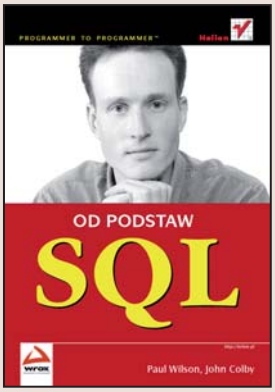
SQL. Od podstaw



Autorzy: Paul Wilton, John Colby
Tłumaczenie: Mikołaj Szczepaniak
ISBN: 83-7361-999-2
Tytuł oryginału: Beginning SQL
Format: B5, stron: 473

Dane i informacje to dziś najcenniejszy towar. Systemy zarządzania bazami danych to potężne narzędzia, pozwalające nie tylko na przechowywanie danych, ale także na ich przetwarzanie, modyfikowanie i wydobywanie w oparciu o przeróżne kryteria. Podstawą wszystkich operacji na danych zgromadzonych w bazach jest język SQL – narzędzie służące do manipulowania zbiorami informacji. SQL, przyjęty i zatwierdzony przez międzynarodowe organizacje i komitety standaryzacyjne, jest wykorzystywany w niemal wszystkich systemach zarządzania bazami danych. Każdy z producentów dodaje do niego "własne" elementy, ale rdzeń języka pozostaje taki sam niezależnie od platformy.

Książka „SQL. Od podstaw" to podręcznik języka SQL i omówienie zagadnień związanych z projektowaniem baz danych przeznaczone dla osób wkraczających dopiero w świat baz danych. Przedstawia podstawowe wyrażenia języka SQL, służące do wprowadzania danych do bazy, wyciągania ich oraz manipulowania nimi. Czytając tą tę książkę, dowiesz się, czym są złączenia i podzapytania, postaci normalne baz danych oraz transakcje i perspektywy. Poznasz sposoby projektowania tabel, zabezpieczania zgromadzonych w nich informacji oraz metody podnoszenia wydajności i szybkości działania baz danych.

- Struktura relacyjnych baz danych
- Wpisywanie danych do bazy
- Wydobywanie i porządkowanie danych
- Normalizacja i postaci normalne
- Projektowanie baz danych
- Operacje matematyczne, grupujące i agregujące
- Stosowanie złączeń i podzapytań
- Tworzenie i wykorzystywanie perspektyw
- Mechanizmy transakcyjne
- Podnoszenie wydajności bazy danych i optymalizowanie zapytań

Dzięki tej książce poznasz wszystko – znajdziesz omówienie tego, co może okazać się potrzebne podczas projektowania i korzystania z baz danych.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

1

Wprowadzenie do języka SQL

Na dobry początek zastanowimy się, czym w ogóle są bazy danych oraz kiedy i dlaczego należy z nich korzystać. W dalszej części rozdziału przejdziemy do omawiania strukturalnego języka zapytań (SQL) i przyjrzymy się jego związkom z bazami danych oraz technikom wykorzystywania w praktycznych zastosowaniach. Po poznaniu podstaw i teoretycznych możliwości języka SQL dowiesz się, jak można za jego pomocą stworzyć bazę danych. Niniejszy rozdział przeprowadzi Cię też przez proces budowania struktury przykładowej bazy danych, która będzie wykorzystywana w kolejnych rozdziałach tej książki.

Po przeczytaniu tego rozdziału powinieneś nie tylko rozumieć, jak to się dzieje, że bazy danych umożliwiają efektywne organizowanie i wyszukiwanie potrzebnych informacji, ale też wiedzieć, jak tworzyć w pełni funkcjonalne bazy danych (gotowe do przyjmowania i składowania nowych danych). Zanim jednak zagłębisz się w pisanie kolejnych wierszy kodu języka SQL, powinieneś uzyskać niezbędne podstawy teoretyczne.

Krótka historia baz danych

Bazy danych we współczesnej formie istnieją od lat sześćdziesiątych — są efektem badań między innymi firmy IBM. Wysiłki laboratoriów badawczych koncentrowały się wokół automatyzacji typowych zadań biurowych, w szczególności zadań składowania i indeksowania danych, które wcześniej wymagały pracy ludzkich rąk i jako takie były bardzo czasochłonne. Z czasem koszt mocy obliczeniowej i pamięci znacznie się obniżył, co pozwoliło na dużą popularyzację komputerów w obszarze składowania i indeksowania danych. Prekursorem zastosowań baz danych był Charles W. Bachman, który w roku 1973 otrzymał Nagrodę Turinga za swoje pionierskie dokonania w dziedzinie technologii baz danych. W roku 1970 Ted Codd, naukowiec zatrudniony w laboratoriach firmy IBM, opublikował pierwszy artykuł poświęcony relacyjnym bazom danych.

Mimo że IBM był absolutnym liderem badań nad bazami danych, to firma Honeywell Information Systems, Inc. w roku 1976 wprowadziła na rynek swój produkt komercyjny, który bazował co prawda na tych samych regułach co system informacyjny firmy IBM, ale został zaprojektowany i zaimplementowany zupełnie niezależnie od prac tej firmy.

We wczesnych latach osiemdziesiątych ubiegłego stulecia zbudowano pierwsze systemy baz danych oparte na standardzie SQL — w tamtym okresie ukazała się druga wersja systemu Oracle firmy Oracle, system SQL/DS firmy IBM oraz bardzo wiele innych systemów opracowanych przez inne firmy.

Teraz, skoro wiesz co nieco na temat źródeł pochodzenia baz danych, możesz przystąpić do lektury materiału poświęconego bardziej praktycznym zagadnieniom — temu, czym są bazy danych oraz kiedy należy z nich korzystać.

Identyfikacja baz danych

Zastanawiasz się pewnie, czym tak naprawdę jest baza danych.

Darmowy internetowy słownik techniki komputerowej (*Free On-Line Dictionary of Computing* — patrz strona internetowa *http://foldoc.doc.ic.ac.uk*) definiuje bazę danych jako „jeden lub wiele strukturalnych zbiorów trwałych danych, zwykle związanych z oprogramowaniem umożliwiającym aktualizowanie i wykonywanie zapytań na tych danych. Prosta baza danych może mieć postać pojedynczego pliku obejmującego wiele rekordów, z których każdy zawiera ten sam zbiór pól (gdzie każde pole ma ustaloną z góry, stałą szerokość).”.

Spróbujmy rozbić tę definicję na mniejsze, bardziej zrozumiałe składniki; po pierwsze, zgodnie z zacytowanym opisem, baza danych składa się ze strukturalnych zbiorów danych, co oznacza, że zawiera kolekcje danych. Przykładowo, baza danych może zawierać szczegółowe informacje na temat osiągnięć wuja Bronka na polu golfowym lub dane na temat wszystkich książek w jakiejś bibliotece. Najprawdopodobniej mieszanie tych danych ze sobą nie byłoby dla Ciebie korzystne; innymi słowy, zapewne nie chciałbyś szukać informacji o interesującej Cię książce wśród niezwiązanych z nimi danych o wynikach pojedynków golfowych. Krótko mówiąc, bazy danych ułatwiają organizowanie danych. Bazy danych przechowują swoje kolekcje danych w **tabelach** (pojęcie tabeli wyjaśnię bardziej szczegółowo w rozdziale 2.).

Przytoczona definicja mówi też, że bazy danych są zwykle związane z oprogramowaniem umożliwiającym aktualizowanie i wykonywanie zapytań na danych. Do przykładów oprogramowania baz danych, które znajdują zastosowanie w świecie rzeczywistym, należy Access firmy Microsoft, system 10g firmy Oracle, DB2 firmy IBM, MySQL firmy MySQL AB oraz SQL Server 2000 firmy Microsoft. Wymienione programy często są nazywane bazami danych, choć tak naprawdę są systemami zarządzania bazami danych (ang. *Database Management System — DBMS*). Baza danych to **zbiory** (kolekcje) wzajemnie powiązanych danych zgrupowane w jednym bycie. Przykładowo, mógłbyś stworzyć bazę danych w programie Access, nazwać ją `MojaBazaDanych`, dołączyć do nowej bazy rozmaite kolekcje danych i zarządzać całością za pomocą oprogramowania Microsoft Access.

I wreszcie przedstawiona definicja stwierdza, że (tak jak we wspomnianym przykładzie bazy danych Access) prosta baza danych może mieć postać jednego pliku z wieloma rekordami,

z których każdy dzieli się na **pola**. Czym jednak są rekordy i pola? Pole jest pojedynczym elementem danych opisującym określony przedmiot. Takim „przedmiotem" może być np. osoba — wówczas pojedynczym elementem danych o osobie może być data urodzenia. Jeśli reprezentowanym „przedmiotem" będzie adres domu, elementem danych może być składnik tego adresu, np. ulica. W przypadku książki konkretnym fragmentem danych składowanym w pojedynczym polu może być rok wydania, inne pole może zawierać tytuł, jeszcze inne może reprezentować nazwisko autora. Przykładowo, rekord reprezentujący tę książkę w polu `Rok wydania` powinien zawierać wartość `2005`, w polu `Tytuł` wartość `Beginning SQL`, a w polu `Autor` wartości `Paul Wilton` i `John Colby`. Wszystkie te pola odnoszą się do określonego przedmiotu: książki zatytułowanej *Beginning SQL*. Pola te razem tworzą strukturę nazywaną **rekordem**. Każda książka ma swój własny rekord, a wszystkie te rekordy są przechowywane w bazie danych w ramach szerszej struktury nazywanej **tabelą**. Pojedyncza baza danych może zawierać jedną lub wiele tabel. Jeśli masz trudności w opanowaniu informacji przedstawionych do tej pory, nie przejmuj się — pojęcia pól i rekordów będą się pojawiały w dalszej części tego rozdziału jeszcze wielokrotnie.

Mam nadzieję, że rozumiesz już koncepcję bazy danych, której celem jest ułatwienie procesów składowania, organizowania i wydobywania (przeszukiwania) danych. Ostatnim pojęciem wymagającym wyjaśnienia jeszcze w tym punkcie jest termin **relacyjna baza danych**, który dotyczy bazy danych zawierającej zorganizowane i wzajemnie połączone informacje (występujące w odpowiednich relacjach). Wszystkie rekordy takiej bazy danych są zorganizowane w ramach tabel. Powiązane dane, np. szczegóły dotyczące sprzedawców, są grupowane w jednej tabeli. Szczegóły na temat sprzedawanych przez nich samochodów mogą być składowane w innej tabeli określającej relacje pomiędzy reprezentowanymi samochodami a sprzedawcami, którzy je sprzedali — przykładowo, sprzedawca *X* mógł sprzedać samochód *Y* w dniu *Z*. Na rysunku 1.1 przedstawiono jedną z tabel przykładowej bazy danych. Na pierwszy rzut oka przedstawiona struktura może Ci przypominać arkusz kalkulacyjny, którego wiersze reprezentują Twoje rekordy, natomiast kolumny zawierają pola dla tych rekordów. Podczas lektury rozdziału 3. odkryjesz, że tabele należy traktować raczej jak zbiory danych.

Użytkownicy : Tabela

Identyfikator użytkownika	Imię i nazwisko	Adres poczty elektronicznej	Hasło	Otrzymywanie wiadomości	Kraj
1	Janina Jezierska	janina@mojadomena.com	shhhhhhhh	1	Polska
2	John Doe	john@mailserver.org	fkjghskj	0	USA
3	Beci B. Bandanas	beci.danas.co.uk	xyz123	0	Wlk. Brytania
4	Katarzyna Wróbel	kw@poczta.pl	1234567890	1	Polska
5	Bronisław Gil	bronek@gil.net	098765ss	0	Polska
6	Gandalf	gandalf@wizards.net	gfkjghkj	1	Śródziemie
7	Lucy Atall	lucy@sunny.com	jghsdkjfgh	0	Kanada

Rekord: 4 z 7

Rysunek 1.1.

Większość współczesnych systemów zarządzania bazami danych jest relacyjna — mówi się nawet o relacyjnych systemach zarządzania bazami danych (ang. Relational Database Management System — RDBMS). Tego typu systemy wyróżniają się łatwością i efektywnością składowania danych i generowania żądanych wyników. Umożliwiają generowanie odpowiedzi na rozmaite zapytania, włącznie z zapytaniami, których obsługa nigdy nie była brana pod uwagę przez programistę bazy danych.

W jakich okolicznościach należy korzystać z baz danych

Skoro istnieje mnóstwo alternatywnych sposobów przechowywania danych, po co w ogóle robić sobie kłopot i tworzyć bazę danych? Jakie są zalety takiego rozwiązania?

Podstawową zaletą baz danych jest szybkie i efektywne wydobywanie (wyszukiwanie) danych. Baza danych ułatwia też logiczne organizowanie danych. Systemy zarządzania bazami danych z reguły są przystosowane do błyskawicznego wyszukiwania potrzebnych danych w sposób odpowiadający Twoim oczekiwaniom. Wydobywanie danych z bazy danych jest nazywane **wykonywaniem zapytań**. Często będziesz też miał do czynienia z pojęciem **zapytanie języka SQL** (lup po prostu **zapytanie SQL**), które w największym uproszczeniu oznacza dowolny kod języka SQL mający na celu wydobywanie informacji zapisanych w bazie danych. Problematyka zapytań zostanie omówiona bardziej szczegółowo w dalszej części tego rozdziału.

Relacyjne bazy danych mają jeszcze jedną istotną zaletę — umożliwiają definiowanie relacji łączących poszczególne dane (jak choćby w przypadku wspominanej bazy danych o sprzedaży samochodów). Jeśli przechowujesz szczegółowe dane o sprzedaży i informacje o sprzedawcach w tak powiązanych bazach danych, znalezienie odpowiedzi na pytanie „Ile samochodów sprzedał *X* w styczniu?" staje się bardzo proste. Gdybyś jednak upchnął wszystkie te informacje w jednym wielkim pliku tekstowym, znalezienie odpowiedzi na to pytanie (wykonanie takiego zapytania) byłoby niezwykle trudne i czasochłonne.

Bazy danych umożliwiają też definiowanie reguł zapewniających zachowanie spójności danych po ich dodawaniu, aktualizowaniu i usuwaniu. Wyobraź sobie raz jeszcze salon samochodowy z dwoma sprzedawcami o identycznym imieniu i nazwisku (np. Jan Nowak). Można tak zaprojektować bazę danych, aby każdy sprzedawca miał przypisany unikalny identyfikator (w ten sposób unikniemy sytuacji, w której obaj Janowie będą myleni); w przeciwnym przypadku jednoznaczne określenie, kto sprzedał poszczególne samochody, byłoby po prostu niemożliwe. Pozostałe systemy składowania danych, w tym pliki tekstowe i arkusze kalkulacyjne, nie udostępniają tego typu mechanizmów i zwykle dopuszczają do wprowadzania do swoich struktur nawet najbardziej dziwacznych danych. W dalszych rozdziałach zostaną przedstawione techniki definiowania także innych reguł ograniczających ryzyko naruszenia spójności danych. Przykładowo, w bazie danych możesz stworzyć regułę, zgodnie z którą numer PESEL pracownika będzie musiał spełniać warunek unikalności, lub jeśli zostanie sprzedany jakiś samochód i odpowiednia tabela będzie wskazywała, że sprzedaży dokonał pracownik z identyfikatorem 123, możesz wprowadzić mechanizm sprawdzający, czy w jednej z tabel bazy danych znajdują się wszystkie wymagane informacje na temat tego pracownika.

Właściwie zaprojektowana i skonfigurowana baza danych minimalizuje nadmiarowość danych. Wróćmy raz jeszcze do przykładu salonu samochodowego — wszystkie szczegółowe informacje na temat sprzedawcy mogą być składowane w jednym miejscu bazy danych, a do jego identyfikacji można używać unikalnego identyfikatora. Kiedy wprowadzisz inne dane związane z konkretnym sprzedawcą (np. o sprzedanych przez niego samochodach), będziesz mógł wykorzystać ten unikalny identyfikator do przeszukiwania tych danych. Taki unikalny identyfikator często ma postać liczby, której przechowywanie wymaga znacznie mniej przestrzeni niż odpowiednie imię i nazwisko.

Baza danych przechowuje surowe (nieprzetworzone) dane — same fakty przy braku „inteligentnych" mechanizmów. Baza danych o sprzedanych samochodach może zawierać markę,

model i cenę poszczególnych pojazdów, ale raczej nie będzie zawierała liczby samochodów sprzedanych w poszczególnych miesiącach, ponieważ można tę liczbę wyznaczyć na podstawie już przechowywanych informacji (wspomnianych surowych danych).

Inaczej jest w przypadku arkuszy kalkulacyjnych, które mogą zawierać przetworzone dane, np. wartości średnie lub wyniki analiz statystycznych. Rola bazy danych sprowadza się do przechowywania informacji, a za ich przetwarzanie odpowiada zwykle program **frontonu** lub odpowiedni interfejs użytkownika. Przykładami programów frontonów są strony internetowe wyświetlające informacje wydobyte z bazy danych oraz programy powiązane z danymi bazy danych i umożliwiające użytkownikowi ich przeglądanie.

