helion.pl

Idź do

- Spis treści
- Przykładowy rozdział

Katalog książek

- Katalog online
- Zamów drukowany katalog

Twój koszyk

- Dodaj do koszyka

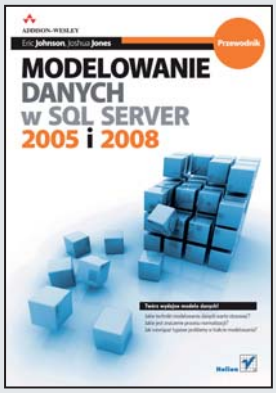
Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

Czytelnia

- Fragmenty książek online

Modelowanie danych w SQL Server 2005 i 2008. Przewodnik



Autor: Eric Johnson, Joshua Jones
Tłumaczenie: Wojciech Moch
ISBN: 978-83-246-2090-6
Tytuł oryginału: A Developer's Guide to Data Modelling for SQL Server. Covering SQL Server 2005 and 2008
Format: 168x237, stron: 280

Twórz wydajne modele danych!

- Jakie techniki modelowania danych warto stosować?
- Jakie jest znaczenie procesu normalizacji?
- Jak rozwiązać typowe problemy w trakcie modelowania?

Model danych jest niezwykle istotnym etapem tworzenia systemu informatycznego, ponieważ rzutuje on bezpośrednio na wydajność rozwiązania oraz komfort pracy programisty. Warto zatem poznać najlepsze techniki modelowania danych i wszystkie związane z nimi procesy.

Dzięki tej książce zrozumiesz podstawowe techniki modelowania danych oraz dowiesz się, jak gromadzić wymagania dotyczące modelu. Ponadto zapoznasz się z elementami wykorzystywanymi w logicznych i fizycznych modelach danych. Czwarty – niezwykle istotny – rozdział wprowadzi Cię w tematykę normalizacji modelu, dzięki czemu zrozumiesz, jak istotny to proces! W trakcie lektury kolejnych rozdziałów nauczysz się rozwiązywać typowe problemy, występujące w trakcie modelowania, oraz uświadomisz sobie, jak istotną rolę pełnią w nim indeksy. Pojawiające się tu przykłady dotyczą bazy danych SQL Server firmy Microsoft, niewątpliwie jednak książka ta przyda się również osobom związanym z innymi platformami bazodanowymi.

- Techniki modelowania danych
- Elementy wykorzystywane w logicznych modelach danych
- Elementy wykorzystywane w fizycznych modelach danych
- Proces normalizacji modelu danych
- Sposoby efektywnego gromadzenia wymagań
- Interpretacja oraz dokumentacja wymagań
- Proces tworzenia modelu logicznego
- Sposób wykorzystania SQL Server w celu stworzenia modelu fizycznego
- Zastosowanie i znaczenie indeksów
- Przygotowanie warstwy abstrakcji w SQL Server
- Rozwiązywanie typowych problemów w trakcie procesu modelowania

Dowiedz się wszystkiego o modelowaniu danych i Twórz wydajne rozwiązania!

Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2008

SPIS TREŚCI

Wstęp **13**

O autorach **15**

Część I Teoria modelowania danych **17**

Rozdział 1. Przegląd technik modelowania danych **19**

Bazy danych 20
Systemy zarządzania relacyjnymi bazami danych *21*
Dlaczego dobrze zaprojektowany model danych jest tak ważny 22
Spójność danych *22*
Skalowalność *23*
Spełnianie wymagań biznesowych *25*
Łatwe odczytywanie danych *26*
Poprawianie wydajności *28*
Proces modelowania danych 29
Teoria modelowania danych *29*
Wymagania biznesowe *31*
Budowanie modelu logicznego *33*
Budowanie modelu fizycznego *34*
Podsumowanie 35

Rozdział 2. Elementy wykorzystane w logicznych modelach danych **37**

Encje 37
Atrybuty 38
Typy danych *39*
Klucze główne i obce *43*
Domeny *44*
Atrybuty z pojedynczą wartością i z wieloma wartościami *45*
Spójność referencji 46
Relacje 47
Typy relacji *48*
Opcje relacji *52*
Liczność *53*

Używanie podtypów i typów nadrzędnych ... 54
Definicje podtypów i typów nadrzędnych ... 54
Kiedy używać klastrów podtypów ... 56
Podsumowanie ... 56

Rozdział 3. Fizyczne elementy modeli danych ... 57

Fizyczne przechowywanie danych ... 57
Tabele ... 57
Widoki ... 59
Typy danych ... 61
Spójność referencji ... 70
Klucze główne ... 70
Klucze obce ... 74
Ograniczenia ... 76
Implementowanie spójności referencji ... 78
Programowanie ... 81
Procedury składowane ... 81
Funkcje użytkownika ... 82
Wyzwalacze ... 83
Integracja z CLR ... 85
Implementowanie typów nadrzędnych i podtypów ... 85
Tabela typu nadrzędnego ... 86
Tabele podtypów ... 87
Tabele typu nadrzędnego i podtypów ... 87
Typy nadrzędne i podtypy — podsumowanie ... 88
Podsumowanie ... 88

Rozdział 4. Normalizowanie modelu danych ... 91

Czym jest normalizacja? ... 91
Postaci normalne ... 91
Określanie postaci normalnych ... 99
Denormalizacja ... 100
Podsumowanie ... 102

Część II Wymagania biznesowe ... 105

Rozdział 5. Gromadzenie wymagań ... 107

Przegląd zagadnień związanych ze zbieraniem wymagań ... 108
Zbieranie wymagań krok po kroku ... 108
Prowadzenie wywiadów ... 108
Obserwacje ... 111
Istniejące procesy i systemy ... 112
Przypadki użycia ... 114

Potrzeby biznesowe ... 120
Szukanie złotego środka między ograniczeniami technicznymi i potrzebami biznesowymi ... 121
Zbieranie danych użytkowych ... 121
Odczyty a zapisy ... 121
Wymagania dotyczące przechowywania danych ... 122
Wymagania transakcyjne ... 123
Podsumowanie ... 124

Rozdział 6. Interpretowanie wymagań ... 125

Mountain View Music ... 125
Analiza danych na temat wymagań ... 127
Identyfikowanie użytecznych informacji ... 127
Identyfikowanie informacji nadmiarowych ... 128
Definiowanie wymagań modelu ... 129
Interpretowanie wyników wywiadów ... 129
Interpretacja diagramów przepływu ... 134
Interpretowanie istniejących systemów ... 137
Interpretowanie przypadków użycia ... 139
Określanie atrybutów ... 141
Określanie reguł biznesowych ... 143
Definiowanie reguł biznesowych ... 145
Liczność ... 146
Wymagania wobec danych ... 146
Dokumentowanie wymagań ... 147
Lista encji ... 147
Lista atrybutów ... 147
Lista relacji ... 148
Lista reguł biznesowych ... 148
Spojrzenie w przyszłość — recenzja ... 148
Dokumentacja projektowa ... 148
Podsumowanie ... 150

Część III Tworzenie modelu logicznego ... 151

Rozdział 7. Tworzenie modelu logicznego ... 153

Tworzenie diagramów modelu danych ... 153
Sugestie dotyczące nazewnictwa ... 153
Standardy notacji ... 156
Narzędzia do modelowania ... 159
Wykorzystywanie wymagań do budowania modelu ... 160
Lista encji ... 160
Lista atrybutów ... 164
Dokumentacja relacji ... 165
Reguły biznesowe ... 166

Budowanie modelu 167
Klucze główne 168
Relacje 169
Domeny 170
Atrybuty 170
Podsumowanie 172

Rozdział 8. Typowe problemy przy modelowaniu danych 173

Problemy z encjami 173
Zbyt mało encji 173
Zbyt wiele encji 176
Problemy z atrybutami 177
Jeden atrybut zawierający różne dane 177
Niewłaściwe typy danych 179
Problemy z relacjami 183
Relacje typu jeden-do-jednego 183
Relacje typu wiele-do-wielu 184
Podsumowanie 185

Część IV Tworzenie modelu fizycznego 187

Rozdział 9. Tworzenie modelu fizycznego za pomocą serwera SQL Server 189

Nazewnictwo obiektów 189
Ogólne reguły nazewnictwa 191
Nazywanie tabel 194
Nazywanie kolumn 195
Nazwy widoków 195
Nazywanie procedur składowanych 195
Nazywanie funkcji użytkownika 196
Nazywanie wyzwalaczy 196
Nazywanie indeksów 196
Nazywanie typów danych użytkownika 197
Nazywanie kluczy głównych i kluczy obcych 197
Nazywanie ograniczeń 197
Tworzenie modelu fizycznego 198
Modelowanie tabel na podstawie encji 198
Używanie relacji do modelowania kluczy 208
Modelowanie kolumn za pomocą atrybutów 209
Implementowanie reguł biznesowych w modelu fizycznym 209
Implementowanie reguł biznesowych za pomocą ograniczeń 210
Implementowanie reguł biznesowych za pomocą wyzwalaczy 212
Implementowanie zaawansowanej liczności 214
Podsumowanie 216

