O'REILLY®

Wydanie III

Wprowadzenie do SQL

Jak generować, pobierać i obsługiwać dane

Helion

Alan Beaulieu

Tytuł oryginału: Learning SQL: Generate, Manipulate, and Retrieve Data, 3rd Edition

Tłumaczenie: Agnieszka Górczyńska

ISBN: 978-83-283-7779-0

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

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/wprsq3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Krótkie wprowadzenie

Zanim zakaszemy rękawy i zabierzemy się do pracy, warto poznać nieco historii związanej z technologią baz danych, aby dzięki temu lepiej zrozumieć, jak ewoluowały bazy danych i język zapytań SQL. Dlatego rozpoczynam od przedstawienia podstawowych koncepcji bazy danych oraz historii skomputeryzowanego przechowywania danych i ich pobierania.

Jeżeli chcesz od razu przystąpić do wykonywania zapytań, możesz przejść do rozdziału 3. Jednak później zachęcam do powrotu do dwóch pierwszych rozdziałów książki, gdyż dzięki temu lepiej zrozumiesz historię i przeznaczenie języka SQL.

Wprowadzenie do baz danych

Baza danych to zbiór powiązanych ze sobą informacji. Na przykład książka telefoniczna to baza danych imion, nazwisk, numerów telefonów i adresów osób mieszkających w danym regionie. Nie ulega wątpliwości, że książka telefoniczna to wszechobecna i najczęściej używana baza danych, ale jednocześnie nie jest ona pozbawiona wad:

- Znalezienie numeru telefonu danej osoby może być czasochłonne, zwłaszcza jeśli książka telefoniczna zawiera ogromną liczbę wpisów.
- Wpisy w książce telefonicznej są indeksowane jedynie według nazwiska i imienia, więc znalezienie osób mieszkających pod określonym adresem jest teoretycznie możliwe, ale nie będzie praktycznym zastosowaniem dla tej konkretnej bazy danych.
- Od chwili wydrukowania książki telefonicznej zamieszczone w niej informacje z każdym dniem stają się coraz mniej aktualne, ponieważ mieszkańcy przeprowadzają się lub zmieniają numer telefonu.

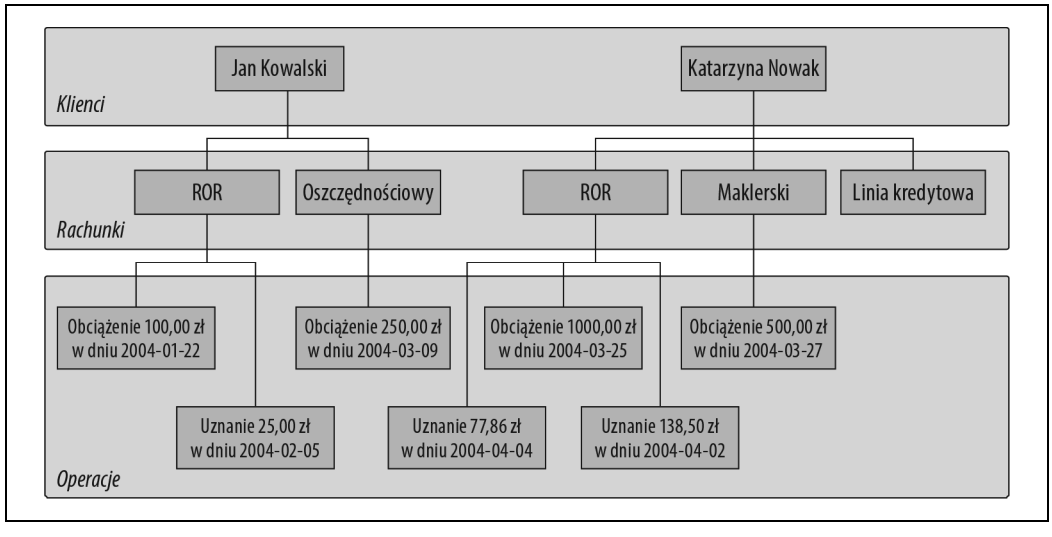
Dokładnie te same wady występują dla każdego ręcznie obsługiwanego magazynu danych, np. przechowywanej w szafce kartoteki pacjentów. Z powodu kłopotliwości papierowych baz danych jednymi z pierwszych aplikacji opracowanych dla komputerów były *systemy baz danych*, czyli skomputeryzowane magazyny danych oferujące mechanizmy przechowywania i pobierania informacji. Skoro system bazy danych przechowuje informacje w postaci elektronicznej, a nie na papierze, to taki system może znacznie szybciej pobierać żądane dane, indeksować je na wiele różnych sposobów, a także dostarczać użytkownikom najaktualniejsze informacje.

Wczesne systemy baz danych zarządzały danymi, które były przechowywane na taśmach magnetycznych. Jednak na początku było więcej taśm niż odczytujących je urządzeń, więc technicy musieli na żądanie wkładać odpowiednie taśmy do urządzeń, gdy zachodziła potrzeba uzyskania dostępu do określonych danych. Ponieważ w tamtym czasie komputery były wyposażone w niewielkie ilości pamięci, ogólnie rzecz biorąc, wiele żądań dotyczących tych samych danych wymagało wielokrotnego odczytywania informacji z taśm. Wprawdzie wczesne systemy baz danych były znacznym usprawnieniem względem papierowych baz danych, ale jednocześnie było im daleko do możliwości, jakie oferuje obecnie dostępna technologia. (Nowoczesne systemy baz danych potrafią obsługiwać petabajty danych dostarczane przez klastry serwerów, z których każdy może buforować dziesiątki gigabajtów danych w charakteryzującej się dużą szybkością działania pamięci. Chyba zanadto wybiegłem w przyszłość).

Nierelacyjne systemy baz danych

W tej sekcji znajdują się informacje na temat nierelacyjnych systemów baz danych. Jeżeli interesuje Cię język SQL, pomiń te kilka stron i przejdź do następnej sekcji.

W pierwszych dekadach skomputeryzowanych systemów baz danych informacje były przechowywane i przedstawiane na wiele różnych sposobów. Na przykład w *hierarchicznym systemie baz danych* informacje były przedstawiane w postaci jednej lub więcej struktur drzewa. Na rysunku 1.1 pokazałem, jak dane finansowe dotyczące dwóch osób mogą być przedstawione za pomocą struktur drzewa.

