Microsoft

Itzik Ben-Gan

Microsoft® SQL Server® 2012 Podstawy języka T-SQL

przełożył Leszek Biolik

APN Promise, Warszawa 2012

Microsoft® SQL Server® 2012: Podstawy języka T-SQL

Authorized Polish translation of English edition of **Microsoft® SQL Server® 2012 T-SQL Fundamentals** ISBN: 978-0-735-65814-1

APN PROMISE SA, biuro: ul. Kryniczna 2, 03-934 Warszawa
tel. +48 22 35 51 600, fax +48 22 35 51 699
e-mail: mspress@promise.pl

Książka ta przedstawia poglądy i opinie autora. Przykłady firm, produktów, osób i wydarzeń opisane w niniejszej książce są fikcyjne i nie odnoszą się do żadnych konkretnych firm, produktów, osób i wydarzeń, chyba że zostanie jednoznacznie stwierdzone, że jest inaczej. Ewentualne podobieństwo do jakiejkolwiek rzeczywistej firmy, organizacji, produktu, nazwy domeny, adresu poczty elektronicznej, logo, osoby, miejsca lub zdarzenia jest przypadkowe i niezamierzone.

ISBN: 978-83-7541-101-0

Przekład: Leszek Biolik
Redakcja: Marek Włodarz
Korekta: Ewa Swędrowska
Skład i łamanie: MAWart Marek Włodarz

Spis treści

Przedmowa

Jestem bardzo szczęśliwy, że Itzik Beg-Gan znalazł czas i siły, by napisać książkę dotyczącą podstaw języka T-SQL. Od wielu lat Itzik, korzystając ze swoich bogatych doświadczeń w nauczaniu i konsultacjach związanych z produktem Microsoft SQL, pisze książki na temat zaawansowanego programowania, pozostawiając istotną lukę nie tylko dla nowicjuszy czy mniej doświadczonych użytkowników, ale także dla wielu ekspertów korzystających z produktu SQL Server i pełniących role, w których programowanie T-SQL nie stanowi najwyższego priorytetu.

W odniesieniu do języka T-SQL Itzik jest na świecie jedną z najbardziej kompetentnych osób. W rzeczywistości my, czyli członkowie zespołu projektowego SQL Server, zwracamy się do niego po porady dotyczące większości nowych rozszerzeń języka, jakie planujemy wprowadzić. Jego oceny i przeprowadzane z nim konsultacje stanowią ważną część naszego procesu projektowania produktu SQL Server.

Dla osoby, będącej ekspertem w danym temacie, nigdy nie jest prostym zadaniem napisanie książki, która jest wprowadzeniem do tej tematyki; w tym względzie jednak ogromnym atutem Itzika są wieloletnie doświadczenia w nauczaniu podstawowych, jak i zaawansowanych klas programowania. Doświadczenia te stanowią wielką wartość przy rozróżnianiu podstaw języka T-SQL od bardziej zaawansowanych tematów. Jednak w tej książce autor nie stosuje prostego unikania zaawansowanej tematyki; nie obawia się podejmowania spójnych opisów złożonych tematów, takich jak teoria zbiorów, logika predykatów czy model relacyjny, wprowadzając je w zrozumiały sposób i na tyle szczegółowo, by czytelnicy zdali sobie sprawę z ich istotnej roli dla języka SQL. W rezultacie powstała książka, dzięki której czytelnicy nie tylko dowiedzą się, co i jak działa w T-SQL, ale także będą wiedzieć dlaczego.

W książkach i podręcznikach programowania nie ma lepszego sposobu przedstawiania tematyki, niż omawianie jej na podstawie dobrych przykładów. W niniejszej książce zawarto wiele przykładów, a wszystkie przykłady można pobrać z witryny autora pod adresem: *http://tsql.solidq.com*. T-SQL to dialekt oficjalnych standardów ISO i ANSI dotyczących języka SQL, ale zawierający szereg rozszerzeń, które mogą usprawnić wyrazistość i zwięzłość naszego kodu T-SQL. W wielu prezentowanych przykładach pokazano krok po kroku rozwiązania przedstawione w języku T-SQL i ich odpowiedniki w ANSI SQL. Jest to ogromna zaleta dla czytelników, którzy znają język SQL w wersji standardu ANSI, a którzy nie znają T-SQL, a także dla programistów, którzy muszą pisać kod SQL, który będzie można w łatwy sposób instalować na wielu różnych platformach bazodanowych.

Silne powiązania autora z zespołem produktu SQL Server widoczne są wyraźnie w opisie specyficznych rozwiązań w SQL Server określanych mianem ABC (Appliance, Box, Cloud), które przedstawiono w rozdziale 1 „Podstawy zapytań i programowania T-SQL". Do tej pory używanie pojęcia „ABC" widziałem jedynie w zespole Microsoft SQL Server, jestem jednak pewny, że tylko kwestią czasu jest jego szerokie upowszechnienie. Itzik opracował i testował przykłady zamieszczone w książce zarówno w odniesieniu do rozwiązań „B" (box), jak i „C" (cloud) produktu SQL Server, a Dodatek wskazuje, w którym miejscu możemy rozpocząć pracę z wersją SQL Server w chmurze, czyli produktem nazywanym Windows Azure SQL Database. Z tego względu możemy posługiwać się niniejszą książką jako punktem początkowym naszych doświadczeń z chmurą. W witrynie Azure znaleźć można informacje, jak uzyskać darmową subskrypcję usług Azure, by wykonywać przykłady przytoczone w książce.

Rozszerzenie o usługi chmury dla SQL Server to wyjątkowo istotna kwestia, której nie powinniśmy pomijać. Osobiście uważam ją za tak ważną, że zrobię coś, czego nigdy dotąd nie zrobiłem pisząc słowo wstępne – zareklamuję inną książkę (przepraszam Itzik, ale muszę to zrobić!). Moje zainteresowania chmurą i przekonanie o jej zaletach gwałtownie wzrosły po przeczytaniu książki *The Big Switch* napisanej przez Nicholasa G. Carra (W.W. Norton and Company, 2009) i chcę właśnie podzielić się tym doświadczeniem. Jest to wspaniała książka, która porównuje postęp, jaki dokonuje się dzięki obliczeniom w chmurze, do procesu elektryfikacji wprowadzanego na początku dwudziestego wieku. Moja wiara w przyszłość chmury została jeszcze bardziej utrwalona po obejrzeniu prezentacji Jamesa Hamilton „Cloud Computing Economies of Scale" na konferencji MIX10 (nagranie dostępne jest pod adresem *http://channel9.msdn.com/events/MIX/ MIX10/EX01*).

Itzik wskazuje na jeszcze jedną zmianę związaną z technologią chmury, z której powinniśmy zdawać sobie sprawę. Przyzwyczailiśmy się do kilkuletniej przerwy pomiędzy kolejnymi wersjami SQL Server, ale dzięki chmurze ten wzorzec zmieni się istotnie; powinniśmy być przygotowani na kilka mniejszych wydań produktu w chmurze (nazywanych Aktualizacjami usługi), które każdego roku będą instalowane w centrach danych firmy Microsoft na całym świecie. Z tego względu Itzik rozsądnie postępuje, dokumentując rozbieżności T-SQL pomiędzy SQL Server a Windows Azure SQL Database w swojej witrynie *http://tsql.solidq.com*, a nie w książce – dzięki temu znacznie łatwiej jest prezentować aktualne informacje.

Książkę czyta się z wielkim zainteresowaniem, a co więcej, pozwala się ona cieszyć z odkrywania prezentowanych w niej nowych wskazówek dotyczących T-SQL.

Lubor Kollar, SQL Server Development Team, Microsoft

Wprowadzenie

Niniejsza książka pełni rolę przewodnika osób podejmujących pierwsze kroki w języku T-SQL (nazywanym także Transact-SQL), który jest opracowanym w firmie Microsoft dialektem standardów ISO i ANSI języka SQL. Dzięki książce poznamy teorię konstruowania zapytań i programowania w języku T-SQL oraz sposoby projektowania kodu T-SQL w celu uzyskiwania i modyfikowania danych. Ponadto w książce zamieszczono ogólny opis programowalnych obiektów.

Pomimo że książka pomyślana jest dla Czytelników początkujących, nie jest jedynie zbiorem procedur, według których mają postępować – książka wykracza poza elementy składni T-SQL i wyjaśnia logikę działającą w tle języka i jego elementów.

Od czasu do czasu w książce pojawiają zagadnienia, które dla osób poznających język T-SQL mogą być uważane za tematykę zaawansowaną – z tego też względu zapoznawanie się z tym fragmentami jest opcjonalne. Jeśli dość pewnie czujemy się w omówionym do tej pory materiale, możemy przejść do tematów bardziej zaawansowanych; w przeciwnym razie spokojnie możemy opuścić te fragmenty i powrócić do nich, gdy już nabierzemy większego doświadczenia. Fragmenty uważane za bardziej zaawansowane są w tekście zaznaczone jako opcjonalne.

Wiele aspektów SQL jest unikatowych dla tego języka i różnią się znacznie od innych języków programowania. Niniejsza książka ułatwia przyswojenie sobie właściwego sposobu myślenia i pozwala dobrze poznać elementy języka oraz postępować zgodnie z najlepszymi zaleceniami praktycznymi programowania w języku SQL.

Książka nie są związana z konkretną wersją – obejmuje jednak elementy języka, które zostały wprowadzone w ostatnich wersjach SQL Server, wliczając w to SQL Server 2012. Podczas omawiania ostatnio wprowadzonych elementów języka specyfikowana jest wersja produktu, w której dany element został dodany.

Produkt SQL Server, oprócz tego, że jest rozwiązaniem dostępnym w siedzibie, jest także dostępny jako usługa w chmurze o nazwie Windows Azure SQL Database (poprzednio SQL Azure). Przykłady kodu przytaczane w książce były testowane zarówno w instalacjach SQL Server w siedzibie, jak i w odniesieniu do SQL Database. W powiązanej z książką witryną sieci Web (*http://tsql.solidq.com*) udostępnione są informacje dotyczące problemów zgodności pomiędzy tymi rozwiązaniami – na przykład funkcje dostępne w SQL Server 2012, które nie są jeszcze dostępne w SQL Database.

W celu uzupełnienia procesu nauczania w książce zamieszczono ćwiczenia, które pozwalają poznane informacje utrwalić w praktyce. Od czasu do czasu w książce pojawiają się ćwiczenia opcjonalne, które są bardziej zaawansowane. Ćwiczenia

te przeznaczone są dla Czytelników, którzy dobrze poznali omawiany materiał i chcą sami sprawdzić swoje umiejętności, rozwiązując trudniejsze problemy. Ćwiczenia opcjonalne dla Czytelników zaawansowanych są odpowiednio oznaczone.

Dla kogo przeznaczona jest ta książka

Niniejsza książka skierowana jest dla programistów korzystających z języka T-SQL, administratorów baz danych, osób zajmujących się rozwiązaniami BI, autorów raportów, analityków, architektów baz danych i zaawansowanych użytkowników, którzy dopiero rozpoczynają pracę z SQL Server i muszą tworzyć kwerendy albo kod przy użyciu języka Transact-SQL.

Założenia

Największe korzyści książka ta przyniesie osobom, które mają już doświadczenie w pracy z systemami Windows i aplikacjami opartymi na tych systemach. Ponadto osoby te powinny znać podstawowe pojęcia dotyczące systemów zarządzania relacyjnymi bazami danych. Przydatne będzie też podstawowe doświadczenie w tworzeniu oprogramowania.

Kto nie powinien czytać tej książki

Nie każda książka nadaje się dla każdego czytelnika. W książce omówiono podstawy języka i głównie skierowana jest ona do osób w praktyce korzystających z języka T-SQL, które nie mają w tym wielkiego doświadczenia. Tym niemniej, wielu czytelników poprzedniego wydania tej książki uważa, że pomimo doświadczeń zdobytych w kolejnych latach pracy książka ta nadal jest przydatna i uzupełnia brakującą wiedzę.

Organizacja książki

Książka rozpoczyna się od przedstawienia teoretycznych podstaw konstruowania zapytań i programowania w języku T-SQL (rozdział 1), co stanowi fundament dla pozostałej części książki, a także dla procesów tworzenia tabel i definiowania integralności danych. W rozdziałach 2 do 8 poruszane są różnorodne aspekty uzyskiwania i modyfikowania danych, w rozdziale 9 omówione są kwestie współbieżności i transakcji, a na koniec rozdział 10 stanowi przegląd obiektów programowalnych. Poniżej przedstawiono listę rozdziałów wraz z krótkim ich opisem:

- Rozdział 1 „Podstawy zapytań i programowania T-SQL” – teoretyczne podstawy SQL, teoria zbiorów i logika predykatów; analizy modelu relacyjnego; opisy architektury SQL Server; wyjaśnienie sposobów tworzenia tabel i definiowania integralności danych.

