POZNAJ MOŻLIWOŚCI SQL SERVER!

Apress

SQL Server

Wstęp dla programistów

Wydanie IV

Robin Dewson

Helion

Tytuł oryginału: Beginning SQL Server for Developers, 4th Edition

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-1267-8

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

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

Printed in Poland.

Spis treści

ROZDZIAŁ 5

Definiowanie tabel

Twoja utworzona właśnie baza danych musi w jakiś sposób przechowywać informacje. Bo inaczej jaki byłby z niej pożytek? Pierwszą rzeczą, którą trzeba się zająć, jest zdefiniowanie tabel.

Aby baza mogła funkcjonować, musi zawierać przynajmniej jedną tabelę. Może ich zawierać wiele, a w zależności od tworzonego rozwiązania liczba tabel może być bardzo duża. Dlatego dla Ciebie, jako programisty, bardzo ważna jest jak najlepsza znajomość tabel, ich struktur i zawartości. Celem niniejszego rozdziału jest przekazanie Ci solidnej podstawowej wiedzy niezbędnej do pracy z tabelami. Dzięki niej będziesz mógł tworzyć inne obiekty związane z tabelami, takie jak widoki, wyzwalacze itp.

Projekt tabeli jest bardzo ważny. Każda tabela musi zawierać w swoich kolumnach odpowiednie informacje, niezbędne do utworzenia poprawnych relacji. Jedną z cech dobrego programisty czy administratora bazy danych jest umiejętność sprawdzenia, czy ostateczny projekt bazy przedstawia poprawne rozwiązanie, aby w przyszłości podczas wdrażania go uniknąć uciążliwego wprowadzania poprawek. Jeżeli na przykład zaprojektujesz system, w którym tabele będą zawierały podstawowy błąd i trzeba będzie przenosić kolumny między tabelami, będziesz musiał zweryfikować kod każdej aplikacji, która z tych tabel korzysta. Taka operacja może wymagać włożenia mnóstwa pracy. W rozdziale 3. dowiedziałeś się, jak projektować bazy danych, a teraz będziesz tworzył bazę zawierającą tabele. Wiesz już, jakie tabele będą Ci potrzebne i jakie informacje muszą zawierać.

Aby tworzyć poprawne tabele, musisz poznać następujące zagadnienia opisane w tym rozdziale:

- Jak tworzyć tabele zapisywane na dysku i w pamięci?
- Jakie są typy danych, które można przechowywać?
- W jaki sposób i gdzie zapisywane są tabele na dysku i w pamięci?
- Jak tworzyć tabele za pomocą aplikacji SQL Server Management Studio i edytora zapytań?
- Jak tworzyć bardziej zaawansowane tabele, np.:
  - zawierające unikatowe wartości,
  - obsługujące specjalne stany danych?
- Jak przetwarzać duże ilości danych tekstowych i binarnych?

Czym jest tabela?

Tabela to repozytorium danych, w którym jednostki informacji są rozmieszczone w jednej lub wielu kolumnach. Tabele mogą zawierać wiele wierszy albo nie zawierać żadnych. Arkusz programu Excel można traktować jak tabelę, aczkolwiek jest ona bardzo prosta, ponieważ nie obowiązują w niej żadne reguły dotyczące zawartych danych albo jest ich niewiele. Gdy spojrzysz na rysunek 5.1, zauważysz, że w pierwszych trzech kolumnach znajdują się takie informacje, jak imię, nazwisko i data urodzenia, natomiast dane w czwartej kolumnie mają dowolny format — jest to np. numer pokoju, numer domu i numer mieszkania. Dane nie są jednolite. W rzeczywistości kolumny w Excelu mogą zawierać dowolne dane.

	A	B	C	D	E	F	G
1	Filip	Dąbrowski	1971-12-12	Pokój 123	Hotel Sobieski	Pl. Zawiszy 1	Warszawa
2	Zuzanna	Kamińska	1982-04-30	Mieszk. 2	Blok 71		
3	Julia	Kowalczyk	1969-07-21	133	ul. Parkowa	Gliwice	
4	Kacper	Kowalski	1991-12-01	22			

Rysunek 5.1. *Arkusz programu Excel zawierający dane adresowe*

Tabele w bazie danych serwera SQL Server odróżnia od innych tabel konieczność określenia konkretnego typu danych przechowywanych w każdej kolumnie. Wybranego na początku typu danych dla danej kolumny nie da się zmienić bez naruszenia danych we wszystkich wierszach tabeli. W przypadku Excela jeden wiersz może zawierać tekst, drugi liczbę, kolejny walutę itd. W tabeli w bazie danych jest to niemożliwe. Można w niej przechowywać wszystkie wymienione wartości, ale zapisane jako tekst, dlatego zastosowanie bazy danych w opisanym przypadku jest bezzasadne.

Podczas tworzenia tabeli dla każdej kolumny należy wybrać określony typ danych. Dlatego tworzenie tabeli wymaga dokładnych przemyśleń, aby kolumny zawierały dane najbardziej odpowiedniego typu. Nie ma powodu, aby wybierać ogólny typ danych (na przykład tekst), dopuszczający wszystkie możliwe wartości, ponieważ później trzeba będzie zweryfikować projekt takiej tabeli. Wybranie typu tekstowego dla wszystkich kolumn może być przyczyną problemów z użyciem operacji matematycznych, na przykład przy dodawaniu podatku do ceny lub dodawaniu liczby dni do daty zamówienia w celu określenia daty dostawy towaru.

Zadaniem tabeli jest przechowywanie określonych informacji. Tabela musi mieć opisową nazwę i jedną lub wiele kolumn, z których każda również musi mieć zrozumiałą nazwę i określony typ danych.

Aby zacząć tworzyć tabele, musisz połączyć się z serwerem SQL Server, używając konta z przypisanymi właściwymi rolami serwerową i bazodanową, na przykład *sysadmin* lub *db_ddladmin*, umożliwiającymi tworzenie tabel. Tabela po utworzeniu musi być umieszczona w bazie danych, co osiąga się przez przypisanie tabeli do schematu. Jak pamiętasz, w rozdziale 4. opisane zostało grupowanie obiektów, stanowiące podstawę ich bezpieczeństwa. Tobie, jako programiście, umieszczenie obiektów w schematach pozwoli utworzyć bazę o bardziej logicznej strukturze i skróci czas programowania, ponieważ obiekty będą uporządkowane w określony sposób.

Niektóre typy danych mają na stałe określoną ilość zajmowanego przez nie miejsca, natomiast w przypadku innych typów Ty musisz zadecydować, ile maksymalnie znaków mogą zajmować. Kolumna, która będzie zawierać nazwiska, musi przechowywać dane tekstowe. Nie ma sensu ustalać maksymalnej długości danych w tej kolumnie na 10 znaków, ponieważ wiele nazwisk jest dłuższych. Podobnie nie można konfigurować maksymalnej długości na 1000 znaków. Musisz wybrać rozsądny kompromis na podstawie analizy danych, którą przeprowadziłeś w rozdziale 3. Ustawienia dotyczące wielkości danych można określić na podstawie istniejącej bazy danych, którą będziesz rozbudowywał, albo innej, zawierającej podobne dane. Na przykład, jeżeli projektujesz bazę danych do obsługi sprzedaży, ilość miejsca dla numeru produktu można określić na podstawie magazynowej bazy danych, zawierającej wszystkie szczegółowe informacje o produktach, w tym ich numery.

■ **Uwaga** Nawet jeżeli używasz danych o zmiennej długości, dla których ustala się maksymalną liczbę znaków, serwer SQL Server i tak zapisuje tylko faktycznie wprowadzone dane. Możesz zatem utworzyć kolumnę na nazwiska, mogącą pomieścić maksymalnie 1000 znaków. Miejsce i tak nie będzie tracone, ponieważ większość nazwisk jest znacznie krótsza. Musisz jednak zachować rozsądną i przemyślaną równowagę. Nazwisko o długości 1000 znaków ją zakłóca.

Wiersze danych przechowywanych w tabeli powinny być ze sobą logicznie powiązane. Jeżeli definiowana tabela ma przechowywać informacje o klientach, to nie powinna zawierać żadnych innych danych. Absolutnie nie powinieneś umieszczać w niej informacji niezwiązanych z klientami. Byłoby logicznie nieuzasadnione umieszczanie w takiej tabeli na przykład danych o zamówieniach klientów. Podczas definiowania kolumny nie ulegaj pokusie umieszczania w niej więcej niż jednego rodzaju danych. Na przykład kolumna, która ma zawierać datę złożenia zamówienia, może również zawierać datę jego realizacji, ale pojawi się wtedy problem określenia, co oznacza określona data. Dzięki utworzeniu osobnych kolumn dla obu rodzajów dat sprawa będzie zupełnie jasna.

■ **Uwaga** Są dwa rodzaje tabel — tabele systemowe i tabele użytkownika. Z tabel systemowych nie można korzystać bezpośrednio, niektóre zawarte w nich informacje są dostępne za pomocą widoków systemowych.

Innym typem tabeli dostępnym dla programistów w serwerze SQL Server jest tabela plikowa (*FileTable*). Jest to tabela specjalnego typu, umożliwiająca przechowywanie plików o nieokreślonej strukturze, na przykład dokumentów programu Word czy arkuszy Excel. Dzięki technologii *FILESTREAM* można w serwerze SQL Server przeszukiwać takie pliki. W tej książce opisane są tylko zwykłe tabele użytkownika.

Jest jeszcze trzeci typ tabel — przechowywanych w pamięci. Są one opisane w osobnym rozdziale 7., ponieważ tabele tego typu pod wieloma względami różnią się od typowych tabel zapisywanych na dysku. Różnice obejmują stosowane typy danych, wymagania, jakie musi spełnić baza i tabela, oraz aspekty wydajnościowe. Na razie pamiętaj jedynie, że tabele i zawarte w nich dane mogą być umieszczane na dysku lub w pamięci.

■ **Uwaga** Choć ten rozdział jest poświęcony tabelom zapisywanym na dysku, wiele informacji tu zawartych dotyczy również tabel zapisywanych w pamięci.

Typy danych w serwerze SQL Server

W poprzednich rozdziałach wiele dowiedziałeś się o serwerze SQL Server, jeszcze przed utworzeniem swojej pierwszej tabeli. Jednak uzyskanie tych informacji przed utworzeniem tabeli i zabezpieczeniem bazy jest bardzo ważne, ponieważ pozwala uniknąć nieprzyjemnych konsekwencji, gdy coś zacznie iść źle. Wiesz również, dlaczego należy być ostrożnym, nadając użytkownikom uprawnienia do tworzenia tabel. W tej części rozdziału poznasz typy danych dostępne w serwerze SQL Server. Niektóre z nich mogą być przypisane do kolumn w tabeli, a inne użyte w kodzie T-SQL.

Tabele można definiować za pomocą aplikacji SQL Server Management Studio, edytora zapytań *Query Editor* lub narzędzia do projektowania baz danych w serwerze SQL Server. Możesz również użyć innych metod, na przykład narzędzia `sqlcmd`, narzędzi programistycznych czy języków programowania, jednak w tej książce skupimy się na trzech wymienionych wcześniej metodach. Pierwszą tabelę utworzysz za pomocą aplikacji SQL Server Management Studio. Będzie to tabela *Klienci*, zawierająca dane o wszystkich klientach. Zanim jednak to zrobisz, musisz poznać różne typy danych, które można przechowywać w tabeli.

Typy danych w tabelach

Serwer SQL Server oferuje wiele typów danych, które można określić dla każdej kolumny. Ta część rozdziału zawiera opis różnych typów i pomoc w wyborze właściwego typu dla każdej kolumny tabeli. Opisane typy nazywane są **typami podstawowymi**. Cała książka skupia się tylko na typach podstawowych. Możesz jednak tworzyć własne niestandardowe typy, co ma swoje zalety. Jeżeli chcesz mieć spójne dane w określonej kolumnie w różnych tabelach, zazwyczaj na tym samym serwerze, ale nie jest to konieczność, możesz utworzyć własny typ danych i nadać mu nazwę. Taki typ może być następnie wykorzystany podczas definiowania tabeli. Przykładem mogą być numery NIP, które muszą mieć taki sam format. Możesz utworzyć własny typ danych oparty na typie podstawowym i nadać mu nazwę. Następnie możesz utworzyć kolumnę zawierającą numery NIP, wykorzystując własny typ danych zamiast podstawowego. Zademonstruję ten sposób po opisaniu typów podstawowych.

■ **Uwaga** Do tworzenia bardziej złożonych typów danych i uzyskania dodatkowych funkcjonalności możesz wykorzystać technologię .NET.

Niektóre typy danych mogą wydawać Ci się podobne, jednak pamiętaj, że każdy ma swoje określone przeznaczenie. Na przykład, jeżeli nie musisz przechowywać danych zapisanych w formacie Unicode, nie używaj typów o nazwach zaczynających się na literę *n*. Tekst zapisany w formacie Unicode zajmuje więcej miejsca, ponieważ format ten obejmuje większy zbiór znaków, który może być obsługiwany przez serwer SQL Server. Ponadto, jeżeli największa liczba zapisana w kolumnie będzie równa 100, nie używaj typu, który pozwala na zapisanie największej możliwej liczby, ponieważ byłaby to strata miejsca na dysku.

Przyjrzyjmy się dostępnym typom podstawowym, których możesz użyć w tabeli. Później poznasz typy stosowane w programach. We wszystkich przykładach opisanych w tej książce są stosowane różne opisane niżej typy danych.

Typ char/nchar

Dane typu `char` mają stałą długość, określaną podczas definiowania kolumny, i mogą zawierać maksymalnie 256 znaków. Jeżeli zdefiniujesz kolumnę zawierającą dane o długości 20 znaków, taka liczba znaków będzie zapisywana. Jeżeli dane będą miały mniej znaków, dodatkowe miejsca zostaną wypełnione spacjami. Jeżeli typ kolumny będzie zdefiniowany jako `char(10)`, ciąg `"aaa"` zostanie zapisany jako `"aaa       "`. Stosuj ten typ danych wtedy, gdy dane w kolumnie muszą mieć stałą długość, jak na przykład identyfikatory klientów lub numery kont. Jeżeli zdefiniujesz kolumnę bez podania długości danych, zostanie przyjęta wartość 1. Nie jest to jednak dobrą praktyką; dla przejrzystości wszystkie kolumny powinny mieć ustaloną długość.

Typ varchar/nvarchar

Typ `varchar` służy do przechowywania danych alfanumerycznych, podobnie jak typ `char`. Różnica polega na tym, że każdy wiersz może zawierać różną liczbę znaków, do maksymalnej określonej dla danej kolumny. Jeżeli kolumna ma zdefiniowany typ `varchar(50)`, dane w tej kolumnie mogą mieć maksymalną długość 50 znaków. Jeżeli jednak zostanie wpisany ciąg składający się tylko z trzech znaków, zostaną zajęte tylko trzy miejsca na dysku. Ten typ doskonale nadaje się w sytuacjach, gdy dane nie mają określonej długości, na przykład nazwiska lub opisy. Maksymalna długość danych typu `varchar` jest równa 8000 znaków. Jeżeli zdefiniujesz kolumnę, nie podając długości, tj. stosując typ `varchar()`, zostanie przyjęta domyślna długość 1.

Stosując w definicji typu stałą `max`, możesz ominąć ograniczenie długości do 8000 znaków. Użyj tej opcji, jeżeli wiesz, że dane przynajmniej w jednym wierszu przekroczą długość 8000 znaków. Ponadto stałej `max` możesz używać zamiast typu `text` do przechowywania dużych bloków tekstu.

Typ text/ntext

Typ `text` służy do przechowywania danych, których długość przekracza 8000 znaków. Nie stosuj jednak tego typu, ponieważ jest on nieaktualny i zostanie wycofany. Zamiast niego stosuj typ `varchar(max)`.

Typ image

Typ `image` jest bardzo podobny do typu `text`, z tym wyjątkiem, że służy do przechowywania danych binarnych. Jednakże typ ten, podobnie jak `text` i `ntext`, będzie wycofany i zamiast niego należy stosować typ `varbinary(max)`.

Typ int

Typ `int`, lub `integer`, służy do przechowywania wartości liczbowych bez miejsc po przecinku (liczby całkowite). Zakres liczb jest ograniczony, mogą to być wartości od –2 147 483 648 do 2 147 483 647.

Typ bigint

Typ `bigint`, lub `big integer`, jest podobny do typu `int`, jednak umożliwia przechowywanie znacznie większych liczb, z zakresu od –9 223 372 036 854 775 808 do 9 223 372 036 854 775 807.

Typ smallint

Typ `smallint`, lub `small integer`, służy do przechowywania małych liczb całkowitych, z zakresu od –32 768 do 32 767. Zachowaj ostrożność podczas definiowania kolumn tego typu i upewnij się, że na pewno przechowywane liczby nie przekroczą dopuszczalnych wartości. Podczas tworzenia kolumn zawierających ten typ danych zawsze istnieje niebezpieczeństwo, że kiedyś będziesz musiał wrócić do definicji kolumny i zmienić jej typ. Dlatego na wszelki wypadek stosuj typ `int`.