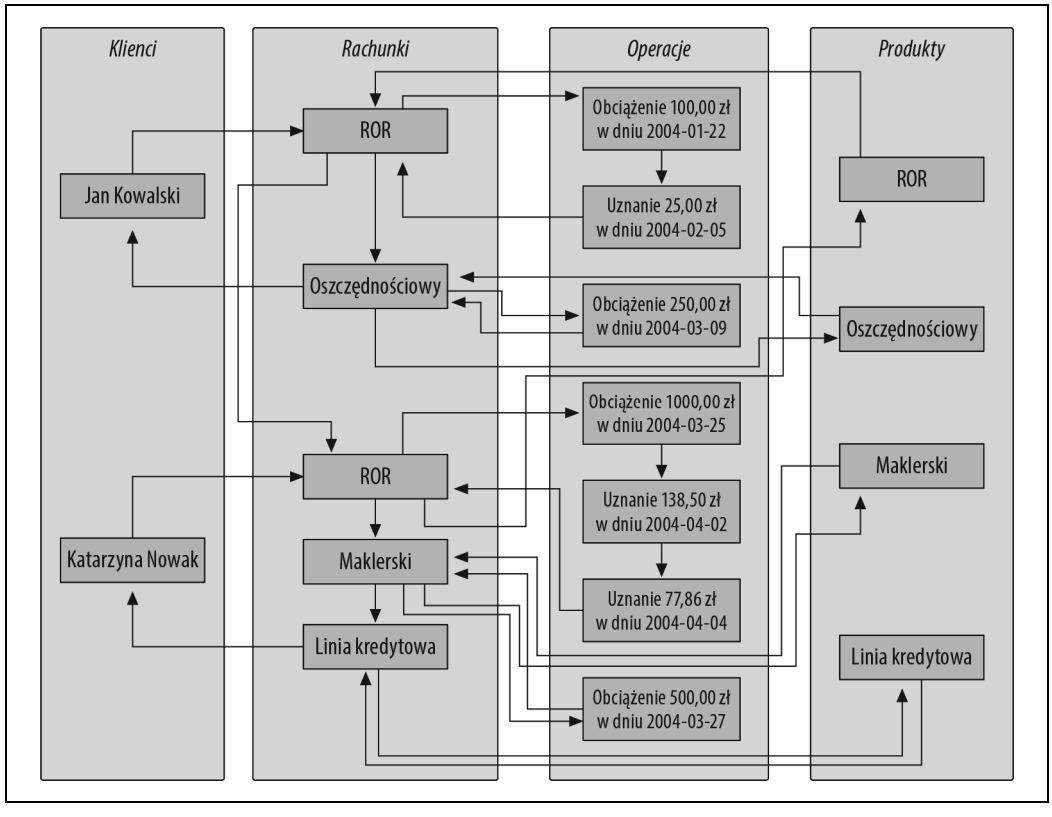


Rysunek 1.1. Hierarchiczny widok danych finansowych

Jan i Katarzyna mają oddzielne struktury drzewa przedstawiające ich rachunki i przeprowadzane na nich operacje. Taki hierarchiczny system bazy danych dostarcza narzędzia przeznaczone do wyszukiwania struktury drzewa określonego klienta, a następnie poruszania się po tym drzewie w poszukiwaniu żądanych rachunków i/lub operacji. Każdy węzeł drzewa może mieć zero lub jeden

element nadrzędny albo zero, jeden lub więcej elementów potomnych. Taka konfiguracja jest określana mianem *hierarchii z pojedynczym elementem nadrzędnym*.

Inne często spotykane podejście to *sieciowy system bazy danych*. Udostępnia on zbiory rekordów i łączy definiujących relacje zachodzące między poszczególnymi rekordami. Na rysunku 1.2 pokazałem, jak informacje o tych samych rachunkach Jana i Katarzyny można zapisać w takim systemie.



Rysunek 1.2. Sieciowy widok danych finansowych

Aby wyszukać transakcje przeprowadzone na rachunku maklerskim Katarzyny, trzeba wykonać następujące kroki:

1. Wyszukanie rekordu klienta — Katarzyny Nowak.
2. Podążanie za łączem od rekordu Katarzyny Nowak do listy jej rachunków.
3. Poruszanie się po łańcuchu rachunków aż do znalezienia rachunku maklerskiego.
4. Podążanie za łączem od rekordu rachunku maklerskiego do listy przeprowadzonych na nim operacji.

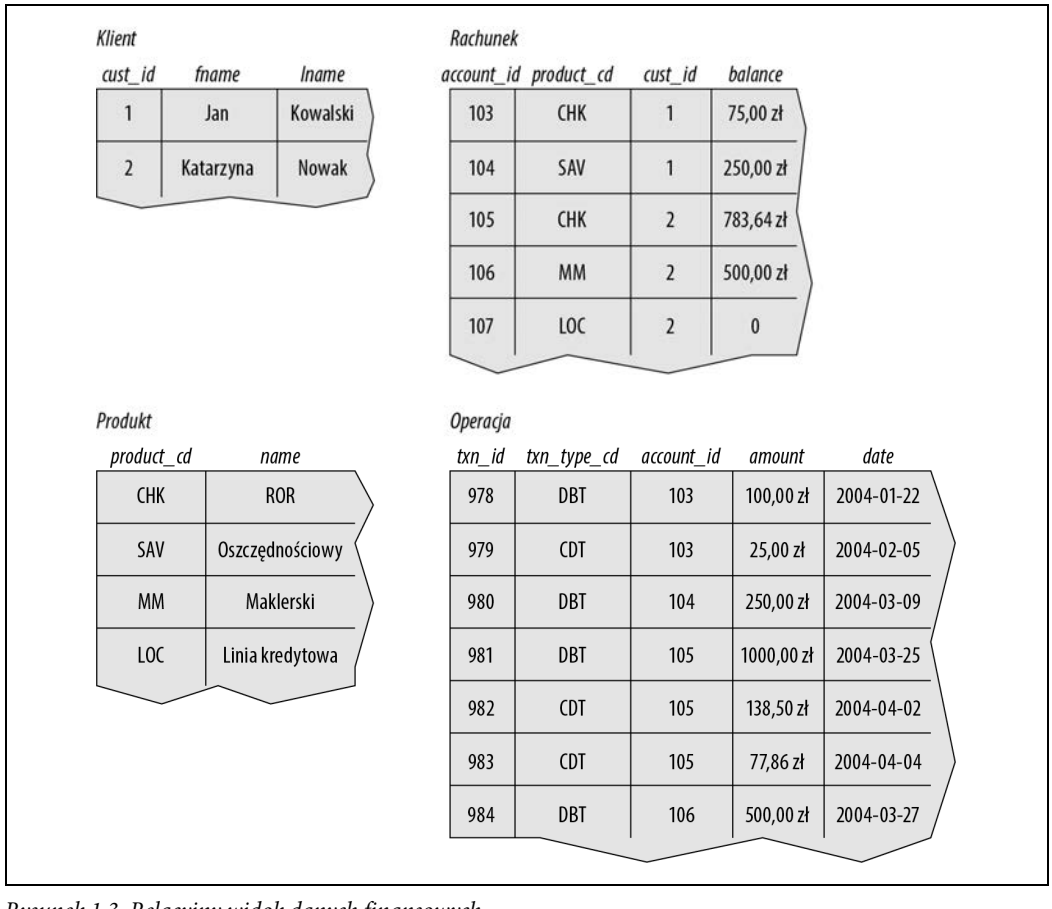
Jedna z interesujących funkcjonalności sieciowego systemu bazy danych została pokazana za pomocą zbioru rekordów `product` po prawej stronie na rysunku 1.2. Zwróć uwagę na to, że każdy rekord `product` (ROR, rachunek oszczędnościowy itd.) prowadzi do rekordów `account` będących