- Rozdział 2 „Kwerendy dotyczące pojedynczej tabeli” – różnorodne aspekty konstruowania kwerend dotyczących pojedynczej tabeli przy użyciu polecenia *SELECT*.
- Rozdział 3 „Złączenia” – opis zapytań dotyczących wielu tabel przy użyciu złączeń (join), a w tym złączeń typu iloczyn kartezjański, złączenia wewnętrzne i zewnętrzne.
- Rozdział 4 „Podkwerendy” – omówienie kwerend używanych wewnątrz innych kwerend (mechanizm nazywany czasem podkwerendą).
- Rozdział 5 „Wyrażenia tablicowe” – omówienie tabel pochodnych, wyrażeń CTE (Common Table Expression), widoków, wbudowanych funkcji zwracających tabele i operatora *APPLY*.
- Rozdział 6 „Operatory zbiorowe” – omówienie operatorów *UNION*, *INTERSECT* i *EXCEPT*.
- Rozdział 7 „Zaawansowane kwestie tworzenia zapytań” – omówienie funkcji okien, operatorów *PIVOT* i *UNPIVOT* oraz praca z operatorami *GROUPING SETS*.
- Rozdział 8 „Modyfikowanie danych” – wstawianie, aktualizowanie, usuwanie i scalanie danych.
- Rozdział 9 „Transakcje i współbieżność” – omówienie kwestii współdziałania połączeń użytkowników, którzy jednocześnie korzystają z tych danych; rozdział opisuje pojęcia, takie jak transakcje, blokady, poziomy izolacji czy zakleszczenia.
- Rozdział 10 „Obiekty programowalne” – omówienie możliwości programowania przy użyciu T-SQL w SQL Server.

W książce zamieszczono także dodatek „Rozpoczynamy”, który ułatwia skonfigurowanie środowiska, pobranie kodów źródłowych książki, zainstalowanie przykładowej bazy danych *TSQL2012*, rozpoczęcie pisania kodu w odniesieniu do produktu SQL Server oraz poznanie sposobów uzyskania pomocy dzięki dokumentacji SQL Server Books Online.

Wymagania systemowe

Dodatek „Rozpoczynamy” zawiera informacje, których wersji produktu SQL Server 2012 można użyć do pracy z przykładowym kodem zamieszczonym w tej książce. Poszczególne wersje SQL Server mogą mieć różne wymagania systemowe i programowe, a te wymagania są dokładnie opisane w dokumentacji SQL Server Books Online w sekcji „Hardware and Software Requirements for Installing SQL Server 2012”. W Dodatku wyjaśniono również, jak korzystać z dokumentacji SQL Server Books Online.

Jeśli jesteśmy połączeni z usługą SQL Database, a sprzęt i oprogramowanie jest utrzymywane przez firmę Microsoft, wymagania te nie są istotne.

Przykłady kodu

Niniejsza książka powiązana jest z witryną sieci Web, która udostępnia wszystkie przykłady kodu używane książce, poprawki, a także dodatkowe zasoby.

http://tsql.solidq.com

W dodatku „Rozpoczynamy" znaleźć można informacje szczegółowe na temat kodów źródłowych.

Poprawki i pomoc techniczna

Dołożyliśmy wszelkich starań, by zapewnić największą dokładność informacji prezentowanych w książce i materiałach pomocniczych. Informacje o zmianach i korektach zauważonych po opublikowaniu książki zamieszczane będą w witrynie Microsoft Press (oreilly.com):

http://go.microsoft.com/FWLink/?Linkid=248718

Informacje o niezauważonych jeszcze błędach prosimy przekazywać za pośrednictwem tej samej strony.

Dodatkowa pomoc możliwa jest za pośrednictwem adresu e-mail Microsoft Press Book Support: *mspinput@microsoft.com*.

Prosimy pamiętać, że pod wymienionymi powyżej adresami nie jest oferowana pomoc techniczna dla oprogramowania firmy Microsoft.

Oczekujemy na Wasze uwagi

W firmie Microsoft Press najwyższym priorytetem jest zadowolenie czytelników, a wszelkie komentarze są mile widziane i stanowią cenną wartość. Swoje opinie o książce można przesłać na adres:

http://www.microsoft.com/learning/booksurvey

W krótkim czasie zapoznajemy się ze wszystkim komentarzami i uwagami. Z góry dziękujemy za wszelkie opinie!

Pozostańmy w kontakcie

Warto rozmawiać! Jesteśmy na Twitterze: *http://twitter.com/MicrosoftPress*.

Podziękowania

Wiele osób przyczyniło się do powstania tej książki, czy to bezpośrednio, czy pośrednio i wszystkim im należą się serdeczne podziękowania i uznanie.

Dla Lilach, za zrozumienie i wsparcie wszystkich moich poczynań oraz za wyrozumiałość dla niekończących się godzin spędzonych nad SQL.

Dla moich rodziców, Mila i Gabi, oraz rodzeństwa Mickey i Ina, za stałe wsparcie i akceptowanie nieobecności, co teraz jest trudniejsze niż kiedykolwiek. Mamo, wszyscy wierzymy, że będzie dobrze, dzięki Twojej sile i determinacji. Tato, dzięki, że jesteś tak pomocny.

Dla członków zespołu projektowego Microsoft SQL Server; Lubor Kollar, Tobias Ternstrom, Umachandar Jayachandran (UC) i z pewnością wielu innych. Dziękuję za wielki wysiłek i czas spędzony na spotkaniach ze mną i poświęcony na udzielanie mailowych odpowiedzi na moje pytania i wątpliwości. Sądzę, że w produktach SQL Server 2012 i SQL Database widać poczynione wielkie inwestycje w T-SQL i mam nadzieję, że ten kierunek zostanie zachowany.

Dla zespołu redaktorów w firmach O'Reilly Media i Microsoft Press; dziękuję Kenowi Jones za poświęcone godziny i zainicjowanie projektu. Dziękuję Russellowi Jones za uruchomienie i przeprowadzenie projektu po stronie O'Reilly. Dziękuję także Kristen Borg, Kathy Krause i wszystkim innym, którzy pracowali nad tą książką.

Dla Herberta Albert i Gianluca Hotz, redaktorów technicznych książki, wykonaliście doskonałą pracę, dzięki której książka jest lepsza i bardziej precyzyjna.

Dla SolidQ, mojej firmy w ostatniej dekadzie: to wielka satysfakcja pracować w tak wspaniale prowadzonej firmie. Pracownicy firmy to nie tylko koledzy – to partnerzy, przyjaciele i rodzina. Fernando G. Guerrero, Douglas McDowell, Herbert Albert, Dejan Sarka, Gianluca Hotz, Jeanne Reeves, Glenn McCoin, Fritz Lechnitz, Eric Van Soldt, Joelle Budd, Jan Taylor, Marilyn Templeton, Berry Walker, Alberto Martin, Lorena Jimenez, Ron Talmage, Andy Kelly, Rushabh Mehta, Eladio Rincón, Erik Veerman, Jay Hackney, Richard Waymire, Carl Rabeler, Chris Randall, Johan Åhlén, Raoul Illyés, Peter Larsson, Peter Myers, Paul Turley – dziękuję Wam i wielu innym osobom.

Dla zespołu redakcyjnego produktu SQL Server Pro, czyli Megan Keller, Lavon Peters, Michele Crockett, Mike Otey i z pewnością jeszcze wiele innych osób; pisałem dla tego magazynu od ponad dekady i jestem wdzięczny, że mogłem podzielić się moją wiedzą z czytelnikami magazynu.

Dla osób z grupy MVP produktu SQL Server – Alejandro Mesa, Erland Sommarskog, Aaron Bertrand, Tibor Karaszi, Paul White i wielu innych oraz dla prowadzącego

program MVP, Simona Tien; jest to wspaniały program i jestem wdzięczny, że mogłem w nim uczestniczyć. Niespotykany jest poziom umiejętności w tej grupie i zawsze cieszyłem się na nasze spotkania, zarówno, by podzielić się pomysłami, jak i po prostu pogadać przy piwie. Jestem przekonany, że w dużej mierze dodanie przez Microsoft funkcjonalności T-SQL do produktu SQL Server zostało zainspirowane przez członków MVP i całą społeczność produktu SQL Server. To wspaniale przekonać się, że ta synergia przekształca się w tak znaczące i ważne rezultaty.

Dla Q2, Q3 i Q4, thanQ.

Na koniec, podziękowania kieruję do moich studentów: nauczanie SQL jest moją siłą napędową, moją pasją. Dziękuję, że mogę spełniać moje powołanie i dziękuję za wszystkie interesujące pytania, które pozwalają pogłębiać wiedzę.

Itzik Ben-Gan

ROZDZIAŁ 1

Podstawy zapytań i programowania T-SQL

Wyruszamy w podróż do krainy innej niż wszystkie – miejsca, które rządzi się własnym zbiorem praw. Jeśli książka ta jest pierwszym krokiem poznawania języka T-SQL (Transact-SQL), powinniście się czuć tak jak Alicja przed rozpoczęciem jej przygód w Krainie Czarów. Dla mnie podróż ta nie ma końca, a po drodze stale odkrywam coś nowego. Zazdroszczę Wam – niektóre z najbardziej ekscytujących odkryć wciąż są przed Wami!

Z językiem T-SQL związany jestem od wielu lat: nauczanie, seminaria, pisanie i konsultacje związane z tą tematyką. Dla mnie T-SQL to coś więcej niż tylko język programowania – to sposób myślenia. Do tej pory uczyłem i pisałem często na temat zaawansowanych kwestii, a nie opisywałem podstaw języka. Nie dlatego, że podstawy T-SQL są proste czy łatwe – wręcz przeciwnie: widoczna prostota języka jest myląca. Mógłbym opisywać elementy składni języka w sposób bardzo powierzchowny i na tej podstawie można w kilka minut rozpocząć pisanie kwerend. W praktyce jednak podejście takie utrudnia zrozumienie istoty języka i wydłuża proces jego poznawania.

Rola przewodnika osób podejmujących pierwsze kroki to duża odpowiedzialność. Chciałbym mieć pewność, że poświęciłem wystarczająco dużo czasu i wysiłku, analizując i poznając język, zanim zabrałem się za opisywanie jego podstaw. Język T-SQL nie jest prosty; właściwe opanowanie podstaw to znacznie więcej, niż zrozumienie elementów składni i kodowania kwerend, które zwracają odpowiednie wyniki. W zasadzie trzeba wyrzucić z pamięci to, co się wie na temat innych języków programowania i zacząć myśleć w kategoriach języka T-SQL.

Teoretyczne podstawy

SQL to akronim nazwy *Structured Query Language* (strukturalny język zapytań). Jest to standardowy język, opracowany do tworzenia zapytań i zarządzania danymi w systemach zarządzania relacyjnymi bazami danych (RDBMS). RDBMS to system zarządzania bazą danych oparty na modelu relacyjnym (model semantyczny przedstawiania danych), który z kolei bazuje na dwóch działach matematyki: teorii zbiorów i logice predykatów. Wiele innych języków programowania i różnych aspektów obliczeń komputerowych powstało i rozwijało się w dużej mierze w oparciu o intuicję. Inaczej jest w przypadku SQL, w takim stopniu, w jakim SQL bazuje na modelu

relacyjnym, opiera się na trwałym fundamencie – matematyce stosowanej. Tak więc T-SQL wspierany jest przez solidne podstawy. Firma Microsoft udostępnia język T-SQL jako dialekt (lub rozszerzenie) języka SQL w oprogramowaniu zarządzania danymi produktu Microsoft SQL Server, czyli w swoim systemie RDBMS.

Niniejszy podrozdział stanowi krótki opis teoretycznych podstaw języka SQL, teorii zbiorów i logiki predykatów, modelu relacyjnego i cyklu życia danych. Ponieważ książka ta nie jest ani podręcznikiem matematycznym, ani książką o modelowaniu danych i projektowaniu, przedstawione w niej informacje teoretyczne są przekazywane w swobodnej formie i tym samym nie są pełne. Celem jest opisanie kontekstu języka T-SQL i przedstawienie kluczowych punktów, które są integralną częścią procesu prawidłowego zrozumienia opisu działania języka T-SQL w dalszej części książki.

Niezależność języka

Model relacyjny jest niezależny od języka. Możemy więc implementować model relacyjny przy użyciu innego języka niż SQL, na przykład przy użyciu C# w modelu klas. Obecnie typowym rozwiązaniem są systemy RDBMS, które obsługują także inne języki niż dialekt SQL, czego przykładem jest integracja CLR w SQL Server.

Ponadto już na początku trzeba zdawać sobie sprawę, że SQL na różne sposoby odbiega od modelu relacyjnego. Niektórzy nawet twierdzą, że SQL powinien zastąpić nowy język (taki, który jest bardziej zgodny z modelem relacyjnym). Na razie jednak, SQL jest branżowym językiem używanym przez wszystkie wiodące systemy RDBMS.

