

Antywzorce języka SQL

Zobacz, jak tego nie robić!

- Jak nie projektować modelu logicznego i fizycznego bazy danych?
- Jak nie zadawać zapytań SQL?
- Jak nie wytwarzać aplikacji?



Helion



Bill Karwin

Tytuł oryginału: SQL Antipatterns: Avoiding the Pitfalls of Database Programming

Tłumaczenie: Mikołaj Szczepaniak

ISBN: 978-83-246-3482-8

Copyright © 2011 The Pragmatic Programmers, LLC
All rights reserved

Copyright © 2012 by Helion S.A.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/antysq.zip>

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

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/antysq>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Rozdział 1. Wprowadzenie	11
1.1. Dla kogo jest ta książka	13
1.2. Co znajduje się w tej książce	14
1.3. Czego nie ma w tej książce	16
1.4. Konwencje	18
1.5. Przykładowa baza danych	19
1.6. Podziękowania	22
Część I. Antywzorce logicznego projektu bazy danych	23
Rozdział 2. Przechodzenie na czerwonym świetle	25
2.1. Cel: przechowywanie atrybutów wielowartościowych	26
2.2. Antywzorec: listy elementów oddzielonych przecinkami	27
2.3. Jak rozpoznać ten antywzorec	31
2.4. Usprawiedliwione użycia tego antywzorca	31
2.5. Rozwiązanie: utworzenie tabeli łączącej	32
Rozdział 3. Naiwne drzewa	37
3.1. Cel: przechowywanie i uzyskiwanie hierarchii	38
3.2. Antywzorec: zawsze odwołuj się do rodzica	39
3.3. Jak rozpoznać ten antywzorec	43
3.4. Usprawiedliwione użycia tego antywzorca	44
3.5. Rozwiązanie: należy użyć alternatywnych modeli drzew	46

Rozdział 4. Identyfikator potrzebny od zaraz	61
4.1. Cel: wypracowanie konwencji dla kluczy głównych	62
4.2. Antywzorec: jeden rozmiar dla wszystkich	64
4.3. Jak rozpoznać ten antywzorec	69
4.4. Usprawiedliwione użycia tego antywzorca	70
4.5. Rozwiązanie: klucz skrojony na miarę	70
Rozdział 5. Wpis bez klucza	75
5.1. Cel: uproszczenie architektury bazy danych	76
5.2. Antywzorec: rezygnacja z ograniczeń	77
5.3. Jak rozpoznać ten antywzorec	80
5.4. Usprawiedliwione użycia tego antywzorca	81
5.5. Rozwiązanie: deklarowanie ograniczeń	81
Rozdział 6. Encja-atrybut-wartość	85
6.1. Cel: obsługa zmiennych atrybutów	86
6.2. Antywzorec: zastosowanie uniwersalnej tabeli atrybutów	87
6.3. Jak rozpoznać ten antywzorec	93
6.4. Usprawiedliwione użycia tego antywzorca	94
6.5. Rozwiązanie: modelowanie podtypów	95
Rozdział 7. Związki polimorficzne	103
7.1. Cel: odwołania do wielu rodziców	104
7.2. Antywzorec: zastosowanie dwuzadaniowego klucza obcego	105
7.3. Jak rozpoznać ten antywzorec	109
7.4. Usprawiedliwione użycia tego antywzorca	110
7.5. Rozwiązanie: uproszczenie relacji	110
Rozdział 8. Atrybuty wielokolumnowe	117
8.1. Cel: przechowywanie atrybutów wielowartościowych	118
8.2. Antywzorec: utworzenie wielu kolumn	118
8.3. Jak rozpoznać ten antywzorec	122
8.4. Usprawiedliwione użycia tego antywzorca	123
8.5. Rozwiązanie: utworzenie tabeli zależnej	124

Rozdział 9. Tribble metadanych	127
9.1. Cel: zapewnienie skalowalności	128
9.2. Antywzorzec: klonowanie tabel lub kolumn	129
9.3. Jak rozpoznać ten antywzorzec	134
9.4. Usprawiedliwione użycia tego antywzorca	135
9.5. Rozwiązanie: partycjonowanie i normalizacja	137
Część II. Antywzorce fizycznego projektu bazy danych ...	141
Rozdział 10. Błędy zaokrąglenia	143
10.1. Cel: stosowanie liczb ułamkowych zamiast liczb całkowitych	144
10.2. Antywzorzec: stosowanie typu danych FLOAT	144
10.3. Jak rozpoznać ten antywzorzec	149
10.4. Usprawiedliwione użycia tego antywzorca	150
10.5. Rozwiązanie: stosowanie typu danych NUMERIC	150
Rozdział 11. 31 smaków	153
11.1. Cel: ograniczenie zakresu danych kolumny do określonych wartości	154
11.2. Antywzorzec: określanie wartości w definicji kolumny	155
11.3. Jak rozpoznać ten antywzorzec	159
11.4. Usprawiedliwione użycia tego antywzorca	160
11.5. Rozwiązanie: określanie akceptowanych wartości na poziomie danych	160
Rozdział 12. Pliki-widma	165
12.1. Cel: przechowywanie obrazów i innych dużych zasobów	166
12.2. Antywzorzec: przekonanie o konieczności stosowania plików	167
12.3. Jak rozpoznać ten antywzorzec	171
12.4. Usprawiedliwione użycia tego antywzorca	172
12.5. Rozwiązanie: stosowanie typów danych BLOB	173
Rozdział 13. Strzelanie indeksami	177
13.1. Cel: optymalizacja wydajności	178
13.2. Antywzorzec: stosowanie indeksów bez żadnego planu	179
13.3. Jak rozpoznać ten antywzorzec	184
13.4. Usprawiedliwione użycia tego antywzorca	184
13.5. Rozwiązanie: zasada MENTOR dla indeksów	185

Część III. Antywzorze zapytań	193
Rozdział 14. Strach przed nieznanym	195
14.1. Cel: odróżnianie brakujących wartości	196
14.2. Antywzorzec: stosowanie NULL jako zwykłej wartości (lub odwrotnie)	197
14.3. Jak rozpoznać ten antywzorzec	201
14.4. Usprawiedliwione użycia tego antywzorca	202
14.5. Rozwiązanie: stosowanie NULL jako unikatowej wartości	203
Rozdział 15. Niejasne grupy	209
15.1. Cel: uzyskiwanie wiersza z największą wartością w skali grupy	210
15.2. Antywzorzec: odwołania do niegrupowanych kolumn	211
15.3. Jak rozpoznać ten antywzorzec	214
15.4. Usprawiedliwione użycia tego antywzorca	215
15.5. Rozwiązanie: jednoznaczne stosowanie kolumn	216
Rozdział 16. Losowy wybór	223
16.1. Cel: uzyskiwanie przypadkowego wiersza	224
16.2. Antywzorzec: losowe sortowanie danych	225
16.3. Jak rozpoznać ten antywzorzec	226
16.4. Usprawiedliwione użycia tego antywzorca	227
16.5. Rozwiązanie: brak ustalonej kolejności... ..	228
Rozdział 17. Wyszukiwarka nędzarza	233
17.1. Cel: pełne przeszukiwanie tekstu	234
17.2. Antywzorzec: predykaty dopasowywania wzorców	235
17.3. Jak rozpoznać ten antywzorzec	236
17.4. Usprawiedliwione użycia tego antywzorca	237
17.5. Rozwiązanie: stosowanie narzędzi odpowiednio dobrych do realizowanych zadań	237
Rozdział 18. Zapytanie-spaghetti	251
18.1. Cel: ograniczenie liczby zapytań SQL-a	252
18.2. Antywzorzec: rozwiązanie złożonego problemu w jednym kroku	253
18.3. Jak rozpoznać ten antywzorzec	256
18.4. Usprawiedliwione użycia tego antywzorca	257
18.5. Rozwiązanie: dziel i zwyciężaj	257