Rozdział 10. Kilka słów na temat indeksów 217

Przegląd indeksów 217
Czym są indeksy? *218*
Rodzaje *220*
Wymagania dotyczące korzystania z bazy danych 226
Odczyty i zapisy *226*
Dane transakcji *228*
Określanie właściwych indeksów 228
Przeglądanie wzorów dostępu do danych *228*
Równoważenie indeksów *229*
Indeksy pokrywające *230*
Statystyki indeksów *230*
Rozważania na temat obsługi indeksów *231*
Implementowanie indeksów w serwerze SQL Server 231
Konwencje nazewnictwa *231*
Tworzenie indeksów *232*
Grupy plików *233*
Konfigurowanie konserwacji indeksów *233*
Podsumowanie 235

Rozdział 11. Tworzenie warstwy abstrakcji w serwerze SQL Server 237

Czym jest warstwa abstrakcji? 237
Po co używać warstwy abstrakcji? 238
Bezpieczeństwo *238*
Elastyczność i możliwość rozbudowy *240*
Związek warstwy abstrakcji z logicznym modelem danych 241
Związek warstwy abstrakcji z programowaniem zorientowanym obiektowo 241
Implementowanie warstwy abstrakcji 243
Widoki *243*
Procedury składowane *245*
Inne składniki warstwy abstrakcji *248*
Podsumowanie 248

Część V Dodatki 251

Dodatek A Przykładowy model logiczny 253

Dodatek B Przykładowy model fizyczny 257

Dodatek C Zarezerwowane słowa serwera SQL Server 2008 261

Dodatek D Zalecane standardy nazewnictwa 263

Skorowidz 265

ROZDZIAŁ 2.

ELEMENTY WYKORZYSTANE W LOGICZNYCH MODELACH DANYCH

Proszę sobie wyobrazić, że ktoś poprosił nas o wybudowanie domu. Jednym z pierwszych pytań, jakie w takiej sytuacji należałoby sobie postawić, jest: „Czy mam wszystkie potrzebne narzędzia i materiały?" Aby na takie pytanie odpowiedzieć, potrzebujemy projektu tego domu. To właśnie z projektu będziemy w stanie wywnioskować, jakie narzędzia i materiały będą niezbędne. Oznacza to, że na początek musimy sobie przygotować projekt. Jeżeli wcześniej tego nie robiliśmy, to najprawdopodobniej będziemy musieli dowiedzieć się, jak należy go wykonać.

Przygotowany przez nas logiczny model danych, podobnie jak projekt domu, stanie się podstawą wszystkich prac nad fizyczną bazą danych. Oprócz tego logiczny model danych jest swego rodzaju wysokopoziomowym widokiem na bazę danych, który można zaprezentować wszystkim stronom biorącym udział w projekcie. Z tych właśnie powodów logiczny model danych może być całkowicie oderwany od technicznych szczegółów związanych z konkretnym systemem zarządzania bazami danych. Informacje zawarte w modelu logicznym definiują jedynie sposób, w jaki baza danych zbudowana na podstawie tego modelu będzie spełniała biznesowe wymagania klienta. Zanim jednak przystąpimy do konstruowania modelu logicznego, koniecznie musimy poznać wszystkie narzędzia, jakich będziemy potrzebowali.

W tym rozdziale omówimy obiekty i koncepcje powiązane z procesem tworzenia logicznego modelu danych. Obiekty te wykorzystamy ponownie w rozdziale 7., w którym zaczniemy tworzyć model danych dla firmy Mountain View Music. Na razie opiszemy temat encji oraz atrybutów i zobaczymy, jak można tworzyć relacje między nimi.

Encje

Encje reprezentują logiczne grupy danych, dlatego w modelu logicznym są najważniejszym elementem, który definiuje sposób zapisu danych w bazie. Typowymi przykładami encji są klienci, zamówienia lub produkty. Każda encja, która powinna reprezentować pojedynczy typ informacji, składa się z całej kolekcji egzemplarzy tej encji. **Egzemplarz** (ang. *instance*) encji jest bardzo podobny do rekordu bazy danych. W teorii modelowania danych pojęcia *egzemplarza*, *rekordu* lub *wiersza*

są używane zamiennie. W tej książce egzemplarz będzie pojawiał się wyłącznie w encjach, natomiast rekordy lub wiersze będą elementami fizycznych tabel lub widoków.

Często możemy ulegać pokusie traktowania poszczególnych encji jako tabel (w końcu między tabelami a encjami istnieją zwykle związki typu jeden do jednego), trzeba jednak pamiętać, że logiczna encja może być reprezentowana przez wiele tabel, a wiele encji może zostać zgromadzonych w ramach jednej tabeli. Zadaniem encji jest identyfikacja wszystkich informacji, które będą zapisywane w bazie danych.

Jednym ze sposobów na określenie, co można zakwalifikować jako encję, jest myślenie o encjach jak o rzeczownikach. Encje zwykle są obiektami, które można opisać za pomocą rzeczownika: zamówieniami, samochodami, trąbkami albo telefonami, czyli przedmiotami znanymi nam na co dzień. Właściwe zdefiniowanie encji na potrzeby modelu jest niezwykle istotne, dlatego to zadanie stanowi zawsze jedną z większych części procesu projektowania.

Definiując encje, powinniśmy przede wszystkim zajmować się ich przeznaczeniem, a dopiero w dalszej kolejności zastanawiać się nad ich atrybutami i innymi szczegółami (atrybuty zostaną opisane w następnym podrozdziale). Wywiady prowadzone z pracownikami firmy i innymi osobami odpowiedzialnymi za jej funkcjonowanie w ramach procesu gromadzenia wymagań biznesowych (będzie o tym mowa w rozdziale 5.) pozwolą nam określić najczęściej używane w firmie rzeczowniki, a co za tym idzie, najistotniejsze encje. Po rozpoczęciu projektowania modelu to właśnie z tych notatek będziemy korzystać do określania wszystkich niezbędnych encji. Musimy przy tym bardzo ostrożnie przeglądać i filtrować sporządzone wcześniej notatki, tak żeby wykorzystać tylko te informacje, które są rzeczywiście istotne w aktualnym projekcie.

Atrybuty

Każda encja jest opisywana przez pewne szczegółowe informacje, które są właśnie jej atrybutami. Załóżmy, że musimy utworzyć encję przechowującą wszystkie informacje opisujące kapelusze. Otrzyma ona nazwę *Kapelusze*, a następnie będziemy mogli określić, jakie informacje, czyli atrybuty, musimy w niej zachować: kolor, nazwę producenta, styl, materiał itp. W czasie konstruowania modelu definiujemy kolejkę atrybutów, które będą przechowywały te dane w ramach encji. Definicja atrybutu składa się z jego nazwy, opisu, celu i typu danych (typy danych zostaną omówione w następnym podrozdziale).

Należy się przy tym wystrzegać dodawania do encji atrybutów, które tak naprawdę powinny być częścią innej encji. Typowy błąd polega na bezpośrednim przekształcaniu danych z fizycznej dokumentacji, takiej jak wydrukowane arkusze kalkulacyjne lub podręczniki, w encje i atrybuty modelu logicznego. Na przykład informacje na temat klienta są często fizycznie łączone z informacjami na temat zamówienia. Można z tego wysnuć błędne założenie, że informacje dotyczące klienta, takie jak adres lub numer telefonu, są atrybutami zamówienia. W rzeczywistości klient i zamówienie są całkowicie niezależnymi encjami. Zapisanie atrybutów klienta w ramach

encji zamówienia spowoduje niepotrzebne skomplikowanie procesu odczytywania danych i może doprowadzić do powstania projektu, który nie będzie poddawał się skalowaniu.

Chcąc prawidłowo modelować atrybuty encji, musimy poznać kilka najważniejszych pojęć: typy danych, klucze, domeny i wartości. W kolejnych podrozdziałach nieco dokładniej je omówimy.

Typy danych

Oprócz informacji opisowych w definicji atrybutu trzeba podać jeszcze **typ danych** (ang. *data type*). Jak sama nazwa wskazuje, definiuje on typ informacji, jaka będzie zapisywana w atrybucie. Na przykład atrybut może być ciągiem znaków, liczbą albo wartością logiczną typu prawda-fałsz.

W modelach logicznych określenie typów danych atrybutów nie jest absolutnie wymagane. Ze względu na to, że typ danych jest określeniem fizycznego sposobu zapisu danych, określenie typu danych atrybutu następuje w czasie tworzenia modelu fizycznego. Trzeba jednak pamiętać, że określając typ danych już w trakcie przygotowywania modelu logicznego, możemy uzyskać kilka ważnych korzyści:

- Zespół tworzący model fizyczny będzie dysponował już pewnym przewodnikiem i nie będzie musiał wyszukiwać informacji w zbiorze wymagań (czasami może się to okazać działaniem wielokrotnym).
- Będziemy mogli wykryć nieścisłości powstające pomiędzy encjami przechowującymi ten sam typ danych (na przykład numer telefonu), i to jeszcze przed powstaniem modelu fizycznego.
- W ramach ułatwienia procesu budowania fizycznej bazy danych możemy zdefiniować typy danych ściśle związane z używanym systemem relacyjnych baz danych. Można to zrobić tylko wtedy, gdy docelowy system zarządzania bazą danych będzie znany jeszcze przed rozpoczęciem prac nad modelem.