Baza danych umożliwia też udostępnianie i współdzielenie informacji. Te same dane mogą być współużytkowane przez wielu użytkowników pracujących na jednym komputerze lub przez wielu użytkowników pracujących na wielu komputerach połączonych za pośrednictwem sieci z internetem. Jeśli dealer samochodowy ma swoje oddziały w Poznaniu, Gliwicach i Warszawie, może skonfigurować pojedynczy komputer z bazą danych, która za pośrednictwem sieci będzie udostępniana pracownikom wszystkich oddziałów. Takie rozwiązanie jest nie tylko możliwe, ale też bezpieczne, ponieważ bazy danych mają precyzyjnie definiowane struktury i dodatkowo wymuszają przestrzeganie reguł chroniących zawierane dane. Co więcej, bazy danych umożliwiają dostęp do zawieranych informacji więcej niż jednemu użytkownikowi jednocześnie — ewentualne zmiany wprowadzane w tym trybie są obsługiwane przez system zarządzania bazą danych. Wyobraź sobie chaos, jaki miałby miejsce, gdyby zamiast systemu bazy danych użyto Excela i gdyby arkusz kalkulacyjny był modyfikowany przez dwóch sprzedawców jednocześnie. Naturalnie konieczne byłoby rejestrowanie zmian wprowadzanych przez obu użytkowników, jednak w praktyce utrwalane byłoby tylko działania użytkownika, który zapisał arkusz jako ostatni (wszelkie wcześniejsze zmiany drugiego użytkownika byłyby nadpisywane).

Bazy danych ułatwiają też współdzielenie informacji pomiędzy różnymi systemami (zamiast wykorzystania i konwertowania formatów właściwych dla poszczególnych rozwiązań, np. konkretnego programu, konkretnego producenta czy określonego systemu operacyjnego). Przykładowo, arkusz kalkulacyjny Excela można łatwo odczytać na komputerze klasy PC z systemem operacyjnym Windows i zainstalowanym pakietem Microsoft Office, ale już w systemie UNIX, Macintosh czy Linux podobna operacja stwarza poważne problemy, ponieważ komputery kontrolowane przez te systemy obsługują dane w inny sposób. Nawet na komputerze z systemem Windows niezbędna jest instalacja pakietu Microsoft Office. Zupełnie inaczej jest w przypadku baz danych, które można zainstalować wraz z systemami zarządzania bazami danych na jednym komputerze i w prosty sposób udostępniać pozostałym użytkownikom sieci lokalnej lub internetu.

Rozwiązania alternatywne względem baz danych, a więc pliki tekstowe i arkusze kalkulacyjne, mają jedną wielką zaletę (która w jakimś sensie jest ich słabością) — elastyczność. W plikach tekstowych tak naprawdę nie obowiązują żadne reguły — możesz w dowolnym momencie wstawiać dowolne dane tekstowe. To samo dotyczy (choć w nieco innym wymiarze) arkuszy kalkulacyjnych. Możesz co prawda żądać od użytkowników wpisywania danych w ramach predefiniowanej struktury, ale w praktyce nie masz możliwości wymuszania przestrzegania swoich zaleceń. Zastosowanie bazy danych pozwala ograniczyć dostęp użytkowników do samych danych przy jednoczesnym zakazie modyfikowania gotowej struktury.

Kolejną istotną zaletą baz danych jest bezpieczeństwo. Większość systemów zarządzania bazami danych umożliwia tworzenie „użytkowników" celem definiowania rozmaitych poziomów zabezpieczeń. Zanim ktoś uzyska dostęp do bazy danych, musi się zalogować jako konkretny użytkownik. Każdy użytkownik ma przypisane pewne prawa i ograniczenia. Osoba odpowiedzialna za administrację ma nieograniczone możliwości w zakresie edycji danych, zmiany struktury, dodawania i usuwania użytkowników itd. Pozostali użytkownicy mogą tylko przeglądać dane bez możliwości ich modyfikowania (często można nawet ograniczyć zakres udostępnianych danych). Wiele systemów zarządzania bazami danych zapewnia na tyle szczegółowe poziomy zabezpieczeń, że administrator może bardzo precyzyjnie określać uprawnienia poszczególnych użytkowników. Odpowiednie mechanizmy nie ograniczają się do strategii „wszystko albo nic", która umożliwiałaby albo przyznawanie pełnych uprawnień, albo całkowicie zakazywała dostępu.

Bazy danych są stosowane niemal wszędzie. Przetwarzanie danych było jednym z podstawowych powodów stworzenia pierwszych komputerów i do teraz stanowi ich główne zastosowanie. Niemal każdy człowiek i każda firma w pewnym punkcie swojej działalności korzystają z bazy danych. Tego typu rozwiązania są powszechnie stosowane w komputerach osobistych do składowania lokalnie wykorzystywanych danych oraz w firmowych sieciach komputerowych, gdzie bazy danych umożliwiają współdzielenie informacji — przykładowo, większość sklepów internetowych używa baz danych. Kiedy odwiedzasz duże sklepy internetowe, w większości przypadków wyświetlane informacje na temat oferowanych produktów pochodzą z bazy danych. Zamiast tworzyć każdą stronę ręcznie, właściciele dużych sklepów używają szablonów dla sprzedawanych książek czy płyt CD, a do wydobywania z bazy danych szczegółowych informacji na temat poszczególnych towarów służą odpowiednie wyrażenia języka zapytań SQL. Wyobraź sobie, ile czasu potrzebowałby sklep internetowy Amazon do przygotowania każdej ze swoich stron ręcznie!

Bazy danych doskonale zdają egzamin wszędzie tam, gdzie niezbędne jest przeszukiwanie, sortowanie i regularne aktualizowanie ogromnych ilości danych. Podczas lektury kilku kolejnych rozdziałów przekonasz się, że bazy danych w połączeniu z językiem zapytań SQL umożliwiają uzyskiwanie potrzebnych odpowiedzi we wskazanym porządku.

Stosowane w tej książce systemy zarządzania bazami danych

Bazy danych doskonale nadają się do przechowywania informacji, systemy zarządzania bazami danych zapewniają mechanizmy przeszukiwania tych informacji, a dołączane oprogramowanie zwykle umożliwia też przeglądanie danych. Jednak jak można korzystać z tak reprezentowanych danych poza oprogramowaniem systemu zarządzania bazą danych? System operacyjny, niezależnie od tego, czy jest to Windows, UNIX, Linux czy Macintosh, oferuje rozmaite sposoby zaglądania do systemu zarządzania bazą danych i wydobywania zapisanych tam informacji. Jeśli jednak chcesz udostępniać te dane użytkownikom zewnętrznym, musisz umieścić odpowiedni kod w autonomicznej aplikacji (uruchamianej na komputerze tego użytkownika) lub skonstruować odpowiednią stronę internetową (otwieraną w oknie przeglądarki internetowej użytkownika). Nie jesteś ograniczony do żadnego konkretnego języka programowania (tak naprawdę jedynym wymaganiem jest możliwość łączenia wyrażeń wybranego języka z oprogramowaniem systemu zarządzania bazą danych).

Możesz oczywiście kupić dowolną liczbę dostępnych na rynku i bardzo zróżnicowanych systemów zarządzania bazami danych, jednak celem tej książki jest prezentacja języka SQL, który nie tylko jest standardem (więcej informacji na temat standardów znajdziesz w następnym podrozdziale), ale także jest obsługiwany przez zdecydowaną większość współczesnych systemów zarządzania bazami danych. Istnieją jednak sytuacje, w których standardowe rozwiązania nie umożliwiają realizacji wszystkich oczekiwanych zadań. Istnieją też rozwiązania, które różni producenci systemów zarządzania bazami danych zaimplementowali w odmienny, niespójny sposób. Niniejsza książka szczegółowo opisuje rozwiązania zastosowane w systemach Microsoft Access, Microsoft SQL Server, IBM DB2, MySQL oraz Oracle 10.

Strukturalny język zapytań (SQL)

Pierwszym pytaniem, na jakie należy odpowiedzieć, brzmi: „Czym właściwie jest język SQL i jak można z niego korzystać podczas pracy z bazą danych?”. SQL pełni trzy główne funkcje:

- tworzenia bazy danych i definiowania jej struktury,
- wykonywania na bazie danych zapytań w celu uzyskania danych niezbędnych do wygenerowania odpowiedzi,
- kontrolowania bezpieczeństwa bazy danych.

Proces definiowania struktury bazy danych obejmuje takie działania jak tworzenie nowych tabel i pól bazy danych, budowa reguł dla danych itp. Odpowiednie wyrażenia należą do podjęzyka SQL nazywanego językiem kontroli danych (ang. *Data Control Language — DCL*). Język DCL opisano w dalszej części tego rozdziału, natomiast problematyka wykonywania zapytań na bazie danych zostanie omówiona w następnym punkcie.

I wreszcie język DCL umożliwia zarządzanie zabezpieczeniami bazy danych. Ogólnie, za zapewnienie bezpieczeństwa baz danych odpowiadają ich administratorzy.

Przygotowywanie wyrażeń języka SQL za każdym razem, gdy konieczna jest zmiana struktury lub reguł bezpieczeństwa bazy danych, na pierwszy rzut oka robi wrażenie rozwiązania dość pracochłonnego — i rzeczywiście tak jest! Większość współczesnych systemów baz danych oferuje możliwość wprowadzania zmian za pomocą przyjaznego interfejsu użytkownika (bez konieczności wpisania choćby jednego wiersza w języku SQL).

Wprowadzenie do zapytań języka SQL

Zapytania SQL są najbardziej popularnym zastosowaniem tego języka. Za obsługę zapytań i manipulowanie danymi odpowiada specjalny podjęzyk języka SQL nazywany językiem manipulacji danymi (ang. *Data Manipulation Language — DML*). Język SQL umożliwia przekazywanie zapytań (czyli tak naprawdę pytań) do bazy danych; baza danych generuje wówczas zbiór danych, który stanowi odpowiedź na otrzymane zapytanie. Przykładowo, w przypadku bazy danych zawierającej szczegółowe informacje na temat sprzedawców, transakcji sprzedaży samochodów, typów sprzedawanych aut itd. może zaistnieć konieczność sprawdzenia,

ile samochodów było sprzedawanych przez poszczególnych sprzedawców w kolejnych miesiącach i ile pieniędzy uzyskano z tej sprzedaży. Okazuje się, że można napisać pojedyncze (złożone) wyrażenie języka SQL, które będzie reprezentowało to pytanie i dla którego baza danych wygeneruje zbiór danych stanowiących odpowiedź na otrzymane żądanie. Zapytanie języka SQL składa się z różnych wyrażeń, klauzul i warunków. **Wyrażenie** jest poleceniem lub rozkazem. Przykładowo, wyrażenie może mieć postać: „daj mi jakieś dane". **Klauzula** określa pewne ograniczenia dla danego wyrażenia; każde z tych ograniczeń jest definiowane w formie **warunków**. Przykładowo, zamiast żądania „daj mi jakieś dane" możesz wywołać zapytanie: „daj mi dane tylko dla sprzedaży z maja", gdzie „tylko dla" jest klauzulą określającą interesującą Cię datę. W tym przypadku warunkiem jest „z maja". Jeśli żądane dane nie spełniają tak określonego kryterium (w tym przypadku „z maja"), nie są dla Ciebie interesujące. W kodzie języka SQL takie żądanie mogłoby mieć następującą postać:

```
SELECT ModelSamochodu
FROM SprzedażSamochodów
WHERE DataSprzedażySamochodu BETWEEN '1 maj 2005' AND '31 maj 2005';
```

Wyrażenie `SELECT` mówi systemowi bazy danych, że chcesz z niej wybrać (wyselekcjonować) pewne dane. Następnie wymieniasz dane, które Cię interesują (w tym przypadku jest to tylko pole `ModelSamochodu`). W dalszej części wyrażenia określasz miejsce, z którego mają pochodzić interesujące Cię dane (w tym przypadku z tabeli nazwanej `SprzedażSamochodów`). Na końcu tego wyrażenia zdefiniowałeś warunek. Powyższe wyrażenie stwierdza, że chcesz otrzymać tylko te dane, dla których określone warunki są spełnione. W tym przypadku warunek mówi, że wartość pola daty nazwanego `DataSprzedażySamochodu` musi należeć do przedziału od pierwszego do trzydziestego pierwszego maja 2005 roku. Wiele przykładów kodu języka SQL podobnych do powyższego zostanie omówionych w rozdziale 3., gdzie szczegółowo przeanalizujemy stosowane w tym języku wyrażenia, klauzule i warunki.

Zestawienie języka SQL z pozostałymi językami programowania

Teraz, kiedy już wiesz, do czego można użyć języka SQL, możesz ten język porównać z innymi popularnymi językami programowania. Trzeba przyznać, że język SQL nie ma wiele wspólnego z takimi językami **proceduralnymi** jak C++, Visual Basic, Pascal czy innymi językami programowania trzeciej generacji, które umożliwiają programiście pisanie wyrażeń „krok po kroku" i określanie w ten sposób, co dokładnie należy robić, aby osiągnąć określony cel. Wróćmy do przykładu sprzedaży samochodów — Twoim celem może być wyselekcjonowanie wszystkich informacji na temat transakcji dokonanych w gliwickim salonie firmy w lipcu. W największym uproszczeniu można przyjąć, że Twój język proceduralny musiałby postępować według następujących instrukcji:

1. Wczytaj dane o sprzedaży do pamięci operacyjnej komputera.
2. Wydobądź pojedyncze elementy danych z całego zbioru informacji o sprzedaży.
3. Sprawdź, czy każdy z tych elementów dotyczy lipca i salonu w Gliwicach.
4. Jeśli tak, odpowiednio oznacz ten element.

5. Przejdź do następnego elementu danych i kontynuuj ten proces aż wszystkie elementy zostaną sprawdzone.

6. Jeszcze raz przejrzyj w pętli cały zbiór danych wynikowych i wyświetl elementy zgodne z określonym warunkiem.

SQL jest jednak językiem **deklaratywnym**, co oznacza, że zamiast określać wprost, co należy zrobić, aby otrzymać interesujący Cię wynik, określasz tylko, jaki chcesz uzyskać wynik, a za dobór działań niezbędnych do wygenerowania oczekiwanych przez Ciebie wyników odpowiada sam język. Jeśli w przypadku bazy danych reprezentującej sprzedaż samochodów użyjesz języka SQL, będziesz tylko musiał określić interesujący Cię rezultat, czyli zastosować wyrażenie podobne do poniższego:

`SELECT` wszystkie dane z tabeli sprzedaży `WHERE` transakcje odbywały się w lipcu w salonie mieszczącym się w Gliwicach.

Okazuje się, że język SQL jest stosunkowo łatwy w czytaniu. Odpowiedni kod tego języka tak naprawdę miałby następującą postać:

```
SELECT * FROM TransakcjeSprzedaży WHERE DataSprzedaży = "Lipiec 2005" AND
MiejsceSprzedaży = "Gliwice";
```

Znak gwiazdki oznacza, że należy zwrócić dane ze wszystkich pól danego rekordu.

Znacznie więcej informacji na temat wyrażenia `SELECT` języka SQL znajdziesz w rozdziale 3.

Standardy języka SQL

Podobnie jak w przypadku samych baz danych także oryginalna wersja języka SQL powstała w dużej mierze dzięki staraniom inżynierów firmy IBM. Wielu innych producentów wykorzystało jednak standard tej firmy do opracowania własnych wersji języka zapytań. Istnienie tak wielu różnych dialektów języka SQL z punktu widzenia programisty stanowi niemały problem, mimo że w roku 1986 Amerykański Narodowy Instytut Normalizacji, a w roku 1987 Międzynarodowa Organizacja Normalizacyjna oficjalnie przyjęły standard tego języka. Ten krok pomógł co prawda w minimalizowaniu różnic pomiędzy poszczególnymi dialektami języka SQL, jednak nadal istnieją pewne różnice.

W poniższej tabeli zawarto krótkie podsumowanie rozmaitych standardów i ich rozszerzeń.

Materiał prezentowany w naszej książce koncentruje się na standardach SQL-92, SQL-99 i SQL-2003, ponieważ zdecydowana większość wyspecyfikowanych w nich rozwiązań została zaimplementowana w większości współczesnych relacyjnych systemów zarządzania bazami danych (RDBMS). Prezentowane przykłady działają w większości tych systemów (choć często wymagają nieznacznych modyfikacji). Zdarza się jednak, że różnice pomiędzy technikami realizacji określonych zadań w różnych relacyjnych systemach zarządzania bazami danych uniemożliwiają wykonywanie „standardowego" kodu bez bardzo istotnych zmian — tego typu przypadki występują w tej książce bardzo rzadko i są odpowiednio oznaczane.