Typ tinyint

Typ `tinyint`, lub `tiny integer`, służy do przechowywania jeszcze mniejszych liczb niż typ `smallint`, tj. z zakresu od 0 do 255. Typ ten może być na przykład stosowany do przechowywania numerów województw.

Typ decimal/numeric

Oba powyższe typy służą do przechowywania liczb z tego samego zakresu wartości i o tej samej precyzji. Zakres wartości obejmuje liczby od $-10^{38}+1$ do $10^{38}-1$. Precyzja zawiera się w przedziale od 0,00000000000000000000000000000000000001 do 10 000 000 000 000 000 000 000 000 000 000. Bądź jednak ostrożny, ponieważ nie możesz przechowywać liczb zawierających po 38 cyfr zarówno przed przecinkiem, jak i po nim. Liczby mogą składać się w sumie z maksymalnie 38 cyfr. Zatem im większa jest wymagana dokładność, czyli liczba cyfr po przecinku, tym mniejsza może być liczba cyfr przed przecinkiem.

Typ float

Moim zdaniem typy `float` i `real` są dla początkującego programisty najbardziej niebezpiecznymi typami danych. Powinny być stosowane jedynie w sytuacjach wyjątkowych i należy ich unikać, jeżeli wymagane jest zachowanie pełnej dokładności liczb. Typ `float` jest stosowany do przechowywania wartości o zmiennej liczbie cyfr po przecinku, z zakresu od –1,79E+308 do 1,79E+308. Muszę Cię jednak ostrzec: to nie są wartości w 100% precyzyjne, ponieważ liczby, ze względu na sposób zapisu liczby w formacie binarnym, mogą być różnie zaokrąglane. Mogą pojawić się problemy z liczbami zawierającymi cyfrę 3, 6 lub 7 po przecinku. Liczby te są zaokrąglane, ponieważ czasami zawierają więcej cyfr po przecinku, niż można zapisać. Typowym przykładem jest liczba *pi*.

Typ real

Typ `real` jest bardzo podobny do typu `float`, z tym wyjątkiem, że służy do przechowywania liczb z zakresu jedynie od –3,40E+38 do 3,40E+38. Są to również przybliżone wartości. Ze względu na zakres wartości typy `real` i `float` wydają się idealne. Jeżeli jednak musisz przechowywać dokładne wartości, sprawdź inne typy.

Typ money

Typ `money` służy do przechowywania liczb z maksymalnie czterema cyframi po przecinku. Jeżeli potrzebna jest większa dokładność, musisz użyć innego typu, na przykład `decimal`. Typ `money` nie zawiera symbolu waluty, dlatego nie można go stosować do przechowywania wartości wyrażonych w różnych walutach. Typ ten obejmuje wartości z zakresu od –922 337 203 685 477,5808 do 922 337 203 685 477,5807. Jeżeli musisz zapisać symbol waluty ($ dla dolara, € dla euro itp.), musisz go przechowywać w osobnym miejscu.

Typ smallmoney

Typ podobny do `money`, ale przeznaczony do przechowywania wartości z zakresu od –214 748,3648 do 214 748,3647.

Typ date

Typ `date` służy do przechowywania dat od 1 stycznia 1 r. do 31 grudnia 9999 r. Stosowany jest format zapisu *`RRRR-MM-DD`*. W wersjach oprogramowania SQL Server 2008 i starszych nie można było stosować osobnych typów do przechowywania daty i czasu, chyba że utworzyło się je za pomocą technologii .NET. W efekcie trzeba było zapisywać niepotrzebne dane. Na przykład często zdarza się, że kolumna typu `datetime` zawiera tylko daty. Wprowadzenie typu `date` jest dużym usprawnieniem oprogramowania, ponieważ dzięki niemu możliwość popełnienia pomyłki jest mniejsza i uzyskuje się właściwe wyobrażenie o danych przechowywanych w kolumnie.

Typ datetime

Typ `datetime` służy do przechowywania daty i czasu z zakresu od 1 stycznia 1753 r. do 31 grudnia 9999 r. Jednak wartości tego typu mogą zawierać nie tylko datę, ale również oznaczenia czasu. Jeżeli do kolumny typu `datetime` wpiszesz tylko datę, zostanie przyjęta domyślna godzina 12:00:00.

Typ datetime2

Typ `datetime2`, podobnie jak `datetime`, służy do przechowywania daty i czasu. Główna różnica polega na większej precyzji zapisu ułamków sekund. Ponadto, typ ten umożliwia przechowywanie dat z zakresu od 1 stycznia 1 r. do 31 grudnia 9999 r. Stosowany jest format zapisu *`RRRR-MM-DD gg:mm:ss[.nnnnnnn]`*.

Typ smalldatetime

Typ bardzo podobny do `datetime`, ale obejmujący zakres dat od 1 stycznia 1990 r. do 6 czerwca 2079 r. Nietypowy zakres dat wynika z binarnego sposobu zapisywania danych tego typu.

Typ datetimeoffset

Ten typ możesz stosować, gdy musisz przechowywać dane z określonej strefy czasowej. Data i czas są przechowywane w formacie UTC (Coordinated Universal Time, uniwersalny czas koordynowany). Typ ten umożliwia określenie różnicy czasu, która ma być dodana lub odjęta w zależności od żądanej strefy czasowej. Na przykład data i czas 24 marca 2014 18:00 EST (czas wschodnioamerykański) zostanie zapisana jako 2014-03-24 13:00:00 +05:00. Symbol +05:00 oznacza, że czas EST jest przesunięty o 5 godzin w przód w stosunku do czasu UTC. Stosowany format zapisu to *`RRRR-MM-DD gg:mm:ss[.nnnnnnn] [+|-]gg:mm`*.

Typ time

Użyj tego typu, jeżeli chcesz przechowywać tylko czas w formacie 24-godzinnym. Stosowany jest format zapisu *gg:mm:ss[.nnnnnnn]*. Podobnie jak w przypadku typu `date`, typ ten został wprowadzony, aby można było definiować kolumny zawierające tylko oznaczenia czasu. Jak wynika z formatu, kolumna może zawierać dane z dokładnością do 100 nanosekund. Jest to większa dokładność niż w przypadku typu `datetime`.

Typ hierarchyid

W wersjach oprogramowania starszych niż SQL Server 2008 tworzenie hierarchicznych struktur danych było skomplikowane i wymagało stosowania relacji samołączących. Przykładem hierarchicznych danych może być struktura pracownicza firmy obejmująca pracowników najniższego szczebla, poprzez dyrektorów, po prezesa. Teraz jednak można stosować typ `hierarchyid`, umożliwiający określenie, w którym miejscu w hierarchii wierszy znajduje się bieżący wiersz. Dane tego typu mają zmienną długość i zajmują do 892 bajtów.

Typ geometry

Typ `geometry` jest to planarny typ danych utworzony w technologii CLR (Common Language Runtime, środowisko uruchomieniowe wspólnego języka), umożliwiający przechowywanie „płaskich" informacji geograficznych. Dane tego typu mogą opisywać jeden z jedenastu obiektów geograficznych, na przykład punkt, krzywą, wielokąt itp. W każdej kolumnie można przechowywać dane tylko jednego obiektu. Przechowywane dane zawierają informację o typie obiektu.

Typ geography

Typ utworzony w technologii CLR, służący do przechowywania „zakrzywionych" informacji geograficznych. Dane opisują te same obiekty, co typ `geometry`, jednak wartości wyrażone są w stopniach długości i szerokości geograficznej.

■ **Uwaga** Aby dokładnie opisać charakter i sposób wykorzystania typów `geometry` i `geography`, potrzebny byłby osobny rozdział, a nawet cała książka. Typy te nie są używane w tej książce; można je znaleźć w publikacji *Beginning Spatial with SQL Server 2008* Alastaira Aitchisona (wyd. Apress, 2009).

Typ rowversion

Niestandardowy typ danych stosowany w kolumnach, w których nie będą przechowywane zwykłe wartości. Typ `rowversion` służy do przechowywania liczb binarnych generowanych przez serwer SQL Server, stanowiących unikatowe identyfikatory wierszy w bazie danych. Po każdorazowej modyfikacji danych wartość w kolumnie tego typu jest zmieniana tak, aby uwzględniała czas dokonania modyfikacji. Dlatego typ ten służy do bardziej zaawansowanych zastosowań, gdy wymagane jest rejestrowanie historii wprowadzanych zmian.

■ **Uwaga** Typ `rowversion` nosi również nazwę `timestamp`, spotykaną w bazach danych utworzonych w starszych wersjach oprogramowania SQL Server.

Typ uniqueidentifier

Typ `uniqueidentifier` służy do przechowywania identyfikatorów GUID (Globally Unique Identifier, identyfikator globalnie unikatowy). Jest on podobny do typu `timestamp` pod tym względem, że wartość jest tworzona przez serwer SQL Server podczas wstawiania lub modyfikacji danych. Wartość ta jest generowana na podstawie adresu karty sieciowej, identyfikatora procesora oraz bieżącej daty i czasu. Jeżeli komputer nie jest wyposażony w kartę sieciową, wartość jest generowana tylko na podstawie informacji o komputerze. Wygenerowane wartości są unikatowe w skali całego świata.

Typ binary

Ten typ danych służy do przechowywania danych w formacie binarnym. Jest on głównie używany do zapisywania takich danych, jak znaczniki lub ich kombinacje. Na przykład może to być znacznik opisujący klienta. Załóżmy, że chciałbyś wiedzieć, czy dany klient jest aktywny (znacznik = 1), czy coś zamówił w ciągu ostatniego miesiąca (znacznik = 2), czy jego ostatnie zamówienie przekracza 1000 zł (znacznik = 4) oraz czy spełnia on warunki programu lojalnościowego (znacznik = 8). W typowym rozwiązaniu dane te zajmowałyby cztery kolumny tabeli. Jednak dzięki typowi binarnemu można stosować tylko jedną wartość. Na przykład wartość 13 jest wynikiem dodania liczb 1+4+8, oznaczających, że dany klient jest aktywny, jego ostatnie zamówienie przekroczyło wartość 1000 zł i że spełnia on kryteria programu lojalnościowego. Jeżeli zdefiniujesz kolumnę typu `binary` z ustaloną długością, wszystkie dane będą miały zadaną wielkość.

Typ varbinary

Typ `varbinary` jest bardzo podobny do typu `binary`, z tą różnicą, że wielkość danych w każdym wierszu może być różna w zależności od wartości. Typ `varbinary(max)` umożliwia zapisywanie danych przekraczających długość 8000 znaków i może być stosowany np. do przechowywania obrazów.

Typ bit

Typ `bit` służy do przechowywania wartości 0 lub 1. Zazwyczaj wartość taka jest stosowana do oznaczania wartości logicznych *prawda* (1) lub *fałsz* (0).

Typ xml

Do przechowywania danych w formacie XML można zamiast `varchar(max)` użyć specjalnego typu. W zapytaniach dostępne są odpowiednie polecenia służące do pobierania i przetwarzania takich danych. W wersjach oprogramowania starszych niż SQL Server 2005 dane XML były traktowane niemal jak dane nieokreślonego typu i dostępne funkcjonalności serwera do przetwarzania takich danych były bardzo ograniczone.

Typy danych w programach

Są jeszcze trzy dodatkowe typy danych, które można stosować w programach. Przyjrzyjmy im się teraz.

Typ cursor

Dane przechowywane w pamięci noszą nazwę **kursora**. Kursor przypomina tabelę, ponieważ składa się z kolumn i wierszy, ale na tym podobieństwa się kończą. Na przykład w kursorach nie ma indeksów. Kursory są wykorzystywane podczas przetwarzania danych z tabel wiersz po wierszu.

Typ table

Typ `table` przypomina zarówno kursor, jak i tabelę. Składa się z wierszy i kolumn, jednak dane nie mogą być indeksowane. W takim przypadku należy je traktować jak zwykłą tabelę „przygotowywaną na bieżąco". Ponieważ jest to bardziej złożony temat, więcej o kursorach i danych typu `table` dowiesz się w dalszej części książki, w rozdziale 17.

Typ sql_variant

Możliwe jest stosowanie typu reprezentującego różne typy danych. Szczerze mówiąc, nie polecam jego stosowania, ponieważ nie daje on pewności, jakiego typu danych można się spodziewać. Zanim określisz typ kolumny, musisz być pewny, jakiego typu dane będą w niej umieszczane. Typ `sql_variant` można stosować w programach i w definicjach kolumn, jednak powyższa charakterystyka dotyczy obu przypadków. Ten typ danych nie będzie stosowany w tej książce.

Kolumny są czymś więcej niż zwykłymi repozytoriami danych

Typ przypisany do kolumny określa rodzaj przechowywanych w niej danych. Jednak definicja kolumny daje większe możliwości. Do kolumny można wpisać wartość początkową albo nie umieszczać w niej żadnej wartości.

Wartości domyślne

Gdy do tabeli jest dodawany nowy wiersz, serwer SQL Server może w niektórych kolumnach umieszczać domyślne wartości, na przykład datę i czas wstawienia wiersza, dzięki czemu nie musi tego robić programista. Wartości domyślne mogą być dowolnymi poprawnymi wartościami danego typu. Wartość domyślna może być zastąpiona inną, ponieważ nie jest wpisywana raz na zawsze.

Generowanie wartości IDENTITY

Podczas dodawania nowego wiersza do tabeli przydatne może być nadawanie mu unikatowego, lecz łatwego do interpretacji numeru identyfikacyjnego, który może być użyty jako odnośnik do wiersza w innej tabeli. W bazie *MojeFinanse* będzie utworzona tabela zawierająca listę transakcji. Tabela ta będzie musiała być powiązana z tabelą zawierającą dane klientów. Zamiast próbować łączyć tabele za pomocą wartości, które nie gwarantują unikatowości wiersza (jak na przykład imię i nazwisko), można użyć unikatowego liczbowego identyfikatora, który daje taką możliwość, o ile będzie powiązany z unikatowym indeksem. Jeżeli w tabeli *Klienci* znajduje się wiersz z identyfikatorem równym 100, który powiążesz z wierszami w tabeli *Transakcje*, będziesz mógł pobrać z nich dane o wszystkich transakcjach zrealizowanych przez klienta o identyfikatorze 100. Oznacza to jednak, że gdy będziesz chciał dodać do tabeli nowy wiersz z danymi klienta, będziesz musiał sprawdzić za pomocą kodu T-SQL lub tabeli zawierającej „następne" numery klientów, jaki ma być kolejny numer tego klienta. Nie przejmuj się tym jednak, ponieważ w definicji tabeli można użyć bezcennej opcji `IDENTITY`.

■ **Uwaga** Informacja dla czytelników, którzy korzystali z bazy Microsoft Access: słowo kluczowe `IDENTITY` pełni podobną funkcję jak `AutoNumber`.

Wybierając w definicji kolumny opcję `IDENTITY`, informujesz serwer SQL Server o tym, że:

- Kolumna ma zawierać wartość generowaną przez serwer.
- Będzie określona wartość początkowa (tzw. *ziarno*).
- Kolejne identyfikatory mają być zwiększane o zadaną wartość.
- Przydzielaniem identyfikatorów ma zarządzać serwer.

Gdyby serwer SQL Server nie był w stanie wykonać powyższych operacji, Ty musiałbyś to zrobić. Dlatego dzięki opcji `IDENTITY` w definicji kolumny możesz używać generowanych wartości i tworzyć stabilne, niezawodne i unikatowe połączenia między dwiema tabelami i nie zdawać się na nieprecyzyjne kryteria wyboru wierszy. Musisz jednak pamiętać, że liczby nie są generowane w kolejności rosnącej i mogą pojawić się w nich przerwy, jak się przekonasz w kolejnym przykładzie.

Unikatowe identyfikatory danych

Opcja `IDENTITY` idealnie sprawdza się w sytuacjach, gdy musisz utworzyć w tabeli kolumnę zawierającą unikatowe wartości. Jest jeszcze alternatywna metoda, polegająca na wykorzystaniu unikatowych wartości zwracanych przez systemową funkcję `NEWID()`. Ta metoda ma kilka mankamentów, między innymi generowane dane zawsze zajmują 16 bajtów (typ `bigint` zajmuje tylko 8 bajtów). Ponadto, jeżeli chcesz otrzymać tę wartość, musisz ją wygenerować przed dodaniem wiersza do tabeli (wartości generowane w przypadku opcji `IDENTITY` można odczytać za pomocą zmiennej systemowej), a korzystanie z tej wartości przy przygotowywaniu raportów wykorzystywanych poza bazą danych jest niewygodne. Poza tym, jeżeli kolumna zawierająca takie wartości jest wykorzystywana jako indeks, pojawia się problem z wydajnością bazy.

W normalnym trybie użytkownik nie umieszcza tej wartości w kolumnie; robi to automatycznie serwer SQL Server. Jednak jeżeli zmienisz opcję bazy danych za pomocą polecenia `SET IDENTITY_INSERT ON`, będziesz mógł wpisywać własne wartości. Tej operacji nie wykonuje się często, jednak zapoznasz się z kilkoma scenariuszami, w których ta funkcjonalność będzie przydatna. Używałem tej opcji kiedyś w przeszłości, gdy archiwizowałem dane z bazy produkcyjnej, a potem odtwarzałem je wraz z początkowymi wartościami identyfikatorów.