Większość dostępnych programów do modelowania danych pozwala na wybranie spośród typów danych stosowanego systemu zarządzania bazą danych. Skoro mamy pracować z serwerem Microsoft SQL Server, możemy od razu korzystać z typów danych obsługiwanych przez ten serwer. Przyjrzyjmy się teraz różnym typom danych stosowanym w modelach logicznych.

Alfanumeryczne

Wszystkie modele danych zawierają dane **alfanumeryczne**, czyli dowolne dane w postaci ciągu znaków, niezależnie od tego, czy zawierają one znaki alfabetu, czy też cyfry (pod warunkiem że te nie biorą udziału w obliczeniach matematycznych). Przykładami tego typu danych mogą być nazwy, adresy albo numery telefonów. Konkretne typy danych używane do przechowywania informacji alfanumerycznych to *char*, *nchar*, *varchar* i *nvarchar*. Jak można wywnioskować z nazw tych typów, są one przeznaczone do przechowywania znaków, takich jak litery, cyfry i znaki specjalne.

Wszystkie te typy danych wymagają określenia ich długości. Mówiąc ogólnie, długość definiuje całkowitą liczbę znaków, jaka może znaleźć się w danym atrybucie. Jeżeli mamy pewność, że dany atrybut będzie przechowywać tylko dwuliterowe skróty nazw państw, to możemy zdefiniować go jako *char(2)*. W ten sposób definiujemy atrybut jako alfanumeryczne pole mieszczące w sobie dokładnie dwa znaki. Atrybuty typu *char* przechowują dokładnie tyle znaków, ile określa ich definicja — nie mniej i nie więcej, niezależnie od tego, ile znaków faktycznie do nich wprowadzono.

Każdy zapewne zauważył już, że istnieją aż cztery rodzaje znakowego typu danych: dwa z przedrostkiem *var* i dwa z przedrostkiem *n* (jeden z tych typów zawiera oba te przedrostki). Pola o zmiennej długości są definiowane jako zawierające nie więcej znaków, niż zostało to podane w definicji pola. Aby zobrazować różnicę między typami *char* i *varchar*, można podać, że definicja *char(10)* tworzy pole o długości dziesięciu znaków, nawet jeżeli konkretny egzemplarz tego pola będzie miał długość sześciu znaków. W takim przypadku pozostałe cztery znaki zostaną automatycznie dopisane. Jeżeli ten atrybut zostanie zdefiniowany jako *varchar(10)*, to w podobnej sytuacji zapisanych zostanie tylko sześć znaków.

Przedrostek *n* oznacza, że znaki będą zapisywane w formacie **Unicode**. Jest to międzynarodowa, niezależna od platformy specyfikacja przeznaczona do zapisywania znaków. Używanie Unicode'u pozwala na budowanie systemów pracujących ze znakami różnych języków. Systemy takie mogą bez żadnych problemów wymieniać między sobą dane tekstowe, właśnie dzięki zastosowaniu wspólnej metody zapisu znaków. Jeżeli tylko musimy zapisać znaki wykraczające poza standardowy zestaw ASCII konieczne jest zastosowanie Unicode'u.

Podstawową różnicą między systemami korzystającymi z Unicode'u a systemami używającymi tylko znaków ASCII jest to, że każdy znak Unicode zajmuje dwa bajty, podczas gdy znaki ASCII zajmują tylko jeden znak (jeżeli są zapisywane dane o zmiennej długości, może się okazać, że jest potrzebny więcej niż jeden bajt). Problem z systemami zapisującymi znaki w jednym bajcie polega na tym, że nie są one w stanie skutecznie zapisać pewnych znaków, takich jak japońskie znaki Kanji albo koreańskie znaki Hangul. W tym przypadku trzeba rozważyć, czy ważniejsza jest wygoda zapisywania danych, czy może wydajność pracy. Zagadnienie to będziemy omawiać dokładniej w rozdziale 3. Na razie proszę pamiętać, że jeżeli chcemy przechowywać w bazie pewne typy tekstów, niezbędne może okazać się użycie znaków w formacie Unicode.

Numeryczne

Dane **numeryczne** to dowolne dane, które muszą zostać zapisane w postaci liczbowej. Na wszystkich typach danych numerycznych możemy wykonywać obliczenia matematyczne. Wśród najogólniejszych typów danych numerycznych znajdziemy typy całkowite (*integer*), dziesiętne (*decimal*), walutowe (*money*), zmiennoprzecinkowe (*float*) i rzeczywiste (*real*).

Typy **całkowite** zawsze są zapisywane w postaci liczby całkowitej. Ten typ danych pozwala na zapisywanie liczb dodatnich i ujemnych, a dodatkowo istnieje on w kilku wielkościach, z których każda pozwala na przechowywanie określonego

zakresu wartości. Typy **dziesiętne** są liczbami o z góry zdefiniowanej wielkości i dokładności. **Wielkość** typu oznacza w tym przypadku ogólną liczbę cyfr, jakie można zapisać w danym polu, natomiast **dokładność** określa, ile z tych cyfr zostanie zapisanych po przecinku. Typ **walutowy** jest przeznaczony do zapisywania wartości kwot, a jego dokładność jest uzależniona od stosowanego właśnie systemu zarządzania bazami danych. Typ **zmiennoprzecinkowy** opisuje liczby zmiennoprzecinkowe, a zatem liczby zapisywane z pewnym przybliżeniem. Wartości tego typu są najczęściej zapisywane w notacji naukowej. Dodatkowo możliwe jest zdefiniowanie liczby bitów, w której mają być przechowywane liczby. Typy **rzeczywiste** są właściwie identyczne z typami zmiennoprzecinkowymi, ale typy zmiennoprzecinkowe umożliwiają przechowywanie znacznie większych liczb.

Podobnie jak w przypadku typów danych alfanumerycznych szczegółowe informacje na temat fizycznego zapisywania danych tych typów zostały przedstawione w rozdziale 3.

Logiczne

Typy **logiczne** (boolean) umożliwiają przechowywanie tylko trzech wartości: prawdy (`TRUE`), fałszu (`FALSE`) lub zera (`NULL`). Dane przechowywane w polach tego typu są logiczne, fizycznym typem danych jest jednak jeden bit. Może on przechowywać wartość 1, 0 lub `NULL`, które można przetłumaczyć jako prawdę, fałsz i nic. Logiczne typy danych są używane do przechowywania wyników różnych wyrażeń logicznych. Często stosowane są też jako przełączniki lub flagi, na przykład wskazujące, że dany pojazd jest aktualnie niesprawny.

BLOB i CLOB

Nie wszystkie dane przechowywane w bazie danych muszą mieć postać czytelną dla użytkownika. Na przykład baza danych zbierająca informacje sklepu internetowego oprócz danych opisujących poszczególne produkty może również zawierać ich obrazy. Binarne dane składające się na informacje o obrazie nie nadają się do czytania w postaci ciągu znaków, ale mogą zostać zapisane w bazie danych, tak żeby mogła z nich skorzystać aplikacja. Tego rodzaju dane są zwykle nazywane **wielkimi obiektami binarnymi** (ang. *binary large object* — BLOB).

Takie informacje w serwerze SQL Server najczęściej są zapisywane za pomocą następujących typów danych: *binary*, *varbinary* i *image*. Podobnie jak w przypadku typów znakowych przedrostek *var* oznacza, że dany atrybut może przechowywać wartości o zmiennej długości. W związku z tym typ **binary** definiuje atrybut o stałej wielkości, przechowujący dane binarne, a typ **varbinary** określa tylko *maksymalną* wielkość atrybutu przechowującego dane binarne. Z kolei typ **image** określa, że atrybut zawiera dane binarne o zmiennej wielkości, podobne do typu *varbinary*. Typ ten daje jednak znacznie większe możliwości przechowywania danych.

Dane znakowe również mogą się pojawiać w formach o wiele dłuższych, niż przewidują to typowe typy danych alfanumerycznych, o których mówiliśmy wcześniej. Co zrobić, jeżeli w danym atrybucie musimy zapisać tekst dowolnej wielkości,

na przykład cały życiorys? W tej sytuacji z pomocą przychodzą nam dwa typy **wielkich obiektów znakowych** (ang. *character large object* — CLOB), czyli *text* i *ntext*. Te dwa typy danych są przeznaczone do przechowywania dużych ilości tekstu w postaci jednego pola. Podobnie jak było to w przypadku pozostałych typów znakowych, przedrostek *n* oznacza, że tekst w danym atrybucie jest zapisywany w formacie Unicode. Z tych typów danych należy korzystać wtedy, gdy w ramach jednego atrybutu encji musimy zapisać naprawdę duże ilości danych alfanumerycznych.

