

Zacznij bezboleśnie tworzyć bazy danych w języku SQL

Rusz głową!

SQL



Pomóż Grześkowi poprawić relacje ze swymi danymi



Przestań w końcu mylić klucz główny z kluczem obcym



Raz na zawsze naucz się, co jest normalne



Wczytaj ważne pojęcia dotyczące zapytań SQL prosto do swego mózgu



Uchronią przed wstydliwą koniecznością modyfikacji swoich tabel



Sprawdź swoją znajomość SQL, wykonując dziesiątki ćwiczeń

O'REILLY®

Lynn Beighley

Helion

Autorka książki SQL. Rusz głową!



Lynn Beighley

Lynn chciała pisać fantasykę, lecz utknęła w cieles autorki książek na tematy techniczne. Kiedy odkryła, że honoraria za książki techniczne wystarczają na płacenie rachunków, nauczyła się akceptować ten stan rzeczy i czerpać z niego radość.

Po powrocie do szkoły, gdzie chciała zdobyć tytuł magistra informatyki, Lynn pracowała dla firm, których nazwy określane są tajemniczo brzmiącymi akronimami NRL oraz LANL.

Jednak nadszedł czas, gdy Lynn odkryła Flasha i napisała swój pierwszy bestseller.

Lynn miała niewłaściwe wyczucie czasu i przeniosła się do Doliny Krzemowej tuż przed wielkim krachem. Spędziła kilka lat, pracując w firmie Yahoo!, pisząc książki i kursy treningowe. W końcu jednak posłuchała swego twórczego, pisarskiego zacięcia i przeniosła się do Nowego Jorku, gdzie obroniła tytuł magistra sztuk pięknych na kierunku kompozycji literackiej.

Jej praca magisterska w stylu książek Rusz głową! została przedstawiona pełnemu audytorium profesorów i studentów. Odebrano ją bardzo dobrze, a Lynn obroniła tytuł, dokończyła książkę SQL. Rusz głową! i wprost nie może się doczekać swego następnego projektu literackiego.

Lynn kocha podróże, gotowanie i tworzenie rozbudowanych opowieści o całkowicie nieznanych osobach. I trochę boi się klaunów.



Widok z okna Lynn

*) Gra słów, w której „SQRL” to skrót od angielskiego słowa „squirrel” oznaczającego wiewiórkę – przyp. tłum.

Spis treści (skrócony)

Wprowadzenie	25
1. Dane i tabele: <i>Na wszystko znajdzie się odpowiednie miejsce</i>	37
2. Polecenie SELECT: <i>Pobieranie podarowanych danych</i>	87
3. DELETE i UPDATE: <i>Są szanse, że wszystko będzie w porządku</i>	153
4. Projektowanie dobrych tabel: <i>Po co być normalnym?</i>	193
5. Polecenie ALTER: <i>Korygowanie przeszłości</i>	231
6. Zaawansowane zastosowanie polecenia SELECT: <i>Nowy sposób spojrzenia na dane</i>	267
7. Projektowanie baz danych składających się z wielu tabel: <i>Wyrastamy z naszych starych tabel</i>	311
8. Złączenia i operacje na wielu tabelach: <i>Czy nie możemy się wszyscy dogadać?</i>	373
9. Podzapytania: <i>Zapytania w zapytaniach</i>	409
10. Złączenia zewnętrzne, złączenia zwrotne oraz unie: <i>Nowe manewry</i>	447
11. Ograniczenia, widoki i transakcje: <i>Zbyt wielu kucharzy psuje bazę danych</i>	483
12. Bezpieczeństwo: <i>Zabezpieczanie swych dóbr</i>	521
A Pozostałości: <i>Dziesięć najważniejszych zagadnień (których nie opisaliśmy wcześniej)</i>	551
B Instalacja MySQL-a: <i>Spróbuj to zrobić sam</i>	569
C Przypomnienie narzędzi: <i>Wszystkie nowe narzędzia SQL</i>	575
Skorowidz	583

Spis treści (z prawdziwego zdarzenia)



Wprowadzenie

Twój mózg myśli o SQL-u. Czytając książkę, Ty starasz się czegoś nauczyć, natomiast Twój mózg wyświadcza Ci przysługę, dbając o to, by te informacje nie zostały zbyt długo w Twej głowie. Twój mózg myśli sobie: „Lepiej zostawić miejsce na jakieś ważne rzeczy, takie jak: których dzikich zwierząt należy unikać albo czy jeźdzenie nago na snowboardzie jest dobrym pomysłem, czy nie”. Zatem w jaki sposób możesz przekonać swój mózg, by uznał, że poznanie SQL-a to dla Ciebie kwestia życia lub śmierci?

Dla kogo jest ta książka?	26
Wiemy, co sobie myślisz	27
Metapożnanie: myślenie o myśleniu	29
Oto co możesz zrobić, aby zmusić swój mózg do posłuszeństwa	31
Przeczytaj to	32
Nasi wspaniali recenzenci	34
Podziękowania	35

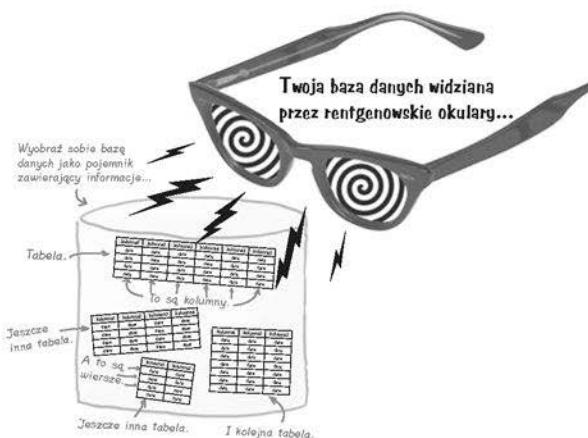
Dane i tabele

1

Na wszystko znajdzie się odpowiednie miejsce

Czy także i Ty nie cierpisz gubienia czegokolwiek? Niezależnie od tego, czy są to kluczyki do samochodu, bon uprawniający do zakupu mieszkania z 25-procentową zniżką, dane używanej aplikacji, nie ma nic gorszego niż **niemożność sprostania własnym potrzebom...** wtedy gdy tego najbardziej potrzebujemy. A jeśli chodzi o używane aplikacje, to trzeba wiedzieć, że nie ma lepszego miejsca na przechowywanie ważnych informacji niż **tabele**. A zatem przewróć kartkę i zacznij, krok za krokiem, poznawać świat **relacyjnych baz danych**.

Definiowanie danych	38
Przeanalizuj swoje dane pod względem kategorii	43
Co znajduje się w bazie danych?	44
Twoja baza danych widziana przez rentgenowskie okulary...	46
Bazy danych zawierają powiązane ze sobą informacje	48
Tabele w zbliżeniu	49
Przejmij kontrolę!	53
Tworzenie tabeli: Polecenie CREATE TABLE	55
Tworzenie bardziej złożonych tabel	56
Przekonajmy się, jak łatwo można pisać kod SQL	57
Utwórzmy w końcu tabelę moje_kontakty	58
Twoja tabela jest gotowa	59
Spotkajmy się z niektórymi typami danych SQL	60
Twoja tabela bez tajemnic	64
Nie można ponownie stworzyć już istniejącej bazy danych lub tabeli!	66
Do kosza ze starą tabelą — czas na nową	68
Aby dodać dane do tabeli, będziesz musiał skorzystać z polecenia INSERT	70
Utworzenie polecenia INSERT	73
Wariacje na temat polecenia INSERT	77
Kolumny bez wartości	78
Zerknij na swoją tabelę, używając polecenia SELECT	79
SQL bez tajemnic: Wyznania wartości NULL	80
Kontrola wewnętrznych wartości NULL	81
NOT NULL pojawia się w wynikach polecenia DESC	83
Wypełnij pustkę słowem kluczowym DEFAULT	84
Przybornik SQL	86



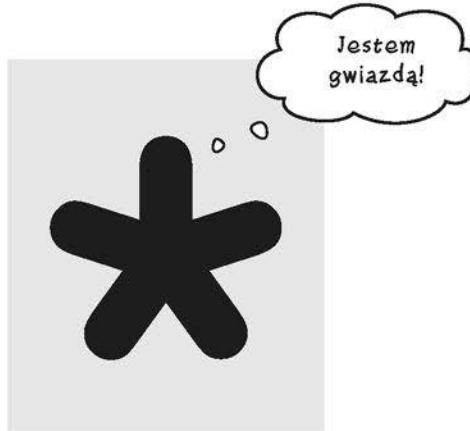
Polecenie SELECT

2

Pobieranie podarowanych danych

Czy naprawdę lepiej jest dawać, niż brać? W przypadku korzystania z baz danych najprawdopodobniej częściej będziesz musiał pobierać z nich dane, niż je zapisywać. I właśnie w tych wszystkich sytuacjach przydadzą Ci się informacje przedstawione w tym rozdziale: poznasz w nim bardzo przydatne polecenie **SELECT** i dowiesz się, jak uzyskać dostęp do tych wszystkich ważnych informacji, które wcześniej zapisywałeś w swoich tabelach. Co więcej, dowiesz się także, jak stosować klauzulę **WHERE** i operatory **AND** i **OR**, by nie tylko pobierać dane, lecz wyświetlać te, które są Ci potrzebne.

Pobierać dane czy nie pobierać?	88
Lepsza postać polecenia SELECT	91
Co oznacza gwiazdka (*)	92
Jak poszukiwać wartości różnych typów?	98
Kolejne problemy ze znakami przestankowymi	99
Niedopasowane apostrofy	100
Polecenia INSERT z danymi zawierającymi apostrofy	102
Pobieranie konkretnych kolumn w celu ograniczenia wyników	107
Okręsianie kolumn w celu zwiększenia szybkości zapytania	107
Łączenie zapytań	114
Odnajdywanie wartości liczbowych	117
Lagodne operatory porównania	120
Odnajdywanie wartości liczbowych przy użyciu operatorów porównania	122
Odnajdywanie danych tekstowych przy użyciu operatorów porównania	125
Być ALBO nie być	127
Różnica pomiędzy operatorem AND a OR	130
By odszukać NULL, użyj operatora IS NULL	133
Jak można zaoszczędzić czas dzięki jednemu słowu: LIKE	135
Zew wieloznaczności	135
Pobieranie zakresów przy użyciu operatora AND i operatorów porównania	139
Lepszy sposób — operator BETWEEN	140
Operator IN — w kręgu zainteresowania...	143
...lub poza nim — NOT IN	144
Więcej o operatorze NOT	145
Przybornik SQL	150



DELETE i UPDATE

3

Są szanse, że wszystko będzie w porządku

Cały czas zmieniasz zdanie? Teraz nie przysporzy Ci to najmniejszego problemu! Dzięki poleceniom **DELETE** i **UPDATE**, które poznasz w tym rozdziale, nie będziesz już dłużej musiał ponosić konsekwencji decyzji podjętych pół roku temu, kiedy to zapisałeś w bazie dane o spodniach w kształcie dzwonów, które właśnie z powrotem zaczynały być modne. Dzięki poleceniu **UPDATE** będziesz mógł zmieniać dane, natomiast polecenie **DELETE** pozwoli usiąć z bazy dane, które nie będą Ci już dłużej potrzebne. Jednak w tym rozdziale nie tylko pokażemy Ci te dwa nowe polecenia SQL, lecz także nauczymy, jak można używać ich w precyzyjny sposób, by przez przypadek nie usunąć danych, które cały czas są potrzebne.

Klowni są przerażający	154
Śledzenie kłownów	155
Klowni są w ciągłym ruchu	156
Jak są zapisywane informacje o kłownach?	160
Gonzo, mamy problem	162
Jak pozbyć się rekordu — polecenie DELETE	163
Stosowanie naszego nowego polecenia DELETE	165
Reguły polecenia DELETE	166
Dwa kroki — INSERT i DELETE	169
Stosuj polecenie DELETE rozważnie	174
Problemy z nieprecyzyjnymi poleceniami DELETE	178
Modyfikowanie danych przy użyciu polecenia UPDATE	180
Reguły stosowania polecenia UPDATE	181
UPDATE odpowiada kombinacji INSERT-DELETE	182
Polecenie UPDATE w akcji	183
Aktualizacja miejsc wystąpień kłownów	186
Zaktualizuj ceny drinków	188
A cheemy tylko jednego polecenia UPDATE	190
Przybornik SQL	192



Projektowanie dobrych tabel

Po co być normalnym?

4

Dotychczas tworzyłeś tabele bez zwracania na nie szczególnej uwagi.

I wszystko było w porządku, tabele działały bez problemów. Mogłeś w nich zapisywać, modyfikować, usuwać i pobierać dane. Jednak wraz ze zwiększeniem się ilości danych w tabelach zaczynasz zauważać, że są rzeczy, które mogłeś zrobić wcześniej, by ułatwić sobie w przyszłości tworzenie klauzul WHERE. Innymi słowy, musisz **znormalizować** swoje tabele.

Dwie wędkarskie tabele	194
Tabele dotyczą związków	198
Dane atomowe	202
Dane atomowe a Twoje tabele	204
Reguły danych atomowych	205
Dlaczego warto być normalnym?	208
Zalety normalizacji tabel	209
Klowni nie są normalni	210
W połowie drogi do 1NF	211
Reguły KLUCZA GŁÓWNEGO	212
Dążenie do pierwszej postaci NORMALNEJ	215
Poprawianie tabeli Grześka	216
Oryginalna postać polecenia CREATE TABLE	217
Pokaźcie mi moją tabelę kase	218
Polecenie oszczędzające czas	219
Tworzenie tabeli z KLUCZEM GŁÓWNYM	220
1, 2, 3... automatycznie inkrementowane	222
Dodawanie KLUCZA GŁÓWNEGO do istniejącej tabeli	226
Modyfikacja tabeli i dodanie KLUCZA GŁÓWNEGO	227
Przybornik SQL	228

Chwileczkę, ale ja już mam tabelę z całą masą informacji. Chyba nie jesteście poważni, jeśli oczekujecie, że usunę tabelę jak w pierwszym rozdziale i będę ponownie wpisywać całą jej zawartość tylko i wyłącznie po to, żeby stworzyć kolumnę klucza głównego.



Polecenie ALTER

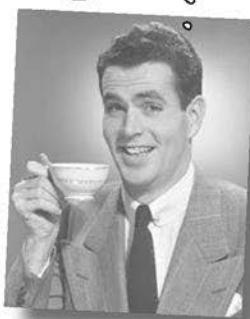
5

Korygowanie przeszłości

Czy kiedykolwiek marzyłeś, by mieć możliwość naprawiania błędów popełnionych w przeszłości? Cóż, teraz masz taką możliwość. Dzięki poleceniu ALTER możesz zastosować wszystkie nowe wiadomości i doświadczenia, by zmodyfikować strukturę tabel utworzonych kilka dni, miesięcy lub nawet lat wcześniej. Ale to jeszcze nie wszystko — możesz to zrobić bez wprowadzania jakichkolwiek zmian w istniejących danych. Skoro tu dotarłeś, zapewne już wiesz, co naprawdę oznacza **normalizacja**, i będziesz mógł zastosować tę wiedzę do zapewnienia odpowiedniej struktury swym tabelom; tym starym i nowym.

Odpicuj moją tabelę

Nadszedł czas,
by zmienić Twoją wypróbowaną,
lecz przestarzałą tabelę na cudeńko,
jakim u nie oprą się żadne informacje,
i przenieść ją na taki poziom
bazodanowego odpicowania, o którym
nawet Ci się nie śniło.



Musimy wprowadzić kilka zmian	232
Modyfikowanie tabel	237
Ekstremalne metamorfozy tabel	238
Zmiana nazwy tabeli	239
Musimy poczynić pewne plany	241
Przebrajanie kolumn	242
Zmiany strukturalne	243
Polecenia ALTER i CHANGE	244
Zmiana dwóch kolumn przy użyciu jednego polecenia SQL	245
Szybko! Usuń tę kolumnę	249
Dokładniejsza analiza nieatomowej kolumny lokalizacji	256
Poszukaj wzorca	257
Kilka wygodnych funkcji łańcuchowych	258
Użyj bieżącej kolumny do zapisania wartości w innej kolumnie	263
Sposób działania połączonych poleceń UPDATE i SET	264
Przybornik SQL	266

Zaawansowane zastosowanie polecenia SELECT

Nowy sposób spojrzenia na dane

6

Nadszedł czas, byś dodał do swojego SQL-owego przybornika nieco finezji.

Już wiesz, jak pobierać dane przy użyciu polecenia SELECT i klauzuli WHERE. Jednak czasami będziesz potrzebował nieco większej precyzji niż ta, jaką one zapewniają. W tym rozdziale dowiesz się o określaniu kolejności oraz grupowaniu danych, jak również o wykonywaniu operacji matematycznych na wynikach zapytań.

Reorganizacja Filmoteki Bazodanowa	268
Kilka problemów z bieżącą tabelą	269
Dopasowywanie istniejących danych	270
Okręslenie zawartości nowej kolumny	271
Polecenie UPDATE z wyrażeniem CASE	274
Wygląda na to, że mamy problem	276
Do tabel może się wkrąść bałagan	281
Potrzebujemy możliwości organizowania danych zwracanych przez polecenie SELECT	282
Wypróbuj klauzulę ORDER BY	285
Sortowanie według jednej kolumny	286
Klauzula ORDER z dwoma kolumnami	289
Klauzula ORDER operująca na wielu kolumnach	290
Uporządkowana tabela filmów	291
Zmiana kolejności dzięki użyciu DESC	293
Problem najlepszej sprzedawczyni grupy Młode Gospodarstwa	295
Funkcja SUM zsumuje wszystko za nas	297
Zsumuj wszystko za jednym razem dzięki użyciu GROUP BY	298
Funkcja AVG z klauzulą GROUP BY	299
Funkcje MIN i MAX	300
Liczymy dni	301
Pobieranie unikalnych wartości	303
LIMIT-owianie ilości wyników	306
Ograniczenie tylko do drugiego miejsca	307
Przybornik SQL	310

FILMOTeka BAZODANOWA



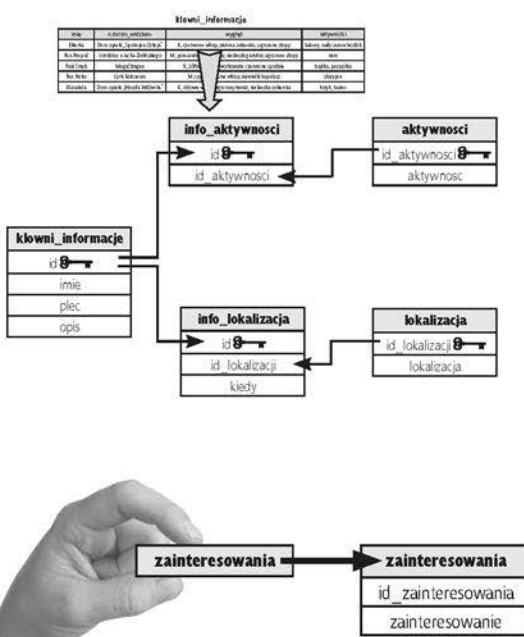
Projektowanie baz danych składających się z wielu tabel

7

Wyrastamy z naszych starych tabel

Kiedyś może nadjść moment, gdy pojedyncza tabela przestanie Ci wystarczać.

Twoje dane stały się bardziej złożone i jedna tabela, której do tej pory używaleś, nie jest już w stanie sprostać Twoim potrzebom. W tabeli jest bardzo dużo powtarzających się, nadmiarowych informacji, które niepotrzebnie zajmują miejsce na dysku i spowalniają zapytania. Wytrwałeś, używając tej jednej tabeli, tak długo, jak to było możliwe. Jednak świat, w którym żyjemy, jest ogromny i czasami potrzeba będzie **więcej niż jednej tabeli**, by zapisać wszystkie dane, zapanować nad nimi i w ostatecznym rozrachunku cały czas być panem własnej bazy danych.



Szukamy partnerki na randkę dla Wieśka	312
Wszystko stracone... ale zaraz	323
Wyobraź sobie dodatkową tabelę	324
Nowe tabele w bazie danych z informacjami o kłownach	325
Schemat bazy danych klowni_informacje	326
W jaki sposób z jednej tabeli zrobić dwie	328
Łączenie tabel	330
Ograniczanie klucza obcego	335
Dlaczego należy zwracać sobie głowę kluczami obcymi?	336
TWORZENIE tabeli z KLUCZEM OBCYM	337
Zależności pomiędzy tabelami	339
Wzorce danych: jeden-do-jednego	339
Wzorce danych: kiedy używać tabel połączonych zależnością jeden-do-jednego	340
Wzorce danych: jeden-do-wielu	341
Wzorce danych: dochodzimy do zależności wiele-do-wielu	342
Wzorce danych: potrzebujemy tabeli łączącej	345
Wzorce danych: wiele-do-wielu	346
W końcu w 1NF	351
Klucz złożony korzysta z wielu kolumn	352
Zapis uproszczony	354
Częściowa zależność funkcjonalna	355
Przechodnia zależność funkcjonalna	356
Druga postać normalna	360
Trzecia postać normalna (w końcu)	366
I tak oto Robert i lista_grzesia żyli od tej pory długo i szczęśliwie	369
Przybornik SQL	372



Złączenia i operacje na wielu tabelach

Czy nie możemy się wszyscy dogadać?

8

Witam w skomplikowanym świecie wielu tabel. Wspaniale jest mieć w bazie więcej niż jedną tabelę, jednak abyś mógł się nimi posługiwać, będziesz musiał poznać *nowe narzędzia i techniki*. Jednoczesne korzystanie z wielu tabel może być kłopotliwe, dlatego też, by wszystko było jasne, będziesz potrzebował nazw zastępczych (nazywanych także „aliasami”). Oprócz tego **złączenia** pomogą Ci połączyć tabele, dzięki czemu będziesz mógł korzystać ze wszystkich danych, zapisanych w wielu różnych miejscach. Przygotuj się — teraz ponownie **odzyskasz pełną kontrolę nad swoją bazą danych**.

Powtarzamy się, cały czas się powtarzamy...	374
Wypełnianie tabel	375
Zainteresowania — kłopotliwa kolumna	378
WCiąż wykazujemy zainteresowanie	379
Aktualizacja wszystkich zainteresowań	380
Pobieranie wszystkich zainteresowań	381
Wiele dróg prowadzących w to samo miejsce	382
CREATE, SELECT oraz INSERT — (prawie) jednocześnie	382
CREATE, SELECT i INSERT — jednocześnie	383
O co chodzi z tym AS?	384
Nazwy zastępcze kolumn	385
Nazwy zastępcze, a kto by ich potrzebował?	386
Wszystko co chciałbyś wiedzieć o złączeniach wewnętrznych	387
Złączenie kartezjańskie	388
Zrozumienie złączeń wewnętrznych	393
Złączenie wewnętrzne w akcji: złączenie równościowe	394
Złączenie wewnętrzne w akcji: złączenie różnorodnościowe	397
Ostatni rodzaj złączeń wewnętrznych: złączenia naturalne	398
Złączone zapytania?	405
Nazwy zastępcze tabel i kolumn bez tajemnic: Dlaczego się ukrywacie?	406
Przybornik SQL	408

...i tak naprawdę
to właśnie stąd się
biorą malutkie tablice
z wynikami.



Podzapytania

Zapytania w zapytaniach

9

Janie, dwuczęściowe zapytanie, proszę. Złączenia są wspaniałe, jednak czasami musisz zadać swojej bazie danych więcej niż jedno zapytanie. Bądź użyć wyników jednego zapytania jako danych wejściowych dla drugiego. I właśnie w takich sytuacjach przydają się podzapytania. Pozwolą Ci one uniknąć powielania danych, sprawią, że Twoje zapytania staną się bardziej dynamiczne, a może nawet pozwolą się wkręcać na koncerty i przyjęcia dla wyższych sfer. (No, na to ostatnie bym nie liczyła, ale dwa punkty z trzech to i tak nieźle).

Grzesiek wchodzi na rynek pracy	410
Baza Grześka wzbogacona o nowe tabele	411
Grzesiek używa złączenia wewnętrznego	412
Ale Grzesiek chce użyć innych zapytań	414
Podzapytania	416
Łączymy dwa zapytania w zapytanie z podzapytaniem	417
Jakby jedno pytanie nie wystarczało: poznajcie podzapytanie	418
Podzapytanie w działaniu	419
Podzapytania i ich reguły	421
Podstawowe informacje o tworzeniu podzapytań	424
Podzapytanie jako kolumna polecenia SELECT	427
Inny przykład: Podzapytanie ze złączeniem naturalnym	428
Podzapytania nieskorelowane	429
SQL bez tajemnic: Wybór optymalnego sposobu realizacji zapytania, jeśli dostępna jest większa liczba możliwości	430
Nieskorelowane podzapytania zwracające wiele wartości: IN oraz NOT IN	433
Podzapytania skorelowane	438
(Przydatne) Podzapytanie skorelowane używające operatora NOT EXISTS	439
EXISTS i NOT EXISTS	440
Uslugi pośrednictwa pracy Grześka — zaproszenie do współpracy!	442
W drodze na imprezę...	443
Przybornik SQL	444

**zapytanie ZEWNĘTRZNE
zapytanie WEWNĘTRZNE**

Zapytanie zewnętrzne.

```
SELECT jakas_kolumna, inna_kolumna
FROM tabela
WHERE kolumna = (SELECT kolumna FROM tabela);
```

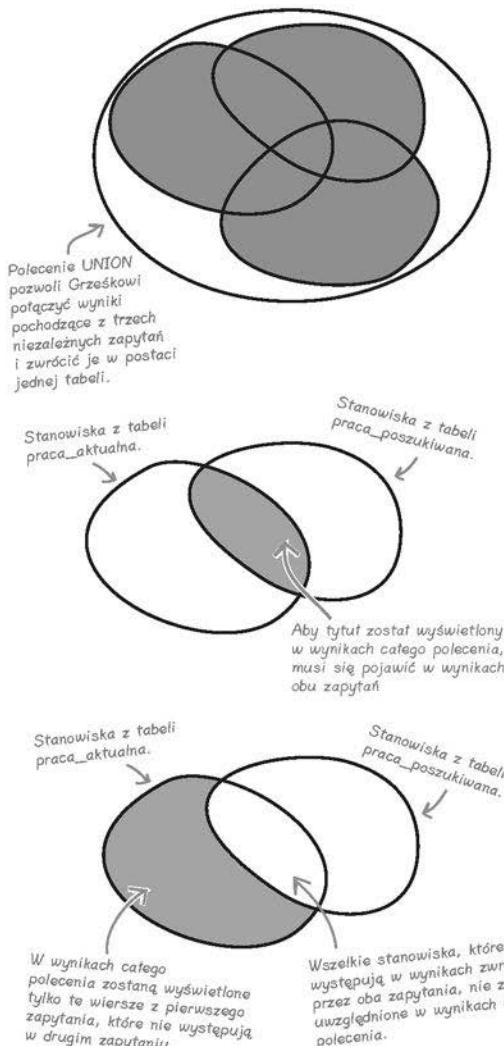
Zapytanie wewnętrzne.

Złączenia zewnętrzne, złączenia zwrotne oraz unie

10

Nowe manewry

To wszystko, czego do tej pory dowiedziałeś się o złączeniach, to jedynie pół prawdy na ich temat. Widziałeś już złączenia krzyżowe tworzące wszystkie możliwe pary rekordów pochodzących z dwóch tabel oraz złączenia wewnętrzne zwieracjące jedynie pasujące do siebie rekordy obu tabel. Nie spotkałeś się jeszcze natomiast ze **złączaniem zewnętrznymi**, zwieraczącymi także te wiersze, które *nie mają pasujących odpowiedników w drugiej tabeli*, **złączaniem zwrotnym**, które łączą tabelę z nią samą, oraz **uniami**, które scalają wyniki z kilku różnych zapytań. Kiedy poznasz te wszystkie sztuczki, będziesz w stanie pobierać dane z bazy dokładnie w taki sposób, jaki będzie Ci potrzebny. (Nie zapomnijmy także ujawnić całej szokującej prawdy o podzapytaniach!).



Porządkи w starych danych	448
Kluczem są dwie strony złączenia — lewa i prawa	449
Oto lewostronne złączenie zewnętrzne	450
Złączenia zewnętrzne i wielokrotne dopasowania	455
Prawostronne złączenie zewnętrzne	456
Podczas gdy my radośnie złączaliśmy zewnętrznie...	459
Moglibyśmy utworzyć nową tabelę	460
Gdzie w schemacie umieścimy nową tabelę?	461
Klucz obcy odwołujący się do tej samej tabeli	462
Łączenie tabeli z nią samą	463
Potrzebujemy złączenia zwrotnego	465
Inny sposób zwieracania informacji z wielu tabel	466
Można zastosować polecenie UNION	467
Polecenie UNION ma swoje ograniczenia	468
Reguły stosowania poleceń UNION w działaniu UNION ALL	469
Utworzenie tabeli na podstawie wyników polecenia UNION	470
Polecenia INTERSECT i EXCEPT	471
Skończyliśmy ze złączaniami, czas zajść się czymś nowym	472
Podzapytania i złączenia — studium porównawcze	473
Zamiana podzapytania na złączenie	474
Złączenie zwrotne jako podzapytanie	479
Firma Grześka rozwija się	480
Przybornik SQL	482

Ograniczenia, widoki i transakcje

11

Zbyt wielu kucharzy psuje bazę danych**Twoje bazy danych rozrosły się i muszą z nich korzystać także inne osoby.**

Problem polega na tym, że niektóre z nich nie będą znały języka SQL równie dobrze jak Ty. Musisz zatem mieć jakieś sposoby, by uniemożliwić im wprowadzanie nieprawidłowych danych, techniki pozwalające ukryć przed nimi wybrane dane oraz mechanizmy, które uchronią te osoby przed wzajemnym przeszkadzaniem sobie, gdy jednocześnie będą modyfikować zawartość bazy.

W tym rozdziale zaczniemy ochroniać nasze dane przed błędami popełnianymi przez inne osoby.

Witamy w Bazie Obronnej, w części 1.

Grzesiek zatrudnił pomoceńników	484
Pierwszy dzień Kuby: Dopisywanie nowego klienta	485
Kuba unika wartości NULL	486
Trzy miesiące później	487
Uwaga, KONTROLA: dodawanie OGRANICZEŃ SPRAWDZAJĄCYCH	488
Sprawdzanie płci	489
Praca Franka staje się nużąca	491
Tworzenie widoku	493
Oglądanie własnych widoków	494
Jak właściwie działa widok?	495
Czym są widoki	496
Wstawianie, aktualizacja i usuwanie danych przy wykorzystaniu widoków	499
Sekret polega na tym, by udawać, że widok jest prawdziwą tabelą	500
Widoki z klauzulą CHECK OPTION	503
Twój widok może pozwalać na aktualizację danych, jeśli...	504
Kiedy widok przestanie być potrzebny	505
Kiedy dobrej bazie przydarzy się coś złego	506
Co się stało w bankomacie	507
Kolejne kłopoty z bankomatami	508
To nie marzenia, to transakcje	510
Klasyczny test ACID	511
SQL pomaga nam zarządzać swoimi transakcjami	512
Co powinno się stać w bankomacie	513
Jak umożliwić korzystanie z transakcji w MySQL-u	514
Wypróbuj transakcje samodzielnie	515
Przybornik SQL	518



12

Bezpieczeństwo

Zabezpieczanie swych dóbr

Włożyłeś niezwykle dużo wysiłku i czasu w utworzenie swojej bazy danych.

I na pewno byłbyś zdruzgotany, gdyby przydarzyło się jej coś złego. Jednak musisz zapewnić **dostęp do swoich danych** innym osobom i obawiasz się, że mogłyby zapisać w bazie niewłaściwe dane lub, co gorsza, **usunąć nie te dane, które powinny**. Na szczęście w tym rozdziale dowiesz się, w jaki sposób można dodatkowo **zabezpieczyć** bazę danych oraz obiekty w niej umieszczone oraz w jaki sposób uzyskać pełną kontrolę nad tym, **kto i jakie operacje na bazie może wykonywać**.

Problemy użytkowników	522
Zapobieganie błędom w bazie kloonów	523
Zabezpieczanie konta administratora	525
Dodanie nowego użytkownika	526
Dokładnie określ, czego poszczególni użytkownicy potrzebują	527
Prosta postać polecenia GRANT	528
Różne wersje polecenia GRANT	531
Usuwanie uprawnień	532
Usuwanie uprawnień przydzielonych dzięki GRANT OPTION	533
Precyzyjne usuwanie	534
Współużytkowane konta przysparzają problemów	538
Stosowanie roli	540
Usuwanie roli	540
Stosowanie roli z klauzulą WITH ADMIN OPTION	542
Łączenie poleceń CREATE USER i GRANT	547
Lista Grześka stała się usługą globalną!	548
Czy już coś wiesz o filii Listy Grzesia w swoim mieście?	549
Korzystaj z SQL-a w swoich własnych projektach, ponieważ także Ty możesz odnieść taki sukces jak Grzesiek	549
Przybornik SQL	550



Root



Ważniak



Doktorek



Głupek



Pyskacz



Szczęściarz



Śpioch



Ziewacz

Pozostałości

A

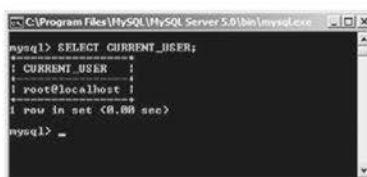
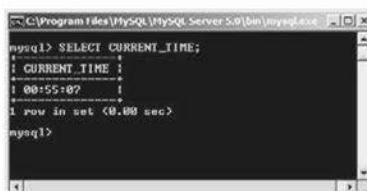
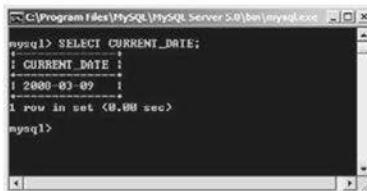
Dziesięć najważniejszych zagadnień (których nie opisaliśmy wcześniej)

Nawet po tym wszystkim jest jeszcze coś więcej. Jest jeszcze dosłownie kilka rzeczy, o których, jak sądzimy, powinieneś wiedzieć. Nie czulibyśmy się w porządku, gdybyśmy je całkowicie zignorowali i nie poświęcili im choćby krótkiej wzmianki. A zatem: nim będziesz mógł odłożyć tę książkę na półkę, przeczytaj o tych drobnych, lecz ważnych sprawach.

Poza tym kiedy skończysz lekturę tego dodatku, pozostaną Ci jeszcze dwa następne... oraz indeks... no i może jeszcze jakieś reklamy... a potem już naprawdę skończysz. Obiecamy!

Nr 1.	Znajdź i zainstaluj graficzny program do obsługi używanego systemu zarządzania bazami danych	552
Nr 2.	Słowa zastrzeżone i znaki specjalne	554
Nr 3.	ALL, ANY oraz SOME	556
Nr 4.	Dodatkowe informacje o typach danych	558
Nr 5.	Tabele tymczasowe	560
Nr 6.	Rzutowanie typów	561
Nr 7.	Kim jesteś? Która jest godzina?	562
Nr 8.	Przydatne funkcje matematyczne	563
Nr 9.	Indeksowanie dla poprawy szybkości działania zapytań	565
Nr 10.	Dwuminutowy kurs PHP i MySQL-a	566

A	ABSOLUTE ACTION ADD ADMIN AFTER AGGREGATE ALIAS ALL ALLOCATE ALTER ANY ARE ARRAY AS ASSERT ATTENTION AT AUTHORIZATION
B	BEFORE BEGIN BINARY BIT BLOB BOTH BREADTH BY
C	CALL CASCADE CASCaded CASE CAST CATALOG CHAR CHARACTER CHECK CLASS CLOB CLOSE COLLATE COLUMN COMMIT COMPLETION CONNECT CONNECTION CONSTRAINT CONSTRAINTS CONSTRUCTOR CURRENT CORRESPONDING CREATE CROSS CUBE CURRENT CURRENT_DATE CURRENT_PATH CURRENT_ROLE CURRENT_TIMESTAMP CURRENT_TIMESTAMP CURRENT_USER CURSOR CYCLE
D	DATA DATE DAY DEADLOCATE DEC DECIMAL DECLARE DEFERRABLE DEFERRED DELETE DEPTH DEREf DESC DESCRIBE REPOSITORY DESTROY DESTRUCTOR DETERMINISTIC DICTIONARY DIAGNOSTICS DISCONNECT DISTINCT DOMAIN DOUBLE DROP DYNAMIC
E	EACH ELSE END END EXEC EQUALS ESCAPE EVERY EXCEPT EXCEPTION EXEC EXECUTE EXTERNAL FALSE FETCH FIRST FLOAT FOR FOREIGN FOUND FROM FREE FULL FUNCTION
F	GENERAL GET GLOBAL GO GOTO GRANT GROUP GROUPING
G	HAVING MOST HOUR
I	IDENTITY IGNORE IMMEDIATE IN INDICATOR INITIALIZE INITIALLY INNER INPUT INSERT INT INTEGER INTERSECT INTERVAL INTO IS ISOLATION ITERATE
J	JOIN
K	KEY
L	LANGUAGE LARGE LAST LATERAL LEADING LEFT LESS LEVEL LIKE LIMIT LOCAL LOCALTIME LOCALTIMESTAMP LOCATOR
M	MAP MATCH MINUTE MODIFIES MODIFY MODULE MONTH
N	NAMES NATIONAL NATURAL NCHAR INCLDB NEW NEXT NO NONE NOT NULL NUMERIC
O	OBJECT OF OFF OLD ON ONLY OPEN OPERATION OPTION OR ORDER ORDINALITY OUT OUTER OUTPUT
P	PAD PARAMETER PARAMETERS PARTIAL PATH POSTFIX PRECISION PREFIX PREORDER PREPARE PRESERVE PRIMARY PRIOR PRIVILEGES PROCEDURE PUBLIC
Q	
R	READ READS REAL RECURSIVE REF REFERENCES REFERENCING RELATIVE RESTRICT RESULT RETURN RETURNS REVOKe RIGHT ROLE ROLLBACK ROLLUP ROUTINE ROW ROWS
S	SAVEPOINT SCHEMA SCROLL SCOPE SEARCH SECOND SECTION SELECT SEQUENCE SESSION SESSION_USER SET SETS SIZE SMALLINT SOME SPACE SPECIFICTYPE SQL SQLEXCEPTION SQLSTATE SQLWARNING START STATE STATEMENT STATIC STRUCTURE SYSTEM USER
T	TABLE TEMPORARY TERMINATE THAN THEN TIME TIMESTAMP TIMEZONE HOUR TIMEZONE MINUTE TO TRAILING TRANSACTION TREAT TRIGGER TRUE
U	UNDER UNION UNIQUE UNKNOWN UNNEST UPDATE USAGE USER USING
V	VALUE VALUES VARCHAR VARIABLE VARYING VIEW
W	WHEN WHENEVER WHERE WITH WITHOUT WORK WRITE
X	
Y	YEAR
Z	ZONE



Instalacja MySQL-a

Spróbuj to zrobić sam

Cała zdobyta przez Ciebie wiedza i umiejętności nie na wiele się zdadzą, jeśli nie wykorzystasz ich w praktyce. W tym dodatku znajdziesz instrukcje dotyczące instalacji swojego własnego serwera MySQL, którego będziesz mógł używać do zabawy i pracy.



Zacznij działać i to szybko!	570
Instrukcje i rozwiązywanie problemów	570
Proces instalacji MySQL-a w systemie Windows	571

Przypomnienie narzędzi

Wszystkie nowe narzędzia SQL

W tym dodatku, w jednym jedynym miejscu, zostały po raz pierwszy zebrane wszystkie podpowiedzi dotyczące SQL-a... ale będą tu tylko przez jedną noc (żartujemy!) Znajdziesz tu podsumowanie wszystkich porad i informacji o SQL-u, jakie zamieściliśmy w całej książce. Poświeć chwilę, by przejrzeć listę, i rozkoszuj się poczuciem dumy — już znasz wszystkie te rady i informacje.



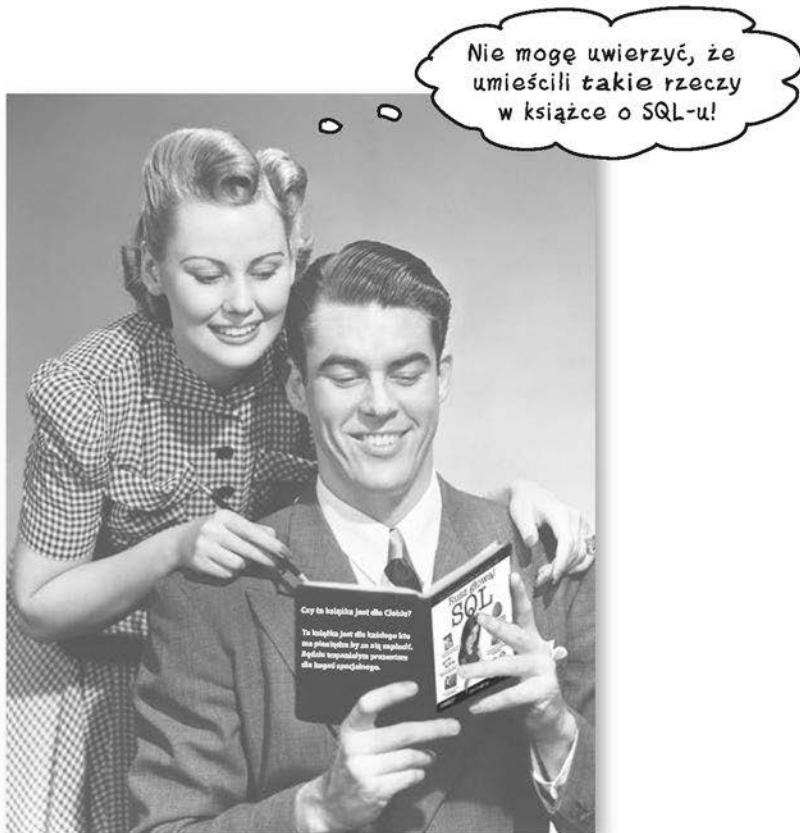
Symboli	576
---------	-----



Skorowidz	583
-----------	-----

Jak korzystać z tej książki

Wprowadzenie



W tej części odpowiadamy na pytanie:
„Dlaczego autorzy UMIEŚCILI
te wszystkie rzeczy w książce
o języku SQL?”

Dla kogo jest ta książka?

Jeśli możesz odpowiedzieć twierdząco na każde z poniższych pytań:

- ① Czy masz dostęp do komputera zainstalowanym systemem zarządzania relacyjnymi bazami danych, takim jak Oracle, MS SQL lub MySQL? Albo do komputera, na którym możesz zainstalować MySQL lub inny system zarządzania bazami danych?
- ② Czy chcesz nauczyć się, zrozumieć i zapamiętać, jak należy tworzyć tabele i bazy danych oraz pisać zapytania, wykorzystując przy tym najlepsze i najnowsze standardy?
- ③ Czy wolisz stymulujące rozmowy przy posiłku od suchych i nudnych wykładów akademickich?

Pomożemy Ci poznać pojęcia stosowane w języku SQL oraz jego składnię w taki sposób, że będzie Ci zdecydowanie łatwiej go zrozumieć i używać dokładnie w taki sposób, w jaki chcesz to robić.

to ta książka nadaje się dla Ciebie.

Kto raczej nie powinien sięgać po tę książkę?

Jeśli możesz odpowiedzieć twierdząco na *którekolwiek* z poniższych pytań:

- ① Doskonale znasz podstawową składnię języka SQL i poszukujesz książki, która pomoże Ci opanować zaawansowane zagadnienia projektowania i korzystania z baz danych.
- ② Jesteś już doświadczonym programistą SQL i szukasz książki informacyjnej.
- ③ Boisz się spróbować czegoś innego? Wolałbyś raczej poddać się leczeniu kanałowemu niż połączyć paski z kratą? Nie wierzasz, że książka techniczna może być poważna, jeśli pojęcia programistyczne będą personifikowane?

Jeśli jednak chciałbyś sobie co nieco przypomnieć, a nigdy tak naprawdę nie rozumiałeś postaci normalnych i relacji jeden-do-wielu oraz lewostronnych złączeń zewnętrznych, to ta książka jest przeznaczona dla Ciebie.

to ta książka nie nadaje się dla Ciebie.



[Notatka od działu marketingu:
ta książka jest dla każdego,
która ma kartę kredytową].

Wiemy, co sobie myślisz

„Jakim cudem *to* może być poważna książka o SQL-u?”

„Po co te wszystkie obrazki?”

„Czy w taki sposób można się czegokolwiek *nauczyć*?”

Wiemy także, co sobie myśli Twój mózg.

Twój mózg pragnie nowości. Zawsze szuka, przegląda i wyczekuje czegoś niezwykłego. W taki sposób został stworzony i to pomaga mu przetrwać.

Zatem co Twój mózg robi z tymi wszystkimi rutynowymi, zwyczajnymi, normalnymi informacjami, jakie do niego docierają? Otóż robi wszystko co tylko może, aby nie przeszkadzały w jego *najważniejszym* zadaniu — zapamiętywaniu rzeczy, które mają *prawdziwe znaczenie*. Twój mózg nie traci czasu i energii na zapamiętywanie nudnych informacji; one nigdy nie przechodzą przez filtr „to jest w oczywisty sposób całkowicie nieważne”.

W jaki sposób Twój mózg wie, co jest istotne? Założmy, że jesteś na codziennej przechadzce i nagle przed Tobą staje tygrys, co się dzieje wewnątrz Twej głowy i ciała?

W dzisiejszych czasach jest mało prawdopodobne, abyś stał się przekąską dla tygrysa. Ale Twój mózg wciąż obserwuje. W końcu nigdy nic nie wiadomo.

Neurony płoną. Emocje szaleją. *Adrenalina napływa falami*.

I właśnie dlatego Twój mózg wie, że...

To musi być ważne! Nie zapominaj o tym!

Ale wyobraź sobie, że jesteś w domu, albo w bibliotece. Jesteś w bezpiecznym miejscu — przytulnym i pozbawionym tygrysów. Uczysz się. Przygotowujesz się do egzaminu. Albo poznajesz jakiś trudny problem techniczny, którego rozwiązanie, według szefa, powinno zająć Ci tydzień, a najdalej dziesięć dni.

Jest tylko jeden, drobny problem. Twój mózg stara się Ci pomóc. Próbuje zapewnić, że te w oczywisty sposób nieistotne informacje nie zajmą cennych zasobów w Twojej głowie. Zasobów, które powinny zostać wykorzystane na zapamiętanie naprawdę ważnych rzeczy. Takich jak tygrysy. Takich jak zagrożenie, jakie niesie ze sobą pożar. Takich jak to, że już nigdy w życiu nie powinieneś jeździć na snowboardzie w krótkich spodenkach.

Co gorsza, nie ma żadnego sposobu, aby powiedzieć mózgowi: „Hej, mózgu mój, dziękuję ci bardzo, ale niezależnie od tego, jak nudna jest tak książka i jak nieznaczne są emocje, jakich aktualnie doznaję, to jednak naprawdę chciałbym zapamiętać wszystkie te informacje”.

Twój mózg myśli, że właśnie *TO* jest istotne.



Wspaniale.
Pozostało jeszcze jedynie
560 głupich, nudnych
i drętwych stron.



Wyobrażamy sobie, że Czytelnik tej książki jest uczniem

A zatem chcesz się czegoś nauczyć? W pierwszej kolejności powinieneś więc to poznać, a następnie postarać się tego nie zapomnieć. To nie polega jedynie na wtłoczeniu do głowy takich faktów. Najnowsze badania prowadzone w dziedzinie przyswajania informacji, neurobiologii i psychologii nauczania pokazują, że uczenie się wymaga czegoś więcej niż tylko czytania tekstu. My wiemy, co potrafi pobudzić nasze mózgi do działania.

Oto niektóre z głównych zasad niniejszej książki:



Wyobraź to sobie wizualnie. Rysunki są znacznie łatwiejsze do zapamiętania niż same słowa i sprawiają, że uczenie staje się znacznie bardziej efektywne (studia nad przypominaniem sobie i przekazywaniem informacji wykazują, że użycie rysunków poprawia efektywność zapamiętywania o 89%). Poza tym rysunki sprawiają, że informacje stają się znacznie bardziej zrozumiałe. Wystarczy umieścić słowa bezpośrednio na lub w okolicach rysunku, do którego się odnoszą, a nie na następnej stronie, a prawdopodobieństwo, że osoby uczące się będą w stanie rozwiązać problem, którego te słowa dotyczą, wzrośnie niemal dwukrotnie.

Stosuj konwersacje i personifikację.

Według najnowszych badań w testach końcowych studenci uzyskiwali wyniki o 40% lepsze, jeśli treść

była przekazywana w sposób bezpośredni, w pierwszej osobie i w konwencji rozmowy, a nie w sposób formalny. Zamiast wykładać, opowiadaj historyjki. Używaj zwyczajnego języka. Nie traktuj swojej osoby zbyt poważnie. Kiedy byłbyś bardziej uważny: podczas stymulującej rozmowy przy obiedzie czy podczas wykładu?



Zmus uczniów do głębszego zastanowienia się. Innymi słowy, jeśli nie zmusisz neuronów do aktywnego wysiłku, w Twojej głowie nie zdarzy się nic wielkiego. Czytelnik musi być zmotywowany, zaangażowany, zaciekawiony i podekscytowany rozwiązywaniem problemów, wyciąganiem wniosków i zdobywaniem nowej wiedzy. A osiągnięcie tego wszystkiego jest możliwe poprzez stawianie wyzwań, zadawanie ćwiczeń i pytań zmuszających do zastanowienia oraz poprzez zmuszanie do działań, które wymagają zaangażowania obu półkul mózgowych i wielu zmysłów.

Zdobądź — przyciągnij na dłużej — uwagę i zainteresowanie czytelnika. Każdy znalazł się kiedyś w sytuacji, gdy bardzo chciał się czegoś nauczyć, lecz zasypiał po przeczytaniu pierwszej strony. Mózg zwraca uwagę na rzeczy niezwykłe, interesujące, dziwne, przyciągające wzrok, nieoczekiwane. Jednak poznawanie nowego technicznego zagadnienia wcale nie musi być nudne. Jeśli będzie to zagadnienie interesujące, Twój mózg przyswoi je sobie znacznie szybciej.



Wyzwól emoce. Teraz już wiemy, że zdolność zapamiętywania informacji jest w znacznej mierze zależna od ich zawartości emocjonalnej. Zapamiętujemy to, na czym nam zależy. Zapamiętujemy w sytuacjach, w których coś odczuwamy. Oczywiście nie mamy tu na myśli wzruszających historii o chłopcu i jego psie. Chodzi nam o emoce

takie jak zaskoczenie, ciekawość, radosne podekscytowanie, „a niech to...” i uczucie satysfakcji — „Jestem wielki!” — jakie odczuwamy po poprawnym rozwiązaniu zagadki, nauczeniu się czegoś, co powszechnie uchodzi za trudne, lub zdaniu sobie sprawy, że znamy więcej szczegółów technicznych niż Robert z działu inżynierii.



Metapoznanie: myślenie o myśleniu

Jeśli naprawdę chcesz się czegoś nauczyć i jeśli chcesz się tego nauczyć szybciej i dokładniej, to zwracaj uwagę, jak Ci na tym zależy. Myśl o tym, jak myślisz. Poznawaj sposób, w jaki się uczysz.

Większość z nas w czasie dorastania nie uczestniczyła w zajęciach z metapoznania albo teorii nauczania. Oczekiwano od nas, że będziemy się uczyć, jednak nie *uczono* nas, jak mamy to robić.

Jednak zakładamy, że jeśli trzymasz w ręku tę książkę, to chcesz nauczyć się SQL-a. I prawdopodobnie nie chcesz na to stracić zbyt wiele czasu. Jeśli chcesz *zapamiętać* wszystkie zdobyte informacje, musisz je wcześniej *zrozumieć*. Aby w jak największym stopniu skorzystać z tej książki, bądź z jakiekolwiek innej książki, lub z dowolnych prób uczenia się czegokolwiek, musisz wziąć odpowiedzialność za swój mózg. Myśl o tym, czego się uczysz.

Sztuczka polega na tym, aby przekonać mózg, że poznawany materiał jest Naprawdę Ważny. Kluczowy dla Twojego dobrego samopoczucia. Tak ważny jak tygrys stojący naprzeciw Ciebie. W przeciwnym razie będziesz prowadzić nieustającą wojnę z własnym mózgiem, który ze wszystkich sił będzie się starać, aby nowe informacje nie zostały utrwalone.



A zatem w jaki sposób zmusić mózg, aby potraktował SQL jak głodnego tygrysa?

Można to zrobić w sposób wolny i męczący lub szybki i bardziej efektywny. Wolny sposób polega na wielokrotnym powtarzaniu. Oczywiście wiesz, że jesteś w stanie nauczyć się i zapamiętać nawet najnudniejsze zagadnienie, moźliwie je „wkuwając”. Po odpowiedniej liczbie powtórzeń Twój mózg stwierdzi: „*Wydaje się, że to nie jest dla niego szczególnie ważne, lecz w kółko to czyta i powtarza, więc przypuszczam, że jakąś wartość to jednak musi mieć*”.

Szybszy sposób polega na zrobieniu *czegokolwiek, co zwiększy aktywność mózgu*, a najlepiej czegoś, co wyzwoli kilka różnych typów aktywności. Wszystkie zagadnienia, o których pisaliśmy na poprzedniej stronie, są kluczowymi elementami rozwiązania i udowodniono, że wszystkie z nich potrafią pomóc w zmuszeniu mózgu, aby pracował na Twoją korzyść. Badania wykazują na przykład, że umieszczenie słów na opisywanych rysunkach (a nie w innych miejscach tekstu na stronie, na przykład w nagłówku lub wewnątrz akapitu) sprawia, że mózg stara się zrozumieć relację pomiędzy słowami i rysunkiem, a to z kolei zwiększa aktywność neuronów. Większa aktywność neuronów to z kolei większe szanse, że mózg uzna informacje za warte zainteresowania i, ewentualnie, zapamiętania.

Prezentowanie informacji w formie konwersacji pomaga, gdyż ludzie zdają się wykazywać większe zainteresowanie w sytuacjach, gdy uważają, że biorą udział w rozmowie, gdyż oczekuje się od nich, że będą śledzić jej przebieg i brać w niej czynny udział. Zadziwiające jest, iż mózg zdaje się nie zważyć na to, że rozmowa jest prowadzona z książką! Z drugiej strony, jeśli sposób przedstawiania informacji jest formalny i suchy, mózg postrzega to tak samo jak w sytuacji, gdy uczestniczysz w wykładzie na sali pełnej sennych słuchaczy. Nie ma potrzeby wykazywania jakiegokolwiek aktywności.

Jednak rysunki i przedstawianie informacji w formie rozmowy to jedynie początek.

Jak korzystać z tej książki

Oto co zrobiliśmy:

Używaliśmy **rysunków**, ponieważ Twój mózg zwraca większą uwagę na obrazy niż na tekst. Jeśli chodzi o mózg, to faktycznie jeden obraz jest wart 1024 słów. W sytuacjach, gdy pojawiał się zarówno tekst, jak i rysunek, umieszczaliśmy tekst *na rysunku*, gdyż mózg działa bardziej efektywnie, gdy tekst jest *wewnątrz* czegoś, co opisuje, niż kiedy jest umieszczony w innym miejscu i stanowi część większego fragmentu tekstu.

Stosowaliśmy **powtórzenia**: wielokrotnie podawaliśmy tę samą informację na *różne* sposoby i przy wykorzystaniu różnych środków przekazu oraz odwoływaliśmy się do *różnych zmysłów*. Wszystko po to, aby zwiększyć szanse, że informacja zostanie zakodowana w większej ilości obszarów Twojego mózgu.

Używaliśmy pomysłów i rysunków w zaskakujący sposób, ponieważ Twój mózg oczekuje i pragnie nowości; poza tym staraliśmy się zawrzeć w nich *chociaż trochę emocji*, gdyż mózg jest skonstruowany w taki sposób, iż zwraca uwagę na biochemię związaną z emocjami. Prawdopodobieństwo zapamiętania czegoś jest większe, jeśli to „coś” sprawia, że coś poczujemy, nawet jeśli to uczucie nie jest niczym więcej niż lekkim *rozbawieniem, zaskoczeniem lub zainteresowaniem*.

Używaliśmy bezpośrednich zwrotów i przekazywaliśmy treści *w formie konwersacji*, gdyż mózg zwraca większą uwagę, jeśli uważa, że prowadzisz rozmowę, niż gdy jesteś jedynie biernym słuchaczem prezentacji. Mózg działa w ten sposób, nawet gdy *czytasz* rozmowę.

Zamieściliśmy w książce ponad 80 **ćwiczeń**, ponieważ mózg uczy się i pamięta więcej, gdy coś *robi*, niż gdy o czymś *czyta*. Poza tym podane ćwiczenia stanowią wyzwania, choć nie są przesadnie trudne, gdyż właśnie takie preferuje większość osób.

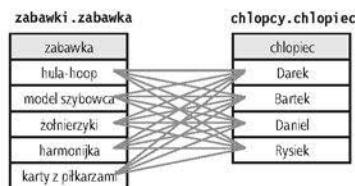
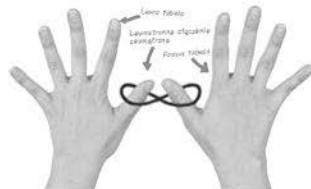
Stosowaliśmy wiele **stylów nauczania**, gdyż Ty możesz preferować instrukcje opisujące krok po kroku sposób postępowania, ktoś inny woli analizowanie zagadnienia opisanego w ogólny sposób, a jeszcze inne osoby — przejrzenie przykładowego fragmentu kodu. Jednak niezależnie od ulubionego sposobu nauki *każdy* skorzysta na tym, że te same informacje będą przedstawiane kilkakrotnie na różne sposoby.