Rozdział 19. Ukryte kolumny	263
19.1. Cel: ograniczyć ilość wpisywanego kodu	264
19.2. Antywzorzec: skrót prowadzący na manowce	265
19.3. Jak rozpoznać ten antywzorzec	267
19.4. Usprawiedliwione użycia tego antywzorca	268
19.5. Rozwiązanie: należy wprost nazywać kolumny	269
Część IV. Antywzorce wytwarzania aplikacji	273
Rozdział 20. Czytelne hasła	275
20.1. Cel: odzyskiwanie lub resetowanie hasel	276
20.2. Antywzorzec: przechowywanie hasel w formie zwykłego tekstu	276
20.3. Jak rozpoznać ten antywzorzec	279
20.4. Usprawiedliwione użycia tego antywzorca	280
20.5. Rozwiązanie: przechowywanie zabezpieczonych kodów hasel	281
Rozdział 21. Wstrzykiwanie SQL-a	289
21.1. Cel: pisanie dynamicznych zapytań języka SQL	290
21.2. Antywzorzec: wykonywanie niesprawdzonych danych wejściowych jako kodu	291
21.3. Jak rozpoznać ten antywzorzec	299
21.4. Usprawiedliwione użycia tego antywzorca	300
21.5. Rozwiązanie: nie ufać nikomu	301
Rozdział 22. Obsesja czystości pseudokluczy	309
22.1. Cel: sprzątnięcie danych	310
22.2. Antywzorzec: wypełnianie luk	311
22.3. Jak rozpoznać ten antywzorzec	314
22.4. Usprawiedliwione użycia tego antywzorca	314
22.5. Rozwiązanie: zapomnieć o problemie	315
Rozdział 23. Przymykanie oczu na zło	321
23.1. Cel: pisać mniej kodu	322
23.2. Antywzorzec: ścinanie zakrętów	323
23.3. Jak rozpoznać ten antywzorzec	326

23.4. Usprawiedliwione użycia tego antywzorca	327
23.5. Rozwiązanie: elegancka obsługa błędów	327
Rozdział 24. Immunitet dyplomatyczny	331
24.1. Cel: stosowanie najlepszych praktyk	332
24.2. Antyworzec: kod SQL-a jako obywatel drugiej kategorii	333
24.3. Jak rozpoznać ten antyworzec	334
24.4. Usprawiedliwione użycia tego antywzorca	335
24.5. Rozwiązanie: ustanowienie możliwie szerokiej kultury jakości	336
Rozdział 25. Magiczna fasola	347
25.1. Cel: upraszczanie modeli w architekturze model-widok-komponent	348
25.2. Antyworzec: model jako rekord aktywny (Active Record)	350
25.3. Jak rozpoznać ten antyworzec	356
25.4. Usprawiedliwione użycia tego antywzorca	357
25.5. Rozwiązanie: model zawierający rekord aktywny	358
Dodatki	365
Dodatek A. Reguły normalizacji	367
A.1. Co to oznacza, że baza jest relacyjna?	368
A.2. Mity dotyczące normalizacji	371
A.3. Czym jest normalizacja?	372
A.4. Zdrowy rozsądek	383
Dodatek B. Bibliografia	385
Skorowidz	387

Rozdział 10.

Błędy zaokrągleń

10,0 razy 0,1 prawie nigdy nie jest równe 1,0.

Brian Kernighan

Nasz szef prosi nas o wygenerowanie raportu podsumowującego koszty pracy programistów według projektów (koszt ma być wyznaczany na podstawie czasu poświęconego na usuwanie poszczególnych błędów). Każdy programista reprezentowany w tabeli Konta otrzymuje inną stawkę godzinową, zatem musimy nie tylko rejestrować liczby godzin potrzebnych do usunięcia poszczególnych błędów (w tabeli Bledy), ale też mnożyć te wartości przez atrybut `stawka_godzinowa` zdefiniowany dla wyznaczonych programistów.

Plik `Błędy-zaokrągleń/wprowadzenie/cost-per-bug.sql`

```
SELECT b.id_bledu, b.godziny * k.stawka_godzinowa AS koszt_bledu
FROM Bledy AS b
JOIN Konta AS k ON (b.przypisany_do = k.id_konta);
```

Obsługa tego zapytania wymaga utworzenia nowych kolumn w tabelach `Bledy` i `Konta`. Obie kolumny powinny obsługiwać wartości ułamkowe, ponieważ musimy śledzić koszty możliwie precyzyjnie. Decydujemy się więc zdefiniować nowe kolumny jako `FLOAT`, ponieważ właśnie ten typ danych obsługuje wartości ułamkowe.

```
Plik Błędy-zaokrągleń/wprowadzenie/float-columns.sql
```

```
ALTER TABLE Bledy ADD COLUMN godziny FLOAT;  
ALTER TABLE Konta ADD COLUMN stawka_godzinowa FLOAT;
```

Obie kolumny aktualizujemy na podstawie zapisów w dziennikach prac nad eliminowaniem błędów i stawek godzinowych programistów. Testujemy jeszcze generowanie raportu i jesteśmy gotowi do codziennego przygotowywania potrzebnych zestawień.

Następnego dnia do naszego gabinetu wchodzi szef z kopią raportu o kosztach projektu. „Te liczby nie zgadzają się” — wycedził przez zaciśnięte zęby. „Dla porównania wykonałem te obliczenia ręcznie i okazało się, że Twój raport jest nieprecyzyjny — nieznacznie, różnice wynoszą kilka złotych. Jak to wyjaśnisz?”. Jesteśmy przerażeni. Co może nie zgadzać się w tak prostych obliczeniach?

10.1. Cel: stosowanie liczb ułamkowych zamiast liczb całkowitych

Liczba całkowita jest wyjątkowo przydatnym typem danych, ale umożliwia reprezentowanie tylko wartości całkowitoliczbowych, jak 1, 327 czy -19. Ten typ danych nie oferuje możliwości reprezentowania wartości ułamkowych, na przykład 2,5. Jeśli więc potrzebujemy liczb gwarantujących większą precyzję niż liczby całkowite, musimy zastosować inny typ danych. Na przykład kwoty pieniężne zwykle są reprezentowane przez liczby z dwoma miejscami dziesiętnymi, na przykład 19,95 zł.