Data i czas

Niemal każdy istniejący model danych wymaga zastosowania encji zawierających atrybuty opisujące daty lub czas. Pozwalają one na określenie momentu, w którym wprowadzono zmiany do zamówienia, daty zatrudnienia nowego pracownika albo zdefiniowanie czasu dostawy towarów. Każdy system zarządzania relacyjnymi bazami danych ma swoją własną implementację typów danych pozwalających na zapisywanie daty i czasu. W przypadku serwera SQL Server 2008 mamy do dyspozycji aż sześć typów danych tego rodzaju. Jest to duży postęp w stosunku do poprzednich wersji tego serwera, które oferowały jedynie dwa typy: *datetime* i *smalldatetime*. Każdy z tych typów danych przechowuje informacje na temat daty lub czasu. Różnice wynikają jedynie z dokładności zapisanej daty i zakresu dostępnych wartości.

Przyjrzyjmy się starszym typom. Typ **datetime** przechowuje dane dotyczące daty i czasu z dokładnością do jednej milisekundy. Załóżmy, że do tabeli zawierającej kolumnę typu *datetime* wpisujemy datę 12.01.2008 oraz godzinę 18.00.

W bazie danych zostanie zapisana wartość:

```
12/01/2008 18:00:00.000
```

Typ *smalldatetime* zapisałby tę samą wartość w postaci:

```
12/01/2008 18:00
```

Typ *datetime* może przechowywać dowolną datę z zakresu od 1 stycznia 1753 roku do 31 grudnia 9999 roku, a typ *smalldatetime* pozwala na zapisanie daty z zakresu od 1 stycznia 1900 roku do 6 czerwca 2079 roku. Powodem wybrania tych właśnie dat, trzeba przyznać, że dość dziwnych, są wymagania związane z metodami zapisu na poziomie dysku twardego oraz z metodami manipulacji na datach w serwerze SQL Server.

Jak już wcześniej wspominaliśmy, SQL Server 2008 wprowadza cztery nowe typy danych związanych z datą i czasem: *date*, *time*, *datetime2* i *datetimeoffset*. Nowe typy danych przechowują informacje o dacie i czasie w sposób dużo elastyczniejszy niż poprzednie. Najprostsze są oczywiście typy **date** i **time**, ponieważ pozwalają na zapisanie jedynie daty lub jedynie czasu. Typ **datetime2** otrzymał niezbyt dobrą nazwę, ale poza tym jest bardzo podobny do typu *datetime*. Jedyna różnica polega na tym, że nowy typ danych pozwala na zdefiniowanie dokładności zapisywania ułamków sekund od zera do siedmiu miejsc po przecinku. Z kolei typ danych **datetimeoffset** różni się od typu *datetime* możliwością zapisania oprócz daty i czasu

pewnej wartości przesunięcia. Nie musi być ono związane z żadną strefą czasową ani nawet z czasem Greenwich. Musimy zatem pamiętać, względem której strefy czasowej należy je liczyć.

Omówiliśmy już sporo szczegółów, ale ponownie przypomnimy, że jeszcze dłuższy wywód na temat metod przechowywania informacji w ramach poszczególnych typów danych znajduje się w rozdziale 3.

W czasie projektowania modelu logicznego możemy ulec pokusie nadawania atrybutom różnych typów danych. Takie praktyki mogą jednak powodować wiele problemów podczas dalszych prac. Większość programów do modelowania danych jest w stanie wygenerować model fizyczny na podstawie modelu logicznego, a zatem wybranie niewłaściwych typów danych w modelu logicznym może doprowadzić do nieprawidłowości w modelu fizycznym, szczególnie gdy pracuje nad nim kilka osób. Należy często zaglądać do zebranych wymagań biznesowych, tak żeby uzyskać pewność, że faktycznie definiujemy atrybuty zgodne z danymi używanymi w firmie. Ułatwi nam to również rozmowy na temat modelu danych prowadzone z pracownikami, którzy nie zajmują się techniczną stroną projektu.

Klucze główne i obce

Klucz główny (ang. *Primary Key* — PK) jest atrybutem lub grupą atrybutów, która jednoznacznie identyfikuje każdy egzemplarz danej encji. Zawsze musi zawierać dane i nigdy nie może być pusty. Jako przykłady kluczy głównych można podać numery identyfikacyjne pracowników albo numery ISBN książek. W czasie tworzenia modelu musimy pamiętać o tym, że niemal każda zawarta w nim encja musi mieć klucz główny, nawet jeżeli musimy go utworzyć sztucznie za pomocą jakiejś liczby.

Jeżeli dane nie mają przypisanego klucza głównego, to często niezbędne jest dodanie do tabeli kolumny, która będzie służyła jako taki właśnie klucz. Tego rodzaju klucze nazywane są **kluczami zastępczymi**. Co prawda, taka praktyka jest już bardziej zbliżona do fizycznego projektowania bazy danych, ale modelowanie klucza zastępczego pozwala na tworzenie relacji na podstawie kluczy głównych. Takie klucze są zwykle liczbami, których wartość jest zwiększana dla każdego wiersza dodawanego do tabeli. W przypadku serwera SQL Server liczby te nazywane są **tożsamościami** (ang. *Identity*).

Inna reguła dobrego modelowania mówi, że nie należy używać opisowych atrybutów jako kluczy głównych. Na przykład w tabelach opisujących pracowników często jako klucz główny jest używany numer PESEL. Nie jest to dobre rozwiązanie z co najmniej kilku powodów. Po pierwsze, chodzi o względy bezpieczeństwa i zachowanie prywatności. Wiele kradzieży tożsamości dochodzi do skutku właśnie dlatego, że złodziej ma dostęp do numeru PESEL ofiary. Po drugie, numery PESEL powinny być unikatowe, ale nie można wykluczyć sytuacji, w której zostaną zmienione.

Po trzecie, może się okazać, że przyjdzie nam zarejestrować pracowników zagranicznych, którzy nie mają numeru PESEL. W takiej sytuacji kuszące może być wygenerowanie po prostu sztucznego numeru, ale co będzie, jeżeli zagraniczny pracownik otrzyma obywatelstwo, a razem z nim własnym numer PESEL? W takiej

sytuacji wiersze encji zależnych mogą być połączone z prawdziwym lub nieprawdziwym numerem PESEL, przez co odczytywanie danych bardzo się skomplikuje, a co gorsza, w bazie mogą pojawić się osierocone wiersze.

Mówiąc ogólnie, klucz główny powinien:

- być całkowicie unikatowy i *absolutnie* niezmienny,
- składać się z atrybutów, które nigdy nie będą puste,
- korzystać z nieznaczących danych, jeżeli tylko jest to możliwe.

Z kluczem głównym jest ściśle powiązany **klucz obcy** (ang. *Foreign Key* — FK). Składają się na niego atrybuty dane encji, które bazują na kluczu (najczęściej na kluczu głównym) innej encji. Przyjrzyjmy się przykładowej encji *Pracownik* oraz całkowicie nowej encji *Pojazd*. Jeżeli chcielibyśmy wiedzieć, któremu pracownikowi przypisano dany samochód, to musielibyśmy połączyć te dwie encje relacją. W takiej sytuacji w encji *Pojazd* musi istnieć klucz obcy wskazujący na klucz główny w tabeli *Pracownik*. Najprościej można to zrealizować, dodając do encji *Pojazd* atrybut przechowujący numer pracownika, któremu przypisano dany samochód. Atrybuty w encji referującej mogą być atrybutami kluczowymi, ale również niekluczowymi. Oznacza to, że klucz obcy w encji referującej może składać się z tych samych atrybutów co jej klucz główny, ale może zostać zbudowany z całkowicie innego zestawu atrybutów. Taka kombinacja klucza głównego i klucza obcego pozwala na zachowanie spójności w logicznych relacjach między encjami.

Domeny

Zaczynając budować model danych, z pewnością zauważymy, że w kontekście danych, nad którymi pracujemy, pewne encje zawierają podobne atrybuty. Bardzo często ze względu na spójność informacji część danych związana z funkcjonowaniem aplikacji lub firmy musi mieć taką samą postać we wszystkich encjach. Status, adres, numer telefonu i adres e-mail są przykładami atrybutów, które najprawdopodobniej będą musiały mieć identyczną postać w wielu encjach. Zamiast żmudnie tworzyć i ewentualnie modyfikować te atrybuty w poszczególnych encjach, możemy skorzystać z domen.

Domena jest definicją atrybutu, która jest przechowywana w ramach modelu logicznego, ale poza jakąkolwiek encją. Jeżeli w encji zostanie wykorzystany atrybut będący częścią domeny, to domena ta zostanie dołączona do encji. Trzeba jednak pamiętać, że model danych nie przedstawia widocznej informacji o tym, że dany atrybut jest częścią pewnej domeny. Większość narzędzi do modelowania udostępnia jednak specjalną sekcję lub dokument, taki jak **słownik danych**, w którym są przechowywane informacje o domenach. Jeżeli jakiekolwiek zmiany zostaną wprowadzone do domeny, to wszystkie związane z nią atrybuty we wszystkich encjach oraz dokumentacja zawierająca dane o domenach zostaną odpowiednio dopasowane.