typu danego produktu. Dlatego rekordy `account` są dostępne z poziomu wielu miejsc (np. rekordów `customer` i `product`), co pozwala sieciowej bazie danych działać w charakterze *hierarchii z wieloma elementami nadrzędnymi*.

Hierarchiczne i sieciowe systemy baz danych wciąż są w użyciu. Ponadto hierarchiczne systemy baz danych przeżywają swoją drugą młodość dzięki obszarowi usług katalogowych, np. Microsoft Active Directory i Apache Directory Server (to ostatnie rozwiązanie jest typu open source). Jednak od lat siedemdziesiątych ubiegłego wieku popularnością cieszy się nowy sposób przedstawiania danych — znacznie bardziej rygorystyczny, choć jednocześnie łatwy do zrozumienia i implementacji.

Model relacyjny

W 1970 roku doktor E.F. Codd z laboratorium badawczego IBM opublikował pracę zatytułowaną *A Relational Model of Data for Large Shared Data Banks*, w której zaproponował prezentowanie danych w postaci zestawu *tabel*. Zamiast wskaźników używanych do poruszania się między powiązanymi encjami, nadmiarowe dane są używane do łączenia rekordów w różnych tabelach. Na rysunku 1.3 pokazałem, jak informacje o rachunkach Jana i Katarzyny można przedstawić w kontekście relacyjnym.



Klient

cust_id	fname	lname
1	Jan	Kowalski
2	Katarzyna	Nowak

Rachunek

account_id	product_cd	cust_id	balance
103	CHK	1	75,00 zł
104	SAV	1	250,00 zł
105	CHK	2	783,64 zł
106	MM	2	500,00 zł
107	LOC	2	0

Produkt

product_cd	name
CHK	ROR
SAV	Oszczędnościowy
MM	Maklerski
LOC	Linia kredytowa

Operacja

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	100,00 zł	2004-01-22
979	CDT	103	25,00 zł	2004-02-05
980	DBT	104	250,00 zł	2004-03-09
981	DBT	105	1000,00 zł	2004-03-25
982	CDT	105	138,50 zł	2004-04-02
983	CDT	105	77,86 zł	2004-04-04
984	DBT	106	500,00 zł	2004-03-27

Rysunek 1.3. Relacyjny widok danych finansowych

Cztery tabele zamieszczone na rysunku 1.3 przedstawiają cztery omówione dotychczas encje: `customer` (klient), `product` (produkt), `account` (rachunek) i `transaction` (operacja). Na górze tabeli `customer` możesz zobaczyć trzy *kolumny*: `cust_id` (zawiera identyfikator klienta), `fname` (zawiera imię klienta) i `lname` (zawiera nazwisko klienta). Analizując dalej tę tabelę, zauważysz dwa *rekordy*, po jednym dla danych dotyczących Jana Kowalskiego i Katarzyny Nowak. Liczba kolumn, które mogą znajdować się w tabeli, będzie się różniła w zależności od serwera, ale ogólnie rzecz biorąc, jest to liczba na tyle duża, że nie powinna stanowić problemu — np. Microsoft SQL Server umożliwia umieszczenie do 1024 kolumn w tabeli. Z kolei liczba rekordów, jakie można wstawić w tabeli, jest bardziej kwestią ograniczeń fizycznych (np. wielkość dostępnej przestrzeni dyskowej) i ograniczeń związanych z późniejszą obsługą (np. jaką wielkość może osiągnąć tabela, zanim praca z nią okaże się trudna) niż ograniczeń samego oprogramowania serwera bazy danych.

Każda tabela w relacyjnej bazie danych zawiera informacje unikatowo identyfikujące rekord w danej tabeli (tzw. *klucz podstawowy*) oraz informacje dodatkowe niezbędne do pełnego opisania encji. Ponownie analizując tabelę `customer`, dostrzeżesz, że kolumna `cust_id` zawiera inną wartość dla każdego klienta, np. Jana Kowalskiego można unikatowo zidentyfikować na podstawie identyfikatora o wartości 1. Żadnemu innemu klientowi nigdy nie zostanie przypisany identyfikator o takiej wartości, a żadne inne informacje nie są niezbędne do odszukania w tabeli `customer` danych dotyczących Jana Kowalskiego.

Każdy serwer bazy danych dostarcza mechanizm pozwalający na unikatowe generowanie zbiorów liczb używanych jako wartości klucza podstawowego. Dlatego nie musisz się zajmować sprawdzaniem, jakie liczby zostały przypisane jako klucze podstawowe.