Naszym celem jest przechowywanie wartości numerycznych niebędących liczbami całkowitymi i wykorzystywanie tych liczb w obliczeniach arytmetycznych. Istnieje jeszcze jeden cel, który jednak jest na tyle oczywisty, że rzadko się o nim wprost mówi — wyniki naszych obliczeń arytmetycznych muszą być **prawidłowe**.

10.2. Antywzorzec: stosowanie typu danych FLOAT

Większość języków programowania obsługuje typy danych dla liczb całkowitych (zwykle pod nazwą `float` lub `double`). Język SQL obsługuje podobny typ danych nazwany `FLOAT`. Wielu programistów odruchowo używa typu

danych `FLOAT` SQL-a wszędzie tam, gdzie potrzebują ułamkowych danych liczbowych, ponieważ są przyzwyczajeni do programowania z wykorzystaniem tego typu danych.

Typ danych `FLOAT` języka SQL, tak jak typ danych `float` w większości języków programowania, koduje liczby rzeczywiste w formacie binarnym zgodnie ze standardem IEEE 754. Efektywne używanie tego typu wymaga zrozumienia wybranych cech liczb zmiennoprzecinkowych reprezentowanych w tym formacie.

Zaokrąglanie z konieczności

Wielu programistów nie zdaje sobie sprawy z istnienia pewnej ważnej cechy tego formatu liczb zmiennoprzecinkowych: nie wszystkie wartości, które możemy zapisywać w formie liczb dziesiętnych, mogą być reprezentowane w formie binarnej. W tej sytuacji niektóre wartości wymagają zaokrąglania do bardzo zbliżonych wartości.

Aby lepiej zrozumieć kontekst tych zaokrągleń, wystarczy przeanalizować przypadek prostej liczby wymiernej, na przykład jednej trzeciej, która jest reprezentowana przez okresowy ułamek dziesiętny $0,333\dots$. Prawdziwej wartości nie można reprezentować w formie dziesiętnej, ponieważ musielibyśmy zapisać nieskończoną liczbę cyfr. Liczba cyfr po przecinku to tzw. precyzja reprezentacji odpowiedniej wartości, zatem w pełni prawidłowa reprezentacja ułamka okresowego wymagałaby **nieskończonej precyzji**.

Rozwiązaniem kompromisowym jest stosowanie **skończonej precyzji**, tj. wybór wartości liczbowej możliwie zbliżonej do oryginalnej liczby ułamkowej, na przykład $0,333$. Wadą tego rozwiązania jest to, że ta wartość nie jest dokładnie tą samą liczbą, którą chcemy reprezentować.

$$\begin{aligned} 1/3+1/3+1/3 &= 1,000 \\ 0,333+0,333+0,333 &= 0,999 \end{aligned}$$

Nawet jeśli zwiększymy precyzję, suma trzech przybliżeń jednej trzeciej wciąż nie będzie równa oczekiwanej wartości $1,0$. Kompromis polegający na stosowaniu skończonej precyzji dla ułamków okresowych jest jednak niezbędny.

$$\begin{aligned} 1/3+1/3+1/3 &= 1,000000 \\ 0,333333+0,333333+0,333333 &= 0,999999 \end{aligned}$$

Oznacza to, że niektóre prawidłowe wartości liczbowe, które możemy sobie bez trudu wyobrazić, w ogóle nie mogą być reprezentowane z zastosowaniem metody skończonej precyzji. Część programistów uważa, że

takie rozwiązanie jest usprawiedliwione — skoro wartości złożonych z nieskończonej liczby cyfr i tak nie da się zapisać, każda zapisywana przez nas liczba z natury rzeczy ma skończoną precyzję i tak też powinna być przechowywana w formie binarnej, prawda? Niestety nie.

Zgodnie ze standardem IEEE 754 liczby zmiennoprzecinkowe są reprezentowane w systemie liczbowym o podstawie 2. Oznacza to, że wartości wymagające nieskończonej precyzji w systemie binarnym nie pokrywają się ze zbiorem wartości, które wymagają takiej reprezentacji w systemie dziesiętnym. Niektóre wartości, które wymagają skończonej precyzji w systemie dziesiętnym, na przykład 59,95, wymagają nieskończonej precyzji, jeśli miałyby być dokładnie reprezentowane w systemie binarnym. Typ danych FLOAT nie oferuje takich możliwości, zatem stosuje najbliższą obsługiwaną wartość w systemie liczbowym o podstawie 2, czyli wartość odpowiadającą liczbie 59.950000762939 w systemie dziesiętnym.

Niektóre wartości przypadkowo wymagają skończonej precyzji w obu formatach. Jeśli zrozumiemy szczegóły przechowywania liczb w formacie IEEE 754, teoretycznie będziemy potrafili przewidywać, jak poszczególne wartości dziesiętne będą reprezentowane w formacie binarnym. W praktyce jednak większość programistów nie wykonuje podobnych obliczeń dla każdej stosowanej przez siebie liczby zmiennoprzecinkowej. Nie możemy zagwarantować, że kolumna FLOAT w bazie danych będzie dostatecznie precyzyjna, zatem nasza aplikacja powinna zakładać, że każda wartość w tej kolumnie mogła zostać zaokrąglona.

Niektóre bazy danych obsługują pokrewne typy danych nazwane DOUBLE PRECISION i REAL. Precyzja oferowana przez te typy danych i sam typ FLOAT zależy co prawda od implementacji bazy danych, ale wszystkie te typy reprezentują wartości zmiennoprzecinkowe ze skończoną liczbą cyfr binarnych, zatem sposób zaokrąglania liczb we wszystkich przypadkach jest podobny.

Stosowanie typu FLOAT w języku SQL

Niektóre bazy danych kompensują wspomnianą niedokładność i wyświetlają właściwe wartości.

```
Plik Błędy-zaokrągleń/anty/select-rate.sql
```

```
SELECT stawka_godzinowa FROM Konta WHERE id_konta = 123;
```

Zwraca: 59.95

Poznać format IEEE 754

Pierwsze propozycje dotyczące standardowego formatu binarnego dla liczb zmiennoprzecinkowych pojawiły się jeszcze w 1979 roku. Specyfikacja, która ostatecznie zyskała status standardu w 1985 roku, jest obecnie powszechnie implementowana w rozmaitych formach oprogramowania, przede wszystkim w językach programowania i mikroprocesorach.

Format składa się z trzech pól niezbędnych do zakodowania wartości zmiennoprzecinkowej: pola **części ułamkowej**, pola **wykładnika**, do którego należy podnieść tę część ułamkową, oraz jednobitowego pola **znaku**.

Jedną z zalet standardu IEEE 754 jest właśnie stosowanie wykładnika, dzięki czemu można w tym formacie reprezentować zarówno bardzo małe, jak i bardzo duże wartości. Format obsługuje nie tylko liczby rzeczywiste, ale też dużo większy przedział wartości niż tradycyjne, stałoprzecinkowe formaty reprezentacji liczb całkowitych. Jeszcze większy przedział reprezentowanych wartości oferuje format podwójnej precyzji. Oznacza to, że opisywane formaty dobrze sprawdzają się w zastosowaniach naukowych.