Podawaliśmy informacje przeznaczone dla *obu półkul Twojego mózgu*, gdyż im bardziej mózg będzie zaangażowany, tym większe jest prawdopodobieństwo nauczenia się i zapamiętania podawanych informacji i tym dłużej możesz koncentrować się na nauce. Ponieważ angażowanie tylko jednej półkuli mózgu często oznacza, że druga będzie mogła odpocząć, zatem będziesz mógł uczyć się bardziej produktywnie przez dłuższy czas.

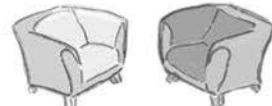
Dodatkowo zamieszczaliśmy **opowiadania** i ćwiczenia prezentujące *więcej niż jeden punkt widzenia*, ponieważ mózg uczy się dokładniej, gdy jest zmuszony do przetwarzania i podawania własnej opinii.

Stawialiśmy przed Tobą **wyzwania**, zarówno poprzez podawanie ćwiczeń, jak i stawiając *pytania*, na które nie zawsze można odpowiedzieć w prosty sposób; a to dlatego, że mózg uczy się i zapamiętuje, gdy musi *popracować* nad czymś (to tak samo, jak nie można zdobyć dobrej formy fizycznej, obserwując ćwiczenia w telewizji). Jednak dołożyliśmy wszelkich starań, aby zapewnić, że gdy pracujesz, to robisz *dokładnie to, co trzeba*. Aby ani jeden dendryt nie musiał przetwarzać trudnego przykładu ani analizować tekstu zbyt lapidarnego lub napisanego trudnym żargonem.

Personifikowaliśmy tekst. W opowiadaniach, przykładach, rysunkach i wszelkich innych możliwych miejscach tekstu staraliśmy się personifikować tekst, gdyż jesteś osobą, a Twój mózg zwraca większą uwagę na *osoby* niż na *rzeczy*.



Pogawędkи przy kominku



CELNE SPOSTRZEŻENIA





Oto co możesz zrobić, aby zmusić swój mózg do posłuszeństwa

A zatem zrobiliśmy co w naszej mocy. Reszta zależy od Ciebie. Możesz zacząć od poniższych porad. Posłuchaj swojego mózgu i określ, które sprawdzają się w Twoim przypadku, a które nie dają pozytywnych rezultatów. Spróbuj czegoś nowego.

*Wytnij te porady
i przyklej na lodówce.*

① Zwolnij. Im więcej rozumiesz, tym mniej musisz zapamiętać.

Nie ograniczaj się jedynie do czytania. Przerwij na chwilę lekturę i pomyśl. Kiedy znajdziesz w tekście pytanie, nie zaglądaj od razu na stronę odpowiedzi. Wyobraź sobie, że ktoś faktycznie zadaje Ci pytanie. Im bardziej zmusisz swój mózg do myślenia, tym większe będą szanse, że się nauczysz i zapamiętasz dane zagadnienie.

② Wykonuj ćwiczenia. Rób notatki.

Umieszczaliśmy je w tekście, gdybyśmy jednak zrobili je za Ciebie, to niczym nie różniłoby się to od sytuacji, w której ktoś wykonywałby ćwiczenia fizyczne za Ciebie. I nie ograniczaj się jedynie do czytania ćwiczeń. Używaj ołówka. Można znaleźć wiele dowodów na to, że fizyczna aktywność podczas nauki może poprawić jej wyniki.

③ Czytaj fragmenty oznaczone jako „Nie istnieją głupie pytania”.

Chodzi tu o wszystkie fragmenty umieszczone z boku tekstu. Nie są to fragmenty opcjonalne — stanowią one część podstawowej zawartości książki! Nie pomijaj ich.

④ Niech lektura tej książki będzie ostatnią rzeczą, jaką robisz przed pójściem spać. A przynajmniej ostatnią czynnością stanowiącą wyzwanie intelektualne.

Pewne etapy procesu uczenia się (a w szczególności przenoszenie informacji do pamięci długotrwałej) mają miejsce po odłożeniu książki. Twój mózg potrzebuje trochę czasu dla siebie i musi dodatkowo przetworzyć dostarczone informacje. Jeśli podczas tego czasu koniecznego na wykonanie dodatkowego „przetwarzania” zmusisz go do innej działalności, to część z przyswojonych informacji może zostać utracona.

⑤ Pij wodę. Dużo wody.

Twój mózg pracuje najlepiej, gdy dostarcza się mu dużo płynów. Odwodnienie (które może następować, nawet zanim poczujesz pragnienie) obniża zdolność percepcji.

⑥ Rozmawiaj o zdobywanych informacjach. Na głos.

Mówienie aktywuje odmienne fragmenty mózgu. Jeśli próbujesz coś zrozumieć lub zwiększyć szanse na zapamiętanie informacji na dłużej, powtarzaj je na głos. Jeszcze lepiej — staraj się je na głos komuś wytłumaczyć. W ten sposób nauczysz się szybciej, a oprócz tego będziesz mógł odkryć pomysły, o których nie wiedziałeś podczas czytania tekstu książki.

⑦ Posłuchaj swojego mózgu.

Zaobserwuj, kiedy Twój mózg staje się przeciążony. Jeśli zauważysz, że zaczynasz czytać побieżnie i zapominać to, o czym przeczytałeś przed chwilą, to najwyższy czas, żeby sobie zrobić przerwę. Po przekroczeniu pewnego punktu nie będziesz się uczył szybciej, „wciskając” do głowy więcej informacji; co gorsza, może to zaszkodzić całemu procesowi nauki.

⑧ Poczuj coś!

Twój mózg musi wiedzieć, że to, czego się uczysz, ma znaczenie. Z zaangażowaniem śledź zamieszczane w tekście opowiadania. Nadawaj własne tytuły zdjęciom. Zalewanie się łzami ze śmiechu po przeczytaniu głupiego dowcipu i tak jest lepsze od braku jakiekolwiek reakcji.

⑨ Zaprojektuj coś!

Zastosuj informacje zdobywane podczas lektury tej książki do stworzenia czegoś nowego, co właśnie projektujesz, lub do zmodyfikowania jakiegoś starego projektu. Po prostu zrób coś, co pozwoli Ci zdobyć doświadczenie wykraczające poza ćwiczenia i przykłady prezentowane w książce. Wszystko, czego Ci będzie w tym celu potrzeba, to problem do rozwiązania... problem, którego rozwiązanie możesz ulepszyć, stosując prezentowane w książce techniki.

Przeczytaj to

To książka do nauki, a nie encyklopedia. Celowo usunęliśmy wszystko, co mogłoby Ci przeszkadzać w nauce, niezależnie od tego, nad czym pracujesz w danym miejscu książki. Podczas pierwszej lektury książki należy zaczynać od jej samego początku, gdyż kolejne rozdziały bazują na tym, co widziałeś i czego się dowiedziałeś wcześniej.

Zaczniemy od nauczenia Cię podstawowej składni języka SQL, następnie przejdziemy do pojęć i zagadnień związanych z projektowaniem baz danych, aby w końcu zająć się technikami tworzenia złożonych zapytań.

Choć tworzenie prawidłowo zaprojektowanych baz danych i tabel ma bardzo duże znaczenie, to jednak zanim będziesz mógł to robić, musisz poznać i zrozumieć składnię języka SQL. I dlatego zaczniemy od przedstawienia poleceń SQL, które będziesz mógł sam wypróbować. Dzięki temu od razu będziesz mógł coś zrobić i na pewno będziesz tym bardzo podekscytowany. Dopiero w nieco dalszej części książki pokażemy Ci zasady prawidłowego projektowania tabel. Do tego czasu będziesz już doskonale znał potrzebne polecenia SQL i będziesz mógł skoncentrować się na *nauce pojęć*, a nie na sprawach technicznych.

Nie przedstawimy wszystkich istniejących poleceń SQL, funkcji ani słów kluczowych.

Choć mogliśmy podać w tej książce każde polecenie SQL, funkcję oraz słowo kluczowe, to jednak doszliśmy do wniosku, że wolałbyś korzystać z książki, którą bez trudu mógłbyś podnieść i która nauczy Cię jedynie najważniejszych i najbardziej przydatnych poleceń, funkcji i słów kluczowych. Przedstawimy Ci tylko te z nich, które musisz znać i z których będziesz korzystał w 95 procentach przypadków. A kiedy skończysz czytać tę książkę, będziesz na tyle pewny swych umiejętności i znajomości języka SQL, że bez najmniejszych problemów odszukasz funkcję, która będzie Ci potrzebna do napisania całkowicie odratowanego zapytania.

Nie przedstawimy wszystkich istniejących systemów zarządzania relacyjnymi bazami danych.

Dostępnych systemów zarządzania relacyjnymi bazami danych jest całkiem sporo, oto tylko kilka z nich: Standard SQL, MySQL, Oracle, MS SQL, PostgreSQL, DB2. Gdybyśmy chcieli opisać wszystkie możliwe wariacje składni wszystkich istniejących poleceń, to niniejsza książka miałaby znacznie, znacznie więcej stron. My natomiast lubimy drzewa, więc koncentrujemy się na bazie Standard SQL, z niewielkim ukłonem w stronę MySQL-a. Wszystkie przykłady zamieszczone w niniejszej książce działają w MySQL-u; a niemal wszystkie będą działały także w innych, podanych powyżej systemach zarządzania relacyjnymi bazami danych.

Ćwiczenia SĄ obowiązkowe.

Ćwiczenia oraz wszelkie dodatkowe polecenia nie są jedynie dodatkami — stanowią integralną część podstawowej treści książki. Niektóre z nich zostały umieszczone po to, by pomóc w zapamiętaniu informacji, inne, by pomóc w zrozumieniu opisywanego materiału, a jeszcze inne — by pomóc Ci w praktycznym zastosowaniu zdobytej wiedzy. Nie pomijaj ich!

Powtórzenia są celowe i ważne.

Jedną z cech wyróżniających serię książek Rusz głową! jest to, iż naprawdę bardzo, bardzo, bardzo zależy nam na tym, abyś wszystko zrozumiał i przyswoił. Chcielibyśmy także, abyś po zakończeniu lektury niniejszej książki pamiętał informacje, które w niej zamieściliśmy. W większości książek informacyjnych i encyklopedycznych nie kładzie się nacisku na przyswojenie i zapamiętanie informacji, jednak w tej znajdziesz wiele pojęć, które pojawiają się kilka razy. Badania mózgu wykazują, że przeniesienie informacji do pamięci długotrwałej wymaga zazwyczaj minimum trzech „powtórzeń”.

Przykładowe kody są jak najbardziej zwięzłe.

Czytelnicy niejednokrotnie zwracali nam uwagę, że przeglądanie 200 wierszy kodu w poszukiwaniu dwóch linijek, które należy zrozumieć, może być frustrujące. W większości przykładów zamieszczonych w tej książce dodatkowy kod, który nie jest bezpośrednio związany z omawianymi zagadnieniami, został w jak największym stopniu skrócony; dzięki temu fragmenty, których musisz się nauczyć, są przejrzyste i proste. Nie należy zatem oczekwać, że podawane przykłady będą solidne, ani nawet, że będą kompletnie — zostały one opracowane wyłącznie pod kątem nauki, a nie pod kątem zapewnienia pełnej funkcjonalności.

Wiele z poleceń i zapytań przedstawionych w tej książce udostępniliśmy na stronach WWW, dzięki czemu będziesz je mógł skopiować i wkleić do okna terminalu lub do innego programu, którego używasz do obsługi bazy. Wszystkie przygotowane przykłady znajdziesz pod adresem <ftp://ftp.helion.pl/przyklady/sqlrug.zip>

Do ćwiczeń z cyklu „Wysil szare komórki” nie podawaliśmy odpowiedzi.

Do niektórych w ogóle nie można podać jednej dobrej odpowiedzi; w innych przypadkach to doświadczenie, które zdobywasz, rozwiązuając te ćwiczenia, ma dać Ci możliwość określenia, czy i kiedy podana odpowiedź będzie poprawna. W niektórych ćwiczeniach z tej serii znajdziesz także podpowiedzi, które ułatwią Ci znalezienie rozwiązania.

Nasi wspaniali recenzenci



Cary Collett



Steve Milano



Shelley Rheams



Jamie Henderson



LuAnn Mazza

Ogromne podziękowania należą się naszemu zespołowi recenzentów. Wychwycili ono masę rażących błędów, błędów subtelnych oraz marnych błędów literowych. Bez nich niniejsza książka byłaby nieporównywalnie gorsza. Wykonali wspaniałą robotę, jeśli chodzi o odnajdywanie i usuwanie błędów.

Cary Collett wykorzystał podczas poprawiania tej książki swoje 15-letnie doświadczenia wyniesione z pracy z literackimi nowicjuszami w laboratoriach rządowych i ostatnio w sektorze finansowym, i teraz z radością oczekuje na chwilę, kiedy znów będzie się mógł zająć sprawami niezwiązonymi z pracą zawodową, takimi jak: gotowanie, spacery, czytanie i terroryzowanie swoich psów.

LuAnn Mazza znalazła nieco czasu mimo swojej pracy zawodowej w Illinois, gdzie pracuje jako programistka i analityk, by pisać bardzo cenne i celne recenzje. Cieszymy się, że już będzie mieć nieco więcej czasu na swoje hobby, takie jak jazda na rowerze, fotografia, komputery, muzyka czy też tenis.

Kiedy **Steve Milano** nie programuje w jednym z przynajmniej kilku języków, co jest jego codziennym zajęciem, ani nie recenzuje książki SQL. Rusz głową!,

ani nie gra punk rocka ze swoją kapelą Onion Flavored Rings w jakichś dusznych piwnicach, to można go znaleźć w domu z jego dwoma kotami Ralphem i Squeakiem.

„**Shelley**” **Moira Michelle Rheams** jest magistrem pedagogiki, ma tytuły MCP, MCSE i prowadzi Early Childhood Education Program (program wczesnego nauczania) w Delgado Community College w West Bank Campus w Nowym Orleanie. Obecnie fascynuje się tworzeniem kursów edukacyjnych prowadzonych w internecie, które sprostałyby nowym wymaganiom mieszkańców miasta odbudowującego się po huraganie Katrina. Dziękujemy Ci za to, że zdołałaś umieścić pracę nad tą książką w swoim i tak przepięknym terminarzu.

Jamie Henderson jest starszym architektem systemów; ma fioletowe włosy i dzieli swój wolny czas pomiędzy grę na wiolonczeli, czytanie, gry wideo i oglądanie filmów na DVD.

To właśnie dzięki temu fantastycznemu zespołowi przykłady i ćwiczenia zamieszczone w tej książce faktycznie będą robić to, co powinny, a Ty po skończeniu lektury będziesz bardzo dobrym i pewnym siebie programistą SQL. Ich dbałość o detale sprawiła, że byliśmy mniej przemądrzali i protekcyjonalni, a czasami nawet nie aż tak bardzo „zakręceni”.

Podziękowania

Moim redaktorom:

Przede wszystkim chciałabym podziękować mojemu redaktorowi, **Brettowi McLaughlinowi**, za nie jeden, lecz dwa obozy treningowe Rusz głową!. Brett był więcej niż redaktorem — był połączeniem doradcy i tragarza. Ta książka absolutnie nie powstała bez jego przewodnictwa, wsparcia i zainteresowania. Nie tylko urzekł mnie już od pierwszego spotkania, lecz także jego akceptacja mojego czasami nieco przesadnego poczucia humoru sprawiła, że praca nad tą książką była najwspanialszym doświadczeniem zawodowym, jakie kiedykolwiek miałam. W czasie prac nad tym projektem Brett udzielił mi całej masy porad, wskazówek i znacznie więcej niż jedynie korepetycji. Dzięki, Brett!



Brett McLaughlin ↑



← Catherine Nolan

Z powodu niesłychanego pecha, jaki miałam pod koniec prac redakcyjnych nad tą książką, redaktor **Catherine Nolan** ma teraz olbrzymi wrzód. To tylko dzięki niej ta książka nie ukazała się w roku 2008, a może lepiej byłoby powiedzieć, że tylko dzięki niej w ogóle się ukazała. Sytuację, jaka zaistniała pod koniec prac, można by porównać z żonglowaniem kociakami, ale Catherine nie upuściła ani jednego. Strasznie były mi potrzebne wyznaczone terminy, a ona doskonale sobie z tym poradziła. Sądzę, że byłam jej największym zawodowym wyzwaniem. Mam nadzieję, że jej następny projekt będzie realizowany znacznie sprawniej i spokojniej; bardzo sobie na to zasłużyła.

W wydawnictwie O'Reilly:

Redaktor projektu, **Louise Barr**, była nie tylko wspaniałą przyjaciółką, lecz także doskonałą projektantką grafiki. W jakiś tajemniczy sposób była w stanie przekształcić moje szalone pomysły w imponujące rysunki, które cudownie wyjaśniały nawet bardzo trudne pojęcia. Wszystkie te wspaniałe projekty graficzne zdobiące książkę są jej autorstwa i nie mam najmniejszych wątpliwości, że w wielu miejscach tej książki także Ty będziesz chciał jej podziękować.

Niemniej jednak ta książka trafiłaby do druku z całą masą błędów, gdyby nie korekta techniczna, którą przeprowadził **Sanders Kleinfeld**. Wykonał on wspaniałą robotę jako redaktor wykonawczy i przygotował tę książkę do druku. Oprócz tego Sanders wyszedł znacznie poza zakres swych obowiązków i wskazał kilka pojęciowych przepaści, których krawędzie trzeba było ze sobą połączyć.



↑ Lou Barr

I w końcu chciałabym podziękować **Kathy Sierra** oraz **Bertowi Batesowi** za stworzenie tej wspaniałej serii książek oraz za najlepszy i najbardziej wyzywający duchowo trening, jaki przeszłam na obozie szkoleniowym Rusz głową!. Nawet nie chcę myśleć, o ile trudniej byłoby mi napisać tę książkę, gdyby nie te trzy dni. Poza tym ostatnie redakcyjne komentarze Berta były boleśnie trafne i w ogromnym stopniu wpłynęły na jakość książki.

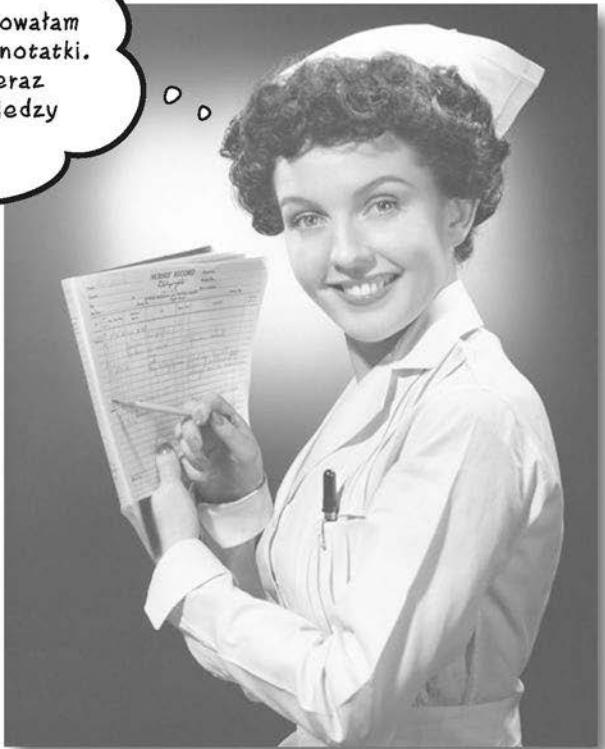
1. Dane i tabele



Na wszystko znajdzie się odpowiednie miejsce



Kiedyś wszystkie informacje o moich pacjentach notowałam na papierze, lecz ciągle gubiłam te notatki. W końcu nauczyłam się SQL-a i teraz już nikt mi się nie gubi. Trochę wiedzy o tabelach jeszcze nikomu nie zaszkodziło!



Czy także i Ty nie cierpisz gubienia czegokolwiek? Niezależnie od tego, czy są to kluczyki do samochodu, bon uprawniający do zakupu mieszkania z 25-procentową zniżką, dane używanej aplikacji, nie ma nic gorszego niż **niemożność sprostania własnym potrzebom**... wtedy gdy tego najbardziej potrzebujemy. A jeśli chodzi o używane aplikacje, to trzeba wiedzieć, że nie ma lepszego miejsca na przechowywanie ważnych informacji niż **tabele**. A zatem przewróć kartkę i zacznij, krok za krokiem, poznawać świat **relacyjnych baz danych**.

Samoprzylepne karteczki

Definiowanie danych

Grzegorz zna wiele samotnych osób. Lubi notować sobie informacje o zainteresowaniach swoich przyjaciół i poznawać ich ze sobą. Grześ ma bardzo dużo informacji o swoich znajomych, zapisanych na samoprzylepnych karteczkach, takich jak te przedstawione poniżej:



Grzesiek używa tego systemu już od bardzo długiego czasu. Jednak weszły tygodniu rozszerzył swoje dane o osoby poszukujące pracy, przez co ich ilość zaczęła szybko rosnąć. Bardzo szybko...



To jedynie kilka spośród notatek Grześka



WYSIL SZARE KOMÓRKI

Czy istnieje lepszy sposób zorganizowania tych informacji?
Co Ty byś zrobił?



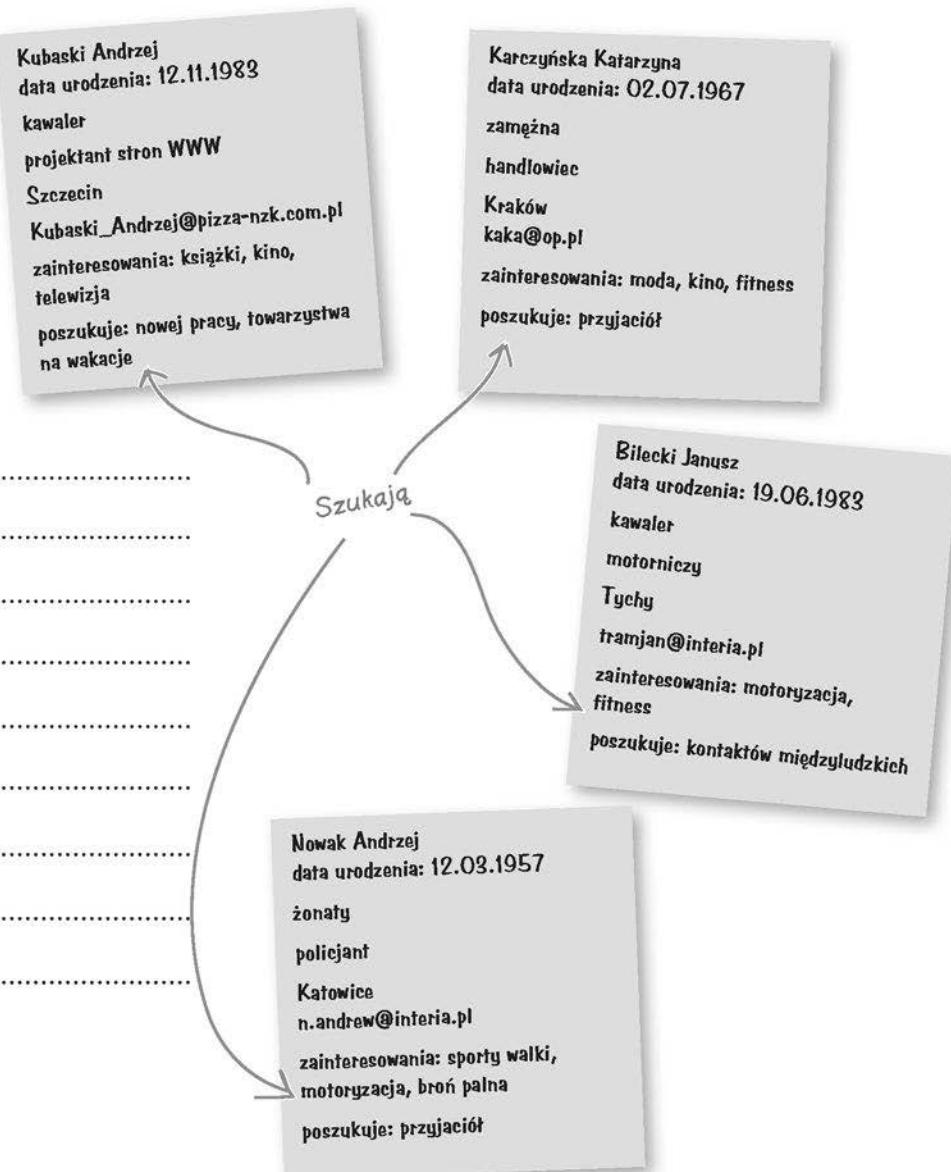
Dokładnie. Baza danych jest właśnie tym, czego nam potrzeba.

Jednak zanim będziesz mógł zacząć tworzyć bazy danych, musisz dowiedzieć się czegoś więcej o *rodzajach* danych, jakie będą w nich przechowywane, oraz poznać sposoby ich *kategoryzacji*.

Zaostrz ołówek



Oto kilka notatek Grześka. Poszukaj podobnych informacji, które Grzesiek notuje na temat każdej z osób. Każdej z odnalezionych wspólnych informacji nadaj nazwę określającą kategorię, do jakiej dana informacja należy. Następnie zapisz te nazwy w pustych wierszach na poniższym rysunku.



Rozwiążanie zadania

Zaostrz ołówek



Rozwiążanie

Oto kilka notatek Grześka. Poszukaj podobnych informacji, które Grzesiek notuje na temat każdej z osób. Każdej z odnalezionych wspólnych informacji nadaj nazwę określającą kategorię, do jakich dana informacja należy. Następnie zapisz te nazwy w pustych wierszach na poniższym rysunku.

Skoro już określiliśmy kategorie, możemy ich użyć do zorganizowania danych Grześka.



Stan

Imię

Nazwisko

Jak widać, rozróżniamy osobno imię oraz nazwisko. Dzięki temu łatwiej nam będzie sortować dane.

Imię

Nazwisko

Data urodzenia

Zawód

Stan

Lokalizacja

E-mail

Zainteresowania

Poszukuje

Miejsce zamieszkania

Zawód

E-mail

Nowak Andrzej
data urodzenia: 12.03.1957

żonaty

policjant

Katowice

n.andrew@interia.pl

zainteresowania: sporty walki, motoryzacja, broń palna

poszukuje: przyjaciół

Mendralska Agnieszka
data urodzenia: 19.08.1979

zamężna

administrator systemów Unix

Sandomierz

agitmedra@swiatkawy.com.pl

zainteresowania: aktorstwo, taniec

poszukuje: nowej pracy

Grzesiek przypisał już pewnym informacjom nazwy kategorii, takie jak „Data urodzenia”, „Zainteresowania” czy też „Poszukuje”, i zanotował je na swoich karteczkach.

Przeanalizuj swoje dane pod względem kategorii

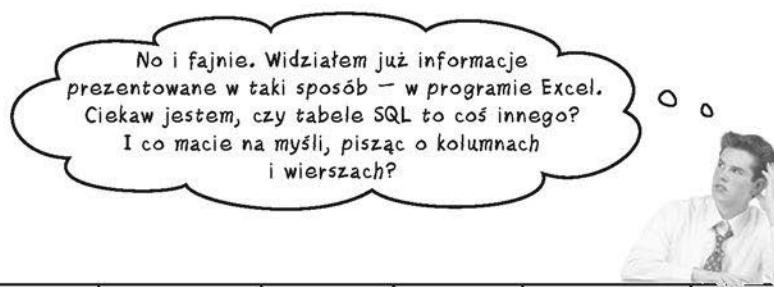
A teraz przeanalizuj swoje dane w nieco inny sposób. Gdybyś pociął każdą karteczkę na mniejsze kawałki, a następnie rozłożył je w poziomym rzędzie, to otrzymałbyś coś przypominającego poniższy rysunek:

Agnieszka	Mendralská	19.08.1979	administrator systemów Unix	zamężna	Sandomierz	agimedra@swiatkawy.com.pl	aktorstwo, taniec	nowej pracy
-----------	------------	------------	-----------------------------	---------	------------	---------------------------	-------------------	-------------

Jeśli teraz potniesz na kawałczki karteczkę, na której zapisałesz nazwy wyróżnionych wcześniej kategorii informacji, i umieścisz je nad kawałczkami z samymi informacjami, uzyskasz coś przypominającego poniższy rysunek:

Imię	Nazwisko	Data urodzenia	Zawód	Stan	Lokalizacja	E-mail	Zainteresowania	Poszukuje
Agnieszka	Mendralská	19.08.1979	administrator systemów Unix	zamężna	Sandomierz	agimedra@swiatkawy.com.pl	aktorstwo, taniec	nowej pracy

A poniżej te same informacje, które na poprzedniej stronie przedstawiliśmy na karteczkach, zostały zapisane w formie **TABELI** składającej się z **kolumn** i **wierszy**.



nazwisko	imie	email	data_urodzenia	zawod	lokalizacja	stan	zainteresowania	szuka
Barańska	Anna	aria@megadeski.com.pl	01.07.1962	inżynier lotniczy	Warszawa	panna, ale ma partnera	gry RPG, programowanie	nowej pracy
Homilski	Janusz	hojan@pizza-nzk.com.pl	10.09.1966	administrator systemów komputerowych	Darłówka	kawaler	wędrówki piesze, pisanie blogów	przyjaciół, kobiety na randki
Samotek	Adrian	asamotek@pizza-nzk.com.pl	02.12.1975	inżynier lotniczy	Warszawa	żonaty	gry RPG, programowanie	niczego
Mendralská	Agnieszka	agimedra@swiatkawy.com.pl	19.08.1979	administrator systemów Unix	Sandomierz	zamężna	aktorstwo, taniec	nowej pracy

Co znajduje się w bazie danych?

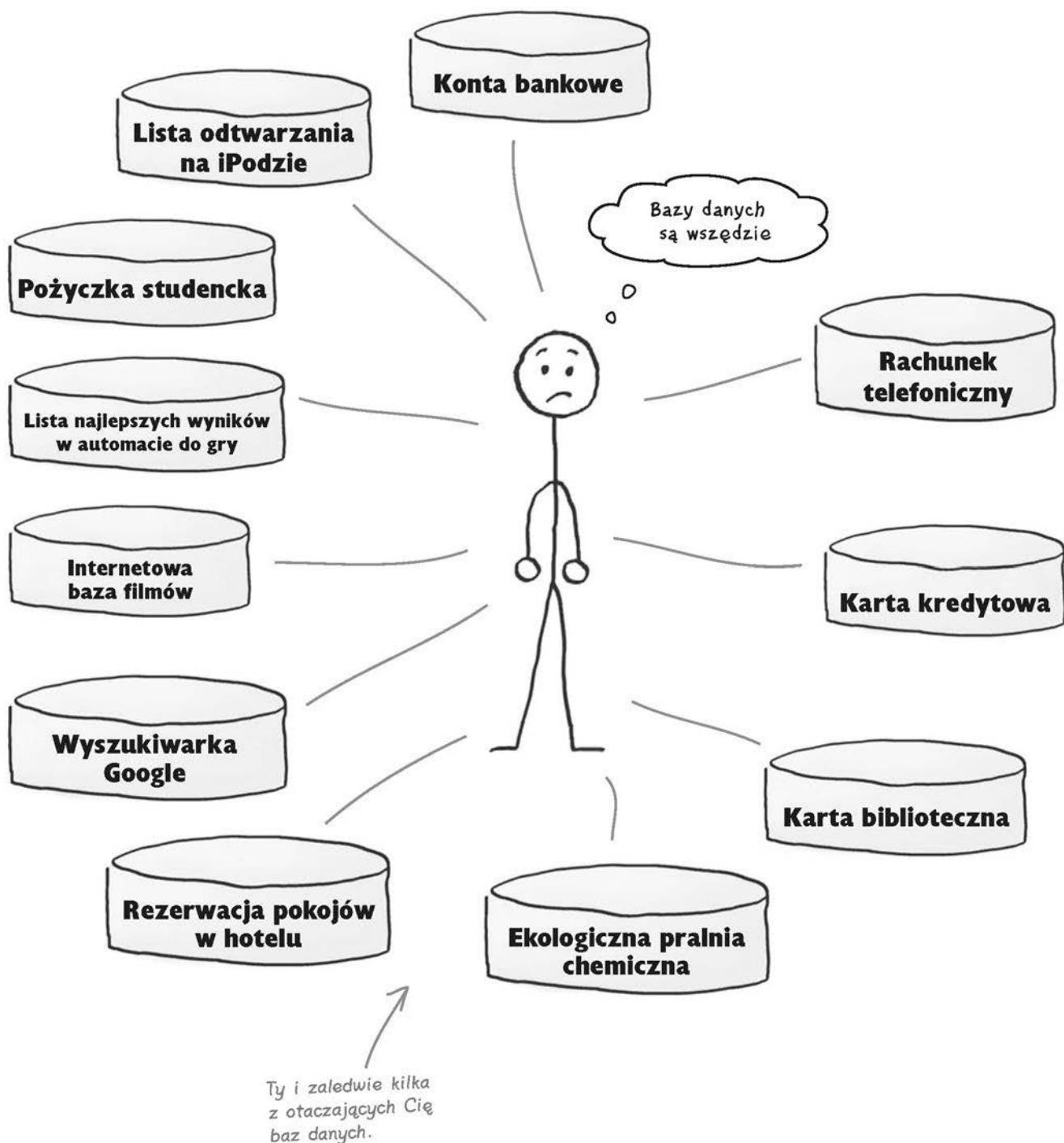
Zanim zajmiemy się znalezieniem szczegółowych odpowiedzi na pytanie, czym są tabele, wiersze i kolumny, warto się zatrzymać i przyjrzeć całemu zagadnieniu w sposób nieco bardziej ogólny. Pierwszą strukturą SQL, jaką musisz poznać, strukturą, która przechowuje wszystkie tabele, jest **baza danych**.

Baza danych to swoisty pojemnik, przechowujący tabele oraz inne, powiązane z nimi struktury SQL.

Za każdym razem, gdy szukasz czegoś w internecie, robisz zakupy w internetowym sklepie, korzystasz z cyfrowego wideo, kupujesz bilety na wyścigi lub zamawiasz pizzę, niezbędne informacje są pobierane z bazy danych w wyniku *wykonania zapytania*.



We wszelkiego typu diagramach oraz schematach przepływu sterowania bazy danych są zazwyczaj przedstawiane w formie cylindrów. A zatem, jeśli zobaczyś coś takiego, od razu będziesz wiedział, że chodzi o bazę danych.





Anatomia bazy danych



Twoja baza danych widziana przez rentgenowskie okulary...

Baza danych zawiera tabele.

Tabela jest wewnętrzną strukturą bazy danych; są w niej umieszczane informacje. Informacje w tabelach są rozmieszczone w **wierszach** i **kolumnach**.

Czy pamiętasz te kategorie, które wyróżniłeś na jednej z poprzednich stron? Każda z nich stanie się **kolumną** tabeli. W jednej kolumnie mogłyby się znaleźć na przykład następujące wartości: Kawaler, Żonaty, Rozwiedziony.

Z kolei **wiersz** tabeli zawiera wszystkie informacje o jednym obiekcie. Wracając do przykładu nowej tabeli Grześka: w jednym wierszu znalazłyby się wszystkie informacje dotyczące jednej osoby. Oto przykład informacji, jakie mogłyby się znaleźć w jednym wierszu: Jacek, Jackowski, kawaler, pisarz, jj@megadeski.com.pl.

Informacje przechowywane w bazie danych są umieszczane w **tabelach**.



Bazodanowy objazd



Bądź tabelą

Poniżej przedstawionych zostało kilka karteczek oraz tabela. Twoim zadaniem jest wcielenie się w rolę częściowo wypełnionej tabeli i uzupełnienie wszystkich pustych miejsc w celu uzyskania wewnętrznego spokoju i harmonii. Kiedy rozwiążesz zadanie, przewróć kartkę i sprawdź, czy Ty i tabela staliście się jednością.

Paczki u Donalda

5

25.4

z dżemem

8:56

ciuste

Kawiarnia Gwiezdny Pył

23.4

z dżemem

9

7:43

niemal doskonale

Paczki u Donalda

7

24.4

za mało dżemu

10:35

z dżemem

z dżemem

twardawe, ale smaczne

6

Chrupki Kąsek

26.4

9:39

Użyj jednego z pól jako nazwy tabeli, która określi jej zawartość.



sklep				
			9	
		25.4	5	
				za mało dżemu



Bazodanowy objazd

Bądź tabelą. Rozwiążanie



Nie przejmuj się, jeśli podane przez Ciebie nazwy kolumn nie są dokładnie takie same jak nasze.

Twoim zadaniem było wcielenie się w rolę częściowo wypełnionej tabeli i uzupełnienie wszystkich pustych miejsc w celu uzyskania wewnętrznego spokoju i harmonii.

Nazwę tabeli bez większych problemów możesz określić na podstawie zawartości przedstawionych karteczek.

paczki_z_dzemem

sklep	godzina	data	ocena	komentarze
Kawiarnia Gwiezdny Pył	7:43	23.4	9	niemal doskonałe
Pączki u Donalda	8:56	25.4	5	tluste
Chrupki Kąsek	9:39	26.4	6	twardawe, ale smaczne
Pączki u Donalda	10:35	24.4	7	za mało dzemu

Bazy danych zawierają powiązane ze sobą informacje

Wszystkie tabele umieszczone w bazie danych powinny być w jakiś sposób ze sobą **powiązane**. Na przykład poniżej przedstawiono tabele, które mogłyby się znaleźć w bazie danych dotyczącej pączków:

Oto baza danych zawierająca trzy tabele. Baza ta nosi nazwę moje_przekaski.

Nazwy baz danych i tabel są zazwyczaj zapisywane wyłącznie małymi literami.

moje_przekaski

Tabela zawierająca informacje o pączkach z dzemem.

paczki_z_dzemem

sklep	godzina	data	ocena	komentarze
Kawiarnia Gwiezdny Pył	7:43	23.4	9	niemal doskonałe
Pączki u Donalda	8:56	25.4	5	tluste
Chrupki Kąsek	9:39	26.4	6	twardawe, ale smaczne
Pączki u Donalda	10:35	24.4	7	za mało dzemu

paczki_lukrowane

sklep	godzina	data	ocena	komentarze
Kawiarnia Gwiezdny Pył	9:39	26.5	8	ciepły, ale nie gorący
Pączki u Donalda	7:43	23.5	4	mało lukru
Chrupki Kąsek	8:56	25.5	6	smaczny
Ciastkarnia u Misia	10:35	24.5	7	nieco twardawy

inne_ciastka

sklep	godzina	data	ciastko	ocena	komentarze
Kawiarnia Gwiezdny Pył	10:35	13.3	mleczka	6	zbyt dużo przypraw
Pączki u Donalda	7:43	15.3	karpatka	8	zbyt mało fali
Chrupki Kąsek	9:39	16.3	jagodzianka	4	zbyt mało owoców
Ciastkarnia u Misia	8:56	21.3	napoleonka	9	prawie doskonała, bez pudru

Tabela zawierająca informacje o innych przekąskach, które nie są pączkami.

Tabela zawierająca informacje o pączkach z lukrem.



Tabele w zblizeniu

Kolumna to fragment informacji przechowywanej w tabeli.

Wiersz to jeden zbiór kolumn, opisujący atrybuty konkretnego obiektu. Wiersze i kolumny wspólnie składają się na tabelę.

Poniżej pokazano, jak mogłyby wyglądać tabela zawierająca dane z Twojego notesu z adresami. Często zamiast słowa **kolumna** można się spotkać ze słowem **pole**. W kontekście baz danych oba te terminy mają to samo znaczenie. Oprócz tego zamiennie są także stosowane terminy **wiersz** i **rekord**.

imie	nazwisko	adres	miejscowosc	wojewodztwo	id_num
------	----------	-------	-------------	-------------	--------

To są kolumny tabeli.

Jurek	Edamski	dane	dane	dane	dane
Alan	Konieczny	dane	dane	dane	dane
Maria	Milecka	dane	dane	dane	dane
Lola	Zielińska	dane	dane	dane	dane

A to są jej wiersze.

Połącz wszystkie wiersze, a uzyskasz tabelę.

imie	nazwisko	adres	miejscowosc	wojewodztwo	id_num
Jurek	Edamski	dane	dane	dane	dane
Alan	Konieczny	dane	dane	dane	dane
Maria	Milecka	dane	dane	dane	dane
Lola	Zielińska	dane	dane	dane	dane

Tworzenie własnych tabel



Czy zatem moje karteczki zawierają wystarczająco dużo informacji, by zapisać je w tabeli?

Oczywiście. Bez problemu można określić kategorie informacji o poszczególnych osobach.

Te kategorie staną się kolumnami tabeli. Z kolei każda karteczka zostanie przekształcona na jeden wiersz tabeli. Możesz zatem zapisać wszystkie informacje z karteczek do swojej nowej tabeli.

Kategorie określone na stronie 43:

Imię	Nazwisko	Data urodzenia	Zawód	Stan	Lokalizacja	E-mail	Zainteresowania	Poszukuje
Agnieszka	Mendralská	19.08.1979	administrator systemów Unix	zamężna	Sandomierz	agimedra@swiatkawy.com.pl	aktorstwo, taniec	nowej pracy

Teraz już wiesz, że kategorie są nazywane kolumnami.

Dane z jednej karteczki rozmieszczone w formie wiersza.

nazwisko	imie	email	data_urodzenia	zawod	lokalizacja	stan	zainteresowania	szuka
Barańska	Anna	ania@megadeski.com.pl	01.07.1962	inżynier lotniczy	Warszawa	panna, ale ma partnera	gry RPG, programowanie	nowej pracy
Homiński	Janusz	hojan@pizza-nzk.com.pl	10.09.1966	administrator systemów komputerowych	Darłówka	kawaler	wędrówki piesze, pisanie blogów	przyjaciół, kobiety na randki
Samotek	Adrian	asamotek@pizza-nzk.com.pl	02.12.1975	inżynier lotniczy	Warszawa	żonaty	gry RPG, programowanie	niczego
Mendralská	Agnieszka	agimedra@swiatkawy.com.pl	19.08.1979	administrator systemów Unix	Sandomierz	zamężna	aktorstwo, taniec	nowej pracy

Wiesz także, że informacje zapisane na każdej karteczkę można umieścić w jednym wierszu nazywanym rekordem.

No... nareszcie.
Powiedziecie w takim razie,
jak mam utworzyć moją tabelę?

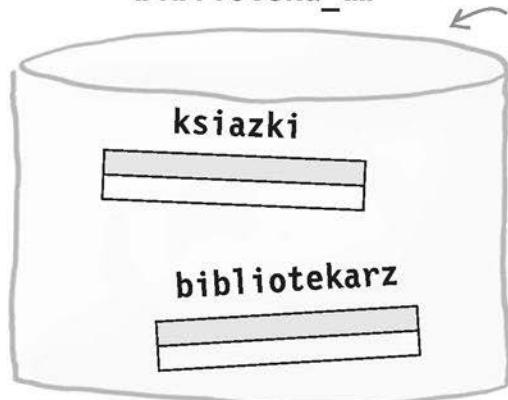




Ćwiczenie

Przyjrzyj się poniższym bazom danych i tabelom. Zastanów się, jakie kategorie informacji możesz znaleźć w każdej z nich. Podaj nazwy kolumn, jakie najprawdopodobniej mogłyby się w nich znaleźć.

biblioteka_db



Baza danych z informacjami o bibliotece i zgromadzonych w niej książkach.

ksiazki:

bibliotekarz:

klient_informacje:

konto:

sklepinternetowy_db

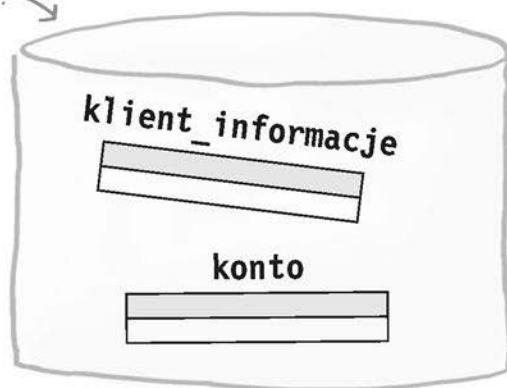


← Baza danych sklepu internetowego.

produkty:

koszyk:

bank_db



Rozwiążanie ćwiczenia



Rozwiążanie ćwiczenia

Przyjrzyj się poniższym bazom danych i tabelom. Zastanów się, jakie kategorie informacji możesz znaleźć w każdej z nich. Podaj nazwy kolumn, jakie najprawdopodobniej mogłyby się w nich znaleźć.

Nie przejmuj się, jeśli podane przez Ciebie nazwy kolumn nie są dokładnie takie same jak nasze.

biblioteka_db



Baza danych z informacjami o bibliotece i zgromadzonych w niej książkach.

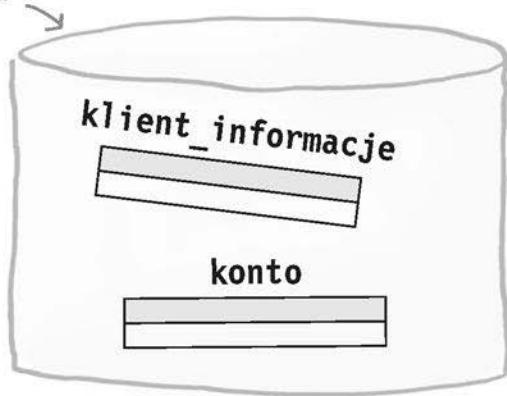
ksiazki: tytuł, autor, cena, kod.....

bibliotekarz: imię, nazwisko, adres.....

klient_informacje: imię, nazwisko, adres,
numer_konta, numer_ubezp

konto: stan, wpłaty, wypłaty.....

bank_db



Baza danych banku.

sklepinternetowy_db



Baza danych sklepu internetowego.

produkty: nazwa, wymiary, cena.....

koszyk: wartosc_sumaryczna, id_klienta.....

Przejmij kontrolę!

A teraz uruchom swój system zarządzania relacyjną bazą danych SQL (ang. relational database management system — RDBMS) i otwórz okno wiersza poleceń lub interfejs graficzny pozwalający na komunikację z systemem. Poniżej przedstawiliśmy okno wiersza poleceń pozwalające na zarządzanie serwerem MySQL.

```
D:\Programming\MySQL\MySQL Server 5.0\bin\mysql.exe
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 5.0.45-community-nt MySQL Community Edition <GPL>
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Ten zamykający nawias kątowy oznacza wiersz polecenia.
Za chwilę będziesz w nim wpisywać polecenia.

W pierwszej kolejności musisz utworzyć bazę danych, która będzie przechowywać tabele.

W SQL-u w nazwach baz danych i tabel nie można używać odstępów, dlatego zamiast nich został zastosowany znak podkreślenia.

- 1 Wpisz poniższy wiersz z poleceniami, by utworzyć bazę danych o nazwie `lista_grzesia`.

`CREATE DATABASE lista_grzesia;`

CREATE DATABASE jest polecienniem.

Nazwa bazy danych to `lista_grzesia`.

Wpisane polecenie musi się kończyć średnikiem.

```
D:\Programming\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> CREATE DATABASE lista_grzesia;
Query OK, 1 row affected (0.03 sec)
mysql>
```

Oto rezultaty zwrocone przez RDBMS. Oznacza to, że polecenie zostało wykonane prawidłowo.

Czy przeczytałeś wprowadzenie?

Uwaga!

Czy przeczytałeś wprowadzenie?

Do wykonywania poleceń i testowania baz danych używamy serwera bazy danych MySQL. Oznacza to, że polecenia, jakie będziesz musiał wydawać w używanym systemie zarządzania bazą danych, mogą wyglądać nieco inaczej. Informacje o sposobie instalowania serwera MySQL można znaleźć w dodatku B.

- 2 Teraz musisz przekazać systemowi obsługi baz danych, której bazy faktycznie chcesz używać.



Nie istniejąca grupa pytań

Q: Dlaczego muszę tworzyć bazę danych, skoro chcę korzystać tylko z jednej tabeli?

O: Język SQL wymaga, by wszystkie tabele były umieszczane w jakiejś bazie danych. Przemawiają za tym ważkie powody. Jedna z cech języka SQL jest możliwość kontrolowania dostępu do tabel przy wykorzystaniu użytkowników. Możliwość udzielenia lub zabronienia dostępu do całej bazy danych niejednokrotnie jest znacznie wygodniejsza niż określanie praw dostępu do wszystkich tabel umieszczonych w tej bazie.

Q: Zauważylem, że polecenie CREATE DATABASE zostało zapisane wielkimi literami. Czy taki sposób zapisu jest konieczny?

O: Niektóre systemy obsługi baz danych faktycznie wymagają, by pewne słowa kluczowe były zapisywane wielkimi literami. Jednak język SQL nie zwraca uwagi na wielkość znaków. Oznacza to, że polecenia nie muszą być zapisywane wielkimi literami, choć jest to uważałe za dobrą praktykę programistyczną. Przyjrzyjmy się zastosowanemu wcześniej poleceniu CREATE:

```
CREATE DATABASE lista_grzesia;  
Dzięki zastosowaniu wielkich liter już na pierwszy rzut oka możemy odróżnić wydane polecenie (CREATE DATABASE) oraz nazwę bazy danych (lista_grzesia).
```

Q: Czy jest coś, co powiniem wiedzieć o sposobie określania nazw baz danych, tabel i kolumn?

O: Na pewno warto, by nazwy te były opisowe. Czasami będzie to wymagało zastosowania więcej niż jednego słowa. Ponieważ w nazwach nie można używać odstępów, zatem jeśli składa się one z kilku wyrazów, wyrazy są przeważnie oddzielane od siebie znakami podkreślenia. Poniżej podaliśmy kilka przykładowych sposobów zapisu nazw, z którymi możesz się spotkać w praktyce:

lista_grzesia

listagrzesia

Listagrzesia

listaGrzesia

Ogólnie rzecz biorąc, lepiej unikać stosowania wielkich liter, aby uniknąć zamieszania. Język SQL nie rozpoznaje bowiem wielkości liter.

Q: A jeśli wolałbym użyć nazwy „listaGrzesia” i nie oddzielać słów znakiem podkreślenia?

O: Nic nie stoi na przeszkodzie, byś tak zrobił. Należy tylko pamiętać o tym, by zachować spójność i konsekwentnie używać jednego sposobu zapisu nazw. Jeśli wolisz nadać swojej bazie danych nazwę listaGrzesia, w której nie będzie znaków podkreślenia, a drugi wyraz zostanie zapisany z wielkiej litery, to tej samej konwencji powinieneś użyć do określania nazw wszystkich innych tabel w bazie danych, na przykład: mojeKontakty.

Teraz wszystkie operacje będą wykonywane na bazie danych lista_grzesia!

Q: A gdybym chciał nadać bazie danych nazwę zapisaną w języku angielskim, czy mógłbym ją nazwać na przykład: greg's_list?

O: Nie. Nazwa powinna mieć postać: gregs_list. W języku SQL apostrofy i cudzysłów są używane w innym celu. Choć istnieją sposoby pozwalające na ich zastosowanie w nazwach, to jednak znacznie łatwiej jest po prostu z nich zrezygnować.

Q: Zauważylem także, że na końcu polecenia CREATE DATABASE został umieszczony średnik. Czy był on konieczny, a jeśli tak, to do czego służy?

O: Owszem, średnik jest potrzebny — oznacza on zakończenie polecenia.

Zastosowanie wielkich liter oraz znaków podkreślenia pomaga w pisaniu kodu polecień SQL (choć sam język SQL ich nie wymaga!).

Obsługiwany z poziomu wiersza poleceń klient bazy danych, którego w tej książce używamy do prezentowania polecień SQL, nie pozwala na stosowanie polskich znaków diakrytycznych; dlatego też w prezentowanych polecenях oraz na rysunkach polskie litery nie są stosowane. Niedogodność tę można ominąć, stosując bardziej zaawansowane programy do obsługi bazy danych MySQL, choćby MySQL Administrator i MySQL Query Browser, które można pobrać z witryny <http://www.mysql.com/>.

Tworzenie tabeli: Polecenie CREATE TABLE

Zobaczmy, jak utworzyć tabelę na informacje dotyczące pączków. Założmy, że miałeś problem z określeniem rodzaju oferowanych pączków jedynie na podstawie nazwy ciastkarni, dlatego wpadłeś na pomysł, by ułatwić sobie życie, tworząc tabelę. Poniżej zamieściliśmy jedno polecenie, jakie musisz wpisać w oknie konsoli. Kiedy to zrobisz, naciśnij klawisz *Enter*; w ten sposób poinformujesz system obsługi bazy danych, że należy wykonać polecenie.

lista_paczekow

nazwa_paczka	typ_paczka
Jagodowiec	z nadzieniem
Pączek cynamonowy	pierścien
Gwiazda rocka	ciastko plecone
Karmelowiec	ciastko plecone
Jabłkowiec	z nadzieniem

Oto polecenie SQL, które pozwoli nam utworzyć tabelę — zwróć uwagę na wielkie litery.

Nawias otwierający rozpoczętu listę kolumn, jakie należy utworzyć.

CREATE TABLE lista_paczekow

Nazwa pierwszej kolumny tabeli.

nazwa_paczka VARCHAR(25)

Nazwa drugiej kolumny tabeli.

typ_paczka VARCHAR(20)

Nawias zamykający kończy listę kolumn.

Ten średnik informuje system obsługi baz danych, że dotarł do końca polecenia.

Zastosowana nazwa tabeli powinna być zapisana małymi literami, a zamiast odstępów należy użyć znaków podkreślenia.

Naciśnij klawisz *Enter*, by dalszą część polecenia wpisywać w nowym wierszu i dzięki temu poprawić jego przejrzystość.

Poszczególne kolumny są oddzielane do siebie przecinkami.

To jest TYP DANYCH. W tym przypadku jego nazwa — **VARCHAR** — (skrót od angielskich słów Variable Character — łańcuch znaków o zmiennej długości) informuje nas, że informacje w polu będą zapisywane jako tekst. Wyrażenie (20) umieszczone za nazwą kolumny oznacza, że długość zapisywanej w niej tekstu nie będzie mogła przekroczyć 20 znaków.



Hej! A co ze mną? Co z poleceniem
CREATE TABLE, które pozwoli utworzyć
moją tabelę lista—grzesia?

Tworzenie bardziej złożonych tabel

Czy pamiętasz kolumny, które miały się znaleźć w tabeli Grześka? Zapisaliśmy je na samoprzylepnej karteczkce. Teraz musimy ją znaleźć, by stworzyć odpowiednie polecenie CREATE TABLE.

Użyjesz polecenia CREATE TABLE,
by zmienić tę listę nazw kolumn na...

...tę tabelę.

nazwisko
imię
email
data_urodzenia
zawód
miejscowość
stan
zainteresowania
szuka

nazwisko	imie	email	data_urodzenia	zawod	lokalizacja	stan	zainteresowania	szuka



WYSIL SZARE KOMÓRKI

Pod jakimi względami nazwy z karteczki różnią się od nazw kolumn wynikowych tabeli? Dlaczego różnice te są ważne?

Przekonajmy się, jak łatwo można pisać kod SQL

Dowiedziałeś się już, że aby utworzyć tabelę, należy skategoryzować informacje i zorganizować je w postaci kolumn. Następnie dla każdej z kolumn należy określić odpowiedni typ danych oraz długość. Po oszacowaniu, jaką długość powinny mieć poszczególne kolumny, napisanie polecenia **CREATE TABLE** będzie już banalnie proste.



Zaostrz ołówek

Kod zamieszczony poniżej z lewej strony to polecenie CREATE TABLE służące do utworzenia tabeli w nowej bazie danych Grześka. Spróbuj odgadnąć przeznaczenie jego poszczególnych wierszy. Podaj także przykładowe dane, jakie mogą być zapisywane w poszczególnych kolumnach.