Wprawdzie można zdecydować się na użycie połączenia kolumn `fname` i `lname` jako klucza podstawowego (klucz podstawowy składający się z co najmniej dwóch kolumn jest nazywany *kluczem złożonym*), ale zdarza się, że klientami banku są osoby mające to samo imię i nazwisko. Dlatego zdecydowałem się dodać do tabeli `customer` kolumnę przeznaczoną do użycia w charakterze klucza podstawowego.

W omawianym przykładzie klucz podstawowy składający się z kolumn `fname` i `lname` będzie nazywany *kluczem naturalnym*, podczas gdy klucz podstawowy na podstawie kolumny `cust_id` jest nazywany *kluczem zastępczym*. Decyzja co do tego, czy należy stosować klucze naturalne czy zastępcze, należy do projektanta bazy danych. W tym konkretnym przypadku wybór jest prosty — nazwisko może ulec zmianie (np. po przyjęciu nazwiska małżonka), a kolumny klucza podstawowego nigdy nie powinny ulegać zmianie po przypisaniu wartości.

Część tabel zawiera także informacje wykorzystywane do poruszania się między tabelami — w tym miejscu znajdują się wspomniane wcześniej „dane nadmiarowe”. Na przykład tabela `account` zawiera kolumnę o nazwie `cust_id` z unikatowym identyfikatorem użytkownika, który otworzył dany rachunek. Tabela `account` ma również kolumnę `product_cd`, zawierającą unikatowy identyfikator produktu, z którym jest zgodny dany rachunek bankowy. Te kolumny są określane mianem *kluczy zewnętrznych* i pełnią taką samą rolę jak linie łączące encje w hierarchicznych i sieciowych diagramach informacji o rachunkach bankowych. Jeżeli analizujesz określony rekord rachunku i chcesz otrzymać więcej informacji o kliencie, który otworzył dany rachunek, pobierz wartość kolumny

cust_id i wykorzystaj ją do odszukania odpowiedniego rekordu w tabeli customer (w żargonie związanym z relacyjnymi bazami danych jest to tzw. *złączenie* — więcej informacji ogólnych na temat złączeń zamieściłem w rozdziale 3., a dokładniejsze informacje znajdziesz w rozdziałach 5. i 10.).

Wielokrotne przechowywanie tych samych danych może wydawać się marnotrawstwem, choć w modelu relacyjnym wyraźnie wiadomo, które dane nadmiarowe mogą być przechowywane. Dlatego w przypadku tabeli account właściwe jest dodanie kolumny zawierającej unikatowy identyfikator klienta, który otworzył dany rachunek bankowy, zbędne zaś będzie wstawianie do tej tabeli kolumn zawierających imię i nazwisko klienta. Jeżeli klient zmieni np. nazwisko, to najlepiej, aby odpowiednią zmianę trzeba było wprowadzić tylko w jednym miejscu w bazie danych. W przeciwnym razie, jeśli dane zostaną zmienione w jednym miejscu, a stare pozostawione w innym, wówczas informacje przechowywane w takiej bazie danych będą niewiarygodne. Właściwym miejscem dla danych dotyczących klienta jest tabela customer (ang. *klient*), a w innych tabelach powinna być wstawiana tylko wartość cust_id. Nieodpowiednie jest również wstawianie wielu fragmentów informacji do pojedynczej kolumny, np. gdy kolumna name zawiera imię i nazwisko, a kolumna address zawiera nazwę ulicy, numer budynku, kod pocztowy, nazwę miejscowości i nazwę województwa. Podczas projektowania bazy danych należy się upewnić, że poszczególne, niezależne fragmenty informacji znajdują się w tylko jednym miejscu (z wyjątkiem kluczy zewnętrznych) — jest to określane mianem *normalizacji*.

Jeżeli raz jeszcze spojrzysz na rysunek 1.3, to być może będziesz się zastanawiać, jak wykorzystać te tabele do wyszukania operacji Jana Kowalskiego na rachunku ROR. Trzeba zacząć od wyszukania w tabeli customer unikatowego identyfikatora Jana Kowalskiego. Następnie w tabeli account trzeba znaleźć rekord, którego kolumna cust_id zawiera unikatowy identyfikator Jana Kowalskiego, a kolumna product_cd ma wartość odpowiadającą rekordowi w tabeli product zawierającemu kolumnę name o wartości ROR. Dalej w tabeli transaction trzeba odszukać wszystkie rekordy, których kolumna account_id ma wartość równą unikatowemu identyfikatorowi w tabeli account. Wprawdzie to może brzmieć skomplikowanie, ale tę operację da się przeprowadzić za pomocą pojedynczego zapytania w SQL, o czym się wkrótce przekonasz.

Wybrana terminologia

W poprzedniej sekcji pojawiła się nowa terminologia, więc warto poświęcić nieco czasu na zapoznanie się z oficjalnymi definicjami. W tabeli 1.1 wymieniłem pojęcia, które będą używane w pozostałej części książki, oraz ich definicje.

Tabela 1.1. Wybrane pojęcia i ich definicje

Pojęcie	Definicja
Encja	To jest coś interesującego dla społeczności użytkowników bazy danych. Mogą to być klienci, części zamienne, lokalizacje geograficzne itd.
Kolumna	Pojedynczy fragment danych przechowywany w tabeli
Rekord	Zbiór kolumn, które razem opisują encję bądź akcję przeprowadzaną dla tej encji
Tabela	Zbiór rekordów przechowywany w pamięci (nietrwałym) lub w trwałym magazynie danych
Zbiór wynikowy	Inna nazwa dla tabeli tymczasowej, ogólnie rzecz biorąc, jest to wynik zapytania SQL
Klucz podstawowy	Jedna lub więcej kolumn, które mogą być używane jako unikatowy identyfikator dla poszczególnych rekordów w tabeli
Klucz zewnętrzny	Jedna lub więcej kolumn, które mogą być używane razem w celu zidentyfikowania pojedynczego rekordu w innej tabeli

Co to jest SQL?

Codd nie tylko opracował definicję modelu relacyjnego, ale jeszcze zaproponował język o nazwie DSL/Alpha, przeznaczony do przeprowadzania operacji na danych znajdujących się w tabelach relacyjnych. Wkrótce po opublikowaniu informacji przez Codda w firmie IBM powstała grupa, której zadaniem było przygotowanie prototypu opartego na ideach Codda. Wynikiem pracy owej grupy była uproszczona wersja DSL/Alpha o nazwie SQUARE. Usprawnienia wprowadzone w SQUARE doprowadziły do powstania języka SEQUEL, którego nazwa później została skrócona do SQL. Gdy SQL stał się językiem stosowanym do operowania na danych w relacyjnych bazach danych, ewoluował (jak się przekonasz podczas lektury książki) do postaci języka przeznaczonego do przeprowadzania operacji na danych używanych w różnych technologiach bazodanowych.