Rok	Nazwa	Alternatywna nazwa	Zmiany
1986	SQL-86	SQL-87 (data akceptacji przez ISO)	Pierwsza publikacja standardu organizacji ANSI i ISO
1989	SQL-89		Nieznaczne udoskonalenia oryginalnej wersji
1992	SQL-92	SQL2	Zasadnicze zmiany względem oryginalnego standardu (SQL2 wciąż pozostaje najbardziej popularnym standardem)
1999	SQL-99	SQL3	Aktualizuje standard z roku 1992 przez dodanie nowych sposobów selekcji danych, nowych reguł w zakresie integralności danych i pewnych elementów struktury obiektowej
2003	SQL-2003		Wprowadza obsługę formatu XML i pól z automatycznie generowanymi wartościami

Mimo że standardy stanowią istotne ułatwienie w poszukiwaniu części wspólnej różnych implementacji języka SQL w ramach istniejących relacyjnych systemów zarządzania bazami danych, tak naprawdę ważne jest tylko to, czy poszczególne rozwiązania działają w praktyce. Zamiast bez końca dyskutować o wyższości jednych „standardów" nad innymi, niniejsza książka zawiera informacje, które powinny Ci pomóc w realizacji określonych zadań w rzeczywistym świecie baz danych. Zgodnie z tą deklaracją możemy od razu przejść do kolejnego podrozdziału, w którym pokażę Ci, jak za pomocą języka SQL możesz tworzyć własne bazy danych.

Tworzenie baz danych

Do tej pory zajmowaliśmy się niemal wyłącznie tym, czym jest i kiedy należy używać bazy danych. Teraz przyjrzymy się dokładniej składnikom bazy danych, jej strukturze oraz związaną z tym terminologią. Na końcu przejdziemy od teorii do praktyki — utworzysz swoją pierwszą przykładową bazę danych.

Skoro dysponujesz już podstawową wiedzą, czas omówić technikę budowania efektywnej struktury Twojej bazy danych. Dobry projekt bazy danych upraszcza proces wydobywania danych oraz ogranicza nadmiarowość i narzuty pamięciowe przez unikanie powielania tych samych informacji.

Na końcu tego rozdziału będziesz dysponował w pełni funkcjonalną bazą danych, która nie będzie wymagała żadnych modyfikacji przed podjęciem eksperymentów w następnym rozdziale, gdzie użyjemy języka SQL do wstawiania, aktualizowania i usuwania informacji z bazy danych. Co więcej, po przeczytaniu tego rozdziału będziesz potrafił samodzielnie eksperymentować z procesami tworzenia baz danych. Zanim to jednak nastąpi, musisz pogłębić swoją wiedzę w zakresie organizacji i struktury baz danych.

Organizacja relacyjnych baz danych

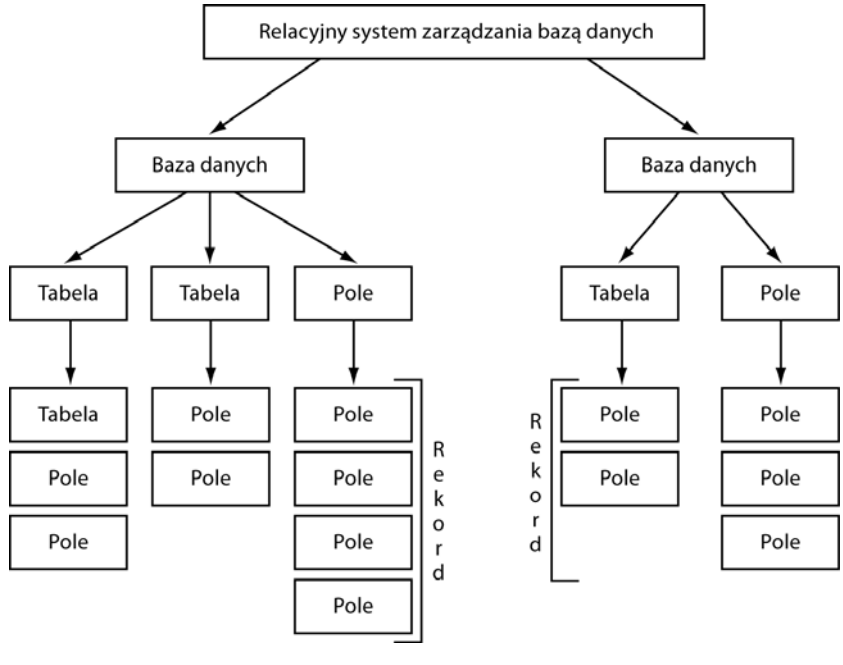
W niniejszym punkcie przeanalizujemy nie tylko organizację systemów baz danych, ale także elementy składające się na ich strukturę. Struktura relacyjnego systemu zarządzania bazą danych obejmuje między innymi bazy danych, tabele i pola. W terminologii baz danych tego typu struktury nazywa się **obiektami**.

System zarządzania bazą danych jest obszernym programem zarządzającym jedną lub większą liczbą baz danych. Każda z nich składa się z tabel, które z kolei zbudowane są z pól. Pole zawiera określony element danych opisujący jakąś „rzecz" — np. wiek osoby, jej wzrost, kolor oczu itp. Tabela zawiera jedno lub wielu pól, zwykle obejmuje też informacje na temat konkretnej „rzeczy" lub przynajmniej dane, które w jakiś sposób są ze sobą powiązane. Przykładowo, dane na temat osoby mogą być składowane w tabeli *Osoba*. Jeśli te informacje dotyczą określonego typu osoby, np. pracownika, jej nazwa może być bardziej ścisła — *Pracownicy*.

Zgodnie z wymową nagłówka, materiał prezentowany w tym punkcie dotyczy relacyjnych baz danych, gdzie kluczowe znaczenie ma określenie **relacyjnych**. Koncepcję relacyjnych baz danych wyjaśnię bardziej szczegółowo za chwilę, na razie wystarczy, jeśli przyjmiemy, że w tego typu strukturze istnieją powiązania (pewnego rodzaju relacje) pomiędzy danymi zawartymi w jednej tabeli a danymi w innej tabeli. Przykładowo, relacja pomiędzy tabelą transakcji sprzedaży samochodów a tabelą sprzedawców może określać, któremu sprzedawcy udało się sprzedać konkretne samochody.

Na rysunku 1.2 przedstawiono podstawową strukturę relacyjnej bazy danych.

Rysunek 1.2.



Na samej górze zaprezentowanego diagramu znajduje się relacyjny system zarządzania bazą danych (RDBMS). Taki system ma postać oprogramowania aplikacyjnego, które zarządza

rozmaitymi bazami danych. W sytuacji przedstawionej na rysunku 1.2 relacyjny system zarządzania bazą danych tworzy dwie bazy danych, ale równie dobrze mógłby tworzyć jedną lub wiele tysięcy baz danych. System RDBMS odpowiada za zapewnienie inteligentnych mechanizmów działających w tle systemu bazy danych. Relacyjny system zarządzania bazą danych wykonuje wszelkie zadania związane z tworzeniem i utrzymywaniem baz danych włącznie z jej wewnętrznymi strukturami. Co więcej, tego typu oprogramowanie odpowiada też za zapewnienie bezpieczeństwa, wykonywanie operacji wstawiania, usuwania i przeszukiwania danych oraz umożliwianie użytkownikom korzystania z systemu bazy danych za pośrednictwem konsoli zarządzania.

Bezpośrednio pod samym relacyjnym systemem zarządzania bazą danych znajdują się zawarte w nim bazy danych. Każda taka baza zawiera kolekcję jednej lub większej liczby tabel. Ściśle mówiąc, Twoja baza danych mogłaby nie zawierać żadnych tabel, jednak wówczas jej utrzymywanie oczywiście nie miałoby sensu! Bazy danych mają charakter niezależnych struktur, co oznacza, że zdarzenia dotyczące tabeli w jednej bazie nie mają wpływu na inne tabele w pozostałych bazach tego samego systemu. Przykładowo, jeśli zechcesz, w każdej ze swoich baz danych możesz stworzyć tabele oznaczone takimi samymi nazwami, a Twoje działania nie spowodują żadnych komplikacji w ramach systemu baz danych. Każda z tworzonych przez Ciebie baz danych otrzymuje własną nazwę lub identyfikator. Sposób obsługi baz danych i tabel w ramach systemu baz danych może być bardzo różny — zależy od konkretnej implementacji oferowanej przez producenta relacyjnego systemu zarządzania bazą danych. Przykładowo, Microsoft Access jednocześnie przetwarza tylko jedną bazę danych, mimo że istnieją techniki łączenia wielu baz danych. Każda baza danych tego systemu jest składowana w osobnym pliku. Pozostałe relacyjne systemy zarządzania bazami danych umożliwiają swoim użytkownikom zarządzanie więcej niż jedną bazą danych z poziomu tej samej konsoli.

W ramach każdej bazy danych istnieje kolekcja tabel zawierających rekordy, które z kolei zawierają właściwe dane. Dobrą analogią występującą w świecie rzeczywistym jest np. rozkład jazdy pociągów lub autobusów. Prosty rozkład jazdy autobusów może mieć postać następującej tabeli:

Początek	Cel	Wyjazd	Przyjazd
Poznań	Gliwice	4:20	10:45
Zaniemyśl	Kórnik	10:12	10:37
Wisła	Katowice	15:30	17:00
Łódź	Łęczyca	11:40	12:35
Gdynia	Sopot	18:05	18:39

Gdyby miała to być rzeczywista tabela w Twojej bazie danych, mógłbyś stworzyć tabelę z odpowiednimi informacjami i nadać jej jakąś oryginalną nazwę, np. `rozkład_jazdy`. Reguły nazywania tabel baz danych z jednej strony są dość elastyczne, z drugiej jednak mogą się nieznacznie różnić w poszczególnych relacyjnych systemach zarządzania bazami danych. Ogólnie, jeśli dana nazwa nie zawiera znaków interpunkcyjnych (z wyłączeniem znaków podkreślenia) i nie jest dłuższa od przyjętego ograniczenia (zwykle równego 128 znakom), system bazy danych nie powinien stwarzać problemów z jej utworzeniem.

W przedstawionym rozkładzie jazdy widać wyraźnie, że odpowiednia tabela składa się z czterech kategorii informacji: początek podróży, cel podróży, godzina wyjazdu i godzina przyjazdu. W terminologii baz danych wymienione kategorie są nazywane **polami**, a każde z tych pól ma własną, unikatową nazwę w ramach tabeli.

Każdy wiersz przedstawionej tabeli rozkładu jazdy zawiera dane właściwe dla jednego aspektu ruchu autobusów. Pierwszy wiersz reprezentuje autobus wyjeżdżający z Poznania o godzinie 4:20 i przyjeżdżający do Gliwic o godzinie 10:45. W bazach danych informacje zawarte w polach takiego wiersza tworzą tzw. **rekord**. Zaprezentowana tabela zawiera pięć takich rekordów. Kolumnę tworzą wszystkie egzemplarze (ze wszystkich rekordów) określonego pola występujące w danej tabeli. Oznacza to, że w przypadku przedstawionego rozkładu jazdy kolumna początku podróży reprezentuje wszystkie dane pola `Początek` dla wszystkich rekordów tej tabeli: `Poznań`, `Zaniemyśl`, `Wisła`, `Łódź` i `Gdynia`.

Podsumowując strukturę relacyjnych baz danych, należy stwierdzić, że relacyjny system zarządzania bazą danych odpowiada za zarządzanie jedną lub większa liczbą baz danych, z których każda zawiera kolekcję jednej lub wielu tabel, a każda tabela zawiera jeden lub wielu rekordów, gdzie każdy rekord jest kolekcją pól.

Przedstawiona do tej pory wiedza stanowi wystarczającą podstawę do wszczęcia eksperymentów z tworzeniem baz danych i tabel za pomocą języka SQL.

Składnia języka SQL

W językach programowania **składnia** jest nie tylko zbiorem reguł, których należy przestrzegać podczas pisania kodu, ale także stosowaną terminologią. Taka składnia bardzo przypomina reguły gramatyczne w językach naturalnych. Przykładowo, reguły gramatyczne języka polskiego mówią, że zdanie powinno się kończyć kropką. Istnieją oczywiście wyjątki od tej reguły. W przypadku zdań pytających zamiast kropki na końcu stawia się znak zapytania. W języku SQL nie ma co prawda zdań, ale są wyrażenia. Każde wyrażenie reprezentuje autonomiczne działanie. Przykładowo, pojedyncze wyrażenie może służyć do wyselekcjonowania określonych danych, do zmiany bazy danych przez dodanie nowej tabeli itd. Wyrażenie powinno się kończyć średnikiem; mimo że wiele systemów baz danych pozwala swoim użytkownikom pomijać ten znak kończący, dobrą praktyką jest konsekwentne stosowanie średnika na końcu każdego wyrażenia.

W niniejszej książce znajdziesz odwołania do trzech kategorii pojęć składniowych: identyfikatorów, literałów i słów kluczowych. **Identyfikator** jest czymś, co jednoznacznie definiuje jakiś element systemu bazy danych w oparciu o taki obiekt jak nazwa bazy danych, nazwa tabeli lub nazwa pola. Jeśli utworzysz bazę danych nazwaną `MojaBazaDanych`, w praktyce zdefiniujesz jej identyfikator (którym będzie właśnie nazwa `MojaBazaDanych`). Identyfikatorem nowo utworzonej tabeli `Sprzedawca` będzie jej nazwa (czyli właśnie `Sprzedawca`) — jeśli będziesz się chciał odwołać do tej tabeli, powinieneś użyć jej identyfikatora:

```
SELECT ImięSprzedawcy
FROM Sprzedawca;
```

Przedstawione wyrażenie wydobywa dane z tabeli `Sprzedawca`. System bazy danych „wie", z której tabeli należy wyselekcjonować żądane dane, ponieważ użyłeś jej identyfikatora (w tym przypadku `Sprzedawca`).

Literał jest faktyczną wartością, np. `120`, `Paweł` lub `10 stycznia 2007`. Przykładowo, jeśli chcesz otrzymać listę wszystkich sprzedawców, którzy mają na imię Bogdan, możesz użyć następującego wyrażenia:

```
SELECT ImięSprzedawcy, NazwiskoSprzedawcy
FROM Sprzedawca
WHERE Imię = 'Bogdan';
```

Powyższe wyrażenie wykorzystuje w swojej instrukcji warunkowej literał `Bogdan`. Jeśli struktura wyrażeń języka SQL wciąż nie jest dla Ciebie jasna, nie przejmuj się — wyrażenia `SELECT` poddamy szczegółowej analizie w rozdziale 3.

Słowo kluczowe jest wyrazem, które ma konkretne znaczenie dla systemu bazy danych przetwarzającego całe wyrażenie. Przykładowo, jeśli powiesz „flob-badob", Twój rozmówca nie będzie miał pojęcia, o co Ci chodzi! Jeśli jednak użyjesz słowa „stop", dla większości mieszkańców Ziemi znaczenie tego wyrazu będzie zrozumiałe. Podobnie dla systemu bazy danych słowo „flob-badob" niczego nie oznacza, ale np. wyraz `SELECT` jest jak najbardziej zrozumiały i ma przypisane określone (specjalne) przesłanie. Użycie tego słowa oznacza: „chcę wyselekcjonować jakieś dane". Każde słowo kluczowe ma przypisane pewne reguły. Jeśli użyjesz słowa `SELECT`, system bazy danych będzie oczekiwał przynajmniej listy danych do wyselekcjonowania i miejsca, gdzie można je znaleźć. Dla słowa `SELECT` istnieją też opcjonalne słowa kluczowe, np. klauzula `WHERE`, za pomocą których można dookreślić rodzaj oczekiwanych danych. Podczas prezentacji każdego nowego słowa kluczowego w tej książce będziesz informowany nie tylko o podstawowych konstrukcjach wymaganych przez system bazy danych, ale też o składnikach opcjonalnych, które można dodać do danego słowa.

Jeśli do tej pory korzystałeś z innych (proceduralnych) języków programowania, być może zastanawia Cię układ kodu. Niektóre języki umożliwiają stosowanie tylko po jednym wyrażeniu na wiersz kodu. Inaczej jest w przypadku języka SQL, którego autorzy przewidzieli możliwość rozciągania wyrażeń na więcej niż jeden wiersz. Przykładowo, oba przedstawione poniżej wyrażenia są prawidłowe:

```
SELECT ModelSamochodu FROM Samochody WHERE MarkaSamochodu = 'Ford';

SELECT ModelSamochodu
FROM Samochody
WHERE MarkaSamochodu = 'Ford';
```

Rozpraszanie kodu w ramach większej liczby wierszy sprawia, że jest on bardziej czytelny (przynajmniej jeśli takiego podziału dokonano w logiczny sposób). W powyższym przykładzie każdą z części wyrażenia `SELECT` umieszczono w osobnym wierszu.