```
CREATE TABLE moje_kontakty
(
    nazwisko VARCHAR(30),
    imie VARCHAR(20),
    email VARCHAR(50),
    data_urodzenia DATE,
    zawod VARCHAR(50),
    lokalizacja VARCHAR(50),
    stan VARCHAR(20),
    zainteresowania VARCHAR(100),
    szuka VARCHAR(100)
);
```

Polecenie CREATE TABLE

Zaostrz ołówek Rozwiążanie

```
CREATE TABLE moje_kontakty
(
    nazwisko VARCHAR(30),
    imie VARCHAR(20),
    email VARCHAR(50),
    data_urodzenia DATE,
    zawod VARCHAR(50),
    lokalizacja VARCHAR(50),
    stan VARCHAR(20),
    zainteresowania VARCHAR(100),
    szuka VARCHAR(100)
);
```

Poniżej opisaliśmy przeznaczenie każdego z wierszy polecenia CREATE TABLE i podaliśmy przykładowe dane, jakie można zapisać w kolumnach wynikowej tabeli.

Tworzy tabelę o nazwie „moje_kontakty”.	
Rozpoczyna listę kolumn tabeli.	
Dodaje do tabeli kolumnę o nazwie „nazwisko”, w której będzie można zapisać do 30 znaków.	‘Kowalski’
Dodaje do tabeli kolumnę o nazwie „imie”, w której będzie można zapisać do 20 znaków.	‘Julian’
Dodaje do tabeli kolumnę o nazwie „email”, w której będzie można zapisać do 50 znaków.	‘j.kowalski@pizza-nzk.com.pl’
Dodaje do tabeli kolumnę o nazwie „data_urodzenia”, w której będzie można zapisać wartość określającą datę.	‘05.09.1980’
Dodaje do tabeli kolumnę o nazwie „zawod”, w której będzie można zapisać do 50 znaków.	‘Dziennikarz’
Dodaje do tabeli kolumnę o nazwie „lokalizacja”, w której będzie można zapisać do 50 znaków.	‘Wrocław, DS’
Dodaje do tabeli kolumnę o nazwie „stan”, w której będzie można zapisać do 20 znaków.	‘Kawaler’
Dodaje do tabeli kolumnę o nazwie „zainteresowania”, w której będzie można zapisać do 100 znaków.	‘Kajakarstwo, gady’
Dodaje do tabeli kolumnę o nazwie „szuka”, w której będzie można zapisać do 100 znaków.	‘Kontaktów międzyludzkich, przyjaciół’
Kończy listę kolumn tabeli, a średnik dodatkowo kończy całe polecenie SQL.	

Utwórzmy w końcu tabelę moje_kontakty

Teraz już dokładnie wiesz, jakie przeznaczenie ma każdy wiersz przedstawionego powyżej polecenia **CREATE TABLE**. Możesz je wpisać w oknie konsoli, kopując i wklejając poszczególne wiersze.

Równie dobrze możesz wpisać całe polecenie w jednym, bardzo długim wierszu:

```
CREATE TABLE moje_kontakty (nazwisko VARCHAR(30), imie VARCHAR(20), email VARCHAR(50), data_urodzenia DATE, zawod VARCHAR(50), lokalizacja VARCHAR(50), stan VARCHAR(20), zainteresowania VARCHAR(100), szuka VARCHAR(100) );
```

Niezależnie od tego, jaki sposób wybierzesz, nim naciśniesz klawisz **Enter**, upewnij się, że przepisałeś je dokładnie w podanej postaci — co do jednego znaku. **data_urodzenia DATE** to nie to samo co **dataurodzenia DATA**, podobnie jak **VARCHAR(3)** to nie to samo co **VARCHAR(30)**.

Możesz nam wierzyć — to jest nasze polecenie SQL, zapisaliśmy je jednak b-a-r-d-z-o małą czcionką, by zmieścić się w jednym wierszu na stronie.

Twoja tabela jest gotowa

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> CREATE TABLE moje_kontakty
-> (
->     nazwisko VARCHAR<30>,
->     imie VARCHAR<20>,
->     email VARCHAR<50>,
->     data_urodzenia DATE,
->     zawod VARCHAR<50>,
->     lokalizacja VARCHAR<50>,
->     stan VARCHAR<20>,
->     zainteresowania VARCHAR<100>,
->     szuka VARCHAR<100>
-> );
Query OK, 0 rows affected (0.27 sec)

mysql> _
```

Czy zauważysz, że naciśnięcie klawisza Enter, gdy cursor był umieszczony za średnikiem, spowodowało zakończenie polecenia i to, że system zarządzania bazą danych je wykonał?

A zatem wszystkie informacje będę zawsze przechowywać w kolumnach typu VARCHAR lub DATE?



Nie. W praktyce będą Ci jeszcze potrzebne inne typy danych, umożliwiające przechowywanie informacji innych rodzajów, takich jak liczby.

Załóżmy, że do tabeli z informacjami o pączkach dodaliśmy kolumnę przeznaczoną do przechowywania ceny. Jest raczej mało prawdopodobne, byśmy chcieli przechowywać tę cenę w kolumnie typu VARCHAR. Informacje zapisywane w kolumnach tego typu są interpretowane jako tekst, a zatem nie moglibyśmy wykonywać na nich żadnych operacji matematycznych. Na szczęście dostępnych jest więcej typów danych, z którymi się jeszcze nie spotkałeś...



Zanim zajmiemy się kolejnymi zagadnieniami, spróbuj określić, jakie inne typy danych, oprócz VARCHAR i DATE, mogą być nam potrzebne.

Spotkajmy się z niektórymi typami danych SQL

Oto kilka najbardziej popularnych typów danych. To właśnie do ich obowiązków należy zapisywanie Twoich informacji w bazie danych w taki sposób, by nic się z nimi nie stało. Poznałeś już dwa z nich: VARCHAR i DATE; nadszedł czas, byś przywitał się także z pozostałymi.



Uwaga!

Podane powyżej nazwy typów danych mogą nie działać w używanym przez Ciebie systemie zarządzania bazami danych!

Niestety nie ma żadnych ogólnie zaakceptowanych nazw typów danych. Dlatego też w systemie, którego będziesz używać, jedna lub kilka z powyższych nazw może mieć inną postać. Aby upewnić się, jakie nazwy typów danych są faktycznie dostępne, powinieneś zatrzymać się do dokumentacji bazy.

JAKIEGO TYPU DANYCH UŻYĆ?

Określ, jaki typ danych będzie się najlepiej nadawał dla poszczególnych kolumn. A skoro zajmiesz się poniższą tabelą, to uzupełnij także pozostałe brakujące komórki.

Te dwie cyfry określają, ilu cyfr baza danych będzie oczekwać przed miejscem dziesiątnym, a ilu po nim.

Nazwa kolumny	Opis	Przykład	Optymalny typ danych
cena	Wartość wszystkich zamówionych towarów.	5678,39	DEC(5,2)
kod_pocztowy			
waga_atomowa	Waga atomowa pierwiastka z dokładnością do sześciu miejsc po przecinku.		
komentarze	Duży fragment tekstu, o długości większej niż 255 znaków.	Jarku, jestem na spotkaniu akcjonariuszy. Właśnie pokazałem demo programu — na ekranie latało mnóstwo małutkich kaczuszek. Czy to miał być jakiś żart? Może jest ci potrzebny urlop, bo poczucie humoru świadczy o tym, że jesteś przemęczony.	
ilosc	Ilość produktów dostępnych w magazynie.		
stawka_podatku		3,755	
tytul_ksiazki		SQL. Rusz głową!	
plec	Jedna litera — M lub K.		CHAR(1)
numer_telefonu	Dziesięć cyfr bez żadnych znaków przestankowych.	0181519523	
wojewodztwo	Dwuliterowy skrót nazwy województwa.	SL, KP	
rocznica		22.11.2006	DATE
ilosc_wygranych_gier			INT
czas_spotkania		10:30 12.4.2020	

Nie istniejąca grupa pytań

P: A czy nie można by zapisywać wszystkich danych tekstowy w kolumnie typu BLOB?

O: Można, ale byłoby to niepotrzebne marnowanie miejsca. Kolumny typu VARCHAR lub CHAR zajmują konkretną ilość miejsca, nie przekraczając przy tym długości 256 znaków. Z kolei BLOB zajmuje znacznie więcej przestrzeni. Wraz z powiększaniem się bazy danych rośnie także ryzyko, że zabraknie wolnego miejsca na dysku. Co więcej, na kolumnach typu BLOB nie można wykonywać pewnych istotnych operacji na łańcuchach znaków, które bez przeszkode

można stosować w kolumnach typu VARCHAR oraz CHAR (poznasz je w dalszej części książki).

P: Do czego będą mi potrzebne liczbowe typy danych, takie jak INT lub DEC?

O: Wszystko sprawdza się do zagadnień związanych z efektywnością działania baz danych oraz używanymi przez nie mechanizmami przechowywania informacji. Wybór optymalnego typu danych dla każdej z kolumn w bazie danych pomoże ograniczyć wielkość tabel i sprawić, że wszelkie operacje na nich będą wykonywane szybciej.

P: I to wszystko? Czy to wszystkie typy danych dostępne w języku SQL?

O: Nie, jednak są to typy najczęściej używane. Co więcej, różne systemy zarządzania bazami danych udostępniają nieco odmienne zestawy typów danych, zatem szczegółowych informacji należy szukać w dokumentacji używanej bazy danych. Polecamy książkę *MySQL. Almanach* (wydaną przez Wydawnictwo Helion), w której można znaleźć informacje o różnicach pomiędzy dostępnymi systemami obsługi relacyjnych baz danych.

JAKIEGO TYPU DANYCH UŻYĆ?

ROZWIĄZANIE

Określ, jaki typ danych będzie się najlepiej nadawał dla poszczególnych kolumn. A skoro zajmiesz się poniższą tabelą, to uzupełnij także pozostałe brakujące komórki.

Kody pocztowe w różnych krajach mogą mieć różną postać i długość, aby wiec zaoszczędzić miejsca w bazie danych, zastosowaliśmy typ VARCHAR. Gdyby zapisywane w tabeli kody pocztowe miały tę samą, ustaloną długość, moglibyśmy zastosować typ CHAR.

Nazwa kolumny	Opis	Przykład	Optymalny typ danych
cena	Wartość wszystkich zamówionych towarów.	5678,39	DEC(5,2)
kod_pocztowy	W zależności od kraju: od czterech do dziesięciu znaków.	44-105	VARCHAR(10)
waga_atomowa	Waga atomowa pierwiastka z dokładnością do sześciu miejsc po przecinku.	4,002602	DEC(10,6)
komentarze	Duży fragment tekstu, o długości większej niż 255 znaków.	Jarku, jestem na spotkaniu akcjonariuszy. Właśnie pokazali demo programu — na ekranie latało mnóstwo małutkich kaczuszek. Czy to miał być jakiś żart? Może jest ci potrzebny urlop, bo poczucie humoru świadczy o tym, że jesteś przemęczony.	BLOB
ilosc	Ilość produktów dostępnych w magazynie.	239	INT
stawka_podatku	Wartość procentowa	3,755	DEC(4,2)
tytul_ksiazki	Łańcuch znaków	SQL. Rusz głową!	VARCHAR(50)
plec	Jedna litera — M lub K.	M	CHAR(1)
numer_telefonu	Dziesięć cyfr bez żadnych znaków przestankowych.	0181519523	CHAR(10)
wojewodztwo	Dwuliterowy skrót nazwy województwa.	SL, KP	CHAR(2)
rocznica	Dzień, miesiąc i rok	22.11.2006	DATE
ilosc_wygranych_gier	Liczba całkowita określająca ilość wygranych gier	15	INT
czas_spotkania	Godzina i data	10:30 12.4.2020	DATETIME

Do zapisywania w bazie danych bieżącej daty i godziny zazwyczaj używany jest typ TIMESTAMP. Typ DATETIME znacznie lepiej nadaje się do zapisywania przyszłych dat.



CELNE SPOSTRZEŻENIA

- ◆ Zanim utworzysz tabelę, podziel dane na kategorie. Zwróć szczególną uwagę na typy danych, jakie będą zapisywane w poszczególnych kolumnach.
- ◆ Użyj polecenia CREATE DATABASE, by utworzyć bazę danych, w której następnie utworzysz swoje tabele.
- ◆ Użyj polecenia USE DATABASE, by wskazać, na jakiej bazie danych chcesz pracować — to właśnie w niej później utworzysz tabele.
- ◆ Wszystkie tabele są tworzone przy użyciu polecenia CREATE TABLE, w którym umieszczana jest lista nazw kolumn oraz ich typów danych.
- ◆ Kilka spośród najpopularniejszych typów danych to: CHAR, VARCHAR, BLOB, INT, DATE oraz DATETIME. Każdy z tych typów kolumn ma odrębne reguły określające, co można w nim zapisywać.



Zaraz, chwileczkę. A gdzie się podzieliła tabela, którą przed chwilą utworzyłam w bazie danych lista—grzesia? Chciałabym sprawdzić, czy zastosowałam w niej odpowiednie typy kolumn.

Bardzo słusznie. Sprawdzanie, czy nie popełniliśmy błędu, jest bardzo ważne.

By sprawdzić, jak wygląda utworzona wcześniej tabela moje_kontakty, można posłużyć się przedstawionym poniżej poleceniem DESC:

→ **DESC moje_kontakty;**

DESC to skrót od angielskiego słowa DESCRIBE — opisywać.

Przekonaj się, jak działa to polecenie.

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> DESC moje_kontakty;
```

Twoja tabela bez tajemnic

Kiedy wpiszesz i wykonasz polecenie DESC, w oknie konsoli zostaną wyświetcone informacje przypominające te przedstawione poniżej:

Tymi informacjami na razie nie musisz się przejmować, już niebawem zajmiemy się nimi szczegółowo.

C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> DESC moje_kontakty;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
nazwisko	varchar(30)	YES		NULL	
imie	varchar(20)	YES		NULL	
email	varchar(50)	YES		NULL	
data_urodzenia	date	YES		NULL	
zawod	varchar(50)	YES		NULL	
lokalizacja	varchar(50)	YES		NULL	
stan	varchar(20)	YES		NULL	
zainteresowania	varchar(100)	YES		NULL	
szuka	varchar(100)	YES		NULL	
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.14 sec)
mysql>



WYSIL
SZARE KOMÓRKI

A co Ty sądzisz? Jakich problemów mogłoby przysporzyć dodanie nowej kolumny do tabeli?



- Magnesiki z SQL-em -

Kod służący do utworzenia bazy danych i nowej tabeli z kolumną określającą płeć osoby został zapisany na magnesikach przyczepionych do łódówki. Czy potrafisz ułożyć magnesiki w odpowiedniej kolejności i odtworzyć prawidłowe polecenia SQL? Kilka magnesików z nawiasami, przecinkami i średnikami spadło z łódówki na podłogę, jednak są one zbyt małe, by je podnieść, więc nie zaprzataj sobie nimi głowy i dodaj odpowiednie znaki tam, gdzie będą potrzebne.

```
email VARCHAR(50)
data_urodzenia DATE
USE lista_grzesia

nazwisko VARCHAR(30)
imie VARCHAR(20)

zainteresowania VARCHAR(100)
szuka VARCHAR(100)

stan VARCHAR(20)

CREATE DATABASE lista_grzesia

zawod VARCHAR(50)
lokalizacja VARCHAR(50)

CREATE TABLE moje_kontakty

plec CHAR(1)
```

Kiedy skończysz, spróbuj wpisać nowe polecenie CREATE TABLE w konsoli SQL i stworzyć nową tabelę zawierającą kolumnę z informacją o płci.

Tabela nie można tworzyć wielokrotnie



- Magnesiki z SQL-em. —

Twoim zadaniem było rozmieszczenie magnesików z kodem SQL w odpowiedniej kolejności, tak by powstały prawidłowe polecenia tworzące bazę danych i nową tabelę kontaktów z kolumną zawierającą informacje o płci.

Rozwiązanie

Baza danych
lista_grzesia
już istnieje.

Nie można ponownie stworzyć już istniejącej bazy danych lub tabeli!

Czy spróbujesz wpisać w konsoli i wykonać to nowe polecenie CREATE TABLE? Jeśli to zrobisz, to już wiesz, że rozwiązanie tego ćwiczenia nie pozwoli nam dodać do tabeli nowej kolumny.

Jeśli spróbujesz wpisać kod polecenia SQL w konsoli i wykonać go, to zapewne uzyskasz wyniki podobne do tych przedstawionych poniżej:

```
CREATE DATABASE lista_grzesia
USE lista_grzesia
CREATE TABLE moje_kontakty
(
    nazwisko VARCHAR(30),
    imie VARCHAR(20),
    email VARCHAR(50),
    plec CHAR(1),
    data_urodzenia DATE,
    zawod VARCHAR(50),
    lokalizacja VARCHAR(50),
    stan VARCHAR(20),
    zainteresowania VARCHAR(100),
    szuka VARCHAR(100)
);
```

Nowa kolumna
przeznaczona na
informacje o płci.

O rany! Wykonanie
polecenia powoduje
wyświetlenie
komunikatu o błędzie.
Wygląda na to, że
nowa tabela nie
została utworzona.

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> CREATE TABLE moje_kontakty
-> (
->     nazwisko VARCHAR(30),
->     imie VARCHAR(20),
->     email VARCHAR(50),
->     plec CHAR(1),
->     data_urodzenia DATE,
->     zawod VARCHAR(50),
->     lokalizacja VARCHAR(50),
->     stan VARCHAR(20),
->     zainteresowania VARCHAR(100),
->     szuka VARCHAR(100)
-> );
ERROR 1050 (42S01): Table 'moje_kontakty' already exists
mysql> -
```

Nieistniejąca grupa pytań

P: Co się stało w ostatnim ćwiczeniu „Magnesiki z SQL-em” — dlaczego pojawił się błąd?

O: Nie można utworzyć tabeli, która już istnieje. Poza tym kiedy już raz utworzyłeś bazę danych, nie ma potrzeby tworzenia jej kolejny raz. Inny błąd, jaki mogłeś popełnić, to pominięcie któregoś z przecinków. Sprawdź także, czy prawidłowo zapisałeś wszystkie słowa kluczowe SQL.

P: A dlaczego po kolumnie szuka VARCHAR(100) nie ma przecinka, jak po wszystkich pozostałych kolumnach?

O: Kolumna szuka jest ostatnią kolumną w tabeli, umieszczoną tuż przed nawiasem zamykającym. Nawias informuje o zakończeniu polecenia SQL i dlatego przecinek nie jest już potrzebny.

P: A zatem, czy istnieje jakiś sposób na dodanie kolumny, czy będę musiał wszystko zacząć od początku?

O: Niestety musisz zacząć od początku, jednak zanim będziesz mógł utworzyć nową tabelę, musisz pozbyć się tej już istniejącej. Ponieważ w tabeli nie ma jeszcze żadnych danych, możemy po prostu usunąć starą tabelę i utworzyć nową.

P: A co można by zrobić, gdyby w tabeli, do której chciałbym dodać kolumnę, były już jakieś dane? Czy można by to zrobić bez konieczności usuwania całej tabeli wraz z jej zawartością?

O: Doskonale pytanie! Owszem, istnieje sposób na modyfikowanie tabeli bez usuwania zapisanych w niej informacji. Opiszymy go nieco później, a na razie, ponieważ tabela jest pusta, usuniemy starą tabelę i utworzymy nową.



To doskonały pomysł, z którego na pewno warto korzystać podczas lektury tej książki.

Dzięki temu będziesz mógł zaznaczać polecenia SQL w edytorze i wklejać je do konsoli za każdym razem, gdy będzie to konieczne. W ten sposób unikniesz konieczności ciągłego wpisywania tych samych poleceń SQL. Co więcej, możesz także kopiować stare polecenia i używać ich jako bazy do tworzenia nowych.

Do kosza ze starej tabelą — czas na nową

- 1 Usunięcie tabeli jest znacznie prostsze niż jej utworzenie.
Wystarczy użyć poniższego polecenia SQL:

To jest polecenie służące do usuwania tabeli...

... a to nazwa tabeli, którą chcemy usunąć.

DROP TABLE moje_kontakty;

Nie zapomnij o średniku na końcu.

```
D:\D:\Programming\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> DROP TABLE moje_kontakty;
Query OK, 0 rows affected (0.04 sec)

mysql> -
```

Polecenie **DROP TABLE usunie tabelę niezależnie od tego, czy będą w niej zapisane jakieś dane, czy nie. Dlatego należy go używać z wielką ostrożnością. Po usunięciu tabeli zostaje ona utracona raz na zawsze wraz ze wszystkimi danymi, jakie były w niej zapisane.**

Polecenie **DROP TABLE usuwa tabelę oraz wszystkie zapisane w niej informacje.**

- 2 Teraz możesz już wykonać nasze nowe polecenie **CREATE TABLE**:

```
C:\C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> CREATE TABLE moje_kontakty
-> (
->   nazwisko VARCHAR(30),
->   imie VARCHAR(20),
->   email VARCHAR(50),
->   plec CHAR(1),
->   data_urodzenia DATE,
->   zawod VARCHAR(50),
->   lokalizacja VARCHAR(50),
->   stan VARCHAR(20),
->   zainteresowania VARCHAR(100),
->   szuka VARCHAR(100)
-> );
Query OK, 0 rows affected (0.33 sec)

mysql> -
```

Tym razem polecenie zostało wykonane prawidłowo.

Kim jestem?



Grupa poprzebieranych w kostiumy słów kluczowych i typów danych SQL bawi się w grę towarzyską o nazwie „Zgadnij, kim jestem”. Jej uczestnicy udzielają Ci podpowiedzi, a Ty próbujesz na ich podstawie odgadnąć, kim oni są. Załóż, że uczestnicy gry zawsze mówią prawdę. Jeśli zdarzy się, że udzielona podpowiedź odnosi się do kilku uczestników gry, to zapisz nazwy ich wszystkich. Nazwy uczestników gry zapisz w pustych miejscach widocznych z prawej strony podpowiedzi.

W dzisiejszej zabawie udział biorą:

CREATE DATABASE, USE DATABASE, CREATE TABLE, DESC, DROP TABLE, CHAR, VARCHAR, BLOB, DATE, DATETIME, DEC oraz INT.

Uczestnicy

Mam Twój numer.

.....

Pomogę Ci pozbyć się niepożądanych tabel.

.....

Najbardziej lubię pytania, na które można odpowiedzieć „Tak” lub „Nie”.

.....

Zapamiętam datę urodzin Twojej mamy.

.....

Ja trzymam w garści wszystkie tabele.

.....

Liczby są super, ale nie cierpię ułamków.

.....

Uwielbiam długie, opisowe wyjaśnienia.

.....

To jest miejsce, w którym wszystko można przechować.

.....

Beze mnie tabela w ogóle by nie istniała.

.....

Doskonale wiem, kiedy w przyszłym tygodniu jesteś umówiony u dentysty.

.....

Jestem ulubieńcem księgowych.

.....

Dzięki mnie poznasz format tabeli.

.....

Bez nas nie byłbyś w stanie utworzyć tabeli.

.....

→ Odpowiedzi znajdziesz na stronie 85

Anatomia bazy danych



No dobrze, poradziłem sobie z utworzeniem nowej wersji tabeli.
Ale w jaki sposób mogę teraz przenieść dane z karteczek do tabeli?

Aby dodać dane do tabeli, będziesz musiał skorzystać z polecenia INSERT

Słowo „insert” w języku angielskim oznacza „wstawać”, co doskonale oddaje działanie i przeznaczenie polecenia INSERT. Przeanalizuj przedstawiony poniżej przykład tego polecenia, by zobaczyć, jak działają jego poszczególne fragmenty. Wartości umieszczone wewnętrz drugiej pary nawiasów muszą być zapisane *w dokładnie takiej samej kolejności, w jakiej zostały podane nazwy kolumn.*

Poniższe polecenie nie nadaje się do wykonania w takiej postaci, w jakiej zostało podane — to jedynie przykład, który ma pokazać ogólną postać polecień INSERT.

Polecenie rozpoczyna się słowami kluczowymi
INSERT INTO.

To jest nazwa tabeli.
W przypadku bazy
danych Grześka jest to
tabela moje_kontakty.

Kolejną częścią polecenia są
nazwy kolumn tabeli. Już wiemy,
że tabela Grześka będzie mieć
takie kolumny jak: nazwisko, imię,
data_urodzenia oraz email.

Tutaj można podać
więcej kolumn, przy
czym za ostatnią nie
należy umieszczać
przecinka.

INSERT INTO nazwa_tabeli (nazwa_kolumny1 , nazwa_kolumny2 , ...)

Kolejne słowo kluczowe
— informuje, że za nim
zostanie podana lista
wartości kolumn.

VALUES ('wartosc1' , 'wartosc2' , ...);

Kolejną część
polecenia INSERT
stanowi lista wartości
kolumn, oddzielonych
od siebie przecinkami.
W przypadku tabeli
Grześka zostaną
w niej zapisane dane
z karteczek.

Można tu zastosować
apostrofy. Trzeba ich
używać za każdym razem,
gdy chcemy zapisać
w tabeli jakiś tekst, nawet
jeśli będzie to pojedynczy
znak, taki jak „M” lub „X”.

Tu zapewne będzie
więcej wartości,
oddzielonych od
siebie przecinkami.
Przy czym po
ostatniej nie należy
wstawać przecinka.

Standardowy
średnik kończący
wszystkie
polecenia SQL.

WAŻNE: wartości muszą być zapisane
dokładnie w takiej samej kolejności jak
nazwy kolumn.

KTO CO ROBI?

Zanim napiszesz polecenie SQL, musisz dopasować nazwy kolumn z odpowiednimi wartościami.

Kolumny

Wartości

imie

'Związków, przyjaciół'

stan

'Kowalska'

szuka

'1980-09-05'

plec

'Autorka tekstów technicznych'

data_urodzenia

'Juliana'

nazwisko

'Panna'

lokalizacja

'K'

zainteresowania

'Warszawa, MZ'

zawod

'ju.ka@pizza-nzk.com.pl'

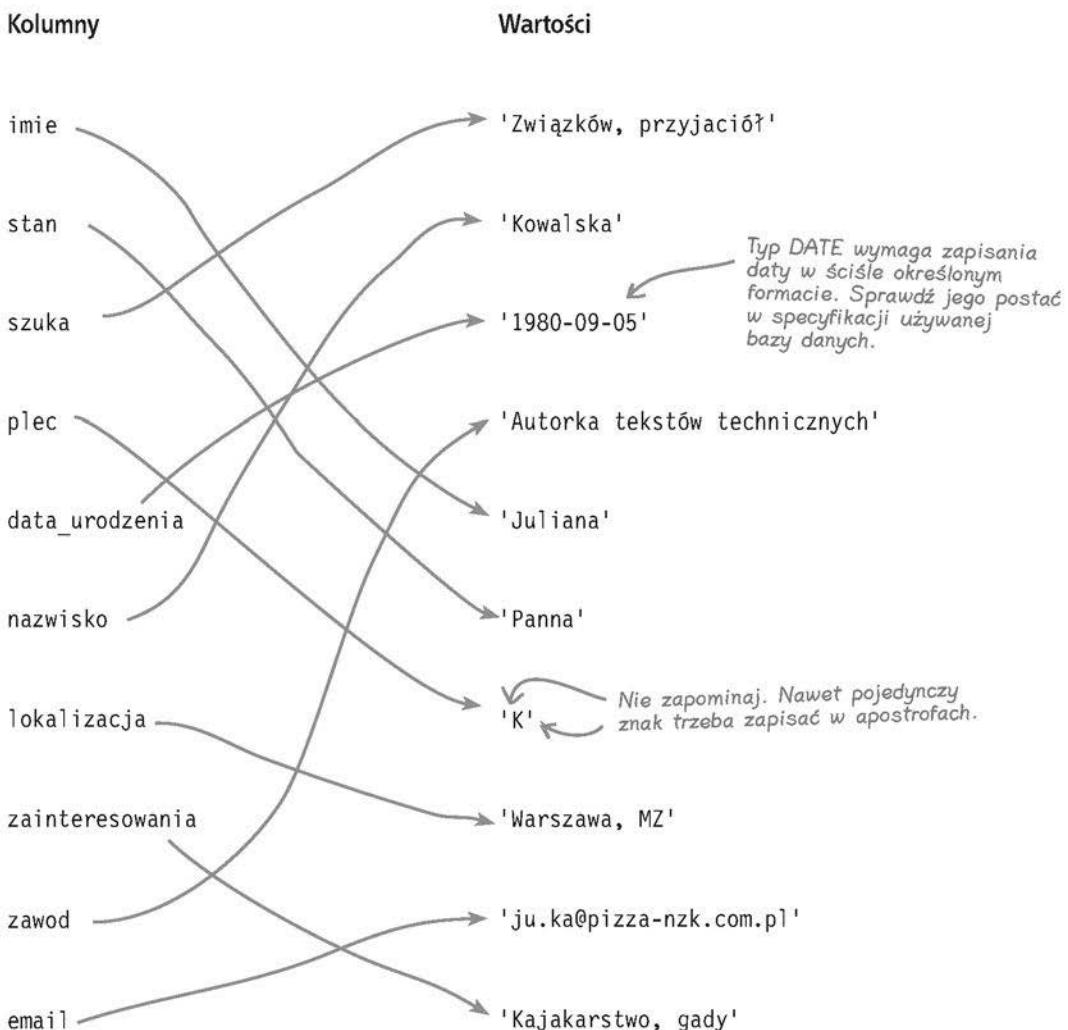
email

'Kajakarstwo, gady'

KTO CO ROBI?

ROZWIĄZANIE

Zanim napiszesz polecenie SQL, musisz dopasować nazwy kolumn z odpowiednimi wartościami.



Utworzenie polecenia INSERT

Nazwy kolumn, oddzielone od siebie przecinkami, zostały zapisane wewnątrz pierwszej pary nawiasów.

Przed wpisaniem pierwszej pary nawiasów wraz z zawartością możesz nacisnąć klawisz Enter, by ułatwić odczytanie kodu polecenia w oknie konsoli.

INSERT INTO moje_kontakty

(nazwisko, imie, email, plec, data_urodzenia, zawod,
lokalizacja, stan, zainteresowania, szuka)

VALUES

Naciśnij klawisz Enter także po zakończeniu pierwszej pary nawiasów oraz po wpisaniu słowa kluczowego **VALUES**.
To także ułatwi analizę kodu.

('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl',
'K', '1980-09-05', 'Autorka tekstów technicznych',
'Warszawa, MZ', 'Panna', 'Kajakarstwo, gady',
'Związków, przyjaciół');

Wartości poszczególnych kolumn zostały zapisane wewnątrz drugiej pary nawiasów. Także i one są oddzielone od siebie przecinkami.

Wszystkie wartości zapisywane w kolumnach typów: VARCHAR, CHAR, DATE oraz BLOB zostały zapisane w apostrofach.



Kolejność ma znaczenie!

Wartości powinny być podane w dokładnie takiej samej kolejności, w jakiej wcześniej podano nazwy kolumn.



Ćwiczenie

Wypróbuj to w domu

Powyżej przedstawiliśmy jeden ze sposobów pozwalających na dodanie wiersza do tabeli. Spróbuj wpisać powyższe polecenie **INSERT**. Najpierw wpisz je w edytorze tekstów. Dzięki temu, jeśli popełnisz jakiś błąd, nie będziesz musiał wpisywać wszystkiego od początku. Zwróć szczególną uwagę na apostrofy i przecinki. Poniżej zapisz komunikat, jaki został wyświetlony po wykonaniu polecenia:



Przed chwilą powiedzieliście, że w poleceniu INSERT wartości typów CHAR, VARCHAR, DATE i BLOB są zapisywane w apostrofach. Czy to oznacza, że wartości liczbowe, umieszczone w takich kolumnach jak: DEC lub INT, nie są zapisywane w apostrofach?

Dokładnie tak.

Poniżej przedstawione zostało polecenie INSERT, którego mógłbyś użyć do zapisania danych w tabeli gromadzącej dane o zakupionych ciastkach. Zwróć uwagę, że w części zawierającej wartości liczba tuzinów zamówionych paczek oraz ich cena nie zostały zapisane w apostrofach.

Kolumna tuziny jest typu INT, gdyż zazwyczaj nie kupuje się potowę tuzina paczek, dlatego wartości zapisywane w tej kolumnie nie muszą mieć części ułamkowej.

Kolumna cena została zdefiniowana jako DEC(4,2), co oznacza, że występuwać mogą cztery cyfry przed przecinkiem i dwie na miejscach dziesiętnych.

```
INSERT INTO zamowione_paczki  
(rodzaj_paczka, tuziny, polewa, cena)  
VALUES  
('z nadzieniem', 3, 'lukier', 3.50)
```

Wartości zapisywane w kolumnach tuziny oraz cena nie muszą być zapisane w apostrofach.

Zaostrz ołówek



Używany system zarządzania relacyjnymi bazami danych zawsze poinformuje Cię, jeśli zapytanie, które chcesz wykonać, będzie nieprawidłowe; jednak zwracane przez niego informacje czasami mogą być mało konkretne. Przyjrzyj się przedstawionym poniżej poleceniom `INSERT`. W pierwszej kolejności spróbuj samemu określić, co jest nie tak z każdym z nich, a następnie wpisz je w konsoli i przekonaj się, co na ich temat powie system zarządzania bazami danych.

```
INSERT INTO moje_kontakty
(nazwisko, imie, email, plec, data_urodzenia, zawod, lokalizacja, stan, zainteresowania, szuka)
VALUES ('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl', 'K', '1980-09-05', 'Autorka tekstów
technicznych', 'Panna', 'Kajakarstwo, gady', 'Związków, przyjaciół');
```

Co jest nie w porządku?

Komunikat wyświetlony przez RDBMS:

```
INSERT INTO moje_kontakty
(nazwisko, imie, plec, data_urodzenia, zawod, lokalizacja, stan, zainteresowania, szuka)
VALUES ('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl', 'K', '1980-09-05', 'Autorka tekstów
technicznych', 'Warszawa, MZ', 'Panna', 'Kajakarstwo, gady', 'Związków, przyjaciół');
```

Co jest nie w porządku?

Komunikat wyświetlony przez RDBMS:

```
INSERT INTO moje_kontakty
(nazwisko, imie, email, plec, data_urodzenia, zawod, lokalizacja, stan, zainteresowania, szuka)
VALUES ('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl', 'K', '1980-09-05', 'Autorka tekstów
technicznych' 'Warszawa, MZ', 'Panna', 'Kajakarstwo, gady', 'Związków, przyjaciół');
```

Co jest nie w porządku?

Komunikat wyświetlony przez RDBMS:

```
INSERT INTO moje_kontakty
(nazwisko, imie, email, plec, data_urodzenia, zawod, lokalizacja, stan, zainteresowania, szuka)
VALUES ('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl', 'K', '1980-09-05', 'Autorka tekstów
technicznych', 'Warszawa, MZ', 'Panna', 'Kajakarstwo, gady', 'Związków, przyjaciół');
```

Co jest nie w porządku?

Komunikat wyświetlony przez RDBMS:

Jeśli to polecenie spowoduje, że system zarządzania bazą danych przestanie odpowiadać, to spróbuj zakończyć je, wpisując znak apostrofu oraz średnik.

Zaostrz ołówek



Rozwiązywanie

Używany system zarządzania relacyjnymi bazami danych zawsze poinformuje Cię, jeśli zapytanie, które chcesz wykonać, będzie nieprawidłowe; jednak zwracane przez niego informacje czasami mogą być mało konkretne. Przyjrzyj się przedstawionym poniżej polecaniom INSERT. W pierwszej kolejności spróbuj określić, co jest nie tak z każdym z nich, a następnie wpisz je w konsoli i przekonaj się, co na ich temat powie system zarządzania bazami danych.

```
INSERT INTO moje_kontakty
```

```
(nazwisko, imie, email, plec, data_urodzenia, zawod, lokalizacja, stan, zainteresowania, szuka) VALUES  
('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl', 'K', '1980-09-05', 'Autorka tekstów technicznych',  
'Panna', 'Kajakarstwo, gady', 'Związków, przyjaciół');
```

Co jest nie w porządku? **Nie określono wartości kolumny "lokalizacja".**

Na liście kolumn umieściliśmy kolumnę „lokalizacja”, jednak w dalszej części zapytania — na liście wartości — pominieliśmy wartości tej kolumny.

Komunikat wyświetlony przez RDBMS: **ERROR 1136 (21S01): Column count doesn't match value count at row 1.**

```
INSERT INTO moje_kontakty
```

```
(nazwisko, imie, plec, data_urodzenia, zawod, lokalizacja, stan, zainteresowania, szuka)  
VALUES ('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl', 'K', '1980-09-05', 'Autorka tekstów  
technicznych', 'Warszawa', 'Panna', 'Kajakarstwo, gady', 'Związków, przyjaciół');
```

Co jest nie w porządku? **Brak kolumny dla adresu poczty elektronicznej na liście kolumn.**

Tym razem mamy już wartości dla wszystkich kolumn tabeli, jednak na liście kolumn pominieliśmy nazwę tej kolumny.

Komunikat wyświetlony przez RDBMS: **ERROR 1136 (21S01): Column count doesn't match value count at row 1.**

```
INSERT INTO moje_kontakty
```

```
(nazwisko, imie, email, plec, data_urodzenia, zawod, lokalizacja, stan, zainteresowania, szuka)  
VALUES ('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl', 'K', '1980-09-05', 'Autorka tekstów  
technicznych' 'Warszawa, MZ', 'Panna', 'Kajakarstwo, gady', 'Związków, przyjaciół');
```

Co jest nie w porządku? **Brak przecinka między dwiema wartościami.**

Brak przecinka pomiędzy wartościami 'Autorka tekstów technicznych' oraz 'Warszawa, MZ'.

Komunikat wyświetlony przez RDBMS: **ERROR 1136 (21S01): Column count doesn't match value count at row 1.**

```
INSERT INTO moje_kontakty
```

```
(nazwisko, imie, email, plec, data_urodzenia, zawod, lokalizacja, stan, zainteresowania, szuka)  
VALUES ('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl', 'K', '1980-09-05', 'Autorka tekstów  
technicznych', 'Warszawa, MZ', 'Panna', 'Kajakarstwo, gady', 'Związków, przyjaciół');
```

Co jest nie w porządku? **Brak znaku apostrofu za wartością ostatniej kolumny.**

Komunikat wyświetlony przez RDBMS: **ERROR 1064 (42000): You have an error in your SQL syntax; check
the manual that corresponds to your MySQL server version for the right syntax
to use near " at line 4.**

Wariacje na temat polecenia INSERT

Istnieją trzy wariacje polecenia **INSERT**, o których powinieneś wiedzieć.

1 Zmiana kolejności kolumn

Można dowolnie zmieniać kolejność nazw kolumn podawanych w poleceniu SQL, o ile tylko podane w dalszej części zapytania wartości zostaną podane w takiej samej kolejności co kolumny.

```
INSERT INTO moje_kontakty
(zainteresowania, imie, nazwisko, plec, email, data_urodzenia,
zawod, lokalizacja, stan, szuka)
VALUES
('Kajakarstwo, gady', 'Juliana', 'Kowalska', 'K', 'ju.ka@pizza-nzk.
com.pl', '1980-09-05', 'Autorka tekstów technicznych', 'Warszawa,
MZ', 'Panna', 'Związków, przyjaciół');
```

Czy zwróciłeś uwagę na kolejność nazw kolumn? A teraz przyjrzyj się wartościom; zostały podane w takiej samej kolejności. A zatem, o ile tylko kolejność wartości w poleceniu **INSERT** będzie odpowiadać kolejności kolumn podanych w poleceniu, nie będzie ona miała najmniejszego znaczenia ani dla Ciebie, ani dla używanego systemu zarządzania bazami danych.

2 Pominiecie nazw kolumn

Można całkowicie pominąć listę nazw kolumn. Jednak w takim przypadku w poleceniu trzeba podać wartości **wszystkich kolumn** tabeli, a co więcej, muszą one zostać zapisane dokładnie **w takiej samej kolejności**, w jakiej poszczególne **kolumny były dodawane podczas tworzenia tabeli**. (A zatem dokładnie sprawdź kolejność, w jakiej podaliśmy kolumny podczas tworzenia tabeli na stronie 73).

```
INSERT INTO moje_kontakty
VALUES
('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl', 'K', '1980-09-05',
'Autorka tekstów technicznych', 'Warszawa, MZ', 'Panna', 'Kajakarstwo,
gady', 'Związków, przyjaciół');
```

Usunęliśmy z polecenia nazwy kolumn. Jeśli jednak chcesz to zrobić, to musisz podać wartości **WSZYSTKICH kolumn** i to **DOKŁADNIE W TAKIEJ SAMEJ kolejności**, w jakiej wступują w tabeli.

3 Pominiecie niektórych kolumn

Można określić wartości tylko wybranych kolumn, a resztę pominąć.

```
INSERT INTO moje_kontakty
(nazwisko, imie, email)
VALUES
('Kowalska', 'Juliana', 'ju.ka@pizza-nzk.com.pl');
```

W tym przypadku w tabeli zapisujemy jedynie część danych. Ponieważ system zarządzania bazami danych nie wie, o jakie dane nam chodzi, musimy jawnie określić nazwy kolumn oraz wartości, jakie będą zapisywane.



Kolumny bez wartości

Wstawmy do tabeli `moje_kontakty` rekord z przedstawionej poniżej, niekompletnej karteczki:



KOLUMNY:	WARTOŚCI:
<code>nazwisko</code>	?
<code>imie</code>	'Pat'
<code>email</code>	'patpostprusz@pizza-nzk.com.pl'
<code>plec</code>	?
<code>data_urodzenia</code>	?
<code>zawod</code>	urzędnik pocztowy
<code>lokalizacja</code>	Pruszków, DS
<code>stan</code>	?
<code>zainteresowania</code>	?
<code>szuka</code>	?

Ponieważ na kartecce nie ma wszystkich informacji, Grzesiek będzie musiał zapisać w tabeli niekompletny rekord. Ale to żaden problem — brakujące informacje będzie mógł później dodać bez najmniejszych problemów.

W tym przypadku zastosujemy wersję polecenia `INSERT`, w której nie musimy podawać wartości wszystkich kolumn, gdyż pozwoli nam ona podać tylko te kolumny, których wartości znamy.

```
INSERT INTO moje_kontakty ↴
  (imie, email, zawod, lokalizacja)
VALUES
  ('Pat', 'patpostprusz@pizza-nzk.com.pl',
  'urzednik pocztowy', 'Pruszkow, DS');
```

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> INSERT INTO moje_kontakty (imie, email, zawod, lokalizacja)
-> VALUES ('Pat', 'patpostprusz@pizza-nzk.com.pl', 'urzednik pocztowy',
-> 'Pruszkow, DS');
Query OK, 1 row affected (0.05 sec)

mysql> -
```

Zerknij na swoją tabelę, używając polecenia SELECT

A może chciałbyś zobaczyć, jak wygląda Twoja tabela? Cóż, w tym przypadku polecenie DESC się nam nie przyda, gdyż wyświetla ono jedynie strukturę tabeli, a nie informacje, które są w niej zapisane. Konieczne zatem będzie zastosowanie prostego polecenia SELECT, które prezentuje dane zgromadzone w tabeli.

```
C:\Program Files\MySQL\MySQL Server 5.0\bin>mysql
mysql> select * from moje_kontakty;
+-----+-----+-----+-----+-----+
| nazwisko | imie | email | date_uwodzenia | plec |
+-----+-----+-----+-----+-----+
| NULL    | Pat   | patpostcyprus@pizzan-nz2k.com.pl | NULL  | NULL
| Kowalska |Juliana| juli.k@pizzan-nz2k.com.pl | 2009-05-05 | female
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

Chcemy pobrać wszystkie dane przechowywane w naszej tabeli...

... a właśnie gwiazdka (*) informuje, że chodzi o WSZYSTKIE kolumny.

A to nazwa tabeli, której zawartość nas interesuje.

SELECT * FROM moje_kontakty;



Na razie nie przejmuj się tym, co robi polecenie SELECT.

Przedstawimy je znacznie bardziej szczegółowo w rozdziale 2. A na razie siądź wygodnie i rozkoszuj się pięknem swojej tabeli, którą w całej okazałości wyświetliło polecenie SELECT.

W każdej kolumnie, której wartości nie podaliśmy, została zapisana wartość NULL.

A teraz spróbuj to zrobić sam. Aby wyniki zostały ładnie wyświetlane, będziesz musiał odpowiednio powiększyć okno konsoli.

**WYSIL
SZARE KOMÓRKI**

Teraz już wiesz, że we wszystkich kolumnach, których wartości nie zostały określone, została umieszczona wartość NULL. Ale co właściwie to NULL oznacza?



SQL bez tajemnic

W tym tygodniu tematem audycji są:
Wyznania wartości NULL

Rusz głową!: Witam, NULL. Muszę przyznać, że nie spodziewałem się ciebie zobaczyć. Przypuszczałem, że w ogóle nie istniejesz. Krążą pogłoski, że, przepraszam za wyrażenie, jesteś zwyczajnym zerem, albo że wcale nie istniejesz.

NULL: Nie mogę uwierzyć, że w ogóle słuchałeś takich kłamstw. Spójrz, jestem tutaj i to jestem całkowicie rzeczywista. Czy zatem uważasz, że jestem niczym, jedynie prochem pod twoimi stopami?

Rusz głową!: Już dobrze, uspokój się, proszę. Chodzi tylko o to, że pojawiasz się zawsze tam, gdzie nie ma żadnej wartości...

NULL: Jasne, ale lepiej, żebybym to ja się pojawiła, niż na przykład zero lub pusty łańcuch znaków.

Rusz głowa!: A co to takiego, ten pusty łańcuch znaków?

NULL: To coś, co powstanie, kiedy zapiszesz obok siebie dwa apostrofy, nie umieszczaając wewnętrz nich żadnych innych znaków. To cały czas jest łańcuch znaków, lecz ma zerową długość. Powstanie, gdy na przykład kolumnie imię w tabeli `moje_kontakty` przypiszesz ''.

Rusz głowa!: Czyli nie jesteś jedynie wymyślnym sposobem oznaczenia „niczego”.

NULL: Powiedziałam już, że nie jestem „niczym”. Jestem... „czymś”. Chodzi o to, że jestem tylko trochę... niezdefiniowana... I to wszystko.

Rusz głowa!: Czy to oznacza, że jeśli porównamy cię do zera lub pustego łańcucha znaków, to okaże się, że nie jesteście równe?

NULL: Absolutnie nie! Nigdy nie będę równa zeru. Co więcej, nigdy nie będę równa innej wartości NULL. My, NULL-e, nigdy nie jesteśmy sobie równe. Wartość może być NULL, jednak nigdy nie będzie **równa** NULL, gdyż NULL jest wartością niezdefiniowaną! Rozumiesz?

Rusz głowa!: Hej, uspokój się, proszę, i pozwól mi to wszystko wyjaśnić. Nie jesteś równa zeru i nie jesteś pustym łańcuchem znaków. A nawet nie jesteś równa samej sobie. Przecież to nie ma najmniejszego sensu!

NULL: Wiem, że to trochę skomplikowane. Kiedy myślę o tym, wyobrażam sobie, że jestem niezdefiniowana. Jestem niczym zawartość zamkniętego pudełka. Wewnątrz niego może być cokolwiek. Nie możesz zatem porównać jednego zamkniętego pudełka z drugim, ponieważ nie wiesz, co w nich jest. Może się nawet okazać, że takie pudełko, czyli ja, jest puste. Ale po prostu tego nie wiesz.

Rusz głowa!: Słyszałem jednak, że czasami jesteś niepożądana. W niektórych przypadkach okazuje się, że NULL może przysparzać problemów.

NULL: Przypnaję, że czasami pojawiam się w miejscach, w których raczej nie chciano by mnie widzieć. Chodzi o to, że niektóre kolumny zawsze powinny mieć jakąś wartość. Weźmy na przykład takie „nazwisko”. Umieszczanie wartości NULL w kolumnie nazwiska w tabeli nie ma najmniejszego sensu.

Rusz głowa!: A zatem nie pojawiłabyś się tam, gdzie by cię nie chciano?

NULL: Oczywiście, że nie! Wystarczy mi tylko o tym powiedzieć. Wystarczy, że przekażesz mi odpowiednią informację podczas tworzenia tabeli i określania jej kolumn.

Rusz głowa!: Tak naprawdę, to wcale nie wyglądasz jak zamknięte pudełko.

NULL: To już przesada. Mam sporo miejsc, które muszę odwiedzić, i wartości, którymi muszę się zająć.

Kontrola wewnętrznych wartości NULL

Niektóre kolumny tabeli zawsze powinny mieć jakieś wartości. Czy pamiętasz niekompletną karteczkę dotyczącą osoby o imieniu Pat, na której nie zapisano nazwiska? Nie będzie łatwo odnaleźć jej (lub jego?), jeśli w tabeli będzie dwadzieścia innych rekordów, w których w polu nazwisko także będzie zapisana wartość NULL. Bardzo łatwo można stworzyć tabelę w taki sposób, by w wybranych kolumnach nie mogły się pojawiać wartości NULL.

CREATE TABLE moje_kontakty

(

nazwisko VARCHAR (30) NOT NULL,

imie VARCHAR (20) NOT NULL

);

Wystarczy dodać słowa NOT NULL tuż za określeniem typu kolumny.

W tym przypadku będziesz musiał określić wartość kolumny w poleceniu INSERT. W przeciwnym razie, jeśli pominiesz wartość, zostanie zgłoszony błąd.

Zaostrz ołówek



```
CREATE TABLE moje_kontakty
(
    nazwisko VARCHAR(30) NOT NULL,
    imie VARCHAR(20) NOT NULL,
    email VARCHAR(50),
    plec CHAR(1),
    data_urodzenia DATE,
    zawod VARCHAR(50),
    lokalizacja VARCHAR(50),
    stan VARCHAR(20),
    zainteresowania VARCHAR(100),
    szuka VARCHAR(100)
);
```

Przyjrzyj się wszystkim kolumnom w poleceniu CREATE TABLE, służącym do tworzenia naszej tabeli moje_kontakty. Do których z nich powinniśmy także dodać słowa NOT NULL? Zastanów się, które kolumny nigdy nie powinny być puste; zaznacz je.

Aby Ci pomóc, zaznaczyliśmy dwie takie kolumny, pozostałe zaznacz sam. Przede wszystkim skoncentruj się na kolumnach, które później mogą być używane do wyszukiwania informacji, oraz na takich, których wartości są unikalne.

Zaostrz ołówek —————
Rozwiązywanie

```
CREATE TABLE moje_kontakty
(
    nazwisko VARCHAR(30) NOT NULL,
    imie VARCHAR(20) NOT NULL,
    email VARCHAR(50),
    plec CHAR(1),
    data_urodzenia DATE,
    zawod VARCHAR(50),
    lokalizacja VARCHAR(50),
    stan VARCHAR(20),
    zainteresowania VARCHAR(100),
    szuka VARCHAR(100)
);
```

Przyjrzyj się wszystkim kolumnom w poleceniu CREATE TABLE, służącym do tworzenia naszej tabeli moje_kontakty. Do których z nich powinniśmy także dodać słowa kluczowe NOT NULL? Zastanów się, które kolumny nigdy nie powinny być puste; zaznacz je.

Aby Ci pomóc, zaznaczyliśmy dwie takie kolumny, pozostałe zaznacz sam. Przede wszystkim skoncentruj się na kolumnach, które później mogą być używane do wyszukiwania informacji, oraz takich, których wartości są unikalne

→
Słowa kluczowe NOT NULL powinny zostać dodane do wszystkich kolumn.

Wszystkie kolumny tabeli będą używane podczas wyszukiwania. Bardzo ważne jest, by mieć pewność, że informacje są kompletne, a w tabeli zostały zapisane prawidłowe dane...

...jeśli jednak w tabeli jest jakąś kolumna, która na pewno będzie musiała zostać wypełniona w przyszłości, po utworzeniu rekordu, to będziesz chciał, by w niej możliwe było zapisywanie wartości NULL.

NOT NULL pojawia się w wynikach polecenia DESC

Poniżej pokazaliśmy, jak wyglądałaby tabela moje_kontakty, gdyby do wszystkich kolumn zostały dodane słowa kluczowe NOT NULL.

Tutaj tworzymy naszą tabelę z wyrażeniem NOT NULL w każdej kolumnie.