Przyjrzyjmy się na przykład atrybutowi numeru telefonu. Bardzo często w modelach logicznych numery telefonów są projektowane jako numery lokalizowane. W Polsce taki numer jest zwykle zapisywany w postaci trzech cyfr numeru kierunkowego, za którymi znajduje się lokalny numer telefonu (XXX-XXXXXXX). Jeżeli

w dalszych pracach nad modelem zdecydujemy się zapisywać również numery międzynarodowe, a atrybut numeru telefonu znalazł się już w wielu encjach, to będziemy zmuszeni do edytowania wszystkich wystąpień tego atrybutu. Jeżeli jednak przygotujemy sobie domenę numeru telefonu i dodamy ją do wszystkich encji przechowujących taki numer, to wszelkie zmiany wprowadzane później do domeny będą od razu przenoszone do wszystkich encji w tym modelu.

Częste stosowanie domen pozwala na uniknięcie niedogodności związanych z małymi różnicami między atrybutami przechowującymi te same dane w różnych encjach. Praktyka ta pozwala na wymuszenie spójności danych i skrócenie czasu projektowania, nie tylko w okresie początkowych prac nad modelem, ale również w czasie funkcjonowania bazy danych.

Atrybuty z pojedynczą wartością i z wieloma wartościami

Wszystkie omawiane do tej pory atrybuty były **atrybutami z pojedynczą wartością** (ang. *single-valued attributes*). Oznacza to, że dla każdego unikatowego elementu w encji istnieje tylko po jednej wartości zapisanej w poszczególnych atrybutach. Niektóre atrybuty mogą jednak w sposób naturalny przyjmować więcej niż jedną wartość. Są one nazywane **atrybutami z wieloma wartościami** (ang. *multivalued attributes*). Zidentyfikowanie ich może być kłopotliwe, ale obsługa jest bardzo prosta.

Bardzo typowym przykładem atrybutu z wieloma wartościami jest numer telefonu. W czasie zapisywania informacji o kliencie najczęściej zapisujemy przynajmniej jeden numer telefonu, ale klienci posiadający kilka telefonów z pewnością nie są wyjątkami. Możemy zatem dodać do encji *Klient* wiele pól opisujących numer telefonu i odpowiednio je ponumerować (na przykład *Telefon1*, *Telefon2* itd.) albo nazwać zgodnie z rodzajem telefonu (na przykład *Dom*, *Komorka*, *Biuro*). Jest to całkiem niezłe wyjście, dopóki nie zechcemy zapisać kilku firmowych numerów naszego klienta. Jest to typowy atrybut z wieloma wartościami, czyli jeden atrybut może przechowywać wiele różnych wartości.

Z pewnością nie chcemy przechowywać wielu rekordów opisujących jednego klienta tylko po to, żeby zapisać dodatkowy numer telefonu. Byłoby to całkowicie sprzeczne z ideą relacyjnej bazy danych i wprowadzałoby problemy z odczytywaniem z niej danych. W związku z tym możemy wprowadzić do modelu nową encję, która będzie przechowywała numery telefonów i będzie związana relacją z encją klientów (bazującą na kluczu głównym tej tabeli). Dzięki temu zyskamy możliwość zapisania dowolnej liczby numerów telefonów dla każdego z klientów. Powstała encja może przechowywać wiele różnych wartości dla każdego klienta, czyli zawierać tylko dwa atrybuty: unikatowy identyfikator klienta i numer telefonu.

Zastosowanie takiej encji jest jedyną metodą na skuteczne rozwiązanie problemów z atrybutami o wielu wartościach. Co więcej, z takiej konstrukcji skorzysta również fizyczna implementacja, ponieważ będzie mogła wykorzystać techniki wyszukiwania oferowane przez system zarządzania bazą danych, które pozwalają na przeszukiwanie encji zależnej w oderwaniu od encji nadrzędnej.

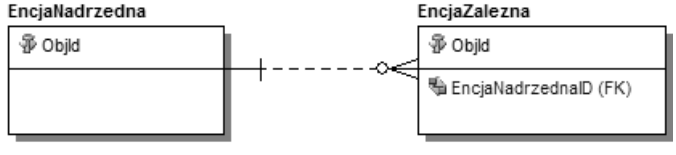
Spójność referencji

Jedną z najważniejszych cech relacyjnej bazy danych jest to, że dane w jednej encji mogą wskazywać na dane w innej encji. Jeżeli taka sytuacja ma miejsce, to niemal zawsze powstaje też wymóg zarządzania taką relacją. Dane w obu encjach muszą być przecież całkowicie spójne. Koncepcja ta nazywana jest **spójnością referencji** (ang. *referential integrity*), a w fizycznej implementacji danych jest najczęściej realizowana za pomocą obiektów takich jak klucze i ograniczenia. Nie można jednak zapomnieć, że spójność referencji musi zostać odpowiednio udokumentowana w modelu logicznym, co gwarantuje przestrzeganie w bazie reguł biznesowych (jak również spójność zapisanych w niej danych).

Załóżmy, że projektujemy bazę danych przechowującą informacje na temat inwentarza biblioteki. W modelu logicznym z pewnością znajdą się encje *Autor*, *Wydawca* i *Tytuł*. Każdy autor może napisać wiele książek, ale dana książka może zostać opublikowana tylko przez jednego wydawcę. Wydawca może publikować wiele książek. Jeżeli użytkownik będzie chciał usunąć dane autora z bazy, to z pewnością zostanie w niej przynajmniej jedna osierocona książka. Po usunięciu danych wydawcy również co najmniej jedna książka w bazie zostanie osierocona.

W związku z tym musimy zdefiniować też akcje, których wykonanie będzie wymuszane za każdym razem, gdy dane będą aktualizowane w ten sposób. Takie definicje nazywane są właśnie spójnością referencji. Mogą one na przykład nakazać usunięcie wszystkich książek związanych z usuwanym właśnie autorem. Możliwe jest też zdefiniowanie reguły zakazującej dodania do bazy książki, jeżeli nie istnieje w niej jeszcze zapis dotyczący autora tej książki. Nie są to może najdoskonalsze przykłady, ale dobrze ilustrują konieczność zarządzania relacjami łączącymi poszczególne encje.

W modelu logicznym spójność referencji jest dokumentowana za pomocą związków tworzonych przez klucze główne i obce. Każda encja powinna mieć atrybut kluczowy, jednoznacznie identyfikujący wszystkie zapisane w niej rekordy. Możemy wykorzystać te klucze do zdefiniowania relacji łączącej encję nadrzędną z encją zależną. Proszę spojrzeć na rysunek 2.1.



Rysunek 2.1. Klucz główny i klucz obcy

Przykład przedstawia prostą relację między dwoma encjami. Po utworzeniu relacji możemy w jej definicji podać dowolne ograniczenia akcji związanych z manipulacjami na danych w encji nadrzędnej i zależnej. Na przykład można zdefiniować, że dowolna próba dopisania danych (`INSERT`) do encji zależnej nie może się powieść, jeżeli w encji nadrzędnej nie istnieje egzemplarz o pasującym kluczu głównym.

Można też określić, że dowolna operacja usuwania danych (DELETE) wykonana na encji nadrzędnej nie może się udać, jeżeli w encji zależnej istnieją jeszcze egzemplarze związane z usuwanym wierszem w encji nadrzędnej. W tabeli 2.1. zostały opisane różne opcje, z jakich możemy skorzystać podczas wykonywania akcji na encji nadrzędnej lub zależnej.

Tabela 2.1. Opcje spójności referencji dla relacji

Encja	Akcja	Dostępne akcje
Encja nadrzędna	INSERT	Brak: wstawienie nowego egzemplarza nie ma wpływu na encję zależną
	UPDATE	Brak: nie ma żadnego wpływu na encję zależną i nie powoduje zablokowania zmian, w wyniku których powstają nieścisłości między encją nadrzędną i zależną
		Ograniczenie: porównuje dane w kluczu głównym encji nadrzędnej z danymi w kluczu obcym encji zależnej. Jeżeli wartości te nie są identyczne, to zmiany są blokowane
		Kaskada: kopiuje wszystkie zmiany z klucza głównego encji nadrzędnej do klucza obcego encji zależnej
		Null: podobne do opcji Ograniczenie. Jeżeli wartości nie są identyczne, to klucz obcy encji zależnej otrzymuje wartość NULL, a zmiany w encji nadrzędnej są wykonywane
	DELETE	Brak: nie ma żadnego wpływu na encję zależną i nie powoduje zablokowania zmian, w wyniku których powstają w encji zależnej wiersze osierocone
		Ograniczenie: porównuje dane w kluczu głównym encji nadrzędnej z danymi w kluczu obcym encji zależnej. Jeżeli wartości te nie są identyczne, to zmiany są blokowane
		Kaskada: usuwa wszystkie wiersze encji zależnej, których klucze obce pasują do klucza głównego wiersza usuwanego w encji nadrzędnej
		Null: podobne do opcji Ograniczenie. Jeżeli wartości nie są identyczne, to klucz obcy encji zależnej otrzymuje wartość NULL, a zmiany w encji nadrzędnej są wykonywane. Ta opcja powoduje powstanie osieroconych wierszy w encji zależnej
Encja zależna	INSERT	Brak: brak ograniczeń
		Ograniczenie: porównuje dane w kluczu głównym encji nadrzędnej z kluczem obcym wartości wstawianej do encji zależnej. Jeżeli wartości te nie są zgodne, to próba wstawienia danych jest blokowana
	UPDATE	Brak: brak ograniczeń
		Ograniczenie: porównuje dane w kluczu głównym encji nadrzędnej z kluczem obcym wartości wstawianej do encji zależnej. Jeżeli wartości te nie są zgodne, to próba usunięcia danych jest blokowana
	DELETE	Brak: brak ograniczeń. Pozwala na usunięcie dowolnego wiersza z encji zależnej

Relacje

Pojęcie *relacyjnej bazy danych* implikuje wykorzystanie w bazie relacji. Jeżeli nie wiemy, w jaki sposób dane są z sobą powiązane, to zastosowanie do ich przechowywania relacyjnej bazy danych nie będzie się w niczym różniło od przechowywania wszystkich dokumentów finansowych w jednym worku. Wypełnianie rocznego

zeznania podatkowego może się w takiej sytuacji okazać prawdziwym koszmarem. Wszystkie niezbędne informacje mamy pod ręką, ale ile czasu zajmie nam wyszukanie ich w tym worku i ostateczne wypełnienie dokumentów?