Cóż, wystarczy już tych nudnych rozważań na temat składni. Bardziej szczegółową analizę poszczególnych aspektów składni tego języka znajdziesz w dalszej części książki (odpowiednie rozwiązania będą omawiane na bieżąco). Przejdźmy teraz do kwestii właściwego tworzenia baz danych.

Tworzenie bazy danych

Pierwszym krokiem w pracy z bazą danych jest oczywiście jej utworzenie. Istnieją dwa główne sposoby tworzenia baz danych.

Po pierwsze, wiele relacyjnych systemów zarządzania bazami danych udostępnia estetyczne, przyjazne użytkownikowi interfejsy frontonów, które czynią proces tworzenia nowych baz danych bardzo prostym. W większości przypadków utworzenie nowej bazy wymaga zaledwie kilku kliknięć myszą i wpisania nazwy bazy danych. Odpowiednie interfejsy frontonów oferują takie systemy jak Microsoft Access, Microsoft SQL Server, Oracle czy IBM DB2. System MySQL co prawda nie udostępnia domyślnego frontonu, istnieje jednak mnóstwo darmowych narzędzi tego typu, np. MySQL Control Center.

W przypadku systemu Microsoft Access użycie odpowiedniego programu jest jedynym sposobem tworzenia nowych baz danych. Pozostałe relacyjne systemy zarządzania bazami danych oferują możliwość używania do tego celu języka SQL. W każdym z tych systemów zaimplementowano nieco inny mechanizm wpisywania i wykonywania wyrażeń języka SQL. Przykładowo, system SQL Server zawiera narzędzie Query Analyzer, DB2 zawiera narzędzie Command Center, a MySQL współpracuje między innymi z narzędziem Control Center. Niezależnie od wykorzystywanego narzędzia utworzenie nowej bazy danych wymaga wykonania następującego wyrażenia języka SQL:

```
CREATE DATABASE mojaPierwszaBazaDanych;
```

To naprawdę żadna filozofia! Kiedy zdobędziesz więcej doświadczenia, odkryjesz, że istnieje mnóstwo dodatkowych opcji, z których możesz w dowolny sposób korzystać, jednak na potrzeby naszej książki w zupełności wystarczy standardowa postać wyrażenia `CREATE DATABASE`.

Nowa baza danych została co prawda nazwana `mojaPierwszaBazaDanych`, ale równie dobrze mogłaby się nazywać jakkolwiek inaczej. Istnieją pewne ograniczenia odnośnie stosowanych nazw — np. dotyczące długości. W systemie DB2 długość nazwy jest ograniczana do ośmiu znaków, natomiast w systemie SQL Server ograniczenie zdefiniowano na poziomie 123 znaków. Zawsze bezpieczniejszym rozwiązaniem jest stosowanie w nazwach wyłącznie liter, cyfr i znaku podkreślenia (unikając jakichkolwiek znaków interpunkcyjnych). Przykładowo, nazwa `Moja_bd` jest prawidłowa, ale już nazwa `£$%^moja&&&bd` najprawdopodobniej nie zostanie zaakceptowana przez system zarządzania bazą danych i (chyba się ze mną zgodzisz) jest dość trudna do wypowiedzenia! Cyfry zwykle są akceptowane, ale większość relacyjnych systemów zarządzania bazami danych nie dopuszcza do stosowania cyfr na początku nazw baz danych. I wreszcie warto pamiętać, że nazwa bazy danych musi być (co oczywiste) unikalna w skali całego systemu zarządzania bazą danych. Gdybyś dwie bazy danych nazwał `mojaBD`, relacyjny system zarządzania bazą danych nie mógłby jednoznacznie stwierdzić, do której bazy danych odwołujesz się w swoich wyrażeniach języka SQL.

Co powinieneś zrobić, jeśli uznasz, że należy usunąć jakąś bazę danych? Ponownie okazuje się, że większość relacyjnych systemów zarządzania bazami danych udostępnia wygodną konsolę użytkownika, za pomocą której możesz w prosty sposób usuwać bazy danych; operacja ta jest jednak możliwa także na poziomie języka SQL. Wbrew pozorom do usuwania baz danych nie służy wyrażenie *`delete database`*, zamiast niego masz do dyspozycji wyrażenie `DROP DATABASE`, po którym musi występować nazwa usuwanej bazy danych.

Aby usunąć bazę danych mojaPierwszaBazaDanych, należy wykonać następujące wyrażenie:

```
DROP DATABASE mojaPierwszaBazaDanych;
```

Tego polecenia nie należy jednak stosować w sposób lekkomyślny! Usunięcie bazy danych z relacyjnego systemu zarządzania bazą danych może spowodować trwałą utratę wszystkich zapisanych w niej informacji.

W systemie Oracle zastosowano nieco inne podejście do procesu usuwania baz danych. Zamiast wywoływać wspomniane polecenie DROP DATABASE, należy raz jeszcze utworzyć odpowiednią bazę danych! Jeśli masz już bazę danych nazwaną mojaPierwszaBazaDanych, relacyjny system zarządzania bazą danych usunie ją w odpowiedzi na wykonanie polecania:

```
CREATE DATABASE mojaPierwszaBazaDanych;
```

Także do tego polecenia należy podchodzić bardzo ostrożnie.

Kolejnym krokiem po utworzeniu bazy danych jest dodanie do niej tabel. Zanim jednak będziesz mógł dodawać tabele, musisz się zapoznać z dostępnymi typami danych.

Typy danych

Poza światem techniki komputerowej poszczególne rodzaje informacji dość naturalnie można podzielić na odpowiednie kategorie. Cena towarów w sklepie są traktowane jak dane numeryczne. Jeśli spytasz o drogę z Warszawy do Gdańska, oczekujesz instrukcji słownych w postaci „skręć w prawo…”. W przypadku baz danych pojęcie **typu danych** odnosi się do klasyfikacji różnych rodzajów składowanych informacji niezależnie od tego, czy są to liczby, znaki czy daty. Takie podejście ułatwia systemom baz danych interpretowanie i właściwe przetwarzanie wstawianych wartości. Oznacza to, że podobnie jak w świecie niezwiązanym z bazami danych, różne typy informacji podlegają odpowiedniej kategoryzacji, jednak w tym przypadku proces ten ma zdecydowanie bardziej formalny charakter. Wracając do przykładu rozkłady jazdy autobusów, w poniższej tabeli przedstawiono typy danych składowane w poszczególnych polach:

Pole	Typ danych	Przykład
Początek	Dane znakowe	Poznań, Zaniemyśl
Cel	Dane znakowe	Gliwice, Kórnik
Wyjazd	Czas	4:20, 10:12
Przyjazd	Czas	10:45, 10:37

W tym momencie aż ciśnie się na usta pytanie: „Po co w ogóle stosuje się różne typy danych?”. Dlaczego nie można po prostu wszystkich informacji traktować jak danych tekstowych? Głównym powodem takiego rozwiązania jest efektywność. Ilość wykorzystywanej pamięci jest znacznie mniejsza, a operacje dostępu są znacznie szybsze, jeśli system bazy danych „wie”, z jakimi danymi ma do czynienia. Przykładowo, liczba 243 787 452 może być składowana zaledwie w 4 bajtach pamięci komputerowej. Przechowywanie tej samej liczby w formie tekstu (danych znakowych) zajęłoby aż 9 bajtów.

Typy danych pełnią jeszcze jedną ważną rolę: pozwalają relacyjnemu systemowi zarządzania bazą danych określić zbiór możliwych do wykonania operacji na składowanych informacjach. Przykładowo, w przypadku danych numerycznych wyrażenie `123 + 123` będzie interpretowane jak dodawanie, którego wynikiem będzie liczba `246`. Gdyby jednak oba operandy były danymi tekstowymi, relacyjny system zarządzania bazą danych zinterpretowałby znak plusa jako operator złączenia dwóch łańcuchów znakowych w jeden (w tym przypadku `123123`).

Jakie więc typy danych ma do dyspozycji użytkownik relacyjnego systemu zarządzania bazą danych? Niestety, okazuje się, że dostępne typy różnią się w zależności od producentów tego rodzaju systemów. Problem w tym, że mimo zdefiniowania w specyfikacjach ANSI SQL (takich jak SQL-92, SQL-99 czy SQL-2003) standardowych typów danych, istnieją zasadnicze różnice w kwestii praktycznych rozwiązań zaimplementowanych przez poszczególnych producentów relacyjnych systemów zarządzania bazami danych. Nie wszystko jednak stracone. Wspomniane standardy są obsługiwane w wystarczającym stopniu, aby oprzeć na nich przykłady prezentowane i analizowane w tej książce. Kiedy już opanujesz podstawowe typy danych standardu ANSI SQL, odkrywanie typów danych obsługiwanych przez Twój relacyjny system zarządzania bazą danych nie powinno stanowić większego problemu. Typów właściwych dla Twojego systemu będziesz mógł używać jako uzupełnienia dla omawianych w tym podpunkcie typów podstawowych.

W poniższej tabeli przedstawiono podzbiór najczęściej stosowanych typów danych standardu ANSI SQL wraz z odpowiednimi nazwami stosowanymi w takich relacyjnych systemach zarządzania bazami danych jak SQL Server, IBM DB2 itp.

ANSI SQL	MS Access	SQL Server 2000	IBM DB2	MySQL	Oracle 10
Znakowy	`char`	`char`	`char`	`char`	`char`
Znakowy o zmiennej długości	`varchar`	`varchar`	`varchar`	`varchar`	`varchar`
Regionalny znakowy	`char`	`nchar`	`graphic`	`char`	`nchar`
Regionalny znakowy o zmiennej długości	`varchar`	`nvarchar`	`vargraphic`	`varchar`	`nvarchar`
Całkowitoliczbowy	`number (long integer)`	`int`	`int`	`int`	`int`
Mały całkowitoliczbowy	`number (integer)`	`smallint`	`smallint`	`smallint`	`smallint`
Rzeczywisty	`number (double)`	`real`	`real`	`real`	`real`
Dziesiętny	`number (decimal)`	`decimal`	`decimal`	`decimal`	`decimal`
Daty	`date`	`datetime`	`date`	`date`	`date`
Czasu	`time`	`datetime`	`time`	`time`	`time`

Powyższa tabela zawiera co prawda tylko niewielki podzbiór wszystkich możliwych typów danych dostępnych w poszczególnych relacyjnych systemach zarządzania bazami danych, ale przedstawiony zestaw na tym etapie w zupełności wystarczy. Zwróć uwagę na fakt obsługi typów `nchar` i `nvarchar` w systemie Oracle — typy te są obsługiwane

tylko podczas tworzenia nowej bazy danych i wyznaczania zbioru znaków Unicode (np. `AL16UTF16`); w przeciwnym przypadku system Oracle domyślnie odrzuca zarówno typ `nchar`, jak i typ `nvarchar`.

W poniższej tabeli wymieniono i opisano poszczególne typy danych włącznie z zajmowaną przez nie przybliżoną ilością pamięci oraz przykładami najczęstszych zastosowań. Dla poszczególnych typów danych użyto nazw zgodnych ze standardem ANSI.

Typ danych	Opis	Wymagana pamięć	Przykład
Znakowy	Reprezentuje dane tekstowe. Znakiem może być litera, cyfra lub znak interpunkcyjny. Musisz z góry określić liczbę reprezentowanych znaków. Jeśli rzeczywista liczba znaków będzie mniejsza od zadeklarowanej, relacyjny system zarządzania bazą danych dopełni strukturę dodatkowymi spacjami.	Jeden bajt na każdy zarezerwowany znak	`char(8)` rezerwuje przestrzeń dla ośmiu znaków i zajmuje około 8 bajtów przestrzeni pamięciowej.
Znakowy o zmiennej długości	Typ podobny do znakowego z tą różnicą, że długość danych tekstowych jest zmienna. Ten typ danych wykorzystuje tylko pamięć niezbędną do przechowywania rzeczywiście reprezentowanych znaków.	Jeden bajt na każdy składowany znak	`varchar(8)` rezerwuje przestrzeń dla maksymalnie ośmiu znaków. Składowanie tylko jednego znaku wymaga jednak jedynie 1 bajta pamięci, składowanie dwóch znaków wymaga 2 bajtów itd.; takie pole może zająć maksymalnie 8 bajtów.
Regionalny znakowy	Typ podobny do znakowego z tą różnicą, że wykorzystuje po dwa bajty do reprezentowania każdego ze znaków. Takie rozwiązanie umożliwia reprezentowanie większego zakresu znaków (co jest szczególnie przydatne w przypadku danych tekstowych wyrażonych w języku innym niż angielski).	Dwa bajty na każdy zarezerwowany znak	`nchar(8)` rezerwuje przestrzeń dla ośmiu znaków i zajmuje 16 bajtów pamięci niezależnie od liczby rzeczywiście składowanych znaków.
Regionalny znakowy o zmiennej długości	Typ podobny do znakowego o zmiennej długości z tą różnicą, że wykorzystuje po dwa bajty do reprezentowania każdego ze znaków. Takie rozwiązanie umożliwia reprezentowanie większego zakresu znaków (co jest szczególnie przydatne w przypadku danych tekstowych wyrażonych w języku innym niż angielski).	Dwa bajty na każdy składowany znak	`nvarchar(8)` rezerwuje przestrzeń pamięciową dla ośmiu znaków. Ilość faktycznie wykorzystywanej pamięci zależy od liczby rzeczywiście składowanych znaków.

Typ danych	Opis	Wymagana pamięć	Przykład
Całkowitoliczbowy	Liczby całkowite z przedziału od –2 147 483 648 do 2 147 483 647.	Cztery bajty	`int` zajmuje 4 bajty przestrzeni pamięciowej niezależnie od wielkości reprezentowanej liczby.
Mały całkowitoliczbowy	Liczby całkowite z przedziału od –32 768 do 32 767.	Dwa bajty	`smallint` zajmuje 2 bajty przestrzeni pamięciowej niezależnie od wielkości reprezentowanej liczby.
Rzeczywisty	Liczby zmiennoprzecinkowe z przedziału od –3,40E+38 do 3,40E+38. Liczby tego typu mogą zawierać maksymalnie osiem cyfr po znaku dziesiętnym; np. 87,12342136.	Cztery bajty	`real` zajmuje 4 bajty przestrzeni pamięciowej niezależnie od wielkości reprezentowanej liczby.
Dziesiętny	Liczby zmiennoprzecinkowe, które dodatkowo umożliwiają deklarowanie wartości maksymalnej i ilości cyfr po znaku dziesiętnym. Liczby tego typu muszą się mieścić w przedziale od $-10^{38}+1$ do $10^{38}-1$.	5 – 17 bajtów	`decimal(38,12)` ustawia liczbę, której długość nie przekracza 38 cyfr, z czego 12 cyfr występuje po znaku dziesiętnym.
Daty	Reprezentuje datę.	Cztery bajty	`date`, np. `1 Dec 2006` lub `12/01/2006`. Warto pamiętać o istotnych różnicach w interpretowaniu formatów daty; przykładowo, w Wielkiej Brytanii zapis `12/01/2006` oznacza 12 stycznia 2006, natomiast w Stanach Zjednoczonych ten sam zapis to 1 grudnia 2006 roku.
Czasu	Reprezentuje godzinę.	Trzy bajty	`time`, np. `17:54:45`.

Zwróć uwagę na fakt, że w kolumnie *Wymagana pamięć* uwzględniono tylko przestrzeń potrzebną do przechowywania właściwych danych. Relacyjne systemy zarządzania bazami danych wymagają oczywiście nieznacznie więcej przestrzeni do składowania informacji na temat fizycznego rozmieszczenia poszczególnych danych w pamięci. Tym dodatkowym narzutem nie powinieneś się jednak przejmować, przynajmniej dopóki Twoja baza danych nie jest zbyt obszerna i nie stosuje zaawansowanych mechanizmów. Same wielkości tych narzutów są ściśle uzależnione od stosowanego systemu RDBMS.

Niektóre z wymienionych powyżej typów danych nie wymagają dodatkowych wyjaśnień, pozostałe zasługują na chwilę uwagi. Poniżej szczegółowo omówiono wybrane typy danych (począwszy od typu znakowego).

Znaki

Jeśli chcesz w swojej bazie danych składować tekst, powinieneś użyć znakowego typu danych. Pamiętaj, że pojęcie **łańcucha** odnosi się do jednego lub wielu znaków występujących w ramach jednej struktury. Istnieją cztery możliwe odmiany znakowych typów danych:

- stałej długości,
- zmiennej długości,
- o rozmiarze jednego bajta na znak (typy `char` i `varchar`),
- o rozmiarze dwóch bajtów na znak (typy `nchar` i `nvarchar`).

