczone jest stosowanie indeksów dla zmiennych tablicowych. W istocie do wersji SQL Server 2014 w ogóle nie było możliwe tworzenie indeksów dla zmiennych tablicowych. Począwszy od SQL Server 2014 przy tworzeniu zmiennej tablicowej można dołączyć pewne indeksy.

CURSOR

Innym typem danych, którego nie można użyć jako kolumny w tabeli, jest kursor. Zasadniczo stosowalność tego typ jest bardzo ograniczona, ale zdarzają się sytuacje, w których jest najlepszym typem danych dla określonego zadania. Przy rozważaniu tego typu

danych trzeba mieć na uwadze potencjalny wpływ na wydajność powiązany ze stosowaniem kursorów.

Typ danych CURSOR jest zazwyczaj używany jako zmienna. Jednak ten typ może być również użyty jako format wyjściowy z procedury składowanej. W każdym przypadku typ danych CURSOR przyjmuje jako wejście zbiór danych i wykonuje działania na każdym rekordzie, wiersz po wierszu. Jako że kursor może przechowywać zbiór danych, możliwe jest również, że nie będzie zawierać żadnych danych. Oznacza to, że typ danych CURSOR jest *nullable*.

Większość zastosowań tego typu danych ma związek z tworzeniem i używaniem kursora. Aby móc utworzyć kursor, konieczne jest zadeklarowanie zmiennej lokalnej jako typ CURSOR. Podobnie jak w przypadku innych zmiennych lokalnych, możliwe jest albo zadeklarowanie kursora i wypełnienie go wartościami, albo zadeklarowanie i kursora i wypełnienie go przy użyciu wyrażenia operującego na zbiorach. Typu tego mogą używać funkcje służące do przetwarzenia kursorów. Zaliczają się do nich funkcje OPEN, FETCH, CLOSE, DEALLOCATE oraz CURSOR_STATUS. Dodatkowo istnieją systemowe procedury składowane, które mają typ danych CURSOR.

Jednym z kluczowych czynników dla zrozumienia różnych dostępnych typów danych jest znajomość zastosowań tych typów. W niektórych przypadkach użycie właściwego typu może oznaczać oszczędność miejsca. W innych sytuacjach wybór niewłaściwego typu danych może powodować znaczące problemy wydajnościowe. Będziemy również chcieli zachować spójność w stosowaniu i odwoływaniu się do danych w naszym kodzie T-SQL i w obiektach bazodanowych. Jeśli SQL Server musi porównać dwa różne typy, konieczne jest przekonwertowanie przynajmniej jednego z nich, aby były takie same. Proces ten nazywamy niejawną konwersją. Koszt procesora powiązany z niejawnymi konwersjami może być znaczący i należy go unikać, o ile to możliwe. Najlepszym sposobem unikania niejawnych konwersji jest używanie tych samych typów danych dla pól, które będą porównywane. Największym wyzwaniem jest to, że niekiedy trzeba wielu lat, aby się przekonać, że niewłaściwy typ danych ma negatywny wpływ na wydajność aplikacji.

Rozdział 2

Obiekty bazodanowe

Tworzenie kodu T-SQL, który działa szybko i wydajnie wykorzystuje możliwości sprzętu, oprócz właściwych typów danych wymaga znajomości jeszcze wielu innych struktur. Typy danych pomagają w ustaleniu, jak powinny być przechowywane nasze dane, ale kolejnym krokiem jest zaprojektowanie procesu dostępu do tych danych. Jedną z największych zalet, ale i wad posługiwania się językiem T-SQL jest mnogość opcji dostępu do danych. Zakładam tutaj, że Czytelnik wie już, jak pisać kod T-SQL odczytujący, wstawiający, aktualizujący lub usuwający dane.

W tym rozdziale przedstawię różne metody, których możemy używać w celu interakcji z naszymi danymi. Istnieją obiekty, które pozwalają spójnie i szybko zebrać razem wymagane informacje. Będziemy też potrzebować obiektów bazodanowych, które wykonują niewielkie, szybkie działania i których kod będzie można ponownie wykorzystać do różnych celów. Niektóre obiekty bazodanowe pozwalają tymczasowo przechować informacje w celu ponownego użycia w ramach tego samego wsadu lub połączenia. Pewne obiekty mogą wykonywać pewne działania jako skutek aktywności występującej na serwerze lub w innych obiektach bazy danych. Choć T-SQL najlepiej sprawdza się dla działań opartych na zbiorach, może się również okazać, że konieczne będzie przetwarzanie danych w pętli, po jednym rekordzie na raz.