Sekwencje zamiast opcji IDENTITY

Kolumny z opcją `IDENTITY` idealnie sprawdzają się w sytuacji, gdy do tabeli rzadko wstawiane są niewielkie wiersze, ponieważ za każdym razem nieznacznie spada wydajność bazy. Jakiś czas temu programiści zgłosili do Microsoft potrzebę rozwiązania tego problemu i zwiększenia wydajności bazy. Wtedy Microsoft opracował tzw. **sekwencje**. Kolumna z identyfikatorami jest definiowana w konkretnej tabeli, natomiast sekwencja służy do generowania według określonego schematu serii kolejnych wartości, które mogą być użyte w wielu tabelach.

Pierwsza korzyść ze stosowania sekwencji polega na tym, że serwer SQL Server nie tylko generuje kolejne wartości pojedynczo (jak w przypadku opcji `IDENTITY`), ale również może umieścić ich zakres w pamięci podręcznej, dzięki czemu dostęp do nich i odczyt odbywają się szybciej. Odczytywanie wartości z zapisanego w pamięci zbioru danych przebiega sprawniej niż ich generowanie przy każdorazowym wstawianiu wiersza w przypadku opcji `IDENTITY`.

Ponadto sekwencję można zdefiniować tak, że po osiągnięciu maksymalnej wartości zacznie odliczanie od początku. W ten sposób generowane wartości można wykorzystywać wielokrotnie.

Można również generować kolejną wartość i zapisywać ją w zmiennej przed wstawieniem wiersza. W ten sposób można poznać wartość „identyfikacyjną", wykorzystywaną podczas dodawania wartości w tabelach podrzędnych. W przypadku opcji `IDENTITY` wartość przypisaną do wiersza można było odczytać dopiero po jego wstawieniu. Dzięki możliwości zapisania takiej wartości w zmiennej można ją ponownie wykorzystać i uniknąć powstawania przerw spowodowanych np. odwołaniem nieudanej transakcji wstawienia wiersza. Przerwy jednak i tak się pojawią, jeżeli serwer zostanie ponownie uruchomiony lub zostanie zatrzymana usługa SQL Server Windows. Dlatego użycie lokalnej zmiennej nie jest idealnym rozwiązaniem problemu przerw w generowanych wartościach.

Za pomocą jednej sekwencji można generować wartości wykorzystywane w wielu tabelach i potencjalnie w różnych kolumnach, czego nie można osiągnąć za pomocą opcji `IDENTITY`, która dotyczy tylko jednej kolumny tabeli. Dzięki możliwości wykorzystania sekwencji w wielu tabelach można rejestrować kolejność przetwarzania danych. Załóżmy, że mamy jedną tabelę z danymi o przyjętych wnioskach i dwie tabele z przetworzonymi danymi — jedną z zaakceptowanymi i drugą z odrzuconymi wnioskami. Sprawdzając w obu tabelach wartości wygenerowane przez sekwencję, można określić kolejność przetwarzania wniosków.

■ **Ostrzeżenie** Jeżeli sekwencji użyjesz w transakcji, a transakcja zostanie odwołana, wtedy sekwencja nie zostanie cofnięta i wygenerowana wartość będzie wyglądała tak, jakby została wykorzystana.

Za pomocą sekwencji można programowo wpisywać wartości do kolumny. Ta funkcjonalność rozwiązuje problemy pojawiające się w przypadku zastosowania opcji `IDENTITY`, gdy wpisana wartość zmienia się w niekontrolowany sposób. W przypadku użycia sekwencji po wpisaniu do kolumny wielu wartości, a nawet zmianie wartości, błędnie wpisaną wartość można zmienić, jak się o tym przekonasz w rozdziale 17., w podrozdziale „Modyfikacja sekwencji". Metoda ta ma jednak tę niedogodność, że programista musi pamiętać, że dana kolumna jest logicznie połączona z sekwencją, którą należy wykorzystywać i nie wpisywać wartości ręcznie. Sekwencję można wskazać jako domyślną wartość odpowiedniej kolumny, co będzie pokazane za chwilę w ćwiczeniu. W ten sposób za pomocą sekwencji można generować wartości, podobnie jak w przypadku opcji `IDENTITY`. W ćwiczeniach w niniejszej sekcji sekwencje nie będą wskazywane jako domyślne wartości kolumn, dzięki czemu będziesz mógł dokładnie prześledzić, co będzie się działo.

Zastosowanie wartości NULL

Używając kodu T-SQL bądź włączając lub wyłączając opcję *Allow Nulls* (wartości `NULL` dozwolone) w narzędziu *Table Designer* (projektant tabel), można zdefiniować kolumnę z opcjami `NULL` lub `NOT NULL`. Te dwa przeciwstawne ograniczenia określają, czy do danej kolumny muszą być wpisywane dane, czy nie. Wartość `NULL` oznacza, że kolumna nie zawiera absolutnie żadnych danych. Jest to specjalny stan danych, oznaczający, że wartość w kolumnie jest nieokreślona.

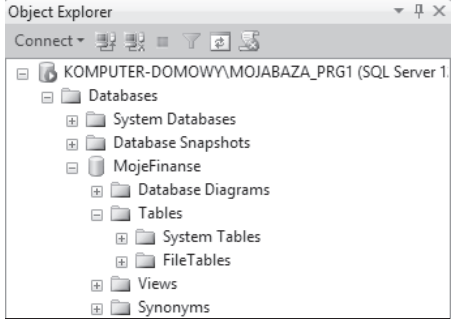
Jeżeli kolumna zawiera wartość `NULL`, oznacza to, że nie zostały do niej wpisane żadne dane lub że dane, które znajdowały się w niej wcześniej, zostały usunięte. Oznacza to również, że w celu sprawdzenia, czy w danej kolumnie znajduje się ta wartość, musisz w kodzie T-SQL użyć odpowiedniej instrukcji. Przeanalizujmy przykład tabeli, w której kolumna została skonfigurowana do przechowywania tekstu, ale w jednym z wierszy znajduje się wartość `NULL`. Jeżeli użyjesz zapytania wykonującego operacje na ciągach znaków, wartość `NULL` może spowodować wystąpienie błędu. Zastosowanie wartości `NULL` jest opisane w rozdziale 10.

Tworzenie tabel za pomocą aplikacji SQL Server Management Studio

Nadszedł czas, abyś utworzył swoją pierwszą tabelę w przykładowej bazie danych. Każda firma ma określoną grupę klientów i musi przechowywać ich dane. Tworzona tabela będzie zawierała dane klientów, m.in. ich nazwiska, oraz identyfikatory adresów zapisanych w zewnętrznym systemie. Jedynym produktem oferowanym przez Twoją firmę jest konto bankowe, na którym mogą znajdować się niezaksięgowane środki. Oznacza to, że Twoja tabela będzie zawierała również numery kont bankowych, ich bieżące stany i salda.

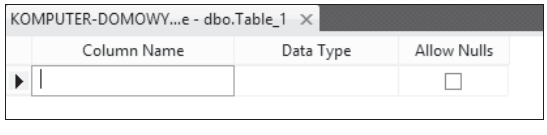
Wypróbuj: tworzenie tabeli

1. Upewnij się, że aplikacja SQL Server Management Studio jest otwarta.
2. Rozwiń panel *Object Explorer* tak, aby była widoczna baza danych *MojeFinanse* utworzona w rozdziale 3.
3. Rozwiń sekcję *MojeFinanse*, aby pojawiła się sekcja *Tables* (tabele), jak na rysunku 5.2.



Rysunek 5.2. *Baza danych MojeFinanse bez tabel w sekcji Tables*

4. Kliknij prawym przyciskiem sekcję *Tables* i wybierz polecenie *New/Table* (nowa/tabela). Otworzy się okno *Table Designer* (projektant tabel), które na początku wygląda jak na rysunku 5.3.



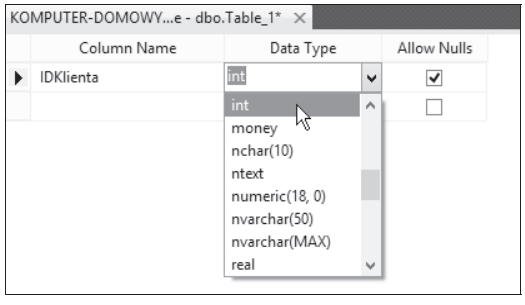
Rysunek 5.3. *Tworzenie pierwszej tabeli, na razie bez kolumn*

5. W tym oknie musisz wprowadzić szczegółowe informacje opisujące każdą kolumnę tabeli. W pierwszej kolumnie okna, *Column Name* (nazwa kolumny), wpisz nazwę nowej kolumny swojej tabeli.

Wskazówka Unikaj stosowania spacji w nazwach kolumn. Lepiej stosuj nazwy bez spacji, jak to zrobiłem w kolumnie *IDKlienta*, lub zamiast nich używaj znaku podkreślenia (_). Stosowanie spacji w nazwach jest całkowicie poprawne, jednak nazwy takie trzeba w kodzie SQL umieszczać w nawiasach kwadratowych ([]), co jest bardzo uciążliwe.

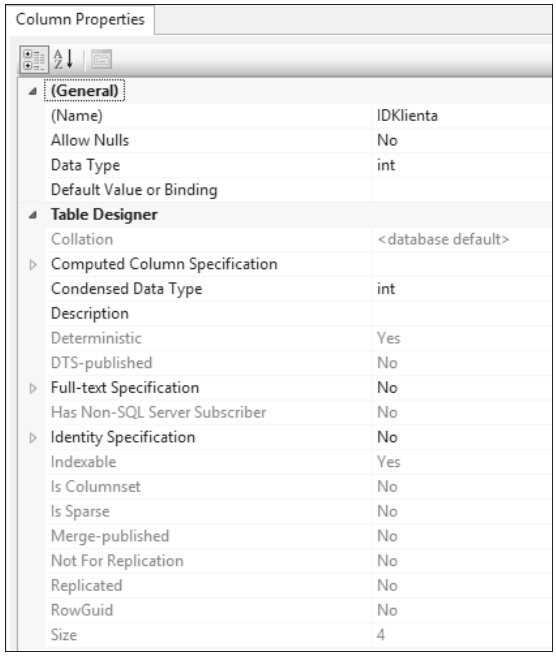
---

6. Zwróć uwagę, że w tej chwili panel *Column Properties* (właściwości kolumny) jest pusty. Informacje będą się w nim pojawiać dopiero wtedy, gdy po wprowadzeniu nazwy kolumny określisz typ danych. Panel *Column Properties* jest równie ważny, jak górna część okna, w którym określa się nazwy kolumn i ich typy.
7. Jedną z pierwszych funkcjonalności aplikacji SQL Server Management Studio, ułatwiającej tworzenie tabeli, jest rozwijana lista typów danych. Dzięki niej nie musisz pamiętać wszystkich typów danych dostępnych w serwerze SQL Server. W liście zawierającej wszystkie potrzebne typy wystarczy wybrać ten najbardziej odpowiedni. W tym przypadku wybierz typ `int`, jak na rysunku 5.4.

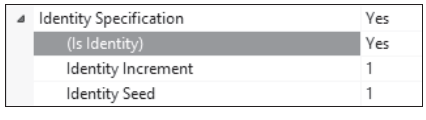


Rysunek 5.4. *Wybór typu danych*

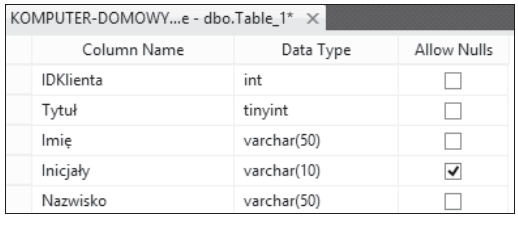
8. Ostatnią ważną opcją ustawianą podczas tworzenia kolumny jest *Allow Nulls* (dopuszczalne wartości `NULL`). Jeżeli jej nie zaznaczysz, kolumna będzie musiała zawierać dane. Jeżeli pozostawisz tę opcję w domyślnym stanie, zmienianym na zaznaczony po wpisaniu nazwy kolumny, będą dopuszczalne wartości `NULL`, co nie jest zalecanym ustawieniem, jeżeli kolumna musi zawierać dane (nazwisko klienta, numer zamówienia itp.) lub gdy kolumna będzie wykorzystywana w zapytaniach wyszukujących dane, w relacjach lub indeksach. Wartości `NULL` mogą być dozwolone w kolumnach przechowujących dane liczbowe, dzięki czemu zamiast wpisywać do niej zera, można nic nie wpisywać. W naszym przypadku wymagane będzie wpisywanie danych w każdym wierszu tabeli, ponieważ tworzona kolumna będzie wykorzystywana w relacjach. Dlatego musisz wyłączyć automatycznie zaznaczoną opcję.
9. Teraz sekcja *Column Properties* będzie wyglądała tak, jak na rysunku 5.5. Poświęćmy chwilę na przejrzenie jej zawartości. Widoczna jest w niej nazwa kolumny, informacja, czy dopuszczalne są wartości `NULL`, oraz typ przechowywanych danych. Zawartość tej sekcji zmienia się w zależności od wybranego typu danych.
10. Tworzona kolumna będzie służyła do identyfikacji danych w tabeli i łączenia jej z innymi tabelami. Jeżeli jeszcze nie skonfigurowałeś kolumny, rozwiń w panelu *Column Properties* sekcję *Identity Specification* (specyfikacja identyfikacyjna) i ustaw opcję *Is Identity* (do identyfikacji) na *Yes*, jak na rysunku 5.6. Opcje *Identity Increment* (przyrost wartości identyfikującej) oraz *Identity Seed* (początkowa wartość identyfikująca) zostaną ustawione na 1. Opcje przyrostu i wartości początkowej możesz zmieniać według uznania, ale zazwyczaj stosowane są domyślne wartości.
11. Teraz możesz dodać kolejne kolumny tabeli, jak na rysunku 5.7. Kolumna *IDTytułuKlienta* ma przypisany typ `integer`, ponieważ będzie służyła jako odniesienie do tabeli zawierającej tytuły, np. *Pani*, *Pan* itp. Dzięki temu nie będzie możliwości wpisywania niepożądanych dowolnych wartości, na przykład „P.", „pani" itp. Niewiele osób stosuje dodatkowe określenia oprócz imienia i nazwiska, ale niektórzy używają inicjałów. Dlatego w następnej kolumnie muszą być dopuszczalne wartości `NULL`. W definicji kolumny *Inicjał* zaznacz opcję *Allow Nulls*, jak na rysunku 5.7. Zmień długość danych w tej kolumnie na 10 znaków, co w zupełności powinno wystarczyć. Dla ułatwienia poniżej znajduje się lista kolumn:
    - *IDTytułuKlienta*: `tinyint`
    - *Imię*: `varchar(50)`
    - *Inicjały*: `varchar(10)` oraz dopuszczalne wartości `NULL`
    - *Nazwisko*: `varchar(50)`



Rysunek 5.5. *Dokładniejsze informacje o właściwościach kolumny*



Rysunek 5.6. *Definiowanie kolumny służącej do identyfikacji danych*



Rysunek 5.7. *Kolumna dopuszczająca wartości NULL*

12. Teraz zdefiniuj pozostałe kolumny pokazane na rysunku 5.8. Zalecam zaznaczenie opcji *Allow Nulls* w kolumnach *TypKonta* i *OperacjeKsięgowane*. Wartości `NULL` będą stanowiły dobry wskaźnik, że dany klient nie ma konta, ponieważ jest właśnie tworzone. Jest to dobry przykład zastosowania wartości `NULL` w celu uniknięcia pomyłek. Gdyby wprowadzanie danych w obu kolumnach było obowiązkowe, w kolumnie *IDTypuKonta* musiałaby być wpisywana jakaś wartość np. *„Brak konta”*. Podobnie wartość `NULL` w kolumnie *OperacjeKsięgowane* będzie oznaczać, że nie ma żadnych niezaksięgowanych operacji, natomiast wartość zero, że wszystkie operacje zostały zaksięgowane albo że suma operacji obciążających i uznających rachunek jest równa zeru. Ostatnie dwie kolumny będą informowały, czy konto jest otwarte, czy zamknięte. Dzięki wskazaniu jako domyślnej wartości w kolumnie *DataOtwarcia* funkcji serwera `GETDATE()`, zwracającej bieżącą datę, nie będzie trzeba wpisywać danych w tej kolumnie po każdorazowym dodaniu do tabeli

nowego wiersza, ponieważ serwer wpisze datę automatycznie (bez godziny, ponieważ kolumna zawiera tylko daty). Jako domyślną wartość można podać stałą, funkcję lub formułę. W kolumnie *DataZamknięcia* będą dopuszczalne wartości `NULL`, ponieważ wpisana w niej data będzie stanowiła doskonałą wskazówkę, że konto zostało całkowicie zamknięte. Pełny układ tabeli jest pokazany na rysunku 5.8, a poniżej znajduje się lista definiowanych kolumn:

- *Adres1*: `varchar(100)`
- *Adres2*: `varchar(100)` oraz dopuszczalne wartości `NULL`
- *Adres3*: `varchar(100)` oraz dopuszczalne wartości `NULL`
- *Miasto*: `int`
- *KodPocztowy*: `varchar(20)` oraz dopuszczalne wartości `NULL`
- *Województwo*: `tinyint`
- *TypKonta*: `tinyint` oraz dopuszczalne wartości `NULL`
- *OperacjeZaksięgowane*: `money` oraz dopuszczalne wartości `NULL`
- *OperacjeNiezaksięgowane*: `money` oraz dopuszczalne wartości `NULL`
- *DataOtwarcia*: `date` oraz domyślna wartość zdefiniowana jako funkcja `GETDATE()`
- *DataZamknięcia*: `date` oraz dopuszczalne wartości `NULL`

KOMPUTER-DOMOWY...e - dbo.Table_1*

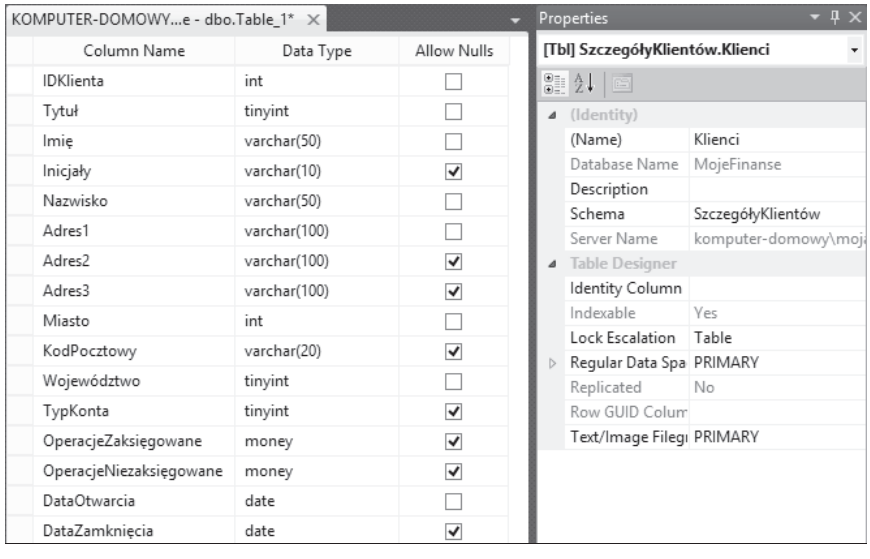
Column Name	Data Type	Allow Nulls
IDKlienta	int	☐
Tytuł	tinyint	☐
Imię	varchar(50)	☐
Inicjały	varchar(10)	☑
Nazwisko	varchar(50)	☐
Adres1	varchar(100)	☐
Adres2	varchar(100)	☑
Adres3	varchar(100)	☑
Miasto	int	☐
KodPocztowy	varchar(20)	☑
Województwo	tinyint	☐
TypKonta	tinyint	☑
OperacjeZaksięgowane	money	☑
OperacjeNiezaksięgowane	money	☑
DataOtwarcia	date	☐
DataZamknięcia	date	☑

Rysunek 5.8. *Kolumny tabeli zostały zdefiniowane*

■ **Ostrzeżenie** Podanie stałej jako domyślnej wartości w definicji kolumny pozwala oszczędzić czas, jednak pogarsza wydajność bazy danych, ponieważ serwer SQL Server przy każdorazowym wpisaniu nowego wiersza musi utworzyć i wpisać nową wartość. Wydajność pogarsza się jeszcze bardziej, jeżeli wartość domyślna jest opisana przy użyciu bardziej złożonej formuły. Jeżeli szybkość wpisywania danych jest kwestią priorytetową, rozważ zastosowanie innych metod.

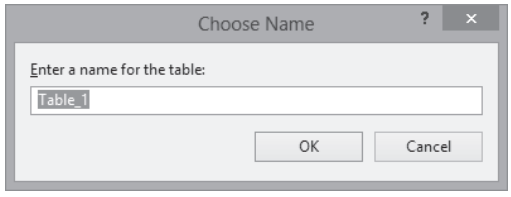
Uwaga W rozdziale 3., w opisie normalizacji bazy, wspomniałem, że dane należy zdenormalizować. Tabela *Klienci* jest jednym z miejsc, w którym wymagana jest niewielka denormalizacja, zwiększająca szybkość dostępu do danych. Aby przyspieszyć proces pobierania danych w chwili, gdy klient stoi przy bankomacie, została utworzona kolumna zawierająca numer konta, dzięki czemu do bankomatu będzie wysyłany tylko jeden wiersz danych. Następnie system sprawdzi numer karty klienta w celu wyświetlenia wartości operacji zaksięgowanych i niezaksięgowanych. Dane te będą dodatkowo przechowywane w zdefiniowanych później tabelach *ProduktyKlienta* i *Transakcje*. Gdyby numer konta klienta był zapisywany tylko w tabeli *ProduktyKlienta*, należałoby odczytywać dwa wiersze danych przed wysłaniem ich do bankomatu: jeden zawierający numer konta i drugi z wartością operacji. Jeżeli bankomatów będzie dużo i w godzinach szczytu będą realizować wiele transakcji, szybkość wykonywania operacji będzie miała krytyczne znaczenie i niewielkie zmiany w projekcie, jak wyżej opisana, mogą mieć ogromne znaczenie. Dla Ciebie, jako programisty, podstawowym zadaniem jest tworzenie optymalnie działających baz danych.

13. Zanim zapiszesz tabelę, musisz określić jej kilka właściwości, na przykład schemat. Po prawej stronie okna znajduje się panel *Properties*, pokazany na rysunku 5.9. Jeżeli go nie ma, naciśnij klawisz *F4* lub wybierz polecenie menu *View/Properties Window* (widok/okno właściwości). Przede wszystkim musisz w nim wpisać nazwę tabeli *Klienci* i jakiś opis. Następnie zajmij się schematem. Gdy rozwiniesz pole *Schema*, pojawi się lista dostępnych schematów, do których może być przypisana tabela. W rozdziale 4. utworzyłeś schemat *SzczegółyKlientów*, przeznaczony na tę tabelę.



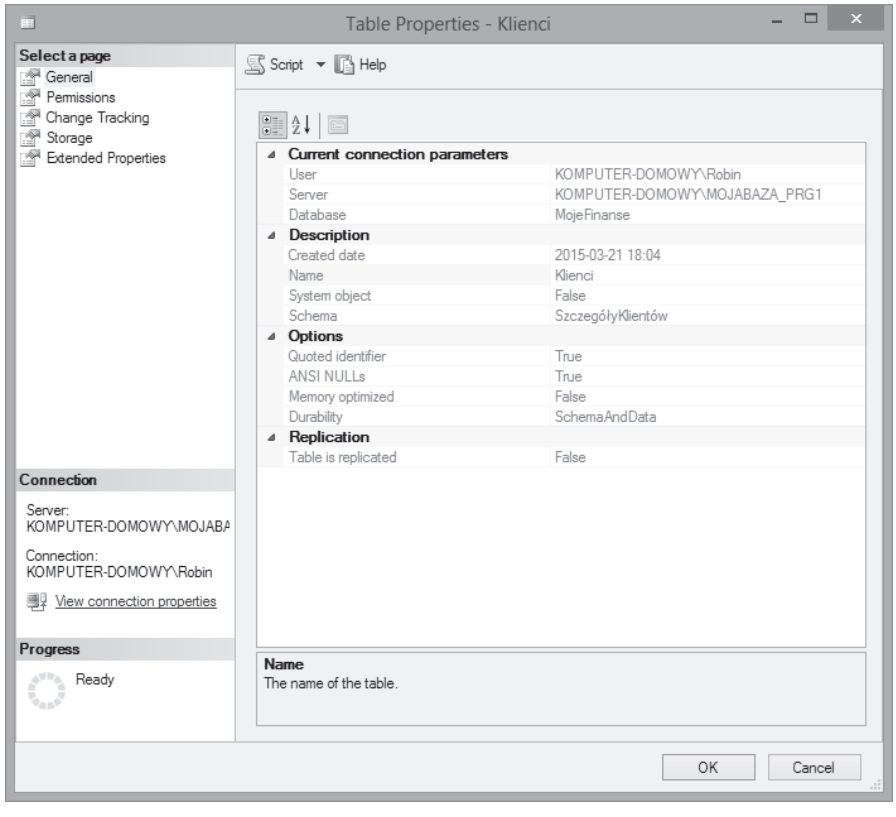
Rysunek 5.9. *Właściwości tabeli*

14. Teraz, gdy zakończyłeś określanie właściwości tabeli, możesz ją zapisać, klikając przycisk *Save* (zapisz) z ikoną dyskietki lub zamykając okno do projektowania tabel przez naciśnięcie symbolu *X*. Pojawi się komunikat z pytaniem, czy chcesz zapisać wprowadzone zmiany. Po kliknięciu przycisku *Yes* pojawi się następne okno, pokazane na rysunku 5.10, z pytaniem o nazwę tabeli, jeżeli nie wprowadziłeś jej wcześniej w panelu *Properties*. Jeżeli jednak to zrobiłeś, tabela zostanie zapisana pod podaną nazwą i wrócisz do aplikacji SQL Server Management Studio.



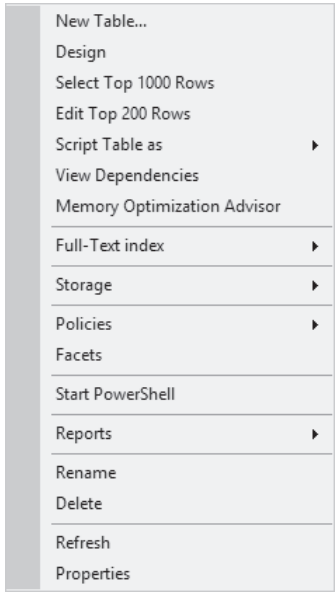
Rysunek 5.10. *Zapisywanie tabeli, której nazwa nie została określona*

15. Zawartość panelu *Object Explorer* może nie być automatycznie odświeżana i sekcja *Tables* może nie zawierać nowej tabeli. Jeżeli jej tam nie ma, zaznacz panel *Object Explorer* i odśwież jego zawartość, naciskając klawisz *F5* lub klikając ikonę odświeżania. Jeżeli teraz klikniesz tabelę prawym przyciskiem myszy i wybierzesz polecenie *Properties*, pojawi się okno pokazane na rysunku 5.11, zawierające ważne informacje o tabeli. Widoczna jest w nim data utworzenia tabeli, jej nazwa i nazwa schematu, do którego tabela należy.



Rysunek 5.11. *Właściwości tabeli*

Menu pojawiające się po kliknięciu tabeli prawym przyciskiem myszy zawiera wiele innych poleceń (patrz rysunek 5.12). Choć wiele z nich jest opisanych w różnych miejscach książki, opiszę je krótko, nim pójdziemy dalej.



Rysunek 5.12. *Podręczne menu tabeli*

- *New Table* (nowa tabela): opisane właśnie tworzenie nowej tabeli.
- *Design* (projektowanie): przejście do trybu projektowania (patrz rysunek 5.8), umożliwiającego zmianę definicji tabeli. To polecenie daje również dostęp do innych atrybutów tabeli, na przykład indeksów i relacji.
- *Select Top 1000 Rows* (wybierz pierwszych 1000 wierszy): wyświetlenie pierwszego tysiąca wierszy danych. Liczbę zwracanych wierszy można zmienić poleceniem menu *Tools/Options/SQL Server Object Explorer* (narzędzia/opcje/eksplorator obiektów SQL Server).
- *Edit Top 200 Rows* (edytuj pierwszych 200 wierszy): wyświetlenie i edycja pierwszych 200 wierszy tabeli. Liczbę wierszy również można zmienić poleceniem menu *Tools/Options/SQL Server Object Explorer*, jednak jej zwiększenie może przysporzyć problemów, szczególnie w przypadku wolnego łącza pomiędzy klientem a serwerem. Jest kilka sposobów użycia tej opcji w taki sposób, aby do zmiany danych było potrzebnych 200 lub mniej wierszy. Jednak stanowczo odradzam używania tej opcji w środowisku produkcyjnym.
- *Script Table as* (utwórz skrypt tabeli): wyświetlenie kilku opcji tworzenia skryptów, na przykład zawierających instrukcje `CREATE` lub `DROP` do tworzenia i usuwania tabeli, jak również `SELECT`, `DELETE`, `UPDATE` i `INSERT` do odczytywania, usuwania, zmieniania i wstawiania danych. Tworzenie tabeli za pomocą specjalnego okna do projektowania oraz generowanie skryptu, który można umieścić w repozytorium kodów źródłowych wraz z innymi kodami, jest bardzo wygodne. Jest to również doskonały sposób na wprowadzanie zmian za pomocą skryptów w produkcyjnych bazach danych. W dalszej części rozdziału dowiesz się, jak to zrobić.
- *View Dependencies* (pokaż zależności): wyświetlenie listy procedur składowanych, widoków i obiektów wykorzystujących daną tabelę. Jest to przydatne polecenie w sytuacji, gdy trzeba zmienić właściwości tabeli, na przykład dodać nowe kolumny. Jako programista będziesz nieustannie rozbudowywał swój system o nowe funkcjonalności, a wyszukiwanie zależności tabeli ułatwia rozeznanie się w obiektach wymagających sprawdzenia i ewentualnie zmiany.

■ **Ostrzeżenie** Nie zakładaj, że informacje dostępne za pomocą polecenia *View Dependencies* są w 100% wiarygodne. Podczas tworzenia procedur składowanych zdarza się, że procedura zależna nie istnieje. Po utworzeniu takiej procedury nie jest tworzona zależność. Jednakże takie przypadki nie zdarzają się często, a przyczyna problemu powinna być znana.

- *Memory Optimization Advisor* (doradca optymalizacji pamięci): otwarcie kreatora analizującego tabelę i przedstawiającego operacje wymagane do przekształcenia tabeli z zapisanej na dysku w zapisaną w pamięci. Kreator jest przedstawiony i omówiony w rozdziale 7.
- *Full-Text index* (indeks pełnotekstowy): obsługa specjalnego typu indeksu tabeli zawierającej dane tekstowe. Dane mogą być zapisane w sposób umożliwiający indeksowanie słów kluczowych. Przykładem może być system służący do przechowywania dokumentów Word i innych publikacji, w których słowa kluczowe są indeksowane, aby można było je łatwo wyszukiwać. Polecenie umożliwia również tworzenie wyszukiwarek, które zapisują strony WWW i wykorzystują przeszukiwanie pełnotekstowe.
- *Storage* (zapisywanie): obsługa zaawansowanej funkcjonalności serwera SQL Server, służącej do dzielenia, czyli **partycjonowania**, danych w tabeli. Partycjonowanie może być zastosowane w celu zwiększenia wydajności operacji wykonywanych na tabelach zawierających bardzo dużą liczbę wierszy. Jeżeli tabela składa się z milionów wierszy, część z nich może być umieszczona w jednej partycji, druga część w drugiej partycji itd. Ta funkcjonalność wykracza poza zakres niniejszej książki, a ponadto jej zastosowanie wymaga dokładnych przemyśleń. Została jednak krótko opisana w dalszej części rozdziału.
- *Policies/Facets* (polityki/obszary): przeglądanie i uruchamianie polityk dotyczących danej tabeli. Ta funkcjonalność została pokazana w rozdziale 4., w którym dowiedziałeś się, jak wymagać odpowiedniego ustawienia opcji *ANSI NULL* w bazie danych.
- *Start PowerShell* (otwórz powłokę PowerShell): otwarcie wiersza poleceń, umożliwiającego tworzenie skryptów w języku PowerShell.
- *Reports* (raporty): generowanie zdefiniowanych wcześniej raportów.
- *Rename* (zmień nazwę): zmiana nazwy tabeli. Zmiana nazwy tabeli na wczesnym etapie projektowania jest dopuszczalna, ale jeżeli nie jest to absolutnie konieczne, należy tego unikać. Jeżeli musisz zmienić nazwę tabeli, użyj polecenia *View Dependencies*, aby sprawdzić, jakie inne ustawienia należy jeszcze zmienić. Nie zapomnij też o zmianie kodu .Net lub Java, w którym może być użyta nazwa tabeli.
- *Delete* (usuń): usunięcie tabeli z bazy danych. Zanim usuniesz tabelę, bardzo zalecam użycie przycisku *Show Dependencies* w oknie do usuwania tabeli, będącego odpowiednikiem polecenia *View Dependencies*. Usuwanie tabeli opiszę pod koniec niniejszego rozdziału.
- *Refresh* (odśwież): odświeżenie listy tabel. Niekiedy aplikacja SQL Server Management Studio nie odświeża automatycznie ani nie aktualizuje na bieżąco listy tabel. Niektóre operacje, na przykład zwinięcie i ponowne rozwinięcie sekcji tabel, powodują odświeżenie listy, jednak wybranie opisanego polecenia, kliknięcie przycisku odświeżania w pasku narzędzi panelu *Object Explorer* lub naciśnięcie klawisza *F5* powoduje odświeżenie zawartości bieżącej sekcji. Jeżeli znajdujesz się na poziomie bazy danych, wówczas naciśnięcie klawisza *F5* lub kliknięcie przycisku odświeżania powoduje uaktualnienie listy wszystkich obiektów na wszystkich poziomach bazy danych.
- *Properties* (właściwości): otwarcie okna z właściwościami tabeli, jak na rysunku 5.11.