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> CREATE TABLE moje_kontakty
    -> (
    ->     nazwisko VARCHAR<30> NOT NULL,
    ->     imie VARCHAR<20> NOT NULL,
    ->     email VARCHAR<50> NOT NULL,
    ->     plec CHAR<1> NOT NULL,
    ->     data_urodzenia DATE NOT NULL,
    ->     zawod VARCHAR<50> NOT NULL,
    ->     lokalizacja VARCHAR<50> NOT NULL,
    ->     stan VARCHAR<20> NOT NULL,
    ->     zainteresowania VARCHAR<100> NOT NULL,
    ->     szuka VARCHAR<100> NOT NULL
    -> );
Query OK, 0 rows affected (0.23 sec)

mysql> DESC moje_kontakty;
+-----+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nazwisko | varchar(30) | NO |   |   |       |
| imie | varchar(20) | NO |   |   |       |
| email | varchar(50) | NO |   |   |       |
| plec | char(1) | NO |   |   |       |
| data_urodzenia | date | NO |   |   |       |
| zawod | varchar(50) | NO |   |   |       |
| lokalizacja | varchar(50) | NO |   |   |       |
| stan | varchar(20) | NO |   |   |       |
| zainteresowania | varchar(100) | NO |   |   |       |
| szuka | varchar(100) | NO |   |   |       |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.16 sec)

mysql>
```

A oto i wyświetlona struktura tabeli. Zwróć uwagę na słowo NO w kolumnie Null.

Wypełnij pustkę słowem kluczowym DEFAULT

Jeśli w naszej tabeli jest jakaś kolumna, w której zazwyczaj zapisywana jest konkretna, dobrze nam znana wartość, to możemy zdefiniować w takiej kolumnie wartość **domyślną** — do tego właśnie służy słowo kluczowe DEFAULT. Wartość podana za tym słowem kluczowym będzie automatycznie zapisywana w każdym wierszu tabeli, jeśli żadna inna wartość nie zostanie jawnie określona. Wartość domyślna musi mieć ten sam typ, co kolumna, w której będzie zapisywana.

CREATE TABLE lista_paczekow

(

nazwa_paczka VARCHAR(25),

typ_paczka VARCHAR(20),

cena_paczka DEC(3,2) NOT NULL DEFAULT 1.00

) ;

- Chcemy mieć pewność, że w tej kolumnie zawsze będzie zapisana jakaś wartość. Nie tylko możemy sprawić, że nie będzie się w niej pojawiać wartość NULL, lecz także możemy jej przypisać DOMYŚLNA, wartość, na przykład 1,00 zł.



- To właśnie ta wartość zostanie zapisana w tabeli w kolumnie cena_paczka, jeśli żadna inna wartość nie zostanie jawnie określona.

lista_paczekow

nazwa_paczka	typ_paczka	cena_paczka
Jagodowiec	z nadzieniem	2.00
Pączek cynamonowy	pierścień	1.00
Gwiazda rocka	ciastko plecone	1.00
Karmelowiec	ciastko plecone	1.00
Jabłkowiec	z nadzieniem	1.40

Oto jak wyglądałaby nasza tabela, gdyby podczas zapisywania w niej danych w rekordach dotyczących pączków pączek cynamonowy, gwiazda rocka oraz karmelowiec nie została określona wartość kolumny cena_paczka.

Zastosowanie słowa kluczowego DEFAULT sprawi, że w pustej kolumnie zostanie zapisana określona wartość domyślna.

Kim jestem?



ze strony 69

Grupa poprzedzanych w kostiumy słów kluczowych i typów danych SQL bawi się w grę towarzyską o nazwie „Zgadnij, kim jestem”. Jej uczestnicy udzielają Ci odpowiedzi, a Ty próbujesz na ich podstawie odgadnąć, kim oni są. Załóż, że uczestnicy gry zawsze mówią prawdę. Jeśli zdarzy się, że udzielona odpowiedź odnosi się do kilku uczestników gry, to zapisz nazwy ich wszystkich. Nazwy uczestników gry zapisz w pustych miejscach widocznych z prawej strony podpowiedzi.

W dzisiejszej zabawie udział biorą:

**CREATE DATABASE, USE DATABASE, CREATE TABLE, DESC,
DROP TABLE, CHAR, VARCHAR, BLOB, DATE, DATETIME,
DEC oraz INT.**

Mam Twój numer.

Pomogę Ci pozbyć się niepożądanych tabel.

Najbardziej lubię pytania, na które można odpowiedzieć „Tak” lub „Nie”.

Zapamiętam datę urodzin Twojej mamy.

Ja trzymam w garści wszystkie tabele.

Liczby są super, ale nie cierpię ułamków.

Uwielbiam długie, opisowe wyjaśnienia.

To jest miejsce, w którym wszystko można przechować.

Beze mnie tabela w ogóle by nie istniała.

Doskonale wiem, kiedy w przyszłym tygodniu jesteś umówiony u dentysty.

Jestem ulubieńcem księgowych.

Dzięki mnie poznasz format tabeli.

Bez nas nie byłbyś w stanie utworzyć tabeli.

Uczestnicy

DEC, INT

DROP TABLE

CHAR(1)

Mozesz sobie przyznać dodatkowe punkty, jeśli w odpowiedzi podałeś: (1).

DATE

CREATE DATABASE

INT

BLOB

CREATE TABLE

CREATE DATABASE

DATETIME

DEC

DESC

**CREATE DATABASE, USE DATABASE,
DROP TABLE**



Przybornik SQL

A zatem opanowałeś już materiał z pierwszego rozdziału książki i wiesz, jak tworzyć bazy danych i tabele, w jaki sposób zapisywać w tabelach dane kilku najczęściej stosowanych typów oraz jak wymusić, by w wybranych kolumnach zawsze były wartości.

CREATE DATABASE

Używaj tego polecenia do utworzenia nowej bazy danych, która będzie zawierać wszystkie Twoje tabele.

USE DATABASE

Pozwala wskazać bazę danych, na której chcesz pracować, na przykład by utworzyć w niej tabele.

NULL oraz NOT NULL

Powinieneś także zastanowić się i wskazać kolumny, w których nie mogą występować wartości NULL, na przykład dlatego, że utrudniłoby to późniejsze wyszukiwanie i sortowanie danych. Podczas tworzenia takich kolumn będziesz musiał użyć w nich słów kluczowych NOT NULL.

DEFAULT

To słowo kluczowe pozwala określić domyślną wartość kolumny, która będzie używana, jeśli podczas zapisywania rekordu do tabeli wartość danej kolumny nie zostanie jawnie podana.

CREATE TABLE

Rozpoczyna proces tworzenia tabeli, jednak będziesz także musiał znać NAZWY KOLUMN oraz TYPY DANYCH. Będziesz je musiał określić, analizując rodzaje informacji, jakie chcesz przechowywać w tabeli.

DROP TABLE

To polecenie pozwala usunąć tabelę z bazy, na przykład w sytuacji, gdy tworząc ją popełnisz jakiś błąd. Jednak powinieneś z niego korzystać, jeszcze zanim zaczniesz dodawać do tabeli dane przy użyciu polecenia INSERT.

CELNE SPOSTRZEŻENIA

- ◆ Jeśli chcesz sprawdzić lub poznać strukturę tabeli, to możesz ją wyświetlić przy użyciu polecenia DESC.
- ◆ Polecenia DROP TABLE można używać do usuwania tabel. Stosuj je z dużą ostrożnością.
- ◆ Aby zapisać w tabeli jakieś informacje, możesz zastosować jedną z kilku wersji polecenia INSERT.
- ◆ Wartość NULL to wartość niezdefiniowana. Nie jest to ani wartość pusta, ani zero. Kolumna, w której zapisano tę wartość, jest wartością NULL, jednak *nie jest równa* wartości NULL.
- ◆ W kolumnach, których wartości nie zostały określone w poleceniu INSERT, domyślnie jest zapisywana wartość NULL.
- ◆ Można zaznaczyć, że w kolumnie nie mogą być zapisywane wartości NULL — wystarczy podczas jej tworzenia dodać do definicji kolumny słowa kluczowe NOT NULL.
- ◆ Jeśli podczas tworzenia kolumny tabeli umieścis w jej definicji słowo kluczowe DEFAULT i pewną wartość, to będzie ona automatycznie używana, jeśli podczas zapisywania rekordu w tabeli wartość kolumny nie zostanie jawnie podana.

2. Polecenie SELECT

Pobieranie podarowanych danych

SELECT * FROM prezenty
WHERE zawartosc = "drogocenna";



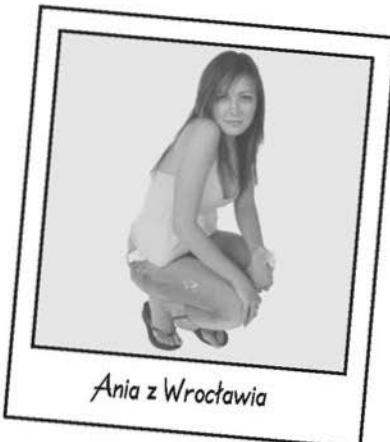
Czy naprawdę lepiej jest dawać, niż brać? W przypadku korzystania z baz danych najprawdopodobniej częściej będziesz musiał **pobierać z nich dane**, niż je zapisywać. I właśnie w tych wszystkich sytuacjach przydadzą Ci się informacje przedstawione w niniejszym rozdziale: poznasz w nim bardzo przydatne polecenie **SELECT** i dowiesz się, jak **uzyskać dostęp do tych wszystkich ważnych informacji**, które wcześniej zapisywałeś w swoich tabelach. Co więcej, dowiesz się także, w jaki sposób stosować klauzulę **WHERE** i operatory **AND** i **OR**, by nie tylko pobierać dane, lecz wyświetlać tylko te, które są Ci potrzebne.

Pobierać dane czy nie pobierać?

Grzesiek skończył właśnie zapisywać wszystkie informacje ze swoich karteczek w tabeli `moje_kontakty`. Teraz może się wreszcie odprężyć. Kupił sobie dwa bilety na koncert i chce zaprosić na randkę jedną z osób ze swoich karteczek — dziewczynę z Wrocławia.

Grzesiek musi znaleźć jej adres poczty elektronicznej, zatem wyświetla zawartość tabeli przy użyciu polecenia **SELECT** przedstawionego w rozdziale 1.

SELECT * FROM moje_kontakty;



Dane o Ani są zapisane
gdzieś w tabeli Grzesia...



Bądź Grześkiem

Twoim zadaniem jest wcielenie się w rolę Grzęska. Musisz przejrzeć fragment danych z tabeli `moje_kontakty` przedstawionych na następnej stronie i odszukać w nich Anię z Wrocławia.

Tabela moje_kontakty ma całkiem sporo kolumn. Poniżej widocznych jest tylko kilka pierwszych.



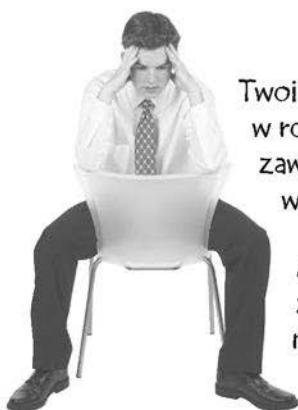
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe			
nazwisko	imie	email	plec
Kubaski	Andrzej	Kubaski_Andrzej@pizza-nzk.com.pl	M
Nowak	Joanna	now.jn@op.pl	K
Symonowicz	Anna	anna_tymonowicz@pizza-nzk.com.pl	K
Oczkiewicz	Adam	oczan@zygzyg.eu	M
Karczynska	Marta	karmar@interia.pl	K
Zefirek	Monika	moniaze@pizza-nzk.com.pl	K
Harmonia	Ania	a_harmonia@mojamusa.pl	K
Mikulski	Janek	mikunek@wp.pl	M
Tkacz	Martyna	mptkacz@o2.eu	K
Maliniak	Konrad	kon_malinka@op.pl	M
Piekarska	Kasia	mysia_kasia@o2.eu	K
Kowalski	Mirek	super_mi@op.pl	M
Raczyńska	Anna	a.raczyńska@mbiznes.eu	K
Nerczynski	Piotr	p.nera@wp.pl	M
Kowalski	Jerzy	j.kowalski@betrendy.pl	M
Piekarska	Marta	piekat@onet.pl	K
Nowak	Andrzej	n.andrew@interia.pl	M
Maliniak	Maciej	kiszka@zygzyg.eu	M
Lewandowicz	Barbara	bbaska@pizza-nzk.com.pl	K
Kowalska	Anna	ania_k@gwiezdnypl.com.pl	K
Bilecki	Janusz	tramjan@interia.pl	M
Kowalska	Zosia	zetka@op.pl	K
Kubaski	Milosz	dj_milo@nfalaf.lm	M
Zofik	Tadeusz	t.zofik@pizza-nzk.com.pl	M
Kowalska	Magda	miziak@op.pl	K
Tarczynski	Piotr	p.tarczynski@ynteria.pl	M
Maliniak	Piotr	p_malinka@op.pl	M
Raczyńska	Zofia	z_rf@o2.pl	K
Buczyńska	Anna	buna@pizza-nzk.com.pl	K
Arbacki	Melchior	melchior_a@pizza-nzk.com.pl	M
Karczynska	Katarzyna	kaka@op.pl	K
Mikulski	Krzysztof	s.laandch@op.pl	M
Nowak	Milena	milena_ner@o2.pl	K

lokalizacja
Gliwice, SL
Krakow, MP
Wroclaw, DS
Elblag, WM
Katowice, SL
Tychy, SL
Wroclaw, DS
Poznan, WP
Wroclaw, DS
Lubartow, LU
Katowice, SL
Warszawa, MZ
Krakow, MP
Gdansk, PM
Gliwice, SL
Lodz, LD
Katowice, SL
Slupsk, PM
Wroclaw, DS
Wroclaw, DS
Tychy, SL
Katowice, SL
Warszawa, MZ
Mikolow, SL
Warszawa, MZ
Poznan, WP
Katowice, SL
Warszawa, MZ
Wroclaw, DS
Poznan, WP
Krakow, MP
Warszawa, MZ
Warszawa, MZ

To jeszcze nie jest koniec tabeli! Grzesiek miał naprawdę sporo karteczek.



Bądź Grzeskiem. Rozwiążanie



Twoim zadaniem było wcielenie się w rolę Grześka i przejrzenie fragmentu zawartości tabeli `moje_kontakty` w poszukiwaniu Ani z Wrocławia.

Musiałeś odszukać wszystkie Anie z Wrocławia i zapisać ich imiona, nazwiska oraz adresy poczty elektronicznej.

Tymonowicz Anna: anna_tymonowicz@pizza-nzk.com.pl

Harmonia Ania: a_harmonia@mojamuza.pl

Kowalska Anna: ania_k@gwiezdnypl.com.pl

Buczyńska Anna: buna@pizza-nzk.com.pl

Oto wszystkie Anie w tabeli,
wraz z ich adresami e-mail.

Grzesiek szuka tylko „Ani” lub
„Anny”; jeśli w wynikach znajdziesz
jakąś „Ankę”, to możesz ją pominąć.

Nawiązanie kontaktu

To wyszukiwanie trwało **zdecydowanie za długo** i było **niezwykle męczące**. Co więcej, istnieje naprawdę duże i realne zagrożenie, że przeszukując dane w taki sposób, Grzesiek przeoczy jedną z Ani... może nawet właśnie tę, której szuka.

Teraz, kiedy Grzesiek odszukał już adresy wszystkich Ani z Wrocławia, może napisać do nich maila i poznać odpowiedzi...

Do: Symonowicz, Anna <anna_tymonowicz@pizza-nzk.com.pl>
Od: Grzesiek <grzegorz@listagreska.com.pl>
Temat: Czy spotkaliśmy się w kawiarni Gwiezdny Pył?

Od dawna szukam takiego kowboja jak ty. Wpadnij około 17, to sobie coś upichcimy...

Do: Harmonia, Ania <a_harmonia@mojamuza.pl>
Od: Grzesiek <grzegorz@listagreska.com.pl>
Temat: Czy spotkaliśmy się w kawiarni Gwiezdny Pył?

Aktualnie chodzę z fantastycznym chłopakiem — Maćkiem. Spotkaliśmy się na imprezie w akademiku u mojej kumpelki...

Do: Kowalska, Anna <ania_k@gwiezdnypl.com.pl>
Od: Grzesiek <grzegorz@listagreska.com.pl>
Temat: Czy spotkaliśmy się w kawiarni Gwiezdny Pył?

Nie jestem ta Anią, której szukasz; ale na pewno jest uroczą osobką. Gdyby wam nie wyszło, odezwij się do mnie.

Do:
Od: Grzesiek <grzegorz@listagreska.com.pl>
Temat: Czy spotkaliśmy się w kawiarni
Gwiezdny Pył?

Oczywiście, że cię pamiętam! Szkoda, że nie odezwałeś się wcześniej. Umówilam się już ze swoim byłym, który chce, żebyśmy znowu byli razem.

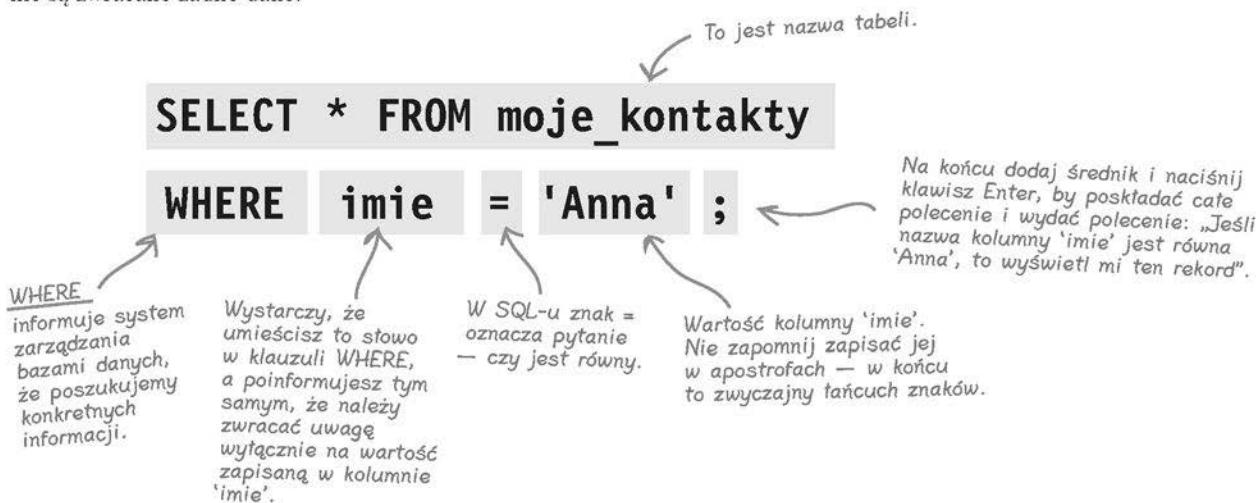
WYSIL SZARE KOMÓRKI

Czy możesz zaproponować jakiś sposób zapisania polecenia SQL, który pozwoliłby nam wyświetlić tylko te rekordy, w których kolumna imię zawiera słowo "Anna"?

Lepsza postać polecenia SELECT

Poniżej przedstawiliśmy wersję polecenia SELECT, której zastosowanie pomogłoby Grześkowi odszukać Anię znacznie szybciej i łatwiej — bez konieczności żmudnego przeglądania całej zawartości tabeli. W tym poleceniu zastosowaliśmy klauzulę **WHERE**, która przekazuje systemowi zarządzania bazami danych informacje o tym, czego konkretnie należy poszukiwać. Zawęża ona uzyskiwane wyniki i zwraca tylko te wiersze, które spełniają zadany warunek.

Znak równości umieszczony w klauzuli WHERE służy do tego, by sprawdzić, czy każda z wartości zapisanych w kolumnie imie jest równa lub odpowiada słowu 'Anna'. Jeśli wartość pasuje, to cały wiersz jest zwracany; w przeciwnym razie nie są zwracane żadne dane.



Poniżej zamieściliśmy rysunek przedstawiający okno konsoli, w którym zostały wyświetlane wyniki wykonania powyższego polecenia — jego zadaniem było wyświetlenie wszystkich rekordów, w których kolumna imie zawiera wartość 'Anna'.

```

C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe

mysql> SELECT * FROM moje_kontakty WHERE imie = 'Anna';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| nazwisko | imie | email | plec | data_urodzenia | zawod | lokalizacja | stan | zainteresowania | szuka |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Symonowicz | Anna | anna_tymonowicz@pizza-nzk.com.pl | K | NULL | NULL | Krakow, DS | | |
| Raczyńska | Anna | a.raczyńska@biznes.eu | K | NULL | NULL | Krakow, MP | |
| Kowalska | Anna | anna_k@witezdznypl.com.pl | K | NULL | NULL | Wroclaw, DS | |
| Buczynska | Anna | buna@pizza-nzk.com.pl | K | NULL | NULL | Wroclaw, DS | |
| Nowak | Anna | anianowak00@pizza-nzk.com.pl | K | NULL | NULL | Krakow, MP | |
| Jozwiak | Anna | anorek@pizza-nzk.com.pl | K | NULL | NULL | Gdansk, PM | |
| Zygmunt | Anna | us@pizza-nzk.com.pl | K | NULL | NULL | Nowy Targ, MP | |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

```

Oto wyniki wykonania naszego polecenia SELECT.



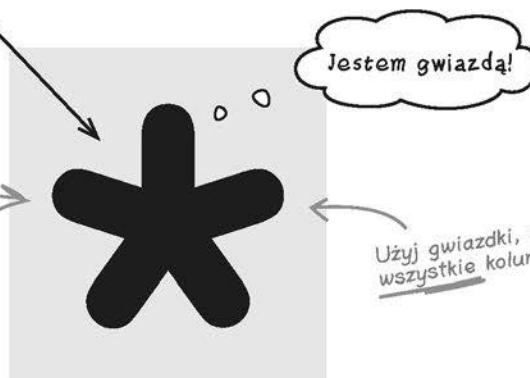
Ej, zaraz, zaraz. Nie przemycie tej gwiazdki tak bez żadnego słowa wyjaśnienia.
Co ona oznacza?

Co oznacza gwiazdka (*)

Gwiazdka informuje system zarządzania bazami danych o tym, że należy zwrócić wartości ze **wszystkich** kolumn tabeli.

```
SELECT * FROM moje_kontakty  
WHERE imie = 'Anna';
```

Jeśli w poleceniu SQL zobaczyś kod `SELECT *`, wyobraź sobie, że prosisz system zarządzania **WSZYSTKICH KOLUMN TABELI**.



Nie istnieją grupie pytania

P: A jeśli nie będę chciał pobierać wartości **wszystkich** kolumn? Czy mogę użyć jakiegoś innego znaku zamiast gwiazdki?

O: Innego znaku nie możesz użyć, jednak zwracanie **wszystkich** kolumn nie jest jedynym rozwiązaniem, jakim dysponujemy. Już niedługo dowiesz się, w jaki sposób można zwracać wartości jedynie wybranych kolumn, dzięki czemu wyniki zapytania będą łatwiejsze do zinterpretowania.

P: Czy ta gwiazdka to znak mnożenia?

O: Tak, to dokładnie ten sam znak umieszczonego na klawiszu z cyfrą 8. Wciśnij klawisz `Shift` i naciśnij ten klawisz. Jednak choć jest to ten sam znak, to w terminologii języka SQL jest on zawsze określany jako *gwiazdka*. I całkiem słusznie, gdyż łatwiej jest powiedzieć `SELECT gwiazdka FROM...` niż `SELECT znak mnożenia FROM...`.

P: Czy są jeszcze inne znaki mające podobnie jak gwiazdka specjalne znaczenie?

O: Owszem, w języku SQL są stosowane jeszcze inne znaki specjalne — nazywane także znakami zarezerwowanymi. Poznasz je w dalszej części książki. Jednak jak na razie gwiazdka jest jedynym z takich znaków, który powinieneś znać. Jest ona także jedynym znakiem specjalnym używanym w pierwszej części polecenia `SELECT`.



Ćwiczenie

Salon Rusz głośna! chce wzbogacić swoje menu o wybór drinków. Bazując na wiedzy zdobytej w rozdziale 1., stwórz poniższą tabelę i zapisz w bazie zamieszczone w niej dane.

Poniższa tabela należy do bazy danych o nazwie drinki. Zawiera ona tabelę o nazwie proste_drinki, w której są gromadzone przepisy na napoje tworzone na bazie jedynie dwóch składników.

proste_drinki

nazwa	skladnik_główny	ilosc1	skladnik_dodatkowy	ilosc2	wskazówki
Blackthorn	tonik	45	sok ananasowy	30	wymieszać z lodem, odcedzić do szklanki koktajlowej z plasterkiem cytryny
Blue Moon	woda sodowa	45	sok z jagód	22	wymieszać z lodem, odcedzić do szklanki koktajlowej z plasterkiem cytryny
Oh My Gosh	nektar brzoskwienny	30	sok ananasowy	30	wymieszać z lodem, odcedzić do wysokiej szklanki
Lime Fizz	Sprite	45	sok z cytryny	22	wymieszać z lodem, odcedzić do szklanki koktajlowej
Kiss on the Lips	sok wiśniowy	60	nektar morelowy	210	podawać z lodem i słomką do picia
Hot Gold	nektar brzoskwienny	90	sok pomarańczowy	180	wlać gorący sok pomarańczowy do kubka, dodać do niego nektaru brzoskwiennego
Lone Tree	woda sodowa	45	sok wiśniowy	22	wymieszać z lodem, odcedzić do szklanki koktajlowej
Greyhound	woda sodowa	45	sok z grapefruita	150	podawać z lodem, dobrze wymieszać
Indian Summer	sok jabłkowy	60	gorąca herbata	180	wlać sok do kubka, dodać herbaty
Bull Frog	mrożona herbata	45	lemoniada	150	podawać z lodem i plasterkiem cytryny
Soda and It	woda sodowa	60	sok z winogron	30	wstrąsnąć w szklance koktajlowej, podawać bez lodu

ilosc1 oraz ilosc2
to wartości podawane
w mililitrach

→ Odpowiedź znajdziesz na stronie 151



Zanim zaczniesz, przemyśl i zaplanuj rozwiązanie.

Uważnie dobieraj typy danych poszczególnych kolumn i nie zapominaj o wartościach NULL. Później porównaj swoją odpowiedź z rozwiązaniem zamieszczonym na stronie 151.

Wyszukiwanie drinków

Zaostrz ołówek



Na razie nie przejmuj się nieznanymi znakami zastosowanymi w poniższych zapytaniach. Po prostu wpisz je w takiej postaci, w jakiej je podaliśmy, i sprawdź, czy polecenia zadziałają prawidłowo.

Nazwij ten drink

Wykorzystaj utworzoną przed chwilą tabelę **proste_drinki** i spróbuj wykonać poniższe polecenia SQL, nazywane także zapytaniami, na swoim komputerze. Zanotuj sobie, które drinki są wyświetlane jako wyniki poszczególnych zapytań.



```
SELECT * FROM proste_drinki WHERE skladnik_glowny = 'Sprite';
```

Wyszukane drinki:

```
SELECT * FROM proste_drinki WHERE skladnik_glowny = woda sodowa;
```

Wyszukane drinki:

```
SELECT * FROM proste_drinki WHERE ilosc2 = 180;
```

Wyszukane drinki:

```
SELECT * FROM proste_drinki WHERE skladnik_dodatkowy = "sok pomaranczowy";
```

Wyszukane drinki:

```
SELECT * FROM proste_drinki WHERE ilosc1 < 45;
```

Wyszukane drinki:

```
SELECT * FROM proste_drinki WHERE ilosc2 < '30';
```

Wyszukane drinki:

```
SELECT * FROM proste_drinki WHERE skladnik_glowny > 'tonik';
```

Wyszukane drinki:

```
SELECT * FROM proste_drinki WHERE ilosc1 = '45';
```

Wyszukane drinki:



Chwileczkę. Napisaliście „spróbuj wykonać poniższe zapytania”. Czyli zakładacie, że wszystkie te zapytania są prawidłowe i można je wykonać. I ja wasm zaufałam! A okazało się, że jedno z nich nie działa. Nie wspominając o tym, że kilka innych wygląda tak, jakby w ogóle nie miało prawa działać.

Owszem. Masz rację.

Jedno z przedstawionych zapytań nie będzie działać. Pozostałe można wykonać, jednak zwracane przez nie wyniki nie zawsze będą takie, jakich by można oczekiwać.

Aby zdobyć kilka dodatkowych punktów, zapisz poniżej polecenie SQL, które nie będzie działać...

.....

...a które zadziałały niezgodnie z Twoimi oczekiwaniami:

.....

.....

.....

.....

Zaostrz ołówek — Rozwiążanie

Nazwij ten drink

Wykorzystaj utworzoną przed chwilą tabelę **proste_drinki** i spróbuj wykonać poniższe zapytania na swoim komputerze. Zanotuj sobie, które drinki są wyświetlane jako wyniki poszczególnych zapytań.



```
SELECT * FROM proste_drinki WHERE skladnik_glowny = 'Sprite';
```

Wyszukane drinki: Lime Fizz

Zwróć uwagę na znaki apostrofu.

```
SELECT * FROM proste_drinki WHERE skladnik_glowny = woda sodowa;
```

Wyszukane drinki: Błęd

Hm... Wygląda na to, że właściwe zapytanie nie chce działać...

```
SELECT * FROM proste_drinki WHERE ilosc2 = 180;
```

Wyszukane drinki: Hot Cold, Indian Summer

To zapytanie działa.
Używana wielkość jest typu DEC, więc nie trzeba jej zapisywać w apostrofach.

```
SELECT * FROM proste_drinki WHERE skladnik_dodatkowy = "sok pomaranczowy";
```

Wyszukane drinki: Brak wyników

```
SELECT * FROM proste_drinki WHERE ilosc1 < 45;
```

Wyszukane drinki: Oh My Gosh

```
SELECT * FROM proste_drinki WHERE ilosc2 < '30';
```

Wyszukane drinki: Blue Moon, Lime Fizz, Lone Tree

```
SELECT * FROM proste_drinki WHERE skladnik_glowny > 'tonik';
```

Wyszukane drinki: Blue Moon, Lone Tree, Grayhound, Soda and It

Kolejna prawidłowo napisana klauzula WHERE.

```
SELECT * FROM proste_drinki WHERE ilosc1 = '45';
```

Wyszukane drinki: Blackthorn, Blue Moon, Lime Fizz, Lone Tree, Grayhound, Bull Frog

Aby zdobyć dodatkowe punkty, zapisz poniżej polecenie, które nie będzie działać...

To jest klauzula WHERE, która nie będzie działać. Umieszczony na końcułańcuch znaków musi zostać zapisany wewnątrz znaków apostrofu.

WHERE składnik_głowny = woda sodowa;

... oraz polecenia, które nie zadziałyły zgodnie z oczekiwaniami.

WHERE składnik_dodatkowy = "sok pomaranczowy";

To zapytanie działa i nie powoduje wystąpienia jakichkolwiek błędów, jednak nie zwraca ono żadnych wyników. SQL oczekuje zastosowania znaków apostrofu, jak robiliśmy podczas zapisywania danych w tabeli.

WHERE ilosc2 < '30';

To zapytanie działa, pomimo iż nie powinno, gdyż wartości kolumn typu INT nie muszą być zapisywane w apostrofach.

WHERE ilosc1 = '45';

I to podobnie.

Dwa ostatnie z przedstawionych na tej stronie zapytań będą działać, gdyż większość systemów obsług relacyjnych baz danych zapewnia pewną dozę swobody. Zignorują one apostrofy i będą traktować wartości typów DEC oraz INT jako liczby, choć apostrofy sugerują, że należały je uznać za łańcuchy znaków. Przedstawione polecenia SQL nie są całkowicie PRAWIDŁOWE, jednak systemy zarządzania bazami danych zignorują to.

Jak poszukiwać wartości różnych typów?

Aby tworzyć klauzule WHERE, musisz się upewnić, że stosowane wartości różnych typów są prawidłowo zapisane. Poniżej zamieściliśmy konwencje zapisu wartości wszystkich najczęściej używanych typów danych.



My ❤️ apostrofy	Apostrofy? — Nie, dziękuję!
CHAR	DEC
VARCHAR	INT
DATE	
DATETIME, TIME oraz TIMESTAMP	
BLOB	

**Wartości typów
VARCHAR, CHAR,
BLOB, DATE
oraz TIME muszą
być zapisywane
w apostrofach.
Wymóg ten nie
dotyczy jednak
wartości liczbowych,
takich jak wartości
kolumn typów
DEC oraz INT.**

Kolejne problemy ze znakami przestankowymi

Grzesiek zdobył tej nocy kilka dodatkowych kontaktów. Teraz próbuje dodać jeden z nich do swojej tabeli:

```
INSERT INTO moje_kontakty
VALUES
```

```
('Fraszynski', 'Stefan', 's.fraszynski@megaimpreza.com.pl', 'DJ, animator imprez muzycznych', 'Swietochlowice, SL', 'Kawaler', 'jedzenie z McDonald's, muzyka, zabawa', 'ziomali, gitarzystow');
```

Stefan Fraszynski
Data urodzenia: 1.4.1970
DJ, animator imprez muzycznych

Kawaler

Lokalizacja: Świętochłowice, SL
s.fraszynski@megaimpreza.com.pl

Zainteresowania: jedzenie z McDonald's, muzyka, zabawa

Szuka: ziomali, gitarzystow

Ale po próbie wykonania tego polecenia program konsoli chyba się „zawiesił” i przestał odpowiadać. Grześ wpisał nawet kilka średników, próbując zakończyć i wykonać polecenie. Wszystko na nic.

```
Wiersz polecenia - mysql -u root -p
mysql> INSERT INTO moje_kontakty
-> VALUES
-> <'Fraszynski', 'Stefan', 's.fraszynski@megaimpreza.com.pl', 'DJ, animator
imprez muzycznych', 'Swietochlowice, SL', 'Kawaler', 'jedzenie z McDonald's, mu
zyka, zabawa', 'ziomali, gitarzystow'>;
'>
'>
'>
'>
'>
```

Za każdym razem, gdy
Grzesiek naciska klawisz
Enter, pojawia się ten sam
znak monitu: >.



WYSIL
SZARE KOMÓRKI

Jak myślisz, co jest przyczyną problemów Grześka?



Hm... Spójrzcie na ten apostrof, który cały czas pojawia się przed znakiem monitu. Założę się, że w naszym poleceniu **INSERT** jest jakiś problem właśnie z apostrofami.

Niedopasowane apostrofy

Dokładnie! Kiedy Grzesiek próbował wykonać polecenie SQL, program, który je analizował, oczekując parzystej liczby apostrofów — po jednym przed i za każdą wartością typu CHAR, VARCHAR oraz DATE. Jednak nazwa sieci fast foodów **McDonald's** umieszczona w polu zainteresowań utrudniła sprawę, gdyż pojawił się w niej dodatkowy apostrof. A zatem system zarządzania bazami danych wciąż oczekuje na drugi apostrof do pary.



Czy jesteś w stanie odzyskać kontrolę nad oknem konsoli?

Kiedy to zrobisz, w konsoli pojawi się komunikat o błędzie. Na szczęście będziesz już mógł poprawić i ponownie wykonać polecenie.

Zakończ polecenie, wpisując apostrof oraz średnik.

W ten sposób dostarczysz systemowi zarządzania bazą danych dodatkowy apostrof, którego oczekuje.

Po dopisaniu dodatkowego apostrofu i średnika zostanie wyświetlony komunikat o błędzie. A zatem będziesz musiał jeszcze raz wpisać całe, prawidłowe polecenie **INSERT**.

Wpisanie dodatkowego apostrofu i średnika kończy polecenie SQL **INSERT**.

```
ca Wiersz polecenia - mysql -u root -p
mysql> INSERT INTO moje_kontakty
-> VALUES
-> ('Fraszynski', 'Stefan', 's.fraszynski@megainpreza.com.pl', 'DJ, animator
imprez muzycznych', 'Świetochłowice, SL', 'Kawaler', 'jedzenie z McDonald's, mu
zyka, zabawa', 'ziomali, gitarzystow');
'>
'>;
'>;
'>;
'>';

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near 's, mu
ds, muzyka, zabawa', 'ziomali, gitarzystow';
;

' at line 3
mysql> -
```

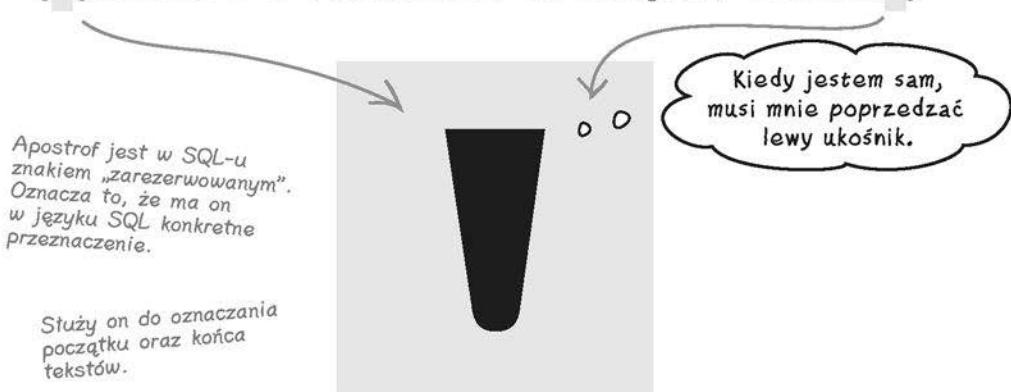
Komunikat dosyć dokładnie opisuje przyczyny zaistniałych problemów
– Błąd składni SQL;
sprawdź dokumentację używanej wersji serwera MySQL, by określić składnię, jakiej należy użyć w miejscu...
Otóż część zapytania, zaczynająca się od dodatkowego apostrofu, została potraktowana jako ciąguch znaków stanowiący wartość kolumny.

Chociaż rekord nie został zapisany w tabeli bazy danych, to jednak znak „,” wyświetlony na początku wiersza pokazuje, że program konsoli ponownie reaguje na polecenia.

Apostrofy są znakami specjalnymi

Kiedy próbujesz zapisać w kolumnie typu VARCHAR, CHAR lub BLOB tekst zawierający apostrof, to musisz poinformować system zarządzania bazami danych, że znak ten *należy do zawartości wiersza* i powinien być w nim zapisany. Jednym z rozwiązań jest poprzedzenie apostrofu znakiem **lewego ukośnika**.

```
INSERT INTO moje_kontakty  
(zainteresowania)  
VALUES  
('jedzenie z McDonald\'s, muzyka, zabawa')
```



P: Czy to jest dokładnie ten sam apostrof?

O: Tak, to jest dokładnie ten sam apostrof. Jednak w języku SQL ma on ściśle określone przeznaczenie. Informuje on oprogramowanie do zarządzania bazami danych, że dane umieszczone pomiędzy dwoma apostrofami są danymi tekstowymi.

P: Jakie typy danych wymagają zastosowania apostrofów?

O: Wszystkie dane tekstowe. Dane tekstowe to dane zapisywane w kolumnach VARCHAR, CHAR, BLOB oraz TIMEDATE. Czyli wszystko, co nie jest liczbą.

P: Czy dane umieszczane w kolumnach DEC oraz INT też powinny być zapisywane w apostrofach?

O: Nie. W przypadku kolumn liczbowych w zapisywanych w nich danych nie ma żadnych odstępów, zatem bez większych problemów można określić, gdzie takie dane się zaczynają i gdzie kończą.

P: A zatem apostrofy są używane wyłącznie do oznaczania wartości zapisywanych w kolumnach tekstowych?

O: Owszem. Kłopot w tym, że w takich wartościach mogą się pojawiać znaki odstępu. Problemy także pojawiają się w przypadkach, gdy dane tekstowe zawierają apostrofy SQL nie potrafi odróżnić apostrofu umieszczonego

wewnętrznie tekstu od apostrofu określającego, gdzie zawartość kolumny się zaczyna i kończy.

P: Czy nie można by ułatwić bazie danych rozróżnienia początku i końca wartości kolumny tekstowej poprzez oznaczanie ich przy użyciu znaków cudzysłowu, a nie apostrofów?

O: Nie. Jeśli używasz polecenia SQL w jakimś języku programowania, takim jak PHP, który stosowaliśmy w przykładach zamieszczonych w dalszej części książki, to nie należy stosować w nich cudzysłowów. W przypadku pisania programów znaki cudzysłowu informują: „tutaj zaczyna się polecenie SQL”, a dzięki temu apostrofy mogą zostać zinterpretowane jako fragment polecenia, a nie element języka programowania.

Polecenia **INSERT** z danymi zawierającymi apostrofy

A zatem musimy w jakiś sposób poinformować system zarządzania bazami danych, że apostrof nie oznacza początku ani końca łańcucha znaków, lecz wchodzi w jego skład.

Rozwiążanie problemu apostrofów poprzez poprzedzenie ich znakiem lewego ukośnika

Problem można rozwiązać, poprzedzając znak apostrofu wchodzący w skład łańcucha znaków znakiem lewego ukośnika.

Rozwiążanie to pozwoli nam poprawić błędne polecenie SQL przedstawione kilka stron wcześniej.

```
INSERT INTO moje_kontakty
```

```
VALUES
```

```
('Fraszynski', 'Stefan', 's.fraszynski@megaimpreza.com.pl',  
'DJ, animator imprez muzycznych', 'Swietochlowice, SL',  
'Kawaler', 'jedzenie z McDonald\'s, muzyka, zabawa',  
'ziomali, gitarzystow');
```

Oznaczenie, że znak apostrofu należy do łańcucha znaków, poprzez poprzedzenie go znakiem lewego ukośnika.

Rozwiążanie problemu apostrofów poprzez poprzedzenie ich dodatkowym znakiem apostrofu

Kolejnym sposobem rozwiązywania problemu z apostrofami w tekście jest poprzedzanie ich dodatkowym znakiem apostrofu.

```
INSERT INTO moje_kontakty
```

```
VALUES
```

```
('Fraszynski', 'Stefan', 's.fraszynski@megaimpreza.com.pl',  
'DJ, animator imprez muzycznych', 'Swietochlowice, SL',  
'Kawaler', 'jedzenie z McDonald''s, muzyka, zabawa',  
'ziomali, gitarzystow');
```

Znak apostrofu można także poprzedzić dodatkowym znakiem apostrofu.



Jakie inne znaki mogą powodować te same problemy?



Ćwiczenie

Jeśli w Twojej tabeli pojawiają się dane, które mogą zawierać apostrofy, to może się zdarzyć, że będziesz musiał je wyszukać. Aby zastosować w poleceniu SELECT klauzulę WHERE z danymi zawierającymi apostrof, będziesz musiał poprzedzić go lewym ukośnikiem. Dokładnie tak samo jak w przypadku zapisywania danych.

Zapisz poniższe polecenie, wykorzystując dwie poznane na poprzedniej stronie metody poprzedzania apostrofów.

```
SELECT * FROM moje_kontakty  
WHERE  
zainteresowania = 'jedzenie z McDonald's,  
muzyka, zabawa'
```

1

.....

.....

.....

2

.....

.....

.....

Która z tych metod wolisz?



Jeśli w Twojej tabeli pojawiają się dane, które mogą zawierać apostrofy, to może się zdarzyć, że będziesz musiał je wyszukać. Aby zastosować w poleceniu SELECT klauzulę WHERE z danymi zawierającymi apostrof, będziesz musiał poprzedzić go lewym ukośnikiem. Dokładnie tak samo jak w przypadku zapisywania danych.

Zapisz poniższe polecenie, wykorzystując dwie poznane na poprzedniej stronie metody poprzedzania apostrofów.

```
SELECT * FROM moje_kontakty  
WHERE  
zainteresowania = 'jedzenie z McDonald's,  
muzיקה, zabawa'
```

1

```
.....  
SELECT * FROM moje_kontakty
```

```
.....  
WHERE
```

```
.....  
zainteresowania = 'jedzenie z McDonald's, muzika, zabawa'
```

Metoda numer 1
— lewy ukośnik

2

```
.....  
SELECT * FROM moje_kontakty
```

```
.....  
WHERE
```

```
.....  
zainteresowania = 'jedzenie z McDonald"s, muzika, zabawa'
```

Metoda numer 1
— dodatkowy znak apostrofu

Wyszukiwanie konkretnych danych

Wiesz już, w jaki sposób można używać polecenia SELECT do pobierania wszelkiego typu danych, w tym także tych, które wymagają stosowania apostrofów, oraz takich, w których apostrofy mogą występować wewnątrz wartości kolumn.

Chwileczkę. Za każdym razem, gdy wykonuję polecenie SELECT *, wyświetlanych danych nie da się odczytać, gdyż rekordy są za długie, nie mieszczą się w oknie konsoli, przez co są wyświetlane w kilku wierszach – w efekcie na ekranie panuje totalny chaos. Czy mogę w jakiś sposób ukryć wszystkie niepotrzebne kolumny w przypadkach, gdy interesuje mnie jedynie na przykład adres poczty elektronicznej wybranej osoby?



Musisz wiedzieć, jak pobierać dane jedynie z wybranych kolumn.

W takich sytuacjach potrzebujemy jedynie nieco większej precyzji. Spróbujmy nieznacznie ograniczyć zwracane informacje. W tym przypadku będzie to oznaczać wyświetlenie mniejszej ilości kolumn. Poprosimy bazę danych o wyświetlenie tylko tych kolumn, które nas interesują.



Ćwiczenie

Wypróbuj to w domu

Zanim spróbujesz wykonać poniższe polecenie SELECT, naszkicuj, jak według Ciebie będzie wyglądać wynikowa tabela danych. (Jeśli musisz sobie przypomnieć strukturę tabeli proste_drinki, znajdziesz ją na stronie 93).

Zastąpiliśmy znak * nazwami trzech kolumn.

```
SELECT nazwa, składnik_główny, składnik_dodatkowy
FROM proste_drinki
WHERE składnik_główny = 'woda sodowa';
```



Wypróbuj to w domu

Zanim spróbujesz wykonać poniższe polecenie SELECT, naszkicuj, jak według Ciebie będzie wyglądać wynikowa tabela danych.

nazwa	skladnik_glowny	skladnik_dodatkowy
Blue Moon	woda sodowa	sok z jagód
Lone Tree	woda sodowa	sok wiśniowy
Greyhound	woda sodowa	sok z grapefruitu
Soda and It	woda sodowa	sok z winogron

Stary sposób

SELECT * FROM proste_drinki

Polecenie zwróciło wszystkie kolumny, przez co wyniki są zbyt szerokie i nie mieścią się w oknie konsoli. Tekst, który się nie mieści, jest przenoszony do następnego wiersza, przez co w prezentowanych wynikach panuje straszny batagan.

```
Wiersz polecenia - mysql -u root -p
mysql> SELECT * FROM proste_drinki;
+-----+-----+-----+-----+-----+-----+
| nazwa          | skladnik_glowny      | ilosc1 | skladnik_dodatkowy | ilosc2 | wskazowki
+-----+-----+-----+-----+-----+-----+
| Blackthorn     | tonik                 | 45.0   | sok ananasowy      | 30.0   | wymieszac z lodem
| odcedzic do szklanki koktailowej z plasterkiem cytryny |
| Blue Moon      | woda sodowa          | 45.0   | sok z jagod         | 22.0   | wymieszac z lodem
| odcedzic do szklanki koktailowej z plasterkiem cytryny |
| Oh My Gosh     | nektar brzoskwienny | 30.0   | sok ananasowy      | 30.0   | wymieszac z lodem
| odcedzic do wysokiej szklanki
| Lime Fizz       | Sprite                | 45.0   | sok z cytryny       | 22.0   | wymieszac z lodem
| odcedzic do szklanki koktailowej
| Kiss on the Lips | sok wiśniowy         | 60.0   | nektar morelowy    | 210.0  | podawac z lodem i
| rurka do picia
| Hot Gold        | nektar brzoskwienny | 90.0   | sok pomaranczowy   | 180.0  | wlac goracy sok p
| omaranczowy do kubka, dolac do niego nektaru brzoskwiennego
| Lone Tree       | woda sodowa          | 45.0   | sok wiśniowy       | 22.0   | wymieszac z lodem
| odcedzic do szklanki koktailowej
| Grayhound       | woda sodowa          | 45.0   | sok grapefruitowy  | 150.0  | podawac z lodem,
| dobrze wymieszac
| Indian Summer   | sok jablkowy         | 60.0   | goraca herbata    | 180.0  | wlac sok do kubka
| dodac hrebaty
| Bull Frog        | mrozona herbata      | 45.0   | lemoniada          | 150.0  | podawac z lodem i
| plasterkiem cytryny
| Soda and It      | woda sodowa          | 60.0   | sok z winogron     | 30.0   | wstrzasnac w szkl
| ance koktailowej, podawac bez lodu
+-----+-----+-----+-----+-----+-----+
11 rows in set <0.00 sec>

mysql> _
```

Pobieranie konkretnych kolumn w celu ograniczenia wyników

O określając, które kolumny mają być zwrócone przez nasze zapytanie, możemy uzyskać wyłącznie interesujące nas wartości. Klauzula WHERE pozwala nam na ograniczenie ilości zwracanych wierszy, natomiast prezentowane tu rozwiązanie — na ograniczenie ilości kolumn. Wszystko polega na tym, by to SQL i baza danych wykonaly za nas brudną robotę.

**SELECT nazwa, składnik_główny, składnik_dodatkowy
FROM proste_drinki;**

... jednak jesteśmy w stanie ograniczyć ilość wyników, wybierając kolumny, których wartości mają być zwrócone.

```
Wiersz polecenia - mysql -u root -p
mysql> SELECT nazwa, składnik_główny, składnik_dodatkowy FROM proste_drinki;
+ nazwa + składnik_główny + składnik_dodatkowy +
| Blackthorn | tonik | sok ananasowy |
| Blue Moon | woda sodowa | sok z jagod |
| Oh My Gosh | nektar brzoskwiśniowy | sok ananasowy |
| Line Fizz | Sprite | sok z cytryny |
| Kiss on the Lips | sok wiśniowy | nektar morelowy |
| Hot Gold | nektar brzoskwiśniowy | sok pomarańczowy |
| Lone Tree | woda sodowa | sok wiśniowy |
| Grayhound | woda sodowa | sok grapefruitowy |
| Indian Summer | sok jabłkowy | gorąca herbatka |
| Bull Frog | mrożona herbata | lemoniada |
| Soda and It | woda sodowa | sok z winogron |
11 rows in set (0.00 sec)

mysql>
```

Określanie kolumn w celu zwiększenia szybkości zapytania

Określanie zwracanych kolumn jest dobrą praktyką programistyczną, którą warto stosować; co więcej, zapewnia ona także inne korzyści. W miarę powiększania się tabel określanie konkretnych kolumn, które mają być zwrócone, może także przyspieszyć uzyskiwanie wyników. Ten wzrost szybkości realizacji zapytań można także zauważać podczas stosowania SQL-a wraz z innymi językami programowania, takimi jak PHP.



Wiele sposobów dotarcia do informacji o drinku „Kiss on the Lips”

Czy pamiętasz jeszcze naszą tabelę `proste_drinki`? Poniższe zapytanie zwróci między innymi nazwę drinka „Kiss on the Lips”.

```
SELECT nazwa FROM proste_drinki  
WHERE  
skladnik_glowny = "sok wisniowy";
```

Na następnej stronie uzupełnij cztery inne polecenia SQL, które także zwrócią nazwę drinka „Kiss on the Lips”.

`proste_drinki`

nazwa	skladnik_glowny	ilosc1	skladnik_dodatkowy	ilosc2	wskazowki
Blackthorn	tonik	45	sok ananasowy	30	wymieszać z lodem, odcedzić do szklanki koktajlowej z plasterkiem cytryny
Blue Moon	woda sodowa	45	sok z jagód	22	wymieszać z lodem, odcedzić do szklanki koktajlowej z plasterkiem cytryny
Oh My Gosh	nektar brzoskwiniowy	30	sok ananasowy	30	wymieszać z lodem, odcedzić do wysokiej szklanki
Lime Fizz	Sprite	45	sok z cytryny	22	wymieszać z lodem, odcedzić do szklanki koktajlowej
Kiss on the Lips	sok wiśniowy	60	nektar morelowy	210	podawać z lodem i słomką do picia
Hot Gold	nektar brzoskwiniowy	90	sok pomarańczowy	180	wlać gorący sok pomarańczowy do kubka, dodać do niego nektaru brzoskwiniowego
Lone Tree	woda sodowa	45	sok wiśniowy	22	wymieszać z lodem, odcedzić do szklanki koktajlowej
Greyhound	woda sodowa	45	sok z grapefruita	150	podawać z lodem, dobrze wymieszać
Indian Summer	sok jabłkowy	60	gorąca herbata	180	wlać sok do kubka, dodać herbaty
Bull Frog	mrożona herbata	45	limonada	150	podawać z lodem i plasterkiem cytryny
Soda and It	woda sodowa	60	sok z winogron	30	wstrząsnąć w szklance koktajlowej, podawać bez lodu

SELECT

WHERE

SELECT

WHERE

SELECT

WHERE

SELECT

WHERE

A teraz napisz trzy polecenia SQL, które pozwolą zwrócić nazwę „Bull Frog”.

1

.....

.....

2

.....

.....

3

.....

.....

Rozwiążanie zadania

Zaostrz ołówek

Rozwiążanie

Uzupełnij cztery inne polecenia SQL, które także zwrócią nazwę drinka „Kiss on the Lips”.

SELECT nazwa **FROM** proste_drinki

WHERE składnik_dodatkowy = 'nektaR morelowy';

SELECT nazwa **FROM** proste_drinki

WHERE ilosc2 = 210;

SELECT nazwa **FROM** proste_drinki

WHERE wskazowki = 'podawać z lodem i slomką do picia';

SELECT nazwa **FROM** proste_drinki

WHERE nazwa = 'Kiss on the Lips';

Takiego zapytania pewnie rzadko kiedy będziesz używać, niemniej jednak daje ono oczekiwane wyniki. Takiego polecenia możesz natomiast używać, aby na przykład upewnić się, że tekst podany w kolumnie z nazwą drinka nie zawiera żadnych błędów typograficznych.

A teraz napisz trzy polecenia SQL, które pozwolą zwrócić nazwę „Bull Frog”.

1 **SELECT** nazwa **FROM** proste_drinki

..... **WHERE** składnik_główny = 'mrożona herbata'

2 **SELECT** nazwa **FROM** proste_drinki

..... **WHERE** składnik_dodatkowy = 'lemoniada'

3 **SELECT** nazwa **FROM** proste_drinki

..... **WHERE** wskazówki = 'podawać z lodem i plasterkiem cytryny'



CELNE SPOSTRZEŻENIA

- ◆ Określając w klauzuli WHERE warunki dotyczące pól tekstowych, stosuj apostrofy.
- ◆ Nie stosuj apostrofów, jeśli warunki używane w klauzuli WHERE operują na polach liczbowych.
- ◆ Jeśli chcesz zwrócić wartości wszystkich kolumn, za słowem kluczowym SELECT umieść *.

- ◆ Jeśli wydałeś polecenie SQL, lecz wszystko wskazuje na to, że system zarządzania bazami danych nie wykonał go, to sprawdź, czy w poleceniu nie brakuje jakiegoś apostrofu.
- ◆ Jeśli tylko możesz, to staraj się pobierać wartości z konkretnych kolumn, a nie ze wszystkich.

Nie istniejąca grupa pytań

P: Co zrobić w sytuacji, gdy polecenie musi zwrócić wszystkie kolumny tabeli? Czy powiniensem podać ich nazwy w poleceniu SELECT, czy użyć znaku *?

O: Jeśli potrzebujesz wszystkich kolumn, to absolutnie powinieneś zastosować znak *. Staraj się go unikać wyłącznie w przypadkach, gdy interesuje Cię jedynie kilka konkretnych kolumn tabeli.

P: Skopiowałem polecenie SQL z internetu i spróbowałem je wykonać na swoim komputerze, jednak cały czas pojawiały się jedynie komunikaty o błędach. Czy robię coś źle?

O: Polecenia skopiowane z internetu mogą czasami zawierać niewidoczne znaki wyglądające zupełnie jak odstępy, lecz mające całkowicie inne znaczenie dla języka SQL. Jednym ze sposobów rozwiązania problemu jest wklejenie takiego polecenia do edytora tekstu. A zatem chcąc użyć polecenia skopiowanego z internetu, najlepiej zrobisz, jeśli najpierw wkleisz go do edytora i dokładniej mu się przyjrzyisz.

P: A zatem powiniensem wklejać polecenia SQL do takiego programu jak Microsoft Word?

O: Nie. Word nie jest dobrym programem do tego celu, gdyż nie będzie w żaden widoczny sposób wyświetlał znaków specjalnych, które ewentualnie będziesz musiał usunąć z polecenia. Znacznie lepiej do tego celu będzie się nadawać zwyczajny Notatnik.

P: A jeśli chodzi o poprzedzanie znaku apostrofu... Czy istnieją jakieś powody przemawiające za zastosowaniem jednej z dwóch dostępnych metod?

O: Raczej nie. My preferujemy zastosowanie znaku ukośnika, jednak tylko dlatego, gdyż w tym przypadku w razie jakichkolwiek problemów łatwiej jest zauważyc miejsca, gdzie wewnątrz tekstu są umieszczone dodatkowe apostrofy. Na przykład łatwiej jest szybko przeanalizować następujący tekst:

'Nie lubie jadac w Mc\ 'Donalds'

niż tekst o postaci:

'Nie lubie jadac w Mc' 'Donalds'

Jednak oprócz tego nie ma powodów przemawiających za jedną bądź drugą z tych metod. Obie zapewniają identyczne możliwości — umieszczanie apostrofów wewnątrz kolumn tekstowych.

Pączek pyta, co Twoja tabela jest w stanie dla Ciebie zrobić...

Aby odszukać najlepszy pączek z lukrem zarejestrowany w tabeli, musisz wykonać co najmniej dwa polecenia SELECT. Pierwsze z nich wybierze z tabeli pączki odpowiedniego typu. Z kolei drugie polecenie znajdzie pączki z oceną równą 10.



Chciałabym odszukać najlepszy pączek z lukrem bez konieczności przeglądania tych wszystkich wyników.

miejsce	godzina	data	typ	ocena	komentarze
Kafeteria Gwiezdny Pył	7.43	23.4	cynamonowy z lukrem	6	zbyt dużo przypraw
Pączki u Donalda	8.56	25.8	zwyczajny z lukrem	5	tłusty
Pączki u Donalda	7.58	26.4	z dżemem	6	nie świeży, ale smaczny
Kafeteria Gwiezdny Pył	10.35	24.4	zwyczajny z lukrem	7	ciepły, ale nie gorący
Chrupki Król	9.38	26.9	z dżemem	6	za mało dżemu
Kafeteria Gwiezdny Pył	7.48	23.4	z dżemem i lukrem	10	doskonali!
Chrupki Król	8.56	25.11	zwyczajny z lukrem	8	lukier z syropu klonowego
Pączki Donald	02	25.5	z dżemem	8	alkohol i smaczne

Wyobraź sobie, że ta tabela zawiera 10 tysięcy rekordów.

- 1 Jednym ze sposobów jest wyszukanie wszystkich pączków odpowiedniego typu:

SELECT miejsce, ocena FROM paczki_oceny

WHERE

typ = 'zwyczajny z lukrem';

Zapytanie musi zwrócić oceny, byśmy mogli wybrać najlepszy pączek; oraz miejsca, abyśmy mogli określić ciastkarnię lub sklep, w którym są sprzedawane najlepsze pączki.

Wszystkie wyniki będą dotyczyć pączków odpowiedniego typu.

Wyniki pierwszego zapytania. Wyobraź sobie, że są ich setki...

miejsce	ocena
Pączki u Donalda	5
Kafeteria Gwiezdny Pył	7
Chrupki Król	8
Kafeteria Gwiezdny Pył	10
Pączki Donald	8

Zapytaj, co możesz zrobić dla swojego pączka

- 2 Możesz także wyszukać pączki z najwyższą oceną:

SELECT miejsce , typ FROM paczki_oceny

WHERE

ocena = 10;

W wynikach pojawiają się wyłącznie pączki z najwyższą oceną.

Musisz teraz znaleźć pączki odpowiedniego typu, a nazwa określi zwycięzcę.

miejsce	typ
Kafeteria Gwiezdny Pył	z dżemem i lukrem
Chrupki Król	zwyczajny z lukrem
Kafeteria Gwiezdny Pył	zwyczajny z lukrem
Paczki u Donalda	z dżemem i lukrem

Wyniki drugiego zapytania. Także w tym przypadku wyobraź sobie, że rekordów są setki...

Tak naprawdę to takie zapytania wcale wiele nie pomagają. Mogłabym w ogóle ich nie zadawać... ale ta tabela zawiera tysiące rekordów... jestem głodna i chcę mojego pączka już teraz!



WYSIL SZARE KOMÓRKI

A teraz zastanów się i powiedz na głos, jaki jest cel wykonywania obu tych poleceń SQL? Jakie warunki muszą spełniać ostateczne wyniki?

Łączenie zapytań

Łączenie zapytań

Nic nie stoi na przeszkodzie, byśmy połączyli oba kryteria wyboru rekordów — typ oraz ocenę — i umieścili je w jednym zapytaniu. Możemy to zrobić dzięki zastosowaniu słowa kluczowego AND. W tym przypadku wyniki zwrócone przez zapytanie będą spełniały oba podane warunki.

SELECT miejsce

Teraz pozostaje nam jedynie wybranie odpowiedniego miejsca.

FROM paczki_oceny

WHERE typ = 'zwyczajny z lukrem'

AND

Użyj słowa kluczowego AND, by połączyć warunki z obu klauzuł WHERE.

ocena = 10;

Oto wyniki zapytania wykorzystującego słowo kluczowe AND. Nawet gdyby wyniki zawierały więcej niż jeden wiersz, to i tak będziemy wiedzieć, że wszystkie reprezentują ciastkarnie, gdzie można dostać zwyczajne paczki z lukrem, które otrzymały najwyższą możliwą ocenę.

miejsce	ocena
Paczki u Donalda	5
Kafeteria Gwiezdny Pył	7
Chrupki Król	8
Kafeteria Gwiezdny Pył	10

AND

miejsce	typ
Kafeteria Gwiezdny Pył	z dżemem i lukrem
Chrupki Król	zwyczajny z lukrem
Kafeteria Gwiezdny Pył	zwyczajny z lukrem

To zapytanie zwraca wyniki spełniające oba kryteria — typ 'zwyczajny z lukrem' i ocena równa 10.



miejsce
Kafeteria Gwiezdny Pył

Mamo? Czy możemy pójść do kafeterii Gwiezdny Pył?





Ćwiczenie

A zatem mogłem odszukać Anię,
używając AND?

Używając tabeli `moje_kontakty`, napisz kilka zapytań dla Grześka. Zwracaj wartości tylko tych kolumn, które są Ci naprawdę potrzebne. I zwracaj baczną uwagę na apostrofy.

Napisz zapytanie, które zwróci adresy poczty elektronicznej osób zajmujących się programowaniem.

.....
.....
.....
.....



Napisz zapytanie, które zwróci imię oraz miejsce zamieszkania wszystkich osób urodzonych tego samego dnia co Ty.

.....
.....
.....
.....

Napisz zapytanie, które zwróci imiona i adresy poczty elektronicznej wszystkich samotnych osób mieszkających w Twoim mieście. Aby zdobyć dodatkowe punkty, zwróć tylko osoby tej płci, z którymi chciałbyś chodzić na randki.

.....
.....
.....
.....

Napisz zapytanie, którego Grzesiek mógł użyć do odszukania wszystkich Ani mieszkających we Wrocławiu.

.....
.....
.....
.....

Rozwiążanie ćwiczenia



Rozwiążanie ćwiczenia

Używając tabeli `moje_kontakty`, napisz kilka zapytań dla Grześka. Zwracaj wartości tylko tych kolumn, które są Ci naprawdę potrzebne. I uważaj na apostrofy.

Napisz zapytanie, które zwróci adresy poczty elektronicznej osób zajmujących się programowaniem.

Chcemy wyświetlić wyłącznie kolumnę `email`:

```
SELECT email FROM moje_kontakty  
WHERE zawód = 'programista';
```

Interesują nas osoby z zawodem 'programista'.

Napisz zapytanie, które zwróci imię oraz miejsce zamieszkania wszystkich osób urodzonych tego samego dnia co Ty.

```
SELECT nazwisko, imię, lokalizacja FROM  
moje_kontakty  
WHERE data_urodzenia = '1978-07-21';
```

Tu powinieneś wpisać swoją datę urodzenia.

Napisz zapytanie, które zwróci imiona i adresy poczty elektronicznej wszystkich samotnych osób mieszkających w Twoim mieście. Aby zdobyć dodatkowe punkty, zwróć tylko osoby tej płci, z którymi chciałbyś chodzić na randki.