Najpierw wyjaśnię różnicę pomiędzy typami danych tekstowych stałej i zmiennej długości. Przeanalizuj poniższy fragment kodu:

```
char(127)
```

Jeśli użyjesz tego kodu, relacyjny system zarządzania bazą danych zarezerwuje obszar pamięci potrzebny do składowania 127 znaków. Jeśli umieścisz w tak zadeklarowanej strukturze tylko 10 znaków, pozostałe 117 bajtów zarezerwowanego obszaru zostanie wypełnionych spacjami (będą więc bezużyteczne). Jeśli planujesz stosowanie tylko 10-znakowych danych, powinieneś rozważyć użycie następującej struktury:

```
char(10)
```

Takie rozwiązanie jest jak najbardziej prawidłowe, pod warunkiem jednak, że nigdy nie będziesz próbował wykorzystać więcej niż 10 znaków (np. 127 znaków zgodnie z oryginalną deklaracją).

Zupełnie inaczej będzie interpretowana deklaracja `varchar(127)`, która nie rezerwuje żadnej pamięci, a jedynie sygnalizuje relacyjnemu systemowi zarządzania bazą danych możliwość umieszczenia w przyszłości maksymalnie 127 znaków w danej strukturze. Jeśli więc umieścisz 10 znaków w tak zadeklarowanej strukturze, zostanie wykorzystany wyłącznie obszar pamięci potrzebny do składowania tych 10 znaków. Równie dobrze mógłbyś w tej strukturze umieścić 127 znaków, jednak wówczas zostanie zajęty obszar potrzebny do przechowania 127 znaków.

Na tym etapie może Ci się wydawać, że stosowanie znakowych typów danych stałej długości w ogóle nie jest uzasadnione. Czyż nie można by zawsze stosować typu `varchar`? Dzieje się tak z dwóch powodów. Po pierwsze, wstawianie i aktualizacja danych znakowych stałej długości jest szybsze — różnica nie jest wielka, ale w przypadku niektórych baz danych aktualizacja może dotyczyć dziesiątek tysięcy rekordów na sekundę, a wówczas nawet najmniejsze różnice mogą decydować o ogólnej wydajności systemu. Oznacza to, że wszędzie tam, gdzie przechowujesz niewielką liczbę znaków i gdzie zasadnicze znaczenie ma szybkość przetwarzania, dane znakowe stałej długości spełniają swoją rolę znacznie lepiej.

Po drugie, jeśli przechowujesz tylko po kilka znaków, różnice w poziomach wykorzystania pamięci można uznać za nieistotne.

Kolejna różnica dotyczy typów danych `char-varchar`, które wykorzystują tylko po jednym bajcie dla każdego znaku, i typów danych `nchar-nvarchar`, które wykorzystują po dwa bajdy dla każdego przechowywanego znaku. System 1-bajtowy wywodzi się z oryginalnego zbioru znaków standardu *ASCII* (ang. *American Standard Code for Information Interchange*), który został opracowany we wczesnych latach sześćdziesiątych jako uniwersalny sposób reprezentowania znaków w systemach komputerowych. Komputery pracują wyłącznie na bitach i „potrafią" przetwarzać tylko liczby binarne, zatem w takiej właśnie postaci są przechowywane znaki (gdzie pojedyncza litera, cyfra lub znak interpunkcyjny jest reprezentowany przez liczbę z przedziału od 0 do 255). Przykładowo, litera *A* jest reprezentowana przez wartość numeryczną 65, litera *B* jest reprezentowana przez liczbę 66 itd. O ile pula 255 liczb w zupełności wystarczy do reprezentowania liter alfabetu języka angielskiego, liczb i niektórych znaków interpunkcyjnych, należy pamiętać, że istnieje mnóstwo innych znaków (szczególnie tych stosowanych w obcych językach, także polskim), których z oczywistych względów nie można reprezentować za pomocą przestarzałego standardu ASCII. Aby obejść ten problem, opracowany został zbiór znaków Unicode. W standardzie Unicode do reprezentowania pojedynczego znaku wykorzystuje się dwa bajty, co łącznie daje pulę aż 65 536 możliwych znaków.

Typy danych `char` i `varchar` wykorzystują tradycyjny, jednobajtowy schemat reprezentowania danych zgodny ze standardem ASCII. Typy danych `nchar` i `nvarchar` obsługują dwubajtowy zbiór znaków Unicode. Decyzja o stosowaniu typów `char-varchar` lub `nchar-nvarchar` powinna zależeć wyłącznie od tego, czy chcesz, aby Twoja baza danych obsługiwała języki inne niż angielski. Niezależnie od wybranego przez Ciebie zbioru znaków technika wykorzystywania obu typów w kodzie języka SQL jest z grubsza taka sama, chyba że Twój system bazy danych zawiera specjalnie zaprojektowane mechanizmy obsługi danych tekstowych (inne dla kodowania ASCII, inne dla kodowania Unicode). Oczywistą wadą typów `nchar` i `nvarchar` jest konieczność zajmowania dwukrotnie większego obszaru pamięci do przechowywania tej samej liczby znaków (ponieważ każdy znak jest reprezentowany przez dwa zamiast jednego bajta).

> *Zanim przejdziemy dalej, powinieneś uzyskać pewną wiedzę na temat maksymalnych liczb składowania znaków. Przykładowo, systemy Microsoft Access i MySQL dopuszczają możliwość umieszczania w polu znakowym (tekstowym) najwyżej 255 znaków. Aby obejść to ograniczenie, w systemie Microsoft Access należy użyć typu danych `memo`, który może zawierać maksymalnie 65 535 znaków, czyli wystarczająco wiele dla większości zastosowań. Jeśli korzystasz z systemu MySQL i chcesz przechowywać większe ilości tekstu, powinieneś użyć typu danych `text`, który także daje możliwość przechowywania 65 535 znaków. Ani typ `memo`, ani typ `text` nie wymagają dodatkowego określania maksymalnej liczby składowanych znaków — taki limit jest z góry ustalany przez sam system bazy danych.*

Dane numeryczne

Najprostszym typem danych numerycznych (zarówno z punktu widzenia ich rozumienia, jak i interpretacji w samym systemie) są liczby całkowite, czyli liczby bez części dziesiętnej (i punktu dziesiętnego). Tego rodzaju liczby są szczególnie przydatne w roli unikalnych identyfikatorów dla rekordów (więcej informacji na ten temat znajdziesz w dalszej części tego rozdziału, kiedy będziesz tworzył przykładową bazę danych z kluczem głównym). Liczby

zmiennoprzecinkowe są obciążone błędami zaokrągleń i z tego powodu nie powinny być traktowane jako unikane (a przynajmniej nie powinny występować w roli unikalnych identyfikatorów rekordów). Co więcej, operacje na liczbach całkowitych są mniej kosztowne z perspektywy relacyjnych systemów zarządzania bazami danych, a mniejsze koszty oznaczają szybszą realizację tych operacji. Oznacza to, że liczby całkowite zapewniają większą efektywność wszędzie tam, gdzie stosowanie części ułamkowej nie jest konieczne. Przykładowo, część ułamkowa jest niezbędna, jeśli któreś z pól Twojej bazy danych reprezentuje wartości pieniężne z uwzględnieniem złotych i groszy.

W niniejszej książce będą wykorzystywane dwa typy danych całkowitoliczbowych: `int` i `smallint`. Różnica pomiędzy tą parą jest bardzo prosta — dotyczy maksymalnego rozmiaru reprezentowanych przez nie liczb i, tym samym, ilości bajtów zajmowanych przez odpowiednie pola w pamięci. Liczby typu `smallint` muszą należeć do przedziału od –32 768 do 32 767, natomiast liczby typu `int` muszą należeć do przedziału od –2 147 483 648 do 2 147 483 647.

Ostatnie dwa typy numeryczne wymienione w tabeli mogą dodatkowo reprezentować części dziesiętne liczb: są to `real` i `decimal`. Typ danych `real` umożliwia przechowywanie liczb z przedziału od –3,40E+38 do 3,40E+38, choć w tym względzie istnieją pewne różnice pomiędzy poszczególnymi relacyjnymi systemami zarządzania bazami danych. Warto zwrócić uwagę na sam zapis 3,40E+38, który jest przykładem tzw. notacji naukowej. Znaczenie tak zapisanej liczby jest równoważne ze stwierdzeniem, że jest to liczba 3,4 pomnożona przez liczbę 10 podniesioną do potęgi 38. Przykładowo, liczbę 539 000 000 w notacji naukowej należałoby zapisać jako 5,39E+8. Typ danych `real` jest przydatny wszędzie tam, gdzie trzeba reprezentować wielkie liczby i gdzie nie jest z góry znana ich precyzja. Jeśli dana liczba jest zbyt duża, by można ją było precyzyjnie reprezentować w ramach pola typu `real`, system bazy danych automatycznie konwertuje ją na notację naukową, jednocześnie obniżając jej precyzję (dostosowując do maksymalnego rozmiaru tego typu danych). `real` nie jest typem danych, z którego będziesz korzystał szczególnie często.

Typ danych `decimal` o tyle przypomina opisany przed chwilą typ `real`, że także umożliwia przechowywanie liczb zmiennoprzecinkowych (w przeciwieństwie do typów `int` i `smallint` reprezentujących liczby całkowite). Liczba zmiennoprzecinkowa to taka, która zawiera część dziesiętną (część ułamkową występującą za punktem dziesiętnym) i której punkt dziesiętny nie jest trwale przypisany do określonej pozycji (prawidłowa jest zarówno liczba 123,445, jak i liczba 4455,5). Liczby całkowite (ang. *whole numbers*, *integers*) nie mogą zawierać części ułamkowych, ponieważ w ogóle nie zawierają punktu dziesiętnego. Typ danych `decimal` jest nie tylko bardziej precyzyjny (dokładny), ale też bardziej elastyczny od typu `real`. Zapewne zastanawiasz się, jak to jest możliwe, że liczby typu `decimal` są bardziej precyzyjne od liczb typu `real`. Odpowiedź na to pytanie oczywiście musi być jednoznaczna — poruszamy się przecież w świecie matematyki, nie socjologii, nie ma więc miejsca na próżne debaty. Typ danych `real` może co prawda zawierać wielkie liczby, ale ponieważ wykorzystuje notację naukową, nie ma gwarancji, że wszystkie cyfry rzeczywiście będą reprezentowane (jeśli zostanie przekroczony określony rozmiar).

Aby nasze rozważania były bardziej zrozumiałe, warto przeanalizować odpowiedni przykład. Jeśli liczbę 101 236,8375 spróbujesz umieścić w polu typu `real`, relacyjny system zarządzania bazą danych automatycznie zapisze liczbę 101236,84. Dlaczego cyfry 7 i 5 zniknęły z końca tej liczby? W niektórych systemach RDBMS (np. SQL Server i DB2) liczby typu `real` mogą zawierać tylko po osiem cyfr. Przytoczona liczba składa się z dziesięciu cyfr, zatem relacyjny system zarządzania bazą danych zaokrągla tę liczbę i odrzuca dwie ostatnie cyfry.

Typ danych `decimal` tym różni się od typu `real`, że przechowuje wszystkie cyfry, które zdoła umieścić w przydzielonym sobie obszarze pamięci. Jeśli umieścisz w polu typu `decimal` liczbę przekraczającą możliwości tego typu, relacyjny system zarządzania bazą danych wygeneruje komunikat o błędzie przepełnienia. Oznacza to, że liczby na lewo od punktu dziesiętnego są zawsze prawidłowe. Typ danych `decimal` zaokrągla jednak wszelkie cyfry na prawo od znaku dziesiętnego, jeśli nie znajdzie wystarczającej przestrzeni pamięciowej.

Elastyczność typu danych `decimal` ujawnia się w momencie, w którym określasz zarówno liczbę składowanych cyfr, jak i liczbę cyfr występujących po prawej stronie znaku dziesiętnego. Poniższy kod nakazuje relacyjnemu systemowi zarządzania bazą danych przydział (rezerwację) pamięci dla 38 cyfr, w tym dla 12 cyfr części ułamkowej:

```
decimal(38,12)
```

Oznacza to, że system RDBMS będzie prawidłowo reprezentował liczbę 101 249 986,8365, mimo że do części dziesiętnej (ostatnich czterech cyfr) doda osiem zer dopełniających zadeklarowany i przydzielony obszar dwunastu cyfr.

Maksymalna liczba cyfr składowanych w polach typu `decimal` w większości relacyjnych systemów zarządzania bazami danych wynosi 38. Im więcej cyfr jest składowanych, tym większe są wymagania w zakresie zajmowanej przestrzeni pamięciowej. Łatwo zauważyć, że liczba typu `decimal(9,2)` będzie wymagała 5 bajtów, natomiast liczba typu `decimal(38,2)` zajmie aż 17 bajtów!

Data i godzina

Godzina jest stosunkowo prosta w reprezentacji i obsłudze. Jest składowana w naturalnym formacie *godziny:minuty:sekundy*. Przykładowo, godzina 15:56:22 może być tłumaczona np. na 5:56 po południu (ang. *P.M.*) i 22 sekundy.

Większość relacyjnych systemów zarządzania bazami danych stosuje zegar 24-godzinny, zatem aby zapisać 5:36 po południu, należy użyć wyrażenia 17:36:00.

Niektóre relacyjne systemy zarządzania bazami danych nie oddzielają daty od godziny — zwykle łączą je w taki sposób, aby godzina (w standardowym formacie *godziny:minuty:sekundy*) występowała po dacie. Przykładowo, w niektórych systemach można spotkać datę i godzinę zapisaną w takiej formie: 1 Mar 2006 10:45:55.

O ile format godziny jest standardem w skali międzynarodowej, daty mogą występować w wielu możliwych i bardzo zróżnicowanych odmianach. Przykładowo, wszystkie poniższe formaty są poprawne w Stanach Zjednoczonych lub w Europie:

- 12 Mar 2006
- Mar 12, 2006
- 12 March 2006
- 12/03/2006
- 03/12/2006
- 03-12-2006

Większość relacyjnych systemów zarządzania bazami danych obsługuje pewien wspólny zbiór formatów daty (zgodnych z przedstawionymi powyżej przykładami). Największym problemem jest interpretacja daty, w której miesiąc określono w formie liczby, a nie nazwy — jak w przypadku 03/12/2006. Amerykanie zinterpretują ten zapis jako 12 dzień marca 2006 roku, ale już dla Europejczyków będzie to 3 dzień grudnia 2006 roku — ot, drobna różnica! Jak się okazuje, daty nie są największym źródłem nieporozumień w świecie baz danych!

Z jeszcze gorszym problemem mamy do czynienia w sytuacji, gdy relacyjny system zarządzania bazą danych z jednej strony został skonfigurowany do przetwarzania dat w formacie amerykańskim, z drugiej strony uzyskuje dostęp do informacji zawartych w bazie danych wykorzystującej format stosowany np. w Wielkiej Brytanii. Takie przypadki wcale nie są rzadkie w firmach, które mają swoje centrale w Stanach Zjednoczonych i oddziały w rozsiane po całym świecie.

Wszędzie tam, gdzie jest to możliwe, należy unikać formatu liczbowego (np. 12/03/2006) i zamiast niego stosować nazwy miesięcy lub przynajmniej ich skróty (np. 12 mar 2006). Niestety, wiele relacyjnych systemów zarządzania bazami danych zwraca daty właśnie w formacie 12/03/2006, nawet jeśli użytkownik wpisał w formularzu lub w kodzie języka SQL datę jako 12 mar 2006. W takim przypadku należy się posłużyć specjalnymi poleceniami formatowania (patrz rozdział 5.), które stanowią skuteczne obejście tego problemu. Zawsze warto też się upewnić co do formatu stosowanego w wykorzystywanym przez nas serwerze RDBMS.

Po tym krótkim wprowadzeniu do typów danych, możemy przejść do czegoś znacznie bardziej interesującego — tworzenia tabel!

Tworzenie, modyfikowanie i usuwanie tabel

W niniejszym podrozdziale zostaną omówione podstawowe techniki tworzenia tabel za pomocą języka SQL. Dowiesz się, jak tworzyć nowe tabele, jak modyfikować tabele już istniejące i wreszcie jak usuwać tabele, których już nie potrzebujesz. Po uzyskaniu odpowiedniej wiedzy będziesz mógł bez trudu utworzyć tabele dla przykładowej bazy danych (omawianej w dalszej części tej książki).

Tworzenie tabeli

Do tworzenia tabel służy wyrażenie `CREATE TABLE` języka SQL. Najprostszy sposób tworzenia tabel wymaga podania ich nazw oraz zdefiniowania odpowiednich kolumn (wraz z typami danych).

> *Więcej zaawansowanych opcji związanych z tabelami i ograniczeniami omówiono w rozdziale 4.*

Podstawową składnię wyrażenia tworzącego tabelę przedstawiono poniżej:

```
CREATE TABLE nazwa_tabeli
(
    nazwa_kolumny typ_danych_kolumny
)
```

CREATE TABLE jest słowem kluczowym wskazującym systemowi bazy danych, co chcesz zrobić — w tym przypadku chodzi oczywiście o utworzenie nowej tabeli. Bezpośrednio za tym słowem kluczowym należy użyć unikalnej nazwy lub identyfikatora nowej tabeli. Dalej (w nawiasach klamrowych) powinna się znaleźć lista definiująca poszczególne kolumny tabeli wraz z właściwymi typami danych. Składnia wyrażenia CREATE TABLE wyda Ci się prostsza, jeśli przeanalizujesz odpowiedni przykład.