Teraz, po utworzeniu tabeli za pomocą aplikacji SQL Server Management Studio, sprawdźmy, jak można to samo zrobić za pomocą edytora zapytań.

Tworzenie tabeli za pomocą edytora zapytań

Kolejna tabela, którą trzeba utworzyć, będzie służyła do przechowywania szczegółów transakcji finansowych zrealizowanych przez każdego klienta. Będą to nie tylko proste operacje wpłaty i wypłaty gotówki, ale również wypłaty dywidendy od akcji i zwrot podatku w przypadku nieopodatkowanych akcji. Projekt bazy zakłada, że szczegóły produktu, którego dotyczy transakcja, będą przechowywane w osobnej tabeli. Musisz więc utworzyć połączenie pomiędzy tabelą z transakcjami i inną tabelą, zawierającą inne powiązane dane. Potrzebne będzie również połączenie pomiędzy tabelą z transakcjami a tabelą *SzczegółyKlientów.Klienci*. Zwróć uwagę na subtelną zmianę nazwy tabeli *Klienci*, zawierającej teraz, zgodnie z wcześniejszym opisem, nazwę schematu. Oznacza to, że w celu uniknięcia pomyłek powinieneś stosować konwencję *nazwa_schematu.nazwa_tabeli*.

Wróćmy do tabeli z transakcjami finansowymi. Jeżeli transakcja dotyczy akcji i nie jest wykonywana operacja finansowa, ten fakt powinien zostać odnotowany w bazie. Innymi słowy, jeżeli klient kupi akcje, w bazie danych powinny być utworzone dwa rekordy: jeden opisujący środki odejmowane z konta w celu zakupu akcji i drugi opisujący liczbę zakupionych akcji.

Teraz utwórzmy tabelę za pomocą edytora *Query Editor*, a nie aplikacji SQL Server Management Studio.

Wypróbuj: tworzenie tabeli za pomocą edytora zapytań

1. Upewnij się, że w edytorze *Query Editor* wybrana jest odpowiednia baza danych; w przeciwnym wypadku może okazać się, że utworzyłeś tabelę w niewłaściwej bazie. Jeżeli tworzysz kod, który będzie wykorzystany w programie napisanym w technologii .NET lub Java, warto umieścić na jego początku poniższy kod T-SQL. W ten sposób połączenie będzie zestawione z właściwą bazą danych. W kilku miejscach tej książki znajdziesz takie przypomnienie.

```
USE MojeFinanse
GO
```

2. W edytorze zapytań wpisz poniższy kod:

```
CREATE TABLE SzczegółyTransakcji.Transakcje
  (IDTransakcji int IDENTITY(1,1) NOT NULL,
  IDKlienta int NOT NULL,
  TypTransakcji smallint NOT NULL,
  DataUtworzenia datetime NOT NULL,
  Kwota numeric(18, 5) NOT NULL,
  Szczegóły nvarchar(50) NULL,
  Uwagi nvarchar(max) NULL,
  IDAkcji int NULL,
  IDProduktu int NOT NULL)
```

Uwaga Zwróć uwagę, że podczas wpisywania kodu w edytorze zapytań kolorowane są słowa kluczowe, na przykład słowa `CREATE TABLE` są niebieskie, a `NOT NULL` szare. Dzięki temu unika się pomyłek podczas wpisywania kodu. Kolory są zdefiniowane w opcjach dostępnych za pomocą polecenia *Tools/Options/Environment/Fonts and Colors* (narzędzia/opcje/środowisko/czcionki i kolory).

---

3. Uruchom kod, naciskając klawisz *F5* lub klikając przycisk *Execute* w pasku narzędzi.
4. W zakładce *Messages* powinien pojawić się następujący komunikat:

```
Command(s) completed successfully.
```

5. Może się jednak pojawić komunikat o błędzie. Powodów może być kilka, począwszy od pomyłki we wpisanym kodzie, do braku uprawnień do tworzenia tabel. Mógłbym sporządzić listę wszystkich komunikatów, które mogłyby się pojawić w tym momencie, ale zajęłaby ona kilka stron. W poniższym przykładzie komunikat informuje, że w wierszu 5. jest błąd:

```
Msg 102, Level 15, State 1, Line 5  // Komunikat 102., poziom 15., stan 1., wiersz 5.
Incorrect syntax near 'NUL'.        // Błędna składnia w pobliżu ciągu 'NUL'
```

6. Teraz przejdź do panelu *Object Explorer*. Jeżeli jest otwarty, musisz odświeżyć sekcję *Tables* (klikając ją prawym przyciskiem myszy i wybierając polecenie *Refresh*). Powinny być w niej widoczne tabele: utworzona wcześniej *SzczegółyKlientów.Klienci* oraz *SzczegółyTransakcji.Transakcje*.

Jak to działa: tworzenie tabeli za pomocą edytora zapytań

Tworzenie tabeli za pomocą edytora zapytań przebiega bardzo podobnie, jak za pomocą aplikacji SQL Server Management Studio, jednakże w tym przypadku nie ma graficznych ułatwień. Cały kod zostanie opisany za chwilę. Jak pamiętasz, aplikacja SQL Server Management Studio pyta o nazwę kolumny, typ danych itp., natomiast w edytorze sam musisz wpisać wszystkie szczegóły. Jednak wiele osób woli tworzyć tabele w ten właśnie sposób.

Wpisując kod ręcznie, pogłębiasz swoją znajomość składni T-SQL, dzięki czemu stajesz się bardziej wartościowy jako programista. Podczas rozmowy kwalifikacyjnej usłyszysz pytania dotyczące tworzenia tabel za pomocą kodu T-SQL, a nie aplikacji SQL Server Management Studio. Ponadto cały kod można zapisać w repozytorium kodu źródłowego, dzięki czemu będzie nie tylko chroniony, ale będzie również tworzył ścieżkę audytu podczas aktualizacji systemu. Znając składnię T-SQL, możesz tworzyć kod również dla innych baz danych, na przykład Sybase, i wykonywać wiele operacji w bazie Oracle. Zapytanie można zapisać w pliku w celu późniejszego wykorzystania i uruchomić je, aby utworzyć tabelę w bazie. Ponadto, jak już wspomniałem, ale co warto ponownie przypomnieć, kod T-SQL można wywoływać za pomocą dowolnego języka programowania i wykonywać operacje, których nie można zrealizować za pomocą interfejsu graficznego. Jeżeli chcesz odnieść sukces jako programista, niezbędna będzie znajomość kodu T-SQL.

Teraz przyjrzyjmy się kodowi T-SQL użytemu do utworzenia tabeli. Kod ten nie zawiera wszystkich dostępnych opcji, ponieważ jest ich bardzo wiele i nie są wykorzystywane w tej książce. Podstawowa składnia polecenia tworzącego tabelę jest następująca:

```
CREATE TABLE [nazwa_bazy].[nazwa_schematu].nazwa_tabeli
  (nazwa_kolumny typ_danych [długość] [IDENTITY(wartość_początkowa, przyrost)] [NULL/NOT NULL])
```

Liczba dostępnych opcji jest większa; przedstawiłem jedynie podstawową składnię. Za pomocą tego polecenia powinieneś być w stanie utworzyć większość tabel. W książce poznasz inne, niewymienione wyżej opcje, a niektóre z nich, na przykład dotyczące indeksów i kluczy, mogą być użyte oddzielnie. Jeszcze inne opcje dotyczą bardzo zaawansowanych zagadnień, nieopisanych w tej książce, takich jak wyszukiwanie pełnotekstowe czy blokowanie danych. Jeżeli nie masz doświadczenia, niewłaściwe użycie takich opcji może być niebezpieczne.

Opcje umieszczone w nawiasach kwadratowych są nieobowiązkowe, jednak w niektórych sytuacjach będą one potrzebne. Weźmy pierwszą opcję, *nazwa_bazy*. Jeżeli jesteś połączony z bazą *master* i chcesz utworzyć tabelę w bazie *MojeFinanse*, musisz przełączyć się na właściwą bazę, używając polecenia `USE`, wspomnianego w pierwszym punkcie powyższego ćwiczenia, lub użyć opcji *nazwa_bazy*. Zazwyczaj będziesz połączony z bazą, w której będziesz tworzył tabele, jednak opcja ta doskonale sprawdza się podczas tworzenia kodu, który będzie uruchamiany bez udziału użytkownika. Zamiast zakładać, że będziesz połączony z właściwą bazą, lepiej użyć opisanej opcji, która daje tę pewność.

Opcja *nazwa_schematu* umożliwia przypisanie tabeli do właściwego schematu, a nie do domyślnego schematu użytkownika połączonego z bazą.

W kolejnych wierszach definiowane są kolumny. Podanie nazwy i typu danych kolumny jest obowiązkowe. Jednak opcja *długość*, w zależności od typu danych, może nie być określona. Przed nazwą pierwszej kolumny musisz umieścić nawias otwierający (a po wpisaniu definicji ostatniej kolumny — nawias zamykający). Definicja każdej kolumny musi być oddzielona od poprzedniej przecinkiem. Tabela może zawierać maksymalnie 1024 kolumny. Jeżeli zbliżysz się do tej liczby, powinieneś zweryfikować projekt tabeli, ponieważ prawdopodobnie trzeba będzie ją zmienić.

Teraz, po utworzeniu jednej tabeli za pomocą aplikacji SQL Server Management Studio i drugiej za pomocą edytora zapytań, utwórzmy jeszcze jedną tabelę, używając narzędzia `sqlcmd`.

Tworzenie tabeli za pomocą narzędzia sqlcmd

Serwer SQL Server oferuje trzecią metodę tworzenia tabel, mianowicie z wykorzystaniem narzędzia `sqlcmd`. Jak wspomniałem w rozdziale 2., jest to narzędzie wywoływane w wierszu poleceń, umożliwiające uruchamianie w wybranej instancji serwera skryptów zapisanych w plikach. Ma ono kilka zastosowań, umożliwia m.in. łatwy dostęp do serwera w trybie jednego użytkownika. Administratorzy zajmujący się wieloma bazami danych, szczególnie pochodzącymi od różnych producentów, używają tego narzędzia, ponieważ oferuje ono jednolity interfejs. Jednym z ważniejszych jego zastosowań jest instalacja baz za pomocą skryptów. Gdy zaczniesz zajmować się dużymi bazami danych, zawierającymi setki procedur składowanych, tabel, funkcji i innych obiektów, każda instalacja bazy danych w środowisku produkcyjnym będzie wymagała wprowadzenia wielu zmian. Ponadto na potrzeby testowania skryptów będziesz korzystał z bazy odtworzonej z kopii zapasowej bazy produkcyjnej. Ładowanie i uruchamianie tych skryptów pojedynczo w aplikacji SQL Server Management Studio będzie co najmniej uciążliwe. Dzięki zastosowaniu opisanego narzędzia można w wierszu poleceń utworzyć proces, w którym uruchomienie gotowych skryptów będzie odbywało się w ramach jednej operacji. Narzędzie `sqlcmd` oferuje znacznie więcej możliwości niż wykonywanie poleceń i instrukcji T-SQL. Umożliwia na przykład odczytywanie zawartości plików zapisanych w określonym folderze.

W aplikacji SQL Server Management Studio w edytorze zapytań można włączyć tryb `sqlcmd`. W tym celu po otwarciu panelu edytora należy wybrać polecenie menu *Query/SQLCMD Mode* (zapytanie/tryb SQLCMD). W ten sposób możesz uruchamiać i tworzyć skrypty w trybie `sqlcmd` w aplikacji SQL Server Management Studio.

W poniższym przykładzie wykorzystasz w narzędziu `sqlcmd` pojedynczy plik. Nie będzie potrzeby używania żadnych dostępnych opcji tego narzędzia.

Wypróbuj: tworzenie tabeli za pomocą narzędzia sqlcmd

1. Otwórz zwykły edytor tekstu, na przykład Notatnik lub pusty edytor zapytań.
2. W wybranym edytorze utworzysz tabelę, która będzie przechowywać typy transakcji zapisanych w tabeli *SzczegółyTransakcji.Transakcje*. W tworzonym skrypcie umieścisz polecenie `USE`, mimo że narzędzie *sqlcmd* będzie połączone z bazą *MojeFinanse*. Dzięki poleceniu `USE` będziesz miał pewność, że skrypt zostanie wykonany we właściwej bazie, ponieważ wskazanie bazy danych podczas nawiązywania połączenia nie jest obowiązkowe i połączenie może zostać nawiązane z inną bazą, domyślną dla danego konta, na przykład *master*.

Poniżej znajduje się kod tworzący tabelę *SzczegółyTransakcji.TypyTransakcji*. Tabela powinna zawierać jeszcze czwartą kolumnę, ale nie będziesz definiował jej teraz. Zostanie ona utworzona w dalszej części rozdziału, w sekcji „Instrukcja ALTER TABLE", poświęconej modyfikacji tabeli.

```
USE MojeFinanse
GO
CREATE TABLE SzczegółyTransakcji.TypyTransakcji
(
   IDTypuTransakcji smallint IDENTITY(1,1) NOT NULL,
   OpisTransakcji varchar(50) NOT NULL,
   TypKredytu bit NOT NULL
)
GO
```

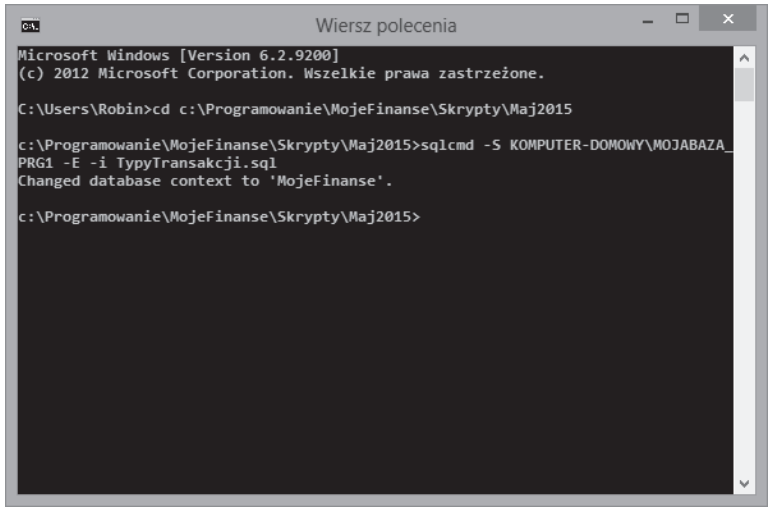
3. Kod po wpisaniu w edytorze zapisz w centralnym folderze o czytelnej nazwie. W tym przykładzie plik został zapisany w następującym miejscu:

   ```
   C:\Programowanie\MojeFinanse\Skrypty\Maj2015
   ```

4. Teraz otwórz wiersz poleceń, korzystając w systemie Windows 8 z listy aplikacji lub w systemie Windows 7 klikając polecenie *Start/Uruchom/cmd* i naciskając klawisz *Enter*.
5. Przejdź do folderu, w którym zapisałeś skrypt, aby łatwiej było go uruchomić. Teraz uruchom narzędzie `sqlcmd` z odpowiednimi parametrami. W tym przypadku są one następujące:
   - `-S` — nazwa serwera wraz z instancją,
   - `-E` — nawiązanie bezpiecznego połączenia (uwierzytelnianie Windows),
   - `-i` — nazwa uruchamianego skryptu.

   ```
   sqlcmd -S KOMPUTER-DOMOWY\MOJABAZA_PRG1 -E -i TypyTransakcji.sql
   ```

   Rysunekk 5.13 przedstawia wygląd wiersza poleceń po wykonaniu powyższego polecenia.



Rysunek 5.13. *Tworzenie tabeli za pomocą narzędzia sqlcmd*

6. Gdy powrócisz do aplikacji SQL Server Management Studio i odświeżysz sekcję *Tables* w bazie *MojeFinanse*, powinieneś znaleźć tam tabelę *SzczegółyTransakcji.TypyTransakcji*.

Teraz, po utworzeniu trzeciej tabeli, opiszę operację zmiany tabeli.

Instrukcja ALTER TABLE

Utworzona tabela *SzczegółyTransakcji.TypyTransakcji* zawiera tylko trzy kolumny, a potrzebne są cztery. Jednym ze sposobów dodania kolumny jest usunięcie tabeli za pomocą polecenia `DROP TABLE` i ponowne jej utworzenie. Jeżeli umieścisz w tabeli dane testowe, a potem stwierdzisz, że brakuje jakiejś kolumny, powyższy sposób — choć nie jest idealny — sprawdzi się. Jednak w środowisku produkcyjnym nie da się usunąć tabeli bez utraty danych. Jest jednak inne rozwiązanie, mianowicie instrukcja `ALTER TABLE`, umożliwiająca w ograniczonym stopniu zmianę definicji tabeli przy zachowaniu jej zawartości. Aplikacja SQL Server Management Studio wykorzystuje to polecenie w narzędziu do graficznej modyfikacji tabel, jednak tutaj pokażę, jak użyć tego polecenia w celu dodania brakującej czwartej kolumny tabeli *SzczegółyTransakcji.TypyTransakcji*.