```
SELECT nazwisko, imię, email  
FROM moje_kontakty  
WHERE lokalizacja = 'Gliwice, Śl'  
AND płeć = 'K';
```

Tu wpisz miejscowości, w której mieszkasz, i kod województwa.

A tu płeć zgodna ze swoimi preferencjami.

Napisz zapytanie, którego Grzesiek mógł użyć do odszukania wszystkich Ani mieszkającej we Wrocławiu.

```
SELECT nazwisko, imię, email  
FROM moje_kontakty  
WHERE lokalizacja = 'Wrocław, DS'  
AND imię = 'Anna'
```

Odnajdywanie wartości liczbowych

Załóżmy, że używając tylko jednego polecenia SQL, chciałbyś odszukać w tabeli `proste_drinki` wszystkie drinki, które zawierają co najmniej 30 mililitry wody sodowej. Oto trudny sposób dotarcia do poszukiwanych informacji. Można zastosować dwa zapytania:

Interesują nas wyłącznie nazwy drinków.

```
SELECT nazwa FROM proste_drinki
WHERE
    skladnik_glowny = 'woda sodowa'
    AND
    ilosc1 = 45
```

Drinki na bazie wody sodowej, przy czym ilość wody wynosi 45 mililitrów.

```
C:\ Wiersz poleceń - mysql -u pio -p
mysql> SELECT nazwa FROM proste_drinki WHERE skladnik_glowny = 'woda sodowa' AND ilosc1 = 45;
+ nazwa
| Blue Moon
| Lone Tree
| Grayhound
+-----+
3 rows in set (0.00 sec)

mysql>
```

SELECT nazwa FROM proste_drinki
WHERE
 skladnik_glowny = 'woda sodowa'
 AND
 ilosc1 = 60

Drinki na bazie wody sodowej, przy czym ilość wody wynosi 60 mililitrów.

```
C:\ Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT nazwa FROM proste_drinki WHERE skladnik_glowny = 'woda sodowa' AND ilosc1 = 60;
+ nazwa
| Soda and It
+-----+
1 row in set (0.00 sec)

mysql>
```

Zastosowanie operatorów porównania



Czyż nie byłoby super,
gdybym mogła w jednym zapytaniu
odszukać w tabeli proste_drinki
wszystkie drinki, które zawierają więcej
niz 30 mililitrów wody sodowej? Ale wiem,
że to jedynie marzenia...

proste_drinki

nazwa	skladnik_glowny	ilosc1	skladnik_dodatkowy	ilosc2	wskazowki
Blackthorn	tonik	45	sok ananasowy	30	wymieszać z lodem, odcedzić do szklanki koktajlowej z plasterkiem cytryny
Blue Moon	woda sodowa	45	sok z jagód	22	wymieszać z lodem, odcedzić do szklanki koktajlowej z plasterkiem cytryny
Oh My Gosh	nekter brzoskwiniowy	30	sok ananasowy	30	wymieszać z lodem, odcedzić do wysokiej szklanki
Lime Fizz	Sprite	45	sok z cytryny	22	wymieszać z lodem, odcedzić do szklanki koktajlowej
Kiss on the Lips	sok wiśniowy	60	nekter morelowy	210	podawać z lodem i słomką do picia
Hot Gold	nekter brzoskwiniowy	90	sok pomarańczowy	180	wlać gorący sok pomarańczowy do kubka, dodać do niego nektaru brzoskwiniowego
Lone Tree	woda sodowa	45	sok wiśniowy	22	wymieszać z lodem, odcedzić do szklanki koktajlowej
Greyhound	woda sodowa	45	sok z grapefruita	150	podawać z lodem, dobrze wymieszać
Indian Summer	sok jabłkowy	60	gorąca herbata	180	wlać sok do kubka, dodać herbaty
Bull Frog	mrożona herbata	45	lemoniada	150	podawać z lodem i plasterkiem cytryny
Soda and It	woda sodowa	60	sok z winogron	30	wstrząsnąć w szklance koktajlowej, podawać bez lodu

Jeden raz wystarczy

Chyba nikogo nie trzeba przekonywać, że stosowanie dwóch zapytań jest stratą czasu, a co ważniejsze, korzystając z takiego rozwiązania, można pominąć drinki, które zawierają na przykład 40 lub 50 mililitrów wody sodowej.

Rozwiązaniem naszego problemu jest zastosowanie **znaku większości**:



SELECT nazwa FROM proste_drinki

WHERE

skladnik_glowny = 'woda sodowa'

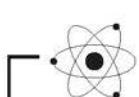
AND

ilosc1 > 30

Zastosowanie OPERATORA WIĘKSZOŚCI pozwoli nam wyszukać wszystkie drinki, które zawierają więcej niż 30 mililitrów wody sodowej.

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT nazwa FROM proste_drinki WHERE
-> skladnik_glowny = 'woda sodowa' AND ilosc1 > 30;
+-----+
| nazwa |
+-----+
| Blue Moon
| Lone Tree
| Grayhound
| Soda and It |
+-----+
4 rows in set (0.00 sec)

mysql> -
```



WYSIL
SZARE KOMÓRKI

A czy nie można by połączyć dwóch przedstawionych zapytań, wykorzystując dodatkowe słowo AND?

~~Łagodne operatory porównania~~

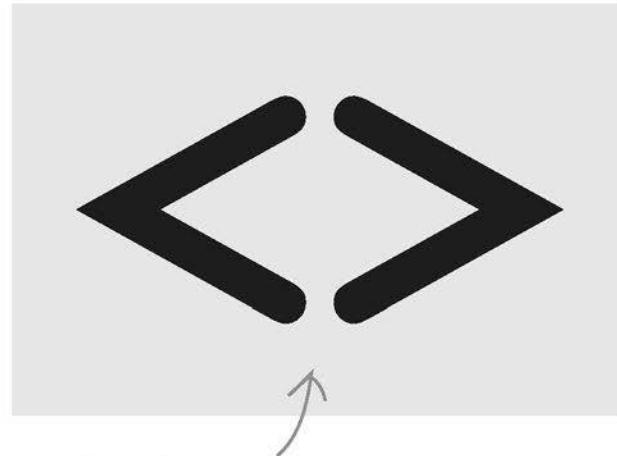
Jak do tej pory w tworzonych przez nas klauzulach WHERE używaliśmy wyłącznie **znaku równości**. Na poprzedniej stronie poznałeś **znak większości >**. Poniżej prezentujemy pozostałe operatory porównania:

Znak równości poszukuje wartości, które są identyczne. Nie przyda się on, jeśli chcemy znaleźć wartości, które są mniejsze lub większe od pewnej wartości.

Ten mylący symbol oznacza „**różny**”. Zwracane przez niego wyniki stanowią dokładne przeciwieństwo wyników, jakie uzyskamy, stosując znak równości. Dwie wartości mogą być albo równe, albo różne od siebie.



Wszyscy znamy i kochamy znak RÓWNOŚCI.



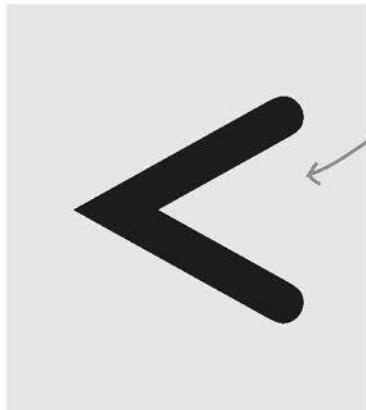
Ten symbol oznacza RÓŻNY. Zwraca on wszystkie rekordy, w których wartość w porównywanej kolumnie jest różna od drugiej podanej wartości.



**WYTĘŻ
UMYSŁ**

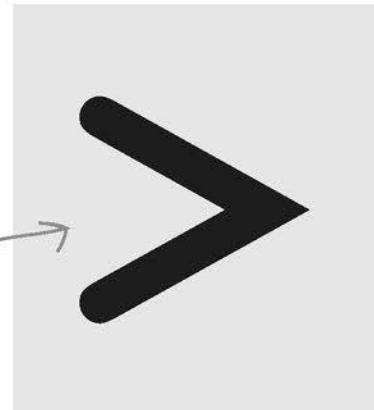
Czy zwróciłeś uwagę, że we wszystkich przedstawionych do tej pory klauzulach WHERE nazwy kolumn były umieszczane z lewej strony operatora porównania? Czy warunki działałyby poprawnie, gdyby kolumny były umieszczane po prawej stronie?

Znak **mniejszości** sprawdza, czy wartość kolumny podanej z jego lewej strony jest mniejsza od liczby umieszczonej po jego prawej stronie. Jeśli wartość kolumny jest mniejsza od wartości zapisanej po prawej stronie operatora, to analizowany wiersz zostanie dodany do wyników.

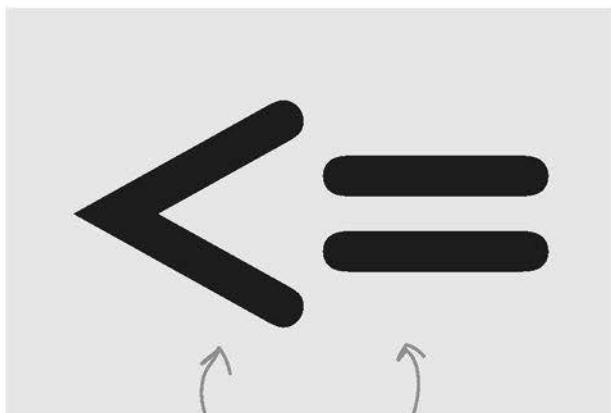


Symbol **MNIEJSZOŚCI** zwraca wszystkie wartości mniejsze od tej podanej w warunku.

A to jest oczywiście znak **WIĘKSZOŚCI**.



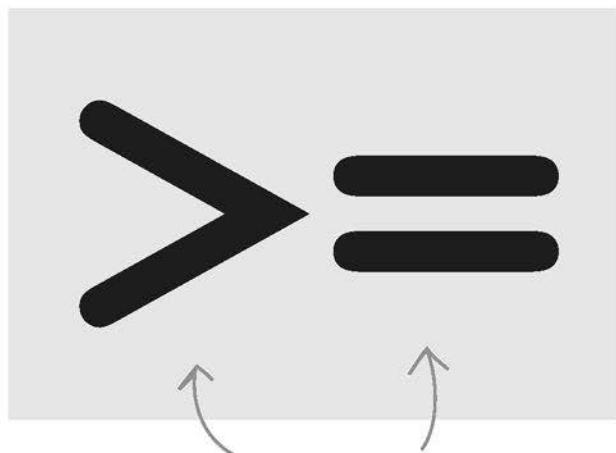
Jedyna różnica pomiędzy znakiem **mniejszy lub równy** a znakiem mniejszości polega na tym, iż w przypadku pierwszego zwracane są także rekordy, w których wartość kolumny jest nie tylko mniejsza, lecz także równa wartości podanej z prawej strony operatora.



Zwracane są wszystkie rekordy, w których wartość kolumny jest MNIEJSZA LUB RÓWNA wartości podanej w warunku.

Znak **większości** jest przeciwnieństwem znaku mniejszości. Sprawdza on, czy wartość kolumny podanej z jego lewej strony jest większa od wartości zapisanej po prawej stronie znaku. Jeśli wartość kolumny jest większa od podanej, to cały rekord zostanie dodany do wyników.

To samo dotyczy znaku **większy lub równy**. W jego przypadku do wyników będą także dołączane rekordy, w których wartość kolumny jest zarówno większa, jak i równa wartości podanej z prawej strony operatora.



Istnieje także operator **WIĘKSZY LUB RÓWNY**.

Odnajdywanie wartości liczbowych przy użyciu operatorów porównania

Salon Rusz głowę! udostępnia swym klientom tabelę zawierającą informacje o cenach oraz wartościach odżywcznych serwowanych drinków. Właściele zdecydowali się bowiem oferować drinki droższe, lecz jednocześnie mniej kaloryczne. Oczywiście wszystko w celu zwiększenia dochodów.

Właściciele używają operatorów porównania, by odszukać wszystkie drinki, które kosztują przynajmniej 8,50 zł i mają mniej niż 50 kalorii. Wszystkie niezbędne informacje są pobierane z tabeli `drinki_informacje`.

Zawartość węglowodanów
w drinku, wyrażona w gramach.

`drinki_informacje`

Ilość kalorii w drinku.

nazwa	cena	węglowodany	kolor	lód	kalorie
Blackthorn	7.50	8.4	żółty	T	33
Blue Moon	6.25	3.2	niebieski	T	12
Oh My Gosh	8.75	8.6	pomarańczowy	T	35
Lime Fizz	6.25	5.4	zielony	T	24
Kiss on the Lips	13.75	42.5	fioletowy	T	171
Hot Gold	8.00	32.1	pomarańczowy	N	135
Lone Tree	9.00	4.2	czerwony	T	17
Greyhound	10.00	14	żółty	T	50
Indian Summer	7.00	7.2	brązowy	N	30
Bull Frog	6.50	21.5	jasnobrązowy	T	80
Soda and It	9.50	4.7	czerwony	N	19

`SELECT nazwa FROM drinki_informacje`

`WHERE`

`cena >= 8.50`

`AND`

`kalorie < 50;`

Ten warunek stwierdza: „Należy odnaleźć wszystkie drinki, które kosztują 8,50 zł lub więcej. W wynikach zostaną uwzględnione także te drinki, które kosztują dokładnie 8,50 zł.”

Z kolei ten warunek stwierdza:
„Należy odnaleźć wszystkie drinki, które mają mniej niż 50 kalorii”.

Ze względu na zastosowanie operatora AND powyższe zapytanie zwróci jedynie te rekordy, które spełniają oba podane warunki. Konkretnie rzecz biorąc, zwróci ono drinki o nazwach: Oh My Gosh, Lone Tree oraz Soda and It.

Zaostrz ołówek



Najwyższa pora, żebyś sam spróbował pomieszać parę drinków. Napisz zapytania, które zwrócią informacje opisane poniżej. Nie zapomnij podać wyników, które zwrócią te zapytania.

Ceny wszystkich drinków z lodem, które mają żółty kolor i więcej niż 33 kalorie.

.....
.....
.....
.....
.....
.....

Wyniki:

Nazwy i kolory wszystkich drinków, które nie zawierają więcej niż 4 gramy węglowodanów i są serwowane z lodem.

.....
.....
.....
.....
.....
.....

Wyniki:

Ceny wszystkich drinków, których wartość energetyczna jest większa lub równa 80 kaloriom.

.....
.....
.....
.....
.....
.....

Wyniki:

Drinki o nazwach Greyhound i Kiss on the Lips wraz z informacją o ich kolorze oraz czy są serwowane z lodem czy nie.

.....
.....
.....
.....
.....
.....

Wyniki:

Rozwiążanie zadania

Zaostrz ołówek



Rozwiążanie

Najwyższa pora, żebyś sam spróbował pomieszać parę drinków. Napisz zapytania, które zwrócą informacje opisane poniżej. Nie zapomnij podać wyników, które zwrócią te zapytania.

Ceny wszystkich drinków z lodem, które mają żółty kolor i więcej niż 33 kalorie.

```
.....SELECT cena FROM drinki_informacje  
.....WHERE lod = 'T'  
.....AND  
.....kolor = 'żółty'  
.....AND  
.....kalorie > 33;
```

Wyniki: ...10,00.....

Nazwy i kolory wszystkich drinków, które nie zawierają więcej niż 4 gramy węglowodanów i są serwowane z lodem.

```
.....SELECT nazwa, kolor FROM drinki_informacje  
.....WHERE  
.....węglowodanu <= 4  
.....AND  
.....lod = 'T';
```

Wyniki: ...Blue Moon, niebieski.....

Ceny wszystkich drinków, których wartość energetyczna jest większa lub równa 80 kaloriom.

```
.....SELECT cena FROM drinki_informacje  
.....WHERE  
.....kalorie >= 80;
```

Wyniki: ...13,75, 8,00, 6,5.....

Drinki o nazwach Greyhound i Kiss on the Lips wraz z informacją o ich kolorze oraz czy są serwowane z lodem czy nie.

```
.....SELECT nazwa, kolor, lod FROM drinki_informacje  
.....WHERE  
.....cena > 9,50;
```

Wyniki: ...Greyhound, żółty, T
...Kiss on the Lips, fioletowy, T.....

Ale to rozwiązanie bazuje na zastosowaniu liczb. Czyli jeśli będę chciała znaleźć wszystkie drinki, których nazwy zaczynają się od określonej litery, to będę mieć pecha?

To zapytanie jest podchwytliwe.
Musisz przeanalizować tabelę
i znaleźć jakąś kolumnę, która
pozwoli Ci wybrać tylko i wyłącznie
dwa interesujące Cię drinki.



Odnajdywanie danych tekstowych przy użyciu operatorów porównania

Porównywanie danych tekstowych zapisywanych w kolumnach, takich jak CHAR lub VARCHAR, działa w podobny sposób. W tym przypadku operacje porównywania wykonywane są w sposób **alfabetyczny**. A zatem założymy, że chcemy pobrać wszystkie drinki, których nazwy zaczynają się na literę „L” lub „Ł”. Poniżej przedstawiliśmy zapytanie, które zwróci wszystkie drinki spełniające te kryteria.

drinki_informacje

nazwa	cena	węglowodany	kolor	lód	kalorie
Blackthorn	7.50	8.4	żółty	T	33
Blue Moon	6.25	3.2	niebieski	T	12
Oh My Gosh	8.75	8.6	pomarańczowy	T	35
Lime Fizz	6.25	5.4	zielony	T	24
Kiss on the Lips	13.75	42.5	fioletowy	T	171
Hot Gold	8.00	32.1	pomarańczowy	N	135
Lone Tree	9.00	4.2	czerwony	T	17
Greyhound	10.00	14	żółty	T	50
Indian Summer	7.00	7.2	brązowy	N	30
Bull Frog	6.50	21.5	jasnobrązowy	T	80
Soda and It	9.50	4.7	czerwony	N	19

**SELECT nazwa
FROM drinki_informacje
WHERE**

nazwa >= 'L'



To zapytanie zwraca drinki, których nazwy rozpoczynają się na literę „L” lub kolejną, ale znajdująca się w alfabetie przed literą „M”.

AND

nazwa < 'M';



**Na razie nie przejmuj się kolejnością,
w jakiej są zwarcane wyniki.**

W dalszej części książki pokażemy, w jaki sposób można sortować wyniki alfabetycznie.

Pobieranie składników drinków

Jeden z barmanów został poproszony o zrobienie drinka zawierającego sok wiśniowy. Barman może wyszukać takie drinki, używając dwóch zapytań:

The screenshot shows a Windows command-line window titled 'C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe'. It contains two SQL SELECT statements:

```
mysql> SELECT nazwa FROM proste_drinki WHERE skladnik_glowny = 'sok wiśniowy';
+-----+
| nazwa |
+-----+
| Kiss on the Lips |
+-----+
1 row in set (0.00 sec)

mysql> SELECT nazwa FROM proste_drinki WHERE skladnik_dodatkowy = 'sok wiśniowy';
+-----+
| nazwa |
+-----+
| Lone Tree |
+-----+
1 row in set (0.00 sec)

mysql>
```

A callout bubble on the left side of the window has the text: "Każde zapytanie sprawdza odrębną kolumnę tabeli." (Each query checks a separate column in the table.)

drinki_informacje

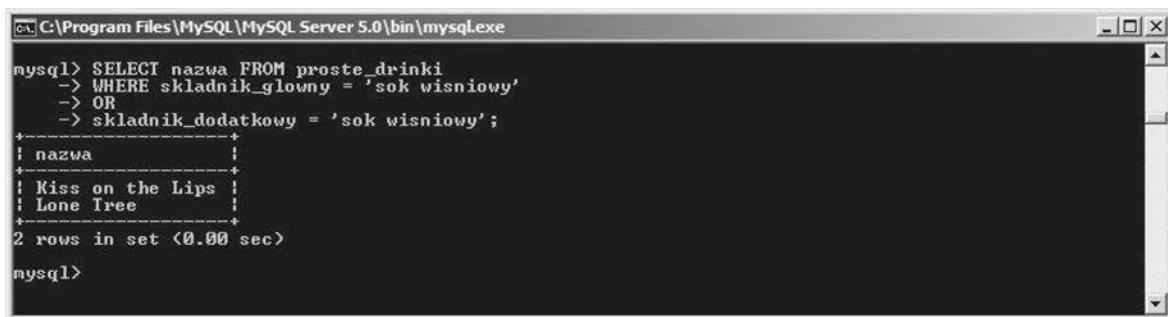
nazwa	cena	węglowodany	kolor	lód	kalorie
Blackthorn	7.50	8.4	żółty	T	33
Blue Moon	6.25	3.2	niebieski	T	12
Oh My Gosh	8.75	8.6	pomarańczowy	T	35
Lime Fizz	6.25	5.4	zielony	T	24
Kiss on the Lips	13.75	42.5	fioletowy	T	171
Hot Gold	8.00	32.1	pomarańczowy	N	135
Lone Tree	9.00	4.2	czerwony	T	17
Greyhound	10.00	14	żółty	T	50
Indian Summer	7.00	7.2	brązowy	N	30
Bull Frog	6.50	21.5	jasnobrązowy	T	80
Soda and It	9.50	4.7	czerwony	N	19

Nie wydaje mi się,
żeby to rozwiązanie było
szczególnie efektywne.
Jestem przekonany, że
musi być jakiś sposób
pozwalający na połączenie
obu tych zapytań.



Być ALBO nie być

Oba zapytania przedstawione na poprzedniej stronie można połączyć przy użyciu operatora OR. Uzyskane w ten sposób zapytanie zwróci wszystkie rekordy, które spełniają **którykolwiek** z podanych warunków. A zatem zamiast dwóch niezależnych zapytań można uzyskać jedno, w którym oba warunki zostaną połączone operatorem OR:



```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT nazwa FROM proste_drinki
-> WHERE skladnik_glowny = 'sok wisniowy'
-> OR
-> skladnik_dodatkowy = 'sok wisniowy';
+-----+
| nazwa      |
+-----+
| Kiss on the Lips |
| Lone Tree   |
+-----+
2 rows in set <0.00 sec>
mysql>
```

Zaostrz ołówek



Z dwóch przedstawionych poniżej poleceń SQL wykreś niepotrzebne fragmenty, a następnie dodaj operator OR, by zmienić je w jedno polecenie.

```
SELECT nazwa FROM proste_drinki WHERE
skladnik_glowny = 'sok pomaranczowy';
```

```
SELECT nazwa FROM proste_drinki WHERE
skladnik_glowny = 'sok jablkowy';
```

Zastosuj zdobyte właśnie umiejętności, by przekształcić dwa powyższe zapytania w jedno polecenie SELECT:

.....
.....

Rozwiążanie kolejnego ćwiczenia

Zaostrz ołówek

Rozwiążanie

Z dwóch przedstawionych poniżej poleceń SQL wykreś niepotrzebne fragmenty, a następnie dodaj operator OR, by zmienić je w jedno polecenie.

~~SELECT nazwa FROM proste_drinki WHERE
skladnik_glowny = 'sok pomaranczowy';~~

OR

~~SELECT nazwa FROM proste_drinki WHERE
skladnik_glowny = 'sok jablkowy';~~

Dzięki dodaniu tego operatora OR zapytanie zwróci wszystkie drinki, w których głównym składnikiem jest sok pomarańczowy LUB jabłkowy.

Musimy pozbyć się średnika, żeby polecenie nie zakończyło się przedwcześnie.

Ten wiersz kodu możemy po prostu skreślić — wskazanie interesującej nas tabeli oraz kolumny zatwarbia analogiczny fragment pierwszego polecenia (które teraz, dzięki zastosowaniu operatora OR, zostało połączone z drugim).

Zastosuj zdobyte właśnie umiejętności, by przekształcić dwa powyższe zapytania w jedno polecenie SELECT:

~~SELECT nazwa FROM proste_drinki
WHERE
skladnik_glowny = 'sok pomaranczowy'~~
~~OR~~
~~skladnik_glowny = 'sok jablkowy';~~

A oto i ostateczna postać zapytania.



Nie można mylić operatorów AND i OR!

Jeśli chcesz, by **WSZYSTKIE** warunki podawane w pytaniu były spełnione, zastosuj operator **AND**.

Gdy chcesz, by rekord był zwracany, jeśli spełniony zostanie **KTÓRYKOLWIEK** z warunków podanych w zapytaniu, użyj operatora **OR**.

Wciąż nie łapiesz? Zatem zerknij na następną stronę.

Nie istnieją głupie pytania

P: Czy w tej samej klauzuli WHERE można użyć większej ilości operatorów AND lub OR?

O: Oczywiście, można. Można je łączyć i używać w dowolnych kombinacjach i ilościach. Co więcej, w tej samej klauzuli WHERE można używać zarówno operatorów OR, jak i AND.

Różnica pomiędzy operatorem AND a OR

Przedstawione poniżej zapytania prezentują wszystkie możliwe kombinacje dwóch warunków połączonych operatorami OR oraz AND.

paczki_oceny

miejsce	godzina	data	typ	ocena	komentarze
Chrupki Król	8:50	27.9	zwyczajny z lukrem	10	niemal doskonaly
Pączki u Donalda	8:59	25.8	NULL	6	tłusty
Kafeteria Gwiezdny Pył	7:35	24.5	cynamonowy z lukrem	5	nieświeży, ale smaczny
Pączki u Donalda	7:03	26.4	z dżemem	7	za mało dżemu

SELECT typ FROM paczki_oceny

WYNIKI

Tak, ten warunek jest spełniony

WHERE miejsce = 'Chrupki Krol' AND ocena = 10;

Tak

zwyczajny z lukrem

WHERE miejsce = 'Chrupki Krol' OR ocena = 10;

zwyczajny z lukrem

WHERE miejsce = 'Chrupki Krol' AND ocena = 3;

Warunek niespełniony

brak wyników

WHERE miejsce = 'Chrupki Krol' OR ocena = 3;

Warunek niespełniony

zwyczajny z lukrem

WHERE miejsce = 'Pyszny Rogal' AND ocena = 10;

brak wyników

WHERE miejsce = 'Pyszny Rogal' OR ocena = 10;

zwyczajny z lukrem

WHERE miejsce = 'Pyszny Rogal' AND ocena = 3;

brak wyników

WHERE miejsce = 'Pyszny Rogal' OR ocena = 3;

brak wyników

Bądź operatorem warunkowym



Poniżej przedstawiona została grupa klauzul WHERE, w których zastosowano operatory logiczne AND i OR. Twoim zadaniem jest wcielić się w rolę tych klauzul i określić, czy zwrócią one jakieś wyniki, czy nie.

`SELECT typ FROM paczki_oceny`

Czy są jakieś wyniki?

`WHERE miejsce = 'Chrupki Krol' AND ocena <> 6;`

.....

`WHERE miejsce = 'Chrupki Krol' AND ocena = 3;`

.....

`WHERE miejsce = 'Pyszny Rogal' AND ocena >= 6;`

.....

`WHERE miejsce = 'Chrupki Krol' OR ocena > 5;`

.....

`WHERE miejsce = 'Chrupki Krol' OR ocena = 3;`

.....

`WHERE miejsce = 'Pyszny Rogal' OR ocena = 6;`

.....

Aby poprawić swoją karmę, zapisz, dlaczego w dwóch przypadkach zwrócone wyniki są inne niż w pozostałych.

Bądź operatorem warunkowym. Rozwiążanie



Poniżej przedstawiona została grupa klauzul WHERE, w których zastosowano operatory logiczne AND i OR. Twoim zadaniem jest wcielić się w rolę tych klauzul i określić, czy zwrócią one jakieś wyniki, czy nie.

SELECT typ FROM paczki_oceny

Czy są jakieś wyniki?

WHERE miejsce = 'Chrupki Krol' AND ocena < 6;

zwyczajny z lukrem

WHERE miejsce = 'Chrupki Krol' AND ocena = 3;

brak wyników

WHERE miejsce = 'Pyszny Rogal' AND ocena >= 6;

brak wyników

WHERE miejsce = 'Chrupki Krol' OR ocena > 5;

zwyczajny z lukrem, NULL, z dżarem

WHERE miejsce = 'Chrupki Krol' OR ocena = 3;

zwyczajny z lukrem

WHERE miejsce = 'Pyszny Rogal' OR ocena = 6;

NULL

Aby poprawić swoją karmę, zapisz, dlaczego w dwóch przypadkach zwrocone wyniki są inne niż w pozostałych.

Dwa zapytania zwróciły NULL.

W przyszłości te wartości NULL mogą Ci przysparzać problemów. Lepiej wpisywać w kolumnach jakąś wartość, niż pozostawiać w nich NULL, gdyż wartości tej nie można w bezpośredni sposób pobrać z tabeli.

By odszukać NULL, użyj operatora IS NULL



Próbowałem bezpośrednio pobrać z tabeli wiersze zawierające wartości NULL, ale mi się nie udało. W jaki sposób mogę odszukać w tabeli wiersze, w których są te wartości?

drinki_informacje

nazwa	cena	węglowodany	kolor	lód	kalorie
Holiday	NULL	14	NULL	T	50
Dragon Breath	7.25	7.2	brązowy	N	NULL

Nie można bezpośrednio pobrać rekordów zawierających w kolumnie wartość NULL.

SELECT nazwa FROM drinki_informacje

WHERE
kalorie = ~~NULL~~; Nie działa, gdyż nic nie jest równe NULL.
NULL to wartość niezdefiniowana.

SELECT nazwa FROM drinki_informacje

WHERE
kalorie = ~~0~~; To zapytanie nie będzie działać, gdyż NULL to nie to samo co zero.

SELECT nazwa FROM drinki_informacje

WHERE
kalorie = ~~'NULL'~~; Z kolei to zapytanie nie będzie działać, gdyż NULL nie jest tańcuchem znaków.

Ale można je pobrać, używając odpowiedniego operatora.

SELECT nazwa

FROM drinki_informacje

WHERE

kalorie IS NULL;

Te słowa kluczowe to nie tańcuchy znaków, zatem nie można ich zapisywać w apostrofach.

Jedynym sposobem bezpośredniego pobrania wierszy zawierających wartości NULL jest użycie słów kluczowych IS NULL.

P: Wspominaliście, że „nie można pobrać wartości NULL w sposób bezpośredni” inaczej niż przy użyciu wyrażenia IS NULL; czy to oznacza, że można je pobrać niejawnie?

O: Owszem. Chcąc pobrać wartość takiej kolumny, można zastosować klauzulę WHERE odwołującą się do dowolnej innej kolumny. W naszej przykładowej bazie uzyskamy wartość NULL, jeśli zastosujemy następujące zapytanie:

SELECT kalorie FROM drinki_informacje WHERE nazwa = 'Dragon Breath';

P: Jak wyglądałyby wyniki wykonania takiego polecenia SQL?

O: Wyglądałyby następująco:

-----+ kalorie -----+
NULL
-----+

W międzyczasie u Grześka...

Grzesiek próbował od jakiegoś czasu wyszukać w swojej tabeli `moje_kontakty` wszystkie osoby mieszkające w miejscowościach w województwie śląskim.
Oto fragment zapytania, nad którym pracował Grzesiek:



```
SELECT * FROM moje_kontakty  
WHERE  
    lokalizacja = 'Katowice, SL'  
    OR  
    lokalizacja = 'Czestochowa, SL'  
    OR  
    lokalizacja = 'B-Biala, SL'  
    OR  
    lokalizacja = 'Bielsko-Biala, SL'  
    OR  
    lokalizacja = 'Chorzow, SL'  
    OR  
    lokalizacja = 'Bytom, SL'  
    OR  
    lokalizacja = 'Gliwice, SL'  
    OR  
    lokalizacja = 'Zabrze, SL'  
    OR  
    lokalizacja = 'Rybnik, SL'  
    OR  
    lokalizacja = 'Pszczyna, SL'
```

W niektórych rekordach Grzesiek dodawał do nazwy miasta dwie literki oznaczające województwo.

Grzesiek obawia się, że czasami zapisywać nazwy miast skrótnie.

Jak można zaoszczędzić czas dzięki jednemu słowu: LIKE

Po prostu jest zbyt dużo możliwych miast i miejscowości, ich kombinacji oraz zbyt duże prawdopodobieństwo popełnienia jakiegoś prostego błędu typograficznego. Zapisanie tych wszystkich warunków połączonych operatorem OR zajęłoby Grześkowi zbyt dużo czasu. Na szczęście istnieje operator, który pozwoli mu zaoszczędzić wiele czasu — LIKE. W połączeniu ze znakiem wieloznaczonym pozwala on na poszukiwanie fragmentu łańcucha znaków i zwraca wszystkie rekordy, które go zawierają.

Grzesiek może zastosować operator LIKE w następujący sposób:

SELECT * FROM moje_kontakty

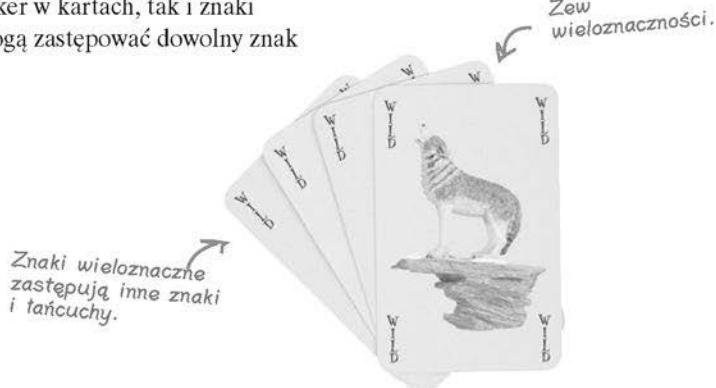
WHERE lokalizacja LIKE '%SL';



Bezpośrednio za pierwszym apostrofem należy umieścić znak procentu. W ten sposób poinformujesz system zarządzania bazami danych, że interesują Cię wszystkie wartości zapisane w kolumnie lokalizacja, które kończą się literami SL.

Zew wieloznaczności

Operator LIKE lubi pojawiać się ze znakami wieloznacznymi. Są to symbole, reprezentujące wartości faktycznie występujące w kolumnach. Podobnie jak dżoker w kartach, tak i znaki wieloznaczne w bazie danych mogą zastępować dowolny znak lub łańcuch znaków.



Czy w przykładach zamieszczonych w tym rozdziale zauważłeś może jakieś inne znaki wieloznaczne?

Kolejne znaki, które lubi operator LIKE

Operator LIKE lubi grać w jednej drużynie ze znakami wieloznaczonymi. Pierwszym z nich jest znak procentu, %, który może reprezentowaćłańcuch o dowolnej długości, składający się z dowolnych znaków.

A large black percentage sign (%) is shown on a light gray background. Three arrows point from text annotations to specific parts of the SQL query:

- An arrow points from the text "Znak procentu może być zastąpiony przez dowolną liczbę dowolnych znaków." to the '%' symbol in the WHERE clause.
- An arrow points from the text "Pasuje do imion kończących się na „na”, takich jak: Anna, Katarzyna itd." to the末尾的 'na' in the WHERE clause pattern '%na';
- An arrow points from the text "SELECT imie FROM moje_kontakty WHERE imie LIKE '%na';" to the entire WHERE clause.

```
SELECT imie FROM moje_kontakty
WHERE imie LIKE '%na';
```

Znak procentu może być zastąpiony przez dowolną liczbę dowolnych znaków.

Pasuje do imion kończących się na „na”, takich jak: Anna, Katarzyna itd.

Drugim znakiem wieloznacznym, z którym lubi przebywać operator LIKE, jest znak podkreślenia „_”. Reprezentuje on jeden, nieznany znak.

A thick black horizontal bar is shown on a light gray background. Three arrows point from text annotations to specific parts of the SQL query:

- An arrow points from the text "Znak podkreślenia może być zastąpiony tylko jednym, nieznanym znakiem." to the underscore character in the WHERE clause pattern '_asia';
- An arrow points from the text "Pasuje do imion, w których przed literami „asia” znajduje się dokładnie jedna, dowolna litera; takich jak Basia lub Kasia." to the末尾的 'asia' in the WHERE clause pattern '_asia';
- An arrow points from the text "SELECT imie FROM moje_kontakty WHERE imie LIKE '_asia';" to the entire WHERE clause.

```
SELECT imie FROM moje_kontakty
WHERE imie LIKE '_asia';
```

Znak podkreślenia może być zastąpiony tylko jednym, nieznanym znakiem.

Pasuje do imion, w których przed literami „asia” znajduje się dokładnie jedna, dowolna litera; takich jak Basia lub Kasia.



- Dopasowywanie magnesów

Do lodówki przyczepiono w różnych miejscach kawałeczki kilku klauzul WHERE wykorzystujących operator LIKE. Czy jesteś w stanie dopasować te klauzule do wyników, jakie pozwalały uzyskać? Niektóre z klauzul mogą zwracać wyniki zawierające większą ilość rekordów. Jeśli jakieś wyniki nie zostaną zwrócone przez żadną z klauzul, to samodzielnie napisz klauzulę WHERE z operatorem LIKE, która pozwoli je uzyskać.

spinacz

Janek

mazowieckie

splot

Nowy Kobrzyniec

mikser

wizjer

Nowy Targ

podkarpackie

pomorskie

Jarek

SQL. Rusz głową!

klin

podlaskie

splendor

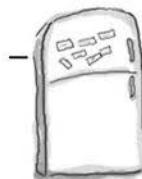
WHERE miejsowosc LIKE 'Nowy %';

WHERE przedmiot LIKE '_zjer';

WHERE tytul LIKE 'SQL.%';

WHERE do_rymu LIKE '%kser';

WHERE imie LIKE 'Ja%';



Dopasowywanie magnesików. Rozwiążanie

Do lodówki przyczepiono w różnych miejscach kawałczki kilku klauzul WHERE wykorzystujących operator LIKE. Czy jesteś w stanie dopasować te klauzule do wyników, jakie pozwalałyby uzyskać? Niektóre z klauzul mogą zwracać wyniki zawierające większą ilość rekordów. Jeśli jakieś wyniki nie zostaną zwrócone przez żadną z klauzul, to samodzielnie napisz klauzulę WHERE z operatorem LIKE, która pozwoli je uzyskać.

WHERE miejscowości LIKE 'Nowy %';
Nowy Targ
Nowy Koźminiec

WHERE przedmiot LIKE '_zjer';
wizjer

WHERE tytuł LIKE 'SQL. %';
SQL. Rusz głową!

WHERE słowo LIKE 'spl%';
splot
splendor

WHERE do_rymu LIKE '%kser';
mikser

WHERE województwo LIKE 'p%' OR województwo LIKE 'm%';
podlaskie
podkarpackie
mazowieckie
pomorskie

WHERE imię LIKE 'Ja%';
Janek
Jarek

WHERE słowo LIKE '__i%';
klin
spinacz

Pobieranie zakresów przy użyciu operatora AND i operatorów porównania

Użytkownicy Salonu Rusz głowę! starają się odnaleźć drinki, których wartość energetyczna należy do pewnego zakresu. Jakiego zapytania powinni użyć, by otrzymać nazwy drinków, których wartość energetyczna jest większa lub równa 30 kalorii, a jednocześnie nie przekracza 60 kalorii?

drinki_informacje

nazwa	cena	węglowodany	kolor	lód	kalorie
Blackthorn	7.50	8.4	żółty	T	33
Blue Moon	6.25	3.2	niebieski	T	12
Oh My Gosh	8.75	8.6	pomarańczowy	T	35
Lime Fizz	6.25	5.4	zielony	T	24
Kiss on the Lips	13.75	42.5	fioletowy	T	171
Hot Gold	8.00	32.1	pomarańczowy	N	135
Lone Tree	9.00	4.2	czerwony	T	17
Greyhound	10.00	14	żółty	T	50
Indian Summer	7.00	7.2	brązowy	N	30
Bull Frog	6.50	21.5	jasnobrązowy	T	80
Soda and It	9.50	4.7	czerwony	N	19

SELECT nazwa FROM drinki_informacje

WHERE

kalorie >= 30



W wynikach zostaną uwzględnione drinki, których wartość energetyczna wynosi dokładnie 30 kalorii, oczywiście jeśli takie znajdują się w tabeli; drinki, których wartość energetyczna wynosi 60 kalorii, oraz wszystkie pozostałe drinki, których wartość energetyczna mieści się pomiędzy tymi dwiema granicami.

AND

kalorie <= 60;



Lepszy sposób — operator **BETWEEN**

Zamiast dwóch warunków można posłużyć się operatorem BETWEEN. Pozwala on skrócić długość zapytania, a jednocześnie zapewnia uzyskanie identycznych wyników. Warto zwrócić uwagę, iż zwracane są także wartości równe granicom zakresu (30 i 60). A zatem operator BETWEEN odpowiada zastosowaniu operatorów porównania `<=` oraz `>=`, lecz nie `<` oraz `>`.

`SELECT nazwa FROM drinki_informacje`

`WHERE`

`kalorie BETWEEN 30 AND 60;`

W wynikach znajdują się także drinki, których wartość energetyczna wynosi dokładnie 30 i 60 kalorii.



To zapytanie zwróci dokładnie takie same wyniki, jak zapytanie przedstawione na poprzedniej stronie; zobacz jednak, o ile tatwiej jest je zapisać.



```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT nazwa FROM drinki_informacje
   -> WHERE kalorie BETWEEN 30 AND 60;
+-----+
| nazwa |
+-----+
| Blackthorn |
| Greyhound |
| Indian Summer |
| Oh My Gosh |
+-----+
4 rows in set (0.00 sec)

mysql> -
```

Zaostrz ołówek



Zapytanie przedstawione na poprzedniej stronie zmodyfikuj w taki sposób, by zwróciło nazwy wszystkich drinków, których wartość energetyczna jest większa od 60 kalorii lub mniejsza od 30.

Spróbuj użyć operatora BETWEEN do sprawdzania danych tekstowych.

Napisz zapytanie, które zwróci nazwy wszystkich drinków, których nazwy zaczynają się od liter pomiędzy G i O.

Jak myślisz, jakie będą wyniki wykonania poniższego zapytania?

```
SELECT nazwa FROM drinki_informacje WHERE  
kalorie BETWEEN 60 AND 30;
```

Rozwiążanie kolejnego ćwiczenia

Zaostrz ołówek



Rozwiążanie

Zmodyfikuj zapytanie przedstawione na poprzedniej stronie w taki sposób, by zwróciło nazwy wszystkich drinków, których wartość energetyczna jest większa od 60 kalorii lub mniejsza od 30.

SELECT nazwa FROM drinki_informacje

WHERE

kalorie < 30 OR kalorie > 60;

Ten warunek zwróci nazwy wszystkich drinków, których wartość energetyczna jest większa od 60 kalorii.

Z kolei ten zwróci nazwy wszystkich drinków, których wartość energetyczna jest mniejsza od 30 kalorii.

Spróbuj użyć operatora BETWEEN do sprawdzania danych tekstowych. Napisz zapytanie, które zwróci nazwy wszystkich drinków, których nazwy zaczynają się od liter pomiędzy G i O.

SELECT nazwa FROM drinki_informacje

WHERE

nazwa BETWEEN 'G' AND 'O';

W ten sposób pobierzemy wszystkie drinki, których nazwy zaczynają się od litery z zakresu od G do O.

Jak myślisz, jakie będą wyniki wykonania poniższego zapytania?

**SELECT nazwa FROM drinki_informacje WHERE
kalorie BETWEEN 60 AND 30;**

Kolejność ma znaczenie, zatem powyższe zapytanie nie zwróci żadnych wyników.

Powyższe zapytanie poszukuje wartości należących do przedziału od 60 do 30. Zakres ten jest pusty, ponieważ nie ma liczby, która byłaby większa od 60 i jednocześnie mniejsza od 30. Aby działanie operatora **BETWEEN** było zgodne z naszymi oczekiwaniami, pierwsza z liczb podanych po jego prawej stronie musi być większa do drugiej liczby.

Operator IN — w kręgu zainteresowania...

Anka, przyjaciółka Grześka, korzysta z jego bazy kontaktów do wybierania chłopaków na randki. Była już na kilku randkach i wpadła na pomysł, by prowadzić własny „rejestr”, w którym będzie gromadzić krótkie opinie o poszczególnych osobach.

Stworzyła zatem tabelę o nazwie `rejestr_randek`. Teraz Anka chciałaby uzyskać listę udanych randek, zatem stworzyła zapytanie, umieszczając w nim jedynie pozytywne opinie.

```
SELECT imie
FROM rejestr_randek
WHERE
opinia = 'inspirujaca'
OR
opinia = 'fantastyczna'
OR
...;
```

Dla każdej pozytywnej opinii trzeba napisać jeden warunek.
To są pozytywne opinie.

`rejestr_randek`

imie	opinia
Olek	inspirująca
Janek	nudna
Iwan	fantastyczna
Borys	no trudno
Maciek	ujdzie
Edward	szkoda czasu
Antek	rewelacyjna
Szymek	dobra
Ignacy	fatalna
Wiesiek	żałosna

Zamiast stosować te wszystkie warunki połączone operatorem OR, można uprościć sobie życie i zapytanie, stosując operator IN. Można go używać, gdy poszukiwane wartości mogą należeć do zbioru kilku wartości tekstowych. Jeśli wartość w kolumnie będzie odpowiadać jednej z wartości podanego zbioru, to rekord zostanie zwrócony w wynikach zapytania.

```
SELECT imie
FROM rejestr_randek
WHERE
opinia IN ('inspirujaca',
'fantastyczna',
'rewelacyjna', 'dobra');
```

Użycie operatora IN informuje system zarządzania bazami danych, że zaraz zostanie podany zbiór wartości.

To jest zbiór pozytywnych opinii o randkach.

```
C:\Program Files\MySQL\MySQL Server 5.0\bin>mysql
mysql> SELECT imie
    -> FROM rejestr_randek
    -> WHERE
    -> opinia IN ('inspirujaca', 'fantastyczna', 'rewelacyjna', 'dobra');
+-----+
| imie |
+-----+
| Olek |
| Iwan |
| Antek |
| Szymek |
+-----+
4 rows in set (0.00 sec)

mysql>
```

...lub poza nim — NOT IN

Oczywiście Anka chciałaby także wiedzieć, kto uzyskał niekorzystną opinię, by w razie telefonu od takiej osoby być akurat w trakcie kąpieli lub mieć już zaplanowane inne, ważne spotkanie.

Aby określić imiona osób, które nie mają pozytywnej opinii, przed operatorem IN będziemy musieli dodać operator NOT. Operator ten daje przeciwnie wyniki, czyli w wynikach zostaną umieszczone wszystkie rekordy, które nie spełniają podanego warunku.

Jeśli wyrażenie
NOT IN zwróci Twoje
imię, to nie masz co
zawracać mi głowy.

```
SELECT imie  
FROM rejestr_randek  
WHERE  
opinia NOT IN ('inspirujaca',  
'fantastyczna', 'rewelacyjna',  
'dobra');
```

Zastosowanie operatorów NOT
IN informuje, że interesują nas
rekordy, których zawartość nie
należy do podanego zbioru.



W wynikach tego zapytania
wykorzystującego wyrażenie NOT IN będą
osoby, które nie uzyskały pozytywnej
opinii po pierwszej randce, więc nie będą
miały drugiej szansy...

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe  
mysql> SELECT imie  
-> FROM rejestr_randek  
-> WHERE  
-> opinia NOT IN ('inspirujaca', 'fantastyczna', 'rewelacyjna', 'dobra');  
+-----+  
| imie |  
+-----+  
| Janek |  
| Borys |  
| Maciek |  
| Edward |  
| Ignacy |  
| Wiesiek |  
+-----+  
6 rows in set (0.00 sec)  
mysql>
```



**WYSIL
SZARE KOMÓRKI**

Dlaczego czasami trzeba będzie
używać wyrażenia NOT IN, a nie
wyłącznie operatora IN?

Więcej o operatorze NOT

Operatora NOT można także używać wraz z operatorami BETWEEN oraz LIKE. Koniecznie należy jednak pamiętać o tym, iż w poleceniach SQL operator NOT musi być umieszczany bezpośrednio za słowem kluczowym WHERE. Oto dwa przykłady prawidłowego zastosowania operatora NOT.

```
SELECT nazwa FROM drinki_informacje
WHERE NOT węglowodany BETWEEN 3 AND 5;
```

W przypadku używania operatora NOT wraz z operatorami AND i OR operator NOT powinien być umieszczany bezpośrednio za operatorami AND i OR.



```
SELECT imie FROM rejestr_randek
WHERE NOT imie LIKE 'A%'
AND NOT imie LIKE 'B%';
```

Nie istniejąca grupa pytań

P: Chwileczkę. Przed chwilą napisaliście, że operator NOT należy zapisywać za słowem kluczowym WHERE. A co zrobić, gdy chcę użyć wyrażenia NOT IN?

O: To jest wyjątek. W tym przypadku nawet przeniesienie operatora NOT i umieszczenie go bezpośrednio za słowem kluczowym WHERE będzie działać. Co więcej, w obu przypadkach uzyskamy identyczne wyniki:

```
SELECT * FROM proste_drinki
WHERE NOT składnik_główny IN ('woda sodowa',
'mrożona herbata');
```

```
SELECT * FROM proste_drinki
WHERE składnik_główny NOT IN ('woda sodowa',
'mrożona herbata');
```

P: A czy w tym przypadku można by zastosować operator porównania <> — różny?

O: Można by, jednak uzyskamy w ten sposób podwójne zaprzeczenie. W tym przypadku znacznie lepszym rozwiązaniem byłoby zastosowanie znaku równości. Dwa przedstawione poniżej zapytania zwracają takie same wyniki:

```
SELECT * FROM proste_drinki
WHERE NOT nazwa <> 'Blackthorn';
```

```
SELECT * FROM proste_drinki
WHERE nazwa = 'Blackthorn';
```

P: A jak operator NOT działa w przypadku wartości NULL?

O: Dokładnie tak, jak można by tego oczekwać. Aby pobrać z kolumny wszystkie wartości, które są różne od NULL, należy wykonać następujące zapytanie:

```
SELECT * FROM proste_drinki
WHERE NOT składnik_główny IS NULL;
```

Jednak także poniższe zapytanie zwróci identyczne wyniki:

```
SELECT * FROM proste_drinki
WHERE składnik_główny IS NOT NULL;
```

P: A co z operatorami AND i OR?

O: Jeśli chcesz zastosować operator NOT w klauzuli, w której występuje także operator AND bądź OR, to NOT należy umieścić bezpośrednio za operatorem AND bądź OR; jak na poniższym przykładzie:

```
SELECT * FROM proste_drinki
WHERE NOT składnik_główny = 'woda sodowa'
AND NOT składnik_główny = 'mrożona herbata';
```



Ćwiczenie

Przepisz każde z poniższych zapytań w taki sposób, by było możliwie jak najprostsze. Możesz przy tym używać operatorów AND, OR, BETWEEN, LIKE, IN, IS NULL oraz operatorów porównania. W razie potrzeby zajrzyj do tabel zamieszczonych we wcześniejszej części rozdziału.

```
SELECT nazwa FROM proste_drinki  
WHERE NOT ilosc1 < 45;
```

.....
.....
.....

```
SELECT nazwa FROM drinki_informacje  
WHERE NOT lod = 'T';
```

.....
.....
.....

```
SELECT nazwa FROM drinki_informacje  
WHERE NOT kalorie < 20;
```

.....
.....
.....

```
SELECT nazwa FROM proste_drinki  
WHERE skladnik_dodatkowy = 'lemoniada'  
OR skladnik_dodatkowy = 'nekter morelowy';  
.....  
.....  
.....
```

```
SELECT nazwa FROM drinki_informacje  
WHERE NOT kalorie = 0;  
.....  
.....  
.....
```

```
SELECT nazwa FROM drinki_informacje  
WHERE NOT weglowodany BETWEEN 3 AND 5;  
.....  
.....  
.....
```

```
SELECT imie FROM rejestr_randek  
WHERE NOT imie LIKE 'A%'  
AND NOT imie LIKE 'B%';  
.....  
.....  
.....
```



Przepisz każde z poniższych zapytań w taki sposób, by było możliwie jak najprostsze. Możesz przy tym używać operatorów AND, OR, BETWEEN, LIKE, IN, IS NULL oraz operatorów porównania. W razie potrzeby zajrzyj do tabel zamieszczonych we wcześniejszej części rozdziału.

```
SELECT nazwa FROM proste_drinki  
WHERE NOT ilosc1 < 45;
```

```
.....  
SELECT nazwa FROM proste_drinki  
.....  
WHERE ilosc1 >= 45;.....
```

```
SELECT nazwa FROM drinki_informacje  
WHERE NOT lod = 'T';
```

```
.....  
SELECT nazwa FROM drinki_informacje  
.....  
WHERE lod = 'N';.....
```

```
SELECT nazwa FROM drinki_informacje  
WHERE NOT kalorie < 20;
```

```
.....  
SELECT nazwa FROM drinki_informacje  
.....  
WHERE kalorie >= 20;.....
```

```
SELECT nazwa FROM proste_drinki
WHERE składnik_dodatkowy = 'lemoniada'
OR składnik_dodatkowy = 'nekter morelowy';
```

SELECT nazwa FROM proste_drinki

WHERE składnik_dodatkowy BETWEEN 'L' AND 'O';

To zapytanie będzie działać wyłącznie dlatego, że nie dysponujemy żadnymi innymi składnikami dodatkowymi, które spełniałyby podany warunek. Gdyby w tabeli znajdowała się wartość, na przykład: nektar ananasowy, to zapytanie nie działałoby prawidłowo.

```
SELECT nazwa FROM drinki_informacje
WHERE NOT kalorie = 0;
```

SELECT nazwa FROM drinki_informacje

WHERE kalorie > 0;

Wartość energetyczna nie może być mniejsza od zera, a zatem możemy bezpiecznie zastosować znak większości.

```
SELECT nazwa FROM drinki_informacje
WHERE NOT weglowodany BETWEEN 3 AND 5;
```

SELECT nazwa FROM drinki_informacje

WHERE weglowodany < 3

OR

weglowodany > 5;

```
SELECT imie FROM rejestr_randek
WHERE NOT imie LIKE 'A%'
AND NOT imie LIKE 'B%';
```

SELECT imie FROM rejestr_randek

WHERE imie NOT BETWEEN 'A' AND 'B';



Przybornik SQL

A zatem opanowałeś już materiał z drugiego rozdziału książki i dodałeś od swojego SQL-owego przybornika znajomość operatorów. Kompletną listę porad znajdziesz w dodatku C zamieszczonym na końcu książki.

SELECT *

Użyj tej klauzuli, by pobrać wszystkie kolumny tabeli.

Poprzedzaj ' znakiem ukośnika \

Znaki apostrofu umieszczone w tekście poprzedzaj dodatkowym znakiem apostrofu bądź znakiem ukośnika.

= < > > < <= =>

Masz do swej dyspozycji kilka operatorów porównania.

IS NULL

Zastosuj to wyrażenie, by sprawdzać, czy w kolumnach tabeli znajduje się kłopotliwa wartość NULL.

BETWEEN

Ten operator pozwala na pobieranie zakresów wartości.

LIKE oraz % i _

Używaj operatora LIKE wraz ze znakami wieloznacznymi, by przeszukiwać pola zawierającełańcuchy znaków.

AND i OR

Dzięki operatorom AND i OR możesz łączyć wyrażenia warunkowe umieszczone w klauzuli WHERE, by uzyskać dodatkową precyzję.

NOT

Operator NOT pozwala zanegować warunek i uzyskać odwrotne wyniki.

↑ ↗
Nowe narzędzie
— operatory!



Rozwiązanie ćwiczenia

Grzesiek chce utworzyć tabelę z informacjami o drinkach, do której barmani mogliby zaglądać w poszukiwaniu przepisów na drinki serwowane podczas randek umawianych przez Grześka. Bazując na informacjach zdobytych w rozdziale 1., utwórz tabelę i zapisz w niej przedstawione poniżej dane.

Tabela należy do bazy danych o nazwie drinki. Zawiera ona tabelę o nazwie proste_drinki z kilkoma przepisami na drinki składające się jedynie z dwóch składników.

```
CREATE DATABASE drinki;
USE drinki;
CREATE TABLE proste_drinki
(nazwa VARCHAR(16), skladnik_glowny VARCHAR(30), ilosc1 DEC(4,1),
skladnik_dodatkowy VARCHAR(30), ilosc2 DEC(4,1), wskazowki VARCHAR(250));
```

Zawsze warto tworzyć pola tekstowe z pewnym zapasem długości, na wypadek gdyby trzeba w nich było zapisać tekst dłuższy, niż się początkowo planowało.