Zaletą relacyjnych baz danych jest to, że pozwalają na wydajne przechowywanie i odczytywanie danych. Identyfikacja i implementacja właściwych relacji w modelu logicznym to dwa najważniejsze kroki w procesie projektowania. Chcąc prawidłowo zidentyfikować wszystkie relacje, musimy znać wszystkie dostępne możliwości, wiedzieć, jak je prawidłowo rozpoznawać, i podjąć decyzję o tym, której relacji należy użyć.

Typy relacji

Pod względem logicznym istnieją trzy różne relacje łączące encje: jeden-do-jednego, jeden-do-wielu i wiele-do-wielu. Każda z nich opisuje sposób, w jaki dwie encje złączą się z sobą. Należy pamiętać o tym, że te relacje są tylko *relacjami logicznymi*. Ich fizyczna implementacja jest kolejnym krokiem, o którym będziemy mówić w rozdziale 9.

Relacje jeden-do-jednego

Relacja **jeden-do-jednego** łącząca dwie encje jest ich bezpośrednim połączeniem, na co wskazuje jej nazwa. Każdy rekord w jednej encji ma dokładnie jeden pasujący do niego rekord w drugiej encji. Nie mniej i nie więcej. Proszę sobie wyobrazić dwoje ludzi rzucających piłką. W tej sytuacji jest dokładnie jeden rzucający i jeden łapiący. Może być tylko jeden rzucający i tylko jeden łapiący (ktoś, kto faktycznie złapał piłkę).

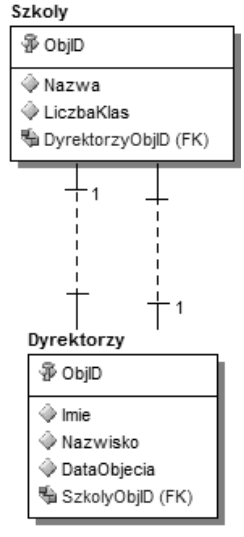
Po co w takim razie wybierać relację typu jeden-do-jednego? Skoro jeden rekord w jednej encji jest związany z dokładnie jednym rekordem w drugiej encji, to czy nie lepiej byłoby połączyć te encje? Proszę spojrzeć na rysunek 2.2.



Rysunek 2.2. Encja Szkoły

Każda szkoła może mieć tylko jednego dyrektora, a każdy dyrektor może zarządzać tylko jedną szkołą. W tym przykładzie wszystkie atrybuty encji dyrektora są przechowywane w encji szkoły. To rozwiązanie skupia wszystkie informacje w ramach jednej encji, ale jest bardzo mało elastyczne. Przy każdej zmianie danych szkoły *lub* dyrektora konieczne jest odczytanie całego rekordu, a następnie jego uaktualnienie. Oprócz tego szkoła bez dyrektora lub dyrektor bez szkoły będą tworzyć rekord

wypełniony tylko w połowie. Co gorsza, taka konstrukcja powoduje problemy przy odczytywaniu danych. Jeżeli chcielibyśmy napisać raport zawierający informacje o dyrektorach, to bylibyśmy zmuszeni do odczytywania również danych szkół. A co będzie, jeżeli chcielibyśmy gromadzić informacje o pracownikach dyrektorów? W takim przypadku musielibyśmy powiązać pracowników z połączoną encją szkół i dyrektorów, a nie tylko z encją dyrektorów. Proszę się teraz przyjrzeć rysunkowi 2.3.



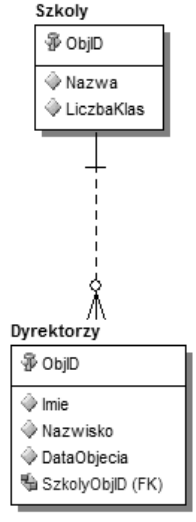
Rysunek 2.3. Encje Szkoly i Dyrektorzy

W tym przykładzie mamy już dwie encje: *Szkoly* i *Dyrektorzy*. Każda z nich ma atrybuty opisujące dany rodzaj obiektu. Oprócz tego w encji *Dyrektorzy* jest zapisana referencja na szkołę, którą dany dyrektor prowadzi, a w encji *Szkoly* znajdziemy referencję dyrektora opiekującego się daną szkołą. Takie rozwiązanie jest dużo bardziej elastyczne, ponieważ listy szkół i dyrektorów są zarządzane całkowicie niezależnie. Proszę jednak zauważyć, że encje te łączy relacja jeden-do-jednego, która pozwala na nałożenie ograniczeń umożliwiających utrzymanie spójności danych.

Relacje jeden-do-wielu

W najczęściej występujących relacjach typu **jeden-do-wielu** jeden rekord w pierwszej encji może mieć zero lub więcej pasujących rekordów w drugiej encji. Istnieją setki przykładów takich relacji, stosowanych najczęściej w ramach łączenia danych nagłówkowych z informacjami szczegółowymi. Na przykład zamówienia są często przechowywane jako lista rekordów **nagłówkowych** w jednej encji, druga encja gromadzi natomiast **szczegółowe** informacje na temat poszczególnych zamówień. Takie rozwiązanie pozwala na umieszczenie w każdym zamówieniu wielu pozycji bez konieczności tworzenia wielu rekordów zawierających te same informacje wysokiego poziomu, takie jak data zamówienia, dane klienta itp.

Wróćmy do przykładu ze szkołami i dyrektorami. Co się stanie, jeżeli pojawi się zarządzenie pozwalające pracować kilku dyrektorom w jednej szkole? Taka sytuacja natychmiast tworzy relację jeden-do-wielu łączącą encje *Dyrektorzy* i *Szkoły*, tak jak pokazano na rysunku 2.4.



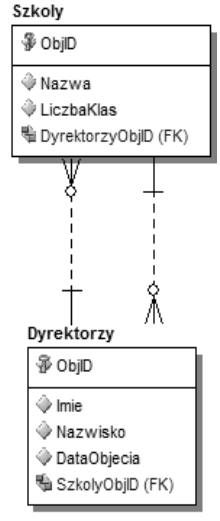
Rysunek 2.4. Encje Szkoły i Dyrektorzy — relacja jeden-do-wielu

Jak widać, taka relacja łącząca te dwie encje stwarza *możliwość* powiązania kilku dyrektorów z jedną szkołą. Jest ona skalowalna, ponieważ obie encje mogą być aktualizowane i zarządzane całkowicie niezależnie od siebie.

Relacje wiele-do-wielu

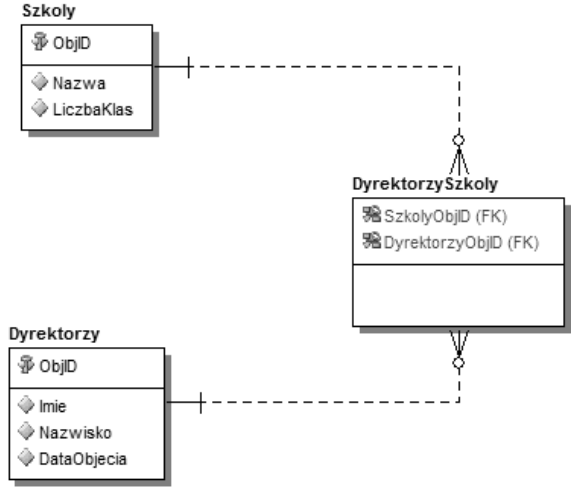
Spośród wszystkich rodzajów relacji relacje wiele-do-wielu (nazywane również relacjami nieokreślonymi) należą do najbardziej skomplikowanych do zidentyfikowania i zaprojektowania. W relacjach wiele-do-wielu rekordy z jednej encji mogą łączyć się z wieloma rekordami w drugiej encji, a jednocześnie rekordy z drugiej encji mogą wiązać się z wieloma rekordami z pierwszej encji. Proszę sobie wyobrazić części samochodowe, a w szczególności siedzenia montowane w każdym aucie. W dowolnym samochodzie znajdziemy przynajmniej dwa rodzaje siedzeń: fotele dla kierowcy i pasażera oraz dużą kanapę dla pasażerów siedzących z tyłu. Producenci samochodów niemal zawsze wykorzystują istniejące siedzenia w kilku modelach samochodów. Jeśli przełożymy te rozważania na encje, to okaże się, że *Siedzenie* może znaleźć się w wielu *Samochodach*, a każdy *Samochód* może zawierać wiele *Siedzeń*.