Bodaj najbardziej popularnym zastosowaniem ułamkowych wartości liczbowych jest reprezentowanie kwot pieniężnych. Stosowanie standardu IEEE 754 dla tego rodzaju wartości nie jest konieczne, ponieważ opisany w tym rozdziale format skalowanych liczb dziesiętnych pozwala równie łatwo i bardziej precyzyjnie obsługiwać kwoty pieniężne.

Dobrymi źródłami wiedzy o tym formacie są artykuł opublikowany na Wikipedii (http://pl.wikipedia.org/wiki/IEEE_754) oraz artykuł Davida Goldberga zatytułowany „What Every Computer Scientist Should Know About Floating-Point Arithmetic” [Gol91].

Artykuł Goldberga został też przedrukowany (jest dostępny pod adresem <http://www.validlab.com/goldberg/paper.pdf>).

Rzeczywista wartość przechowywana w kolumnie typu FLOAT nie musi jednak odpowiadać tej wartości. Wystarczy pomnożyć tę wartość przez miliard, aby odkryć pewne rozbieżności:

Plik Błędy-zaokrągleń/anty/magnify-rate.sql

```
SELECT stawka_godzinowa * 1000000000 FROM Konta WHERE id_konta = 123;
```

Zwraca: 59950000762. 939

Można było oczekiwać, że wartość zwrócona przez poprzednie zapytanie po przemnożeniu wyniesie 5995000000,000. Jak widać, wartość 59,95 została zaokrąglona do wartości, która może być reprezentowana w systemie ze skończoną precyzją oferowanym przez format binarny IEEE 754. W tym przypadku różnica jest mniejsza niż jedna milionowa, zatem jej dokładność powinna wystarczyć na potrzeby wielu obliczeń.

Przybliżenie nie gwarantuje jednak wystarczającej dokładności dla innych obliczeń. Dobrym przykładem jest użycie wartości typu `FLOAT` w wyrażeniu porównującym liczby.

Plik Błędy-zaokrągleń/anty/inexact.sql

```
SELECT * FROM Konta WHERE stawka_godzinowa = 59.95;
```

Wynik: zbiór pusty, brak pasujących wierszy

Jak wiemy, wartość przechowywana w kolumnie `stawka_godzinowa` w rzeczywistości jest nieznacznie większa niż 59.95. W tej sytuacji, mimo że przypisaliliśmy tej kolumnie wartość 59.95 w wierszu z identyfikatorem konta 123, zapytanie nie jest dopasowywane do powyższego zapytania.

Typowym obejściem tego problemu jest traktowanie wartości zmiennoprzecinkowych jako „praktycznie równe”, jeśli dzieląca je różnica nie przekracza jakiegoś niewielkiego progu. Wystarczy odjąć jedną wartość od drugiej i użyć dostępnej w języku SQL funkcji wartości bezwzględnej `ABS()`, aby uzyskana różnica była dodatnia. Jeśli wynik jest równy zero, obie wartości są dokładnie równe. Jeśli wynik jest dostatecznie mały, obie wartości można traktować jako praktycznie równe. Poniższe zapytanie prawidłowo odnajduje interesujący nas wiersz:

Plik Błędy-zaokrągleń/anty/threshold.sql

```
SELECT * FROM Konta WHERE ABS(stawka_godzinowa - 59.95) < 0.000001;
```

Różnica jest jednak na tyle duża, że porównanie z nieco większą precyzją zakończy się niepowodzeniem:

Plik Błędy-zaokrągleń/anty/threshold.sql

```
SELECT * FROM Konta WHERE ABS(stawka_godzinowa - 59.95) < 0.0000001;
```

Dobór właściwego progu zależy od konkretnej liczby, ponieważ wartość bezwzględna różnicy dzielącej wartości dziesiętne od zaokrąglonych reprezentacji binarnych jest różna w przypadku poszczególnych liczb.

Innym przykładem problemu wynikającego z nieprecyzyjnego charakteru typu `FLOAT` są obliczenia polegające na agregowaniu wielu wartości. Jeśli na przykład użyjemy funkcji `SUM()` do dodania wszystkich wartości zmiennoprzecinkowych w jakiejś kolumnie, błędy zaokrążeń poszczególnych liczb skumulują się w wyznaczonej sumie.

Plik Błędy-zaokrążeń/anty/cumulative.sql

```
SELECT SUM(b.godziny * k.stawka_godzinowa) AS koszty_projektu
FROM Bledy AS b
JOIN Konta AS k ON (b.przypisany_do = k.id_konta);
```

Skumulowany efekt niedokładności liczb zmiennoprzecinkowych jest jeszcze bardziej widoczny podczas wyznaczania zagregowanego iloczynu zbioru liczb (zamiast ich sumy). Poszczególne różnice wydają się niewielkie, ale z czasem mocno rosną. Jeśli na przykład pomnożymy wartość 1 przez współczynnik równy dokładnie 1,0, wynik zawsze będzie wynosił 1. Nie ma znaczenia, ile razy zastosujemy ten mnożnik. Jeśli jednak użyjemy współczynnika 0,999, wynik będzie inny. Jeśli kolejno pomnożymy liczbę 1 przez wartość 0,999 tysiąc razy, otrzymamy wynik równy około 0,3677. Im więcej operacji mnożenia wykonamy, tym większa będzie ta rozbieżność.

Dobrym przykładem stosowania operacji wielokrotnego mnożenia jest wyznaczanie łącznego oprocentowania na potrzeby kalkulacji finansowych. Używanie niedokładnych liczb zmiennoprzecinkowych powoduje błędy, które początkowo wydają się zupełnie niegroźne, ale z czasem, skumulowane, zaczynają stwarzać poważne problemy. Stosowanie precyzyjnych wartości w aplikacjach finansowych jest więc bardzo ważne.

10.3. Jak rozpoznać ten antywzorzec

Niemal każde użycie typów danych `FLOAT`, `REAL` lub `DOUBLE PRECISION` jest podejrzane. Większość aplikacji korzystających z liczb zmiennoprzecinkowych w rzeczywistości nie potrzebuje przedziału wartości obsługiwanego przez formaty zgodne ze standardem IEEE 754.

Korzystanie z typów danych `FLOAT` w języku SQL wydaje się o tyle naturalne, że podobny typ (często nawet pod tą samą nazwą) występuje w większości języków programowania. Okazuje się jednak, że można wybrać lepszy typ danych.

10.4. Usprawiedliwione użycia tego antywzorca

FLOAT jest dobrym typem danych w sytuacji, gdy potrzebujemy liczb rzeczywistych z przedziału większego niż ten obsługiwany przez typy INTEGER i NUMERIC. Naukowe aplikacje często wskazuje się jako przykład uzasadnionego stosowania typu FLOAT.