SQL ma obecnie ponad 40 lat i zdążył w tym czasie ulec znacznym zmianom. W połowie lat osiemdziesiątych ubiegłego stulecia instytut ANSI rozpoczął pracę nad pierwszym standardem języka SQL, który został opublikowany w 1986 roku. Dalsze prace prowadziły do kolejnych wydań standardu SQL w latach 1989, 1992, 1999, 2003, 2006, 2008, 2011 i 2016. Razem ze zmianami w podstawowej funkcjonalności języka wprowadzono nowe funkcje dostarczające m.in. możliwości zorientowane obiektowo. W późniejszych wydaniach standardu skoncentrowano się na integracji technologii powiązanych z SQL-em, np. języka XML (ang. *Extensible Markup Language*) i formatu JSON (ang. *JavaScript Object Notation*).

Rozwój języka SQL idzie w parze z usprawnieniami w modelu relacyjnym, ponieważ wynikiem wykonania zapytania SQL jest tabela (w tym kontekście nazywana *zbiorem wynikowym*). Dlatego nowa trwała tabela może zostać utworzona w relacyjnej bazie danych przez wstawienie zbioru wynikowego zapytania. Z kolei zapytanie może korzystać z tabel zarówno trwale przechowywanych w magazynie danych, jak i tymczasowych, a zbiór wynikowy jednego zapytania może dostarczać dane wejściowe dla innego (wyjaśnię to dokładnie w rozdziale 9.).

Pozostała jeszcze jedna ważna kwestia dotycząca języka SQL: jego nazwa jest akronimem o wielu znaczeniach, choć wiele osób upiera się przy określeniu *strukturalny język zapytań* (ang. *Structured Query Language*). Do języka odwołujemy się za pomocą skrótu SQL.

Klasy zapytań SQL

Język SQL został podzielony na wiele oddzielnych części. W książce skoncentrowałem się na *zapytaniach schematu SQL* używanych do definiowania struktur danych przechowywanych w bazie danych, na *zapytaniach danych SQL* stosowanych do przeprowadzania operacji na wcześniej zdefiniowanych strukturach danych oraz na *zapytaniach transakcyjnych SQL* wykorzystywanych do oznaczania początku i końca transakcji oraz do wycofywania transakcji (te koncepcje omówiłem w rozdziale 12.). Na przykład aby utworzyć nową tabelę w bazie danych, można skorzystać z zapytania schematu `CREATE TABLE`, natomiast podczas wstawiania danych do nowej tabeli wymagane jest użycie zapytania danych SQL, czyli `INSERT`.

Aby mieć przedsmak owych zapytań, spójrz na zapytanie schematu SQL, którego wynikiem działania jest utworzenie tabeli o nazwie `corporation`:

```
CREATE TABLE corporation
 (corp_id SMALLINT,
  name VARCHAR(30),
  CONSTRAINT pk_corporation PRIMARY KEY (corp_id)
 );
```

To zapytanie tworzy tabelę z dwiema kolumnami, `corp_id` i `name`, przy czym pierwsza z nich jest kluczem podstawowym tabeli. W rozdziale 2. poznasz nieco więcej szczegółów związanych z tym zapytaniem, np. poszczególne typy danych oferowane przez serwer MySQL. Kolejne zapytanie powoduje wstawienie do tabeli `corporation` rekordu z informacjami o Acme Paper Corporation:

```
INSERT INTO corporation (corp_id, name)
VALUES (27, 'Acme Paper Corporation');
```

Działanie tego zapytania polega na wstawieniu do tabeli `corporation` nowego rekordu z wartością `27` dla kolumny `corp_id` i wartością `Acme Paper Corporation` dla kolumny `name`.

Spójrz teraz na proste zapytanie `SELECT`, które pobiera dane przed chwilą wstawione do tabeli:

```
mysql> SELECT name
    -> FROM corporation
    -> WHERE corp_id = 27;
+------------------------+
| name                   |
+------------------------+
| Acme Paper Corporation |
+------------------------+
```

Wszystkie elementy bazy danych utworzone za pomocą zapytań schematu SQL są przechowywane w specjalnym zbiorze tabel zwanym *słownikiem danych*. Są to „dane dotyczące bazy danych" i noszą nazwę *metadanych* (więcej informacji na ich temat znajdziesz w rozdziale 15.). Podobnie jak w przypadku samodzielnie tworzonych tabel, do pobierania informacji z tabel słownika danych używa się zapytań `SELECT`. Dzięki temu można poznać aktualne struktury danych zastosowane w bazie danych. Jeżeli zostaniesz poproszony o przygotowanie raportu dotyczącego nowych rachunków utworzonych w ubiegłym miesiącu, możesz na stałe umieścić nazwy kolumn tabeli `account` znane w chwili tworzenia raportu lub wykonać zapytanie do słownika danych w celu ustalenia bieżącego zbioru kolumn i dynamicznego wygenerowania raportu w trakcie każdego wykonywania zapytania.

W większości książek nacisk kładzie się na obszar danych języka SQL, czyli związany z zapytaniami `SELECT`, `UPDATE`, `INSERT` i `DELETE`. Dokładniej omówione w rozdziale 2. zapytania schematu SQL pozwalają projektować i tworzyć proste tabele. Ogólnie rzecz biorąc, zapytania schematu SQL nie wymagają zbyt obszernego wyjaśnienia (poza omówieniem ich składni), podczas gdy zapytania danych SQL, choć jest ich mniej, często wymagają dokładniejszego omówienia. Wprawdzie spróbuję skoncentrować się na wprowadzeniu wielu zapytań schematu SQL, ale w większości rozdziałów książki będziesz mieć do czynienia z zapytaniami danych SQL.