Zależnie od celów, może istnieć jeden lub więcej obiektów bazodanowych spełniających nasze potrzeby. Każdy z tych obiektów ma swoje miejsce i cel, a posługiwanie się nimi zwykle wiąże się z jakimiś zaletami i wadami. W tym rozdziale pokażę różne scenariusze pokazujące zarówno pozytywne, jak i negatywne konsekwencje posługiwania się każdym z tych obiektów. Rozpoczniemy od omówienia widoków, stanowiących jedną z fundamentalnych koncepcji T-SQL.

Widoki

Czym jest widok? Podobnie jak w przypadku słownikowej definicji tego wyrazu, widok (*view*) w T-SQL jest sposobem zebrania wielu różnych elementów i umieszczeniu ich razem w celu zbudowania pojedynczego, spójnego obrazu. W tym rozdziale omówię niektóre opcje dostępne przy używaniu widoków. Podobnie jak w przypadku dowolnego innego narzędzia, używanie widoków ma swoje zalety, ale istnieją też związane z nimi zagrożenia, jeśli zostaną niewłaściwie użyte.

Widoki zdefiniowane przez użytkownika

Termin *widok zdefiniowany przez użytkownika* (*user-defined view* – UDV) to pełna nazwa podstawowej wersji widoku. Jednym z głównym efektów widoku jest prostota. Jest to sposób, dzięki któremu aplikacje i użytkownicy mogą uzyskiwać dostęp do złożonych zbiorów informacji bez konieczności rozumienia wszystkich wielopoziomowych zależności i relacji w bazie danych. Co również istotne, stosowanie widoków zapewnia pewne dodatkowe funkcjonalności pozwalające na ochronę i zabezpieczenie danych. Przedstawię również przykłady widoków, które mogą pomóc w poprawianiu wydajności, ale także sytuacje, w których widoki mogą nie być właściwym wyborem.

W przypadku standardowych widoków definiowanych przez użytkownika SQL Server nie przechowuje fizycznie rzeczywistych danych zwracanych przez widok. Oznacza to, że ilekroć widok jest wywoływany, wyrażenie zawarte w jego definicji zostanie użyte do pobrania rzeczywistych, istniejących danych. Jedną z zalet tej metody jest to, że użytkownicy uzyskujący dostęp do tych widoków potrzebują prostszego kodu, który jest bardziej przejrzysty i łatwiejszy do zrozumienia. Inną cechą widoków jest to, że użytkownicy mogą mieć uprawnienie do widoku, ale nie do powiązanych z nim tabel. W ten sposób możemy pozwolić użytkownikom na dostęp do części, ale nie wszystkich danych zawartych w tabelach tworzących widok.

Zacznijmy od porównania wydajności widoku z wydajnością tego samego zapytania użytego jako kwerenda ad hoc albo jako procedura składowana. Listing 2-1 przedstawia zapytanie, którego użyjemy jako podstawy dla porównań.

Listing 2-1. *Zapytanie dla celów analizy*

```
SELECT meal.MealTypeName, rec.RecipeName, rec.ServingQuantity, ing.
IngredientName
FROM dbo.Recipe rec
     INNER JOIN dbo.MealType meal
```

```
ON rec.MealTypeID = meal.MealTypeID
INNER JOIN dbo.RecipeIngredient recing
ON rec.RecipeID = recing.RecipeID
INNER JOIN dbo.Ingredient ing
ON recing.IngredientID = ing.IngredientID
```

Zapytanie to zostanie użyte do porównania wpływu widoku na wydajność. Jak widać, jego logika jest bardzo prosta. Listing 2-2 pokazuje, jak utworzyć widok na podstawie tego zapytania.

Listing 2-2. *Tworzenie widoku*

```
CREATE VIEW dbo.AvailableMeal
AS
SELECT meal.MealTypeName, rec.RecipeName, rec.ServingQuantity,
    ing.IngredientName
FROM dbo.Recipe rec
     INNER JOIN dbo.MealType meal
     ON rec.MealTypeID = meal.MealTypeID
     INNER JOIN dbo.RecipeIngredient recing
     ON rec.RecipeID = recing.RecipeID
     INNER JOIN dbo.Ingredient ing
     ON recing.IngredientID = ing.IngredientID
```

Po utworzeniu widoku znacznie prostsze staje się wydobywanie tych informacji, które zwracało oryginalne zapytanie. Listing 2-3 pokazuje użycie widoku w celu upraszczania pobierania danych z SQL Server.

Listing 2-3. *Wywoływanie widoku*

```
SELECT MealTypeName, RecipeName, IngredientName
FROM dbo.AvailableMeal
```

Uproszczony dostęp do danych niewątpliwie jest czymś przyjemnym, pojawia się jednak pytanie o wydajność – innymi słowy, jak sprawnie działa widok w porównaniu do kwerendy ad hoc. W ogólności możemy oczekiwać, że widok będzie działać tak samo sprawnie, jak zapytanie istniejące w jego wnętrzu. Rysunek 2-1 pokazuje rzeczywisty plan wykonania dla kwerendy ad hoc.