Poniższy fragment kodu języka SQL tworzy tabelę w oparciu o przedstawiony wcześniej przykład rozkładu jazdy:

```
CREATE TABLE Rozkład_jazdy
(
    miejsce_wyjazdu varchar(75),
    miejsce_przyjazdu varchar(75),
    godzina_wyjazdu time,
    godzina_przyjazdu time
);
```

Microsoft SQL Server nie obsługuje typu danych `time`*, zatem będziesz musiał zmienić ten typ na* `datetime`*. Jeśli używasz systemu Oracle, będziesz musiał zmienić typ danych* `time` *na* `date`*, ponieważ typ* `date` *w tym systemie reprezentuje zarówno datę, jak i godzinę.*

Podczas analizy tego i wszystkich kolejnych przykładów prezentowanych w tej książce powinieneś uruchomić odpowiednie narzędzie swojego relacyjnego systemu zarządzania bazami danych, które umożliwi Ci napisanie i wykonanie prezentowanego kodu SQL. Szczegóły związane z instalacją tego typu narzędzi znajdziesz w dodatku B na końcu książki.

Przeanalizujmy teraz ten kod wiersz po wierszu. W pierwszej kolejności należy określić, że tworzona tabela ma się nazywać Rozkład_jazdy:

```
CREATE TABLE Rozkład_jazdy
```

Następnie (w nawiasach klamrowych) określasz cztery pola składające się na każdy z rekordów nowej tabeli. Definicja każdego pola wymaga zadeklarowania nazwy identyfikującej i typu danych:

```
(
    miejsce_wyjazdu varchar(75),
    miejsce_przyjazdu varchar(75),
    godzina_wyjazdu time,
    godzina_przyjazdu time
)
```

Definicje poszczególnych pól muszą być oddzielone przecinkami. Zwróć uwagę na elegancki układ przedstawionego kodu języka SQL, w którym wyrażenie CREATE TABLE i definicje poszczególnych pól znajdują się w osobnych wierszach. Takie rozmieszczanie elementów zapytań nie jest konieczne, ale czyni kod znacznie bardziej czytelnym i — tym samym —

ułatwia odnajdywanie ewentualnych błędów w sytuacji, gdy wynik wykonywania zapytań jest niezgodny z oczekiwaniami programisty.

Jak widać, tworzenie tabel jest bardzo proste. Tego typu wyrażenia mogą jednak być znacznie bardziej skomplikowane (patrz rozdział 4.). W następnym punkcie zajmiemy się technikami modyfikowania struktury tabel. Przypuśćmy, że chcesz dodać nowe pole, usunąć pole istniejące lub dokonać innej zmiany w strukturze utworzonej wcześniej tabeli. Język SQL oferuje na szczęście bogaty zestaw konstrukcji składniowych w tym zakresie — w szczególności wyrażenie `ALTER TABLE`.

Modyfikowanie istniejącej tabeli

Kluczem do modyfikowania istniejących tabel jest wyrażenie `ALTER TABLE`. Za jego pomocą można dodawać i usuwać kolumny tabeli. Standard ANSI SQL nie dopuszcza jednak możliwości stosowania wyrażeń `ALTER TABLE` do zmiany oryginalnych typów danych istniejących kolumn. Mimo to w wielu relacyjnych systemach zarządzania bazami danych zaimplementowano rozszerzone wersje wyrażenia `ALTER TABLE`, które obsługują dodatkowe mechanizmy modyfikowania definicji kolumn.

Aby dodać nową kolumnę, użyj przedstawionej poniżej podstawowej składni:

```
ALTER TABLE nazwa_tabeli
  ADD nazwa_pola typ_danych
```

`ALTER TABLE` jest słowem kluczowym, które sygnalizuje systemowi bazy danych konieczność wykonania określonej operacji. Po samym słowie kluczowym `ALTER TABLE` należy podać nawę modyfikowanej tabeli. I wreszcie przedstawione powyżej wyrażenie wskazuje, że chcesz dodać nową kolumnę (bezpośrednio po słowie kluczowym `ADD` podałeś jej nazwę i typ danych — forma deklarowania dodawanej kolumny nie różni się od tej stosowanej podczas tworzenia tabeli).

Aby usunąć istniejącą kolumnę, należy się posłużyć niemal identyczną składnią — z tą różnicą, że tym razem wskazujesz systemowi bazy danych na konieczność usunięcia kolumny i podajesz wyłącznie jej nazwę (bez typu reprezentowanych przez nią danych):

```
ALTER TABLE nazwa_tabeli
  DROP COLUMN nazwa_pola
```

Myślę, że kilka przykładów wystarczy do pełnego wyjaśnienia tego zagadnienia. Aby dodać do tabeli `Rozkład_jazdy` kolumnę nazwaną `kursuje_w_weekendy` typu `char(1)`, należy użyć następującego wyrażenia języka SQL:

```
ALTER TABLE Rozkład_jazdy
  ADD kursuje_w_weekendy char(1)
```

Aby usunąć tę samą kolumnę, należy wykonać wyrażenie przedstawione poniżej:

```
ALTER TABLE Rozkład_jazdy
  DROP COLUMN kursuje_w_weekendy
```

Wyrażenie `DROP COLUMN` nie jest obsługiwane w systemie DB2 firmy IBM.

Pamiętaj, że podobnie jak w przypadku operacji usuwania tabeli także usuwanie kolumn najczęściej oznacza trwałe usunięcie zawartych w nich danych. Wyrażenia DROP COLUMN należy więc używać bardzo ostrożnie!

W kolejnym punkcie zostanie omówiona technika usuwania istniejących tabel za pomocą języka SQL.

Usuwanie istniejącej tabeli

Prawdopodobnie znasz już pewien wzorzec usuwania struktur za pomocą wyrażeń języka SQL — słusznie się domyślasz, że usuwanie tabel jest możliwe dzięki poleceniu DROP TABLE. Podstawową składnię tego polecenia przedstawiono poniżej:

```
DROP TABLE nazwa_tabeli
```

Aby usunąć tabelę Rozkład_jazdy, należy wykonać następujące wyrażenie:

```
DROP TABLE Rozkład_jazdy
```

W niniejszym podrozdziale zaledwie otarłeś się o szeroki temat dodawania i modyfikowania tabel oraz kolumn baz danych. W rozdziale 4. opisano potencjalne komplikacje związane z procesem usuwania tabel w sytuacji, gdy zawarte tam dane są wykorzystywane (wskazywane) przez inne tabele.

Powinieneś już dysponować podstawową wiedzą wystarczającą do tworzenia użytecznych baz danych i tabel. Materiał zawarty w ostatnim podrozdziale tego rozdziału przeprowadzi Cię przez proces tworzenia przykładowej bazy danych, z której będziesz wielokrotnie korzystał w kolejnych rozdziałach tej książki. Najpierw jednak musisz się zapoznać z technikami właściwego projektowania baz danych i tworzenia efektywnych rozwiązań w tym zakresie.

Dobry projekt bazy danych

W podrozdziale przeanalizuję kilka podstawowych reguł i koncepcji, które mogą Ci pomóc w przygotowywaniu efektywnych i prawidłowo zaprojektowanych baz danych. O ile rozdział 4. zawiera bardziej szczegółową analizę tych zagadnień, w niniejszym podrozdziale przedstawiono materiał podstawowy, który na początek powinien Ci w zupełności wystarczyć. Swoją pracę zawsze powinieneś zaczynać od wykonania pierwszego i najważniejszego kroku — rozważenia, do czego potrzebujesz bazy danych.

Gromadzenie i analiza rzeczywistych potrzeb związanych z danymi

Zanim utworzysz swoją bazę danych i napiszesz pierwsze wyrażenie języka SQL, w pierwszej kolejności musisz spokojnie przemyśleć to, dlaczego w ogóle chcesz utworzyć bazę danych. Nie chodzi oczywiście o uzasadnianie tego typu decyzji tym, że ktoś zapłacił Ci mnóstwo forsy za realizację takiego projektu (choć nie twierdzę, że przesłanki finansowe należy

lekceważyć)! Powinieneś raczej zadać sobie (lub docelowemu odbiorcy bazy danych) pytanie o rodzaj składowanych danych i mechanizmy korzystania z tak zapisanych informacji. Przykładowo, wyobraź sobie, że postanowiłeś założyć klub miłośników kina i że jako osoba zafascynowana nowymi technologiami zdecydowałeś o zastosowaniu nowocześniejszej formy składowania szczegółowych danych o członkach tego klubu niż sterta teczek w szufladzie! Szybko doszedłeś do przekonania, że prowadzenie takiego klubu będzie znacznie prostsze, jeśli wykorzystasz bazę danych. Na tym etapie powinieneś się dobrze zastanowić, do czego będziesz używał swojej bazy danych i jakie informacje powinny się w niej znaleźć. W przypadku bazy danych o członkach klubu naturalnym rozwiązaniem będzie przechowywanie szczegółowych informacji o miłośnikach filmów. Być może powinieneś mieć możliwość śledzenia popularności klubu przez rejestrowanie szczegółowych danych o frekwencji na organizowanych spotkaniach. Na dobry początek powinieneś zapisać listę wszystkich informacji, które w przyszłości będziesz chciał przechowywać w swojej bazie danych.

Przypuśćmy, że chcesz mieć możliwość kontaktowania się z poszczególnymi członkami klubu za pośrednictwem tradycyjnej poczty lub poczty elektronicznej. Zapewne chciałbyś też wysyłać im życzenia z okazji urodzin (jeśli tak, bardzo dobrze to o Tobie świadczy!). I wreszcie musisz mieć możliwość sprawdzania, czy członkowie klubu opłacają roczne składki — powinieneś więc wiedzieć, kiedy poszczególni członkowie podpisali swoje deklaracje członkowskie. Poniższa lista podsumowuje informacje, które chcesz składować w swojej bazie:

- Imię i nazwisko
- Data urodzenia
- Adres pocztowy
- Adres poczty elektronicznej
- Data przystąpienia do klubu

Celem organizowanych spotkań jest śledzenie ich popularności w poszczególnych miejscowościach oraz analiza zaangażowania poszczególnych członków. Poniższa lista obejmuje dane niezbędne do realizacji tych dwóch celów:

- Data spotkania
- Miejsce spotkania
- Lista obecnych członków klubu

Wiesz już, jakiego rodzaju informacje chcesz przechowywać w swojej bazie danych, następnym krokiem jest logiczny podział tych danych, który wskaże właściwą strukturę tabel.

Logiczny podział danych

Na razie nie musisz się martwić o nazwy tabel i kolumn ani o typy danych — na tym etapie powinieneś raczej określić przyszłą strukturę tabel w ramach tworzonej bazy danych.

W pierwszym odruchu możesz zdecydować o wrzuceniu wszystkich wyodrębnionych informacji do jednej wielkiej tabeli, która musiałaby zawierać następujące kolumny:

- Imię i nazwisko
- Data urodzenia
- Adres pocztowy
- Adres poczty elektronicznej
- Data przystąpienia do klubu
- Data spotkania
- Miejsce spotkania
- Uczestnictwo członka klubu w spotkaniu

Dla przykładowych danych opisujących troje członków klubu i jedno spotkanie należałoby utworzyć następującą tabelę z trzema rekordami:

Imię	Data urodzenia	Adres	Email	Data przystąpienia	Data spotkania	Lokalizacja	Czy członek klubu uczestniczył w spotkaniu?
Marcin	27 lutego 1972	ul. Kwiatowa 4/1, Poznań	marcin@cokolwiek.com	10 stycznia 2005	30 marca 2005	Górna Wilda, Poznań	T
Janina	12 grudnia 1967	ul. Chopina 2, Gliwice	Janina@serwer.net	12 stycznia 2005	30 marca 2005	Górna Wilda, Poznań	N
Kasia	22 maja 1980	ul. Długa 12, Puszczykowo	kasia@mail.pl	23 stycznia 2005	30 marca 2005	Górna Wilda, Poznań	T

Wygląda nieźle (przynajmniej jak na początek). Takie rozwiązanie jest jednak dalekie od ideału. W jaki sposób byłyby reprezentowane informacje o więcej niż jednym spotkaniu? Jedną z możliwości jest po prostu tworzenie nowego rekordu dla każdego spotkania, czyli czegoś podobnego do poniższej tabeli:

Imię	Data urodzenia	Adres	Email	Data przystąpienia	Data spotkania	Lokalizacja	Czy członek klubu uczestniczył w spotkaniu?
Marcin	27 lutego 1972	ul. Kwiatowa 4/1, Poznań	marcin@cokolwiek.com	10 stycznia 2005	30 marca 2005	Górna Wilda, Poznań	T
Marcin	27 lutego 1972	ul. Kwiatowa 4/1, Poznań	marcin@cokolwiek.com	10 stycznia 2005	28 kwietnia 2005	Jeżyce, Poznań	T
Janina	12 grudnia 1967	ul. Chopina 2, Gliwice	Janina@serwer.net	12 stycznia 2005	30 marca 2005	Górna Wilda, Poznań	N
Janina	12 grudnia 1967	ul. Chopina 2, Gliwice	Janina@serwer.net	12 stycznia 2005	28 kwietnia 2005	Jeżyce, Poznań	T
Kasia	22 maja 1980	ul. Długa 12, Puszczykowo	kasia@mail.pl	23 stycznia 2005	30 marca 2005	Górna Wilda, Poznań	T
Kasia	22 maja 1980	ul. Długa 12, Puszczykowo	kasia@mail.pl	23 stycznia 2005	28 kwietnia 2005	Jeżyce, Poznań	T

Taka metoda reprezentowania danych o członkach klubu i organizowanych spotkaniach co prawda zdaje egzamin, ale jest nieefektywna i trudna w konserwacji (szczególnie jeśli liczba członków i spotkań będzie się zwiększała).

W czym właściwie tkwi problem?

Po pierwsze, mamy tu do czynienia z niepotrzebnym powielaniem danych. Za każdym razem, gdy będziesz chciał umieścić w tak zaprojektowanej tabeli informacje o kolejnym spotkaniu, będziesz musiał raz jeszcze powtórzyć szczegółowe dane o wszystkich członkach klubu. Oznacza to, że Twoja baza danych będzie się bardzo szybko rozrastała. Co więcej, każda operacja aktualizacji danych będzie wymagała znacznych nakładów. Przykładowo, jeśli zmieni się adres zamieszkania Janiny, będziesz musiał zaktualizować wszystkie powiązane z nią rekordy (zamiast jednego zaktualizujesz po jednym rekordzie dla każdego ze spotkań). Kolejnym problemem będzie obsługa operacji wydobycia danych o członku klubu. Wybór jednego z rekordów dotyczących odpowiedniej osoby będzie wyjątkowo trudny; możesz przecież wydobywać potrzebne dane z więcej niż jednego rekordu.

Po ponownym przeanalizowaniu oryginalnej organizacji bazy danych zapewne dojdziesz do wniosku, że należałoby zagwarantować możliwość składowania szczegółowych danych na temat więcej niż jednego spotkania, zatem zamiast tworzyć więcej niż po jednym rekordzie dla poszczególnych członków klubu powinieneś utworzyć nową kolumnę dla każdego spotkania — takie rozwiązanie wiązałoby się z koniecznością zdefiniowania w ramach jednej tabeli następujących pól:

- Imię i nazwisko
- Data urodzenia
- Adres pocztowy
- Adres poczty elektronicznej
- Data przystąpienia do klubu
- Data spotkania 1
- Miejsce spotkania 1
- Uczestnictwo członka klubu w spotkaniu 1
- Data spotkania 2
- Miejsce spotkania 2
- Uczestnictwo członka klubu w spotkaniu 2
- Data spotkania 3
- Miejsce spotkania 3
- Uczestnictwo członka klubu w spotkaniu 3

Taka organizacja kolumn przynajmniej w jakimś stopniu ograniczy problem powielania danych, ale spowoduje też mniejszą elastyczność całej struktury. Przypuśćmy, że chcesz przechowywać rekordy np. o ostatnich dziesięciu spotkaniach. Aby ten cel osiągnąć, będziesz po-

trzebował aż 30 kolumn. Co więcej, za każdym razem, gdy będziesz chciał dodać informacje o nowym spotkaniu, będziesz musiał ponownie zaprojektować swoją bazę danych (przebudować jej strukturę). Taka organizacja znacznie utrudnia pisanie wyrażeń języka SQL wydobywających informacje z Twojej bazy danych.

Najwyższy czas, abyś podzielił dane na logiczne części. W tym przypadku gromadzisz informacje o dwóch różnych rodzajach obiektów — członkach klubu i ich uczestnictwie w zebraniach. Istnieje oczywista relacja pomiędzy tymi obiektami. Bez członków nie byłoby przecież mowy o klubowych spotkaniach! Nietrudno się więc domyślić, że dane powinny zostać podzielone pomiędzy dwie tabele, z których jedna będzie zawierała informacje o klubowiczach, a druga o ich spotkaniach.