Wróćmy jednak do naszego przykładu ze szkołami. Co będzie, jeżeli zostanie podjęta decyzja, że jeden dyrektor może zarządzać kilkoma szkołami, a jedna szkoła może znajdować się pod opieką kilku dyrektorów? Na rysunku 2.5. przedstawiliśmy encje *Szkoły* i *Dyrektorzy* związane relacjami pozwalającymi na tworzenie wielu połączeń między obiema encjami.



Rysunek 2.5. Encje Szkoly i Dyrektorzy — relacja wiele-do-wielu

Rozpatrując tę sprawę czysto teoretycznie, trzeba powiedzieć, że wszystkie relacje mogą istnieć tylko między dwoma encjami. Pod względem logicznym mamy więc relację między encjami *Szkoly* i *Dyrektorzy*. Pod względem technicznym możemy zastosować notację z dwoma relacjami typu jeden-do-wielu łączącymi te encje, ale skierowanymi przeciwnie. Innym rozwiązaniem jest zastosowanie relacji z symbolem „wiele" umieszczonym po obu jej końcach. Z praktycznego punktu widzenia najwygodniej jednak będzie utworzyć trzecią encję obrazującą taką relację, podobnie jak zostało to przedstawione na rysunku 2.6.



Rysunek 2.6. Encja Szkoly i Dyrektorzy — relacja wiele-do-wielu z trzecią encją

Niektórzy stwierdzą, że jest to pogwałcenie ideału modelu logicznego, który nie powinien zawierać żadnych elementów fizycznej implementacji. Zastosowanie trzeciej encji łączącej encje *Szkoły* i *Dyrektorzy* za pomocą ich identyfikatorów jest odwzorowaniem fizycznej implementacji stosowanej wobec wielu relacji typu wiele-do-wielu. Fizycznie nie jest możliwe odwzorowanie relacji tego typu bez wykorzystania trzeciej tabeli, nazywanej czasami **tabelą łącznikową** (ang. *join table*). Oznacza to, że wykorzystanie tej tabeli w modelu logicznym nie jest ściśle zgodne ze wskazówkami modelowania logicznego. Umieszczenie dodatkowej encji w modelu logicznym może jednak ułatwiać zapamiętanie, dlaczego została użyta taka relacja. Będzie też pomocą dla przyszłych współpracowników poznających dopiero wszystkie relacje w tym modelu.

Zastosowanie trzeciej encji pozwala ponadto na wprowadzenie dodatkowych atrybutów opisujących każdy egzemplarz takiej relacji. Na przykład w encji *Dyrektorzy_Szkoły* można dodać informację, od kiedy dany dyrektor zajmuje się określoną szkołą. Jeżeli istnieje wiele takich kombinacji, to informacja o czasie zajmowania stanowiska przez daną osobę w danej szkole może być bardzo przydatna.

Relacje typu wiele-do-wielu są stosowane zadziwiająco często, ale zawsze należy podchodzić do nich bardzo ostrożnie i dokładnie je dokumentować, tak żeby nie było żadnych nieporozumień w trakcie tworzenia fizycznej implementacji modelu.

Opcje relacji

Skoro znasz już różne rodzaje relacji, musimy omówić jeszcze opcje, które możemy definiować w relacjach poszczególnych typów. Opcje te pozwalają na dokładniejsze sterowanie zachowaniem każdej relacji z osobna.

Relacje identyfikujące i nieidentyfikujące

Jeżeli klucz główny encji zależnej wymaga dołączenia klucza głównego encji nadrzędnej, to znaczy, że relacja łącząca te encje jest **relacją identyfikującą** (ang. *identifying relation*). Wynika to z tego, że unikatowy atrybut encji zależnej wymaga zastosowania wyjątkowego atrybutu encji nadrzędnej, aby jednoznacznie zidentyfikować odpowiedni egzemplarz encji. Jeżeli takie wymaganie nie istnieje, to relacja jest definiowana jako **nieidentyfikująca** (ang. *non-identifying relation*).

W przypadku relacji identyfikującej klucz główny encji nadrzędnej jest faktycznie jednym z atrybutów składających się na klucz główny encji zależnej. Oznacza to, że klucz obcy encji zależnej jest częścią lub nawet całością jej klucza głównego. W przypadku relacji nieidentyfikującej klucz główny encji nadrzędnej jest tylko niekluczowym atrybutem encji zależnej.

Zaledwie niewielka część relacji to relacje identyfikujące, ponieważ większość encji zależnych może być referowanych całkowicie niezależnie od encji nadrzędnej. W relacjach typu wiele-do-wielu są jednak często stosowane relacje identyfikujące, jako że dodatkowa encja wiąże z sobą klucze główne encji nadrzędnej i zależnej. Na przykład encja *Szkoły_Dyrektorzy* z prezentowanego wcześniej rysunku 2.6. zawiera w całości klucze główne encji *Szkoły* (*SzkołyID*) i *Dyrektorzy* (*DyrektorzyID*).

Proszę zauważyć, że jest to typowe w przypadku relacji typu wiele-do-wielu. Klucz główny tabeli łącznikowej zawsze składa się z kluczy głównych dwóch pozostałych tabel. Dzięki temu, że klucze główne tabeli nadrzędnej i zależnej są od razu widoczne w tabeli łącznikowej, możemy bez wahania powiedzieć, że jest to relacja identyfikująca.

Relacje nieidentyfikujące są zdecydowanie bardziej rozpowszechnione. Można je rozpoznać po tym, że atrybuty klucza głównego encji nadrzędnej są w encji zależnej atrybutami niekluczowymi. Jeżeli nie istnieje jakiś ważny powód do tworzenia relacji identyfikującej, to większość tworzonych w modelu relacji będzie relacjami nieidentyfikującymi.

Relacje opcjonalne i obowiązkowe

Każda relacja w bazie danych musi zostać określona jako opcjonalna lub obowiązkowa. O relacjach **obowiązkowych** można myśleć jak o relacjach typu „musi mieć", podczas gdy relacje **opcjonalne** można określać jako relacje „może mieć". Na przykład jeżeli mamy encję *Pracownik* oraz encję *Biuro*, to każdy pracownik „musi mieć" swoje biuro. Relacja łącząca te dwie encje opisuje biuro danego pracownika. W takiej sytuacji jest tworzona relacja nieidentyfikująca, a ponieważ w encji *Pracownik* nie możemy wpisać wartości `NULL` do klucza obcego identyfikującego biuro, okazuje się, że jest to również relacja obowiązkowa. Wskazuje ona, że każdy pracownik ma przynajmniej jedno biuro, a nawet jeżeli musi pracować w innych biurach, to jedno z nich traktowane jest jako przypisane do niego.

Teraz zajmijmy się sytuacją, w której samochody są przypisywane do pracowników firmy. Można ją odzwierciedlić w modelu danych, łącząc z sobą encje *Pracownik* i *Samochód*. Jak można się domyślić, pracownik „może mieć" samochód, choć wcale nie musi, co odpowiada definicji relacji opcjonalnej.

Liczność

W przypadku każdej z opisywanych relacji podawaliśmy jedynie jej ogólny rodzaj: jeden-do-jednego, jeden-do-wielu lub wiele-do-wielu. Opis takiej relacji zawiera liczbę rekordów w encji nadrzędnej, wiązanych z pewną liczbą rekordów w encji zależnej. Chcąc dokładniej wymodelować faktyczną relację łączącą te dane, musimy podać nieco więcej informacji opisujących te zależności — niezbędne jest określenie **liczności** (ang. *cardinality*) danej relacji.

W przypadku relacji jeden-do-jednego jej liczność jest już z góry narzucona. Jednoznacznie stwierdzamy, że na każdy rekord w encji nadrzędnej może przypadać dokładnie jeden rekord w encji zależnej. W sposób bardziej opisowy można powiedzieć, że na każdy rekord w encji nadrzędnej przypada zero lub jeden rekord w encji zależnej. Jeżeli jednak chcemy zaznaczyć, że w obu encjach zawsze musi być dokładnie jeden rekord, to opis liczności relacji powinien brzmieć: na każdy rekord w encji nadrzędnej przypada dokładnie jeden rekord w encji zależnej. Liczność relacji typu jeden-do-jednego jest zazwyczaj zapisywana jako `[1:1]`.