Za pomocą polecenia `ALTER TABLE` można dodawać, usuwać i modyfikować kolumny tabeli. Usunięcie kolumny powoduje usunięcie danych tylko z danej kolumny i żadnej innej, jednak przed wykonaniem takiej operacji należy się dobrze zastanowić.

Podczas dodawania nowej kolumny należy rozważyć dwa przypadki: czy wszystkie istniejące wiersze mają zawierać wartości `NULL`, czy też powinna być użyta jakaś wartość domyślna? Dodanie za pomocą polecenia `ALTER TABLE` nowej kolumny, która powinna zawierać dane i dlatego musi być zdefiniowana z opcją `NOT NULL`, wymaga poświęcenia trochę czasu. Jest tak dlatego, ponieważ we wszystkich istniejących wierszach w nowej kolumnie będą znajdowały się wartości `NULL` albo wartości domyślne. Bez zdefiniowanych zasad biznesowych serwer SQL Server nie jest w stanie określić, jakie wartości powinny być automatycznie wpisane. Gdybyś na przykład po utworzeniu tabeli *SzczegółyKlientów.Klienci* dodał do niej kolumnę *DataOtwarcia*, serwer SQL Server nie miałby żadnej możliwości określenia daty otwarcia rachunku dla każdego klienta z osobna, ponieważ nie jest możliwy dostęp do warunków umowy zawartej z klientem.

■ **Uwaga** Praktyką często stosowaną podczas tworzenia nowych kolumn jest zezwolenie na umieszczanie wartości `NULL`, ponieważ określona wartość domyślna może nie być właściwa w niektórych wierszach.

W kolejnym ćwiczeniu dodasz nową kolumnę do tabeli bez określania wartości domyślnej. Kolumna będzie mogła zawierać wartości `NULL`. Później zmienisz tabelę tak, aby wartość `NULL` nie była dozwolona.

Wypróbuj: dodawanie nowej kolumny do tabeli

1. Przede wszystkim otwórz edytor zapytań i upewnij się, że jesteś połączony z bazą *MojeFinanse*. Następnie wpisz kod dodający nową kolumnę do tabeli *SzczegółyTransakcji.TypyTransakcji*. Format polecenia jest bardzo prosty. Po słowach `ALTER TABLE` musisz podać nazwę tabeli poprzedzoną nazwą schematu, a następnie listę rozdzielonych przecinkami kolumn, które chcesz dodać. W tym przypadku będzie to tylko jedna kolumna. Musisz określić nazwę kolumny, typ danych, ich długość (jeżeli jest wymagana) i na koniec zadecydować, czy będą dopuszczalne wartości `NULL`. Ponieważ nie chcesz, aby w istniejących wierszach były umieszczane wartości domyślne, musisz zdefiniować kolumnę dopuszczającą użycie wartości `NULL`.

```
ALTER TABLE SzczegółyTransakcji.TypyTransakcji
ADD ZmianaStanuKonta bit NULL
GO
```

2. Jeżeli tabela zawiera dane, możesz je zmienić tak, aby każdy wiersz zawierał w nowej kolumnie poprawną wartość, i zablokować możliwość wpisywania w nowych wierszach wartości `NULL`. W przypadku tabeli *SzczegółyTransakcji.TypyTransakcji* wartość `NULL` w kolumnie *ZmianaStanuKonta* nie jest dopuszczalna, dlatego trzeba zmienić opcję `NULL`. Nowa kolumna będzie zawierała wartości 0 lub 1, oznaczające wartości logiczne *prawda* i *fałsz*. Musisz zatem ponownie użyć polecenia `ALTER TABLE`, ale tym razem wykorzystasz w nim instrukcję `ALTER COLUMN` z nazwą kolumny, którą chcesz zmienić. Za tą instrukcją musisz określić

zmiany, jakie zamierzasz wprowadzić. Chociaż nie zmieniasz zdefiniowanego dla tej kolumny typu danych, musisz ponownie wskazać ich typ i długość. Na koniec powinieneś poinformować serwer SQL Server, że kolumna nie może zawierać wartości NULL.

```
ALTER TABLE SzczegółyTransakcji.TypyTransakcji
ALTER COLUMN ZmianaStanuKonta bit NOT NULL
GO
```

■ **Uwaga** Gdyby tabela *SzczegółyTransakcji.TypyTransakcji* zawierała dane, musiałbyś najpierw uaktualnić każdy wiersz tak, aby w zmienianej kolumnie znajdowały się wartości 0 lub 1. Dopiero wtedy serwer SQL Server mógłby pomyślnie zmienić właściwości kolumny.

3. Uruchom powyższy kod, aby skorygować tabelę *SzczegółyTransakcji.TypyTransakcji*.

Tworzenie pozostałych tabel

Teraz, po utworzeniu trzech tabel, musisz zdefiniować kolejne cztery. Osiągniesz to za pomocą kodu wpisywanego w edytorze zapytań. Następna sekcja nie zawiera żadnych nowych informacji, dlatego podany jest w niej tylko kod. Wpisz go zatem i uruchom. Potem otwórz aplikację SQL Server Management Studio i odśwież zawartość panelu *Object Explorer*, w którym zobaczysz nowe tabele. Zwróć uwagę, że tabele *SzczegółyKlientów.ProduktyKlientów* oraz *SzczegółyKlientów.ProduktyFinansowe* nie mają kolumn z opcją IDENTITY. Te dwie tabele zostaną wykorzystane w rozdziale 17. do zaprezentowania działania obiektu SEQUENCE, stanowiącego alternatywę dla opcji IDENTITY.

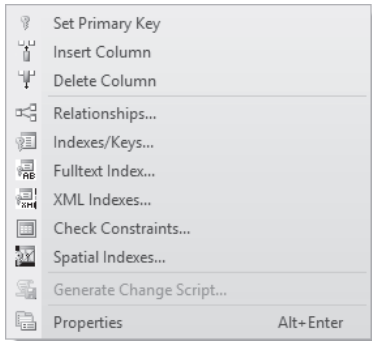
```
USE MojeFinanse
GO
CREATE TABLE SzczegółyKlientów.ProduktyKlientów(
  IDProduktuKlienta int NOT NULL,
  IDKlienta int NOT NULL,
  IDProduktuFinansowego int NOT NULL,
  Kwota money NOT NULL,
  Częstotliwość smallint NOT NULL,
  OstatniaWpłata datetime NOT NULL,
  OstatniaNależność datetime NOT NULL,
  Odnawialny bit NOT NULL
)
ON [PRIMARY]
GO
CREATE TABLE SzczegółyKlientów.ProduktyFinansowe(
  IDProduktu int NOT NULL,
  NazwaProduktu nvarchar(50) NOT NULL
) ON [PRIMARY]
GO
CREATE TABLE SzczegółyAkcji.Akcje(
  IDAkcji int IDENTITY (1,1) NOT NULL,
  Opis varchar(50) NOT NULL,
  SymbolAkcji varchar(50) NULL,
  BieżącaCena numeric(18,5) NOT NULL
)
GO
CREATE TABLE SzczegółyAkcji.CenyAkcji(
  IDCenyAkcji bigint IDENTITY(1,1) NOT NULL,
  IDAkcji int NOT NULL,
  Cena numeric(18, 5) NOT NULL,
  Data datetime NOT NULL
) ON [PRIMARY]
GO
```

Definiowanie klucza podstawowego

W aplikacji SQL Server Management Studio klucz podstawowy można zdefiniować kilkoma kliknięciami myszy. W tej sekcji zobaczysz, jakie to proste. W rozdziale 3. opisałem sposób wykorzystania kluczy podstawowych do powiązania tabeli z innymi tabelami i zapewnienia spójności danych. W rozdziale 6. opisane zostały różnice między kluczem podstawowym a indeksem.

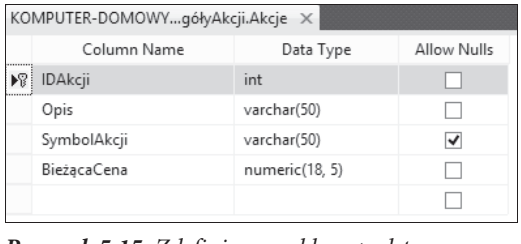
Wypróbuj: definiowanie klucza podstawowego

1. Upewnij się, że aplikacja SQL Server Management Studio jest otwarta, i wybierz w niej bazę *MojeFinanse*. Odszukaj tabelę *SzczegółyAkcji.Akcje*, kliknij ją prawym przyciskiem i wybierz polecenie *Design* (projektuj).
2. W oknie do projektowania tabeli zaznacz kolumnę *IDAkcji*. Dla tej kolumny zostanie utworzony klucz podstawowy. Kliknij kolumnę prawym przyciskiem, aby otworzyć podręczne menu, pokazane na rysunku 5.14.



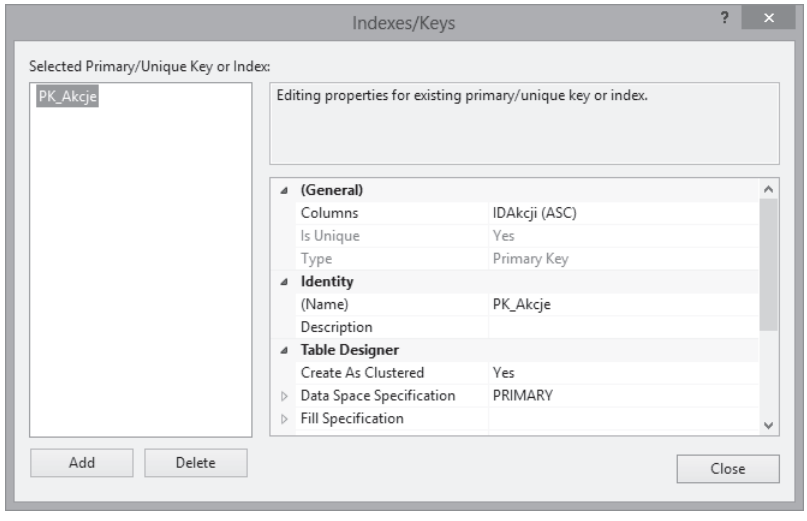
Rysunek 5.14. *Definiowanie klucza podstawowego*

3. Wybierz polecenie *Set Primary Key* (zdefiniuj klucz podstawowy). Zawartość okna zmieni się. W pierwszej kolumnie pojawi się mały symbol klucza. Klucz podstawowy został zdefiniowany tylko dla jednej kolumny, jak pokazuje rysunek 5.15.



Rysunek 5.15. *Zdefiniowany klucz podstawowy*

4. Jednak to, co widać, to nie są wszystkie zmiany. Zmiany wprowadzone do tabeli zapisz, klikając przycisk *Save* lub naciskając klawisze *Ctrl+S*. W pasku narzędzi okna do projektowania tabel kliknij czwarty przycisk od lewej, *Manage Indexes and Keys* (zarządzaj indeksami/kluczami). Pojawi się okno pokazane na rysunku 5.16. Zwróć uwagę na trzecią opcję, *Type* (typ), w sekcji *General*. Ma ona wartość *Primary Key*. Ponadto, przygotowana została za Ciebie definicja klucza, obejmująca nazwę, wskazanie kolumny i informację, że klucz jest unikatowy i klastrowany (więcej informacji na temat indeksów i ich relacji z kluczami podstawowymi zawartych jest w rozdziale 6.).



Rysunek 5.16. *Okno z informacjami o indeksach i kluczach podstawowych*

To wszystko na temat tworzenia i konfigurowania klucza podstawowego. Klucz podstawowy w tabeli *SzczegółyAkcji.Akcje* został skonfigurowany. W tym przykładzie każdy rekord dodawany do tabeli będzie miał nadawany kolejny numer w kolumnie *IDAkcji* (jest to cecha indeksu, jak się dowiesz w rozdziale 6.) i nie będzie możliwości wprowadzenia duplikatu wiersza. Klucz ten będzie później użyty również do połączenia tabeli z innymi tabelami.

Tworzenie relacji

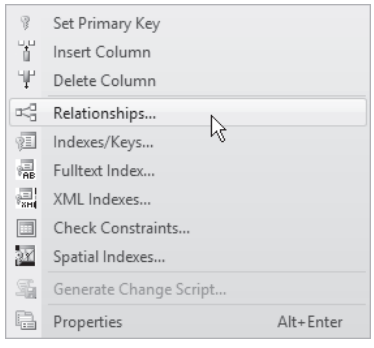
Relacje opisałem w rozdziale 3., jednak do tej pory nie zostały one utworzone. W tej sekcji najpierw utworzysz relację pomiędzy tabelami zawierającymi dane o klientach i o transakcjach. Będzie to relacja typu jeden-do-wielu, wiążąca jeden rekord klienta z wieloma rekordami transakcji. Pamiętaj, że choć dla jednego klienta może być utworzonych wiele rekordów — po jednym dla każdego zakupionego produktu — relacja będzie łączyła rekord z klientem i produktem z rekordami transakcji, ponieważ dla każdego nowego zakupionego produktu będzie tworzony nowy identyfikator w kolumnie *IDKlienta*. Teraz utwórz swoją pierwszą relację.

Wypróbuj: tworzenie relacji

1. Upewnij się, że aplikacja SQL Server Management Studio jest otwarta oraz że wybrana i rozwinięta jest sekcja *MojeFinanse*. Przede wszystkim musisz utworzyć klucz podstawowy w tabeli *SzczegółyKlientów.Klienci*. Wpisz i wykonaj następujący kod:

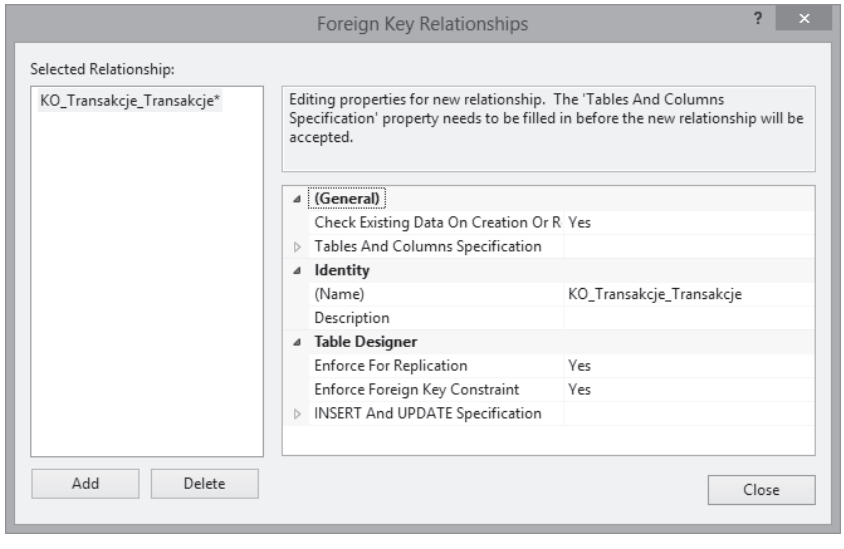
```
ALTER TABLE SzczegółyKlientów.Klienci
ADD CONSTRAINT
  PK_Klienci PRIMARY KEY CLUSTERED
  (
  IDKlienta
  )
WITH( STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
GO
```

2. Odszukaj i zaznacz tabelę *SzczegółyTransakcji.Transakcje*, kliknij ją prawym przyciskiem myszy, aby otworzyć podręczne menu. Wybierz polecenie *Design*, aby otworzyć okno do projektowania tabel.
3. W oknie do projektowania tabel kliknij w dowolnym miejscu prawym przyciskiem myszy i z podręcznego menu wybierz polecenie *Relationships* (relacje) lub w pasku narzędzi kliknij przycisk *Relationships*, trzeci od lewej.



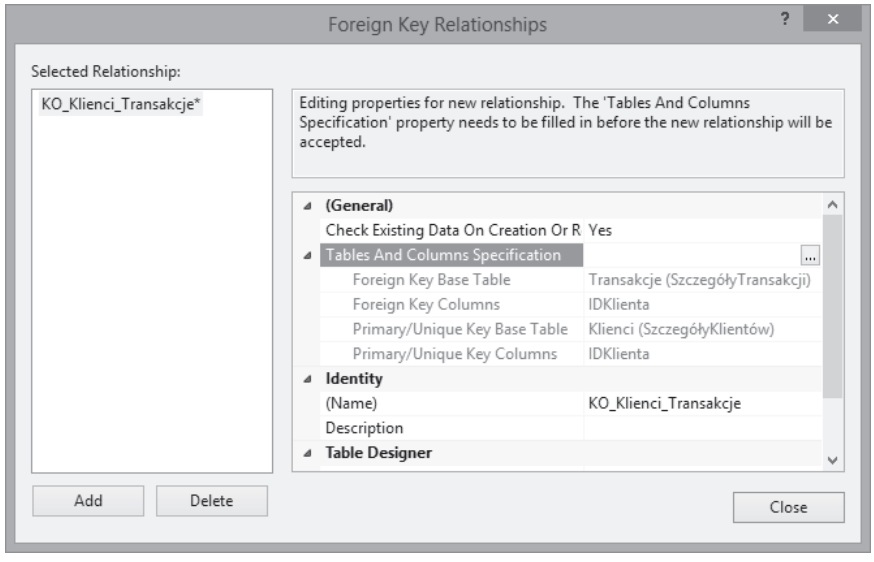
Rysunek 5.17. *Tworzenie relacji*

4. Pojawi się okno *Foreign Key Relationships* (relacje oparte na kluczach obcych). W tej chwili jest ono puste. Kliknij przycisk *Add* (dodaj relację). W oknie pojawią się informacje jak na rysunku 5.18.