```
INSERT INTO proste_drinki
VALUES
('Blackthorn', 'tonik', 45, 'sok ananasowy', 30, 'wymieszac z lodem, odcedzic do szklanki koktajlowej z plasterkiem cytryny'),
('Blue Moon', 'woda sodowa', 45, 'sok z jagod', 22, 'wymieszac z lodem, odcedzic do szklanki koktajlowej z plasterkiem cytryny'),
('Oh My Gosh', 'nekter brzoskwiniowy', 30, 'sok ananasowy', 30, 'wymieszac z lodem, odcedzic do wysokiej szklanki'),
('Lime Fizz', 'Sprite', 45, 'sok z cytryny', 22, 'wymieszac z lodem, odcedzic do szklanki koktajlowej'),
('Kiss on the Lips', 'sok wiśniowy', 60, 'nekter morelowy', 210, 'podawac z lodem i słomką do picia'),
('Hot Gold', 'nekter brzoskiniowy', 90, 'sok pomaranczowy', 180, 'wlac goracy sok pomaranczowy do kubka, dolac do niego nektaru brzoskwiniowego'),
('Lone Tree', 'woda sodowa', 45, 'sok wiśniowy', 22, 'wymieszac z lodem, odcedzic do szklanki koktajlowej'),
('Grayhound', 'woda sodowa', 45, 'sok z grapefruita', 150, 'podawac z lodem, dobrze wymieszac'),
('Indian Summer', 'sok jablkowy', 60, 'goraca herbata', 180, 'wlac sok do kubka, dodac hrebaty'),
('Bull Frog', 'mrozona herbata', 45, 'lemoniada', 150, 'podawac z lodem i plasterkiem cytryny'),
('Soda and It', 'woda sodowa', 60, 'sok z winogron', 30, 'wstrzasnac w szklance koktajlowej, podawac bez lodu');
```

Grupa wartości opisujących każdy drink jest zapisywana wewnętrz pary nawiasów.

Pomiędzy danymi o poszczególnych drinkach umieszczane są przecinki.

3. DELETE i UPDATE

**Są szanse,
że wszystko będzie w porządku**

Czy następnym razem spróbujesz
to zrobić bezpiecznie – używając
poleceńa **DELETE**? Nie stać mnie na
kupowanie cygar za każdym razem,
gdy przychodzę cię odwiedzić.



Cały czas zmieniasz zdanie? Teraz nie przysporzy Ci to najmniejszego problemu! Dzięki polecaniom **DELETE** i **UPDATE**, które poznasz w tym rozdziale, nie będziesz już dłużej musiał ponosić konsekwencji decyzji podjętych pół roku temu, kiedy to zapisałeś w bazie dane o spodniach w kształcie dzwonów, które właśnie z powrotem zaczynały być modne. Dzięki poleceniu **UPDATE** będziesz mógł **zmieniać dane**, natomiast polecenie **DELETE** pozwoli **usunąć z bazy** dane, które nie będą Ci już dłużej potrzebne. Jednak w tym rozdziale nie tylko pokażemy Ci te dwa nowe polecenia SQL, lecz także nauczymy Cię, jak można używać ich w precyzyjny sposób, by przez przypadek nie usunąć danych, które cały czas są potrzebne.

Klowni są przerażający

Załóżmy, że chcielibyśmy śledzić klaunów pracujących w Bazodanowie.

Moglibyśmy w tym celu stworzyć tabelę o nazwie `kłowni_informacje`.

W tej tabeli moglibyśmy utworzyć kolumnę `ostatnio_widziano`, przeznaczoną do zapisywania informacji o miejscu, w jakim ostatnio widziano konkretnego klauna.

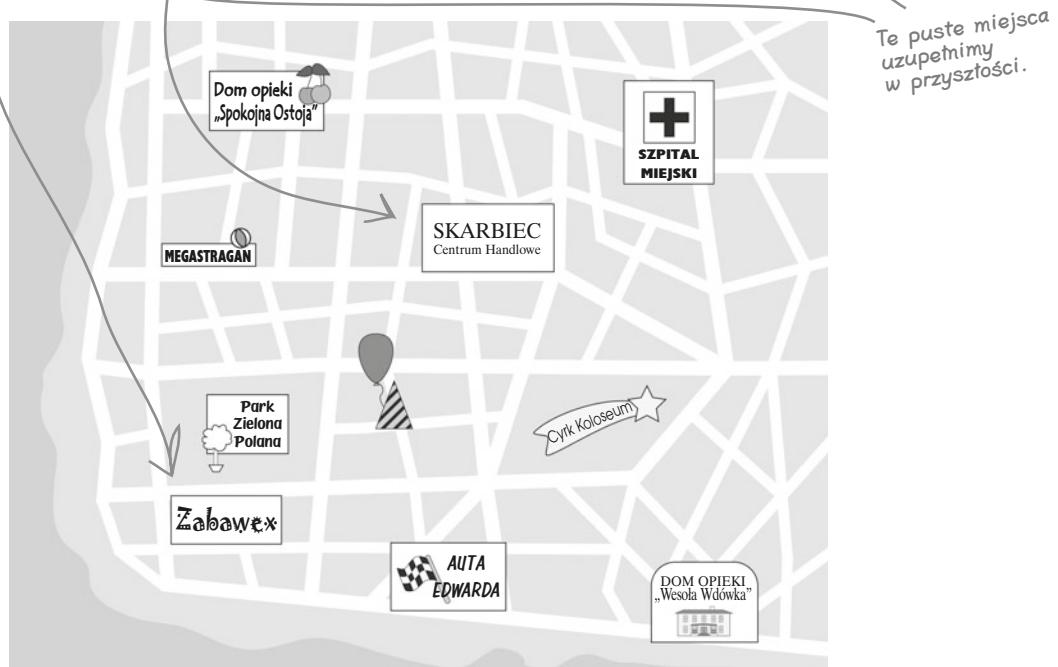


Śledzenie kloonów

A oto i nasza tabela. Możemy nie podawać informacji, których jeszcze nie znamy, i uzupełnić je później. Za każdym razem, gdy zobaczymy nowego klauna, możemy dodać do tabeli nowy wiersz. Najprawdopodobniej będziemy musieli często modyfikować tabelę, by zapisane w niej informacje były aktualne.

Gdzie dany klaun był ostatnio widziany.

kлоoni_informacje			
imie	ostatnio_widziano	wyglad	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółta koszula, workowate czerwone spodnie	trąbka, parasolka
Pan Hobo	Cyrk Koloseum	M, cygaro, czarne włosy, niewielki kapelusz	skrzypce
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Skuter	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
Balbina	Auta Edwarda	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
Gonzo		M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Zabawex	M, zielono-fioletowy kostium, szpiczasty nos	





Klowni są w ciągłym ruchu

Twoim zadaniem jest napisanie polecień SQL, które pozwolą na zarejestrowanie w tabeli klowni_informacje najnowszych informacji o kłownach. Zwróć uwagę, iż nie wszystkie informacje o poszczególnych kłownach uległy zmianie, zatem by podać w poleceniu wszystkie wymagane informacje, będziesz musiał zająrzyć do początkowej tabeli przedstawionej na stronie 155.

Ostatnio widziano śpiewającą Zippo.

INSERT INTO klowni_informacje

VALUES

('Zippo', 'Centrum Handlowe Skarbiec', 'K, pomarańczowy garnitur, workowate spodnie', 'taniec, śpiew');

Pani Smyk zaczęła nosić workowate niebieskie spodnie.

INSERT INTO klowni_informacje

VALUES

('Pani Smyk', 'Zabawex', 'K, zolta koszula, workowane niebieskie spodnie', 'trąbka, parasolka');

Gonza ostatnio widziano w parku Zielona Polana.

Pan Smyk zostałauważony w niewielkim samochodzie.

Pan Hobo ostatnio zabawiał dzieci na przyjęciu u Eryka Szarzyńskiego.

A teraz zapisz, jak będą wyglądać te nowe dane w tabeli `kłowni_informacje` po wykonaniu polecień `INSERT` z poprzedniej strony.

imie	ostatnio_widziano	wygląd	aktywności
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółta koszula, workowate czerwone spodnie	trąbka, parasolka
Pan Hobo	Cyrk Koloseum	M, cygaro, czarne włosy, niewielki kapelusz	skrzypce
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Skuter	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
Balbina	Auta Edwarda	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
Gonzo		M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Zabawex	M, zielono-fioletowy kostium, szpiczasty nos	

Zaostrz ołówek



Rozwiążanie Kłowni są w ciągłym ruchu

Twoim zadaniem było napisanie polecień SQL, które pozwolą na zarejestrowanie w tabeli kloni_informacje najnowszych informacji o kłownach, a następnie pokazanie, jak te informacje wyglądają po ich zapisaniu w tabeli.

Ostatnio widziano śpiewającą Zippo.

Pani Smyk zaczęła nosić workowate niebieskie spodnie.

Gonza ostatnio widziana w parku Zielona Polana.

Pan Smyk zostałauważony w niewielkim samochodzie.

Pan Hobo ostatnio zabawiał dzieci na przyjęciu u Eryka Szarzyńskiego.

INSERT INTO kloni_informacje

VALUES

('Zippo', 'Centrum Handlowe Skarbiec', 'K, pomarańczowy garnitur, workowate spodnie', 'taniec, śpiew');

INSERT INTO kloni_informacje

VALUES

('Pani Smyk', 'Zabawex', 'K, zolta koszula, workowane niebieskie spodnie', 'trabka, parasolka');

INSERT INTO kloni_informacje

VALUES

('Gonzo', 'Park Zielona Polana', 'M, w przebraniu kobiety, kostium w plamki', 'śpiew, taniec');

Nie zapomnij odpowiednio poprzedzić znaków apostrofu umieszczanych w kolumnach typu VARCHAR.

INSERT INTO kloni_informacje

VALUES

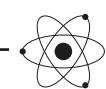
('Pan Smyk', 'Przyjście urodzinowe w Mc'Stecku', 'M, zielono-fioletowy kostium, szpiczasty nos', 'wchodzenie do malego samochodu');

INSERT INTO kloni_informacje

VALUES

('Pan Hobo', 'Przyjście u Eryka Szarzyńskiego', 'M, cygaro, czarne włosy, malutki kapelusz', 'skrzypce');

imie	ostatnio_widziano	wyglad	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółta koszula, workowate czerwone spodnie	trąbka, parasolka
Pan Hobo	Cyrk Koloseum	M, cygaro, czarne włosy, niewielki kapelusz	skrzypce
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Skuter	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
Balbina	Auta Edwarda	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
Gonzo		M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Zabawex	M, zielono-fioletowy kostium, szpiczasty nos	
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Pani Smyk	Zabawex	K, żółta koszula, workowate niebieskie spodnie	trąbka, parasolka
Gonzo	Park Zielona Polana	M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Przyjęcie urodzinowe w Mc'Stecku	M, zielono-fioletowy kostium, szpiczasty nos	wchodzenie do małego samochodu
Pan Hobo	Przyjęcie u Eryka Szarzyńskiego	M, cygaro, czarne włosy, malutki kapelusz	skrzypce



WYSIL
SZARE KOMÓRKI

Jak możesz odszukać bieżące miejsce pobytu konkretnego klauna?

A czy można pobierać dane chronologicznie?

Jak są zapisywane informacje o kłownach?

Informacje o kłownach są zapisywane w bazie przez ochotników. Czasami doniesienia o kłownach leżą w elektronicznej skrzynce pocztowej, zanim zostaną wpisane; a czasami dwie osoby **dzielą cały stosik doniesień** między sobą i **wpisują je jednocześnie**.

Pamiętając o tym, zobaczymy wszystkie wiersze naszej tabeli zawierające informacje o kłownie Zippo. Możemy je pobrać, używając polecenia SELECT:

C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe

```
mysql> SELECT * FROM kloni_informacje WHERE imie = 'Zippo';
```

imie	ostatnio_widziano	wyglad	aktywnosci
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Zippo	Szpital miejski	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Zippo	Zabawex	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Zippo	MegaStragan	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Zippo	Szpital miejski	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew

Te dwa rekordy są dokładnie takie same.

Podobnie te dwa są identyczne.

Te dane powtarzają się we wszystkich rekordach.

Czy istnieje jakiś sposób na pobranie z tabeli jedynie informacji o ostatnim pojawienniu się naszego ulubionego kłowna Zippo? Czy potrafisz dowiedzieć się, gdzie on był widziany?



Jasne, to proste. Wystarczy
zerknąć na ostatni rekord.

Niestety nie możemy być pewni, że ostatni rekord zawiera najnowsze informacje o interesującym nas kłownie.

Może się bowiem zdarzyć, że w tym samym czasie informacje do bazy będzie wprowadzać więcej niż jedna osoba. Poza tym kolejność wiadomości w skrzynce pocztowej nie musi odpowiadać kolejności, w jakiej kłowni pojawiali się w różnych miejscach. Jednak nawet gdyby te problemy nie występowały, to i tak nie można by **założyć**, że wiersze w tabeli odpowiadają chronologicznej kolejności pojawiania się kłownów w mieście.

Istnieje wiele czynników związanych z wewnętrznymi mechanizmami działania bazy danych, które mogą warunkować kolejność, w jakiej poszczególne rekordy są zapisywane w tabeli. Między innymi zależy to od używanego systemu zarządzania bazami danych oraz utworzonych indeksów (zajmiemy się nimi w dalszej części książki).

Nie mamy żadnych gwarancji, że ostatni wiersz tabeli jest wierszem, który został do niej ostatnio dodany.

Gonzo, mamy problem

Mamy pewien problem, ponieważ nie ma żadnych gwarancji, że ostatni rekord tabeli jest najnowszym rekordem, jaki został do niej dodany. Z naszej tabeli możemy pobrać listę udostępniającą nam informacje o tym, gdzie poszczególni kлоuni byli widziani w pewnym czasie. **Jednak naszym podstawowym celem utworzenia tej tabeli była chęć uzyskania informacji, gdzie dany kлоun był widziany ostatnio.**

Ale to jeszcze nie wszystko. Czy zauważysz te powtarzające się rekordy? W ostatnich wynikach mogliśmy zobaczyć dwa rekordy pokazujące, że kлоun Zippo był w tym samym miejscu, robiąc dokładnie to samo. Takie identyczne, powtarzające się rekordy niepotrzebnie zużywają miejsce i wraz ze wzrostem ich ilości mogą się przyczynić do spowalniania działania systemu zarządzania bazą danych. Rekordy w tabeli **nigdy nie powinny się powtarzać**. W dalszej części książki wyjaśnimy, dlaczego takie powielanie rekordów jest złe i w jaki sposób **dobry projekt tabeli** pozwoli nam go unikać. Dowiesz się między innymi, jak tworzyć tabele, w których rekordy nigdy nie będą się powtarzać. Na razie jednak skoncentrujmy się na tym, jak należy poprawić naszą tabelę, byśmy mogli pobierać z niej interesujące nas informacje.

Nie istniejąca grupa pytania

P: Dlaczego nie możemy po prostu założyć, że ostatni rekord jest rekordem najnowszym?

O: Nie ma gwarancji co do kolejności, w jakiej rekordy są zapisywane w tabeli, a już niedługo nauczysz się modyfikować kolejność, w jakiej będą zwracane wyniki poleceń SELECT. Nie możemy mieć pewności, że ostatni wyświetlony rekord *faktycznie* jest ostatnim rekordem dodanym do tabeli. Kolejność rekordów w tabeli może także ulegać modyfikacjom na skutek naszych błędów. Założymy, że wpisaliśmy dwa polecenia INSERT z informacjami o tym samym kлоuniu. Jeśli sami nie zapamiętamy, w którym miejscu kлоun był widziany wcześniej, to po zapisaniu danych w tabeli nie będziemy mieli pewności co do tego, który z tych dwóch rekordów powinien być traktowany jako najnowszy.

P: Założmy, że pamiętamy kolejność. Pozostaje to samo pytanie: Dlaczego nie możemy założyć, że ostatni rekord jest rekordem najnowszym?

O: Rozszerzymy nieco nasz przykład. Chcemy śledzić publiczne wystąpienia tego samego kлоuna w ciągu wielu lat. Może mamy pomocników, którzy także śledzą kлоuna i sami zapisują w bazie informacje o nim, wykonując polecenia INSERT. Po pewnym czasie w bazie mogą się pojawić setki rekordów dotyczących tego samego kлоuna. W razie wykonania zapytania — polecenia SELECT — dotyczącego takiego kлоuna zwrocone zostaną setki rekordów. I co w takiej sytuacji? Będziemy musieli przeglądać te setki rekordów, mając przy tym *nadzieję*, że ostatni z nich będzie zawierał informacje o ostatnim miejscu, w jakim widziano kлоuna.

P: A czy czasami nie mogą pojawić się sytuacje, w których będziemy chcieli przechowywać w bazie takie informacje? Czy dodawanie do bazy nowych wierszy i zachowywanie już istniejących zawsze jest jedynym sensownym rozwiązańiem?

O: Oczywiście. Przyjrzyjmy się naszemu przykładowi. Nasza tabela kłounów w swojej bieżącej postaci nie tylko udostępnia nam informacje o tym, gdzie każdy kлоun był ostatnio widziany, lecz także pozwala na sporządzenie rejestru miejsc, w których pojawiał się wcześniej. Są to potencjalnie użyteczne informacje. Nasz problem polega na tym, że w rekordach nie ma żadnych informacji o tym, kiedy kлоun był widziany w danym miejscu. Jeśli dodamy kolumnę zawierającą datę i czas pojawienia się kлоuna, okaza się, że nasza tabela śledzi kłounów z bardzo dużą dokładnością. Jednak w pierwszej kolejności musimy naprawić wcześniejsze błędy i usunąć z tabeli powtarzające się rekordy.

P: No dobrze, zatem po zakończeniu lektury książki będę wiedzieć, jak należy projektować tabele, w których nie będzie powtarzających się rekordów. Ale co zrobić, jeśli gość, który był zatrudniony na moim stanowisku wcześniej, pozostawił mi bazę danych z nieprawidłowo zaprojektowanymi tabelami?

O: W codziennej praktyce nieprawidłowo zaprojektowane tabele można znaleźć dosyć często i wiele osób uczących się języka SQL wcześniej czy później będzie musiało sprzątać bałagan, który pozostawił po sobie kto inny.

Istnieje kilka technik usuwania powielających się rekordów. Jedna z najlepszych wykorzystuje złączenia — zagadnienie to zostanie opisane w dalszej części książki. Jak na razie nie posiadasz jeszcze wszystkich narzędzi koniecznych do poprawienia danych w tabeli, jednak kiedy skończysz lekturę, będziesz dysponował całą niezbędną do tego wiedzą.

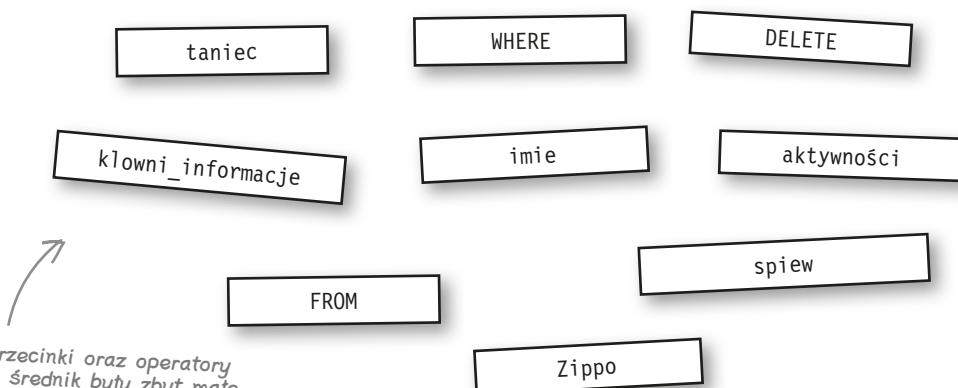
Jak pozbyć się rekordu — polecenie DELETE

Wygląda na to, że będziemy musieli pozbyć się kilku rekordów. Aby tabela była bardziej przydatna do naszych celów, dla każdego klauna musi w niej być zapisany tylko jeden rekord. Skoro wiemy, który rekord reprezentuje ostatnie pojawienie się klauna Zippo, możemy usunąć z tabeli wszystkie pozostałe rekordy dotyczące Zippo, gdyż nie będą nam do niczego potrzebne, i spokojnie czekać na kolejne doniesienia o jej występach.

Narzędziem służącym do usuwania rekordów z tabeli jest polecenie SQL DELETE. Jest w nim stosowana przedstawiona już wcześniej klauzula WHERE. Ciekawe, czy będziesz w stanie domyślić się, jaką składnię ma to polecenie.

Poniżej ponownie przedstawiamy wiersze tabeli dotyczące Zippo:

imie	ostatnio_widziano	wygląd	aktywnosci
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Zippo	Szpital miejski	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Zippo	Zabawex	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Zippo	MegaStragan	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Zippo	Szpital miejski	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew



Apostrofy, przecinki oraz operatory porównania i średnik były zbyt małe i nie chciało się nam ich podnosić z podlogi. Dopusz je, jeśli uznasz, że są gdzieś potrzebne.

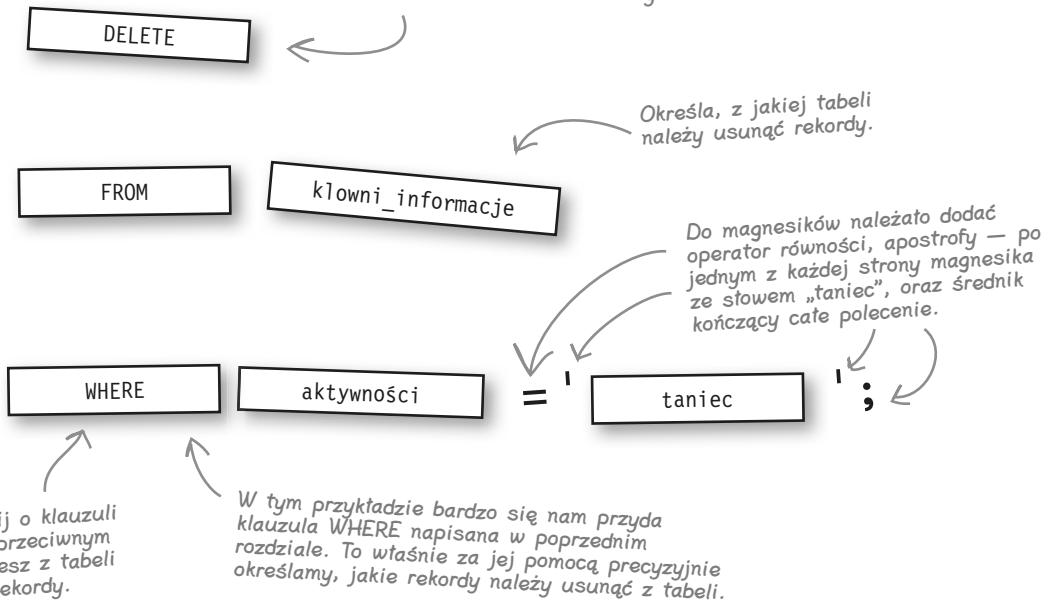
Magnesiki polecenia DELETE. Rozwiążanie



Magnesiki polecenia DELETE. Rozwiążanie

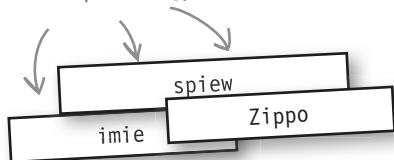
Na małych, prostokątnych magnesikach napisaliśmy proste polecenie SQL, którego można by użyć do usunięcia z tabeli niepotrzebnych rekordów dotyczących kłowna Zippo. Niestety, wszystkie magnesiki odpadły z lodówki. Z powrotem ułóż magnesiki w odpowiedniej kolejności i opisz, do czego według Ciebie służy każda z części nowego polecenia.

W tym przypadku – w odróżnieniu od przypadku polecenia SELECT – nie musimy określać, z jakich kolumn tabeli chcemy usunąć dane. Polecenie DELETE usuwa bowiem całe rekordy.



Klauzul WHERE można używać w poleceniach DELETE w taki sam sposób, w jaki są one używane w poleceniach SELECT.

Te magnesiki nie były nam tym razem potrzebne.



Stosowanie naszego nowego polecenia DELETE

Spróbujmy zatem zastosować polecenie DELETE, które przed chwilą stworzyliśmy. Polecenie to robi dokładnie to, czego można oczekwać — powoduje usunięcie z tabeli wszystkich rekordów spełniających kryteria podane w klauzuli WHERE.

**DELETE FROM klowni_informacje
WHERE
aktywnosci = 'taniec';**

To jest rekord,
który zostanie
usunięty.

imie	ostatnio_widziano	wyglad	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółta koszula, workowate czerwone spodnie	trąbka, parasolka
Pan Hobo	Cyrk Koloseum	M, cygaro, czarne włosy, niewielki kapelusz	skrzypce
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Skuter	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
Balbina	Auta Edwarda	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
Gonzo		M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Zabawex	M, zielono-fioletowy kostium, szpiczasty nos	
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Pani Smyk	Zabawex	K, żółta koszula, workowate niebieskie spodnie	trąbka, parasolka
Gonzo	Park Zielona Polana	M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Przyjęcie urodzinowe w Mc'Steku	M, zielono-fioletowy kostium, szpiczasty nos	wchodzenie do małego samochodu
Pan Hobo	Przyjęcie u Eryka Szarzyńskiego	M, cygaro, czarne włosy, malutki kapelusz	skrzypce



**WYSIL
SZARE KOMÓRKI**

Czy myślisz, że polecenie DELETE pozwala na usunięcie jednej, wybranej kolumny z wiersza?

Reguły polecenia **DELETE**

- Nie można zastosować polecenia **DELETE** do usunięcia zawartości jednej kolumny lub grupy kolumn tabeli.
- Pisząc odpowiednią klauzulę **WHERE**, można użyć polecenia **DELETE** do usunięcia jednego lub większej liczby wierszy tabeli.
- Zobaczyłeś już, w jaki sposób można usunąć pojedynczy wiersz tabeli. Jednak w podobny sposób można także usuwać całe grupy rekordów. W tym celu należy napisać odpowiednią klauzulę **WHERE**, która wskaże poleceniu **DELETE**, jakie rekordy należy usunąć. Taka klauzula **WHERE** jest taka sama, jak klauzula zastosowana w rozdziale 2. w poleceniu **SELECT**. Można w niej używać tych samych konstrukcji, które opisaliśmy w rozdziale 2. — operatorów **LIKE**, **IN**, **BETWEEN** oraz wszystkich operatorów porównań. Dzięki nim można precyzyjnie określić, jakie rekordy mają zostać usunięte.
- I uważaj na polecenie o postaci: **DELETE FROM nazwa_tabeli** — jego wykonanie spowoduje usunięcie całej zawartości tabeli.

Nie istnieją
grupie pytania

P: Czy są jakieś różnice pomiędzy stosowaniem klauzul **WHERE** w poleceniach **DELETE** i w poleceniach **SELECT**?

O: Nie, nie ma. Klauzula **WHERE** jest zawsze taka sama, niemniej jednak działanie polecień **SELECT** i **DELETE** jest diametralnie odmienne. Pierwsze z nich zwraca kopię zawartości kolumn z wierszy spełniających kryteria podane w klauzuli **WHERE**. Natomiast drugie — polecenie **DELETE** — usuwa z tabeli wszystkie wiersze spełniające zadane kryteria. Należy pamiętać, że usuwane są całe wiersze.

Bądź poleceniem DELETE z klauzulą WHERE



Zjednocz się z grupą poleceń DELETE z podanymi klauzulami WHERE, w których są używane operatory AND i OR, i spróbuj określić, czy ich wykonanie spowoduje usunięcie z tabeli jakichś rekordów, czy też nie.

`DELETE FROM paczki_oceny`

Narysuj strzałki wskazujące, które wiersze zostaną usunięte przez każde z przedstawionych polecen:

`WHERE miejsce = 'Chrupki Król' AND ocena < 6;`

`WHERE miejsce = 'Chrupki Król' AND ocena = 3;`

`WHERE miejsce = 'Pyszny Rogal' AND ocena >= 6;`

`WHERE miejsce = 'Chrupki Król' OR ocena > 5;`

`WHERE miejsce = 'Chrupki Król' OR ocena = 3;`

`WHERE miejsce = 'Pyszny Rogal' OR ocena = 6;`

`paczki_oceny`

miejsce	godzina	data	typ	ocena	komentarze
Chrupki Król	8:50	27.9	zwyczajny z lukrem	10	niemal doskonaly
Pączki u Donalda	8:59	25.8	NULL	6	tłusty
Kafeteria Gwiezdny Pył	7:35	24.5	cynamonowy z lukrem	5	nie świeży, ale smaczny
Pączki u Donalda	7:03	26.4	z dżemem	7	za mało dżemu

Bądź polecением DELETE z klauzulą WHERE. Rozwiążanie



Zjednoczyłeś się z grupą poleceń DELETE z podanymi klauzulami WHERE, w których są używane operatory AND i OR, aby spróbować określić, czy ich wykonanie spowoduje usunięcie z tabeli jakichś rekordów, czy też nie.

`DELETE FROM paczki_oceny`

`WHERE miejsce = 'Chrupki Krol' AND ocena <> 6;`

`WHERE miejsce = 'Chrupki Krol' AND ocena = 3;`

`WHERE miejsce = 'Pyszny Rogal' AND ocena >= 6;`

`WHERE miejsce = 'Chrupki Krol' OR ocena > 5;`

`WHERE miejsce = 'Chrupki Krol' OR ocena = 3;`

`WHERE miejsce = 'Pyszny Rogal' OR ocena = 6;`

Narysuj strzałki wskazujące, które wiersze zostaną usunięte przez każde z przedstawionych poleceń:

Brak wierszy spełniających kryteria, wykonanie polecenia nie da żadnych efektów.

Brak wierszy spełniających kryteria, wykonanie polecenia nie da żadnych efektów.

Brak wierszy spełniających kryteria, wykonanie polecenia nie da żadnych efektów.

miejscie	godzina	data	typ	ocena	komentarze
Chrupki Król	8:50	27.9	zwyczajny z lukrem	10	niemal doskonały
Pączki u Donalda	8:59	25.8	NULL	6	tlusty
Kafeteria Gwiezdny Pył	7:35	24.5	cynamonowy z lukrem	5	nie świeży, ale smaczny
Pączki u Donalda	7:03	26.4	z dżemem	7	za mało dżemu

Te wartości NULL mogą Ci przysporzyć problemów w przyszłości. Warto umieścić w kolumnie jakąś wartość, a nie zostawiać w niej wartości NULL, gdyż jej występowania nie można sprawdzić przy użyciu operatora równości.

Dwa kroki — INSERT i DELETE

W naszej tabeli znajduje się tylko jeden rekord dotyczący Klarabeli. Ponieważ chcemy, by dla każdego klauna w tabeli istniał tylko jeden rekord zawierający najnowsze informacje, zatem musimy dodać do tabeli nowy rekord o Klarabeli i usunąć stary.

Jedynie jej nowa aktywność jest inną niż ta aktualnie zapisana w bazie.

Widziano Klarabelę tańczącą w domu opieki „Wesoła Wdówka”. „K, pomarańczowe włosy, ogromny kwiat, niebieska sukienka”

Naszym zadaniem było dodanie tych danych do tej tabeli. By zaoszczędzić nieco miejsca, pokazujemy tu jedynie jeden wiersz tabeli ze strony 165.

imie	ostatnio_widziano	wyglad	aktywnosci
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec

- 1 W pierwszej kolejności użyj polecenia INSERT, by dodać do tabeli nowe informacje (jednocześnie pozostawiając w niej stare).

```
INSERT INTO klowni_informacje
VALUES
( 'Klarabela', 'Dom opieki "Wesoła Wdówka"', 'K, różowe włosy,
ogromny kwiat, niebieska sukienka', 'taniec' );
```

Wstawiając nowy rekord, podaj w nim informacje z poprzedniego rekordu o tym samym klauniu; podawaj nowe dane tylko w tych polach, które uległy zmianie.

→

imie	ostatnio_widziano	wyglad	aktywnosci
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	taniec

- 2 Następnie usuń stary rekord, używając polecenia DELETE z odpowiednią klauzulą WHERE.

```
DELETE FROM klowni_informacje
WHERE
aktywnosci LIKE 'krzyk%'
AND imie = 'Klarabela';
```

Skonstruuj odpowiednią klauzulę WHERE, by odnaleźć i usunąć wcześniejszy rekord.

Teraz w tabeli znajduje się jedynie nowy rekord.

imie	ostatnio_widziano	wyglad	aktywnosci
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	taniec

Zaostrz ołówek

Zaostrz ołówek



Napisz polecenia INSERT i DELETE, by wprowadzić w tabeli drinki_informacje zmiany opisane poniżej. Następnie narysuj nową zawartość tabeli.

drinki_informacje

nazwa	cena	węglowodany	kolor	lód	kalorie
Blackthorn	7.50	8.4	żółty	T	33
Blue Moon	6.25	3.2	niebieski	T	12
Oh My Gosh	8.75	8.6	pomarańczowy	T	35
Lime Fizz	6.25	5.4	zielony	T	24
Kiss on the Lips	13.75	42.5	fioletowy	T	171
Hot Gold	8.00	32.1	pomarańczowy	N	135
Lone Tree	9.00	4.2	czerwony	T	17
Greyhound	10.00	14	żółty	T	50
Indian Summer	7.00	7.2	brązowy	N	30
Bull Frog	6.50	21.5	jasnobrązowy	T	80
Soda and It	9.50	4.7	czerwony	N	19

Zmień wartość energetyczną drinka Kiss on the Lips — obecnie ma ona wynosić 170 kalorii.

.....
.....
.....
.....
.....

Wszystkie kolory „żółty” zmień na „złoty”.

.....
.....
.....
.....
.....

drinki_informacje

nazwa	cena	węglowodany	kolor	lód	kalorie
Blackthorn					
Blue Moon					
Oh My Gosh					
Lime Fizz					
Kiss on the Lips					
Hot Gold					
Lone Tree					
Greyhound					
Indian Summer					
Bull Frog					
Soda and It					

Czy to kolejne z waszych
podchwytliwych ćwiczeń?



Teraz zmień cenę wszystkich drinków, które kosztowały 6,25,
na 8,75, a wszystkich drinków, które kosztowały 8,75, na 11,25.

.....

.....

.....

.....

.....

Zaostrz ołówek. Rozwiążanie



Zaostrz ołówek Rozwiążanie

Napisz polecenia INSERT i DELETE, by wprowadzić w tabeli drinki_informacje zmiany opisane poniżej. Następnie narysuj nową zawartość tabeli.

drinki_informacje

nazwa	cena	węglowodany	kolor	lód	kalorie
Blackthorn	7.50	8.4	żółty	T	33
Blue Moon	6.25	3.2	niebieski	T	12
Oh My Gosh	8.75	8.6	pomarańczowy	T	35
Lime Fizz	6.25	5.4	zielony	T	24
Kiss on the Lips	13.75	42.5	fioletowy	T	171
Hot Gold	8.00	32.1	pomarańczowy	N	135
Lone Tree	9.00	4.2	czerwony	T	17
Greyhound	10.00	14	żółty	T	50
Indian Summer	7.00	7.2	brązowy	N	30
Bull Frog	6.50	21.5	jasnobrązowy	T	80
Soda and It	9.50	4.7	czerwony	N	19

Zmień wartość energetyczną drinka Kiss on the Lips — obecnie ma ona wynosić 170 kalorii.

**INSERT INTO drinki_informacje VALUES ('Kiss on the Lips', 13.75, 42.5,
'fioletowy', 'T', 170);**

DELETE FROM drinki_informacje WHERE kalorie = 171;

Wszystkie kolory „żółty” zmień na „złoty”.

**INSERT INTO drinki_informacje VALUES ('Blackthorn', 7.50, 8.4, złoty,
'T', 33), ('Greyhound', 10.00, 14, złoty, 'T', 50);**

DELETE FROM drinki_informacje WHERE kolor = żółty;

drinki_informacje

nazwa	cena	węglowodany	kolor	lód	kalorie
Blackthorn	7.50	8.4	złoty	T	33
Blue Moon	6.25	3.2	niebieski	T	12
Oh My Gosh	8.75	8.6	pomarańczowy	T	35
Lime Fizz	6.25	5.4	zielony	T	24
Kiss on the Lips	13.75	42.5	fioletowy	T	170
Hot Gold	8.00	32.1	pomarańczowy	N	135
Lone Tree	9.00	4.2	czerwony	T	17
Greyhound	10.00	14	złoty	T	50
Indian Summer	7.00	7.2	brązowy	N	30
Bull Frog	6.50	21.5	jasnobrązowy	T	80
Soda and It	9.50	4.7	czerwony	N	19

Oto jak powinna wyglądać zawartość Twojej tabeli po wprowadzeniu w niej zmian. Być może kolejność rekordów w Twojej tabeli będzie nieco inna, jednak nie przejmuj się — pamiętaj, że tak naprawdę kolejność nie ma żadnego znaczenia.

Czy to kolejne z waszych podchwytliwych ćwiczeń?

To nie jest podchwytliwe pytanie, jednak powinieneś się nad nim trochę zastanowić. Jeśli najpierw zmienisz cenę 6,25 na 8,75, a następnie cenę 8,75 na 11,25, to okaże się, że podniesiesz cenę drinka Blue Moon o 4. A zatem w pierwszej kolejności powinieneś zmienić wyższą cenę (8,75 na 11,25), a dopiero potem niższą (6,25 na 8,75).



Teraz zmień cenę wszystkich drinków, które kosztowały 6,25, na 8,75, a wszystkich drinków, które kosztowały 8,75, na 11,25.

`INSERT INTO drinki_informacje VALUES ('Oh My Gosh', 11.25, 8.6, 'pomarańczowy', 'T', 35);`

`DELETE FROM drinki_informacje WHERE cena = 8.75;`

`INSERT INTO drinki_informacje VALUES ('Blue Moon', 8.75, 3.2, 'niebieski', 'T', 12),`

`('Lime Fizz', 8.75, 5.4, 'zielony', 'T', 24);`

`DELETE FROM drinki_informacje WHERE cena = 6.25;`

Dopisz sobie dodatkowe punkty, jeśli wszystkie nowe dane zapisałeś w tabeli, posługując się tylko jednym poleceniem INSERT.

Stosuj polecenie DELETE rozważnie

Za każdym razem, gdy stosujesz polecenie DELETE, narażasz się na ryzyko usunięcia nie tych rekordów, które planowałeś. Założmy na przykład, że chcemy dodać do tabeli nowy rekord dotyczący Pana Hobo.

Oto informacje, jakie chcemy dodać do tabeli, oraz polecenie INSERT, które to zrobi.

Widziano Pana Hobo
w restauracji McSteck

**INSERT INTO kloni_informacje
VALUES**

('Pan Hobo', 'Restauracja Mc\Steck', 'M, cygaro,
czarne włosy, malutki kapelusz', 'skrzypce');

**Stosuj polecenie DELETE
bardzo ostrożnie.**

Upewnij się, że wykorzystana w nim klauzula WHERE jest precyzyjna i zwraca dokładnie te wiersze, które chcesz usunąć.

Nie zapomnij o lewym ukośniku, który należy umieścić przed znakiem apostrofu.

imie	ostatnio_widziano	wygląd	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółta koszula, workowate czerwone spodnie	trąbka, parasolka
Pan Hobo	Cyrk Koloseum	M, cygaro, czarne włosy, niewielki kapelusz	skrzypce
Klarabelka	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Skuter	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
Balbina	Auta Edwarda	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
Gonzo		M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Zabawex	M, zielono-fioletowy kostium, szpiczasty nos	
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
Pani Smyk	Zabawex	K, żółta koszula, workowate niebieskie spodnie	trąbka, parasolka
Gonzo	Park Zielona Polana	M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Przyjęcie urodzinowe w Mc'Stecku	M, zielono-fioletowy kostium, szpiczasty nos	wchodzenie do małego samochodu
Pan Hobo	Przyjęcie u Eryka Szarzyńskiego	M, cygaro, czarne włosy, malutki kapelusz	skrzypce
USUNIĘTY			
Pan Hobo	Restauracja Mc'Steck	M, cygaro, czarne włosy, malutki kapelusz	skrzypce

A teraz spróbuj wcielić się w postać polecenia DELETE.

Bądź poleceniem DELETE



Poniżej znajdziesz grupę klauzul WHERE przeznaczonych do użycia w poleceniu DELETE, które mają przywrócić porządek w naszej tabeli klowni_informacje przedstawionej na poprzedniej stronie. Określ, które z nich faktycznie nam pomogą, a które przysporzą tylko nowych problemów.

`DELETE FROM klowni_informacje;`

Czy to polecenie nam pomoże?
Jeśli nie, to opisz dlaczego.

.....
.....

`WHERE ostatnio_widziano = 'Szpital miejski';`

.....
.....

`WHERE aktywnosci = 'skrzypce';`

.....
.....

`WHERE ostatnio_widziano = 'Park Zielona Polana'
AND imie = 'Pan Hobo';`

.....
.....

`WHERE ostatnio_widziano = 'Szpital miejski'
AND ostatnio_widziano = 'Park Zielona Polana';`

.....
.....

`WHERE ostatnio_widziano = 'Szpital miejski'
OR ostatnio_widziano = 'Park Zielona Polana';`

.....
.....

`WHERE imie = 'Pan Hobo'
OR ostatnio_widziano = 'Szpital miejski';`

.....
.....

A teraz napisz jedno polecenie DELETE, które usunie z tabeli wszystkie niepotrzebne rekordy dotyczące Pana Hobo, a przy tym nie usunie z bazy żadnych rekordów dotyczących innych klaunów.

.....
.....
.....
.....

Bądź poleceniem DELETE. Rozwiążanie



Poniżej znajdziesz grupę klauzul WHERE przeznaczonych do użycia w poleceniu DELETE, które mają przywrócić porządek w naszej tabeli kloni_informacje przedstawionej na poprzedniej stronie. Określ, które z nich faktycznie nam pomogą, a które przysporzą tylko nowych problemów.

DELETE FROM kloni_informacje;

✓ Rekord dotyczący Skutera także pasuje do tego warunku.

WHERE ostatnio_widziano = 'Szpital miejski';

Nie chcemy usuwać nowego rekordu.

WHERE aktywnosci = 'skrzypce';

WHERE ostatnio_widziano = 'Park Zielona Polana'
AND imie = 'Pan Hobo';

AND oznacza, że oba warunki muszą być spełnione.

WHERE ostatnio_widziano = 'Szpital miejski'
AND ostatnio_widziano = 'Park Zielona Polana';

WHERE ostatnio_widziano = 'Szpital miejski'
OR ostatnio_widziano = 'Park Zielona Polana';

WHERE imie = 'Pan Hobo' OR ostatnio_widziano
= 'Szpital miejski';

A teraz napisz jedno polecenie DELETE, które usunie z tabeli wszystkie niepotrzebne rekordy dotyczące Pana Hobo, a przy tym nie usunie z bazy żadnych rekordów dotyczących innych klaunów.

Czy to polecenie nam pomoże?
Jeśli nie, to opisz dlaczego.

Usuwa tylko jeden rekord Pana Hobo.

Usuwa także jeden rekord klauna Skuter.

Usuwa wszystkie rekordy Pana Hobo,
w tym także nowy.

Usuwa tylko jeden ze starych
rekordów Pana Hobo.

Niczego nie usuwa.

Wraz ze starymi rekordami Pana Hobo
usuwa także rekordy Skutera i Gonza.

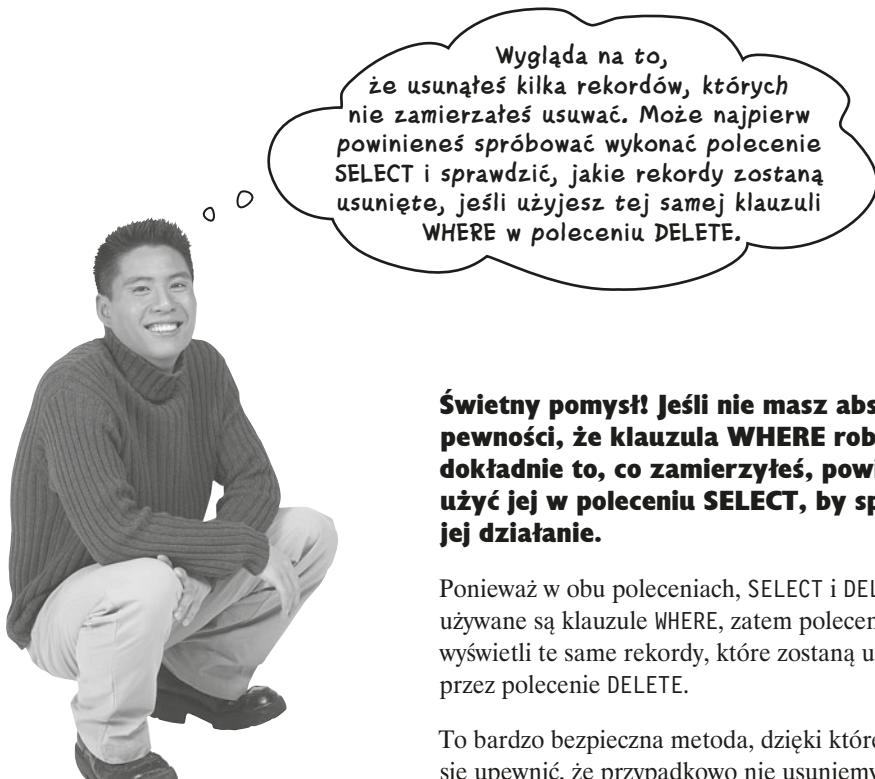
Usuwa wszystkie rekordy Pana Hobo
oraz rekordy Skutera.

DELETE FROM kloni_informacje

WHERE imie = 'Pan Hobo'

AND ostatnio_widziano < >

Restauracja Mc\Steek



Świetny pomysł! Jeśli nie masz absolutnej pewności, że klauzula WHERE robi dokładnie to, co zamierzyłeś, powinieneś użyć jej w poleceniu SELECT, by sprawdzić jej działanie.

Ponieważ w obu poleceniach, SELECT i DELETE, używane są klauzule WHERE, zatem polecenie SELECT wyświetli te same rekordy, które zostaną usunięte przez polecenie DELETE.

To bardzo bezpieczna metoda, dzięki której możemy się upewnić, że przypadkowo nie usuniemy żadnych niepożądanych rekordów. Dodatkowo pozwoli nam ona sprawdzić, czy usuniemy wszystkie rekordy, których chcemy się pozbyć.

Problemy z nieprecyzyjnymi poleceniami DELETE

Polecenie DELETE może przysporzyć problemów. Jeśli nie zachowamy ostrożności i uwagi, może usunąć niewłaściwe informacje. Możemy jednak uniknąć tego problemu, jeśli do naszej wcześniejszej sekwencji dwóch polecień INSERT i DELETE dodamy jeszcze jedno polecenie.

Nasza nowa TRÓJELEMENTOWA SEKWENCJA będzie się składać z następujących czynności:

W pierwszej kolejności, używając polecenia SELECT, pobierz rekordy, które chcesz usunąć.

- 1 W pierwszej kolejności spróbuj pobrać rekordy, które mają być usunięte, by upewnić się, że usuniesz wyłącznie te rekordy, które planujesz, i żadne inne.

**SELECT FROM klowni_informacje
WHERE
aktywnosci = 'taniec';**



imie	ostatnio_widziano	wyglad	aktywnosci
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec

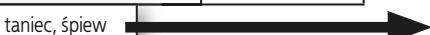
- 2 Następnie wstaw nowy rekord.

Wstaw nowy rekord. Umieść w nim te same dane, co w oryginalnym rekordzie, modyfikując tylko te pola, których wartości mają ulec zmianie.

**INSERT INTO klowni_informacje VALUES
('Zippo', 'Centrum Handlowe Skarbiec', 'K, pomarańczowy garnitur, workowate spodnie', 'taniec, spiew');**



imie	ostatnio_widziano	wyglad	aktywnosci
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec



- 3 W końcu usuń stare rekordy, używając w tym celu polecenia DELETE z tą samą klauzulą WHERE, której użyłeś w poleceniu SELECT w kroku 1.

```
DELETE FROM kloni_informacje
WHERE
aktywnosci = 'taniec';
```

Aby odszukać i usunąć niepotrzebne rekordy, zastosuj tę samą klauzulę WHERE, której użyłeś w poleceniu SELECT w kroku 1.

imie	ostatnio_widziano	wyglad	aktywnosci
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew

Teraz w tabeli mamy tylko nowy rekord o klauniu Zippo.

imie	ostatnio_widziano	wyglad	aktywnosci
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew



Czyż nie byłoby cudownie,
gdyby można było zmienić rekord,
używając tylko jednego polecenia,
i bez obaw, że nowy rekord zostanie
usunięty wraz ze starymi?
Ale wiem, że to zapewne jedynie
moje fantazje...

polecenie
DELETE

taniec	Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie
--------	-------	---------------------------	---

Modyfikowanie danych przy użyciu polecenia UPDATE

Obecnie nie powinieneś już mieć większych problemów z utrzymywaniem porządku w swoich tabelach i aktualizowaniem danych przy użyciu polecień INSERT i DELETE. Poszukujemy także sposobu, który pozwoliłby Ci na niejawne zastosowanie obu tych polecień jednocześnie w celu niejawnego zmodyfikowania wybranego wiersza tabeli.

Jednak zamiast wstawiać nowy wiersz tabeli i usuwać dotychczasowy, można by przecież wykorzystać już istniejący wiersz, modyfikując w nim tylko wybrane kolumny.

W języku SQL dostępne jest polecenie UPDATE, które służy właśnie do tego celu. Zapisuje ono podaną wartość we wskazanej kolumnie bądź kilka wartości w kilku określonych kolumnach. A co więcej, podobnie jak w poleceniach INSERT i DELETE, także w poleceniu UPDATE można stosować klauzulę WHERE, by precyzyjnie określić wiersze, które należy zmodyfikować.

Oto przykład zastosowania polecenia UPDATE:

UPDATE paczki_oceny

W tej klauzuli określamy nowe wartości. → **SET**

typ = 'lukrowany'

WHERE typ = 'zwyczajny z lukrem'; ← *To jest standardowa klauzula WHERE, dokładnie taka sama jak te, których używaliśmy w poleceniach SELECT i DELETE.*

Słowo kluczowe SET informuje system zarządzania bazami danych, że powinien zmienić kolumnę o nazwie podanej z lewej strony znaku równości i zapisać w niej wartość podaną z prawej strony znaku. W powyższym przykładzie zmieniamy wartość w kolumnie 'typ' na 'lukrowany'. Z kolei klauzula WHERE określa, że zmienione mają zostać wyłącznie rekordy, w których w kolumnie typ zapisana jest wartość 'zwyczajny z lukrem'.

paczki_oceny

miejsce	godzina	data	typ	ocena	komentarze
Chrupki Król	8:50	27.9	zwyczajny z lukrem	10	niemal doskonaly
Pačzki u Donalda	8:59	25.8	NULL	6	tłusty
Kafeteria Gwiezdny Pył	7:35	24.5	cynamonowy z lukrem	5	nieświeży, ale smaczny
Pačzki u Donalda	7:03	26.4	z dżemem	7	za mało dżemu



paczki_oceny

miejsce	godzina	data	typ	ocena	komentarze
Chrupki Król	8:50	27.9	lukrowany	10	niemal doskonaly
Pačzki u Donalda	8:59	25.8	NULL	6	tłusty
Kafeteria Gwiezdny Pył	7:35	24.5	cynamonowy z lukrem	5	nieświeży, ale smaczny
Pačzki u Donalda	7:03	26.4	z dżemem	7	za mało dżemu

Reguły stosowania polecenia UPDATE

- Polecenia UPDATE można używać do zmiany zawartości jednej lub kilku kolumn. Wystarczy dodać do klauzuli SET dodatkowe wyrażenia kolumna = wartość, oddzielając je od siebie przecinkami:

UPDATE tabela

```
SET pierwsza_kolumna = 'nowawartosc',
druga_kolumna = 'inna_wartosc';
```

- Polecenie UPDATE może modyfikować jeden wiersz lub grupę składającą się z większej ilości wierszy; zależnie od zastosowanej klauzuli WHERE.

Nie istniejąca grupa pytań

P: Co się stanie, jeśli pominę klauzulę WHERE?

O: W takim przypadku we wszystkich wierszach tabeli, w każdej z kolumn podanych w klauzuli SET zostaną zapisane nowe wartości.

P: W poleceniu UPDATE przedstawionym na poprzedniej stronie zostały użyte dwa znaki równości, przy czym wydaje mi się, że każdy z nich ma inne znaczenie. Czy mam rację?

O: Dokładnie. Znak równości użyty w klauzuli SET określa, że we wskazanej kolumnie należy zapisać podaną wartość; z kolei znak równości zastosowany w klauzuli WHERE pozwala sprawdzić, czy zawartość kolumny jest równa podanej wartości.

P: A czy mógłbym wprowadzić te same zmiany, używając poniższego polecenia:

```
UPDATE paczki_oceny SET typ = 'lukrowane'
WHERE miejsce = 'Chrupki Król';
```

O: Owszem, mógłbyś. Wykonanie tego polecenia spowodowałoby wprowadzenie identycznych modyfikacji w tym samym wierszu tabeli. W przypadku naszej przykładowej tabeli z czterema rekordami takie rozwiązanie dałoby zamierzone efekty. Gdybyś jednak wykonał powyższe polecenie w tabeli zawierającej setki lub tysiące rekordów, to w efekcie zostałby zmodyfikowany typ we wszystkich rekordach dotyczących kawiarni Chrupki Król.

P: O rany! A jak mogę się upewnić, że zmodyfikowane zostaną wyłącznie zamierzone rekordy?

O: Dokładnie tak samo, jak robisz to w przypadku polecenia DELETE. Jeśli nie masz absolutnej pewności, że klauzula WHERE działa prawidłowo, to powinieneś ją sprawdzić, używając polecenia SELECT.

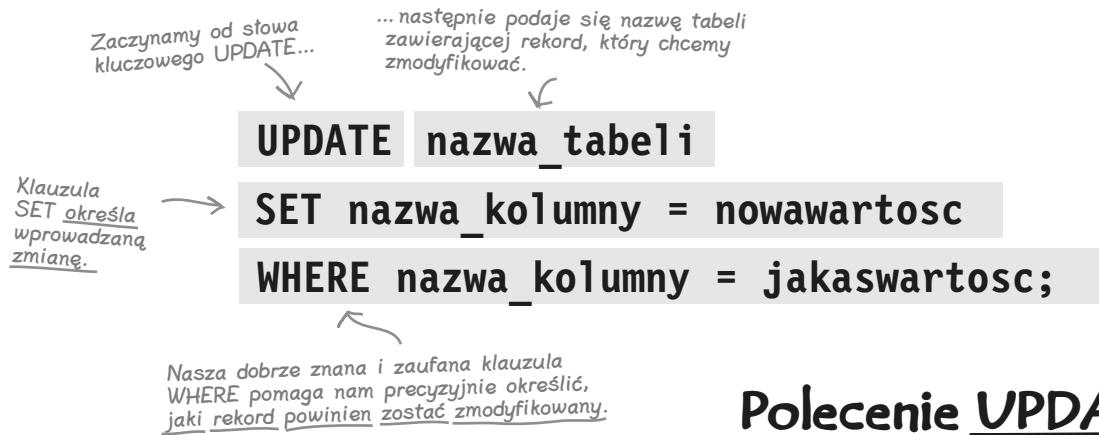
P: Czy w jednym poleceniu UPDATE można umieścić więcej niż jedną klauzulę SET?

O: Nie, jednak trudno wyobrazić sobie sytuację, w której musiałbyś zastosować takie rozwiązanie. W jednej klauzuli SET można bowiem określić nowe wartości dla wszystkich kolumn tabeli, co pokazaliśmy u góry strony.

Nigdy więcej żadnych kombinacji DELETE i INSERT

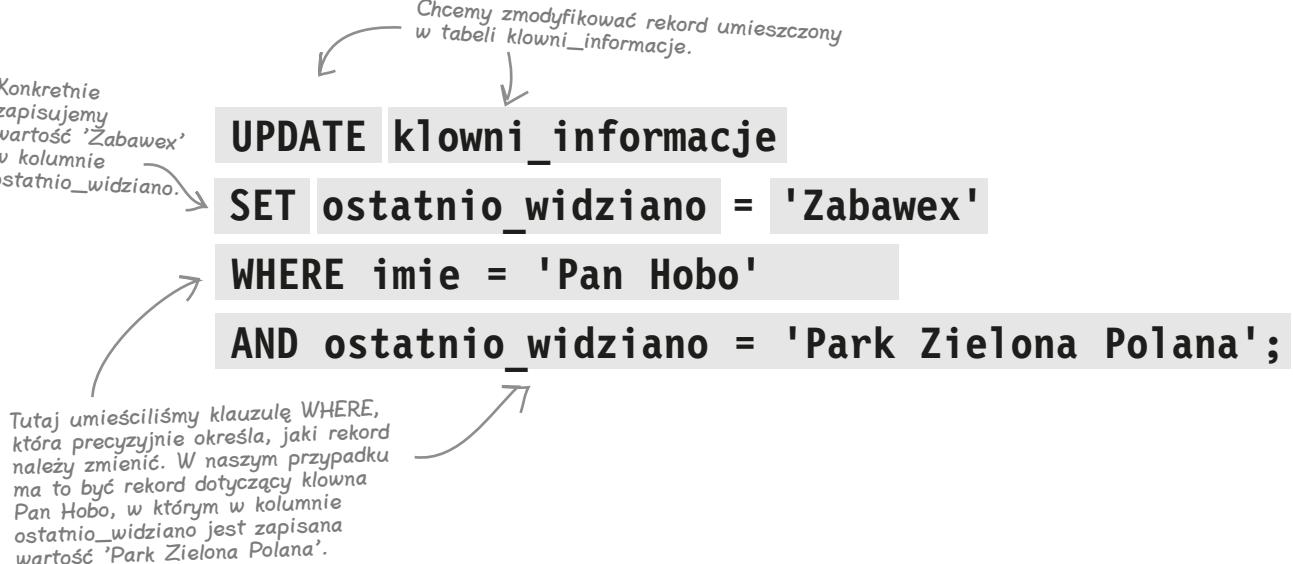
UPDATE odpowiada kombinacji INSERT-DELETE

Zastosowanie polecenia **UPDATE** *nie powoduje usunięcia jakichkolwiek rekordów* z tabeli. Zamiast tego istniejące wcześniej rekordy są **utylizowane poprzez zapisanie w nich aktualnych informacji.**



Zobaczmy teraz, jak polecenie UPDATE sprawdza się w praktyce — zastosujemy je do wprowadzenia zmian w tabeli kloni_informacje.

Polecenie **UPDATE** z powodzeniem może zastąpić kombinację poleceń **DELETE** oraz **INSERT**.



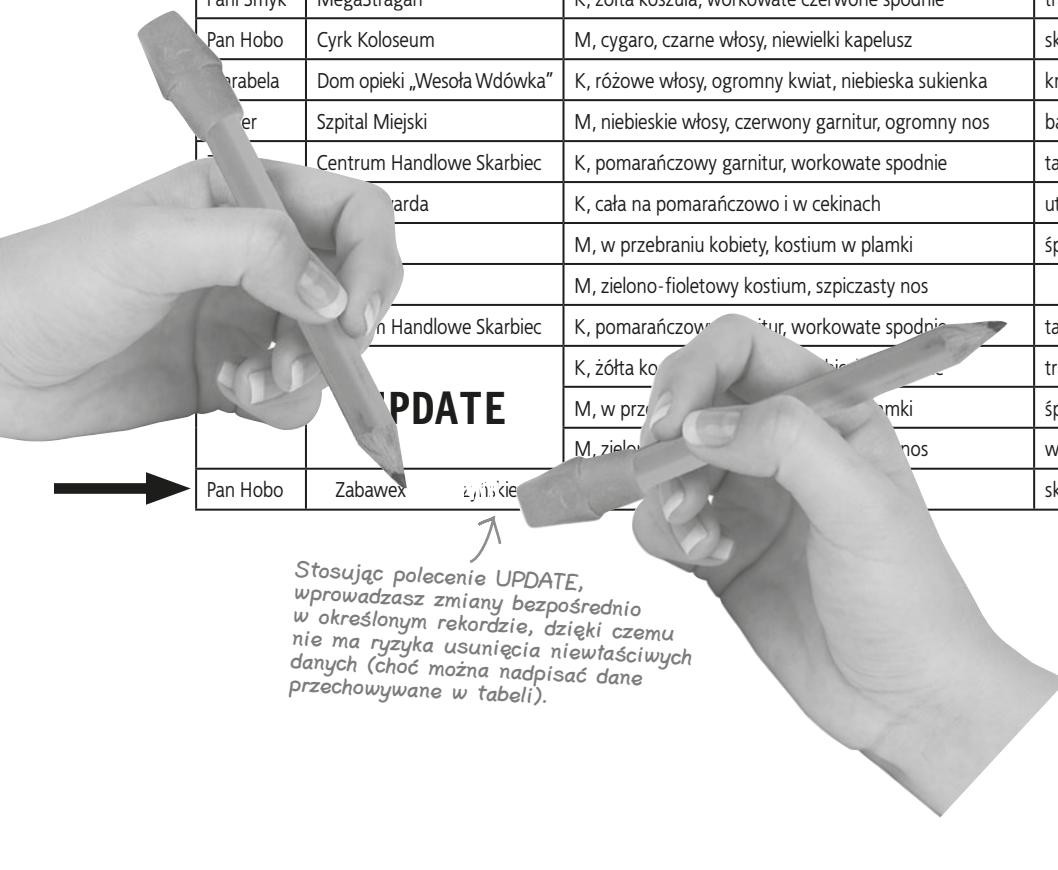
Polecenie UPDATE w akcji

Dzięki zastosowaniu polecenia UPDATE wartość kolumny `ostatnio_widziano` rekordu dotyczącego Pana Hobo uległa zmianie z 'Park Zielona Polana' na 'Zabawex'.

Pan Hobo został zauważony w siedzibie firmy Zabawex.

Oto informacje, jakie chcemy zapisać w tabeli, oraz polecenie UPDATE, którego możemy w tym celu użyć.