Relacje jeden-do-wielu są definiowane skrótem [1:M], a ich liczność można opisać słowami: na każdy rekord w encji nadrzędnej przypada jeden lub więcej rekordów w encji zależnej. Jeżeli jednak chcemy dopuścić sytuację, w której w encji zależnej nie ma rekordów pasujących do rekordów encji nadrzędnej, to można zastosować inny opis liczności takiej relacji: na każdy rekord w encji nadrzędnej przypada zero lub więcej rekordów w encji zależnej. W większości relacji właściwszą interpretacją jest wersja „zero lub więcej", dlatego w czasie tworzenia modelu należy dokładnie określić wariant liczności.

Relacje typu wiele-do-wielu można definiować jako zero lub więcej do zera lub więcej rekordów. Jak się okazuje, ten rodzaj liczności jest niemal zawsze odgórnie przyjmowany za domyślny dla tych relacji, choć oczywiście może zdefiniować wymóg istnienia przynajmniej jednego rekordu w każdej z encji. W takim przypadku relację typu wiele-do-wielu można zapisać jako [M:M].

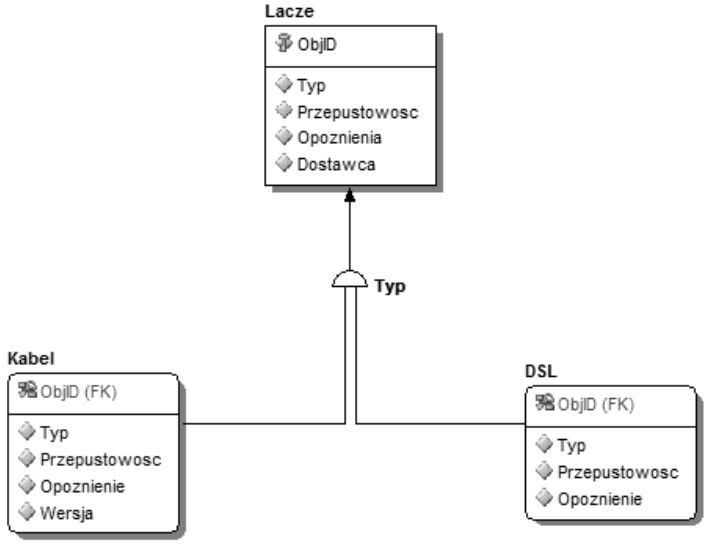
W niektórych programach do modelowania danych możliwe jest bardzo dokładne definiowanie liczności relacji. Można na przykład zdefiniować liczność typu „na każdy rekord w encji nadrzędnej przypada osiem rekordów w encji zależnej". W ten sposób można zdefiniować osoby bezpośrednio podporządkowane kierownikom (osoby podlegające danemu menedżerowi). Firma może przecież wymagać, aby pracujący w niej menedżer miał nie mniej niż cztery, ale nie więcej niż dwadzieścia podporządkowanych sobie bezpośrednio osób. W takiej sytuacji liczność relacji można by opisać jako: przynajmniej cztery, ale nie więcej niż dwadzieścia do jednego. Tego rodzaju liczności wymagają dokładnego udokumentowania, ponieważ większość ludzi przyjmie po prostu ich domyślną postać.

Używanie podtypów i typów nadrzędnych

Definiując encje, które mają być wykorzystane w modelu danych, od czasu do czasu możemy wygenerować taką, która będzie się składać w całości z kilku innych. W takiej sytuacji możemy mieć problemy z określeniem, które atrybuty należą do danej encji i jakie łączą je relacje. Rozwiązaniem może się okazać zastosowanie typów nadrzędnych.

Definicje podtypów i typów nadrzędnych

Typ nadrzędny (ang. *supertype*) jest encją mającą wiele encji zależnych, nazywanych **podtypami** (ang. *subtypes*), które opisują różne warianty tego samego typu nadrzędnego. Kolekcja typu nadrzędnego i wszystkich jego podtypów jest czasami nazywana **klastrem podtypów** (ang. *subtype cluster*). Takie klastry pojawiają się najczęściej przy okazji prac nad pewnymi kategoriami niektórych rzeczy, podobnie jak zostało to przedstawione na rysunku 2.7.



Rysunek 2.7. Prosty klaster podtypów

Załóżmy, że śledzimy informacje na temat produktów związanych z dostępem szerokopasmowym. W tym przykładzie *Lacze* jest encją z kilkoma atrybutami i kluczem głównym. Chcemy jednak, żeby różne rodzaje łączy były zapisywane w osobnych encjach, ponieważ łącza kablowe są oferowane klientom prywatnym i komercyjnym, a łącza DSL — wyłącznie klientom prywatnym. Oba rodzaje łączy *mogłyby* być zapisywane w całkowicie niezależnych encjach, ale w ten sposób stracilibyśmy cały obraz relacji. W encji *Lacze* znajdują się atrybuty, których nie znajdziemy w pozostałych dwóch encjach. Te dwie encje zawierają natomiast atrybuty, których próżno szukać w encji *Lacze*. Poza tym przygotowany projekt musi być otwarty, na wypadek gdybyśmy w przyszłości chcieli dodać kolejne rodzaje łączy szerokopasmowych bez konieczności modyfikowania już istniejących.

Chcąc rozwiązać ten problem, powinniśmy przekształcić encję *Lacze* w typ nadrzędny, a encje *Kabel* i *DSL* w podtypy. W tym celu najpierw musimy utworzyć encje podtypów i nadać im wszystkie niezbędne atrybuty, *z wyjątkiem* klucza głównego. Następnie trzeba przygotować obowiązkową relację identyfikującą, łączącą encję typu nadrzędnego z encjami podtypów. Taka relacja oznacza, że klucz główny encji *Lacze* jest jednocześnie kluczem głównym pozostałych dwóch encji. Na zakończenie niezbędne jest jeszcze wybranie **dyskryminatora** (ang. *discriminator*), czyli atrybutu w encji nadrzędnej, którego wartość określa podtyp danego rekordu. Dyskryminatorem może być zarówno atrybut kluczowy, jak i niekluczowy. W tym przykładzie funkcję tę pełni atrybut *Typ*, który może otrzymać wartość `DSL` lub `Kabel`.

Jeżeli klaster podtypów zawiera już wszystkie możliwe podtypy danego typu nadrzędnego, to mówi się o nim, że jest klastrem **kompletnym**. Jeżeli jednak zawiera tylko część możliwych podtypów, to o takim klastrze mówi się, że jest klastrem

niekompletnym. Opisy klastra to sprawa związana wyłącznie z dokumentacją, ale podobnie jak ze wszystkimi innymi elementami tworzenia modelu, prawidłowe dokumentowanie szczegółów może bardzo ułatwić nam pracę w przyszłości.

Fizyczna implementacja klastra podtypów jest zawsze zależna od konkretnego przypadku. Klastry podtypów mogą być implementowane w postaci relacji typu jeden-do-jednego łączących encje i tabele albo w postaci różnych innych kombinacji tabel i łączących je relacji. Najistotniejszym zagadnieniem, o którym warto pamiętać, jest umieszczanie tego samego klucza głównego we wszystkich encjach składających się na klaster, a także zastosowanie prawidłowych ograniczeń dyskryminatora, tak żeby wszystkie rekordy znalazły się we właściwych tabelach.

Kiedy używać klastrów podtypów

Niemal każdy model danych będzie zawierać encje, które mają atrybuty, a te z kolei przechowują informacje na temat małego podzbioru rekordów znajdujących się w encji. Jeżeli natkniemy się na taką sytuację w swoim modelu danych, powinniśmy dokładniej przyjrzeć się tym atrybutom i sprawdzić, czy nie można z nich zbudować klastra podtypów. Nie należy jednak za wszelką cenę tworzyć relacji budujących klaster podtypów. W ten sposób skonstruujemy tylko zagmatwany model danych, zawierający więcej encji, niż faktycznie potrzeba. Co więcej, istnienie nadmiarowych klastrów podtypów będzie bardzo utrudniało fizyczną implementację modelu, prowadząc do powstawania niepotrzebnych tabel i ograniczeń. To z kolei może przekładać się na pogorszenie wydajności całej bazy danych i w konsekwencji uniemożliwiać jej prawidłową obsługę.

Klastry podtypów mogą być bardzo przydatnym narzędziem, wprowadzającym do modelu danych wysoki poziom elastyczności. Modelowanie danych za pomocą tego rodzaju uogólnionej hierarchii umożliwia wprowadzanie w przyszłości różnych modyfikacji bez konieczności poprawiania istniejących encji, dlatego z pewnością warto poświęcić nieco czasu na poszukiwanie związków logicznych pozwalających na zastosowanie klastrów podtypów.

Podsumowanie

W niniejszym rozdziale omówiliśmy narzędzia stosowane podczas tworzenia logicznego modelu danych. Każdy model danych składa się z obiektów niezbędnych do prawidłowego opisania przechowywanych danych, definicji wszystkich związków łączących poszczególne fragmenty danych oraz istniejących ograniczeń.

Skoro znasz już wszystkie elementy składające się na logiczne modele danych, możesz zajrzeć do rozdziału 3., w którym postaramy się łagodnie przejść od zagadnień związanych z obiektami logicznymi do tych dotyczących ich fizycznej implementacji. Później wykorzystamy te wszystkie informacje do zbudowania modelu danych dla firmy Mountain View Music.