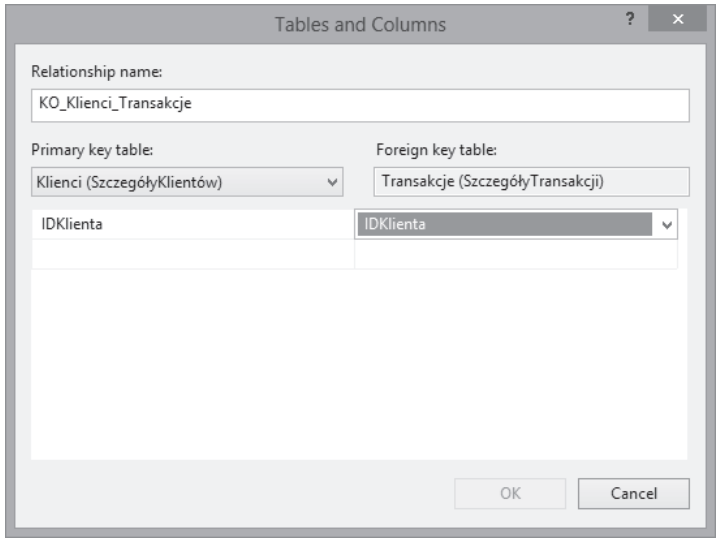
Rysunek 5.18. *Okno Foreign Key Relationships*

5. Najpierw musisz zmienić nazwę na bardziej czytelną. Nierzadko przekonasz się, że najlepiej jest nadawać nazwy według wzorca *KO_TabelaNadrzędna_TabelaPodrzędna* (lub z użyciem skrótów). W tym przypadku zmień nazwę na *KO_Klienci_Transkacje*, ponieważ w relacji wykorzystującej klucz obcy tabelą nadrzędną będzie *SzczegółyKlientów.Klienci*. Następnie rozwiń sekcję *Tables And Columns Specification* (specyfikacja tabel i kolumn), w której można zdefiniować relację. Zwróć uwagę, że po prawej stronie sekcji znajduje się przycisk z wielokropkiem, jak na rysunku 5.19. Aby utworzyć relację, kliknij ten przycisk.



Rysunek 5.19. *Wybór tabel i kolumn tworzących relację*

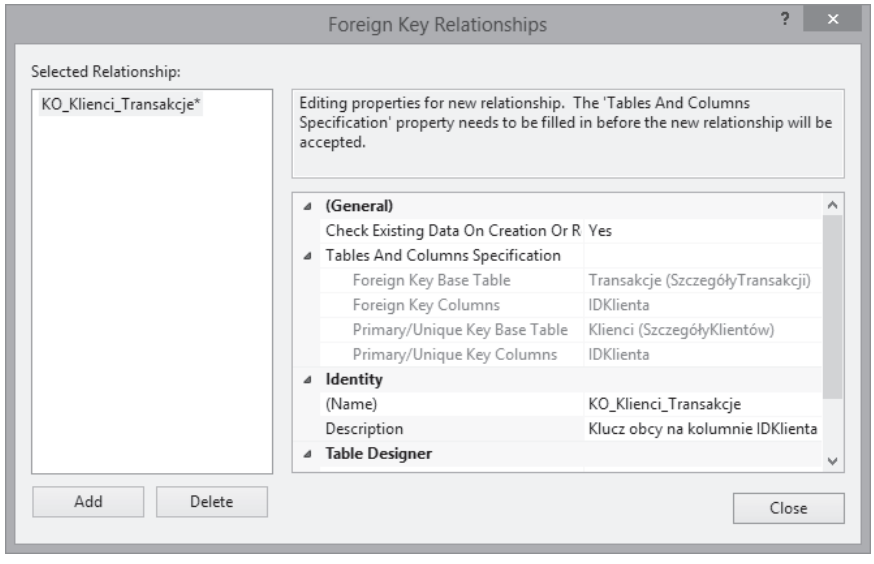
6. Teraz w każdej tabeli musisz wskazać kolumny, które będą tworzyły relację. Każdy rekord klienta będzie połączony z wieloma rekordami transakcji i do tego celu możesz wykorzystać kolumnę *IDKlienta*. Wybierz tę kolumnę w obu tabelach, jak na rysunku 5.20, i kliknij przycisk *OK*.



Rysunek 5.20. *Wybieranie kolumn tworzących relację*

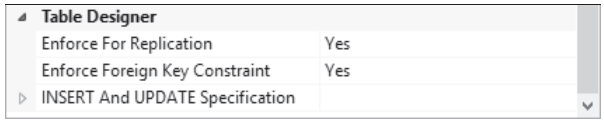
■ **Uwaga** W tym przykładzie obie kolumny mają takie same nazwy, jednak nie jest to regułą. Obie kolumny muszą jedynie zawierać te same informacje.

7. Powrócisz do okna *Foreign Key Relationships*, pokazanego na rysunku 5.21. Zwróć uwagę, że w jego górnej części w szarym obszarze znajdują się szczegóły zdefiniowanego właśnie klucza obcego. W sekcji *Identity* znajduje się również opis tego klucza.



Rysunek 5.21. *Definicja klucza obcego*

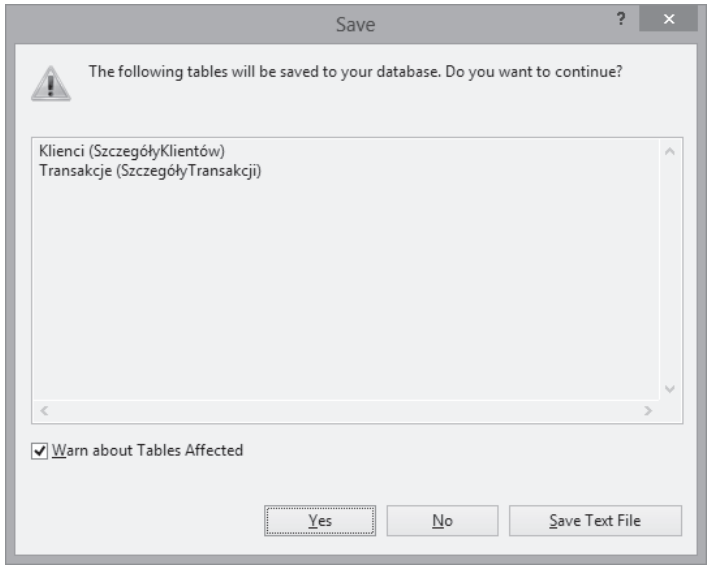
8. Gdy przewiniesz listę w dół, znajdziesz w niej opcję *Enforce for Replication* (sprawdzanie klucza podczas replikacji danych), której nie musisz zmieniać. Replikacja danych oznacza, że dane umieszczane w jednej bazie są replikowane w innej. Możesz wymusić sprawdzanie klucza obcego podczas wstawiania danych z bazy źródłowej do docelowej (w tej tabeli).
9. W dolnej części okna znajdują się jeszcze dwie inne opcje, pokazane na rysunku 5.22. Pozostaw ich wartości domyślne; opiszę je za chwilę.



Rysunek 5.22. *Opcje INSERT And UPDATE Specification*

10. Zamknięcie okna nie spowoduje zapisania wprowadzonych zmian. Nawet jeżeli zapiszesz zmiany, klikając przycisk *Save*, relacja nie zostanie utworzona. Zmiany zostaną wprowadzone dopiero wtedy, gdy zapiszesz lub zamkniesz okno do projektowania tabel. Gdy tak zrobisz, pojawi się okno pokazane na rysunku 5.23, zawierające informację, że obie tabele zostaną zmienione. Aby zapisać zmiany, kliknij przycisk *Yes*.

   Relacja została utworzona, ale co oznaczają opcje, które nie były zmieniane? Poznajmy je teraz.



Rysunek 5.23. *Zapisywanie relacji*

Opcja Check Existing Data on Creation

Jeżeli w którejkolwiek z obu tabel znajdują się dane, ustawienie na *Yes* opcji *Check Existing Data on Creation* (sprawdź istniejące dane w momencie utworzenia relacji), znajdującej się na początku listy w oknie do tworzenia relacji, spowoduje, że z chwilą zapisania relacji serwer SQL Server sprawdzi dane w tabelach. Jeżeli dane będą spełniały warunki relacji, zostanie ona pomyślnie utworzona. Jeżeli jednak jakieś dane nie spełnią warunków, relacja nie powstanie. Na przykład konieczne jest sprawdzenie, czy każda transakcja ma odpowiadający jej rekord klienta. Jeżeli jakieś transakcje nie będą miały odpowiednich rekordów klientów, relacja nie zostanie utworzona. Oczywiście, w takiej sytuacji będziesz musiał podjąć decyzję. Możesz poprawić dane, dodać nadrzędne rekordy w tabeli *Klienci* lub zmienić tabelę *Transakcje* i ponownie utworzyć relację albo zmienić opcje relacji.

Tworząc relację, narzucasz wymaganie, że dane będą poprawne, dlatego możesz wybrać opcję *No*, jeżeli wrócisz do tabel i poprawisz wprowadzone dane. Co się stanie, jeżeli wciąż będzie brakować w nich wierszy? Czy będzie to problem? W opisanym przykładzie wciąż będą istniały rekordy transakcji bez odpowiadających im rekordów klientów. Musisz jednak utworzyć relację, aby uniknąć powstawania takich rozbieżności w przyszłości.

Opcja Enforce Foreign Key Constraints

Po utworzeniu relacji w bazie danych można zapobiec naruszaniu warunków relacji. Jeżeli ustawisz opcję *Check Existing Data on Creation* na *Yes*, jak opisałem wcześniej, możesz być pewny, że spójność danych będzie zachowana. Opcja ta powoduje sprawdzenie tylko istniejących danych i nie ma wpływu na operacje dodawania, usuwania i zmieniania danych. Jednak ustawiając na *Yes* opcję *Enforce Foreign Key Constraints* (wymuś warunki klucza obcego), znajdującą się w dolnej części listy w oknie do projektowania relacji, będziesz miał pewność, że podczas dodawania, usuwania i zmieniania danych warunki relacji nie zostaną naruszone. Zmienianie i usuwanie danych będzie możliwe pod warunkiem, że zachowana będzie ich spójność. Na przykład będzie można zmienić identyfikator klienta w transakcji, o ile identyfikator ten będzie istniał w tabeli *SzczegółyKlientów.Klienci*. Przy próbie dodania przez aplikację danych, które naruszą warunki relacji, zostanie zgłoszony błąd i zmiany w tabeli nie zostaną wprowadzone.

Wybór reguł usuwania i zmieniania danych

Ostatnim zagadnieniem wymagającym omówienia są opcje relacji w sekcji *INSERT And UPDATE Specification* (specyfikacja operacji INSERT i UPDATE), znajdującej się na końcu listy w oknie do projektowania relacji. Są to opcje dotyczące operacji usuwania i zmieniania danych w tabeli nadrzędnej, w tym przypadku *SzczegółyKlientów.Klienci*. W zależności od ustawień na danych mogą być wykonywane następujące operacje:

- *No Action* (brak operacji): jeżeli zostanie usunięty rekord klienta, nie będą wykonywane żadne dalsze operacje, co potencjalnie może spowodować, że w bazie będą zapisane transakcje z identyfikatorami nieistniejących klientów w kolumnie *IDKlienta*. Zaleta tego ustawienia polega na tym, że możesz jako programista napisać kod usuwający transakcje klienta i Ty będziesz kontrolował tę operację. Jest to doskonałe rozwiązanie w przypadku, gdy dane są kopiowane z jednej bazy do innej.
- *Cascade* (kaskada): po usunięciu rekordu klienta usuwane będą również wszystkie rekordy transakcji związane z tym klientem.
- *Set Null* (wpisz wartość NULL): po usunięciu rekordu klienta w kolumnie *IDKlienta* w tabeli *SzczegółyTransakcji.Transakcje* zostaną wpisane wartości `NULL`, o ile będą dopuszczalne. Podczas omawiania zależności między klientami a transakcjami określiliśmy, że kolumna ta nie może zawierać wartości `NULL`, zatem będą pozostawione jej oryginalne wartości. W takim przypadku skutek będzie taki sam, jak po wybraniu opcji *No Action*. Niebezpieczeństwo jednak polega na tym, że w tabeli będą istnieć niepowiązane rekordy. Taki scenariusz jest dopuszczalny, jednak należy zachować ostrożność.
- *Set Default* (wpisz domyślną wartość): podczas definiowania tabeli możesz określić domyślną wartość, która będzie wpisywana do kolumny. Wybranie opisywanej opcji oznacza, że w kolumnie zostanie z powrotem wpisana domyślna wartość. Jest to również niebezpieczne ustawienie, aczkolwiek mniej niż w przypadku użycia opcji *Set Null*, jeżeli domyślna wartość jest poprawna.

Uwaga Jeżeli w którymś momencie postanowisz zaimplementować kaskadowe usuwanie danych, postępuj bardzo ostrożnie, ponieważ mogą zostać usunięte potrzebne dane. Jeżeli z tabeli *SzczegółyKlientów.Klienci* usuniesz rekord klienta, zostaną również usunięte wszystkie jego transakcje. Jest to dobre rozwiązanie, jeżeli będziesz miał zapasową bazę danych, w której będą archiwizowane wszystkie rekordy, jednak niewłaściwe, jeżeli będziesz analizował informacje finansowe. Aby system był spójny i szybki, stosuj kaskadowe usuwanie danych do sprawnego i pełnego usuwania danych klienta, który zamknął swój rachunek.

Tworzenie relacji za pomocą kodu T-SQL

Relację lub ograniczenie można również utworzyć za pomocą instrukcji T-SQL. Służy do tego polecenie `ALTER TABLE`. W poniższym przykładzie zostanie utworzona relacja pomiędzy tabelami *SzczegółyTransakcji.Transakcje* a *SzczegółyAkcji.Akcje*. W ten sposób zostanie zdefiniowana zależność pomiędzy tabelą z transakcjami a tabelą z akcjami, dotycząca transakcji z akcjami. Zostanie utworzona relacja typu zero- lub jeden-do-jednego. Poświęćmy chwilę na poznanie składni kodu T-SQL służącego do tworzenia relacji:

```
ALTER TABLE tabela_podrzędna
WITH NOCHECK|CHECK
ADD CONSTRAINT [nazwa_ograniczenia]
FOREIGN KEY (kolumna_tabeli_podrzędnej, ...,)
REFERENCES [tabela_nadrzędna]([kolumna_tabeli_nadrzędnej, ...,])
```

Do utworzenia relacji służy polecenie `ALTER TABLE`. Po podaniu nazwy w tym poleceniu musisz zadecydować, czy po utworzeniu klucza obcego mają być sprawdzone istniejące dane. Jest to opcja podobna do opisanej wcześniej opcji *Check Existing Data on Creation*.

Teraz przejdźmy do utworzenia relacji. W tym celu musisz przede wszystkim powiadomić serwer SQL Server o planowanej operacji, zatem użyj polecenia `ADD CONSTRAINT`.

Następnie nadaj nazwę tworzonej relacji. Podobnie jak poprzednio, zalecam używanie znaków podkreślenia zamiast spacji. Jeżeli jednak musisz użyć spacji, co szczerze odradzam, umieść nazwę w nawiasach kwadratowych ([]). Wiem, że pisałem o tym wcześniej, ale bardzo ważna jest świadomość skutków używania spacji w nazwach kolumn, tabel i relacji. Za każdym razem, gdy będziesz chciał użyć obiektu zawierającego spacje w nazwie, będziesz musiał ją umieścić w nawiasach. Po co przydawać sobie dodatkowej pracy?

Ponadto, musisz poinformować serwer, czy ma on sprawdzić istniejące dane i czy zostały dotrzymane warunki nowej relacji opartej na kluczu obcym. Powszechnie stosowaną praktyką jest użycie opcji `WITH CHECK`, powodującej sprawdzenie poprawności danych. Można nie sprawdzać danych, jednak musisz mieć wtedy świadomość, że istniejące dane mogą spowodować błędne działanie Twojej aplikacji.

Po podaniu nazwy relacji kolejnym krokiem jest poinformowanie serwera za pomocą polecenia `FOREIGN KEY`, że będzie zdefiniowany klucz obcy. Jak pamiętasz, relacja może być również użyta do innych celów, na przykład wstawiania do kolumny domyślnej wartości.

Podczas definiowania klucza obcego upewnij się, że nazwy wszystkich kolumn są rozdzielone przecinkami i zamknięte w nawiasach. Ostatnim krokiem w tworzeniu relacji za pomocą kodu jest wskazanie wykorzystywanej w niej tabeli nadrzędnej i jej kolumn. Obowiązuje przy tym zasada, że jednemu rekordowi w tabeli nadrzędnej musi odpowiadać tylko jeden rekord w tabeli podrzędnej, a ponadto typ danych w obu kolumnach musi być taki sam.