W tabeli zawierającej dane o członkach klubu powinny się znaleźć następujące informacje:

- Imię i nazwisko
- Data urodzenia
- Adres pocztowy
- Adres poczty elektronicznej
- Data przystąpienia do klubu

Tabela opisująca szczegóły zorganizowanych spotkań powinna obejmować:

- Imię i nazwisko
- Data spotkania
- Miejsce spotkania
- Uczestnictwo członka klubu w spotkaniu

Tabela członków klubu od tej pory będzie miała następującą postać:

Imię	Data urodzenia	Adres	Email	Data przystąpienia
Marcin	27 lutego 1972	ul. Kwiatowa 4/1, Poznań	marcin@cokolwiek.com	10 stycznia 2005
Janina	12 grudnia 1967	ul. Chopina 2, Gliwice	Janina@serwer.net	12 stycznia 2005
Kasia	22 maja 1980	ul. Długa 12, Puszczykowo	kasia@mail.pl	23 stycznia 2005

Poniżej przedstawiono zawartość nowej tabeli przechowującej informacje o uczestnictwie członków klubu w jego zebraniach:

Imię	Data spotkania	Lokalizacja	Czy członek klubu uczestniczył w spotkaniu?
Marcin	30 marca 2005	Górna Wilda, Poznań	T
Marcin	28 kwietnia 2005	Jeżyce, Poznań	T

Imię	Data spotkania	Lokalizacja	Czy członek klubu uczestniczył w spotkaniu?
Janina	30 marca 2005	Górna Wilda, Poznań	N
Janina	28 kwietnia 2005	Jeżyce, Poznań	T
Kasia	30 marca 2005	Górna Wilda, Poznań	T
Kasia	28 kwietnia 2005	Jeżyce, Poznań	T

Rozdzielenie szczegółowych danych o członkach od szczegółów dotyczących ich spotkań pomiędzy dwie tabele pozwala w znacznej mierze wyeliminować nadmiarowość danych. Informacje o klubowiczach są składowane tylko raz, a jedynymi powtarzającymi się danymi są ich imiona, które stanowią logiczny łącznik pomiędzy obiema tabelami. Tak naprawdę to dopiero początek procesu projektowania tabel bazy danych — kolejnym istotnym krokiem jest zdefiniowanie właściwych typów danych.

Dobór właściwych typów danych

Po opracowaniu ogólnego projektu tabeli kolejnym krokiem jest wybór typu danych dla każdego z jej pól. W niektórych przypadkach proces ten jest dość oczywisty. Przykładowo, składowanie imienia członka klubu w polu numerycznym nie miałoby najmniejszego sensu!

Istnieją jednak sytuacje, w których decyzja odnośnie typu danych taka oczywista już nie jest. Przykładowo, chociaż numer telefonu jest liczbą, istnieje co najmniej kilka powodów, by składować takie numery w polach znakowych. Po pierwsze, numery telefonów bardzo rzadko są przedmiotem działań matematycznych. Po drugie, zdarza się, że numery telefonów rozpoczynają się od jednego lub wielu zer. Przykładowo, numer 0618778377 w polu numerycznym byłby reprezentowany jako 618778377; relacyjny system zarządzania bazą danych usunie pierwsze zero, ponieważ z punktu widzenia wartości numerycznych i tak niczego ono nie zmienia, choć jest ważne podczas wybierania numeru.

Podczas wyboru właściwych typów danych należy brać pod uwagę następujące czynniki:

- **Zastosowanie danych.** Czy dane mają być przedmiotem działań matematycznych? Czy dane będą reprezentowały datę lub godzinę? Czy może dane będą służyły do prezentacji zwykłych informacji tekstowych?
- **Rozmiar danych.** Wybierz taki typ danych, który zapewni możliwość składowania największych przewidywanych wartości. Przykładowo, jeśli podejrzewasz, że w Twojej bazie mogą być reprezentowane osoby, których imię i nazwisko osiąga długość 100 znaków, Twoje pole tekstowe musi obsługiwać taką długość. Jeśli definiujesz dane numeryczne, upewnij się, że pole to będzie mogło reprezentować największą możliwą liczbę.
- **Poprawność składowanych informacji.** Przykładowo, gdybyś użył typu danych `integer` do przechowywania wartości pieniężnych, stracisz możliwość reprezentowania części ułamkowych (np. groszy). Typ danych `integer` składuje wartość 2,99 zł jako 2. Nawet w przypadku typów obsługujących części ułamkowe może się okazać, że system bazy danych zaokrąglił przechowywane wartości w górę lub w dół, doprowadzając

ostatecznie do otrzymania niewłaściwych wyników — dotyczy to szczególnie typu danych `real`. Jeśli Twój relacyjny system zarządzania bazą danych obsługuje specjalny typ walutowy, stosuj ten typ we wszystkich polach, które tego wymagają; w przeciwnym razie użyj w miarę bezpiecznego typu `DECIMAL(10,2)`.

- **Znaki spoza języka angielskiego.** Jeśli przewidujesz, że w Twoich polach tekstowych znajdą się znaki spoza alfabetu języka angielskiego, użyj albo typu `nchar`, albo typu `nvarchar`.

Ogólnie, wybór typów danych w wielu sytuacjach wydaje się wręcz oczywisty. Przykładowo, we wspominanej już tabeli `SzczegoloweDaneCzlonkow` bazy danych klubu miłośników kina dobór nazw i typów danych dla pól może być następujący:

Nazwa pola	Typ danych
`Imie`	`nvarchar(75)`
`DataUrodzenia`	`date`
`Adres`	`nvarchar(200)`
`Email`	`nvarchar(200)`
`DataPrzystapienia`	`date`

Jak widać, w przypadku prezentowanych pól tabeli `SzczegoloweDaneCzlonkow` wybrane typy nie powinny budzić najmniejszych wątpliwości. Imię, adres pocztowy i adres poczty elektronicznej są typowymi informacjami tekstowymi, zatem dla każdego z tych pól można wybrać typ `nvarchar`. Podobnie dla pola reprezentującego datę urodzenia wybrano typ `date`. Nieco gorzej jest z wyborem rozmiarów pól, gdzie deklarowane wartości często są efektem samych domysłów. Co prawda nie możesz mieć stuprocentowej pewności, że do Twojego klubu nie zapisze się osoba z imieniem dłuższym niż 75-znakowe, jednak takie przypadki zdarzają się na tyle rzadko, że przyjęta długość wydaje się rozsądnym kompromisem. Dla bezpieczeństwa zawsze należy deklarować długości nieco większe od oczekiwanych.

W poniższej tabeli przedstawiono typy danych dla tabeli `Frekwencja`:

Nazwa pola	Typ danych
`Imie`	`nvarchar(75)`
`DataSpotkania`	`date`
`Lokalizacja`	`nvarchar(200)`
`UczestnictwoCzlonka`	`char(1)`

Także tym razem wybór właściwych typów danych okazał się dość prosty — pola `Imie` i `Lokalizacja` zawierają dane tekstowe, zatem zadeklarowano je jako pola typu znakowego (w tym przypadku `nvarchar`). Pole `DataSpotkania` będzie oczywiście reprezentowało datę, zatem wybór typu `date` był zupełnie naturalny. Pole `UczestnictwoCzlonka` jest o tyle nietypowe, że w założeniu ma służyć do reprezentowania ewentualnego uczestnictwa danego członka klubu w danym spotkaniu. Odpowiedni efekt osiągnięto, stosując pojedynczą literę `T` dla odpowiedzi

tak oraz literę `N` dla odpowiedzi *nie*. W tym przypadku znakowy typ danych jest najlepszy, ponieważ planowane jest składowanie w tym polu tylko jednej litery. Rozwiązanie oparte na typie `char` i określeniu jednoznakowej długości pola zapewnia właściwą efektywność.

Stosowanie klucza głównego

Klucz główny jest polem lub polami, które unikalnie identyfikują rekord spośród innych rekordów w ramach tej samej tabeli bazy danych. Wracając do przykładu klubu kinomanów, jako jego twórca możesz przyjąć, że jesteś też jego pierwszym członkiem — na tym etapie przechowywanie rekordu bazy danych o członkach jest banalnie proste. Kiedy do Twojego klubu przystąpi kilka kolejnych osób i podejmiesz decyzję o składowaniu ich personaliów (imienia, wieku, adresu itp.) w bazie danych, mechanizm identyfikacji według samych imion będzie spełniał swoje zadanie. Wyobraź sobie jednak, że Twój klub zyskuje popularność znacznie szybciej — może się okazać, że klub po jakimś czasie liczy kilka tysięcy członków. Nagle okazuje się, że ryzyko wystąpienia dwóch, trzech lub jeszcze większej liczby takich samych imion bardzo wzrasta, utrudniając tym samym szansę na wyselekcjonowanie właściwych osób z bazy danych! W takich przypadkach niezbędne jest zastosowanie dodatkowego mechanizmu jednoznacznej identyfikacji. Mógłbyś oczywiście użyć kombinacji imienia i wieku, problem jednak w tym, że wiek członków klubu się zmienia, a zmienny identyfikator oznacza poważne problemy w śledzeniu odwołań pomiędzy tabelami. Nie można też wykluczyć sytuacji, w której dwóch członków klubu będzie miało takie samo imię i ten sam wiek. Mógłbyś też użyć kombinacji imienia i adresu, należy jednak pamiętać, że adresy z jednej strony również mogą się zmieniać, a z drugiej są dość długie, co może mieć negatywny wpływ na efektywność przeszukiwania i sortowania bazy danych.

Być może domyśliłeś się już, że rozwiązaniem tego problemu jest przypisanie każdemu z członków klubu unikalnego identyfikatora, który nie może się powtórzyć w dwóch różnych rekordach tabeli `SzczegoloweDaneCzlonkow`. W przypadku tej tabeli identyfikator może się nazywać `IdentyfikatorCzlonka`. Na tym właśnie polega koncepcja klucza głównego — chodzi wyłącznie o wyznaczenie w ramach tabeli unikalnego identyfikatora. Jeśli wiesz, że `IdentyfikatorCzlonka` równy 1234432 reprezentuje dokładnie jedną osobę, selekcjonując odpowiednie informacje z bazy danych możesz mieć pewność, że uzyskałeś oczekiwany rekord.

Klucze główne są też wykorzystywane do łączenia tabel. Przykładowo, jeśli jedna z Twoich tabel zawiera informacje o członkach klubu kinomana, druga szczegółowo opisuje spotkania zorganizowane w ramach tego klubu, możesz połączyć obie tabele właśnie z użyciem klucza głównego. W takim przypadku tabela `SzczegoloweDaneCzlonkow` może mieć następującą postać:

Nazwa pola	Typ danych
`IdentyfikatorCzlonka`	`integer`
`Imie`	`nvarchar(75)`
`DataUrodzenia`	`date`
`Adres`	`nvarchar(200)`
`Email`	`nvarchar(200)`
`DataPrzystapienia`	`date`

Zwróć uwagę na fakt, że kolumna `IdentyfikatorCzlonka` *jest kluczem głównym tej tabeli.*

Tabela `Frekwencja` mogłaby wówczas wyglądać następująco:

Nazwa pola	Typ danych
`DataSpotkania`	`date`
`Lokalizacja`	`nvarchar(200)`
`UczestnictwoCzlonka`	`char(1)`
`IdentyfikatorCzlonka`	`integer`

Kolumna `IdentyfikatorCzlonka` pełni funkcję **klucza obcego** tej tabeli. Klucz obcy zawiera wartość klucza głównego innej tabeli, stanowi zatem jednoznaczne odwołanie (referencję) do zewnętrznej tabeli. W przypadku klubu zrzeszającego kinomanów pole `IdentyfikatorCzlonka` tabeli `Frekwencja` wskazuje na szczegółowe informacje o odpowiednim członku klubu (na właściwy rekord w tabeli `SzczegoloweDaneCzlonkow`).

Masz już klucz główny dla tabeli `SzczegoloweDaneCzlonkow`, pole `IdentyfikatorCzlonka`, ale co z tabelą `Frekwencja`? Tabela `Frekwencja` ma już swój unikalny identyfikator — kombinację pól `DataSpotkania` i `IdentyfikatorCzlonka`, ponieważ jeden klubowicz nie może uczestniczyć w tym samym spotkaniu więcej niż raz! Stosowanie kombinacji tych pól w roli unikalnego identyfikatora jest więc całkowicie bezpieczne. Przedstawione rozwiązanie jest dobrym przykładem jeszcze jednej istotnej cechy kluczy głównych, które nie muszą się składać z jednej kolumny — mogą być połączeniem większej ich liczby (jak w przypadku tabeli `Frekwencja`). Jeśli zechcesz, możesz oczywiście utworzyć unikalny identyfikator spotkania, jednak w tym przypadku nie jest to konieczne, a zatem oznacza niepotrzebną stratę przestrzeni pamięciowej. Argumentem przemawiającym za stosowaniem takich „sztucznych" identyfikatorów jest wyższa szybkość przeszukiwania danych, ale w większości sytuacji takie rozwiązanie nie znajduje uzasadnienia.

Koncepcja klucza głównego sprowadza się do konieczności stosowania kolumny zawierającej unikalną wartość i odpowiedniego generowania tej wartości. Większość relacyjnych systemów zarządzania bazami danych oferuje jednak możliwość wskazywania kolumny pełniącej tę funkcje i automatycznie zarządza zawartymi tam danymi. W szczególności możesz stosować ograniczenia określające, jakie dokładnie dane powinny się znaleźć w tak zadeklarowanej kolumnie. Przykładowo, odpowiednie ograniczenie może zapobiegać umieszczaniu tej samej wartości klucza obcego w dwóch rekordach tabeli. Przecież celem stosowania kluczy głównych jest właśnie zapewnienie unikalności rekordów. W rozdziale 4. znajdziesz szczegółowe omówienie bardziej zaawansowanych i jednocześnie ciekawszych aspektów kluczy głównych oraz ograniczeń, jednak wiedza nabyta przez Ciebie do tej pory w zupełności wystarczy do utworzenia pełnowartościowej bazy danych. Baza, którą utworzysz w następnym podrozdziale, będzie Ci służyła przez wszystkie kolejne rozdziały tej książki.

Tworzenie przykładowej bazy danych

Przykładowa baza danych jest w pewnym sensie przedłużeniem naszych dotychczasowych rozważań na temat bazy danych klubu kinomanów. Przechowywanie omawianych tabel wymaga utworzenia bazy danych. Możesz oczywiście nadać jej nazwę, np. `Klub Filmowy`, jednak z punktu widzenia celów tej książki nazwa samej bazy danych jest nieistotna. W dodatku B opisano całą procedurę tworzenia pustej, przykładowej bazy danych w takich systemach jak Access, SQL Server, DB2, MySQL czy Oracle. Naturalnym następstwem utworzenia bazy danych `Klub Filmowy` jest przystąpienie do jej wypełniania tabelami.

W największym uproszczeniu można przyjąć, że baza danych wykorzystywana w klubie miłośników kina powinna zawierać następujące informacje:

- Szczegółowe dane o klubowiczach, włącznie z imieniem i nazwiskiem, adresem pocztowym, datą urodzenia, datą przystąpienia do klubu i adresem poczty elektronicznej.
- Szczegółowe dane o frekwencji na zorganizowanych spotkaniach.
- Szczegółowe informacje o filmach.
- Preferencje filmowe członków klubu miłośników kina.

Powyżej wymieniono aktualne wymagania odnośnie przykładowej bazy danych, w dalszej części książki ta sama baza danych będzie stopniowo rozwijana i poszerzana.

W poprzednim podrozdziale stworzyłeś tabelę ze szczegółowymi informacjami na temat członków klubu — zwróć uwagę na dodatkowe pola uwzględnione w poniższej wersji tej tabeli:

Nazwa pola	Typ danych	Uwagi
`IdentyfikatorCzlonka`	`integer`	Klucz główny.
`Imie`	`nvarchar(50)`	Zmień typ danych na `vargraphic(50)` w systemie IBM DB2 i `varchar(50)` w systemach MySQL i Microsoft Access. W systemie Oracle nie jest dostępny typ `nvarchar` z domyślnym zbiorem znaków, zatem należy ten typ zmienić na `varchar`. Aby mieć możliwość stosowania typu `nvarchar`, podczas tworzenia bazy danych musisz ustawić zestaw znaków Unicode.
`Nazwisko`	`nvarchar(50)`	Zmień typ danych na `vargraphic(50)` w systemie IBM DB2 i `varchar(50)` w systemach MySQL i Microsoft Access. W systemie Oracle nie jest dostępny typ `nvarchar` z domyślnym zbiorem znaków, zatem należy ten typ zmienić na `varchar`. Aby mieć możliwość stosowania typu `nvarchar`, podczas tworzenia bazy danych musisz ustawić zestaw znaków Unicode.
`DataUrodzenia`	`date`	W systemie Microsoft SQL Server zmień typ danych na `datetime`.