Zobacz także Informacje szczegółowe na temat niezgodności SQL z modelem relacyjnym, a także na temat metod używania SQL zgodnie z tym modelem, znaleźć można w następującej książce: *SQL and Relational Theory: How to Write Accurate SQL Code*, Second Edition, C. J. Date (O'Reilly Media, 2011).

SQL

SQL to język standardów ANSI i ISO, bazujący na modelu relacyjnym, opracowany do tworzenia zapytań i zarządzania danymi w systemach RDBMS.

Na początku lat siedemdziesiątych, dla swojego systemu RDBMS nazwanego System R, firma IBM opracowała język SEQUEL (Structured English QUEry Language). Nazwa języka została później zmieniona na SQL ze względu na dyskusje dotyczące znaku towarowego. Język SQL stał się najpierw standardem ANSI (w roku 1986), a następnie standardem ISO (w roku 1987). Od roku 1986 instytucje ANSI (American

National Standards Institute) i ISO (International Organization for Standardization) co kilka lat udostępniają nowe wersje standardu SQL. Do tej pory opublikowane zostały następujące standardy: SQL-86 (1986), SQL-89 (1989), SQL-92 (1992), SQL:1999 (1999), SQL:2003 (2003), SQL:2006 (2006), SQL:2008 (2008) i SQL:2011 (2011).

Co interesujące, język SQL przypomina angielski i jest również bardzo logiczny. Inaczej niż w przypadku wielu języków programowania, które stosują imperatywne wzorce programowania, SQL używa wzorców deklaratywnych. Oznacza to, że SQL wymaga określenia, co chcemy uzyskać, a nie w jaki sposób ma to być wykonane, co pozwala systemowi RDBMS tworzyć mechanizmy fizyczne wymagane do przetworzenia żądania użytkownika.

Język SQL obejmuje kilka kategorii instrukcji, wliczając w to DDL (Data Definition Language – język definicji danych), DML (Data Manipulation Language – język manipulacji danymi) i DCL (Data Control Language – język sterowania danymi). Język DDL dotyczy definicji obiektów i obejmuje takie polecenia, jak *CREATE*, *ALTER* czy *DROP*. Język DML umożliwia tworzenie zapytań czy modyfikowanie danych i obejmuje instrukcje, takie jak *SELECT*, *INSERT*, *UPDATE*, *DELETE*, *TRUNCATE* i *MERGE*. Typowym nieporozumieniem jest opinia, że DML zawiera tylko polecenia odnoszące się do modyfikowania danych, ale jak zostało już wspomniane, zawiera także instrukcję *SELECT*. Innym częstym nieporozumieniem jest traktowanie instrukcji *TRUNCATE* jako polecenia języka DDL, kiedy w rzeczywistości jest poleceniem języka DML. Język DCL dotyczy uprawnień i obejmuje takie polecenia, jak *GRANT* czy *REVOKE*. Niniejsza książka skupia się na poleceniach języka DML.

T-SQL opiera się na standardowym języku SQL, ale wprowadza także pewne własne, niestandardowe rozszerzenia. Przy pierwszym opisie elementu języka zazwyczaj zaznaczone zostało, czy jest to standardowy element.

Teoria zbiorów

Teoria zbiorów, której twórcą był matematyk Georg Cantor, to jeden z działów matematyki, na której bazuje model relacyjny. Definicja zbioru według Cantora jest następująca:

> *„Zbiorem" M jest spojenie w całość określonych, rozróżnialnych obiektów m (nazywanych „elementami" zbioru M) naszej myśli czy postrzegania.*
> *– Joseph W. Dauben i Georg Cantor (Princeton University Press, 1990)*

Każde słowo tej definicji ma głębokie i kluczowe znaczenie. Definicje zbioru i członkostwa w zbiorze są aksjomatami, które nie są udowadniane. Każdy element jest częścią wszechświata i należy lub nie należy do zbioru.

Rozpocznijmy od słowa *całość* w definicji Cantora. Zbiór powinien być traktowany jako pojedyncza jednostka. Powinniśmy się skupić na zestawie obiektów, a nie na poszczególnych obiektach, tworzących zestaw. Później, kiedy będziemy pisać

kwerendy T-SQL w odniesieniu do tabeli w bazie danych (jak na przykład tabela pracowników), powinniśmy traktować zbiór pracowników jako całość, a nie jak poszczególnych pracowników. Wydawać się to może oczywiste i bardzo proste, jednak wielu programistów ma trudności w przyswojeniu sobie takiego sposobu myślenia.

Słowo *rozróżnialny* oznacza, że każdy element zbioru musi być niepowtarzalny. Powracając do tabel w bazie danych, możemy wymusić unikatowość wierszy w tabeli, definiując ograniczenia klucza. Bez klucza nie będziemy mogli w sposób jednoznaczny identyfikować wierszy i dlatego tabela nie będzie mogła być kwalifikowana jako zbiór, a będzie raczej *wielozbiorem* czy *pojemnikiem*.

Wyrażenie *naszej myśli czy postrzegania* sugeruje subiektywność definicji zbioru. Weźmy pod uwagę salę lekcyjną: jedna osoba może dostrzegać zbiór osób, a inna zbiór uczniów czy zbiór nauczycieli. Z tego względu w definiowaniu zbiorów mamy pokaźną dawkę swobody. Kiedy projektujemy model danych dla naszej bazy danych, proces ten powinien uważnie uwzględniać subiektywne potrzeby aplikacji, by określić właściwe definicje występujących tam jednostek.

Jeśli chodzi o słowo *obiekt*, definicja zbioru nie jest ograniczona tylko do bytów fizycznych, takich jak samochody czy pracownicy, ale odnosi się także do tworów abstrakcyjnych, takich jak linie czy liczby pierwsze.

Prawdopodobnie to, co pomija definicja Cantora, jest równie ważne jak to, co zawiera. Zwróćmy uwagę, że definicja nie wspomina o jakimkolwiek uporządkowaniu elementów zbioru. Nieistotna jest kolejność elementów zbioru. Typowy zapis wymieniający elementy zbioru korzysta z nawiasów klamrowych: {a, b, c}. Ponieważ kolejność nie ma znaczenia, ten zbiór można przedstawić jako {b, a, c} lub {b, c, a}. Wyprzedzając opis, w zestawie atrybutów (nazywanych w SQL *kolumnami*) tworzących nagłówek relacji (w SQL nazywany *tabelą*) zakłada się, że element jest identyfikowany przez nazwę, a nie przez numer porządkowy.

Podobnie rozważmy zbiór krotek (w SQL nazywanych *wierszami*), który stanowi treść relacji; element jest identyfikowany przez wartości klucza, a nie na podstawie położenia. Wielu programistów długo przyswaja sobie ten sposób myślenia w odniesieniu do zapytań dotyczących tabel – nie istnieje żadna kolejność wierszy. Inaczej mówiąc, kwerenda dotycząca tabeli może w dowolnej kolejności zwrócić wiersze tabeli, chyba że wprost zażądamy określonego posortowania danych, na przykład w celu ich określonego zaprezentowania.

Logika predykatów

Logika predykatów, korzeniami sięgająca starożytnej Grecji, jest innym działem matematyki, na którym opiera się model relacyjny. Dr Edgar F. Codd tworząc model relacyjny dostrzegł możliwość połączenia logiki predykatów zarówno z zarządzaniem danymi, jak i tworzeniem kwerend. Mówiąc niezbyt ściśle, *predykat* jest właściwością czy wyrażeniem, które albo jest spełnione, albo nie – mówiąc inaczej, albo jest prawdą,

albo fałszem. Model relacyjny wykorzystuje predykaty do utrzymywania logicznej integralności danych i definiowania struktury danych. Przykładem predykatu użytego do wymuszenia integralności jest ograniczenie zdefiniowane w tabeli nazwanej *Employees* (pracownicy), które zezwala na przechowywanie w tabeli jedynie tych pracowników, których pensja jest większa od zera. Predykatem jest wyrażenie „pensja większa niż 0" (wyrażenie T-SQL: *pensja > 0*).

Predykaty używane są również podczas filtrowania danych w celu zdefiniowania podzbiorów. Jeśli na przykład zachodzi potrzeba przepytania tabeli *Employees* i zwrócenia jedynie tych wierszy, które dotyczą pracowników działu sprzedaży, w naszym filtrze w kwerendzie możemy użyć predykatu „działem jest (równa się) dział sprzedaży" (wyrażenie T-SQL: *dział = 'sprzedaż'*).

W teorii zbiorów predykatów można używać do definiowania zbiorów. Metoda taka jest przydatna, ponieważ nie zawsze możemy zdefiniować zbiór poprzez wymienienie wszystkich jego elementów (na przykład zbiory nieskończone), a czasami dla zwięzłości wygodniej jest zdefiniować zbiór w oparciu o właściwość. Przykładem zbioru nieskończonego zdefiniowanego za pomocą predykatu może być zbiór wszystkich liczb pierwszych, do zdefiniowania którego użyto następującego wyrażenia: „x jest dodatnią liczbą całkowitą większą niż 1, która podzielna jest tylko przez 1 i samą siebie". Dla dowolnej wyspecyfikowanej wartości predykat jest albo prawdą, albo fałszem. Zbiorem wszystkich liczb pierwszych jest zbiór wszystkich elementów, dla których predykat przyjmuje wartość prawda. Przykładem skończonego zbioru zdefiniowanego za pomocą predykatu może być zbiór *{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}*, definiowany jako zbiór wszystkich elementów, dla których następujący predykat ma wartość prawda: „x jest liczbą całkowitą równą lub większą niż 0 i równa lub mniejszą niż 9".

Model relacyjny

Model relacyjny jest modelem semantycznym zarządzania i modyfikowania danych, opartym na teorii zbiorów i logice predykatów. Jak już wcześniej nadmieniono, twórcą modelu jest Dr. Edgar F. Codd, a model został później wyjaśniony i opracowany przez Chrisa Date, Hugha Darwena i innych. Pierwsza wersja modelu relacyjnego została zaproponowana przez Codda w roku 1969 w postaci raportu badawczego firmy IBM nazwanego „Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks". Poprawiona wersja przedstawiona została przez Codda w roku 1970 w dokumencie nazwanym „A Relational Model of Data for Large Shared Data Banks", opublikowanym w miesięczniku *Communications of the ACM*.

Celem modelu relacyjnego jest udostępnienie spójnej reprezentacji danych przy minimalnej redundancji lub jej braku i bez utraty kompletności oraz zdefiniowanie integralności danych (wymuszanie spójności danych) jako części modelu. System RDBMS powinien implementować model relacyjny i dostarczać środków do realizowania kwerend danych oraz do przechowywania, zarządzania i wymuszania integralności

danych. Model relacyjny opiera się na silnych podstawach matematycznych, a nie na intuicji, co oznacza, że mając pewną instancję modelu danych (na podstawie których zostanie później wygenerowana fizyczna baza danych) możemy dokładnie określić, czy projekt ten jest podatny na wady, nie polegając wyłącznie na intuicji.

Model relacyjny korzysta z takich pojęć, jak tezy, predykaty, relacje, krotki, atrybuty i inne. Dla osób niezajmujących się matematyką pojęcia te mogą wydawać się nieco przytłaczające, ale w kolejnych podrozdziałach przedstawiono najważniejsze aspekty modelu w sposób przystępny i nie-matematyczny oraz wyjaśniono, jak pojęcia te odnoszą się do baz danych.

Tezy, predykaty i relacje

Dośc powszechne przekonanie, że pojęcie *relacyjny* wywodzi się z zależności pomiędzy tabelami, nie jest poprawne. „Relacyjny" w rzeczywistości odnosi się do matematycznego pojęcia *relacji*. W teorii zbiorów relacją jest przedstawienie zbioru. W modelu relacyjnym relacją jest zbiór powiązanych informacji, którego odpowiednikiem w SQL jest tabela – aczkolwiek nie jest to dokładny odpowiednik. Kluczowy punkt modelu relacyjnego to fakt, że pojedyncza relacja powinna reprezentować pojedynczy zbiór (na przykład *Klienci*). Warto zwrócić uwagę na to, że operacje na relacjach (oparte na algebrze relacji) dają w wyniku relacje (na przykład złączenie dwóch relacji).

Uwaga Model relacyjny rozróżnia pojęcie *relacji* i *zmiennej relacji,* ale dla uproszczenia nie będę ich rozróżniać – w obu przypadkach będę używać pojęcia *relacji*. Ponadto relacja zbudowana jest z nagłówka i treści. Nagłówek składa się ze zbioru atrybutów (nazywanych w SQL *kolumnami*), gdzie każdy element jest identyfikowany przez nazwę atrybutu i nazwę typu. Treść składa się ze zbioru krotek (w SQL nazywanych *wierszami*), gdzie każdy element jest identyfikowany przez klucz. Dla uproszczenia będę odnosił się do tabeli jak do zbioru wierszy.