Jak widać, operacja jest prosta. Do tworzenia relacji możesz wykorzystać aplikację SQL Server Management Studio, ponieważ dzięki niej jest o wiele mniej wpisywania, a ponadto dokładnie widoczne są zależności pomiędzy kolumnami i można sprawdzić, czy są one tego samego typu. Jednak kod T-SQL można zapisać, by użyć go później w razie potrzeby w środowisku produkcyjnym. Ponadto, kopię kodu można przechowywać w repozytorium, na przykład Visual SourceSafe.

Wypróbuj: tworzenie relacji za pomocą kodu T-SQL

1. W edytorze zapytań wpisz poniższy kod i ruchom go, naciskając klawisz *F5* lub klikając przycisk *Execute*:

```
USE MojeFinanse
GO
ALTER TABLE SzczegółyTransakcji.Transakcje
WITH NOCHECK
ADD CONSTRAINT KO_Transakcje_Akcje
FOREIGN KEY(IDAkcji)
REFERENCES SzczegółyAkcji.Akcje(IDAkcji)
```

2. Powinien pojawić się komunikat o pomyślnym wykonaniu polecenia:

```
Command(s) completed successfully.
```

To wszystko. Relacja została utworzona w drugiej części kodu. Pierwsza zapewnia połączenie się z właściwą bazą danych.

Mimo że w powyższym kodzie użyte jest polecenie `ALTER TABLE`, nie są zmieniane żadne kolumny; tworzone jest jedynie ograniczenie. Relacja jest specjalnego rodzaju ograniczeniem.

Ograniczenie w swojej istocie jest mechanizmem w serwerze SQL Server sprawdzającym poprawność zmienianych danych i związanych z nimi tabel.

Kontrola spójności danych: baza danych czy aplikacja?

Na zakończenie rozdziału pozostał do omówienia jeszcze jeden krótki temat. Dowiedziałeś się, w jaki sposób tworzy się relacje między dwiema tabelami. Metoda ta ma tę zaletę, że referencyjna spójność danych jest gwarantowana przez serwer SQL Server. Jeżeli wykonywany kod T-SQL naruszy tę spójność, serwer zgłosi błąd.

Jednak jest jeszcze inna metoda, polegająca na utrzymaniu referencyjnej spójności danych za pomocą kodu. Można ją osiągnąć na kilka różnych sposobów, na przykład tworząc dodatkowy kod T-SQL wykonywany podczas modyfikacji danych lub definiując wyzwalacz uruchamiany automatycznie po wprowadzeniu zmian albo tworząc kod aplikacji na bazie platform Java, Excel, .NET itp. Aby osiągnąć ten cel, oprócz bazy danych trzeba przemyśleć i zaprojektować inne elementy aplikacji, a w samej aplikacji i bazie danych zastosować specjalne techniki kodowania, zapewniające ścisłą kontrolę danych. Może Cię to dziwić: po co przydawać sobie dodatkowej trudnej pracy? Jeżeli spójność danych jest kontrolowana za pomocą kodu, dopuszczalne jest ładowanie do bazy poprawnych danych w dowolnej kolejności. Na przykład jeżeli do bazy załadowałeś dane o transakcjach, zanim umieściłeś w niej dane o klientach, albo chcesz mieć pełną kontrolę nad archiwizacją danych, wtedy mechanizmy kontroli spójności danych nie są pożądane.

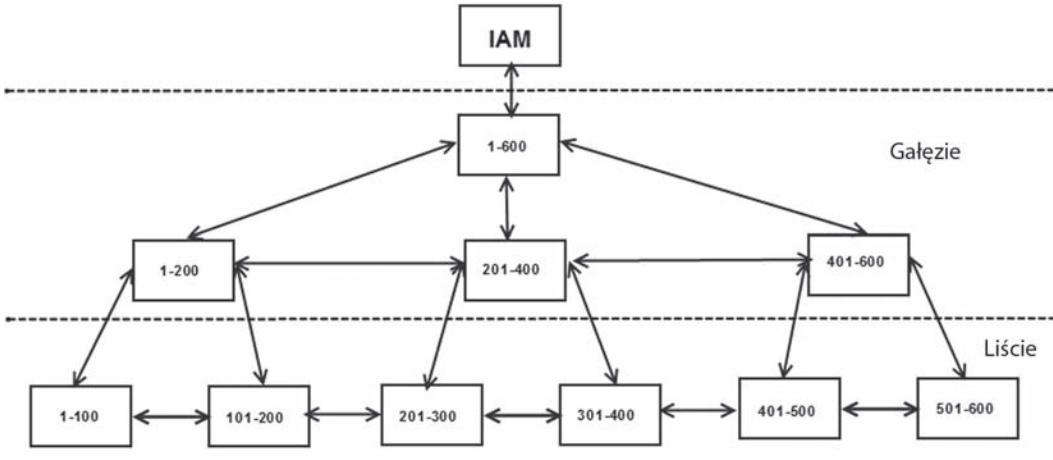
Wybór metody nie jest prosty, jednak uważam, że w większości przypadków najlepiej jest zastosować fizyczne relacje. Jednak jako programista musisz być przygotowany na to, że w miarę nabywania doświadczenia będziesz spotykał się z aplikacjami wykorzystującymi relacje logiczne, dlatego musisz wiedzieć, że takie istnieją. Jeżeli będziesz miał możliwość wyboru, jestem całkowicie przekonany, że relacje powinny być tworzone w bazie danych, a nie w kodzie aplikacji. Jednak wymuszanie warunków relacji na serwerze jest to proces wykonywany podczas każdorazowego wpisywania danych, zajmujący jakiś czas. Ten czas jest krótki, ale określony. W przypadku często wykorzystywanych, szybkich aplikacji przetwarzających duże ilości danych może okazać się, że lepszym rozwiązaniem jest implementacja logicznych relacji za pomocą kodu.

Partycjonowanie danych

Ostatnim tematem dotyczącym tabel, na który chciałbym zwrócić uwagę, jest technika zwana **partycjonowaniem danych**. Może się zdarzyć, że będziesz zajmował się bardzo dużymi ilościami danych — milionami albo nawet setkami milionów wierszy, zajmującymi gigabajty miejsca na dysku. W takiej sytuacji z odczytaniem jednego konkretnego wiersza, nawet przy użyciu indeksu, jest związany koszt czasu potrzebnego na jego odszukanie i zwrócenie. Jedną z metod zwiększenia wydajności bazy jest podzielenie danych na logiczne „kawałki". Na tym, bardzo ogólnie, polega partycjonowanie danych. Weźmy dla przykładu tabelę *SzczegółyTransakcji.Transakcje*. W miarę upływu czasu, gdy Twoja firma będzie się rozwijać, tabela będzie zapełniała się ogromną ilością danych. Niektóre rekordy transakcji mogą się w niej znajdować od miesięcy, a nawet lat, ale mogą być odczytywane rzadko, a nawet wcale. Takie dane możesz oczywiście umieścić w osobnej tabeli, ale mogą istnieć powody, dla których rekordy muszą być zapisane w bieżącej tabeli. Może się nawet zdarzyć, że firma będzie się rozwijać tak prężnie, że dziennie będą tworzone miliony wierszy. Niezależnie od okoliczności, pojawi się konieczność poprawy wydajności niektórych zapytań.

Aby opis był bardziej obrazowy, wyobraź sobie, że tabela *SzczegółyTransakcji.Transakcje* zawiera 10 milionów wierszy, a dziennie dodawanych jest do niej 10 000 nowych rekordów. W tabeli trzeba przechowywać dane tylko przez trzy lata, potem można je usunąć. Większość zapytań operuje na danych z bieżącego miesiąca finansowego, zaczynającego się od pierwszego dnia miesiąca daty w kolumnie *DataOtwarcia*.

Nie wnikając w szczegóły działania bazy danych, możemy stwierdzić, że gdy nie jest zastosowane partycjonowanie danych, serwer SQL Server analizuje indeks tabeli (indeksy są opisane w rozdziale 6.), dzieli dane i tworzy drzewiastą strukturę, zwaną **B-drzewem**. Struktura ta jest wykorzystywana przez serwer do znajdowania wiersza lub wielu wierszy żądanych w zapytaniu. Na rysunku 5.24 przedstawiona jest podstawowa struktura B-drzewa, przypominająca odwrócone drzewo, w którym pień (strona IAM, ang. *index allocation map* — mapa alokacyjna indeksu) dzieli się na gałęzie (strony danych) i liście na ostatnim poziomie. Serwer buduje strukturę B-drzewa tak, aby jego pień (u góry) znajdował się pośrodku struktury (stąd jego nazwa). Podczas wykonywania zapytania serwer porusza się po strukturze, zaczynając od jej wierzchołka, i przechodzi przez każdy poziom na sam dół do liści. Jak widać, struktura przedstawiona na rysunku 5.24 obejmuje niewielką liczbę wierszy danych. Zwróć uwagę, że dane są uporządkowane według kolumny *IDKlienta*, ponieważ takie uporządkowanie danych wydaje się najlepsze.

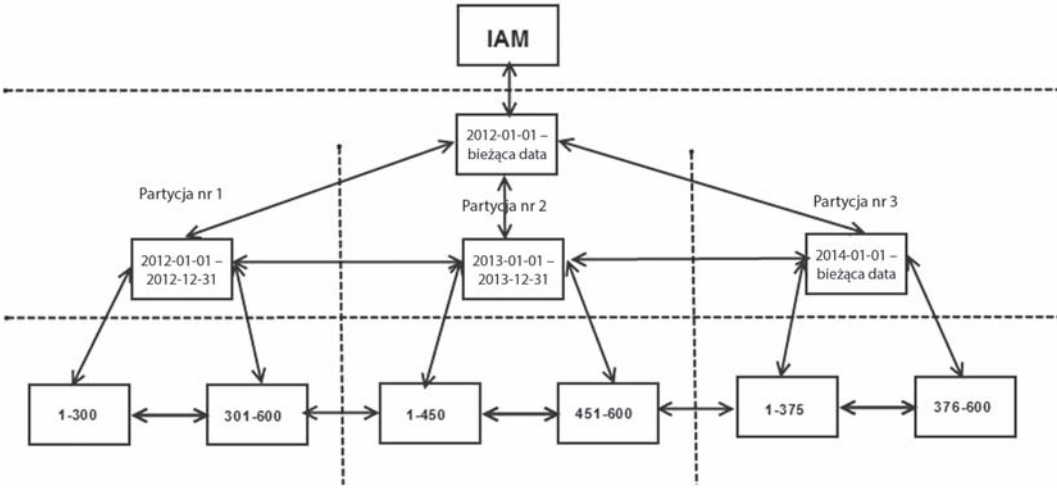


Rysunek 5.24. *Mała struktura B-drzewa*

Jeżeli liczba wierszy jest rzędu milionów, serwer nie tworzy bardzo wysokiej struktury, w której dane są równomiernie dzielone na kolejnych poziomach. Liczba poziomów jest ograniczona i w przypadku dużej ilości danych tworzone jest szerokie, a nie wysokie B-drzewo. Serwer poszerza drzewo, tworząc nowe gałęzie. Możesz sobie wyobrazić, ile pracy musi wykonać serwer w sytuacji, gdy tabela zawiera 10 milionów wierszy, których codziennie przybywa. Dane cały czas są uporządkowane według kolumny *IDKlienta*.

Dzięki podzieleniu (czyli partycjonowaniu) danych według kolumny *DataOtwarcia* na poszczególne miesiące serwer wie, że podczas wyszukiwania transakcji — podczas ogólnego raportowania lub wykonywania zapytania — niektóre gałęzie można całkowicie pominąć i skupić się na przeszukiwaniu tylko tych danych, które zawierają zadaną datę. Rysunekk 5.25 przedstawia przykładową strukturę B-drzewa dla tabeli *SzczegółyTransakcji.Transakcje* podzielonej na partycje obejmujące jeden rok, a każda z nich posiada indeks oparty na kolumnie *IDKlienta*. Jeżeli serwer SQL Server otrzyma zapytanie żądające danych z 2014 roku, będzie wiedział, że może pominąć pierwsze dwie partycje, i przeszuka tylko dane w trzeciej.

■ **Uwaga** Przedstawiony rysunek nie odzwierciedla dokładnie rzeczywistej struktury B-drzewa, ponieważ serwer SQL Server dodatkowo oznacza dane w celu przyspieszenia ich przeszukiwania. Ponadto, układ drzewa na rysunku jest inny niż w rzeczywistości, jednak pokazuje on, w jaki sposób przeszukiwana jest niewielka ilość danych.



Rysunek 5.25. *B-drzewo podzielone na partycje*

Na koniec chciałbym jeszcze dodać, że zgodnie z wymaganiami usunięcie danych za jeden miesiąc, starszych niż trzy lata, zajęłoby serwerowi dużo czasu, ponieważ potencjalnie musiałby usunąć 300 000 wierszy. Dzięki partycjonowaniu można usunąć całą partycję, co trwa o wiele krócej, ponieważ usuwane są całe strony danych, a nie poszczególne wiersze.

Chciałbym wyraźnie zaznaczyć, że przedstawiony opis jest bardzo ogólny, prosty i zawierający przykład niewielkiej ilości danych, specjalnie bez praktycznego ćwiczenia. Partycjonowanie oferuje znacznie więcej możliwości. Jest to bardzo zaawansowane zagadnienie, wymagające odpowiedniego zaprojektowania bazy, rozważenia ryzyka, na jakie mogą być narażone dane, metod ich sprawdzania i kontroli. Jednak jeżeli będziesz miał do czynienia z bardzo dużą ilością danych, warto wiedzieć, że taka technologia jest dostępna.

Podsumowanie

Wiesz już, jak tworzyć tabele. W tym rozdziale zostały opisane różne metody, jest jednak jedna rzecz, o której musisz pamiętać podczas tworzenia lub modyfikowania tabel: jeżeli korzystasz z aplikacji SQL Server Management Studio, zawsze zapisuj tabelę, klikając przycisk *Save*. Gdy zamkniesz okno do projektowania tabel, jednocześnie zostanie podjęta próba zapisania zmian. Jeżeli w definicji tabeli popełnisz błąd, pojawi się odpowiedni komunikat, a wszystkie zmiany zostaną utracone. Będziesz musiał wtedy ponownie otworzyć okno i wprowadzić zmiany.

Naucz się posługiwać aplikacją SQL Server Management Studio i edytorem zapytań *Query Editor*. Jak się przekonasz, edytor zapytań daje większy komfort pracy, a ponadto w miarę postępu prac kod można zapisywać w pliku. Takie same operacje możesz wykonać za pomocą SQL Server Management Studio, jednak zmiany są zapisywane w pliku tekstowym w postaci poleceń SQL, które i tak trzeba wykonywać za pomocą edytora.

Skorowidz

C

D

E

F

G

I

J

K

L

M

N

O

P

T

U

V

W

Z

PEWNIE WKROCZ W ŚWIAT BAZ DANYCH!

SQL Server

Wstęp dla programistów. Wydanie IV

SQL Server to serwer baz danych firmy Microsoft. To oprogramowanie, rozwijane od ponad ćwierćwiecza, cieszy się ogromną popularnością i uznaniem na całym świecie. Charakteryzuje się wysoką wydajnością i bezpieczeństwem, a ponadto daje użytkownikom ogromne możliwości. Przy tych wszystkich zaletach jest również łatwe w instalacji i nietrudno nim administrować. Brzmi zachęcająco? Chcesz wykorzystać potencjał tego narzędzia do własnych zastosowań? Trafiłeś na świetną książkę!

Wprowadzi Cię ona w tajniki pracy z SQL Server 2014. W trakcie lektury kolejnych rozdziałów nauczysz się instalować i konfigurować niezbędne komponenty oraz korzystać z SQL Server Management Studio. Po przygotowaniu środowiska pracy przejdziesz do tworzenia swojej pierwszej bazy danych. Tabele, indeksy, relacje, więzy integralności to pojęcia, które już za chwilę nie będą Ci obce. Poznasz dostępne typy danych oraz wykorzystasz je we właściwy sposób, a co najważniejsze, opanujesz składnię i możliwości języka SQL. Twoją uwagę powinny przykuć rozdziały poświęcone widokom oraz procedurom składowanym — dzięki nim będziesz mógł w pełni wykorzystać SQL Server. Śmiało, zacznij przygodę z bazami danych od tej książki!

Dzięki tej książce:

- zainstalujesz i skonfigurujesz SQL Server 2014
- poznasz zasady projektowania baz danych
- przygotujesz swoją pierwszą bazę danych
- poznasz dostępne typy danych oraz kluczowe pojęcia
- bezboleśnie wejdziesz w świat baz danych

Robin Dewson — swoją przygodę z programowaniem rozpoczął w 1980 roku na komputerze ZX80. Związany z SQL Server od wersji 6.5 (rok wydania: 1996). Pracuje jako konsultant, tworzy również strony oparte na języku Ruby.

Helion

36712 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

Apress

ISBN 978-83-283-1267-8

9 788328 312678

cena: 89,00 zł