W systemie Oracle typu danych FLOAT używa się do wyrażania dokładnie skalowanych wartości liczbowych, zaś do reprezentowania niedokładnych wartości numerycznych stosuje się typ danych BINARY_FLOAT (zgodny ze standardem kodowania IEEE 754).

10.5. Rozwiązanie: stosowanie typu danych NUMERIC

Zamiast typu FLOAT i typów pokrewnych możemy stosować typy danych NUMERIC lub DECIMAL języka SQL dla liczb ułamkowych stałej precyzji.

```
Plik Błędy-zaokrągleń/roz/numeric-columns.sql
```

```
ALTER TABLE Bledy ADD COLUMN godziny NUMERIC(9,2);
```

```
ALTER TABLE Konta ADD COLUMN stawka_godzinowa NUMERIC(9,2);
```

Wymienione typy danych umożliwiają dokładne reprezentowanie wartości numerycznych z maksymalną precyzją określoną podczas definiowania odpowiednich kolumn. Precyzję należy określić w formie argumentu typu danych — obowiązująca składnia przypomina trochę sposób określania długości typu danych VARCHAR. Precyzja to łączna liczba cyfr dziesiętnych, których możemy używać dla wartości w tak zdefiniowanej kolumnie. Precyzja równa 9 oznacza, że możemy przechowywać takie wartości jak 123456789, ale najprawdopodobniej nie będziemy mogli obsłużyć wartości równej 1234567890.¹

¹ W niektórych systemach baz danych rozmiar tej kolumny jest zaokrąglany w górę do najbliższego bajta, słowa lub podwójnego słowa, zatem w pewnych przypadkach maksymalna wartość w kolumnie typu NUMERIC może składać się z większej liczby cyfr, niż to wynika ze wskazanej precyzji.

Istnieje też możliwość określenia skali za pośrednictwem drugiego argumentu tego typu danych. Skala decyduje o liczbie cyfr na prawo od przecinka oddzielającego część całkowitą od ułamkowej. Cyfry skali są odliczane od liczby cyfr precyzji, zatem precyzja 9 ze skalą 2 oznaczają, że możemy przechowywać takie wartości jak 1234567.89, ale już nie wartości 12345678.91 czy 123456.789.

Określane przez nas precyzja i skala są stosowane dla danej kolumny we wszystkich wierszach tabeli. Innymi słowy, nie możemy przechowywać wartości ze skalą 2 w części wierszy i wartości ze skalą 4 w pozostałych. W języku SQL to naturalne, że typ danych kolumny jest konsekwentnie stosowany dla wszystkich wierszy (tak jak w przypadku kolumny typu VARCHAR(20), gdzie każdy wiersz może zawierać łańcuch określonej długości).

Zaletą typów NUMERIC i DECIMAL jest możliwość przechowywania liczb wymiernych bez ryzyka ich zaokrąglenia (jak w przypadku typu FLOAT). Po zapisaniu wartości 59.95 możemy być pewni, że w bazie danych jest przechowywana dokładnie ta liczba. Jeśli porównamy ją ze stałą wartością 59.95, okaże się, że obie wartości są sobie równe.

Plik Błędy-zaokrągleń/roz/exact.sql

```
SELECT stawka_godzinowa FROM Konta WHERE stawka_godzinowa = 59.95;
```

Zwraca: 59.95

Podobnie, jeśli pomnożymy tę wartość przez miliard, otrzymamy oczekiwaną wartość:

Plik Błędy-zaokrągleń/roz/magnify-rate-exact.sql

```
SELECT stawka_godzinowa * 1000000000 FROM Konta WHERE stawka_godzinowa = 59.95;
```

Zwraca: 59950000000

Typy danych NUMERIC i DECIMAL zachowują się identycznie; nie powinny występować żadne różnice w ich działaniu. Istnieje też synonim DEC dla typu danych DECIMAL.

Nadal nie możemy przechowywać wartości wymagających nieskończonej precyzji, na przykład jednej trzeciej. Proponowane typy umożliwiają nam jednak przechowywanie dokładnej reprezentacji liczb w formie, w której zapisujemy je w systemie dziesiętnym.

Jeśli więc potrzebujemy dokładnych wartości dziesiętnych, powinniśmy posługiwać się typem danych `NUMERIC`. Typ danych `FLOAT` nie może reprezentować wielu dziesiętnych liczb wymiernych, zatem wartości tego typu należy zawsze traktować jako niedokładne.

Jeśli tylko możemy tego uniknąć, nie powinniśmy używać typu `FLOAT`.

Skorowidz

.dump, 169

1NF, *Patrz* pierwsza postać normalna

2NF, *Patrz* druga postać normalna

3NF, *Patrz* trzecia postać normalna

4NF, *Patrz* czwarta postać normalna

5NF, *Patrz* piąta postać normalna

A

ABS(), 148

Active Record, 109, 350, 351, 352, 353, 357, 358

Adams, Douglas, 85

adjacency list, *Patrz* lista sąsiedztwa

agregujące zapytania, tworzenie, 28, 33

aktualizacje kaskadowe, 82, 83

algorytm

SHA-1, 282

SHA-256, 281

Anemic Domain Model,

Patrz antywzorzec, anemiczny model dziedzinowy

anomalia, 375

ANSI SQL, standard, 180

antywzorzec, 14, 16

31 smaków, 153

rozpoznawanie, 159

usprawiedliwione użycie, 160

anemiczny model dziedzinowy, 353

atrybuty wielokolumnowe, 117, 123

rozpoznawanie, 122

usprawiedliwione użycie, 123

błędy zaokrąglenia, 143

rozpoznawanie, 149

usprawiedliwione użycie, 150

czytelne hasła, 275

rozpoznawanie, 279

usprawiedliwione użycie, 280

encja-atrybut-wartość, 85, 86

rozpoznawanie, 93

usprawiedliwione użycie, 94

związki encji, 88

identyfikator potrzebny od zaraz, 61

rozpoznawanie, 69

usprawiedliwione użycie, 70

immunitet dyplomatyczny, 331

rozpoznawanie, 334

usprawiedliwione użycie, 335

antywzorzec

- losowy wybór, 223
 - rozpoznawanie, 226
 - usprawiedliwione użycie, 227
- magiczna fasola, 347
 - rozpoznawanie, 356
 - testowanie architektury MVC, 355
 - usprawiedliwione użycie, 357
- naiwne drzewa, 37
 - rozpoznawanie, 43
 - usprawiedliwione użycie, 44
- niejasne grupy, 209
 - rozpoznawanie, 214
 - usprawiedliwione użycie, 215
- obsesja czystości pseudokluczy, 309
 - rozpoznawanie, 314
 - usprawiedliwione użycie, 314
- pliki-widma, 165
 - rozpoznawanie, 171
 - usprawiedliwione użycie, 172
- przechodzenie na czerwonym świetle, 25, 26, 123
 - rozpoznawanie, 31
 - usprawiedliwione użycie, 31
- przymykanie oczu na zło, 321
 - rozpoznawanie, 326
 - usprawiedliwione użycie, 327
- strach przed nieznanym, 195
 - rozpoznawanie, 201
 - usprawiedliwione użycie, 202
- strzelanie indeksami, 177, 185
 - rozpoznawanie, 184
 - usprawiedliwione użycie, 184
- tribble metadanych, 127
 - rozpoznawanie, 134
 - usprawiedliwione użycie, 135
- ukryte kolumny, 263
 - rozpoznawanie, 267
 - usprawiedliwione użycie, 268