SQL — język nieproceduralny

Jeżeli masz doświadczenie w pracy z językami programowania, przywykłeś do definiowania zmiennych i struktur danych, używania konstrukcji warunkowych (np. `if-then-else`), stosowania pętli (np. `do-while`) oraz dzielenia kodu na mniejsze fragmenty przeznaczone do wielokrotnego użycia (np. obiekty, funkcje, procedury). Kod, który tworzysz, jest przekazywany do kompilatora, a ten

generuje plik wykonywalny, który po uruchomieniu wykonuje dokładnie (no cóż, nie zawsze *dokładnie*) to, co zaprogramowałeś. Niezależnie od tego, czy używasz Javy, Pythona, Scali czy innego języka *proceduralnego*, masz pełną kontrolę nad sposobem działania programu.

Proceduralny język programowania definiuje zarówno oczekiwany wynik, jak i mechanizm (proces) prowadzący do wygenerowania tego wyniku. W językach nieproceduralnych również następuje zdefiniowanie żądanego wyniku, ale proces jego wygenerowania jest pozostawiony zewnętrznemu agentowi.

Jednak w przypadku języka SQL trzeba pogodzić się z brakiem pełnej kontroli, ponieważ zapytania SQL definiują niezbędne dane wejściowe i wyjściowe, natomiast sposób wykonania zapytań pozostaje w gestii serwera bazy danych, a dokładniej działającego w nim *optymalizatora*. Zadanie optymalizatora polega na przeanalizowaniu zapytań SQL, ustaleniu sposobu konfiguracji tabel i dostępnych indeksów oraz wybraniu jak najbardziej efektywnej ścieżki wykonania (nie zawsze będzie to najefektywniejszy sposób wykonania zapytania). Większość silników bazy danych pozwala zachować wpływ na decyzje podejmowane przez optymalizator, co jest możliwe za pomocą tzw. *podpowiedzi dla optymalizatora*. Te podpowiedzi sugerują konkretny indeks do użycia. Jednak większość użytkowników SQL nigdy nie osiąga takiego stopnia zaawansowania i zadanie dostrojenia bazy danych pozostawia administratorowi lub ekspertowi w zakresie wydajności działania bazy danych.

Dlatego za pomocą języka SQL nie można tworzyć pełnych aplikacji. O ile nie tworzysz prostego skryptu przeznaczonego do operowania określonymi danymi, SQL trzeba będzie zintegrować z ulubionym językiem programowania. Część producentów baz danych zrobiło to już za użytkownika — przykładami mogą być język PL/SQL stosowane w Oracle, język procedur składowanych w MySQL oraz opracowany przez Microsoft język Transact-SQL. Dzięki tym językom zapytania danych SQL są częścią gramatyki języka, co pozwala na bezproblemowe zintegrowanie zapytań do bazy danych z poleceniami proceduralnymi. Jeżeli korzystasz z języka programowania pozbawionego związku z bazami danych, np. Javy lub Pythona, musisz skorzystać z pakietów narzędziowych lub API, aby można było wykonywać zapytania SQL z poziomu kodu. Część tych pakietów dostarczają producenci serwerów baz danych, natomiast inne są tworzone przez firmy zewnętrzne lub jako projekty open source. W tabeli 1.2 wymieniłem kilka dostępnych opcji w zakresie integracji SQL z wybranymi językami programowania.

Tabela 1.2. Pakiety narzędziowe umożliwiające integrację SQL z językami programowania

Język programowania	Narzędzie
Java	JDBC (Java Database Connectivity)
C#	ADO.NET (Microsoft)
Ruby	Ruby DBI
Pyton	Python DB
Go	Pakiet database/sql

Jeżeli masz jedynie potrzebę interaktywnego wykonywania zapytań SQL, każdy producent bazy danych oferuje przynajmniej proste narzędzie powłoki pozwalające wykonywać zapytania SQL do silnika bazy danych i wyświetlać ich wyniki. Większość producentów dostarcza także narzędzia

graficzne zawierające okno przeznaczone dla zapytań SQL i okno dla wyniku wykonania tych zapytań. Ponadto istnieją narzędzia firm zewnętrznych, np. SQuirrel, pozwalające nawiązać połączenie JDBC z wieloma różnymi serwerami baz danych. Przykłady zamieszczone w książce dotyczą bazy danych MySQL, więc do wykonywania zapytań i formatowania ich wyników będę korzystał z narzędzia powłoki `mysql`, które jest instalowane razem z serwerem MySQL.

Przykłady zapytań SQL

Obiecałem pokazać zapytanie SQL, które zwróci wszystkie operacje przeprowadzone na rachunku ROR Jana Kowalskiego, zatem oto ono:

```
SELECT t.txn_id, t.txn_type_cd, t.txn_date, t.amount
FROM individual i
  INNER JOIN account a ON i.cust_id = a.cust_id
  INNER JOIN product p ON p.product_cd = a.product_cd
  INNER JOIN transaction t ON t.account_id = a.account_id
WHERE i.fname = 'Jan' AND i.lname = 'Kowalski'
  AND p.name = 'ROR';
+--------+-------------+---------------------+--------+
| txn_id | txn_type_cd | txn_date            | amount |
+--------+-------------+---------------------+--------+
|     11 | DBT         | 2008-01-05 00:00:00 | 100.00 |
+--------+-------------+---------------------+--------+
1 row in set (0.00 sec)
```

Nie będę w tym miejscu dokładnie omawiał tego zapytania. Wystarczy wiedzieć, że wyszukuje ono w tabeli `individual` rekord dotyczący Jana Kowalskiego, w tabeli `product` wyszukuje rekord przedstawiający rachunek ROR, w tabeli `account` wyszukuje wszystkie rekordy dla połączenia danej osoby i rachunku, a następnie zwraca cztery kolumny tabeli `transaction` z wszystkimi operacjami przeprowadzonymi na tym rachunku. Jeżeli znasz identyfikator Jana Kowalskiego, `8`, i kod rachunku, `'CHK'`, to można uprościć nieco to zapytanie, wykorzystując te informacje. W takim przypadku zapytanie będzie miało następującą postać:

```
SELECT t.txn_id, t.txn_type_cd, t.txn_date, t.amount
FROM account a
  INNER JOIN transaction t ON t.account_id = a.account_id
WHERE a.cust_id = 8 AND a.product_cd = 'CHK';
```

Wszystkie koncepcje zawarte w tych zapytaniach (i wiele więcej) omówię w kolejnych rozdziałach książki. W tym miejscu chciałem jedynie zaprezentować przykładowe zapytania.