Podczas projektowania modelu danych dla bazy danych przedstawiamy wszystkie dane za pomocą relacji (tabel). Rozpoczynamy od zidentyfikowania tez (ang. *proposition*), które powinny być reprezentowane w bazie danych. Teza jest twierdzeniem, które może być prawdziwe lub fałszywe. Na przykład tezą jest stwierdzenie „Pracownik Itzik Ben-Gan urodził się 12 lutego 1971 roku i pracuje w dziale IT". Jeśli ta teza jest prawdą, sama uwidoczni się w postaci wiersza tabeli *Employees* (pracownicy). Fałszywa teza po prostu się nie uwidoczni. Takie wstępne założenie nazywane jest *założeniem o świecie zamkniętym – CWA (Close World Assumption)*.

Następnym krokiem jest sformalizowanie tez. W tym celu bierzemy rzeczywiste dane (treść relacji) i definiujemy strukturę (nagłówek relacji) – na przykład poprzez tworzenie predykatów. Możemy traktować predykaty jako sparametryzowane tezy. Nagłówek relacji zawiera zbiór atrybutów. Zwróćmy uwagę na użycie pojęcia „zbiór";

w modelu relacyjnym atrybuty są nieuporządkowane i unikatowe. Atrybut jest identyfikowany przez nazwę atrybutu i nazwę typu. Na przykład nagłówek relacji *Employees* może składać się z następujących atrybutów (wyrażonych jako pary – nazwa atrybutu i nazwa typu): *employeeid integer* (identyfikator pracownika, liczba całkowita), *firstname character string* (imię, ciąg znaków), *lastname character string* (nazwisko, ciąg znaków), *birthdate date* (data urodzin, data), *departmentid integer* (identyfikator działu, liczba całkowita).

Typ jest jednym z najbardziej fundamentalnych bloków konstrukcyjnych relacji. Typ ogranicza atrybut do pewnego zestawu możliwych lub poprawnych wartości. Na przykład typ *int* jest zbiorem wszystkich liczb całkowitych z zakresu od –2147483648 do 2147483647. Typ jest w bazie danych jedną z najprostszych form predykatu, ponieważ ogranicza dopuszczane wartości atrybutu. Na przykład baza danych nie będzie akceptować tezy, w której data urodzin pracownika to 31 luty 1971 (nie wspominając o dacie podanej jako coś w rodzaju „abc!"). Zwróćmy uwagę, że nie jesteśmy ograniczeni do typów podstawowych, takich jak liczby całkowite czy ciąg znaków; typ może być również wyliczeniem możliwych wartości, jak na przykład lista możliwych stanowisk pracy. Typ może być złożoną konstrukcją. Prawdopodobnie najlepszym sposobem myślenia o typie jest przyrównanie go do klasy – uwzględniane dane i sposób ich obsługi. Przykładem typu złożonego jest typ geometryczny, który obsługuje wielokąty.

Brakujące wartości

Istnieje aspekt modelu relacyjnego, który jest źródłem wielu emocjonujących dyskusji – czy predykaty powinny być ograniczone do dwuwartościowej logiki. W przypadku dwuwartościowej logiki predykat może mieć wartość prawda albo fałsz. Jeśli predykat nie jest prawdą, musi być fałszem. Używanie dwuwartościowej logiki predykatu jest zgodne z matematycznym prawem zwanym prawem wyłączonego środka. Niektórzy jednak twierdzą, że istnieje przestrzeń dla stosowania trójwartościowej logiki (czy nawet logiki o czterech wartościach), gdzie można by uwzględniać przypadki, w których brakuje wartości. Predykat zastosowany do brakującej wartości nie generuje prawdy lub fałszu, a wartość *nieznaną*. Weźmy dla przykładu atrybut telefonu komórkowego relacji *Employees*. Załóżmy, że brak jest informacji o numerze telefonu niektórych pracowników. W jaki sposób przedstawić ten fakt w bazie danych? W przypadku zaimplementowania trójwartościowej logiki atrybut telefonu komórkowego powinien pozwalać na stosowanie specjalnego znaku dla brakującej wartości. Następnie predykat porównując atrybut telefonu z pewnym określonym numerem wygeneruje wartość *nieznany* dla przypadków, kiedy brak tej wartości. Trójwartościowa logika predykatu generuje jedną z trzech możliwych wartości logicznych – *prawda*, *fałsz* i *nieznany*.

Niektóre osoby są przekonane, że trójwartościowa logika predykatu jest nie-relacyjna, podczas gdy inni są zupełnie odmiennego zdania. W rzeczywistości Codd był zwolennikiem czterowartościowej logiki predykatu twierdząc, że istnieją dwa różne

przypadki wartości brakujących: brakująca, ale mająca zastosowanie (A-Mark) oraz brakująca, ale i bez zastosowania (I-Mark). Przykładem wartości A-Mark jest sytuacja, kiedy pracownik ma telefon komórkowy, ale nie znamy numeru tego telefonu. Przykładem wartości I-Mark jest sytuacja, kiedy pracownik w ogóle nie posiada telefonu. Według Codda do obsługi tych dwóch przypadków wartości brakujących powinny być używane dwa specjalne znaczniki. Język SQL implementuje trójwartościową logikę predykatu, obsługując znacznik *NULL* dla oznaczania ogólnego pojęcia wartości brakującej. Obsługa znaczników *NULL* i trójwartościowej logiki predykatu w SQL jest źródłem różnych pomyłek i złożoności, chociaż można się upierać, że brakujące wartości to nieusuwalna cecha świata rzeczywistego. Ponadto podejście alternatywne, czyli stosowanie wyłącznie dwuwartościowej logiki predykatu, wcale nie jest mniej problematyczne.

Ograniczenia

Jedną z największych zalet modelu relacyjnego jest możliwość definiowania integralności danych jako części modelu. Integralność danych jest osiągana poprzez reguły nazywane *ograniczeniami* (*constraint*), które są definiowane w modelu danych i wymuszane przez system RDBMS. Najprostsze metody wymuszania integralności danych to przypisywanie typu atrybutu z powiązaną z nim możliwością obsługi znaczników *NULL* lub brakiem tej obsługi. Ograniczenia są wymuszane również poprzez sam model; na przykład relacja *Orders*(*orderid*, *orderdate*, *duedate*, *shipdate*) [Zamówienia(id, data, data płatności, data wysyłki)] dla jednego zamówienia dopuszcza trzy różne daty, podczas gdy relacje *Employees*(*empid*) [Pracownicy(id)] i *EmployeeChildren*(*empid*, *childname*) [DzieciPracownika(id pracownika, imię dziecka)] zezwalają na liczbę dzieci z zakresu od 0 do (przeliczalnej) nieskończoności.

Innymi przykładami ograniczeń są *klucze kandydujące* (*candidate keys*), zwane też *potencjalnymi*, które zapewniają integralność krotki*, oraz *klucze obce* (*foreign keys*), które zapewniają integralność referencyjną. Klucz kandydujący to klucz zdefiniowany w oparciu o jeden lub kilka atrybutów, które uniemożliwiają pojawianie się w relacji więcej niż jednej instancji tej samej krotki (wiersza w SQL). Predykat oparty na kluczu kandydującym jednoznacznie identyfikuje wiersz (na przykład pracownika). W relacji można definiować wiele kluczy kandydujących. Na przykład w relacji *Employees* możemy zdefiniować klucze kandydujące w oparciu o atrybut *employeeid* (id pracownika), *SSN* (numer ubezpieczenia) i inne. Zazwyczaj arbitralnie wybieramy jeden z kluczy kandydujących jako *klucz główny* (*primary key*), na przykład *employeeid* w relacji *Employees* i używamy tego klucza jako preferowanej metody identyfikowania

* Krotka (ang. *tuple*) – struktura danych będąca odzwierciedleniem matematycznej *n*-ki, tj. uporządkowanego ciągu wartości. Rekordy relacyjnych baz danych są krotkami, gdyż poszczególne pola rekordów mogą zawierać dane różnych typów. W bazach SQL rekordy są nazywane wierszami (ang. *row*), zaś zbiory krotek definiowane przez relacje – tabelami. (przyp. red.).

wiersza. Wszystkie pozostałe klucze kandydujące nazywane są *kluczami alternatywnymi* (*alternate key*).

Klucze obce są używane do wymuszania integralności referencyjnej. Klucz obcy definiowany jest w oparciu o jeden lub kilka atrybutów relacji (*relacja referencyjna, odwołująca się*) i odnosi się do klucza kandydującego w innej (lub niekiedy w tej samej) relacji. Ta więź ogranicza wartości w atrybutach klucza obcego relacji odwołującej się do wartości, które występują w atrybutach klucza kandydującego relacji, do której następuje odwołanie. Załóżmy na przykład, że relacja *Employees* ma klucz obcy zdefiniowany w oparciu o atrybut *departmentid* (id działu), który odwołuje się do atrybutu *departmentid* klucza głównego w relacji *Departments*. Oznacza to, że wartości *Employees.departmentid* zostaną ograniczone do wartości, które występują w *Departments.departmentid*.

Normalizacja

Model relacyjny definiuje również *reguły normalizacji* (*normalization rules*), nazywane również *postaciami normalnymi* (*normal forms*). Normalizacja to formalny proces matematyczny, który gwarantuje, że każda jednostka będzie reprezentowana przez pojedynczą relację. W znormalizowanej bazie danych zapobiegamy powstawaniu nieprawidłowości i utrzymujemy redundancję na minimalnym poziomie bez utraty danych. Jeśli postępujemy zgodnie z modelowaniem ERM (Entity Relationship Modeling) i przedstawiamy każdą jednostkę i jej atrybuty, normalizacja prawdopodobnie nie będzie potrzebna; normalizację będziemy wtedy stosować jedynie do wzmocnienia poprawności działania modelu. W kolejnych podpunktach skrótowo opisano trzy postaci normalne (1NF, 2NF i 3NF) wprowadzone przez Codda.

1NF Pierwsza postać normalna określa, że krotka (wiersz) w relacji (tabeli) musi być unikatowa, a atrybuty powinny być niepodzielne na mniejsze wartości (atomowość danych). Jest to nadmiarowa definicja relacji; inaczej mówiąc, jeśli tabela rzetelnie odzwierciedla relację, spełnia już pierwszą postać normalną.

Niepowtarzalność wierszy osiągamy definiując dla tabeli unikatowy klucz.

Do atrybutów można stosować tylko te operacje, które są zdefiniowane jak część typu atrybutu. Atomowość atrybutów jest cechą subiektywną, podobnie jak subiektywna jest definicja zbioru. Powstaje na przykład pytanie, czy nazwa pracownika w relacji *Employees* powinna być wyrażana za pomocą jednego atrybutu (nazwisko), dwóch atrybutów (imię i nazwisko) czy trzech atrybutów (imię, drugie imię i nazwisko)? Odpowiedź zależna jest od aplikacji. Jeśli w aplikacji trzeba oddzielnie operować na częściach nazwy użytkownika (na przykład w celu wyszukiwania), sensownie jest je rozdzielać; w przeciwnym razie – nie.

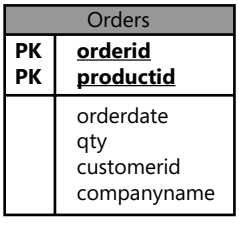
Podobnie jak atrybut może nie być wystarczająco „atomowy" w odniesieniu do potrzeb aplikacji, również dla atrybutu może istnieć możliwość jego podzielenia. Jeśli na przykład dla danej aplikacji atrybut adresu jest uważany za atomowy,

niedołączenie miasta jako części adresu będzie niespełnieniem warunku pierwszej postaci normalnej.

Ta postać normalna jest często rozumiana nieprawidłowo. Niektóre osoby uważają, że próba naśladowania tablic narusza pierwszą postać normalną. Przykładem może być definiowanie relacji *YearlySales* (sprzedaż roczna) za pomocą następujących atrybutów: *salesperson* (sprzedawca), *qty2010* (ilość w 2010), *qty2011* i *qty2012*. W tym przykładzie jednak tak naprawdę nie naruszamy pierwszej postaci normalnej; po prostu nakładamy ograniczenie – ograniczamy dane do trzech określonych lat: 2010, 2011 i 2012.

2NF Druga postać normalna zakłada dwie reguły. Pierwsza reguła określa, że dane muszą spełniać wymagania pierwszej postaci normalnej. Druga reguła dotyczy zależności pomiędzy atrybutami klucza kandydującego a atrybutami, które nie są częścią klucza. Dla każdego klucza kandydującego, każdy atrybut niebędący częścią klucza musi być w pełni funkcjonalnie zależny od całego klucza kandydującego. Inaczej mówiąc, atrybut niebędący częścią klucza nie może być w pełni funkcjonalnie zależny od części klucza kandydującego. Aby to trochę uprościć, jeśli potrzebujemy uzyskać wartość pewnego atrybutu niebędącego częścią klucza, trzeba dostarczyć wartości wszystkich atrybutów klucza kandydującego tej samej krotki. Możemy wyszukać dowolną wartość dowolnego atrybutu dowolnej krotki, jeśli znamy wszystkie wartości atrybutów klucza kandydującego.