Nazwa pola	Typ danych	Uwagi
Ulica	nvarchar(100)	Zmień typ danych na vargraphic(100) w systemie IBM DB2 i varchar(100) w systemach MySQL i Microsoft Access. W systemie Oracle nie jest dostępny typ nvarchar z domyślnym zbiorem znaków, zatem należy ten typ zmienić na varchar. Aby mieć możliwość stosowania typu nvarchar, podczas tworzenia bazy danych musisz ustawić zestaw znaków Unicode.
Miasto	nvarchar(75)	Zmień typ danych na vargraphic(75) w systemie IBM DB2 i varchar(75) w systemach MySQL i Microsoft Access. W systemie Oracle nie jest dostępny typ nvarchar z domyślnym zbiorem znaków, zatem należy ten typ zmienić na varchar. Aby mieć możliwość stosowania typu nvarchar, podczas tworzenia bazy danych musisz ustawić zestaw znaków Unicode.
Wojewodztwo	nvarchar(75)	Zmień typ danych na vargraphic(75) w systemie IBM DB2 i varchar(75) w systemach MySQL i Microsoft Access. W systemie Oracle nie jest dostępny typ nvarchar z domyślnym zbiorem znaków, zatem należy ten typ zmienić na varchar. Aby mieć możliwość stosowania typu nvarchar, podczas tworzenia bazy danych musisz ustawić zestaw znaków Unicode.
KodPocztowy	varchar(6)	
Email	varchar(200)	
DataPrzystapienia	date	W systemie Microsoft SQL Server zmień typ danych na datetime.

Zwróć uwagę na podział pól nazwiska i adresu na mniejsze części. Nazwisko (w oryginalnym projekcie z poprzedniego podrozdziału stosowaliśmy pole Imie) zostało podzielone na pola Imie i Nazwisko, natomiast pole Adres podzielono na pola Ulica, Miasto, Wojewodztwo i KodPocztowy. Podział tych danych poprawia efektywność procesu wyszukiwania konkretnych danych. Przykładowo, gdybyś chciał odszukać wszystkich klubowiczów zamieszkałych w Poznaniu, wystarczy, że w swoim zapytaniu dodałbyś klauzulę sprawdzającą wartość pola Miasto. Gdybyś umieścił ulicę, miasto, województwo i kod pocztowy w jednym polu adresu, wyszukiwanie członków klubu według miasta zamieszkania byłoby znacznie trudniejsze.

Teraz powinieneś przygotować kod języka SQL, który utworzy powyższą tabelę. Jeśli wolisz, do tworzenia tabeli możesz użyć konsoli zarządzania swojego systemu RDBMS.

```
CREATE TABLE SzczegoloweDaneCzlonkow
(
    IdentyfikatorCzlonka integer,
    Imie nvarchar(50),
    Nazwisko nvarchar(50),
    DataUrodzenia date,
    Ulica nvarchar(100),
    Miasto nvarchar(75),
    Wojewodztwo nvarchar(75),
    KodPocztowy varchar(6),
    Email varchar(200),
    DataPrzystapienia date
);
```

W zależności od wykorzystywanego przez Ciebie relacyjnego systemu zarządzania bazami danych być może będziesz zmuszony do zmiany niektórych typów danych (zgodnie z tym, co napisano w poprzedniej tabeli). Jeśli korzystasz z systemu IBM DB2, zamiast typu `varchar` powinieneś stosować typ `vargraphic`. Jeśli natomiast jesteś użytkownikiem systemu Microsoft SQL Server, zamiast typu `date` musisz zastosować typ `datetime`. W systemach MySQL i Microsoft Access nie ma typu `nvarchar` — zamiast niego należy używać typu `varchar`.

Kolejnym krokiem w procesie budowy bazy danych jest stworzenie tabeli `Frekwencja`, która będzie zawierała następujące pola i typy danych:

Nazwa pola	Typ danych	Uwagi
`DataSpotkania`	`date`	W systemie Microsoft SQL Server zmień typ danych na `datetime`.
`Lokalizacja`	`nvarchar(200)`	Zmień typ danych na `vargraphic(200)` w systemie IBM DB2 i `varchar(200)` w systemach MySQL i Microsoft Access. W systemie Oracle nie jest dostępny typ `nvarchar` z domyślnym zbiorem znaków, zatem należy ten typ zmienić na `varchar`. Aby mieć możliwość stosowania typu `nvarchar`, podczas tworzenia bazy danych musisz ustawić zestaw znaków Unicode.
`UczestnictwoCzlonka`	`char(1)`	
`IdentyfikatorCzlonka`	`integer`	Klucz obcy łączący tabelę `Frekwencja` z tabelą `SzczegoloweDaneCzlonkow`.

Tabela `Frekwencja` nie została zmieniona względem oryginalnego projektu omówionego w poprzednim podrozdziale. Wyrażenie języka SQL, które tworzy tę tabelę, powinno mieć następującą postać:

```
CREATE TABLE Frekwencja
(
    DataSpotkania date,
    Lokalizacja nvarchar(200),
    UczestnictwoCzlonka char(1),
    IdentyfikatorCzlonka integer
);
```

Które z pól mogłoby pełnić funkcję unikalnego klucza głównego tej tabeli? Jeśli przyjmiemy, że w danej miejscowości lub dzielnicy może się odbywać tylko jedno spotkanie dziennie, kombinacja tych dwóch pól będzie stanowiła bezpieczny, unikalny identyfikator rekordu. Jeśli nie można wykluczyć dwóch lub większej liczby spotkań tego samego dnia w tym samym miejscu, być może powinieneś zbadać, czy takie spotkania odbywają się o tej samej godzinie; jeśli nie, poza polem daty spotkania należałoby użyć dodatkowego pola godziny spotkania — tabela `Frekwencja` musiałaby zostać poszerzona o kolumnę `GodzinaSpotkania`. Kombinacja pól `DataSpotkania`, `GodzinaSpotkania` i `Lokalizacja` stanowiłaby wówczas unikalny klucz główny. Alternatywnym rozwiązaniem jest oczywiście utworzenie kolumny nazwanej `IdentyfikatorSpotkania`, która zawierałaby unikalną (w skali wszystkich spotkań) liczbę całkowitą i pełniłaby funkcję klucza głównego tabeli `Frekwencja`. Tego typu sytuacje z reguły wymagają konsultacji z przyszłymi użytkownikami bazy danych i dokładnego określenia ich wymagań — często się zdarza, że dopiero na podstawie tak uzyskanych informacji można zaprojektować bazę danych i uniknąć konieczności jej późniejszego przebudowywania.

Pozostaje nam jeszcze dodanie tabel, w których będą przechowywane następujące informacje:

- Szczegółowe informacje o filmach.
- Preferencje filmowe członków klubu miłośników kina.

Najpierw utworzymy nową tabelę nazwaną `Filmy`. Poniżej przedstawiono informacje, które będą składowane w tej tabeli:

- Tytuł filmu.
- Rok premiery.
- Krótkie streszczenie fabuły.
- Dostępność filmu na płytach DVD.
- Ocena filmu według użytkowników (przykładowo, ocena może być wyrażana w skali od 1 do 5, gdzie ocena 1 powinna dotyczyć filmów kompletnie nieudanych, a ocena 5 dzieł ponadczasowych).

I wreszcie musisz przypisać każdy z filmów do kategorii — np. do horrorów, filmów akcji, romansów itp.

Poniżej przedstawiono szkic struktury przyszłej tabeli `Filmy`:

Nazwa pola	Typ danych	Uwagi
`IdentyfikatorFilmu`	`integer`	Klucz główny.
`TytulFilmu`	`nvarchar(100)`	Zmień typ danych na `vargraphic(100)` w systemie IBM DB2 i `varchar(100)` w systemach MySQL i Microsoft Access. W systemie Oracle nie jest dostępny typ `nvarchar` z domyślnym zbiorem znaków, zatem należy ten typ zmienić na `varchar`. Aby mieć możliwość stosowania typu `nvarchar`, podczas tworzenia bazy danych musisz ustawić zestaw znaków Unicode.
`RokPremiery`	`integer`	
`StreszczenieFabuly`	`nvarchar(2000)`	Jeśli używasz systemu Microsoft Access, zmień typ danych na `memo`; jeśli używasz systemu MySQL, zmień ten typ na `text`.
`DostepnyNaDVD`	`char(1)`	
`Ocena`	`integer`	
`IdentyfikatorKategorii`	`integer`	Klucz obcy

Zanim przejdziemy do omawiania kolejnych tabel, musimy przeanalizować dwa istotne aspekty definicji wymienionych powyżej pól tabeli `Filmy`. Po pierwsze, dla pola `StreszczenieFabuly` zastosowano typ danych `nvarchar`, co powoduje określone problemy w systemie Microsoft Access, którego użytkownicy mają do dyspozycji typ `varchar` obsługujący maksymalnie 255 znaków. Ponieważ w tym przypadku niezbędne jest składowanie do 2000 znaków, w systemie Access należy użyć typu `memo`, który dopuszcza możliwość składowania nawet 65 536 znaków. Ponieważ nie można określać długości pól tego typu, automatycznie

jest ona wyznaczana na poziomie 65 536 znaków, zatem deklaracja `StreszczenieFabuly memo` jest równoważna deklaracji `StreszczenieFabuly nvarchar(65536)`.

Drugim ciekawym składnikiem tabeli `Filmy` jest pole `IdentyfikatorKategorii`, które pełni funkcję klucza obcego. Oznacza to, że wartość tego pola musi odpowiadać wartości pola klucza głównego innej tabeli i że pole to reprezentuje relację pomiędzy obiema tabelami. Tabelą zawierającą klucz główny, do którego odwołuje się tabela `Filmy`, jest `KategorieFilmow`. Wspomnianą tabelę utworzymy za chwilę — najpierw przyjrzyj się poniższemu wyrażeniu języka SQL, które tworzy tabelę `Filmy`:

```
CREATE TABLE Filmy
(
    IdentyfikatorFilmu integer,
    TytulFilmu nvarchar(100),
    RokPremiery integer,
    StreszczenieFabuly nvarchar(2000),
    DostepnyNaDVD char(1),
    Ocena integer,
    IdentyfikatorKategorii integer
);
```

Po utworzeniu tabeli `Filmy` możesz utworzyć tabelę kategorii filmów (nazwanej `KategorieFilmow`), która będzie zawierała następujące dane:

Nazwa pola	Typ danych	Uwagi
`IdentyfikatorKategorii`	`integer`	Klucz główny.
`Kategoria`	`nvarchar(100)`	Zmień typ danych na `vargraphic(100)` w systemie IBM DB2 i `varchar(100)` w systemach MySQL i Microsoft Access. W systemie Oracle nie jest dostępny typ `nvarchar` z domyślnym zbiorem znaków, zatem należy ten typ zmienić na `varchar`. Aby mieć możliwość stosowania typu `nvarchar`, podczas tworzenia bazy danych musisz ustawić zestaw znaków Unicode.

Tabela `KategorieFilmow` jest bardzo mała i prosta. Także tworzące tę tabelę wyrażenie języka SQL jest stosunkowo nieskomplikowane:

```
CREATE TABLE KategorieFilmow
(
    IdentyfikatorKategorii integer,
    Kategoria nvarchar(100)
);
```

Ostatnią tworzoną tabelę nazwano `UlubioneKategorie` — rekordy tej tabeli będą reprezentowały ulubione kategorie filmów poszczególnych członków klubu.

Nazwa pola	Typ danych	Uwagi
`IdentyfikatorKategorii`	`integer`	Klucz obcy
`IdentyfikatorCzlonka`	`integer`	Klucz obcy

Jak widać, tabela UlubioneKategorie jest wyjątkowo prosta. Zarówno pole IdentyfikatorKategorii, jak i pole IdentyfikatorCzlonka są kluczami obcymi — pierwsze wskazuje na odpowiedni rekord tabeli KategorieFilmow, drugi na rekord tabeli SzczegoloweDaneCzlonkow. Kombinacja obu tych pól stanowi unikalny klucz główny tabeli UlubioneKategorie. Poniżej przedstawiono wyrażenie języka SQL, które tworzy tę tabelę:

```
CREATE TABLE UlubioneKategorie
(
    IdentyfikatorKategorii integer,
    IdentyfikatorCzlonka integer
);
```

Utworzenie tabeli UlubioneKategorie jest ostatnim krokiem w procesie budowy podstawowej struktury bazy danych. Jak miałeś okazję się przekonać, tworzenie bazy danych jest stosunkowo łatwe! W rozdziale 4. przedstawiono bardziej skomplikowane przykłady i opcje — materiał zawarty w tym podrozdziale stanowi jednak dobrą podstawę do analizy bardziej zaawansowanych rozwiązań.

Podsumowanie

Materiał zawarty w tym rozdziale nie tylko wprowadził Cię w świat języka SQL i projektowania baz danych, ale też zademonstrował technikę pisania kodu języka SQL niezbędnego do tworzenia struktury bazy danych. Wiedza uzyskana podczas lektury tego rozdziału w zupełności wystarcza do samodzielnego eksperymentowania z prostymi projektami baz danych.

Z rozdziału dowiedziałeś się o następujących faktach:

- Bazy danych zapewniają efektywne mechanizmy składowania ogromnych ilości nieprzetworzonych danych. Same bazy danych nie przetwarzają przechowywanych informacji — za przetwarzanie danych odpowiadają wykorzystujące je aplikacje.
- Bazy danych znacznie upraszczają udostępnianie informacji w porównaniu z innymi rozwiązaniami (takimi jak pliki tekstowe, arkusze kalkulacyjne itp.). Bazy danych oferują też mechanizmy zabezpieczające informacje przed dostępem osób niepowołanych oraz umożliwiają definiowanie rozmaitych poziomów uprawnień użytkowników. Możesz ograniczać dostęp do danych pewnej grupie użytkowników i zezwalać na pełny dostęp innym użytkownikom.
- Relacyjne bazy danych zawierają tabele i pola oraz umożliwiają definiowanie relacji pomiędzy danymi składowanymi w różnych tabelach, a także mechanizmy zapewniania spójności bazy danych podczas operacji dodawania nowych i modyfikowania istniejących informacji.
- Bazy danych są częścią większych aplikacji nazywanych systemami zarządzania bazami danych (DBMS).
- SQL jest deklaratywnym językiem programowania; oznacza to, że za pomocą wyrażeń tego języka można opisywać oczekiwane odpowiedzi, a szukanie tych odpowiedzi wśród składowanych informacji należy pozostawić systemowi zarządzania bazą danych.

Po zdobyciu niezbędnej wiedzy podstawowej przystąpiłeś do realizacji kilku zadań praktycznych. W szczególności, utworzyłeś bazę danych i nauczyłeś się prostych konstrukcji języka SQL w zakresie tworzenia tabel. Podczas pracy z językiem SQL poznałeś kilka istotnych aspektów tego języka i samych baz danych:

- Zapoznałeś się z organizacją baz danych. Przekonałeś się, że bazy danych składają się z tabel, które z kolei składają się z rekordów, a każdy z tych rekordów można podzielić na poszczególne pola (kolumny).
- Nauczyłeś się tworzyć bazy danych. Dowiedziałeś się, że można to robić zarówno za pomocą odpowiednich narzędzi relacyjnych systemów zarządzania bazami danych, jak i wyrażeń języka SQL.
- Różne typy danych są składowane w bazie danych na rozmaite sposoby. Dowiedziałeś się, że bazy danych obsługują wiele typów danych reprezentujących dane tekstowe (`char` i `varchar`), dane liczbowe (`integer`, `real` i `decimal`) oraz datę i czas (`time` i `date`). Są to tylko podstawowe typy danych, a wiele relacyjnych systemów zarządzania bazami danych obsługuje znacznie bardziej rozbudowany zestaw typów.
- Zapoznałeś się z regułami projektowania dobrych baz danych.

I wreszcie, na końcu tego rozdziału utworzyłeś przykładową bazę danych dla pozostałych rozdziałów tej książki, stosując techniki i omówione wcześniej wyrażenia języka SQL. Podczas lektury kolejnego rozdziału nauczysz się dodawać, aktualizować i usuwać dane za pomocą odpowiednich poleceń języka SQL. Nie zapomnij tylko o starannym wykonaniu ćwiczeń!

Ćwiczenia

1. Po pewnym czasie klub kinomanów liczy już tylu członków, że spotkania odbywają się regularnie w różnych miastach Polski, co oznacza, że w tabeli `Frekwencja` nasila się problem nadmiarowości danych. Jakie zmiany należałoby wprowadzić w strukturze tabel tej bazy danych.
2. Napisz kod języka SQL potrzebny do wprowadzenia zmian niezbędnych do realizacji ćwiczenia 1. i podziału pola rekordów tej tabeli na bardziej szczegółowe pola `Ulica`, `Miasto` i `Wojewodztwo`.