Przedstawione wcześniej zapytania zawierały trzy różne *klauzule*: `SELECT`, `FROM` i `WHERE`. Praktycznie każde napotkane zapytanie będzie zawierało przynajmniej te trzy klauzule, choć istnieje znacznie więcej klauzul przeznaczonych do ściśle określonych celów. Role poszczególnych klauzul zamieściłem w kolejnym fragmencie kodu:

```
SELECT /* co najmniej jeden element danych */ ...
FROM /* co najmniej jedna lokalizacja */ ...
WHERE /*co najmniej jeden warunek */ ...
```

Większość implementacji SQL traktuje tekst ujęty między znacznikami /* i */ jako komentarz.

Podczas przygotowywania zapytania pierwszym zadaniem jest ustalenie tabel, z których będą pobierane dane, a następnie dodanie tych tabel do klauzuli FROM. Następnie trzeba zdefiniować warunki w klauzuli WHERE i odfiltrować z tych tabel niepotrzebne dane. Kolejnym krokiem jest ustalenie, które kolumny z różnych tabel muszą zostać pobrane, i dodanie ich do klauzuli SELECT. Spójrz na prosty przykład, który pokazuje, jak wyszukać wszystkich klientów o nazwisku Kowalski:

```
SELECT cust_id, fname
FROM individual
WHERE lname = 'Kowalski';
```

To zapytanie pobiera z tabeli individual wszystkie rekordy, których kolumna lname ma wartość w postaci ciągu tekstowego 'Kowalski', a następnie zwraca kolumny cust_id i fname dla tych rekordów.

Poza pobieraniem informacji z bazy danych prawdopodobnie będziesz również wstawiać i modyfikować informacje przechowywane w bazie danych. Spójrz na przykład, który pokazuje, jak wstawić nowy rekord do tabeli product:

```
INSERT INTO product (product_cd, name)
VALUES ('CD', 'Certyfikat depezytowy')
```

Ups, wygląda na to, że popełniono błąd w słowie „depozytowy". To żaden problem. Ten błąd można naprawić za pomocą zapytania UPDATE:

```
UPDATE product
SET name = 'Certyfikat depozytowy'
WHERE product_cd = 'CD';
```

Zwróć uwagę na to, że zapytanie UPDATE również zawiera klauzulę WHERE, podobnie jak zapytanie SELECT. Wynika to z konieczności ustalenia, które rekordy mają zostać zmodyfikowane przez zapytanie UPDATE. W omawianym przykładzie są modyfikowane jedynie te rekordy, których kolumna product_cd ma wartość w postaci ciągu tekstowego 'CD'. Ponieważ kolumna product_cd jest kluczem podstawowym w tabeli product, należy oczekiwać zmodyfikowania przez zapytanie UPDATE dokładnie jednego rekordu (lub żadnego, gdy podana wartość nie istnieje w tabeli). Po każdym wykonaniu zapytania danych SQL otrzymujesz wygenerowaną przez serwer bazy danych informację o liczbie rekordów, których dotyczyło zapytanie. Jeżeli korzystasz z narzędzia interaktywnego, takiego jak wspomniane wcześniej mysql, wówczas otrzymasz komunikat informujący o liczbie rekordów:

- zwróconych przez zapytanie SELECT;
- utworzonych przez zapytanie INSERT;
- zmodyfikowanych przez zapytanie UPDATE;
- usuniętych przez zapytanie DELETE.

Jeżeli używasz języka proceduralnego z jednym z wspomnianych wcześniej pakietów narzędziowych, pakiet będzie zawierał wywołanie przeznaczone do pobrania takich informacji po wykonaniu zapytania danych SQL. Ogólnie rzecz biorąc, dobrze jest sprawdzać te informacje i weryfikować, czy zapytanie nie wykonało czegoś nieoczekiwanego (np. gdy zapomnisz umieścić klauzulę WHERE w zapytaniu DELETE i w ten sposób usuniesz całą zawartość tabeli!).

Co to jest MySQL?

Rozwiązania komercyjne oparte na relacyjnych bazach danych są dostępne od ponad trzech dekad. Do najbardziej dopracowanych i najpopularniejszych produktów komercyjnych w tej dziedzinie zaliczamy:

- Oracle Database firmy Oracle Corporation,
- SQL Server firmy Microsoft,
- DB2 Universal Database firmy IBM.

Wszystkie te serwery baz danych działają w bardzo podobny sposób, choć część z nich jest lepiej przystosowana do obsługi naprawdę ogromnych baz danych, inne zaś lepiej się sprawdzają w obsłudze obiektów, ogromnych plików, ogromnych dokumentów XML itd. Ponadto wszystkie wymienione serwery zapewniają dobrą zgodność z najnowszym standardem ANSI SQL. To jest cenna zaleta i zamierzam pokazać, jak należy tworzyć zapytania SQL, które jedynie po niewielkiej modyfikacji lub nawet bez jakichkolwiek zmian będą mogły być wykonywane na wszystkich platformach.

Poza komercyjnymi serwerami baz danych byliśmy w ciągu ostatnich dwudziestu lat świadkami dość dużej aktywności w społeczności oprogramowania open source. Skutkiem wysiłku tej społeczności było opracowanie dwóch dość często obecnie używanych serwerów baz danych: PostgreSQL i MySQL. Oprogramowanie MySQL jest dostępne bezpłatnie, a jego pobranie i instalacja to niezwykle prosty proces. Z tego powodu wszystkie przykłady zamieszczone w książce dotyczą serwera MySQL w wersji 8.0, a do wykonania zapytań i sformatowania ich danych wyjściowych wykorzystałem działające w powłoce narzędzie `mysql`. Nawet jeśli już używasz innego serwera baz danych i nie planowałeś korzystać z MySQL, zachęcam do pobrania i zainstalowania najnowszej wersji, dodanie przykładowej bazy danych oraz poeksperymentowanie z danymi i przykładami zamieszczonymi w niniejszej książce.

Musisz jednak pamiętać o jednej ważnej kwestii:

> To nie jest książka o implementacji SQL-a w serwerze MySQL.