```
UPDATE kloni_informacje
SET ostatnio_widziano = 'Zabawex'
WHERE imie = 'Pan Hobo'
AND ostatnio_widziano = 'Przyjście u Eryka Szarzynskiego';
```



imie	ostatnio_widziano	wyglad	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółta koszula, workowate czerwone spodnie	trąbka, parasolka
Pan Hobo	Cyrk Koloseum	M, cygaro, czarne włosy, niewielki kapelusz	skrzypce
Barabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Mer	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
	Barbara	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
		M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
		M, zielono-fioletowy kostium, szpiczasty nos	
	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec, śpiew
		K, żółta koszula, spódniczka, biały kapelusz	trąbka, parasolka
		M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
		M, zielono-fioletowy kostium, szpiczasty nos	wchodzenie do małego samochodu
Pan Hobo	Zabawex		skrzypce

Stosując polecenie UPDATE, wprowadzasz zmiany bezpośrednio w określonym rekordzie, dzięki czemu nie ma ryzyka usunięcia niewłaściwych danych (choć można nadpisać dane przechowywane w tabeli).



Aktualizacja miejsc wystąpień klaunów

Tym razem zróbcmy wszystko jak należy. Napisz polecenia UPDATE zapisujące w tabeli nowe dane o klaunkach. Aby Ci ułatwić zadanie, napisaliśmy już pierwsze z nich. Następnie wypełnij tabelę `kłowni_informacje`, by zobaczyć, jak będzie wyglądać jej zawartość po wykonaniu tych poleceń.

Widziano Zippo śpiewającą.

UPDATE kłowni_informacje

SET aktynosc = 'spiew'

WHERE imie = 'Zippo';

Pani Smyk była widziana ostatnio w niebieskich workowatych spodniach.

.....
.....
.....
.....

Widziano Gonza w parku Zielona Polana.

.....
.....
.....
.....

Pana Smyka widziano ostatnio, jak usiłował wcisnąć się do małutkiego samochodu.

.....
.....
.....
.....

Pana Hobo widziano ostatnio na przyjęciu u Eryka Szarzyńskiego.

.....
.....
.....
.....

imie	ostatnio_widziano	wyglad	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółta koszula, workowate czerwone spodnie	trąbka, parasolka
Pan Hobo	Cyrk Koloseum	M, cygaro, czarne włosy, niewielki kapelusz	skrzypce
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Skuter	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
Balbina	Auta Edwarda	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
Gonzo		M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Zabawex	M, zielono-fioletowy kostium, szpiczasty nos	

imie	ostatnio_widziano	wyglad	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk			
Pan Hobo			
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Skuter	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
Zippo			
Balbina	Auta Edwarda	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
Gonzo			
Pan Smyk			

Zaostrz ołówek



Rozwiążanie Aktualizacja miejsc wystąpień kloonów

Twoim zadaniem było napisanie poleceń UPDATE zapisujących w tabeli nowe dane o kloonach oraz wypełnienie tabeli kloni_informacje, by przekonać się, jak będzie wyglądać jej zawartość po wykonaniu tych poleceń.

Widziano Zippo śpiewającą.

Pani Smyk była widziana ostatnio w niebieskich workowatych spodniach.

Widziano Gonza w parku Zielona Polana.

Pana Smyka widziano ostatnio, jak usiłował wcisnąć się do małutkiego samochodu.

Pana Hobo widziano ostatnio na przyjęciu u Eryka Szarzyńskiego.

UPDATE kloni_informacje

SET aktywnosci = 'śpiew'

WHERE imie = 'Zippo';

Nie chcemy usuwać z kolumny innych informacji o wyglądzie Pani Smyk. Uupeńj się, że nie zapomnisz ich podać.

UPDATE kloni_informacje

SET wyglad = 'K, żółta koszula, workowate niebieskie spodnie'

WHERE imie = 'Pani Smyk';

UPDATE kloni_informacje

SET ostatnio_widziano = 'Park Zielona Polana'

WHERE imie = 'Gonzo';

UPDATE kloni_informacje

SET aktywnosci = 'wchodzenie do małutkiego samochodu'

WHERE imie = 'Pan Smyk';

UPDATE kloni_informacje

SET ostatnio_widziano = 'Przyjęcie u Eryka Szarzyńskiego'

WHERE imie = 'Pan Hobo';

imie	ostatnio_widziano	wyglad	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółta koszula, workowate czerwone spodnie	trąbka, parasolka
Pan Hobo	Cyrk Koloseum	M, cygaro, czarne włosy, niewielki kapelusz	skrzypce
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Skuter	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
Balbina	Auta Edwarda	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
Gonzo		M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Zabawex	M, zielono-fioletowy kostium, szpiczasty nos	

Rekordy wydrukowane szarą czcionką
nie uległy zmianie, gdyż nie napisaliśmy
dla nich żadnych polecień UPDATE.

imie	ostatnio_widziano	wyglad	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółta koszula, workowate niebieskie spodnie	trąbka, parasolka
Pan Hobo	Przyjęcie u Eryka Szarzyńskiego	M, cygaro, czarne włosy, niewielki kapelusz	skrzypce
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Skuter	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	śpiew
Balbina	Auta Edwarda	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
Gonzo	Park Zielona Polana	M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Zabawex	M, zielono-fioletowy kostium, szpiczasty nos	wchodzenie do małutkiego samochodu

Zmieniły się jedynie fragmenty rekordów — te, które
wyszczególniłeś w klauzulach SET polecień UPDATE.
W końcu udało Ci się uzupełnić te puste miejsca,
które istniały w tabeli od jej utworzenia na stronie 155.

Zaktualizuj ceny drinków

Czy pamiętasz, jak próbowaliśmy zmieniać ceny wybranych drinków w tabeli `drinki_informacje`?
Chcieliśmy zmienić cenę 6,25 na 8,75, a 8,75 na 11,25.

`drinki_informacje`

nazwa	cena	węglowodany	kolor	lód	kalorie
Blackthorn	7.50	8.4	żółty	T	33
Blue Moon	6.25	3.2	niebieski	T	12
Oh My Gosh	8.75	8.6	pomarańczowy	T	35
Lime Fizz	6.25	5.4	zielony	T	24
Kiss on the Lips	13.75	42.5	fioletowy	T	171
Hot Gold	8.00	32.1	pomarańczowy	N	135
Lone Tree	9.00	4.2	czerwony	T	17
Greyhound	10.00	14	żółty	T	50
Indian Summer	7.00	7.2	brązowy	N	30
Bull Frog	6.50	21.5	jasnobrązowy	T	80
Soda and It	9.50	4.7	czerwony	N	19

Przekonajmy się, w jaki sposób możemy rozwiązać to zagadnienie, używając do tego grupy poleceń UPDATE, z których każde będzie modyfikować cenę jednego, konkretnego rekordu.

```
UPDATE drinki_informacje  
SET cena = 8.75 ← Cena po dodaniu 2,50 zł.  
WHERE nazwa = 'Blue Moon';
```

W klauzuli WHERE wybieramy kolumnę z unikalną wartością, dzięki czemu będziemy mieli pewność, jaki rekord tabeli zostanie zmodyfikowany.

Zaostrz ołówek



Dla każdego wiersza tabeli drinki_informacje napisz polecenie UPDATE, które powiększy cenę drinka w danym wierszu o 2,50 zł.

nazwa	cena	węglowodany	kolor	lód	kalorie
Blackthorn	7.50	8.4	żółty	T	33
Blue Moon	6.25	3.2	niebieski	T	12
Oh My	8.75	8.6	pomarańczowy	T	35
Lime	5.50	5.1	zielony	T	24



Chwileczkę. Czemu każecie nam wykonywać niepotrzebną robotę. Czy nie ma operatora, który pozwoliłby nam zaktualizować ceny wszystkich drinków przy użyciu jednego polecenia UPDATE? Bez konieczności tworzenia i wykonywania osobnego polecenia dla każdego z rekordów tabeli?

Masz rację.

Wygląda na to, że jakiś chytry operator mógłby nam ułatwić zadanie. Spróbujmy zatem zaktualizować ceny wszystkich drinków bez konieczności wykonywania odrębnych poleceń UPDATE dla każdego z nich... i ryzykując przy tym nadpisanie danych, które zmodyfikowaliśmy już wcześniej.

A chcemy tylko jednego polecenia UPDATE

Nasza kolumna z ceną zawiera dane liczbowe. A w języku SQL na **kolumnach liczbowych** można wykonywać proste *operacje matematyczne*. W naszym przypadku wystarczy dodać 2,50 do każdego wiersza, który należy zmodyfikować. Poniżej pokazaliśmy, jak można to zrobić:

```
UPDATE drinki_informacje  
SET cena = cena + 2.50  
WHERE nazwa = 'Blue Moon'  
OR nazwa = 'Oh My Gosh'  
OR nazwa = 'Lime Fizz';
```

Dodajemy 2,50 złotego do wszystkich drinków, które wymagają zmiany ceny (czyli tych, które kosztują 6,25 zł oraz 8,75 zł).

Nie istnieją głupie pytania

P: Czy w przypadku modyfikowania wartości liczbowych mogę wykonywać odejmowanie? Jakie inne operacje są dostępne?

O: Mnożenie, dzielenie, dodawanie i odejmowanie — z tych operacji matematycznych możesz korzystać.

P: Czy możecie mi podać przykład obrazujący, kiedy mógłbym chcieć zastosować operator mnożenia?

O: Oczywiście. Załóżmy, że w tabeli przechowujesz listę produktów, z których każdy ma cenę. Możesz użyć polecenia UPDATE, w którym pomnożysz cenę w każdym rekordzie przez określoną liczbę, by określić cenę danego produktu wraz z podatkiem.

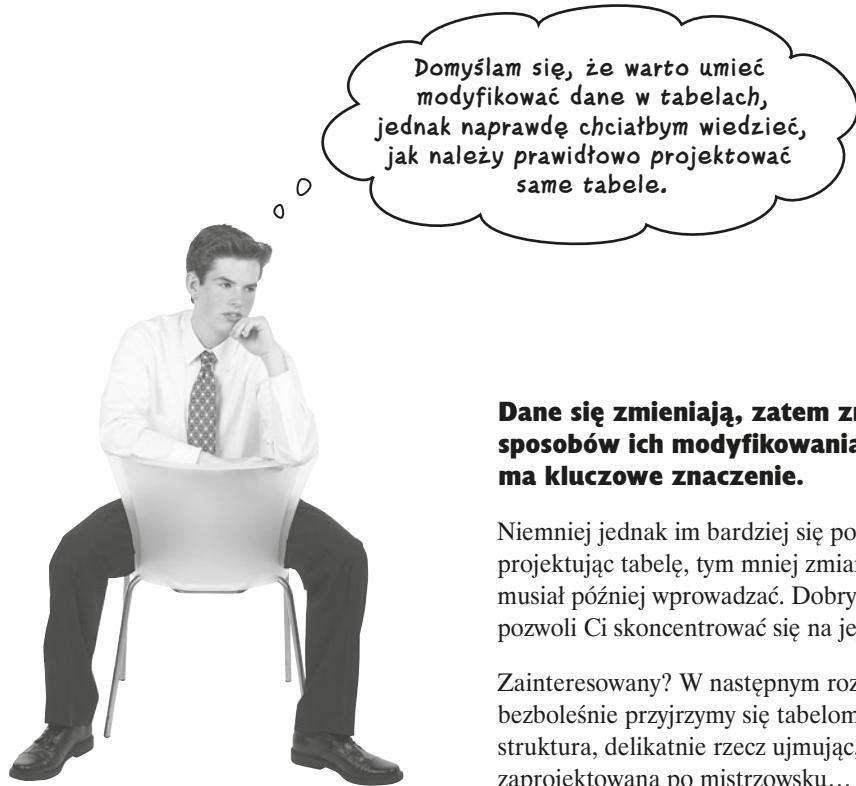
P: No dobrze, a czy oprócz tych prostych operacji matematycznych są jeszcze jakieś inne operatory, których można użyć do modyfikowania wartości w kolumnach tabel?

O: Jest ich całkiem sporo. W dalszej części książki pokażemy dodatkowe operacje, jakie można wykonywać na wartościach liczbowych, oraz operacje wykonywane na łańcuchach znaków.

P: Na łańcuchach znaków? Jakie to są operacje, podajcie, proszę, jakiś przykład.

O: No dobrze. Na przykład możesz użyć funkcji UPPER(), by zamienić wszystkie litery w łańcuchu znaków na wielkie. Podobnie możesz użyć funkcji LOWER(), by zmienić litery na małe.

Polecenia UPDATE mogą operować na większej ilości rekordów w tabeli. Można w nich używać podstawowych operatorów matematycznych, by modyfikować wartości liczbowe.



Dane się zmieniają, zatem znajomość sposobów ich modyfikowania ma kluczowe znaczenie.

Niemniej jednak im bardziej się postarasz, projektując tabelę, tym mniej zmian będziesz musiał później wprowadzać. Dobry projekt tabeli pozwoli Ci skoncentrować się na jej zawartości.

Zainteresowany? W następnym rozdziale bezboleśnie przyjrzymy się tabelom, których struktura, delikatnie rzecz ujmując, nie została zaprojektowana po mistrzowsku...



Przybornik SQL

Rozdział 3. już wkrótce przejdzie do historii. Zamieściliśmy tu jednak krótkie przypomnienie nowych poleceń SQL, które w nim poznałeś.

A zatem opanowałeś już materiał z trzeciego rozdziału książki i dodałeś do swojego SQL-owego przybornika znajomość operatorów. Kompletną listę porad znajdziesz w dodatku C, zamieszczonym na końcu książki.

DELETE

To narzędzie, którego możesz używać do usuwania wierszy z tabeli. Stosuj w nim klauzulę WHERE, by precyzyjnie określić wiersze, których chcesz się pozbyć.

UPDATE

To polecenie aktualizuje istniejącą kolumnę (lub kolumny), zapisując w niej nową wartość. Także w tym poleceniu można używać klauzuli WHERE.

SET

To słowo kluczowe tworzy klauzule używaną w poleceniach UPDATE — to właśnie w tej klauzuli określone są nowe wartości kolumn.

4. Projektowanie dobrych tabel

Po co być normalnym?



Dotychczas tworzyłeś tabele bez zwracania na nie szczególnej uwagi.

I wszystko było w porządku, tabele działały bez problemów. Mogłeś w nich zapisywać, modyfikować, usuwać i pobierać dane. Jednak w miarę zwiększania się ilości danych w tabelach zaczynasz zauważać, że są rzeczy, które mogłeś zrobić wcześniej, by ułatwić sobie w przyszłości tworzenie klauzul WHERE. Innymi słowy, musisz **znormalizować** swoje tabele.

Dwie wędkarskie tabele

Dwóch znajomych wędkarzy, Jacek i Marek, stworzyło tabele do gromadzenia danych o rekordowych połówach. Tabela Marka zawiera kolumny pozwalające na zapisanie łacińskiej nazwy gatunku ryby, nazwy polskiej, wagi złowionej ryby oraz miejsca dokonania połówu. Nie zawiera jednak kolumn pozwalających na zapisanie imienia i nazwiska osoby, która ustanowiła rekord.

polowy_informacje

nazwa	nazwa_gatunkowa	miejsce	waga
bass	M. salmoides	Wigry, PD	1,23 kg
sandacz	S. vitreus	Dziubiele, WM	2,75 kg
pstrąg	O. Clarki	Mrzygłód, PK	1,20 kg
okoń	P. Flavescens	Pisz, WM	0,85 kg
płotka	R. rutilus	Charzykowy, PM	0,65 kg
łuskot	L. Osseus	Czaplinek, ZP	1,10 kg
węgorz	A. anguilla	Swornegacie, PM	1,45 kg
szczupak	E. americanus	Karwica, WM	3,34 kg
złota rybka	C. auratus	Warszawa, MZ	0,35 kg
łosoś	O. Tshawytscha	Toruń, KP	3,10 kg

Ta tabela ma jedynie cztery kolumny. Porównaj ją z tabelą rekordowe_polowy przedstawioną na następnej stronie.



Tabela Jacka także zawiera polską nazwę złowionej ryby oraz jej wagę; jednak oprócz tego Jacek umieścił w niej kolumny pozwalające na zapisanie imienia i nazwiska szczęśliwego wędkarza oraz nazwy województwa, w którym dokonano połówu.

Także ta tabela służy do rejestrowania rekordowych potów wędkarskich, jednak zawiera niemal dwukrotnie więcej kolumn.

rekordowe_połów

imie	nazwisko	nazwa	miejsce	województwo	waga	data
Jan	Kowalski	bass	Wigry	PD	1,23 kg	5.9.1947
Adrian	Bródka	sandacz	Dziubiele	WM	2,75 kg	16.8.1960
Zenon	Krawczyk	pstrąg	Mrzygłód	PK	1,20 kg	23.6.1978
Maria	Popiela	okoń	Pisz	WM	0,85 kg	18.5.1934
Piotr	Drymza	plotka	Charzykowy	PM	0,65 kg	1.8.1965
Ignacy	Wiktorczyk	łuskot	Czaplinek	ZP	1,10 kg	31.9.1988
Krzysztof	Dubała	wegorz	Swornegacie	PM	1,45 kg	12.8.1973
Paweł	Wronek	szczupak	Karwića	WM	3,34 kg	11.6.1995
Andrzej	Książewicz	złota rybka	Warszawa	MZ	0,35 kg	25.9.2003
Roman	Wiertek	łosoś	Toruń	KP	3,10 kg	17.8.1991

Zaostrz ołówek



Dla **obu tabel** napisz zapytanie, które pobierze wszystkie rekordowe połów dokonane w województwie podkarpackim.

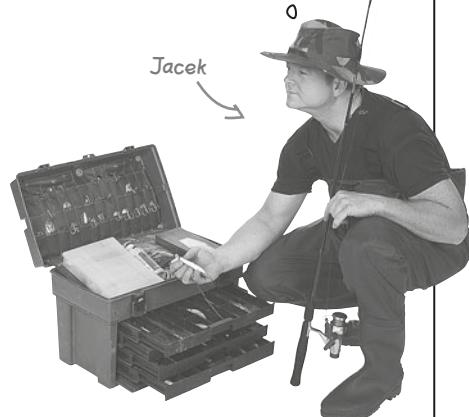
Piszemy artykuły dla magazynu „Weekend z wędką”. Muszę znać imiona i nazwiska wędkarzy, którzy ustanowili rekordowe połów, daty tych połówów oraz ich miejsca.

.....

.....

.....

.....



Rozwiążanie ćwiczenia

Zaostrz ołówek



Rozwiążanie

Dla każdej z tabel napisz zapytanie, które pobierze informacje o rekordowych połówach dokonanych na terenie województwa podkarpackiego.

Niemal nigdy nie muszę wyszukiwać informacji na podstawie województwa. Dlatego informacje o województwie zapisuję w tabeli w tej samej kolumnie, w której umieszczam nazwę miejsca, gdzie dokonano rekordowego połowa.

Musimy zastosować operator `LIKE` i odzukać interesujące nas rekordy na podstawie pola zawierającego połączoną nazwę miejscowości i oznaczenie województwa.

`SELECT * FROM polowy_informacje`

`WHERE miejsce LIKE '%PK';`



nazwa	nazwa_gatunkowa	miejscie	waga
pstrąg	O. Clarki	Mrzygłód, PK	1,20 kg

Często muszę przeszukiwać informacje na podstawie województwa, dlatego też utworzyłem w tabeli odrębną kolumnę określającą województwo, w którym dokonano rekordowego połowa.

To zapytanie odwołuje się bezpośrednio do kolumny „województwo”.

`SELECT * FROM rekordowe_polowy`

`WHERE województwo = 'PK';`



imie	nazwisko	nazwa	miejscie	województwo	waga	data
Zenon	Krawczyk	pstrąg	Mrzygłód	PK	1,20 kg	23.6.1978

.....
Nie istnieja
głupie pytania

P: A zatem tabela Jacka jest lepsza od tabeli Marka?

O: Nie. To dwie różne tabele, stworzone w innych celach. W praktyce Marek rzadko kiedy będzie musiał poszukiwać w swojej tabeli informacji na podstawie województwa, gdyż tak naprawdę interesują go jedynie nazwy gatunkowe i potoczne złowionych ryb oraz, oczywiście, ich waga.

Z drugiej strony, Jacek będzie musiał korzystać z informacji o województwie podczas poszukiwania danych w swojej tabeli. To właśnie z tego powodu w jego tabeli informacje o województwie, w jakim dokonano rekordowych połówów, znalazły się w osobnej kolumnie. To mu ułatwi poszukiwanie danych na podstawie województwa.

P: Czy podczas przeszukiwania tabel powinniśmy unikać stosowania operatora LIKE?

O: Nic nie przemawia za tym, by nie stosować operatora LIKE, niemniej jednak może to przysporzyć pewnych problemów oraz prowadzić do otrzymywania niepożądanych wyników. Jeśli w kolumnach są zapisywane złożone informacje, to operator LIKE nie będzie dostatecznie precyzyjny, by je pobierać.

P: Dlaczego krótsze zapytania są lepsze do długich?

O: Im prostsze jest zapytanie, tym lepiej. Zapytania będą się stawać coraz bardziej skomplikowane wraz z powiększaniem się bazy danych i ilości umieszczonych w niej tabel. Jeśli na samym początku zaczniesz od możliwie jak najprostszych zapytań, to w przyszłości na pewno tego nie pożałujesz.

P: Chcesz przez to powiedzieć, że zawsze powinieneś przechowywać w kolumnach bardzo małe fragmenty informacji?

O: Niekoniecznie. Analizując tabele Jacka i Marka, możesz już zauważyc, że wszystko zależy od tego, w jaki sposób chcesz korzystać z tabel. Na przykład wyobraź sobie tabele dotyczące samochodów — jedną, z której korzysta warsztat mechaniczny, i drugą, używaną przez właściciela komisu samochodowego. Mechanicy mogą potrzebować szczegółowych informacji o każdym samochodzie; z kolei właścicielowi komisu wystarczy marka, model, rok produkcji oraz numer nadwozia.

P: A wyobraźmy sobie adres pocztowy. Czy nie moglibyśmy utworzyć jednej kolumny, w której byłby zapisany cały adres, oraz kilku innych, w których umieścilibyśmy jego poszczególne elementy?

O: Choć takie powielanie danych może Ci się obecnie wydawać całkiem dobrym pomysłem, to jednak pomysł, o ile więcej miejsca na dysku zajmie taka baza, gdy rozrośnie się do potężnych rozmiarów. Poza tym w przypadku powielania danych w poleceniach INSERT i UPDATE pojawią się dodatkowe kolumny, o których będziesz musiał pamiętać.

Przyjrzyjmy się dokładniej, jak należy projektować tabele, by optymalnie pasowały do sposobów, w jakie będziemy z nich korzystać.

Sposób, w jaki masz zamiar korzystać z danych, będzie determinował postać tworzonych tabel.



WYSIL
SZARE KOMÓRKI

SQL jest językiem używanym w relacyjnych bazach danych.
Jak uważasz, czym są te „relacje” w świecie baz danych?

Tabele dotyczą związków

Bazy danych obsługiwane przy użyciu języka SQL są nazywane systemami zarządzania relacyjnymi bazami danych (ang. Relational Database Management System, w skrócie RDBMS). Ale nie zaprzątaj sobie głowy zapamiętywaniem tych nazw. Dla nas najważniejsze jest tylko jedno słowo: RELACYJNE*. Dla Ciebie oznacza ono mniej więcej tyle, że aby zaprojektować odrębową tabelę, musisz zastanowić się i określić, jak poszczególne kolumny są ze sobą powiązane i wspólnie opisują zagadnienie, jakiego dotyczy tabela.

Całe wyzwanie i sztuka polega na tym, by opisać zagadnienie, wykorzystując do tego celu kolumny, które zagwarantują łatwość pobierania danych z tabeli. A to oczywiście zależy od tego, jakie informacje chcemy pobierać z tabeli. Można wyróżnić kilka bardzo ogólnych kroków, jakie należy wykonać, projektując tabele.

1. Wybierz jedno zagadnienie, które ma opisywać tabela.

Czego mają dotyczyć informacje zapisywane w tabeli?

2. Utwórz listę informacji o danym zagadnieniu, których będziesz potrzebował podczas korzystania z tabeli.

Jak będziesz korzystać z tabeli?

3. Na podstawie tej listy podziel informacje o zagadnieniu na elementy, których następnie będziesz mógł użyć, określając organizację tabeli.

Jak będzie Ci najłatwiej przeszukiwać i pobierać dane z tabeli?

* Niektórzy uważają, że słowo „relacyjny” odnosi się do wielu tabel powiązanych ze sobą. Jednak jest to błędna opinia.

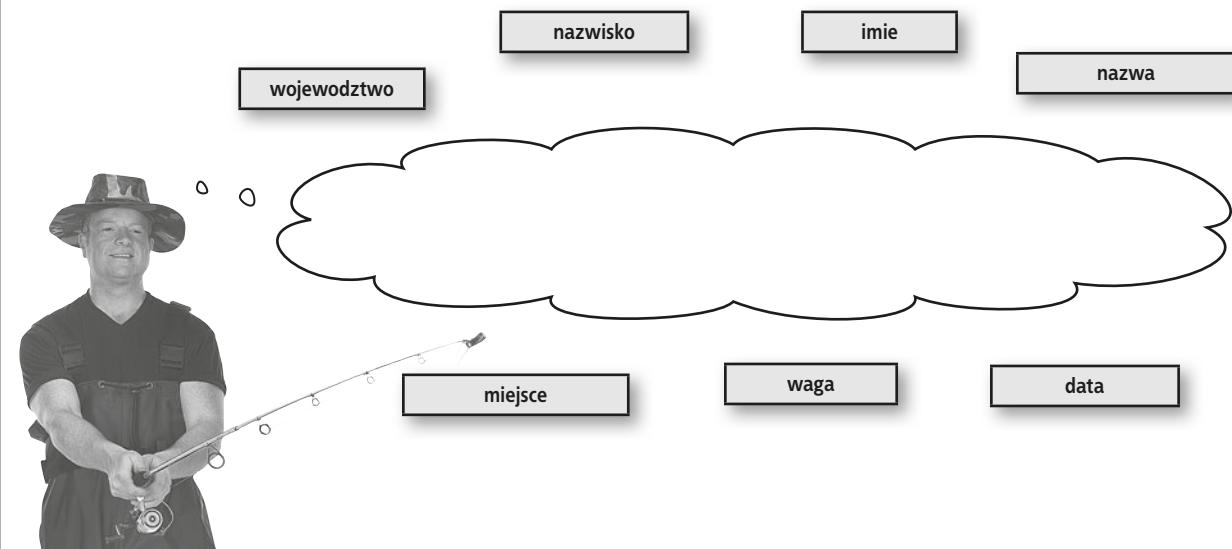


Ćwiczenie

Czy na podstawie przedstawionego poniżej zdania, określającego, w jaki sposób iecholog Marek chce przeszukiwać i pobierać dane z tabeli, potrafisz określić, jakie powinny być jej kolumny? Wpisz nazwy kolumn w pustych prostokątach na poniższym rysunku.



Twoja kolej. Napisz analogiczne zdanie dotyczące Jacka — autora artykułów dla magazynu „Weekend z wędką”, który korzysta z bazy, by notować w niej szczegółowe informacje na potrzeby swoich artykułów. Następnie narysuj strzałki prowadzące od nazw kolumn do miejsca w zdaniu, w którym Jacek nawiązuje do danej kolumny.

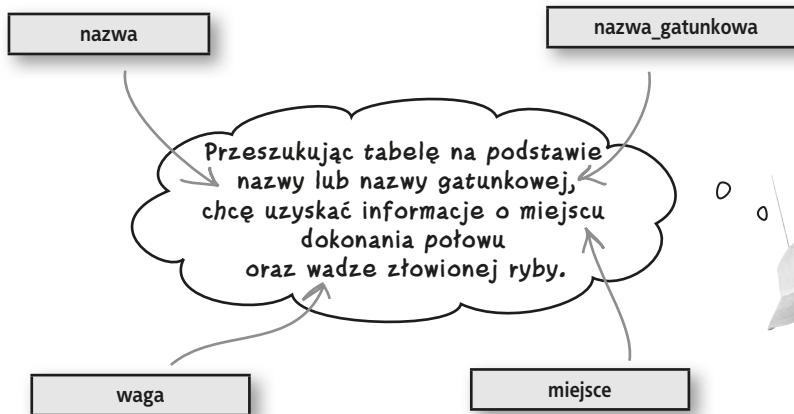


Ćwiczenie. Rozwiążanie

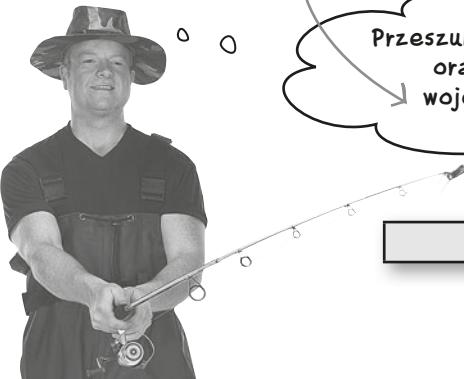
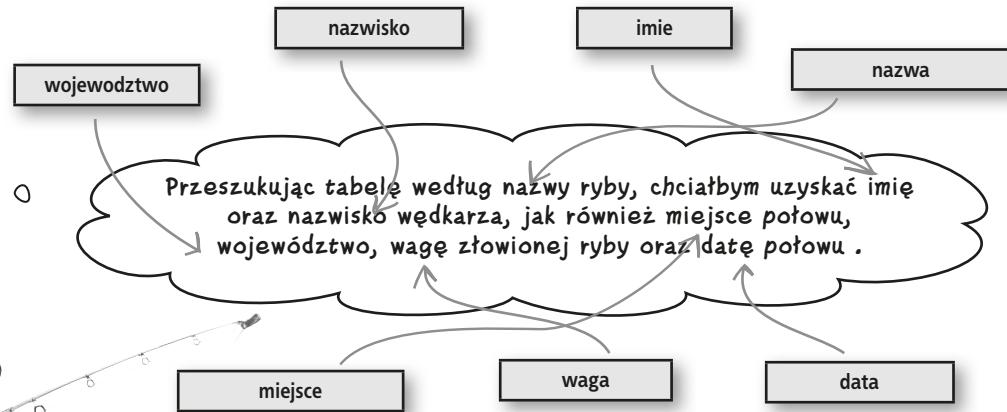


Rozwiążanie ćwiczenia

Czy na podstawie przedstawionego poniżej zdania określającego, w jaki sposób iecholog Marek chce przeszukiwać i pobierać dane z tabeli, potrafisz określić, jakie powinny być jej kolumny? Wpisz nazwy kolumn w pustych prostokątach na poniższym rysunku.



Twoja kolej. Napisz analogiczne zdanie dotyczące Jacka — autora artykułów dla magazynu „Weekend z wędką”, który korzysta z bazy, by notować w niej szczegółowe informacje na potrzeby swoich artykułów. Następnie narysuj strzałki prowadzące od nazw kolumn do miejsca w zdaniu, w którym Jacek nawiązuje do danej kolumny.





Ale dlaczego Jacek ma poprzestawać na tym?
Dlaczego nie podzieli daty na kolumny dni,
miesiący i lat? Nawet kolumnę określającą miejsce
można by dalej podzielić na osobne kolumny
z nazwą ulicy i numerem domu.

Oczywiście, że można by tak zrobić. Jednak nasze dane nie muszą być podzielone aż tak dokładnie.

Przynajmniej nie w tym przypadku. Gdyby jednak Jacek miał zamiar napisać artykuł o tym, gdzie pojechać na wakacje, by złapać dużą rybę, to w takim przypadku mógłby podzielić kolumnę „miejsc” na nazwę ulicy i numer, tak by czytelnicy mogli znaleźć nocleg jak najbliżej rekordowego łowiska.

Jednak Jacek potrzebuje wyłącznie informacji o miejscu i województwie, dlatego utworzył tylko tyle kolumn, ile jest koniecznych, by niepotrzebnie nie powiększać rozmiaru bazy danych. Uznał, że w jego sytuacji nie ma sensu bardziej dzielić danych; innymi słowy, uznał, że jego informacje są danymi *atomowymi*.



WYSIL
SZARE KOMÓRKI

Jak myślisz, co oznacza termin „dane atomowe” w kontekście informacji zapisywanych w relacyjnych bazach danych?

Dane atomowe

Czym jest atom? To niewielki fragment materii, którego nie można lub nie należy dalej dzielić. To samo dotyczy danych. Kiedy zostaną one uznane za dane ATOMOWE, oznacza to, że zostały one już podzielone na **najmniejsze elementy, których nie należy dalej dzielić**.

Dostawa w 30 minut lub gratis

Przyjrzyjmy się na przykład dostarczycielowi pizzy. Aby dostarczyć zamówienie w odpowiednie miejsce, wystarczy, że w jednej kolumnie zapiszemy nazwę ulicy i numer domu. Na jego potrzeby są to dane atomowe. Dostawca nigdy nie będzie poszukiwał samego numeru domu.

W rzeczywistości, jeśli miejsce dostawy została podzielone na nazwę ulicy i numer domu, to zapytania, które musiałby zadawać dostawca, byłyby dłuższe i bardziej złożone, a to spowodowałoby wydłużenie czasu dostarczania pizzy do klienta.



Na potrzeby dostawcy pizzy
adres zapisany w jednej
kolumnie jest wystarczająco
atomową informacją.

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
+-----+
| numer_zamowienia | adres          |
+-----+
|      250 | Mickiewicza 46
|      251 | Rozanskiego 15/2
|      253 | Podlesnicka 23
|      254 | Aleje Zwyciestwa 28/5
|      255 | Szeroka 16
|      256 | Targowa 87/9
|      257 | Zolkiewskiego 15
|      258 | Wilcza 26/2
|      259 | Skowronkow 2/2
+-----+
9 rows in set <0.00 sec>

mysql> SELECT adres FROM pizza_nawynos WHERE numer_zamowienia = 253;
+-----+
| adres          |
+-----+
| Podlesnicka 23 |
+-----+
1 row in set <0.01 sec>

mysql>
```

Lokalizacja, lokalizacja, lokalizacja

A teraz przeanalizujmy przykład pośrednika handlu nieruchomościami. Taki pośrednik mógłby chcieć, by nazwa ulicy została zapisana w osobnej kolumnie. Mógłby bowiem chcieć podać w zapytaniu nazwę ulicy, by uzyskać informacje o wszystkich domach na sprzedaż położonych przy niej. A zatem w przypadku pośrednika handlu nieruchomościami informacjami atomowymi są nazwa ulicy oraz numer domu.



Z kolei w przypadku pośrednika handlu nieruchomościami rozdzielenie nazwy ulicy od numeru domu i zapisanie ich w osobnych kolumnach pozwoli mu odnajdywać wszystkie domy na sprzedaż położone przy konkretnej ulicy za pomocą jednego, prostego zapytania.

ulica_numer	ulica_nazwa	typ_lokalu	cena
23	Al. Niepodleglosci	dom	450000
45/11	Kormoranow	m. wlasnosciowe	230000
90	Warszawska	dom, blizniak	460000
53/2	Olimpijska	apartament	500000
1011	Warszawska	dom	700000
14/1	Pszczynska	m. wlasnosciowe	320000
15	Sowinskiego	dom	380000
16/2	Krucza	apartament	380000
156	Kosynierow	dom	510000
89/3	Sylabusa	apartament	550000

```

10 rows in set <0.00 sec>

mysql> SELECT cena, typ_lokalu FROM nieruchomosci WHERE ulica_nazwa = 'Warszawska';
+-----+-----+
| cena | typ_lokalu |
+-----+-----+
| 460000 | dom, blizniak |
| 700000 | dom |
+-----+-----+
2 rows in set <0.00 sec>

mysql>

```

Dane atomowe a Twoje tabele

Poniżej przedstawiliśmy kilka pytań, które możesz sobie zadać, aby ułatwić sobie określenie danych, jakie należy umieścić w tworzonych tabelach.



1. Co jest podstawowym zagadnieniem opisywanym przez tabelę?

Czy tabela opisuje kłownów, krowy, pączki, czy też polityków?



2. W jaki sposób będziesz korzystał z tabeli, by uzyskiwać informacje o tym zagadnieniu?

Projektuj tabele w taki sposób, aby przeszukiwanie ich było jak najprostsze.



3. Czy kolumny tabeli zawierają dane atomowe, dzięki czemu używane zapytania mogą być krótkie i precyzyjne?

Nie istniejąca grupa pytań

P: Czyż atomy nie są bardzo małe? Czy nie powiniem zatem dzielić swoich informacji na naprawdę bardzo małe elementy?

O: Nie. Tworzenie danych atomowych oznacza podzielenie ich na najmniejsze elementy konieczne do stworzenia wydajnych tabel, a nie na podzielenie ich na najmniejsze możliwe elementy.

Nie należy dzielić danych bardziej niż to konieczne. Jeśli nie potrzebujesz dodatkowych kolumn, to nie twórz ich tylko i wyłącznie dlatego, że mógłbyś to zrobić.

P: W czym może mi pomóc stosowanie danych atomowych?

O: Może Ci pomóc w zapewnieniu, że informacje przechowywane w tabeli będą precyzyjne. Jeśli na przykład utworzysz osobną kolumnę przeznaczoną do przechowywania numeru domu, to będziesz mógł upewnić się, że będą w niej zapisywane wyłącznie liczby.

Stosowanie danych atomowych pozwala także poprawić efektywność wykonywanych zapytań, gdyż same zapytania są łatwiejsze do napisania, a czas ich wykonania jest krótszy. Korzyści, jakie zapewnia nam stosowanie danych atomowych, są tym wyraźniejsze, im więcej jest danych przechowywanych w tabeli.



Zaostrz ołówek

Poniżej przedstawiliśmy dwie oficjalne reguły dotyczące danych atomowych. Dla każdej z nich narysuj dwie hipotetyczne tabele, które nie będą spełniać wytycznych reguły.

Reguła 1.: W kolumnie z danymi atomowymi w jednym wierszu tabeli nie może się znajdować kilka wartości tego samego typu.

Tej reguły nie spełnia tabela Grześka — moje_kontakty, a konkretnie jej pole „zainteresowania”.

Reguła 2.: W tabeli zawierającej dane atomowe nie może być kilku kolumn zawierających dane tego samego typu.

Tej reguły nie spełnia tabela proste_drinki.

Zaostrz ołówek

Rozwiążanie

Poniżej przedstawiliśmy dwie oficjalne reguły dotyczące danych atomowych. Dla każdej z nich narysuj dwie hipotetyczne tabele, które nie będą spełniać wytycznych reguły.

Reguła 1.: W kolumnie z danymi atomowymi w jednym wierszu tabeli nie może się znajdować kilka wartości tego samego typu.

Oczywiście Twoja odpowiedź na pewno będzie inna, ale oto jeden z możliwych przykładów:

pożywienie	składniki
chleb	drożdże, mleko, jajka, mąka
sałatka	pomidor, ogórek, sałata

Czy pamiętasz tabelę Grześka? Zawierała ona kolumnę zainteresowania, w której Grzesiek zapisywał niejednokrotnie kilka różnych zainteresowań danej osoby, przez co przeszukiwanie tabeli było prawdziwym koszmarem.

Ta sama sytuacja występuje w tej tabeli. Wyobraź sobie odszukanie pomidorów wśród tych wszystkich pozostałych składników.

Reguła 2.: W tabeli zawierającej dane atomowe nie może być kilku kolumn zawierających dane tego samego typu.

nauczyciel	student1	student2	student3
Pani Martini	Janek	Romek	Kasia
Pan Grog	Sonia	Tymon	Julia

Zbyt wiele kolumn do podawania studentów!



Ćwiczenie

Skoro już znasz oficjalne reguły oraz trzy kroki pozwalające na stosowanie danych atomowych, przeanalizuj wszystkie tabele przedstawione do tej pory w książce i wyjaśnij, dlaczego zawierają one dane atomowe, albo co sprawia, że zapisywane w nich informacje nie są danymi atomowymi.

Tabela Grześka, ze strony 83

Tabela z ocenami pączków, ze strony 112

Tabela z informacjami o kłownach, strona 155

Tabela drinków, ze strony 93

Tabela połówów wędkarskich, strona 194

Dlaczego warto być normalnym?

Kiedy wyczerpie się Twój limit godzin na konsultacje ze specjalistą do spraw baz danych i będziesz musiał zatrudnić projektantów baz SQL, fajnie by było, gdybyś nie musiał tracić cennych godzin na tłumaczenie im, jak działają Twoje tabele.

Cóż, tworzenie tabel ZNORMALIZOWANYCH oznacza, że są one zgodne z pewnymi standardami, które projektanci baz danych będą rozumieć. Co więcej, na pewno ucieczy Cię fakt, iż nasze tabele zawierające dane atomowe są już w połowie drogi do owej „normalności”.

Zapisywanie w tabeli danych atomowych jest pierwszym krokiem na drodze do tworzenia tabel ZNORMALIZOWANYCH.



Rozwiążanie ćwiczenia

Skoro już znasz oficjalne reguły oraz trzy kroki pozwalające na stosowanie danych atomowych, przeanalizuj wszystkie tabele przedstawione do tej pory w książce i wyjaśnij, dlaczego zawierają one dane atomowe, albo co sprawia, że zapisywane w nich informacje nie są danymi atomowymi.

Tabela Grześka, ze strony 83 Nie jest atomowa. Kolumny „zainteresowania” i „szuka” nie są zgodne z regułą 1.

Tabela z ocenami pączków, ze strony 112 Tabela jest atomowa. W odróżnieniu od kolumn tabeli proste_drinki każda kolumna tej tabeli przechowuje informacje różnego typu. Oprócz tego, w odróżnieniu od kolumny „aktywnosci” tabeli kloonów w każdej z kolumn tej tabeli przechowywany jest tylko jeden element informacji.

Tabela z informacjami o kloonach, strona 155 Tabela nie jest atomowa. W niektórych rekordach w kolumnie „aktywnosci” zapisywanych jest więcej czynności niż jedna, co jest sprzeczne z regułą 1.

Tabela drinków, ze strony 93 Tabela nie jest atomowa. Zawiera ona więcej niż jedną kolumnę „składnik”, co jest sprzeczne z regułą 2.

Tabela połówów wędkarskich, strona 194 Tabela jest atomowa. W każdej kolumnie są zapisywane informacje innego typu. Każda kolumna zawiera także tylko jedną informację.

Zalety normalizacji tabel

1. W tabelach znormalizowanych dane się nie powielają, co pozwala ograniczyć wielkość bazy.

Unikanie powielania danych pozwoli Ci zaoszczędzić miejsce na dysku.

2. Dzięki mniejszej ilości informacji w bazie wszelkie zapytania będą wykonywane szybciej.



Moje tabele nie są aż tak duże. Dlaczego zatem mam sobie zawracać głowę jakąś normalizacją?



Ponieważ nawet w przypadku małych tabel można na tym zyskać.

Poza tym w miarę upływu czasu tabele stają się coraz większe. Jeśli od samego początku Twoje tabele będą znormalizowane, to w przyszłości, gdy zapytania zaczną być wykonywane zbyt wolno, nie będziesz musiał ich modyfikować.

Klowni nie są normalni

Czy pamiętasz tabelę z informacjami o publicznych wystąpieniach kloonów? Śledzenie poczynań kloonów stało się ogólnokrajowym szaleństwem i nasza stara tabela może nie sprostać zwiększym wymaganiom, ponieważ kolumny **wygląd** i **aktywności** zawierają *tak* wiele danych. Na nasze potrzeby ta tabela nie jest atomowa.

Wyszukiwanie danych w tych dwóch kolumnach jest naprawdę trudne, gdyż zawierają one tak wiele informacji!

kloon_i_informacje

imie	ostatnio_widziano	wygląd	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpuś	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółta koszula, workowate czerwone spodnie	trąbka, parasolka
Pan Hobo	Cyrk Koloseum	M, cygaro, czarne włosy, niewielki kapelusz	skrzypce
Klarabela	Dom opieki „Wesoła Wdówka”	K, różowe włosy, ogromny kwiat, niebieska sukienka	krzyk, taniec
Skuter	Szpital Miejski	M, niebieskie włosy, czerwony garnitur, ogromny nos	balony
Zippo	Centrum Handlowe Skarbiec	K, pomarańczowy garnitur, workowate spodnie	taniec
Balbina	Auta Edwarda	K, cała na pomarańczowo i w cekinach	utrzymywanie równowagi, mały samochodzik
Gonzo		M, w przebraniu kobiety, kostium w plamki	śpiew, taniec
Pan Smyk	Zabawex	M, zielono-fioletowy kostium, szpiczasty nos	wchodzenie do malutkiego samochodu

Zaostrz ołówek



Spróbujmy tak zmodyfikować tabelę **kloon_i_informacje**, by stała się ona tabelą atomową. Spróbuj zaproponować inną strukturę tabeli, zakładając przy tym, że chcemy wyszukiwać w niej informacje na podstawie kolumn **wygląd**, **aktywnosci** oraz **ostatnio_widziano**.

W połowie drogi do 1NF

Pamiętasz zapewne, że umieszczenie w tabeli danych atomowych to jedynie połowa drogi do znormalizowania tabeli. Kiedy tabela będzie całkowicie znormalizowana, przyjmie ona PIERWSZĄ POSTAĆ NORMALNĄ (ang. *first normal form*, w skrócie 1NF).

Aby tabela miała pierwszą postać normalną, musi spełniać dwa warunki:

Już wiem, jak
zaspakoić ten
wymóg.

→
**Każdy wiersz danych musi
zawierać wartości atomowe**

→
Aby całkowicie
znormalizować tabelę,
do każdego z jej
rekordów musimy
dodać klucz główny.

**Każdy wiersz danych musi
mieć unikalny identyfikator,
nazywany kluczem głównym
(ang. *primary key*).**



Reguły KLUCZA GŁÓWNEGO

Kolumnę, która ma pełnić rolę klucza głównego tabeli, należy utworzyć specjalnie w tym celu od razu podczas projektowania tabeli. Na kilku kolejnych stronach utworzymy tabelę i wskażemy kolumnę, która będzie jej kluczem głównym. Jednak zanim to zrobimy, przyjrzymy się, czym ten klucz główny jest.



Klucz główny służy do zapewniania unikalności wszystkich rekordów tabeli.

A to oznacza, że wartości w kolumnie klucza głównego nie mogą się powtarzać. Przeanalizujmy przykład tabeli przedstawionej poniżej. Jak myślisz, czy któraś z jej kolumn nadawałaby się na klucz główny?

PESEL	nazwisko	numer	telefon
-------	----------	-------	---------



Numery PESEL mają unikalne wartości i są przypisywane konkretnym osobom. Może zatem ta kolumna nadawałaby się na klucz główny?



W tych trzech kolumnach wartości mogą się powtarzać; na przykład z dużą dozą prawdopodobieństwa możemy zatolić, że w tabeli znajdzie się więcej niż jedna osoba o imieniu Jan; podobnie może się zdarzyć, że kilka osób będzie mieszkać razem i używać tego samego numeru telefonu. Dlatego te trzy kolumny raczej nie będą się nadawały na pełnienie roli klucza głównego tabeli.



Kluczem głównym tabeli nazywamy kolumnę, która sprawia, że każdy wiersz tabeli jest unikalny.



Zadbaj o swoje rekordy, używając kolumny PESEL jako klucza głównego tabeli.

Przy coraz częstszych kradzieżach tożsamości coraz mniej osób zgodzi się na podanie swojego numeru PESEL, co jest zresztą całkowicie zrozumiałe.

Jest to zbyt ważna informacja, by ryzykować jej kradzież. Czy możesz z całkowitą pewnością stwierdzić, że Twoja baza jest bezpieczna? Jeśli nie, to wszystkie te numery PESEL mogą zostać skradzione wraz z tożsamością ich właścicielami.



W kolumnie klucza głównego nie mogą się pojawiać wartości NULL.

Gdyby w kolumnie klucza głównego można było zapisywać wartości NULL, to rekordy nie byłyby unikalne, gdyż wartość NULL mogłaby się pojawić w kilku z nich.



Wartość kolumny klucza głównego musi zostać określona w momencie dodawania rekordu do tabeli.

Jeśli wartość kolumny klucza głównego nie zostanie określona w momencie dodawania rekordu, to ryzykujemy, że pojawi się w niej wartość NULL, co może doprowadzić do powtórzenia się tego samego rekordu w tabeli i naruszenia zasad pierwszej postaci normalnej.



Klucz główny musi być krótki.

Klucz główny musi zawierać tylko te informacje, które są niezbędne dla zapewnienia jego unikalności, i nic więcej.



Wartości w kolumnie klucza głównego nie mogą się zmieniać.

Gdyby można było zmieniać wartości zapisane w kolumnie klucza głównego, to przypadkowo można by podać wartość, która została już użyta. Pamiętaj, wszystkie wartości w kolumnie klucza głównego zawsze muszą być unikalne.



WYSIL SZARE KOMÓRKI

Czy na podstawie tych wszystkich informacji możesz podać przykład dobrego klucza głównego?

Przejrzyj przykładowe tabele, które przedstawiliśmy we wcześniejszej części książki.
Czy w którejś z nich jest kolumna zawierająca naprawdę unikalne wartości?



Chwila, a zatem, skoro nie mogę użyć numeru PESEL jako klucza głównego, a jednocześnie wartości w tej kolumnie wciąż muszą być unikalne, krótkie i niezmienne, to co to może być?

Najlepszym kluczem głównym może być nowy klucz główny.

W przypadku tworzenia kluczy głównych najlepszym i najprostszym rozwiązaniem jest utworzenie kolumny, która będzie zawierać unikalne liczby. Wyobraź sobie tabelę zawierającą informacje o osobach, w której umieścisz dodatkową kolumnę liczbową. W przykładzie przedstawionym poniżej jest to kolumna id.

Gdyby nie kolumna id, to dwa rekordy Janka Kowalskiego byłyby identyczne. Jednak ponieważ istnieje klucz główny, oba te rekordy reprezentują różne osoby. A zatem klucz główny zapewnia unikalność tych dwóch rekordów. Poniższa tabela jest w pierwszej postaci normalnej.

id	nazwisko	imie	pseudonim
1	Kowalski	Janek	Pierun
2	Paciorek	Krystyna	Krycha
3	Kowalski	Janek	Pierun
4	Ślusarczyk	Anna	Ania
5	Paprocki	Krzysztof	Krzych

← Rekord Janka Kowalskiego.

← Ten rekord także zawiera dane Janka Kowalskiego, jednak wartość w kolumnie klucza głównego pokazuje, że jest to unikalny rekord i dotyczy całkowicie innej osoby niż pierwszy Janek Kowalski.



Dla maniaków

W świecie SQL-a trwa niekończąca się debata na temat tego, czy należy używać **syntetycznych** kluczy głównych, czyli specjalnie utworzonych (takich jak nasza kolumna id), czy też kluczy **naturalnych** — tworzonych na podstawie informacji, które już są zapisane w tabeli (takich jak numer nadwozia lub PESEL). Nie preferujemy żadnego z tych rozwiązań, a zagadnieniami związanymi z kluczami głównymi zajmiemy się bardziej szczegółowo **w rozdziale 7**.

Nie istniejąca
grupie pytania

P: Cały czas mówicie o „pierwszej” postaci normalnej. Czy to oznacza, że istnieje też druga postać normalna? A może i trzecia?

O: Owszem, faktycznie istnieją także druga i trzecia postać normalna, a każda z nich narzuca coraz to bardziej wymagające warunki na strukturę i zawartość tabel. Zajmiemy się nimi dokładniej w rozdziale 7.

P: No dobrze, zatem zmieniliśmy nasze tabele w taki sposób, by zawierały dane atomowe. Czy któraś z nich znajduje się już w pierwszej postaci normalnej?

O: Nie. Jak na razie żadna ze stworzonych przez nas tabel nie zawiera ani klucza głównego, ani wartości unikalnych.

P: Wydaje mi się, że kolumna „komentarze” w tabeli o pączkach nie zawiera wartości atomowych. Chodzi mi o to, że nie ma żadnego prostego sposobu przeszukiwania jej zawartości.

O: Masz całkowitą rację. Ta kolumna raczej nie jest szczególnie atomowa. Choć z drugiej strony, projekt naszej tabeli nie narzuca takiej konieczności. Gdybyśmy jednak zdecydowali się na ograniczenie zawartości opinii do ściśle określonej grupy słów, to to pole mogłoby być atomowe. Jednak w takim przypadku zapisywane w bazie opinie nie byłyby spontaniczne.

Dążenie do pierwszej postaci NORMALNEJ

Nadszedł czas, by nieco się cofnąć i zająć się normalizacją naszych tabel. Musimy postarać się, by zapisane w nich informacje były atomowe, i dodać do nich klucze główne. Klucz główny tabeli jest zazwyczaj określany podczas tworzenia tabeli, czyli pisania polecenia CREATE TABLE.



**WYSIL
SZARE KOMÓRKI**

Czy pamiętasz, jak można dodawać kolumny do istniejących tabel?

Poprawianie tabeli Grześka

Poprawianie tabeli Grześka

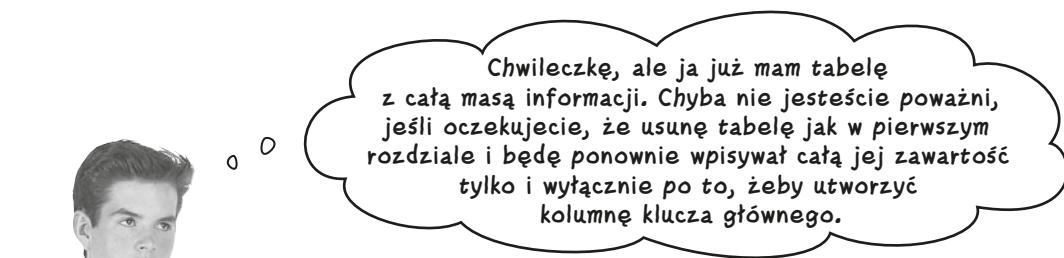
W jaki sposób — bazując na zdobytych już informacjach — powinieneś poprawić tabelę Grześka:

Poprawa tabeli Grześka — krok 1.: Pobierz wszystkie informacje zapisane w tabeli, używając w tym celu polecenia SELECT, i zapisz je gdzieś.

Poprawa tabeli Grześka — krok 2.: Utwórz nową, znormalizowaną tabelę.

Poprawa tabeli Grześka — krok 3.: Zapisz wszystkie stare dane w nowej tabeli, modyfikując dane w każdym z wierszy w taki sposób, by odpowiadały one nowej strukturze tabeli.