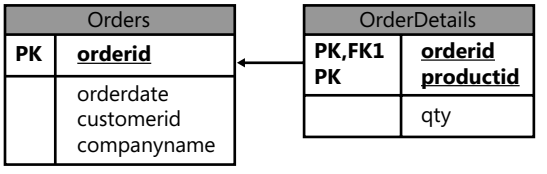
Dla przytoczenia przykładu naruszenia drugiej postaci normalnej załóżmy, że zdefiniowaliśmy relację nazwaną *Orders* (zamówienia), która reprezentuje informacje o zamówieniach i jego pozycjach (rysunek 1-1). Relacja *Orders* zawiera następujące atrybuty: *orderid*, *productid*, *orderdate*, *qty*, *customerid* i *companyname* (identyfikator zamówienia, identyfikator produktu, data zamówienia, ilość, identyfikator klienta, nazwa firmy). Klucz główny zdefiniowany jest w oparciu o atrybuty *orderid* i *productid*.



Rysunek 1-1 Model danych przed zastosowaniem postaci 2NF

Na rysunku 1-1 nie jest spełniona druga postać normalna, ponieważ istnieją atrybuty niebędące częścią klucza, które są zależne tylko od części klucza kandydującego (w tym przykładzie od klucza głównego). Na przykład, możemy znaleźć atrybut *orderdate* zamówienia, a także atrybuty *customerid* i *companyname*, bazując jedynie na samym atrybucie *orderid*. Aby spełniona była druga postać normalna, trzeba rozdzielić pierwotną relację na dwie relacje: *Orders* oraz *OrderDetails* (rysunek 1-2). Relacja *Orders* będzie zawierać atrybuty *orderid*, *orderdate*, *customerid* oraz *companyname*, wraz

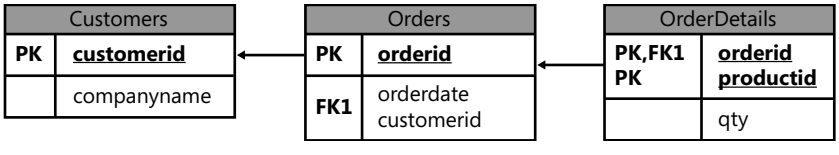
z kluczem głównym zdefiniowanym za pomocą atrybutu *orderid*. Relacja *OrderDetails* będzie zawierać atrybuty *orderid*, *productid* oraz *qty*, wraz z kluczem głównym zdefiniowanym za pomocą atrybutów *orderid* i *productid*.



Rysunek 1-2 Model danych po zastosowaniu postaci 2NF, a przed zastosowaniem postaci 3NF

3NF Trzecia postać normalna również kieruje się dwoma regułami. Dane muszą spełniać drugą postać normalną. Ponadto wszystkie atrybuty niebędące częścią klucza muszą być zależne od kluczy kandydujących w sposób nieprzechodni. Upraszczając, reguła ta oznacza, że wszystkie atrybuty niebędące częścią klucza muszą być wzajemnie niezależne. Inaczej mówiąc, jeden atrybut niebędący częścią klucza nie może być zależny od innego takiego atrybutu.

Opisane poprzednio relacje *Orders* i *OrderDetails* obecnie spełniają drugą postać normalną. Pamiętamy, że w tym momencie relacja *Orders* zawiera atrybuty *orderid*, *orderdate*, *customerid* i *companyname*, wraz z kluczem głównym zdefiniowanym w oparciu o atrybut *orderid*. Oba atrybuty *customerid* i *companyname* zależne są od całego klucza głównego – *orderid*. Na przykład, aby wyszukać atrybut *customerid* reprezentujący klienta, który złożył zamówienie, potrzebny jest cały klucz główny. Podobnie potrzebny jest cały klucz główny, by wyszukać nazwę firmy klienta, który złożył zamówienie. Atrybuty *customerid* i *companyname* są jednak również zależne od siebie samych. Aby spełnione były wymagania trzeciej postaci normalnej, trzeba dodać relację *Customers* (rysunek 1-3) wraz z atrybutami *customerid* (klucz główny) i *companyname*. Następnie możemy usunąć atrybut *companyname* z relacji *Orders*.



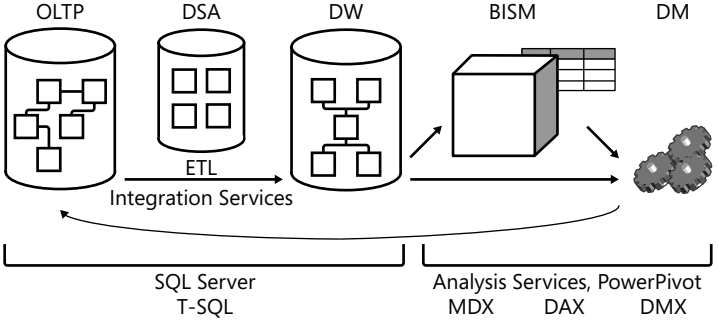
Rysunek 1-3 Model danych po zastosowaniu postaci 3NF

Nieformalnie postaci 2NF i 3NF są zazwyczaj podsumowywane za pomocą następującego stwierdzenia: „Każdy atrybut niebędący częścią klucza jest zależny od klucza, od całego klucza i od niczego poza kluczem – Codd, pomóż proszę".

Istnieją wyższe postaci normalne, poza pierwszymi trzema opracowanymi przez Codda, które wymagają powiązań kluczy głównych i tymczasowych baz danych, ale tematyka ta wykracza poza zakres książki.

Cykl życia danych

Dane zazwyczaj są postrzegane jako statyczne elementy, wprowadzane do bazy danych i które później są wyszukiwane. Jednak w wielu środowiskach dane przypominają bardziej produkt na linii montażowej, przenoszony z jednego środowiska do innego, który w swojej wędrówce podlega stałym modyfikacjom. Niniejszy podrozdział opisuje różne środowiska, w których dane mogą być przechowywane, oraz cechy charakterystyczne zarówno danych, jak i środowisk na każdym etapie cyklu życia danych. Rysunek 1-4 ilustruje cykl życia danych.



Rysunek 1-4 Cykl życia danych

Poniżej wyjaśniono znaczenie poszczególnych akronimów.

- OLTP: Online Transactional Processing
- DSA: Data Staging Area
- DW: Data Warehouse
- BISM: Business Intelligence Semantic Model
- DM: Data Mining
- ETL: Extract, Transform, Load
- MDX: Multidimensional Expressions
- DAX: Data Analysis Expressions
- DMX: Data Mining Extensions

Online Transactional Processing

Dane początkowo są wprowadzane do systemu przetwarzania transakcji w trybie online (Online Transactional Processing – OLTP). System OLTP skupia się na wprowadzaniu danych, a nie na tworzeniu raportów – transakcje dotyczą głównie wstawiania, aktualizowania i usuwania danych. Model relacyjny jest ukierunkowany głównie na systemy OLTP, gdzie znormalizowany model zapewnia zarówno dobrą wydajność wprowadzania danych, jak i ich spójność. W znormalizowanym środowisku każda

tabela reprezentuje pojedynczą jednostkę i minimalizuje redundancję. Jeśli zachodzi potrzeba zmodyfikowania faktu, trzeba go modyfikować tylko w jednym miejscu. Dzięki temu optymalizowana jest wydajność modyfikowania danych i zmniejszane prawdopodobieństwo błędu.

Środowisko OLTP nie jest jednak odpowiednie do raportowania, ponieważ znormalizowany model zazwyczaj korzysta z wielu tabel (jedna dla każdej jednostki) o złożonych powiązaniach. Nawet prosty raport wymaga złożenia wielu tabel, co generuje złożone kwerendy o niskiej wydajności.

W systemie SQL Server możemy implementować bazy danych OLTP, zarządzać nimi i wykonywać kwerendy przy użyciu języka T-SQL.

Data Warehouse

Hurtownie danych DW (Data Warehouse) to środowisko opracowane do pozyskiwania danych i raportowania. Jeśli środowisko to służy całej organizacji, nazywane jest hurtownią danych; jeśli służy jedynie części organizacji (na przykład określonemu działowi) lub tematycznemu obszarowi w organizacji, jest nazywane *data mart* (*minihurtowania, magazyn danych*). Model danych hurtowni danych został opracowany i zoptymalizowany tak, by przede wszystkim zapewnić obsługę funkcji pozyskiwania danych. Model zawiera zamierzoną redundancję, mniejszą liczbę tabel i uproszczone powiązania, co powoduje, że w porównaniu do środowiska OLTP kwerendy są prostsze i działają efektywniej.

Najprostszy projekt hurtowni danych nazywany jest *schematem gwiazdy* (*star schema*). Schemat gwiazdy obejmuje kilka tabel wymiarów i tabelę faktów. Każda tabela wymiarów reprezentuje podmiot, który ma być podstawą analizy danych. Na przykład w systemie obsługi zamówień i sprzedaży będziemy prawdopodobnie chcieli analizować dane według klientów, produktów, pracowników, czasu i tego typu podobnych podmiotów. W schemacie gwiazdy każdy wymiar jest implementowany w postaci pojedynczej tabeli z nadmiarowymi danymi. Na przykład wymiar produktu mógłby zostać zaimplementowany w postaci pojedynczej tabeli *ProductDim*, zamiast trzech znormalizowanych tabel: *Products*, *ProductSubCategories* i *ProductCategories* (*Produkty*, *Podkategorie_produktu* i *Kategorie_produktu*). Jeśli normalizujemy tabelę wymiarów, co spowoduje powstanie wielu tabel reprezentujących ten wymiar, otrzymamy strukturę, która nazywana jest wymiarem *płatka śniegu* (*snowflake dimension*). Schemat zawierający wymiary płatka śniegu nazywany jest *schematem płatka śniegu* (w odróżnieniu do schematu gwiazdy).

W tabeli faktów przechowywane są fakty i miary, takie jak ilość czy wartość każdego istotnego połączenia kluczy wymiarów. Na przykład dla każdego istotnego połączenia klienta, produktu, pracownika i dnia tabela faktów może zawierać wiersz zawierający ilość i wartość. Zwróćmy uwagę, że dane w hurtowni danych są zazwyczaj wstępnie przetworzone (posumowane) do pewnego poziomu rozdrobnienia (na przykład

z podziałem na dni), inaczej niż w przypadku danych w środowisku OLTP, które zwykle są rejestrowane na poziomie transakcji.

Historycznie rzecz biorąc, wczesne wersje systemu SQL Server były głównie ukierunkowane na środowiska OLTP, jednak ostatecznie system SQL Server zaczął również skupiać się na systemach hurtowni danych i zapewniać odpowiednią analizę danych. Możemy implementować hurtownię danych jako bazę danych systemu SQL Server oraz zarządzać danymi i wykonywać kwerendy za pomocą T-SQL.

Proces wyciągania danych z systemów źródłowych (OLTP i inne), opracowywania danych i ładowania ich do hurtowni danych nazywany jest procesem ETL (Extract, Transform, Load). Do obsługi procesu ETL system SQL Server udostępnia narzędzie nazwane Microsoft SQL Server Integration Services (SSIS).

Proces ETL będzie często korzystał z warstwy danych DSA (Data Staging Area – DSA), znajdującej się pomiędzy OLTP a DW. Warstwa DSA umieszczona jest zazwyczaj w relacyjnej bazie danych, takiej jak SQL Server i jest używana jako obszar czyszczenia danych. Warstwa DSA nie jest dostępna dla użytkowników końcowych.

Business Intelligence Semantic Model

Model danych BISM (Business Intelligence Semantic Model) to najnowszy model obsługi całego stosu aplikacji BI opracowany w firmie Microsoft. Jego celem jest zapewnienie różnorodnych, elastycznych, sprawnych i skalowalnych funkcji analizy i raportowania. Architektura modelu obejmuje trzy warstwy: modelu danych, logiki biznesowej i kwerend oraz dostępu do danych.

Model można instalować na serwerze Analysis Services lub PowerPivot. Usługi Analysis Services są dedykowane specjalistom BI i IT, natomiast PowerPivot skierowany jest do użytkowników biznesowych. Dzięki usługom Analysis Services możemy stosować model danych wielowymiarowy lub tabelaryczny (relacyjny). W przypadku Power-Pivot używamy tabelarycznego modelu danych.

Logika biznesowa i kwerendy korzystają z dwóch języków: MDX (Multidimensional Expressions) bazującego na pojęciach wielowymiarowości oraz DAX (Data Analysis Expressions) bazującego na koncepcji tabel.