Celem materiału zamieszczonego w książce jest natomiast pokazanie, jak należy tworzyć zapytania SQL przeznaczone do ich wykonywania (bez konieczności modyfikowania) w MySQL, a także (po wprowadzeniu niewielkich lub nawet bez modyfikacji) w najnowszych wersjach Oracle Database, DB2 i SQL Server.

Nie tylko relacyjne bazy danych

W trakcie dekady, która upłynęła między drugim i trzecim wydaniem książki, w świecie baz danych zaszły ogromne zmiany. Wprawdzie relacyjne bazy danych nadal są często używane i przez pewien czas to się nie zmieni, ale powstały nowe technologie bazodanowe, które mają na celu spełnianie potrzeb firm takich jak Amazon i Google. Do tych technologii zaliczamy m.in. Hadoop, Spark, NoSQL i NewSQL — są to rozproszone, skalowane systemy, zwykle wdrażane w klastrach. Dokładne omówienie tych technologii wykracza poza zakres tematyczny książki, choć mają one jedną ważną cechę współdzieloną z relacyjnymi bazami danych: język SQL.

Ponieważ organizacje często przechowują dane z wykorzystaniem wielu różnych technologii, istnieje potrzeba oddzielenia SQL od konkretnego serwera bazy danych oraz dostarczenia usługi, która będzie zapewniała obsługę wielu baz danych. Na przykład wygenerowanie raportu może wymagać połączenia danych pochodzących z Oracle, Hadoop, plików JSON, plików CVS i plików dzienników zdarzeń systemu UNIX. Powstała nowa generacja narzędzi przeznaczonych do spełniania tego typu wymagań. Najbardziej obiecującym z nich jest Apache Drill, czyli dostępny jako oprogramowanie open source silnik zapytań umożliwiający użytkownikom tworzenie zapytań przeznaczonych do pobierania informacji przechowywanych w większości baz danych lub systemów plików. Więcej informacji na temat Apache Drill znajdziesz w rozdziale 18.

Co się znajduje w magazynie danych?

Ogólnym celem czterech kolejnych rozdziałów jest wprowadzenie zapytań danych SQL ze szczególnym naciskiem na trzy główne klauzule zapytania `SELECT`. Ponadto przedstawię wiele przykładów wykorzystujących schemat Sakila (zaprezentuję go w następnym rozdziale), którego używam we wszystkich przykładach omówionych w książce. Mam nadzieję, że dzięki wykorzystaniu pojedynczej bazy danych będziesz mógł się skoncentrować na przykładach zamiast analizować za każdym razem zastosowane tabele. Jeżeli praca z tym samym zestawem tabel okaże się dla Ciebie nużąca, możesz śmiało zmodyfikować przykładową bazę danych i wzbogacić ją o dodatkowe tabele albo przygotować własną bazę danych do dalszych eksperymentów.

Gdy opanujesz podstawy, w pozostałych rozdziałach książki poznasz dodatkowe koncepcje, z których większość jest niezależna od siebie. Dlatego jeśli poczujesz się zdezorientowany, zawsze możesz przejść dalej i dopiero później powrócić do rozdziału sprawiającego trudność. Po zakończeniu lektury książki i wykonaniu wszystkich zamieszczonych w niej przykładów będziesz na dobrej drodze, aby stać się doświadczonym praktykiem SQL-a.

Jeżeli chcesz dowiedzieć się więcej na temat relacyjnych baz danych oraz historii skomputeryzowanych systemów baz danych i języka SQL, która została pokrótce omówiona w tym wprowadzeniu, oto kilka źródeł wartych lektury:

- C.J. Date, *Relacyjne bazy danych dla praktyków*, Helion;
- C.J. Date, *An Introduction to Database Systems*, wydanie 8., Addison-Wesley;
- C.J. Date, *The Database Relational Model: A Retrospective Review and Analysis*, Addison-Wesley;
- Sekcja *Database management system* w artykule Wikipedii na stronie *https://en.wikipedia.org/wiki/Database#Database_management_system*.

SQL. Znajdź cenne informacje w oceanie danych!

SQL jest idealnym narzędziem do pracy z danymi. Mimo upływu lat jego znaczenie nie maleje, a sam język wciąż jest unowocześniany i rozwijany. Dziś szczególnie przydają się jego ogromne możliwości w zakresie przetwarzania danych. Co ciekawe, SQL pozwala również na stosowanie technik służących do zarządzania ogromnymi zbiorami informacji czy korzystanie z nierelacyjnych baz danych. Osoba, która obok Pythona czy R radzi sobie z SQL i potrafi z morza danych wyodrębnić użyteczne informacje, jest wyjątkowo cennym pracownikiem.

To przystępny podręcznik, dzięki któremu programiści szybko opanują podstawy SQL — nauczą się tworzenia aplikacji bazodanowych, przeprowadzania zadań administracyjnych oraz generowania raportów. Ujęto tu takie zagadnienia jak zapytania SELECT, filtrowanie danych oraz ich konwersja, grupowanie i agregacja. Znalazło się tutaj także wprowadzenie do transakcji, przedstawiono też zasady tworzenia widoków, złączeń i ograniczeń. To wydanie zostało uzupełnione omówieniem funkcji analitycznych, strategii pracy z ogromnymi bazami danych oraz zagadnień związanych z big data. W każdym rozdziale zaprezentowano kluczowe koncepcje SQL, które dodatkowo wyjaśniono na podstawie wielu dokładnie omówionych przykładów. Ćwiczenia zamieszczone na końcu poszczególnych rozdziałów pomogą w sprawdzeniu i utrwaleniu zdobytej wiedzy.

Dzięki książce:

- opanujesz podstawy języka SQL i ważniejszych funkcji zaawansowanych
- zaczniesz pisać zapytania SQL
- nauczysz się tworzyć obiekty bazy danych
- poznasz sposoby współdziałania zbiorów danych i zapytań
- dowiesz się, jak konwertować i przetwarzać dane za pomocą funkcji wbudowanych SQL

Alan Beaulieu od ponad 25 lat projektuje, tworzy i implementuje niestandardowe aplikacje bazodanowe. Opracował internetowy kurs języka SQL dla Uniwersytetu Kalifornijskiego. Prowadzi firmę konsultingową. Specjalizuje się w implementacji rozwiązań dla sektorów usług finansowych i telekomunikacyjnych.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-7779-0

Cena: 79,00 zł