- wpis bez klucza, 75
 - rozpoznawanie, 80
 - usprawiedliwione użycie, 81
- wstrzykiwanie SQL-a, 289
 - rozpoznawanie, 299
 - usprawiedliwione użycie, 300
- wyszukiwarka nędzarka, 233
 - rozpoznawanie, 236
 - usprawiedliwione użycie, 237
- zapytanie-spaghetti, 251
 - rozpoznawanie, 256
 - usprawiedliwione użycie, 257
- złotego młotka, 351
- związki polimorficzne, 103
 - diagram związków encji, 105
 - podobieństwo do antywzorca encja-atrybut-wartość, 106
 - rozpoznawanie, 109
 - usprawiedliwione użycie, 110
- Apache Lucene, 245
- archiwizacja, 135
- atrybuty wielowartościowe,
 - przechowywanie, 118
- AUTO_INCREMENT, 63
- AVG(), 28

B

- Babbage, Charles, 177
- Baruch, Bernard, 309
- bazy danych
 - kopie zapasowe, 169
 - relacyjne, 368
 - spójność, 79
 - uproszczenie architektury, 76
 - wydajność, 178
- BCNF, *Patrz* Boyce'a-Codda,
 - postać normalna
- Berkeley DB, 94

BLOB, 173, 174
błędy, obsługa, 327
Bohr, Niels, 11
Born, Max, 209
Boyce'a-Codda, postać normalna, 377,
378

C

Cassandra, 94
CAST(), 295
CATSEARCH(), 239
CHECK, 155, 157, 159, 160
Class Table Inheritance, 99
close(), 327
closure table, *Patrz* tabela domknięcia
COALESCE(), 113, 207
Codd, E. F., 86, 201
COMMIT, 168
common table expression, *Patrz*
wspólne wyrażenie tablicowe
Concrete Table Inheritance, 97, 98
CONNECT BY PRIOR, 45, 58
CONTAINS(), 240, 241
CONTEXT, 239
ConText, moduł, 239
ConvertEmptyStringToNull, 203
Cosby, Bill, 182
CouchDB, 94
COUNT(), 28, 41
covering index, *Patrz* indeksy
pokrywające
Coveyou, Robert R., 223
CREATE INDEX, 180
CRUD, 350
CTE, *Patrz* wspólne wyrażenie
tablicowe
CTI, *Patrz* Concrete Table Inheritance
CTXCAT, 239

CTXRULE, 240
CTXXPATH, 240
czwarta postać normalna, 379, 380

D

dane hierarchiczne, przechowywanie,
58, 59
dane początkowe, 341
dane semistrukturalne, 100
DAO, 359
Date, C. J., 201
DEC, *Patrz* DECIMAL
DECIMAL, 150, 151
zalety, 151
deklaratywny język programowania, 12
DELETE, 168, 179
denormalizacja, 31
diagram interakcji klas, 355
diagram związków encji, 18, 19
dla listy sąsiedztwa, 39
dla tabeli łączącej, 32
DISTINCT, 214
DKNF, *Patrz* dziedzina-klucz, postać
normalna
dług techniczny, 332
dokumentacja, 336
bezpieczeństwo SQL-a, 338
diagram związków encji, 337
infrastruktura bazy danych, 339
odwzorowania obiektowo-relacyjne,
339
procedury składowane, 338
relacje, 338
tabela, kolumny, perspektywy, 337
wyzwalacze, 338
domain-Key normal form, *Patrz*
dziedzina-klucz, postać normalna
Don't Repeat Yourself, *Patrz* nie
powtarzaj się, zasada

dopasowywanie wzorców, 235, 236, 237
DOUBLE PRECISION, 146
druga postać normalna, 374, 375
DRY, *Patrz* nie powtarzaj się, zasada
drzewiasta struktura danych, 38
 korzeń, 38
 liście, 38
 przykłady stosowania, 38
 węzeł, 38
 węzły niebędące liśćmi, 38
drzewo
 dodawanie liści, 42
 usunięcie poddrzewa, 42
 usunięcie węzła, 42, 43
 zmiana położenia węzła
 lub poddrzewa, 42
dynamiczny SQL, 290
dziedzina-klucz, postać normalna, 382

E

EAW, *Patrz* antywzorzec, encja-
 atrybut-wartość
entity relationship diagram, *Patrz*
 diagram związków encji
Entity-Attribute-Value, *Patrz*
 antywzorzec, encja-atrybut-wartość
ENUM, 155, 157, 159, 160
enumeracja ścieżki, 46, 49, 58
 wady, 48
 wstawianie węzła, 48
ER, *Patrz* diagram związków encji
ETL, 158
execute(), 324

F

fetch(), 324
FILESTREAM, 172
find(), 350

FLOAT, 144, 145, 146
Forster, E. M., 263
Fowler, Martin, 350, 353
FTS, rozszerzenia, 242
full-text search, *Patrz* wyszukiwanie
 pełnotekstowe
functional dependency, *Patrz* zależności
 funkcjonalne
funkcja skrótu, 281
funkcje okien, 315

G

generalized inverted index,
 Patrz uogólniony indeks odwrotny
GENERATOR, 63
generowanie kodu, 261
GIN, *Patrz* uogólniony indeks
 odwrotny
Glass, Robert L., 349
globally unique identifier, *Patrz*
 globalnie unikatowy identyfikator
globalnie unikatowy identyfikator, 316
Gonzalez, Albert, 289, 290
GRANT, 170
GROUP BY, 210, 211, 214
GROUP_CONCAT(), 220
GUID, *Patrz* globalnie unikatowy
 identyfikator

H

Hadoop, 94
hash function, *Patrz* kryptograficzna
 funkcja skrótu
hash(), 286
hasła
 algorytm SHA-1, 282
 algorytm SHA-256, 281
 kryptograficzna funkcja skrótu, 281

MD5(), 282
 przechowywanie, 276, 277
 przechowywanie zabezpieczonych
 kodów haseł, 281
 resetowanie, 286
 sól, 284
 uwierzytelnianie, 278
 wysyłanie pocztą elektroniczną, 279

HBase, 94
 Hibernate, framework, 109, 110
 hierarchia, 59
 Hooper, Grace Murray, 331
 horizontal partitioning sharding,
Patrz partycjonowanie poziome

I

id, kolumna, 64, 65
 IDENTITY, 63
 IEEE 754, standard, 146, 147
 iloczyn kartezjański, 253, 254
 impedance mismatch, *Patrz*
 niezgodność impedancji
 indeks odwrócony, 246
 indeks pełnotekstowy, 238
 indeksy, 178, 179, 182, 191
 błędy, 179
 brak, 179
 pokrywające, 190
 selektywność, 185
 stosowanie bez planu, 179
 zbyt wiele, 181