Warstwa dostępu do danych może uzyskiwać dane z różnych źródeł: relacyjne bazy danych, takie jak DW, pliki, usługi chmury, aplikacje LOB (Line of Business), źródła OData i inne. Warstwa dostępu do danych może buforować dane lokalnie albo może służyć jako warstwa przekazująca dane bezpośrednio ze źródeł. Tryb buforowania może używać jeden z dwóch mechanizmów magazynowania. Jednym z nich jest MOLAP, który pierwotnie opracowany został do obsługi modelu wielowymiarowego, a drugim jest nowszy mechanizm nazwany VertiPaq, który implementuje koncepcję magazynu kolumn i charakteryzuje się wysokim poziomem kompresji i bardzo szybkim przetwarzaniem, poprzez wyeliminowanie potrzeby wstępnego przetwarzania, indeksowania itp.

Zobacz także Tematyka modelu BISM obejmuje sporo zagadnień wymagających dokładnego omówienia – prawdopodobnie zbyt wielu, by zamieścić je w książce o podstawach T-SQL. Dodatkowe informacje o modelu BISM znaleźć można na blogu zespołu Analysis Services: *http://blogs.msdn.com/b/analysisservices/archive/2011/05/16/analysis-services--vision-amp-roadmap-update.aspx*.

Data Mining

Model BISM umożliwia użytkownikom uzyskanie odpowiedzi na wszystkie możliwe pytania, ale zadaniem użytkownika jest postawienie właściwego pytania – by w morzu danych wykryć anomalie, trendy i inne przydatne informacje. W trakcie dynamicznego procesu analizy danych użytkownik przemieszcza się od jednego widoku posumowanych danych do innego, ciągle w różny sposób grupując dane, by wyszukać potrzebne informacje.

Eksploracja danych DM (Data Mining) jest kolejnym etapem; zamiast pozwolić użytkownikowi na wyszukiwanie przydatnych informacji w morzu danych, modele eksploracji wykonują te zadania za użytkownika. Algorytmy eksploracji przeczesują i przesiewają dane, wydobywając przydatne informacje. Eksploracja danych stanowi niewiarygodnie istotną wartość biznesową dla organizacji, ułatwia zidentyfikowanie trendów, dostrzeżenie, które produkty mogą być sprzedawane razem, przewidywanie wyborów klientów w oparciu o określone parametry i rozwiązywanie wielu innych zagadnień.

Usługi Analysis Services obsługują algorytmy eksploracji danych, które spełniają takie potrzeby, wliczając w to tworzenie klastrów, drzewa decyzyjne i inne. Językiem używanym do zarządzania danymi i realizacji kwerend w modelach eksploracji danych jest DMX (Data Mining Extensions).

Architektura SQL Server

Niniejszy podrozdział jest wprowadzeniem do architektury systemu SQL Server, jego odmian i używanych jednostek – instancji SQL Server, baz danych, schematów i obiektów danych – oraz opisuje przeznaczenie wszystkich tych elementów.

Odmiany ABC produktu SQL Server

Przez wiele lat system SQL Server był dostępny tylko w jednej odmianie – w siedzibie, czyli w wersji pudełkowej (box). Niedawno firma Microsoft zdecydowała się na udostępnienie kilku odmian, by klienci mogli wybrać taką, która im najbardziej odpowiada. W okresie opracowywania książki firma Microsoft oferowała trzy odmiany produktu SQL Server, które wewnątrz firmy są nazywane ABC: A dla Appliance, B dla Box oraz C dla Cloud.

Wersja Appliance

Pomysł związany z wyróżnieniem odmiany A wywodzi się z chęci udostępnienia kompletnego rozwiązania, które obejmuje sprzęt, oprogramowanie i usługi. Wersja ta jest utrzymywana lokalnie po stronie klienta. Obecnie mamy dostępnych kilka wersji w tej grupie produktu, a jedną z nich jest PDW (Parallel Data Warehouse). Partnerzy firmy Microsoft wraz z dostawcami sprzętu, takimi jak firmy Dell czy HP, oferują ten produkt. Eksperci z firmy Microsoft i dostawcy sprzętu zapewniają klientowi wydajność, bezpieczeństwo i dostępność jego systemu.

Ta książka skupia się na tematyce opisującej język T-SQL, tak więc można zastanawiać się, jaki język jest używany do interakcji z mechanizmem bazy danych. Jest to zależne od urządzenia. Na przykład PDW nie używa tego samego silnika, co wersja używana w siedzibie – stosowany jest silnik specjalizowany. PDW korzysta z własnej odmiany SQL nazywanej DSQL (Distributed SQL). Długoterminowe cele firmy Microsoft przewidują zastosowanie podobnej obsługi języka w różnych odmianach produktu, ale jak dotąd cel ten nie został jeszcze osiągnięty. Ta książka omawia dialekt języka T-SQL, który jest obsługiwany przez niektóre urządzenia w wersji A, ale przede wszystkim jest stosowany w wersjach dla siedziby i chmury.

Wersja Box

Odmiana Box produktu SQL Server, formalnie nazywana systemem SQL Server *w siedzibie* (*on-premises*), jest tradycyjnym sposobem używania produktu, zazwyczaj instalowanego w siedzibie klienta. Klient jest odpowiedzialny za wszystko – uzyskanie sprzętu, zainstalowanie oprogramowania i obsługę aktualizacji, zapewnienie wysokiego poziomu dostępności i przywracania po awariach (HADR), bezpieczeństwo i wszystko inne.

Klient może instalować wiele instancji produktu na tym samym serwerze (więcej na ten temat w kolejnym podrozdziale) i może pisać kwerendy, które współdziałają z wieloma bazami danych. Istnieje również możliwość przełączania się pomiędzy bazami danych, chyba że jedna z nich jest typu *contained* (zawarta).

Używanym językiem kwerend jest T-SQL. Wszystkie przykłady kodu i ćwiczenia przytoczone w tej książce możemy uruchamiać na implementacji tej wersji systemu SQL Server (w wersji w siedzibie). W Dodatku znaleźć można informacje szczegółowe na temat uzyskania i instalowania wersji testowej SQL Server, a także na temat tworzenia przykładowych baz danych.

Wersja Cloud

Firma Microsoft obsługuje dwie odmiany produktu SQL Server dla chmury (Cloud): chmura prywatna i publiczna. Używanie pojęcia chmury prywatnej może być trochę mylące, ponieważ jest ona utrzymywana lokalnie, ale „prywatna" odmiana produktu korzysta z technologii wirtualizacji. Używany mechanizm jest taki sam jak dla odmiany

typu Box (w związku z tym ten sam język T-SQL używany jest do tworzenia kwerend), jednak stosują się do tego mechanizmu ograniczenia technologii wirtualizacji, takie jak liczba obsługiwanych procesorów czy wielkość pamięci.

Chmura publiczna nazywana jest Windows Azure SQL Database (poprzednio SQL Azure). Produkt utrzymywany jest w centrach danych firmy Microsoft, która jest w całości odpowiedzialna za sprzęt, konserwację, HADR i aktualizację. Klient nadal jest jednak odpowiedzialny za indeksowanie i dostrajanie kwerend.

Uwaga Kolejne odniesienia do produktu „Windows Azure SQL Database" będą używać skróconej nazwy „SQL Database".

Używając produktu SQL Database klient uzyskuje dostęp do wielu baz danych na serwerze w chmurze (oczywiście serwerze jako koncepcji – nie jest istotna i nie interesuje nas fizyczna implementacja), ale może łączyć się jednocześnie z jedną bazą danych. Klient nie może przełączać się pomiędzy bazami danych i nie może pisać kwerend odnoszących się do wielu baz danych.

Mechanizm produktu SQL Database to specjalizowany mechanizm, chociaż firma Microsoft stosuje tę samą podstawę kodu co w przypadku wersji dla siedziby. Tak więc funkcje T-SQL udostępniane w SQL Database są zasadniczo takie same jak w wersji lokalnej. Większość funkcji T-SQL, które poznamy w tej książce, można stosować w obu wersjach (w siedzibie i w chmurze) SQL Server, chociaż istnieje kilka wyjątków, jak funkcje wersji SQL Server T-SQL w siedzibie, które nie zostały jeszcze zaimplementowane w odmianie SQL Database. Dokumentacja Books Online for SQL Database szczegółowo opisuje te funkcje w rozdziale Transact-SQL Reference pod adresem *http://msdn.microsoft.com/en-us/library/windowsazure/ee336281.aspx*. Warto także pamiętać, że w wersji SQL Database większa jest szybkość aktualizacji i instalowania nowych funkcji, niż w przypadku SQL Server wzainstalowanego siedzibie. Z tego względu istnieje możliwość, że niektóre funkcje T-SQL mogą być udostępnione w odmianie SQL Database, zanim pojawią się w wersji SQL Server dla siedziby.

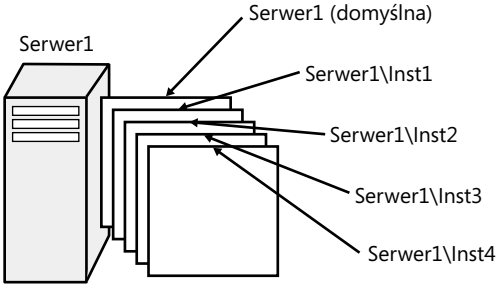
Jak już wspomniano, większość funkcji T-SQL omawianych w tej książce jest już dostępnych lub będą dostępne w wersji SQL Database. W Dodatku, we fragmencie opisującym instalacje przykładowej bazy danych dla tej książki, opisano, jak zainstalować przykładową bazę danych w SQL Database w przypadku, kiedy mamy już dostęp do tej wersji.

Instancje produktu SQL Server

Instancja systemu SQL Server, jak ilustruje to rysunek 1-5, jest instalacją mechanizmu bazy danych lub usługi SQL Server. Na tym samym komputerze możemy instalować wiele instancji produktu SQL Server w wersji dla siedziby. Każda instancja jest całkowicie niezależna od innych pod względem kontekstu zabezpieczeń, danych, którymi

zarządza i w każdym innym. Na poziomie logicznym dwie różne instancje umieszczone na tym samym komputerze nie mają więcej wspólnych elementów, niż dwie instancje umieszczone na różnych komputerach. Rzecz jasna, instancje umieszczone na tym samym komputerze współużytkują fizyczne zasoby serwera, takie jak CPU, pamięć czy dysk.

Jedną z wielu instancji na komputerze możemy skonfigurować jako *instancję domyślną*, natomiast pozostałe muszą być *instancjami nazwanymi*. To, czy instancja jest instancją domyślną, czy nazwaną, określane jest podczas instalacji; decyzji tej nie można później zmienić. W celu połączenia się z instancją domyślną aplikacja kliencka musi wskazać nazwę komputera lub jego adres IP. W celu połączenia się z instancją nazwaną program kliencki specyfikuje nazwę komputera lub adres IP, po których umieszczany jest odwrotny ukośnik (\) i nazwa instancji (określona podczas instalacji). Załóżmy dla przykładu, że mamy dwie instancje SQL Server zainstalowane na komputerze nazwanym Serwer1. Jedna z tych instancji została zainstalowana jako domyślna, a druga jako instancja nazwana – Inst1. W celu podłączenia się do instancji domyślnej, trzeba wyspecyfikować jedynie Serwer1 jako nazwę serwera, natomiast aby połączyć się z instancją nazwaną, trzeba wyspecyfikować zarówno nazwę serwera, jak i nazwę instancji: Serwer1\Inst1.



Rysunek 1-5 Wiele instancji SQL Server na tym samym komputerze

Istnieją różne powody, dla których instalowanych jest wiele instancji systemu SQL Server na tym samym komputerze, jednak wspomnimy tylko o niektórych z nich. Jedną z przyczyn jest zmniejszenie kosztów obsługi. Na przykład w celu przetestowania działania funkcji w odpowiedzi na odebrane zgłoszenia lub odtworzenia napotkanych przez użytkowników błędów w środowisku produkcyjnym dla działu obsługi technicznej potrzebne są lokalne instalacje systemu SQL Server, które imitują środowisko produkcyjne w kontekście wersji, wydania i zainstalowanych pakietów serwisowych produktu SQL Server. Jeśli w organizacji jest wiele środowisk użytkowników, dział techniczny potrzebuje wielu instalacji SQL Server. Zamiast utrzymywania wielu komputerów, na których instalowane są różne instalacje systemu SQL Server i które muszą być obsługiwane oddzielnie, można mieć jeden komputer z zainstalowanymi wieloma instancjami. Oczywiście ten sam rezultat można uzyskać używając maszyn wirtualnych.