INSERT, 161, 179, 266
 integralność danych, zarządzanie, 130
 integralność odwołań, 76, 77
 zarządzanie, 133
 intersection table, *Patrz* tabela łącząca
 inverted index, *Patrz* indeks odwrócony
 IS DISTINCT FROM, 205, 206

IS NOT NULL, 205
 IS NULL, 205
 ISO/IEC 11179, standard, 71

J

Jaywalking, *Patrz* antywzorzec,
 przechodzenie na czerwonym świetle
 jeden-do-wielu, relacja, 27
 język definiowania danych, 15
 język przetwarzania danych, 15
 JOIN, 67
 join dependency, *Patrz* zależności
 złączeniowe

K

kaskadowe aktualizacje, 82, 83
 Kernighan, Brian, 143
 Kirk, James T., 127
 klucz główny, 62, 63, 69, 70, 315
 czemu jest ważny, 64
 niejasności, 62
 unikatowość, 132
 warunki, 64
 wybieranie nazwy, 70, 71
 wybór, 63

klucz naturalny, 72, 314, 318
 klucz obcy, 62, 68, 73, 82, 83
 koszty rezygnacji, 77
 niechęć, 76, 79
 wybieranie nazwy, 71
 zalety, 84

klucz zastępczy, *Patrz* pseudoklucz
 klucz złożony, 65, 68, 69, 72
 kod
 generowanie, 261
 system kontroli, 341
 kolumny, klonowanie, 129
 kontrola jakości, 336

konwencja ponad konfiguracją, 70
 korzeń, 38
 kryptograficzna funkcja skrótu, 281

L

LAST_INSERT_ID(), 48, 68
 law of parsimony, *Patrz* prawo prostoty
 leaky abstractions, *Patrz* nieszczelne abstrakcje
 leaves, *Patrz* liście
 Letwin, Gordon, 78
 liczby całkowite, 144, 317
 liczby zmiennoprzecinkowe, 145, 146, 147
 LIKE, 189, 235
 LIMIT, 230
 lista sąsiedztwa, 40, 41, 44, 57, 58
 lista, ograniczenie długości, 35
 liście, 38
 LOAD_FILE(), 174
 log_min_duration_statement, 187
 LONG, 174
 long_query_time, 186
 Lucene, *Patrz* Apache Lucene

M

Marx, Groucho, 289
 MATCH(), 238
 MAX(), 212
 MD5(), 282
 MEDIUMBLOB, 174
 Mencken, H. L., 347
 MENTOR, 185, 186
 mierzenie, 186
 optymalizacja, 191
 przebudowa, 191
 testowanie, 191
 wskazanie, 189
 wyjaśnienie, 188

metadane, 155
 zmiana, 158
 Microsoft SQL Server, wyszukiwanie pełnotekstowe, 240
 mieszanie danych z metadanymi, 106, 129
 migracja, 340, 341
 mistake-proofing, *Patrz* poka-yoke
 Model-View-Controller,
 Patrz model-widok-komponent
 model-widok-komponent, 349, 360
 MongoDB, 95
 MVC, *Patrz* model-widok-komponent
 mysqldump, 169

N

nadmiarowość, 375, 377
 Neumann, John von, 367
 NF, *Patrz* postacie normalne
 nie powtarzaj się, zasada, 357
 nieskończona precyzja, 145
 nieszczelne abstrakcje, 351
 niezgodność impedancji, 13
 node, *Patrz* węzeł
 noleaf nodes, *Patrz* węzły niebędące liśćmi
 normal form, *Patrz* postacie normalne
 normalizacja
 cele, 372
 mity, 371
 reguły, 367, 383
 NOT NULL, 200, 206
 NULL, 196, 197, 201
 w parametrach zapytań, 198
 w wyrażeniach, 197
 w wyrażeniach logicznych, 204
 w wyrażeniach skalarnych, 203
 wyszukiwanie wartości, 205

NULLIF(), 120
NUMERIC, 150, 151
 zalety, 151

O

obiektywny diagram klas, 87
obiektywny model programowania, 86
object-relational mapping, *Patrz*
 odzworowania obiektowo-relacyjne
obrazy, przechowywanie, 166, 172,
 173, 174, 175
obsługa błędów, 327
Ockham, William, 251, 257
odzworowania obiektowo-relacyjne, 267
okien, funkcje, 315
ON, 67
ON DELETE CASCADE, 43
optymalizacja wydajności, 178
Oracle, indeksowanie tekstu, 239
ORDER BY, 369
ORM, *Patrz* odzworowania
 obiektywno-relacyjne
Orwell, George, 61
otwarty schemat, 88
outer join, *Patrz* złączenie zewnętrzne

P

Paine, Thomas, 117
para nazwa-wartość, 88
partycjonowanie
 pionowe, 138
 poziome, 137
PDO::quote(), 303
perspektywy, 337
perspektywy systemowe, 156
pg_dump, 169
pgFouine, 187

piąta postać normalna, 380, 382
pierwsza postać normalna, 373, 374
plan wykonania zapytania, 188
pliki, 167, 168, 169, 170
podzapytania skorelowane, 217
poka-yoke, 81, 82
polimorfizm, 114
Polymorphic Associations,
 Patrz związki polimorficzne
Popper, Karl, 165
postacie normalne, 372
 Boyce'a-Codda, 377, 378
 czwarta postać normalna, 379, 380
 druga postać normalna, 374, 375
 dziedzina-klucz, 382
 hierarchia, 373
 piąta postać normalna, 380, 382
 pierwsza postać normalna, 373, 374
 szósta postać normalna, 383
 trzecia postać normalna, 376, 377
PostgreSQL, wyszukiwanie tekstu, 241
prawo oszczędności, 257, 258
prawo prostoty, 257
precyzja
 nieskończona, 145
 skończona, 145
prepare(), 324
PRIMARY KEY, 125
procedury składowe, 297
projekt bez schematu, 88
projektowanie defensywne, 367
promiscuous association, *Patrz* związek
 mieszany
pseudoklucz, 63, 70, 71, 72, 315, 317,
 319
 standardy, 63
terminologia, 63

Q

QEP, *Patrz* plan wykonania zapytania
query execution plan, *Patrz* plan
wykonania zapytania
quote(), 294

R

race condition, *Patrz* sytuacja wyścigu
RAND(), 225
Ratcliffe, Mitch, 313
RAW, 174
Reagan, Ronald, 37
REAL, 146
Redis, 95
reguła jednej wartości, 211, 213
 naruszenie, 214
relacja, 368
 jeden-do-wielu, 27
 wiele-do-jednego, 27
 wiele-do-wielu, 32, 69
REVOKE, 170
rman, 169
ROLLBACK, 169
root, *Patrz* korzeń
Ross, Blake, 25
ROW_NUMBER(), 230, 315
ROWID, 63
Ruby on Rails, 71, 109, 340, 351
Rumsfeld, Donald, 195