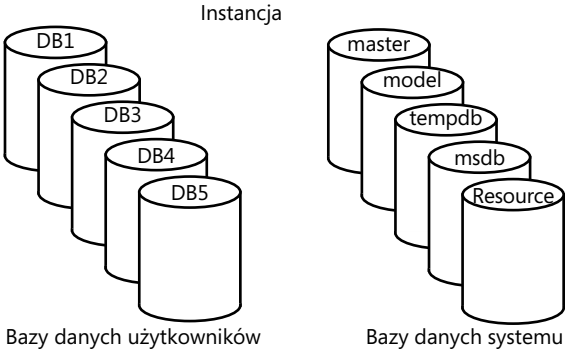
Innym przykładem są osoby, które mają zajęcia podobne do moich – uczą i prowadzą wykłady na temat produktu SQL Server. Dla takich osób bardzo wygodna jest możliwość zainstalowania wielu instancji systemu SQL Server na tym samym komputerze przenośnym. W ten sposób można na przykład prezentować różne wersje produktu, pokazując różnice w ich działaniu.

Jako ostatni przykład można przytoczyć sytuację dostawców usług bazodanowych, którzy czasami potrzebują zagwarantować swoim klientom kompletne odseparowanie danych jednego klienta od danych innych klientów. Zamiast utrzymywania wielu mało wydajnych komputerów z zainstalowanymi odrębnymi instancjami, można utrzymywać wydajne centrum danych, w którym utrzymywanych jest wiele instancji produktu SQL Server. Ostatnio rozwiązania chmury i zaawansowane technologie wirtualizacji również umożliwiają osiągnięcie tych samych celów.

Bazy danych

Bazę danych możemy traktować jako kontener zawierający takie obiekty, jak tabele, widoki, procedury składowane i inne. Każda instancja produktu SQL Server może zawierać wiele baz danych (rysunek 1-6).

Podczas instalowania wersji SQL Server dla siedziby program instalacyjny tworzy kilka systemowych baz danych przechowujących dane systemu, które mają wewnętrzne zastosowania. Po instalacji możemy tworzyć własne bazy danych użytkownika, w których przechowywane są dane aplikacji.



Rysunek 1-6 Przykład wielu baz danych w instancji systemu SQL Server

Systemowe bazy danych utworzone przez program instalacyjny są następujące: *master*, *Resource*, *model*, *tempdb* oraz *msdb*.

- ***master*** Baza danych *master* przechowuje informacje o metadanych dla całej instancji, konfigurację serwera, informacje o wszystkich bazach danych instancji i informacje związane z procesem inicjalizacji.

- ***Resource*** Baza danych *Resource* jest ukryta, przeznaczona tylko do odczytu i przechowuje definicje wszystkich obiektów systemu. Podczas wykonywania zapytań dotyczących obiektów systemowych w bazie danych wydaje się, że znajdują się one w schemacie *sys* lokalnej bazy danych, jednak w rzeczywistości umieszczone są w bazie danych *Resource*.
- ***model*** Baza danych *model* jest używana jako szablon dla nowych baz danych. Każda nowa baza danych jest początkowo tworzona jako kopia bazy *model*. Tak więc, jeśli chcemy, by pewne obiekty (takie jak typy danych) występowały we wszystkich nowych bazach danych, które tworzymy, lub by pewne właściwości bazy danych były w określony sposób skonfigurowane we wszystkich nowych bazach danych, trzeba utworzyć te obiekty i skonfigurować te właściwości w bazie danych *model*. Zwróćmy uwagę, że zmiany zastosowane w bazie danych *model* nie wpłyną na istniejące bazy danych – tylko na nowe bazy danych, które utworzymy w przyszłości.
- ***tempdb*** Baza danych *tempdb* to miejsce przechowywania przez system SQL Server danych tymczasowych, takich jak tabele robocze, obszary sortowania, informacje dotyczące wersji wiersza itp. System SQL Server pozwala tworzyć tabele tymczasowe dla naszego własnego zastosowania, a ich fizyczna lokalizacja to właśnie baza danych *tempdb*. Pamiętajmy, że ta baza danych jest usuwana i ponownie tworzona jako kopia bazy danych *model*, ilekroć ponownie uruchamiana jest instancja systemu SQL Server.
- ***msdb*** Baza danych *msdb* to miejsce, w którym swoje dane przechowuje usługa nazwana SQL Server Agent. Usługa SQL Server Agent odpowiada za automatyzację obejmującą takie jednostki, jak zadania, harmonogramy i alerty. Usługa SQL Server Agent odpowiada także za replikacje. Baza danych *msdb* również przechowuje informacje dotyczące innych funkcji systemu SQL Server, takich jak Database Mail, Service Broker, kopie zapasowe i inne.

W instalacji systemu SQL Server w wersji dla siedziby możemy bezpośrednio łączyć się z systemowymi bazami danych *master*, *model*, *tempdb* i *msdb*. W produkcie SQL Database możemy bezpośrednio łączyć się tylko z bazą danych *master*. Jeśli tworzymy tymczasowe tabele lub deklarujemy zmienne tabel (więcej informacji na ten temat znaleźć można w rozdziale 10 „Obiekty programowalne"), są one tworzone w bazie danych *tempdb*, ale nie możemy bezpośrednio łączyć się z bazą *tempdb* i wprost w niej tworzyć obiekty.

Wewnątrz instancji możemy tworzyć dowolną liczbę baz danych użytkownika. Baza danych użytkownika przechowuje obiekty i dane przeznaczone dla aplikacji.

Na poziomie bazy danych można zdefiniować właściwość nazwaną *collation*, która będzie określać obsługę języka, wrażliwość na wielkość znaków i kolejność sortowania znaków w tej bazie danych. Jeśli podczas tworzenia bazy danych nie zostanie wyspecyfikowana właściwość *collation*, nowa baza danych posługuje się domyślną właściwością *collation* instancji (wybraną podczas instalacji).

Aby uruchomić kod języka T-SQL w konkretnej bazie danych, aplikacja kliencka musi połączyć się z instancją SQL Server i znaleźć się w kontekście, czyli zacząć używać odnośnej bazy danych.

W kwestii zabezpieczeń, aby można było połączyć się z instancją SQL Server, administrator bazy danych (DBA) musi dla nas utworzyć logon. W przypadku instancji SQL Server w wersji dla siedziby logon może być powiązany z naszymi poświadczeniami w systemie Windows i w takim przypadku nazywany jest to *logon uwierzytelniony w systemie Windows*. Przy użyciu uwierzytelnienia systemu Windows nie musimy przedstawiać logonu i hasła podczas łączenia się z systemem SQL Server, ponieważ poświadczenia te zostały już dostarczone podczas logowania się w systemie Windows. W obu produktach, SQL Server w siedzibie i SQL Database, logon może być niezależny od poświadczeń dla systemu Windows i w takim przypadku nazywany jest to *logon uwierzytelniony w systemie SQL Server*. Podczas łączenia się z systemem SQL Server przy użyciu uwierzytelnienia SQL Server trzeba przedstawić zarówno nazwę logowania, jak i hasło.

DBA musi przypisać nasz logon do bazy danych użytkownika w każdej bazie danych, do której chcemy uzyskać dostęp. Baza danych użytkownika jest jednostką, której przydzielane są uprawnienia do obiektów bazy danych.

System SQL Server 2012 obsługuje funkcję nazwaną *contained databases* (*zawarte w sobie, izolowane bazy danych*), która zrywa związek pomiędzy użytkownikiem bazy danych a logonem na poziomie serwera. Użytkownik całkowicie „zawiera się" wewnątrz określonej bazy danych i nie jest powiązany z logonem na poziomie serwera. Podczas tworzenia użytkownika, administrator bazy danych również dostarcza hasło. Podczas łączenia się z systemem SQL Server użytkownik musi wyspecyfikować bazę danych, z którą się łączy, a także nazwę użytkownika i hasło – później użytkownik nie może przełączać się do innych baz danych.

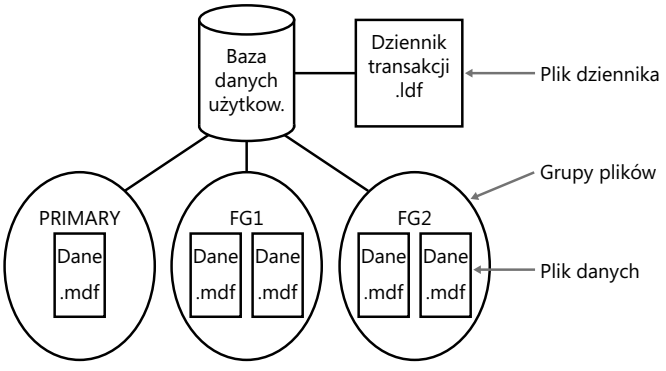
Do tej pory omawiane były logiczne aspekty baz danych. Korzystając z produktu SQL Database zajmujemy się tylko tą logiczną warstwą. Nie musimy zajmować się warstwą fizyczną danych bazy i plików dziennika, bazą *tempdb* itp. Jeśli jednak korzystamy z wersji SQL Server dla siedziby, jesteśmy odpowiedzialni także za warstwę fizyczną. Na rysunku 1-7 przedstawiono schemat fizycznego układu bazy danych.

Baza danych zbudowana jest z plików danych i plików dzienników transakcji. Podczas tworzenia bazy danych definiujemy różne właściwości każdego pliku, a w tym nazwę pliku, lokalizację, początkowy rozmiar, rozmiar maksymalny i wielkość automatycznego przyrostu. Każda baza danych musi mieć co najmniej jeden plik danych i co najmniej jeden plik dziennika (ustawienie domyślne w SQL Server). Pliki danych przechowują dane o obiektach, a pliki dzienników przechowują informacje, które potrzebne są systemowi SQL Server do utrzymywania transakcji.

Chociaż system SQL Server potrafi równolegle zapisywać wiele plików danych, pliki dziennika zapisywane są w trybie sekwencyjnym – w danym momencie może być zapisywany tylko jeden plik dziennika. Z tego względu, inaczej niż w przypadku

plików danych, posiadanie wielu plików dzienników nie jest korzystne dla wydajności. Dodajemy pliki dzienników, jeśli zapełniony zostaje dysk, na którym plik dziennika jest przechowywany.

Pliki danych są uporządkowane w postaci grup logicznych nazwanych grupami plików. Grupa plików jest miejscem docelowym tworzenia obiektów, takich jak tabela czy indeks. Dane obiektu znajdują się w plikach należących do docelowej grupy plików. Grupy plików to mechanizm kontrolowania fizycznej lokalizacji obiektów. Baza danych musi posiadać co najmniej jedną grupę plików nazwaną *PRIMARY,* a opcjonalnie może zawierać także inne grupy plików. Grupa plików *PRIMARY* zawiera podstawowy plik danych (o rozszerzeniu mdf) bazy danych oraz katalog systemowy bazy danych. W grupie *PRIMARY* opcjonalnie możemy dodawać pomocnicze pliki danych (pliki o rozszerzeniu ndf). Grupy plików użytkownika zawierają tylko pomocnicze pliki danych. Możemy określić, która grupa plików zostanie oznaczona jako domyślna. Obiekty tworzone są w domyślnej grupie plików, jeśli polecenie tworzenia obiektu nie specyfikuje wprost innej, docelowej grupy plików.



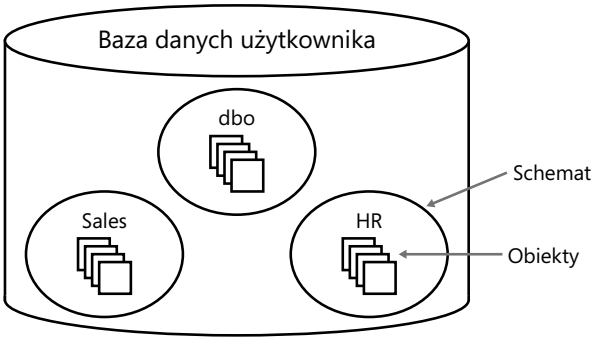
Rysunek 1-7 Układ bazy danych

Rozszerzenia plików mdf, ldf i ndf

Rozszerzenia plików bazy danych mdf i ldf są dość zrozumiałe. Rozszerzenie mdf oznacza Master Data File (nie należy mylić z wzorcową bazą danych), a rozszerzenie ldf oznacza Log Data File (plik danych dziennika). Zgodnie z pewną anegdotą, podczas dyskusji dotyczącej pomocniczych plików danych jeden z projektantów zasugerował żartobliwie, by użyć rozszerzenia ndf, które oznacza „Not Master Data File" – i tak pozostało.

Schematy i obiekty

Wcześniejsze stwierdzenie, że baza danych jest kontenerem obiektów, było pewnym uproszczeniem. Zgodnie z rysunkiem 1-8, baza danych zawiera schematy, a schematy zawierają obiekty. Schemat możemy traktować jak kontener obiektów takich jak tabele, widoki, procedury składowane i innych.



Rysunek 1-8 Baza danych, schematy i obiekty bazy danych

Uprawnienia można kontrolować na poziomie schematu. Na przykład możemy użytkownikowi przydzielić uprawnienie SELECT do schematu, co pozwoli użytkownikowi wykonywać kwerendy danych dla wszystkich obiektów tego schematu. Tak więc zabezpieczenia są jednym z czynników określania, jaki powinien być porządek obiektów w schemacie.

Schemat jest również przestrzenią nazw – jest używany jako prefiks nazwy obiektu. Załóżmy dla przykładu, że mamy tabelę nazwaną *Orders* w schemacie nazwanym *Sales*. Nazwa obiektu kwalifikowana schematem (nazywana również *dwuczęściową nazwą obiektu*) to *Sales.Orders*. Jeśli podczas odnoszenia się do obiektu pominiemy nazwę schematu, system SQL Server zastosuje proces rozpoznawania nazwy schematu – sprawdzenie, czy obiekt istnieje w domyślnym schemacie użytkownika, i jeśli nie, czy istnieje w schemacie *dbo*. Firma Microsoft zaleca, by podczas odwoływania się do obiektów w kodzie zawsze używać dwuczęściowych nazw obiektów. Jeśli nazwa nie jest wyspecyfikowana wprost, powstają pewne względnie niewielkie dodatkowe koszty związane z rozpoznawaniem nazwy obiektu. Tak więc, chociaż koszty te nie są wielkie, dlaczego w ogóle je ponosić? Ponadto, jeśli w różnych schematach istnieje wiele obiektów o tych samych nazwach, możemy też ostatecznie uzyskać nie ten obiekt, który jest potrzebny.

Tworzenie tabel i definiowanie integralności danych

W tym podrozdziale omówiono podstawy związane z tworzeniem tabel i definiowaniem integralności danych przy użyciu poleceń języka T-SQL. W naszym środowisku bez obaw możemy uruchamiać przytaczane przykłady kodu.

Informacje dodatkowe Jeśli nie wiemy, jak uruchamiać kod w systemie SQL Server, odpowiednie informacje na ten temat należć można w Dodatku.

Jak już nadmieniono wcześniej, książka skupia się raczej na języku DML, a nie na DDL. Ważne jednak pozostaje zrozumienie sposobów tworzenia tabel i definiowania integralności danych. Nie będziemy zajmować się tu bardzo szczegółowymi tematami, a jedynie przedstawiony zostanie skrótowy opis kwestii zasadniczych.

Zanim przyjrzymy się kodowi tworzenia tabeli, zapamiętajmy, że tabele znajdują się wewnątrz schematów, a schematy wewnątrz baz danych. W przykładach w książce używana jest przykładowa baza danych *TSQL2012* oraz schemat nazwany *dbo*.

Informacje dodatkowe Szczegółowe wskazówki na temat tworzenia przykładowej bazy danych zamieszczone są w Dodatku.

Przytoczone przykłady korzystają ze schematu nazwanego *dbo*, który jest automatycznie tworzony w każdej bazie danych i jest również używany jako schemat domyślny dla użytkowników, którzy nie są wprost powiązani z innym schematem.

Tworzenie tabel

Poniższy kod tworzy tabelę nazwaną *Employees* w schemacie *dbo* w bazie danych *TSQL2012*.

```
USE TSQL2012;

IF OBJECT_ID('dbo.Employees', 'U') IS NOT NULL
  DROP TABLE dbo.Employees;

CREATE TABLE dbo.Employees
(
  Empid INT NOT NULL,
  firstname VARCHAR(30) NOT NULL,
  lastname VARCHAR(30) NOT NULL,
  hiredate DATE NOT NULL,
  mgrid INT NULL,
  ssn VARCHAR(20) NOT NULL,
  salary MONEY NOT NULL
);
```

Polecenie *USE* ustawia kontekst bieżącej bazy danych definiując, że jest to baza *TSQL2012*. Ważne jest umieszczanie polecenia *USE* w skryptach, które tworzą obiekty, by zapewnić, że SQL Server utworzy obiekty w określonej bazie danych. W przypadku implementacji systemu SQL Server w siedzibie instrukcja *USE* może tak naprawdę zmienić kontekst bazy danych z jednej na inną. W produkcie SQL Database nie możemy przełączać się pomiędzy różnymi bazami danych, ale instrukcja *USE* będzie działać prawidłowo, o ile jesteśmy już połączeni z docelową bazą danych. Tak więc nawet w przypadku SQL Database zalecam umieszczanie polecenia *USE*, by zagwarantować, że podczas tworzenia obiektów jesteśmy połączeni z właściwą bazą danych.

Instrukcja *IF* wywołuje funkcję *OBJECT_ID*, by sprawdzić, czy tabela *Employees* już istnieje w bieżącej bazie danych. Funkcja *OBJECT_ID* jako dane wejściowe akceptuje nazwę i typ obiektu. Typ *'U'* reprezentuje tabelę użytkownika. Funkcja ta zwraca identyfikator obiektu wewnętrznego, jeśli istnieje obiekt o określonej nazwie i typie, w przeciwnym razie zwraca wartość *NULL*. Jeśli funkcja zwraca wartość *NULL*, wiemy, że obiekt nie istnieje. W naszym przypadku kod usuwa tabelę, jeśli już istniała, a następnie tworzy nową. Oczywiście możemy wybrać inne działanie; na przykład, jeśli obiekt już istnieje, możemy po prostu go nie tworzyć.

Instrukcja *CREATE TABLE* jest odpowiedzialna za definiowanie tego, co poprzednio nazywaliśmy nagłówkiem relacji. W tym miejscu specyfikujemy nazwę tabeli oraz definicję atrybutów tabeli (w nawiasach), czyli definicje kolumn.

Zwróćmy uwagę na używanie dwuczęściowej nazwy tabeli *dbo.Employees*, zgodnie z wcześniejszymi zaleceniami. Jeśli pominiemy nazwę schematu, system SQL Server przy uruchamianiu kodu założy, że z bazą danych powiązany jest schemat domyślny.

Dla każdego atrybutu specyfikujemy nazwę atrybutu, typ danych i to, czy może przyjmować wartość *NULL* (cecha nazywana *nullability*).

W tabeli *Employees* atrybuty *empid* (identyfikator pracownika) i *mgrid* (identyfikator menedżera) są zdefiniowane jako typ danych *INT* (czterobajtowa liczba całkowita); atrybuty *firstname* (imię), *lastname* (nazwisko) i *ssn* (numer ubezpieczenia) zostały zdefiniowane jako *VARCHAR* (ciąg znaków o zmiennej długości z wyspecyfikowaną maksymalną liczbą obsługiwanych znaków); atrybut *hiredate* (data zatrudnienia) został zdefiniowany jako typ *DATE* (data), a atrybut *salary* (pensja) jako *MONEY*.

Jeśli nie określimy wprost, czy kolumna dopuszcza bądź nie dopuszcza stosowanie znaczników *NULL*, system SQL Server musi oprzeć się na ustawieniach domyślnych. Standard SQL wymaga, by w przypadku braku tego określenia przyjmować możliwość stosowania wartości *NULL* (dopuszczenie znaczników *NULL*), jednak SQL Server zawiera ustawienie, które pozwala zmienić to działanie. Usilnie zalecam, by być precyzyjnym i nie polegać na ustawieniach domyślnych. Ponadto równie mocno zalecam definiowanie kolumn jako *NOT NULL*, chyba że istnieją jasne powody, dla których trzeba obsługiwać znaczniki *NULL*. Jeśli kolumna nie powinna dopuszczać wartości *NULL*, ale nie wymusiliśmy tego za pomocą ograniczenia *NOT NULL*, można się założyć, że znaczniki *NULL* będą się pojawiać. W tabeli *Employees* wszystkie kolumny

zostały zdefiniowane jako *NOT NULL*, za wyjątkiem kolumny *mgrid*. Wartość *NULL* atrybutu *mgrid* reprezentuje fakt, że pracownik nie ma menedżera, jak w przypadku dyrektora organizacji.

Styl kodowania

Trzeba zdawać sobie sprawę z kilku ogólnych uwag związanych ze stylem kodowania, użyciem znaków niewidocznych (spacja, tabulacja, nowy wiesz itp.) oraz średników. Nie znam żadnego formalnego stylu kodowania. Moją radą jest stosowanie stylu, który dla nas i naszych współpracowników jest wygodny w użyciu. Ostatecznie największe znaczenie ma spójność, czytelność i możliwości utrzymywania kodu. Próbowałem pokazywać te aspekty w kodzie przytaczanym w książce.

Język T-SQL pozwala na stosowanie w kodzie znaków niewidocznych praktycznie w dowolny sposób. Znaki te poprawiają czytelność. Na przykład mógłbym zapisać kod z poprzedniego podrozdziału w jednym wierszu. Kod nie będzie jednak tak czytelny, jak w przypadku podzielenia go na wiele wierszy, w których używane są wcięcia tekstu.

Standardem jest praktyka stosowania średnika do zakończenia poleceń, a w istocie jest to wymóg w kilku innych platformach bazodanowych. SQL Server wymaga stosowania średnika jedynie w szczególnych przypadkach – jednak użycie średnika tam, gdzie nie jest on wymagany, nie powoduje problemów. Zalecam więc, by przyswoić sobie zwyczaj kończenia wszystkich poleceń średnikiem. Średnik nie tylko poprawia czytelność, ale w niektórych sytuacjach może nas ustrzec przed kłopotami (w sytuacji, kiedy średnik jest wymagany i nie został wyspecyfikowany, komunikaty o błędzie generowane przez system SQL Server nie zawsze są całkiem jasne).

Uwaga Dokumentacja systemu SQL Server wskazuje, że niekończenie poleceń T-SQL średnikiem jest przestarzałą funkcją. Oznacza to, że w przyszłych wersjach produktu znajdzie się funkcja wymuszania używania średnika. Jest to jeszcze jeden powód, dla którego warto przyzwyczaić się do średników kończących polecenia, nawet jeśli aktualnie nie są wymagane.

Definiowanie integralności danych

Jak już wspomniano wcześniej, jedną z wielkich zalet modelu relacyjnego jest to, że integralność danych jest jego częścią. Integralność danych wymuszana jako część modelu, a mianowicie jako część definicji tabeli, jest nazywana deklaratywną

integralnością danych. Integralność danych wymuszana za pomocą kodu, jak na przykład przy użyciu procedur składowanych czy wyzwalaczy, jest nazywana proceduralną integralnością danych.

Typ danych, możliwość wyboru obsługi znaczników *NULL* dla atrybutów i nawet sam model danych są przykładami ograniczeń deklaratywnej integralności danych. W tym podrozdziale opiszemy inne przykłady deklaratywnych ograniczeń: klucza głównego, unikatowości, klucza obcego, sprawdzania (check) i domyślnych. Tego typu ograniczenia możemy definiować podczas tworzenia tabeli jako część instrukcji *CREATE TABLE* lub możemy je definiować dla wszystkich już utworzonych tabel przy użyciu instrukcji *ALTER TABLE*. Wszystkie typy ograniczeń, za wyjątkiem domyślnych, mogą być definiowane jako *ograniczenia złożone* – to znaczy bazujące na więcej niż jednym atrybucie.

Ograniczenia klucza głównego

Ograniczenie klucza głównego wymusza unikatowość wierszy, a oprócz tego zabrania stosowania znaczników *NULL* w ograniczających atrybutach. Każdy unikatowy zbiór wartości w atrybutach ograniczeń może w tabeli wystąpić tylko raz – inaczej mówiąc, tylko w jednym wierszu. Próba zdefiniowania ograniczenia klucza głównego dla kolumny, która dopuszcza znaczniki *NULL*, zostanie odrzucona przez system RDBMS. Każda tabela może mieć tylko jeden klucz główny.

Poniżej zamieszczono przykład definiowania ograniczenia klucza główny dla atrybutu *empid* w utworzonej wcześniej tabeli *Employees*.

```
ALTER TABLE dbo.Employees
  ADD CONSTRAINT PK_Employees
  PRIMARY KEY(empid);
```

Dzięki ustanowieniu tego klucza głównego mamy pewność, że wszystkie wartości *empid* będą unikatowe i znane. Próba wstawienia lub zaktualizowania wiersza w taki sposób, że naruszone zostaje ograniczenie, zostanie odrzucona przez system RDBMS i spowoduje wygenerowanie komunikatu o błędzie.

W celu wymuszenia unikatowości logicznego ograniczenia klucza głównego, system SQL Server utworzy w tle unikatowy indeks. Jest on mechanizmem fizycznym używanym przez SQL Server do wymuszania jednoznaczności. Indeksy (niekoniecznie unikatowe) są również używane do przyspieszania kwerend poprzez eliminowanie niepotrzebnego przeszukiwania całej tabeli (działanie podobne do indeksu w książce).

Ograniczenia Unique

Ograniczenia Unique wymuszają unikatowość wierszy, co pozwala zaimplementować w naszej bazie danych koncepcję zmiennych kluczy modelu relacyjnego. Inaczej niż w przypadku kluczy głównych, wewnątrz tej samej tabeli możemy definiować wiele ograniczeń unikatowości. Ponadto ograniczenie limitowane tylko do kolumn