S

salt, *Patrz* sól
SAMPLE, 231
SCOPE_IDENTITY(), 68
seed data, *Patrz* dane początkowe
sekwencje, 68
semistrukturalne dane, 100

separator, wybór, 34
SEQUENCE, 63
SequenceName.CURRVAL(), 68
SERIAL, 63
Serialized LOB, 100
SHA-1, algorytm, 282
SHA-256, algorytm, 281
Single Table Inheritance, 96, 97
single-value rule, *Patrz* reguła jednej
 wartości
skanowanie tabeli, 226
skończona precyzja, 145
skrót, funkcja, 281
Solr, 246
sortowanie, 225, 226, 228
sól, 284
Sphinx Search, 244, 245
Spolsky, Joel, 351
SQL injection, *Patrz* wstrzykiwanie
 SQL-a
SQL Server Profiler, 186
SQL Trace, 186
SQL, język, 12
 automatyczne pisanie kodu, 261
 dopasowywanie do wzorca, 28
 dynamiczny, 290
SQL:2003, standard, 315
SQL-99, standard, 45, 205, 235
SQLite, wyszukiwanie pełnotekstowe,
 242
standard
 ANSI SQL, 180
 IEEE 754, 146, 147
 ISO/IEC 11179, 71
 SQL:2003, 315
 SQL-99, 45, 205, 235
START WITH, 45
STI, *Patrz* Single Table Inheritance
stored procedures, *Patrz* procedury
 składowe

SUM(), 28, 41, 149
surrogate key, *Patrz* pseudoklucz
symbol ucieczki, 294
symbole wieloznaczne, 264, 268, 271
 koszty, 266
symbole zastępcze parametrów, 295
synchronizacja danych, 131
synchronizacja metadanych, 133
sytuacja wyścigu, 68
szósta postać normalna, 383

T

tabela atrybutów, 87
 atrybut, 87
 encja, 87
 wartość, 88
tabela domknięcia, 46, 53, 57, 58
 ilustracja, 54
tabela łącząca, 32, 65, 69
 diagram związków encji, 32
 zalety, 35
tabela pochodna, 217, 218
tabela wiele-do-wielu, *Patrz* tabela łącząca
tabela wirtualna, 242
tabela wyszukiwania, 163
 aktualizowanie wartości, 161
tabele
 klonowanie, 129
 skanowanie, 226
table scan, *Patrz* skanowanie tabeli
TABLESAMPLE, 231
technical debt, *Patrz* dług techniczny
testowanie, 342
 dane początkowe, 344
 izolacja, 342
 klasy odwzorowań
 obiektowo-relacyjnych, 345

ograniczenia, 344
procedury składowane, 344
tabele, kolumny, perspektywy, 343
wyzwalacze, 344
zapytania, 344
testy negatywne, 343
TKProf, 186
Tokyo Cabinet, 95
trzecia postać normalna, 376, 377
typy definiowane przez użytkownika,
 156
Tzu, Sun, 75

U

UDT, *Patrz* typy definiowane przez użytkownika
unikatowość, 132
UNION, 133, 259
uogólniony indeks odwrotny, 242
UPDATE, 179
user-defined types, *Patrz* typy definiowane przez użytkownika
USING, 67
użytkownik
 identyfikacja, 280
 uwierzytelnianie, 280

V

Valéry, Paul, 153

W

wczesne wywoływanie awarii, zasada,
 270
węzeł, 38
węzły niebędące liśćmi, 38
WHERE, 198
wiele-do-jednego, relacja, 27

wiele-do-wielu, relacja, 32, 69
więzy integralności, 76
wildcard, *Patrz* symbole wieloznaczne
WITH, 45, 58
wspólne wyrażenie tablicowe, 45
wstrzykiwanie SQL-a, 290, 291, 292, 293
 zabezpieczenia, 301
wydajność, 178
 optymalizacja, 178
wyrażenia regularne, 235, 236
wyrażenie, 18
wyszukiwanie pełnotekstowe, 238
 Microsoft SQL Server, 240
 SQLite, 242
wzorce, dopasowywanie, 28, 235, 236, 237

Z

zapytania rekurencyjne, 58
zapytania, czas wykonywania, 186
zapytanie, 18
Zawinski, Jamie, 233
zbiory zagnieżdżone, 46, 49, 53, 57, 58
 ilustracja, 50
 wstawianie węzła, 52
 zalety, 51
złączenie, 219
 zewnątrzne, 219
związek mieszany, 105
związki polimorficzne, 105
 definiowanie, 105
 odwracanie odwołań, 110
zapytania, 106

zależności funkcjonalne, 216
zależności złączeniowe, 383
zaokrąglanie, 145
zapytania agregujące, tworzenie, 28, 33

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Ucz się na błędach... Cudzych!

Podobno najlepiej jest uczyć się na cudzych błędach. Ta mądrość sprawdza się w wielu dziedzinach życia, także w informatyce. Ile razy brnąłeś w złe rozwiązanie, aż ktoś powiedział: „Tak się nie robi”? Czy nie pomyślałeś wtedy o książce, w której zamiast metod rozwiązywania danych problemów znalazłbyś informacje, jak do nich nie dopuścić? Chciałbyś, żeby ta książka dotyczyła języka SQL? Oto ona!

Niniejsza publikacja przedstawia zbiór antywzorów w języku SQL. Dzięki niej poznasz błędy najczęściej popełniane przy projektowaniu i wykorzystywaniu baz danych. Dowiesz się z niej, jak nie tworzyć logicznego i fizycznego projektu bazy danych, jak nie zadawać zapytań SQL oraz jak nie wytwarzać aplikacji — a wszystko po to, aby zrozumieć, jak nie popełniać błędów. W każdej z czterech części znajdziesz ogrom interesujących informacji: poznasz zasady przechowywania haseł, błędy pojawiające się w wyniku zaokrągleń czy sposoby radzenia sobie z brakiem integralności bazy danych. Książka ta jest genialną pozycją, dzięki której już nigdy nie zбочysz z drogi w codziennej pracy z bazami danych i językiem SQL!

- **Antywzorce logicznego projektu bazy danych**
 - Modelowanie drzew
 - Tworzenie kluczy głównych
 - Związki polimorficzne
 - Atrybuty wielokolumnowe
- **Antywzorce fizycznego projektu bazy danych**
 - Błędy zaokrągleń
 - Przechowywanie dużych plików
 - Indeksy
- **Antywzorce zapytań**
 - Wykorzystanie NULL-a
 - Grupowanie kolumn
 - Losowe wybieranie wiersza
 - Przeszukiwanie tekstów
 - Optymalizacja zapytań SQL
- **Antywzorce wytwarzania aplikacji**
 - Przechowywanie haseł
 - Wstrzykiwanie SQL



helion.pl
księgarnia internetowa

Nr katalogowy: 7 5 1 4

Księgarnia internetowa
<http://helion.pl>

Zamówienia telefoniczne:
0 801 339900
0 601 339900



Helion

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

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

Cena: 69,00 zł

ISBN 978-83-246-3482-8



9 788324 634828

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

Informatyka w najlepszym wydaniu