Teraz możesz usunąć starą wersję tabeli polecienniem DROP.



Mamy pewność, że tabela Grześka nie jest doskonała.

Tabela Grześka nie jest atomowa i nie ma klucza głównego. Jednak, na szczęście dla Grześka, **nie jesteśmy skazani** na starą tabelę ani **nie musimy kopiować i ponownie wpisywać jej zawartości**.

Możemy dodać do tabeli Grześka klucz główny i zapewnić atomowość danych przy użyciu jednego, nowego polecenia SQL. Jednak zanim to zrobimy, cofnijmy się nieco w przeszłość...



Oryginalna postać polecenia CREATE TABLE

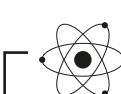
Grzesiek musi dodać do swojej tabeli klucz główny; oprócz tego zdaje sobie sprawę, że jest kilka rzeczy, które może zrobić, by poprawić atomowość informacji zapisywanych w tabeli. Zanim przekonamy się, w jaki sposób można poprawić istniejącą tabelę, przypomnij sobie, jak ją utworzyliśmy.

Oto postać polecenia CREATE TABLE, którego użyliśmy do utworzenia tabeli Grześka dawno temu, w rozdziale 1.:

```
CREATE TABLE moje_kontakty
(
    nazwisko VARCHAR(30),
    imie VARCHAR(20),
    email VARCHAR(50),
    plec CHAR(1),
    data_urodzenia DATE,
    zawod VARCHAR(50),
    lokalizacja VARCHAR(50),
    stan VARCHAR(20),
    zainteresowania VARCHAR(100),
    szuka VARCHAR(100)
);
```

*W tej tabeli
nie ma
kolumny
klucza
głównego.*

*Czy tworząc tabelę,
moglibyśmy poprawić
atomowość tych kolumn?*



WYSIL
SZARE KOMÓRKI

A co zrobić, gdybyśmy nie mieli nigdzie zanotowanego oryginalnego polecenia CREATE TABLE użytego do utworzenia tabeli? Czy istnieje jakiś sposób dotarcia do jego kodu?

Wyświetlanie kodu polecenia CREATE

tablę
Pokażcie mi moją ~~kacę~~

A co by się stało, gdybyś zastosował polecenie DESCRIBE moje_kontakty, by podejrzeć kod użyty do utworzenia tej tabeli? Otóż gdybyś to zrobił, w oknie konsoli ujrzałbyś następujące wyniki:

The screenshot shows a Windows command-line window titled 'C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe'. The window displays the output of a MySQL DESCRIBE command. The table structure for the 'moje_kontakty' table is shown with the following columns:

Field	Type	Null	Key	Default	Extra
nazwisko	varchar(30)	YES		NULL	
imie	varchar(20)	YES		NULL	
email	varchar(50)	YES		NULL	
plec	char(1)	YES		NULL	
data_urodzenia	date	YES		NULL	
zawod	varchar(50)	YES		NULL	
lokalizacja	varchar(50)	YES		NULL	
stan	varchar(20)	YES		NULL	
zainteresowania	varchar(100)	YES		NULL	
szuka	varchar(100)	YES		NULL	

Below the table, the message '10 rows in set <0.01 sec>' is displayed, followed by the mysql> prompt.

Ale nam zależy na dotarciu do kodu polecenia CREATE, a nie do informacji o polach tabeli. W ten sposób będziemy bowiem mogli uniknąć konieczności ponownego wpisywania polecenia i określić, co powinniśmy zrobić już podczas tworzenia tabeli.

Polecenie SQL SHOW CREATE TABLE wyświetli kod polecenia CREATE TABLE, które pozwoli odtworzyć tabelę, oczywiście bez żadnych danych. Dzięki niemu w dowolnej chwili możemy się przekonać, w jaki sposób należy odtworzyć daną tabelę. Spróbuj wykonać następujące polecenie:

SHOW CREATE TABLE moje_kontakty;

Polecenie oszczędzające czas

Rzuć okiem na kod polecenia, którego użyliśmy do utworzenia tabeli, przedstawiony na stronie 217. Następnie porównaj go z zamieszczonym poniżej kodem zwróconym przez polecenie SHOW CREATE TABLE moje_kontakty. Nie są one identyczne, jednak gdybyś wykonał poniższe polecenie CREATE TABLE, to uzyskane wyniki byłyby takie same, jak w przypadku użycia oryginalnego polecenia. Nie musisz usuwać znaków lewego apostrofu ani ustawień dotyczących wartości domyślnych, jeśli jednak to zrobisz, to polecenie będzie bardziej przejrzyste i schludne.

Pomiędzy takimi znakami, określonymi jako lewy apostrof, są zapisywane nazwy kolumn oraz nazwa tabeli. Znaki lewego apostrofu są używane w wynikach generowanych przez polecenie SHOW CREATE TABLE.

```
CREATE TABLE `moje_kontakty` (
  `nazwisko` varchar(30) default NULL,
  `imie` varchar(20) default NULL,
  `email` varchar(50) default NULL,
  `plec` char(1) default NULL,
  `data_urodzenia` date default NULL,
  `zawód` varchar(50) default NULL '',
  `lokalizacja` varchar(50) default NULL '',
  `stan` varchar(20) default NULL '',
  `zainteresowania` varchar(100) default NULL '',
  `szuka` varchar(100) default NULL ''
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Jeśli jawnie nie zażdamy inaczej, to system zarządzania bazą danych przyjmuje, że wartości wszystkich kolumn przyjmują domyślnie wartość NULL.

Podczas tworzenia tabeli warto określić, czy w jej poszczególnych kolumnach mogą być zapisywane wartości NULL, czy też nie.

Nie musisz zwracać uwagi na ostatni wiersz kodu, umieszczony za nawiasem zamykającym. Określa on sposób przechowywania danych oraz używany zbiór znaków. Jak na razie w zupełności wystarczą nam ustawienia domyślne.

Aby wykonać to polecenie, będziesz musiał zmienić nazwę tabeli; no chyba że wcześniej usuniesz oryginalną tabelę.

Choć można by poprawić przejrzystość kodu (usuwając jego ostatni wiersz oraz wszystkie znaki lewego apostrofu), to jednak by utworzyć tabelę, nie trzeba wprowadzać w nim żadnych modyfikacji — wystarczy go skopiować i wkleić w przedstawionej postaci.

Tworzenie tabeli z KLUCZEM GŁÓWNYM

Poniżej przedstawiliśmy kod, który zwróciło polecenie SHOW CREATE TABLE moje_kontakty. Usunęliśmy z niego znaki lewego apostrofu oraz fragment ostatniego wiersza. Na samym początku listy kolumn dodaliśmy kolumnę `id_kontaktu`, której wartości, dzięki zastosowaniu wyrażenia NOT NULL, nie będą mogły przybierać wartości NULL. Z kolei na samym końcu listy dodaliśmy wyrażenie PRIMARY KEY, które określa, że kluczem głównym tabeli ma być nasza nowa kolumna `id_kontaktu`.

Utworzyliśmy nową kolumnę o nazwie `id_kontaktu`, w której będą zapisywane wartości liczbowe. Kolumna ta będzie pełnić funkcję klucza głównego naszej tabeli. Każda wartość w tej kolumnie będzie unikalna, dzięki czemu cała tabela stanie się tabelą atomową.

```
CREATE TABLE moje_kontakty  
(  
    id_kontaktu INT NOT NULL,  
    nazwisko varchar(30) default NULL,  
    imie varchar(20) default NULL,  
    email varchar(50) default NULL,  
    plec char(1) default NULL,  
    data_urodzenia default NULL,  
    zawod varchar(50) default NULL,  
    lokalizacja varchar(50) default NULL,  
    stan varchar(20) default NULL,  
    zainteresowania varchar(100) default NULL,  
    szuka varchar(100) default NULL,  
    PRIMARY KEY (id_kontaktu)  
)
```

Pamiętaj, że zawartość kolumny klucza głównego musi być różna od NULL — NOT NULL! Gdyby w kolumnie klucza głównego pojawiła się wartość NULL lub gdyby można było pominać wartość zapisaną w tej kolumnie, to nie moglibyśmy zagwarantować, że wszystkie wiersze tabeli będą identyfikowane w jednoznaczny, unikalny sposób.

To właśnie w tym miejscu określamy klucz główny tabeli. Składnia używana do tego celu jest naprawdę prosta — wystarczy zapisać słowa PRIMARY KEY, a następnie podać w nawiasach nazwę kolumny klucza głównego. W naszym przypadku kluczem głównym będzie kolumna `id_kontaktu`.

.....
Nie istniejąca
grupie pytania

P: Zatem mówicie, że KLUCZ GŁÓWNY nie może przybierać wartości NULL. Co jeszcze zapewnia jego unikalność?

O: Najprościej rzec ujmując — jedynie Ty. Zapisując w tabeli każdy nowy wiersz, w kolumnie `id_kontaktu` będziesz umieszczał nową, unikalną wartość. Na przykład w pierwszym wydawanym poleceniu `INSERT` w kolumnie `id_kontaktu` zapiszesz wartość 1, w kolejnym wierszu wartość 2 i tak dalej.

P: Wyobrażam sobie, że takie określanie unikalnych wartości KLUCZA GŁÓWNEGO w każdym nowym rekordzie musi być naprawdę kłopotliwe. Czy nie ma żadnego prostszego rozwiązania?

O: Owszem, są prostsze rozwiązania; i to nawet dwa. Pierwszym z nich jest zastosowanie jako klucza głównego jakieś istniejącej kolumny danych, o której wiemy, że ma unikalne wartości. Wspominaliśmy jednak, że takie rozwiązanie może przysparzać pewnych problemów (na przykład w przypadku stosowania numerów PESEL użytkownicy mogą się obawiać przejęcia swoich poufnych informacji przez niepożądaną osobę).

Prostym rozwiązaniem jest utworzenie zupełnie nowej kolumny, przeznaczonej do przechowywania unikalnych wartości klucza głównego, takiej jak kolumna `id_kontaktu` na poprzedniej stronie. W takim przypadku można nakazać systemowi obsługi baz danych, by automatycznie wypełniał wartość takiej kolumny w każdym rekordzie dodawanym do tabeli. Do tego celu służy specjalne słowo kluczowe. Zajrzyj na następną stronę — tam znajdziesz więcej informacji na ten temat.

P: Czy polecenia SHOW mogę używać także do wyświetlania innych informacji, czy tylko do odtworzenia polecenia CREATE TABLE?

O: Polecenia SHOW można także używać do wyświetlania informacji o konkretnych kolumnach:

```
SHOW COLUMNS FROM nazwa_tabeli;
```

To polecenie wyświetli nazwy wszystkich kolumn podanej tabeli wraz ze wszystkimi informacjami na ich temat.

```
SHOW CREATE DATABASE nazwa_bazy_danych;
```

Powysze polecenie, podobnie jak polecenie `SHOW CREATE TABLE nazwa_tabeli`, zwraca kod polecenia `CREATE`, z tym że w tym przypadku będzie to polecenie służące do utworzenia całej bazy danych.

```
SHOW INDEX FROM nazwa_tabeli;
```

Z kolei to polecenie wyświetli wszystkie kolumny, dla których zdefiniowano indeksy, wraz z informacjami na temat typów tych indeksów. Jak na razie jedynym typem indeksów, z jakim się zetknęliśmy, był klucz główny, jednak w dalszej części książki to polecenie stanie się znacznie bardziej przydatne.

Istnieje jeszcze jedna wersja polecenia `SHOW`, i to wersja BARDZO użyteczna:

```
SHOW WARNINGS;
```

Jeśli w oknie konsoli pojawi się informacja, że podczas wykonywania polecenia SQL pojawiły się jakieś ostrzeżenia, to użyj tego polecenia, by wyświetlić ich treść.

Dostępne są jeszcze inne wersje polecenia `SHOW`, jednak te przedstawione powyżej dotyczą zagadnień, którymi się już zajmowaliśmy.

P: No a co ze znakami lewego apostrofu, które zostały użyte w kodzie zwróconym przez polecenie SHOW CREATE TABLE? Czy jesteście pewni, że nie są potrzebne?

O: Są one używane dlatego, że w niektórych przypadkach system zarządzania bazami danych może nie być w stanie określić, że nazwa ma być nazwą kolumny. Jeśli jednak zapiszesz nazwy kolumn pomiędzy znakami lewego apostrofu, to istnieje możliwość stosowania słów kluczowych języka SQL jako nazw kolumn; choć absolutnie nie polecamy takiego rozwiązania.

Na przykład wyobraźmy sobie, że z jakichś niewytyumaczalnych powodów chciałibyśmy nadać kolumnie nazwę `select`. Poniższa deklaracja kolumny jest jednak nieprawidłowa:

```
select varchar(50)
```

Zamiast niej musiałibyśmy użyć następującej deklaracji:

```
`select` varchar(50)
```

P: A niby czemu stosowanie słów kluczowych języka SQL jako nazw kolumn jest złym rozwiązaniem?

O: Oczywiście wolno to robić, jednak nie jest to dobry pomysł. Pomyśl tylko, jak mylące i trudne do zrozumienia mogą się stać zapytania oraz jak męczące będzie wpisywanie tych wszystkich znaków lewego apostrofu. Poza tym `select` nie jest dobrą nazwą dla kolumny, gdyż nie mówi nic o danych, jakie są w niej zapisywane.

1, 2, 3... automatycznie inkrementowane

Dodanie do naszej kolumny `id_kontaktu` słowa kluczowego `AUTO_INCREMENT` nakaże, by w pierwszym wierszu tabeli przyjęła ona wartość 1, a w każdym kolejnym była powiększana o 1.

`CREATE TABLE moje_kontakty`

```
(  
    id_kontaktu INT NOT NULL AUTO_INCREMENT,  
    nazwisko varchar(30) default NULL,  
    imie varchar(20) default NULL,  
    email varchar(50) default NULL,  
    plec char(1) default NULL,  
    data_urodzenia default NULL,  
    zawod varchar(50) default NULL,  
    lokalizacja varchar(50) default NULL,  
    stan varchar(20) default NULL,  
    zainteresowania varchar(100) default NULL,  
    szuka varchar(100) default NULL,  
    PRIMARY KEY (id_kontaktu)  
)
```

To jest to! W większości systemów zarządzania bazami danych wystarczy do deklaracji kolumny dodać słowo kluczowe `AUTO_INCREMENT`. (Użytkownicy MS SQL pamiętajcie: w Waszym przypadku będzie to słowo kluczowe `INDEX`, po którym należy podać wartość początkową oraz liczbę, o której będą powiększane wartości w poszczególnych kolumnach. Bardziej szczegółowe informacje na ten temat można znaleźć w dokumentacji serwera MS SQL).

To słowo kluczowe robi dokładnie to, czego można by się spodziewać: sprawia, że w pierwszym wierszu tabeli, w danej kolumnie zostanie automatycznie zapisana wartość 1, która w kolejnych wierszach będzie automatycznie powiększana o 1.



No dobrze, to chyba faktycznie jest proste. Ale w jaki sposób zapisać w tabeli wiersz, w którym wartość takiej kolumny będzie już określona? Poza tym, czy można zmienić wartość zapisaną w takiej kolumnie?

A jak myślisz, co się w takich przypadkach stanie?

Najlepiej będzie, jeśli sam spróbujesz wykonać takie operacje, i przekonasz się, jakie będą ich skutki.



Ćwiczenie

- 1.** Poniżej, w pustych wierszach, zapisz kod polecenia CREATE TABLE, które pozwoli utworzyć tabelę służącą do przechowywania imion i nazwisk osób. Powinna ona zawierać kolumnę klucza głównego, której wartości będą automatycznie inkrementowane, oraz dwie kolumny atomowe.

.....
.....
.....
.....

- 2.** Wykonaj powyższe polecenie w oknie konsoli bazy danych lub innym programie do zarządzania bazą.
- 3.** Spróbuj wykonać każde z poniższych poleceń INSERT. Zakreśl te, które działają prawidłowo.

```
INSERT INTO twoja_tabela (id, imie, nazwisko)
VALUES (NULL, 'Marysia', 'Krawczyk');
```

```
INSERT INTO twoja_tabela (id, imie, nazwisko)
VALUES (1, 'Janka', 'Krawczyk');
```

```
INSERT INTO twoja_tabela
VALUES ('', 'Bartek', 'Krawczyk');
```

```
INSERT INTO twoja_tabela (imie, nazwisko)
VALUES ('Cecylia', 'Krawczyk');
```

```
INSERT INTO twoja_tabela (id, imie, nazwisko)
VALUES (99, 'Piotr', 'Krawczyk');
```

- 4.** Czy udało się zapisać w tabeli informacje o wszystkich członkach rodziny Krawczyków? Zapisz poniżej zawartość tabeli po wykonaniu wszystkich poniższych polecień.

twoja_tabela

<i>id</i>	<i>imie</i>	<i>nazwisko</i>



- 1.** Poniżej, w pustych wierszach, zapisz kod polecenia CREATE TABLE, które pozwoli utworzyć tabelę służącą do przechowywania imion i nazwisk osób. Powinna ona zawierać kolumnę klucza głównego, której wartości będą automatycznie inkrementowane, oraz dwie kolumny atomowe.

```
CREATE TABLE twoja_tabela
(
    id INT NOT NULL AUTO_INCREMENT,
    imie VARCHAR(20),
    nazwisko VARCHAR(30),
    PRIMARY KEY (id)
);
```

- 2.** Wykonaj powyższe polecenie w oknie konsoli bazy danych lub innym programie do zarządzania bazą.
- 3.** Spróbuj wykonać każde z poniższych poleceń INSERT. Zakreśl te, które działają prawidłowo.

INSERT INTO twoja_tabela (id, imie, nazwisko)
VALUES (NULL, 'Marysia', 'Krawczyk');

INSERT INTO twoja_tabela (id, imie, nazwisko)
VALUES (1, 'Janka', 'Krawczyk');

INSERT INTO twoja_tabela
VALUES ('', 'Bartek', 'Krawczyk');

INSERT INTO twoja_tabela (imie, nazwisko)
VALUES ('Cecylia', 'Krawczyk');

INSERT INTO twoja_tabela (id, imie, nazwisko)
VALUES (99, 'Piotr', 'Krawczyk');

- 4.** Czy udało się zapisać w tabeli informacje o wszystkich członkach rodziny Krawczyków? Zapisz poniżej zawartość tabeli po wykonaniu wszystkich poniższych poleceń.

To ostatnie polecenie „zadziała”, jednak zmodyfikuje wartość kolumny AUTO_INCREMENT.

id	imie	nazwisko
1	Marysia	Krawczyk
2	Bartek	Krawczyk
3	Cecylia	Krawczyk
99	Piotr	d Krawczyk

Wygląda na to, że zgubiłyśmy Jankę; zapewne dlatego, że próbowałyśmy jej przypisać indeks, który został już użyty w rekordzie Marysi. Marysia, Marysia, Marysia!

Nie istniejąca grupa pytań

P. Dlaczego pierwsze polecenie, to z wartością **NULL** w kolumnie **id**, zadziałało? Przecież ta kolumna została zadeklarowana jako **NOT NULL**.

O.: Faktycznie wygląda to tak, jakby to polecenie zadziałało, choć nie powinno. Okazuje się jednak, że w przypadku użycia `AUTO_INCREMENT` wartości `NULL` są ignorowane. Oczywiście, gdybyśmy nie umieścili w deklaracji kolumny słowa kluczowego `AUTO_INCREMENT`, to próba wykonania takiego polecenia spowodowałaby zgłoszenie błędu, a rekord nie zostałby zapisany. Spróbuj sam, a się przekonasz.

Słuchajcie, nie mogę powiedzieć, żebyście mnie przekonali. Jak widać, postugując się poleceniem `SHOW CREATE TABLE`, mogę uzyskać i wykonać kod, który utworzy tabelę; jednak cały czas mam wrażenie, że będę musiał usunąć swoją oryginalną tabelę i ponownie, od nowa, wpisać całą jej zawartość tylko i wyłącznie po to, by dodać kolumnę klucza głównego.



Nie będziesz musiał wpisywać wszystkiego od nowa. Wystarczy, że użyjesz polecenia `ALTER`.

Nie trzeba kopiować danych zapisanych w istniejącej tabeli, a następnie jej usuwać i tworzyć od nowa. Można bowiem modyfikować strukturę istniejących tabel. Jednak żeby to zrobić, musimy sięgnąć po polecenie `ALTER` i kilka używanych przez niego słów kluczowych, które opisaliśmy w rozdziale 5.

Dodawanie KLUCZA GŁÓWNEGO do istniejącej tabeli

Poniżej przedstawiliśmy kod, który pozwoli dodać do tabeli `moje_kontakty` kolumnę klucza głównego, której wartości będą automatycznie inkrementowane. (Polecenie jest naprawdę długie, więc będziesz musiał obrócić książkę, żeby je przeczytać).

`FIRST` to słowo, które nakazuje, by nowa kolumna została dodana jako pierwsza kolumna tabeli. Choć nie ma to większego znaczenia, to jednak umieszczenie klucza głównego jako pierwszej kolumny tabeli jest dobrym pomysłem.

A to kod, który doda do tabeli nową kolumnę. Czy nie wygląda znajomo?

```
ALTER TABLE moje_kontakty  
ADD COLUMN id_kontaktu INT NOT NULL AUTO_INCREMENT FIRST,  
ADD PRIMARY KEY (id_kontaktu);
```

Oto nasze nowe polecenie SQL – `ALTER`.

Działanie stów `ADD COLUMN` jest zgodne z ich znaczeniem w języku angielskim – dodaje do tabeli nową kolumnę. W naszym przypadku kolumna ta będzie nosić nazwę `id_kontaktu`.
Powinieneś już rozpoznać ten kod – wskazuje on kolumnę, która będzie pełnić rolę klucza głównego.

WYSIL SZARE KOMÓRKI

Czy sądzisz, że powyzsze polecenie doda wartości do kolumny `id_kontaktu` w już istniejących rekordach tabeli czy tylko w rekordach, które dopiero będą w niej zapisywane? Jak możesz to sprawdzić?

Modyfikacja tabeli i dodanie KLUCZA GŁÓWNEGO

Sprawdź wyniki działania polecenia ALTER przedstawionego na poprzedniej stronie. Wybierz bazę danych `lista_grzesia` poleciem USE i wykonaj polecenie:

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> ALTER TABLE moje_kontakty
-> ADD COLUMN id_kontaktu INT NOT NULL AUTO_INCREMENT FIRST,
-> ADD PRIMARY KEY (id_kontaktu);
Query OK, 37 rows affected (0.22 sec)
Records: 37 Duplicates: 0 Warnings: 0
mysql>
```

Ten komunikat informuje nas, że polecenie dodało nową kolumnę do 37 rekordów, które już były zapisane w bazie. Ty zapewne nie będziesz ich mieć aż tak wielu.

Zajefajnie...
Teraz już mam kolumnę klucza głównego wraz z odpowiednimi wartościami. Czy mogę użyć polecenia ALTER także do dodania kolumny do zapisywania numeru telefonu?

Aby sprawdzić, co się zmieniło w tabeli Grześka, wykonaj polecenie `SELECT * FROM moje_kontakty;`



Kolumna `id_kontaktu` została dodana jako pierwsza — przed wszystkimi pozostałymi kolumnami.

Ponieważ użyliśmy słowa kluczowego `AUTO_INCREMENT`, zawartość tej kolumny była uzupełniana podczas modyfikowania poszczególnych, istniejących już wierszy tabeli.

	id_kontaktu	nazwisko	imie	email
	1	Kubaski	Andrzej	Kubaski_Andrzej@pizza-nzk.com.pl
	2	Nowak	Joanna	nowak@op.pl
	3	Symbolowicz	Anna	anna_tymonowicz@pizza-nzk.com.pl
	4	Oczkiewicz	Adam	oczan@zygzyg.eu
	5	Karczynska	Marta	karmar@interia.pl
	6	Zefirek	Monika	moniaze@pizza-nzk.com.pl
	7	Harmonia	Ania	a_harmonia@mojamusa.pl
	8	Mikułski	Janek	mikunek@wp.pl
	9	Tkacz	Martyna	mptkacz@o2.eu
	10	Maliniak	Konrad	kon_malinka@op.pl
	11	Piekarska	Kasia	misia_kasia@o2.eu
	12	Swalski	Małgorzata	swaler_m@op.pl

Kiedy w przyszłości będziemy dodawali nowy rekord, to w jego kolumnie `id_kontaktu` zostanie zapisana wartość o jeden większa od największej wartości zapisanej w danej tabeli w kolumnie `id_kontaktu`. A zatem, jeśli w ostatnim dodawanym rekordzie w kolumnie `kontakt_id` znajduje się wartość 23, to w kolejnym rekordzie znajdzie się wartość 24.

Pamiętaj, to nie jest koniec tabeli Grześka; tak naprawdę znajduje się w niej jeszcze wiele innych kontaktów.

Czy Grześkowi uda się dodać kolumnę do zapisywania numeru telefonu? Zajrzyj do rozdziału 5., żeby sprawdzić, czy to możliwe.



Przybornik SQL

A zatem opanowałeś materiał z czwartego rozdziału książki. Przyjrzyj się jeszcze raz wszystkim nowym narzędziom, jakie dodałeś do swojego SQL-owego przybornika. Kompletną listę porad znajdziesz w dodatku C, zamieszczonym na końcu książki.

DANE ATOMOWE – REGUŁA 1.

Jeśli dane mają być atomowe, to w tej samej kolumnie nie może się znajdować kilka elementów danych tego samego typu.

DANE ATOMOWE – REGUŁA 2.

Aby dane w tabeli były atomowe, nie może się w niej znajdować kilka kolumn zawierających informacje tego samego typu.

KLUCZ GŁÓWNY

Kolumna lub zbiór kolumn, które w unikalny sposób identyfikują wiersze tabeli.

AUTO_INCREMENT

Zastosowanie tego stowa kluczowego w deklaracji tabeli sprawi, że w momencie wykonywania polecień INSERT w danej kolumnie będą się automatycznie i magicznie pojawiały unikalne liczby całkowite.

DANE ATOMOWE

Dane w tabeli są atomowe, jeśli zostały już podzielone na najmniejsze elementy, które są Ci potrzebne.

SHOW CREATE TABLE

Skorzystaj z tego polecenia, by poznać składnię polecenia SQL, które pozwoli Ci odtworzyć istniejącą tabelę.

PIERWSZA POSTAĆ NORMALNA (1NF)

Każdy wiersz danych musi zawierać dane atomowe oraz unikalny identyfikator.

Zaostrz ołówek



Rozwiążanie

Spróbujmy tak zmodyfikować tabelę `kłowni_informacje`, by stała się ona tabelą atomową.

Spróbuj zaproponować inną strukturę tabeli, zakładając przy tym, że chcemy wyszukiwać w niej informacje na podstawie kolumn `wyglad`, `aktywnosci` oraz `ostatnio_widziano`.

W tym przypadku nie ma jednego, właściwego rozwiązania.

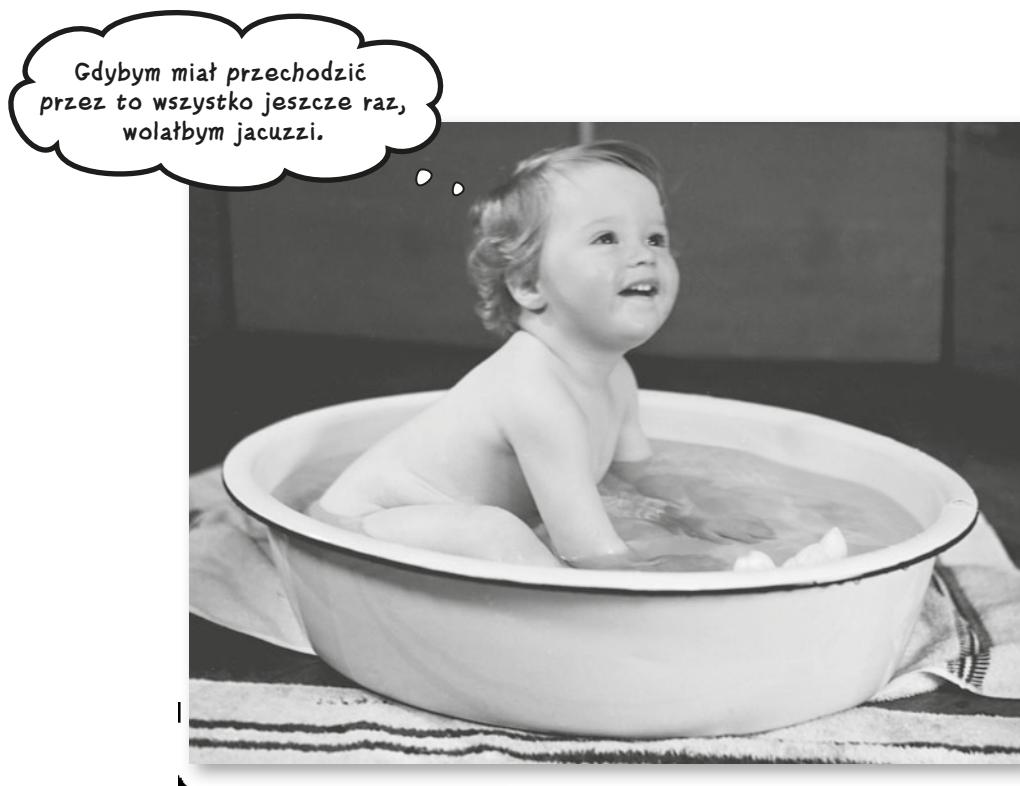
Najlepszą rzeczą, jaką możesz zrobić, jest wydzielenie takich informacji jak płeć, kolor koszuli, kolor spodni, typ kapelusza, instrument muzyczny, środek lokomocji, balony (pole logiczne — tak lub nie), śpiew (także pole logiczne z wartościami tak lub nie), taniec (także pole logiczne).

Aby zapewnić atomowość tabeli, musisz wydzielić te wszystkie aktywności i umieścić je w osobnych kolumnach, i podobnie zrobić z informacjami o wyglądzie.

Dodatkowe punkty możesz sobie przypisać, jeśli chciacie rozbić adres i umieścić te informacje w takich kolumnach jak adres, miejscowości i województwo.

5. Polecenie ALTER

★ Korygowanie przeszłości



Czy kiedykolwiek marzyłeś, by mieć możliwość naprawiania błędów popełnionych w przeszłości? Cóż, teraz masz taką możliwość. Dzięki poleceniu ALTER możesz zastosować wszystkie nowe wiadomości i doświadczenia, by zmodyfikować strukturę tabel utworzonych kilka dni, miesięcy lub nawet lat wcześniej. Ale to jeszcze nie wszystko — możesz to zrobić bez wprowadzania jakichkolwiek zmian w istniejących danych. Skoro tu dotarłeś, zapewne już wiesz, co naprawdę oznacza **normalizacja**, i będziesz mógł zastosować tę wiedzę do zapewnienia odpowiedniej struktury swym tabelom; tym starym i nowym.

Musimy wprowadzić kilka zmian

Grzesiek chce wprowadzić w swojej tabeli kilka zmian, jednak nie chce przy tym stracić żadnych posiadanych już danych.

	id_kontaktu	nazwisko	imie	email
	1	Kubaski	Andrzej	Kubaski_Andrzej@pizza-nzk.com.pl
	2	Nowak	Joanna	nowjn@op.pl
	3	Symonowicz	Anna	anna_tymonowicz@pizza-nzk.com.pl
	4	Oczkiewicz	Adam	oczam@zygzyg.eu
	5	Karczynska	Marta	karmar@interia.pl
	6	Zefirek	Monika	moniaze@pizza-nzk.com.pl
	7	Harmonia	Ania	a_harmonia@mojamuza.pl
	8	Mikulska	Janek	mikunek@wp.pl
	9	Tkacz	Martyna	mptkacz@o2.eu
	10	Maliniak	Konrad	kon_malinka@op.pl
	11	Piekarska	Kasia	munia_kasi@o2.eu
	12	Grzesiek	Grzesiek	grzesiek@o2.eu



Czyli jednak mogę dodać kolumnę na numer telefonu?

Owszem, możesz to zrobić bez najmniejszego problemu, używając polecenia ALTER TABLE.

W rzeczywistości uważamy, że powinieneś zająć się tym osobiście, ponieważ już poznaleś polecenie ALTER. Wykonaj następujące ćwiczenie, by napisać odpowiedni kod polecenia SQL.

Zaostrz ołówek



Przeanalizuj dokładnie kod polecenia ALTER TABLE, które zastosowaliśmy w rozdziale 4., by dodać do tabeli kolumnę klucza głównego. Sprawdź, czy będziesz wiedział, jak samodzielnie napisać polecenie ALTER, które doda do tabeli kolumnę telefon, w której będzie można zapisać do 10 cyfr. Zwróć uwagę, że w tym poleceniu nie będziesz musiał używać wszystkich poznanych słów kluczowych.

ALTER TABLE moje_kontakty

```
ADD COLUMN id_kontaktu INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (id_kontaktu);
```

Poniżej zapisz swoje polecenie ALTER:

.....
.....
.....

Polecenie ALTER pozwala nawet określić, w jakim miejscu ma zostać dodana nowa kolumna. Zobaczmy, czy będziesz w stanie domyślić się, jakiego słowa kluczowego należy użyć i gdzie je umieścić, by nowa kolumna została dodana bezpośrednio za kolumną imię.

Poniżej zapisz swoje nowe polecenie ALTER:

.....
.....
.....

Zaostrz ołówek



Przeanalizuj dokładnie kod polecenia ALTER TABLE, które zastosowaliśmy w rozdziale 4., by dodać do tabeli kolumnę klucza głównego. Sprawdź, czy będziesz wiedział, jak samodzielnie napisać polecenie ALTER dodające do tabeli kolumnę telefon, w której będzie można zapisać do 10 cyfr. Zwróć uwagę, że w tym poleceniu nie będziesz musiał używać wszystkich poznanych słów kluczowych.

ALTER TABLE moje_kontakty

ADD COLUMN id_kontaktu INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (id_kontaktu);

Słowa kluczowe, które poznaliśmy w poprzednim rozdziale, a których tu nie użyjemy, to: NOT NULL, AUTO_INCREMENT oraz FIRST.

Poniżej zapisz swoje polecenie ALTER:

Czytamy czas zajmujemy się modyfikowaniem tabeli moje_kontakty.

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10);

Zaktadamy, że numer telefonu nie może być dłuższy niż 10 znaków. Grzesiek nie przewiduje zapisywania w tabeli numerów telefonów z zagranicy.

Ten fragment polecenia precyzyjnie określa, w jaki sposób należy zmodyfikować tabelę.

Nowa kolumna, którą chcemy dodać do tabeli, nosi nazwę telefon.

Polecenie ALTER pozwala nawet określić, w jakim miejscu ma zostać dodana nowa kolumna. Zobaczmy, czy będziesz w stanie się domyślisć, jakiego słowa kluczowego należy użyć i gdzie je umieścić, by nowa kolumna została dodana bezpośrednio za kolumną imie.

Poniżej zapisz swoje nowe polecenie ALTER:

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)

AFTER imie;

Oto słowo kluczowe AFTER oraz nazwa kolumny tabeli, za którą ma zostać dodana nowa kolumna. To polecenie umieści nową kolumnę telefon bezpośrednio za kolumną imie.

Stosowanie słowa kluczowego AFTER jest opcjonalne. Jeśli go nie użyjesz, to nowa kolumna zostanie umieszczona na samym końcu tabeli.

Dowiedziałeś się już, że modyfikując tabelę, możesz używać słów kluczowych FIRST oraz AFTER nazwa_kolumny. Jednak to nie wszystko — możesz także używać słów: BEFORE nazwa_kolumny oraz LAST. Jak również SECOND, THIRD... chyba już rozumiesz ideę.



Za kulisami



Magnesiki ze słowami kluczowymi SQL

Skorzystaj z przedstawionych u dołu strony magnesów, by zmienić położenie kolumny telefon. Napisz tyle różnych polecień, ile będziesz w stanie, a następnie dla każdego z nich narysuj wynikową strukturę tabeli.

telefon	id_kontaktu	nazwisko	imie	email
---------	-------------	----------	------	-------

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)

id_kontaktu	nazwisko	imie	email	telefon
-------------	----------	------	-------	---------

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)

id_kontaktu	telefon	nazwisko	imie	email
-------------	---------	----------	------	-------

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)

id_kontaktu	nazwisko	telefon	imie	email
-------------	----------	---------	------	-------

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)



Przyklej swoje magnesiki na końcu każdego z tych polecień.

FIRST

nazwisko

LAST

SECOND

AFTER

BEFORE

;

FIFTH

Użyj średnika tyle razy, ile będziesz pożrebował.



Magnesiki ze słowami kluczowymi SQL. Rozwiążanie

Skorzystaj z przedstawionych u dołu strony magnesików, by zmienić położenie kolumny telefon. Napisz tyle różnych poleceń, ile będziesz w stanie, a następnie dla każdego z nich narysuj wynikową strukturę tabeli.

ALTER TABLE my_contacts

ADD COLUMN telefon VARCHAR(10)

FIRST ;

Zastosowanie słowa kluczowego FIRST sprawia, że kolumna telefon zostanie umieszczona przed wszystkimi pozostałymi kolumnami tabeli.

telefon	id_kontaktu	nazwisko	imie	email
---------	-------------	----------	------	-------

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)

LAST ;

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)

FIFTH ;

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)

;

Zastosowanie słowa kluczowego LAST spowoduje, że kolumna telefon zostanie umieszczona na samym końcu tabeli; podobne rezultaty da zastosowanie słowa FIFTH (ang. piąty), jak również całkowite pominięcie określenia położenia nowej kolumny w tabeli.

id_kontaktu	nazwisko	imie	email	telefon
-------------	----------	------	-------	---------

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)

SECOND ;

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)

BEFORE nazwisko ;

Zastosowanie słowa kluczowego SECOND (ang. drugi) sprawi, że nowa kolumna zostanie umieszczona jako druga kolumna tabeli; podobnie zadziała użycie słowa kluczowego BEFORE, jeśli podamy za nim kolumnę nazwisko.

id_kontaktu	telefon	nazwisko	imie	email
-------------	---------	----------	------	-------

ALTER TABLE moje_kontakty

ADD COLUMN telefon VARCHAR(10)

AFTER nazwisko ;

Zastosowanie wyrażenia AFTER nazwisko sprawi, że nowa kolumna zostanie dodana za kolumną nazwisko — czyli jako trzecia kolumna tabeli. Gdybyśmy mieli do dyspozycji magnesik THIRD (ang. trzeci), to jego użycie datoby identyczne wyniki.

id_kontaktu	nazwisko	telefon	imie	email
-------------	----------	---------	------	-------

Modyfikowanie tabel

Polecenie ALTER pozwala na wprowadzanie niemal dowolnych modyfikacji w strukturze tabeli, bez konieczności usuwania i ponownego wpisywania do niej danych. Ale uważaj — zmieniając typ kolumny, możesz ryzykować utratę zapisanych w niej informacji.

Przeróbki Bazodanowa

NASZE USŁUGI NA RZECZ ISTNIEJĄCYCH TABEL

ZMIANA zarówno nazwy, jak i typu danych istniejącej kolumny. *

MODYFIKACJA typu danych oraz położenia istniejącej kolumny. *

DODAWANIE kolumn do tabeli – sam wybierasz typ danych kolumny.

USUWANIE kolumny z tabeli. *

* Może się wiązać z utratą danych; nie udzielamy gwarancji na dane.

To tylko mała modyfikacja, nie będzie boleć.

USŁUGI DODATKOWE

Zmień kolejność kolumn w swojej tabeli
(możliwość dostępna wyłącznie w razie użycia polecenia ADD)




WYSIL SZARE KOMÓRKI

Dlaczego przedstawiona obok tabela może wymagać zmian?

projekty

numer	infooproj	nazwfirmwyk
1	malowanie elewacji domu	SUP-Rem
2	instalacja nowej kuchni	RemBudSzpachelka
3	położenie drewnianej podłogi	Kem-Rem
4	przekładka dachu	Jackowski i Syn

Ekstremalne metamorfozy tabel

Zaczniemy nasze modyfikacje od tabeli, która naprawdę potrzebuje gruntownych zmian.

Witamy w programie
Ekstremalne metamorfozy tabel!
Na kilku następnych stronach przekonasz się, że nawet najgorszą tabelę można przerobić na cudo, z którego będzie dumna każda, nawet najbardziej wymagająca baza danych.

Ta nazwa niewiele nam mówi o przeznaczeniu i zawartości tabeli.
Te nazwy kolumn nic nam nie mówią o danych, jakie są o których mowa, jakie są w nich zapisywane.

projekty

Może warto rozdzielić poszczególne części nazwy znakiem podkreślenia? To prawdopodobnie poprawioby czytelność.

numer	infooproj	nazwfirmwyk
1	malowanie elewacji domu	SUP-Rem
2	instalacja nowej kuchni	RemBudSzpachelka
3	położenie drewnianej podłogi	Kem-Rem
4	przekładka dachu	Jackowski i Syn



Choć nazwy kolumn oraz samej tabeli nie są może najlepsze, to jednak informacje zapisane w tabeli są w porządku i nie chcemy ich w żaden sposób modyfikować.

Zastosujmy polecenie DESCRIBE, by poznać szczegóły struktury tabeli. W ten sposób będziemy mogli przekonać się, czy jest jakaś kolumna będąca kluczem głównym oraz jakie są typy poszczególnych kolumn.

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> DESC projekty;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| numer | int<11> | YES  |      | NULL    |        |
| infooproj | varchar<50> | YES  |      | NULL    |        |
| nazwfirmwyk | varchar<30> | YES  |      | NULL    |        |
+-----+-----+-----+-----+-----+
3 rows in set <0.01 sec>

mysql>
```

Zmiana nazwy tabeli

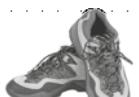
W swojej obecnej postaci tabela przedstawiona na poprzedniej stronie będzie nam przysparzać pewnych problemów. Jednak dzięki zastosowaniu polecenia ALTER przystosujemy ją do przechowywania informacji o projektach modernizacji domów, które przydadzą się zwłaszcza w przypadku bardziej zaniedbanych budynków.

```
ALTER TABLE projekty
RENAME TO lista_projektow;
```

To tak, jakby ktoś mówić po angielsku! Oznacza to, że chcemy ZMIENIĆ NAZWĘ tabeli.

„projekty” to dotychczasowa nazwa tabeli.

„lista_projektow” to nowa nazwa, którą chcemy nadać tabeli.



Ćwiczenie

Poniższy opis pozwoli Ci się zorientować, jakie inne modyfikacje należy wprowadzić w tabeli. Poniższy akapit zawiera zdania opisujące, w jaki sposób mamy zamiar korzystać z tabeli; na ich podstawie określ nazwy kolumn i zapisz je w pustych prostokątach.

id_proj

Aby ZNORMALIZOWAĆ tabelę, dodamy do niej kolumnę klucza głównego, zawierającą unikalny numer projektu. Oprócz tego będziemy potrzebowali kolumn zawierających: opis wszelkich prac wykonywanych w ramach danego projektu, datę rozpoczęcia tych prac, szacunkowy koszt oraz nazwę firmy budowlanej zajmującej się pracami, no i oczywiście telefon do tej firmy.

Rozwiążanie ćwiczenia



Ćwiczenie

Poniższy opis pozwoli Ci się zorientować, jakie inne modyfikacje należy wprowadzić w tabeli. Poniższy akapit zawiera zdania opisujące, w jaki sposób mamy zamierzyć korzystać z tabeli; na ich podstawie określ nazwy kolumn i zapisz je w pustych prostokątach.

Nie przejmuj się, jeśli zaproponowane przez Ciebie nazwy kolumn nie odpowiadają naszym. Skracanie nazw kolumn jest do pewnego stopnia dopuszczalne, o ile tylko przeznaczenie kolumny i zapisywanych w nich danych pozostanie zrozumiałe.

Zadbaj o to, by skrócone nazwy, takie jak „opis_proj”, były zrozumiałe i sensowne nie tylko dla Ciebie, lecz także dla wszystkich innych osób, które mogą korzystać z bazy danych.

opis_proj

id_proj

data_rozp

koszt_szac

firma_nazwa

firma_tel

Aby ZNORMALIZOWAĆ tabelę, dodamy do niej kolumnę klucza głównego, zawierającą unikalny numer projektu. Oprócz tego będziemy potrzebowali kolumn zawierających: opis wszelkich prac wykonywanych w ramach danego projektu, datę rozpoczęcia tych prac, szacunkowy koszt oraz nazwę firmy budowlanej zajmującej się pracami, no i oczywiście telefon do tej firmy.

Musimy poczynić pewne plany

lista_projektow

numer	infooproj	nazwfirmwyk
1	malowanie elewacji domu	SUP-Rem
2	instalacja nowej kuchni	RemBudSzpachelka
3	położenie drewnianej podłogi	Kem-Rem
4	przekładka dachu	Jackowski i Syn

Jak widać, dane, które mają się znaleźć w trzech spośród docelowych kolumn naszej tabeli, już są w niej zapisane. A zatem zamiast tworzyć nowe kolumny, możemy zmienić nazwy już istniejących. W ten sposób nie będziemy musieli ponownie zapisywać w tabeli tych samych danych.



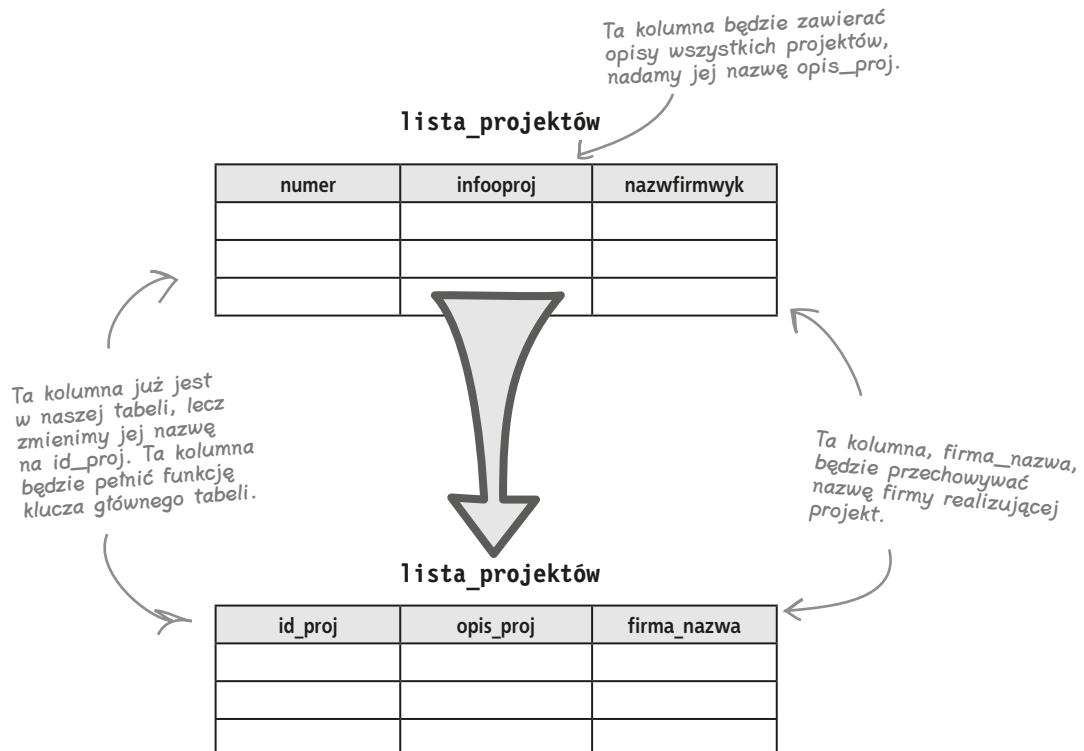
Przezbrajanie kolumn

Dysponujemy już planem i wiemy, od czego zacząć wprowadzanie modyfikacji w naszej tabeli. Możemy użyć polecenia ALTER i za jego pomocą przypisać nowe nazwy trzem istniejącym już kolumnom tabeli:

- ⇒ Kolumna numer stanie się naszym kluczem głównym i przyjmie nazwę **id_proj**.
- ⇒ Kolumna infooproj, zawierająca opisy poszczególnych projektów modernizacji, stanie się kolumną **opis_proj**.
- ⇒ Nazwa kolumny nazwfirmwyk, zawierającej nazwę firmy realizującej projekt, zostanie zmieniona na **firma_nazwa**.



I w ten sposób pozostaje nam jedynie dodać do tabeli trzy kolumny: koszt_szac, firma_tel oraz data_rozp.



Zmiany strukturalne

Zdecydowaliśmy, że wszystkie trzy dotychczasowe kolumny tabeli zostaną przekształcone w kolumny tabeli docelowej. Oprócz modyfikacji samych nazw tych kolumn warto też zwrócić uwagę na typ przechowywanych w nich informacji.

Oto struktura oryginalnej wersji naszej tabeli:

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> DESC lista_projektow;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| numer | int(11) | YES |   | NULL    |       |
| infoopproj | varchar(50) | YES |   | NULL    |       |
| nazwfirmwyk | varchar(30) | YES |   | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set <0.01 sec>

mysql>
```



WYŁĘŻ UMYSŁ

Spójrz na typ każdej z dotychczasowych kolumn tabeli i określ, czy odpowiada on danym, jakie w przyszłości będziemy w nich zapisywali.

Polecenia ALTER i CHANGE

Kolejną czynnością, jaką wykonamy, będzie zmiana nazwy kolumny numer na id_proj i określenie, że jej wartości mają być automatycznie inkrementowane. Następnie zdefiniujemy tę kolumnę jako kolumnę klucza głównego tabeli. Choć wszystkie te operacje mogą się wydawać skomplikowane, jednak w rzeczywistości są całkiem proste. Okazuje się nawet, że możemy je wykonać przy użyciu jednego polecenia SQL:

Tym razem używamy polecenia CHANGE COLUMN, gdyż zmieniamy nie tylko nazwę, lecz także i typ danych kolumny, która wcześniej nosiła nazwę „numer”.

Caty czas operujemy na tej samej tabeli... pamiętaj tylko, że wcześniej zmieniliśmy jej nazwę.

„id_proj” to nowa nazwa, którą chcemy nadać kolumnie...

... poza tym w kolumnie mają być zapisywane automatycznie inkrementowane liczby całkowite i nie będą mogły się w niej pojawiać wartości NULL.

```
ALTER TABLE lista_projektow  
CHANGE COLUMN numer id_proj INT NOT NULL AUTO_INCREMENT,  
ADD PRIMARY KEY ('id_proj');
```

A to jest polecenie, które nakazuje, by nasza nowa kolumna „id_proj” została zastosowana jako klucz główny tabeli.

Zaostrz ołówek



Naszkicuj strukturę tabeli po wykonaniu powyższego polecenia SQL.

Zmiana dwóch kolumn przy użyciu jednego polecenia SQL

A teraz, używając jednego polecenia *SQL*, zmienimy nie jedną, lecz aż dwie kolumny tabeli. Zmienimy nazwy kolumn `infooproj` oraz `nazwfirmwyk`, a przy okazji zmienimy także ich typy danych. W tym celu wystarczy umieścić dwie klauzule `CHANGE COLUMN` w jednym poleceniu `ALTER TABLE`, rozdzielając je od siebie przecinkiem.

```
ALTER TABLE lista_projektow
    CHANGE COLUMN infooproj opis_proj VARCHAR(100),
    CHANGE COLUMN nazwfirmwyk firma_nazwa VARCHAR(30);
```

„`infooproj`” to stara nazwa kolumny, którą chcemy zmienić przy użyciu tego polecenia.

„`opis_proj`” to nowa nazwa kolumny.

Zwiększamy ilość znaków w tej kolumnie, dzięki czemu opisy projektów będą mogły być dłuższe.

Zmienia się także kolumna, która uprzednio nosiła nazwę „`nazwfirmwyk`”...

... teraz nazwiemy ją „`firma_nazwa`”; a to jest nowy typ danych tej kolumny.



Uwaga!

Jeśli zmieniasz typ danych kolumny, możesz ryzykować utratę danych.

Jeśli nowy typ danych nie będzie zgodny z jej dotychczasowym typem danych, to polecenie `ALTER` nie zostanie wykonane, a system zarządzania bazą danych poinformuje Cię o wystąpieniu błędów.

Jednak znacznie gorsze jest to, że jeśli nowy i dotychczasowy typ danych kolumny są ze sobą zgodne, to istnieje niebezpieczeństwo skrócenia danych.

Oto przykład: po zmianie typu danych kolumny z `varchar(10)` na `varchar(1)` słowo „Gonzo” zostanie skrócone do jednej litery — „G”.

To samo dotyczy typów liczbowych. Oczywiście można zmienić typ kolumny na inny, lecz oznacza to, że dane zapisane w kolumnie zostaną skonwertowane do nowego typu, a to może spowodować utratę części tych danych.



Gdybym chciał zmienić typ kolumny, tak by na przykład można w niej było zapisywać więcej znaków, lecz jednocześnie nie chciałbym zmieniać jej nazwy, to mogę powtórnie podać jej nazwę, jak w poniższym przykładzie, prawda?

```
ALTER TABLE moja_tabela  
CHANGE COLUMN nawa_kolumny nazwa_kolumny NOWYTYP;
```

Oczywiście można tak zrobić, istnieje jednak prostsze rozwiązanie.

Mianowicie możesz wykorzystać słowo kluczowe MODIFY. Pozwala ono zmienić typ kolumny i jednocześnie pozostawić jej dotychczasową nazwę.

Załóżmy, że potrzebujesz zwiększyć dopuszczalną ilość znaków w kolumnie przechowującej opis projektu — opis_proj. Chciałbyś zmienić jej typ na VARCHAR(120). Oto co powinieneś w tym celu zrobić:

ALTER TABLE lista_projektow

MODIFY COLUMN opis_proj VARCHAR(120);

Nazwa modyfikowanej kolumny.

Nowy typ kolumny.

No i oczywiście upewnij się, że zmiana typu kolumny nie spowoduje obcięcia danych, które już są w niej zapisane!

Nie istnieje pytania

Q: A co mogę zrobić, aby zmienić kolejność kolumn w tabeli? Czy mogę wykonać polecenie w postaci: **ALTER TABLE MODIFY COLUMN opis_proj AFTER nazwa_kolumny;?**

Q: Okazuje się, że po utworzeniu tabeli nie można już modyfikować kolejności jej kolumn. Najlepszym rozwiązaniem będzie zatem dodanie nowej kolumny w odpowiednim miejscu i usunięcie starej kolumny; jednak w takim przypadku utracisz wszystkie dane zapisane w dotychczasowej kolumnie.

Q: Ale czy nieodpowiednia kolejność kolumn nie będzie mi w przyszłości przysparzać problemów?

Q: Nie. Ponieważ tak się szczęśliwie składa, że kolejność, w jakiej kolumny będą zwracane w wynikach, można określić w zapytaniu SELECT. Kolejność, w jakiej dane są zapisane na dysku, nie ma znaczenia, co pokazują poniższe przykłady:

```
SELECT kolumna3, kolumna1 FROM twoja_tabela;  
albo
```

```
SELECT kolumna 1, kolumna3 FROM twoja_tabela;  
Jak widać, kolejność kolumn może być dowolna.
```



Ćwiczenie



0

Hej, właśnie rozmawiam
przez telefon z moim
agentem. Czy moglibyście
dodać te brakujące
kolumny?

lista_projektow

id_proj	opis_proj	firma_nazwa
1		
2		
3		

Mamy zatem dodać do tabeli projektów trzy nowe kolumny, w których będą zapisywane: **numer telefonu**, **data rozpoczęcia** prac oraz **szacunkowy koszt**.

W poniższych pustych wierszach napisz jedno polecenie ALTER, które doda do tabeli projektów trzy nowe kolumny; zwróć przy tym uwagę na typy danych. Następnie uzupełnij tabelę umieszczoną u dołu strony.

.....

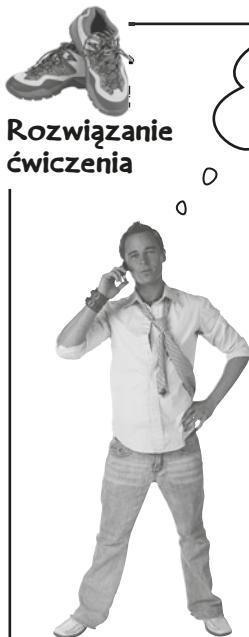
.....

.....

.....

lista_projektow

Rozwiązywanie kolejnego ćwiczenia



Rozwiązywanie ćwiczenia

Hej, właśnie rozmawiam przez telefon z moim agentem.
Czy moglibyście dodać te brakujące kolumny?

lista_projektow

id_proj	opis_proj	firma_nazwa
1		
2		
3		

Wciąż mamy dodać do tabeli projektów trzy nowe kolumny, w których będą zapisywane: **numer telefonu**, **data rozpoczęcia** prac oraz **szacunkowy koszt**.

W poniższych pustych wierszach napisz jedno polecenie ALTER, które doda do tabeli projektów trzy nowe kolumny; zwróć przy tym uwagę na typy danych. Następnie uzupełnij tabelę umieszczoną u dołu strony.

Pole VARCHAR o długości 10 znaków pozwoli nam zapisać numer telefonu wraz z numerem kierunkowym.

ALTER TABLE lista_projektow

Dodajemy nowe kolumny – stąd zastosowaliśmy słowo kluczowe ADD.

- ADD COLUMN firma_tel VARCHAR(10),
- ADD COLUMN data_rozp DATE,
- ADD COLUMN koszt_szac DECIMAL(7,2);

Czy pamiętasz jeszcze typ DEC? Zażądaliśmy, by to pole miało długość 7 cyfr, z czego dwie będą przeznaczone na miejsca po przecinku.

lista_projektow

id_proj	opis_proj	firma_nazwa	firma_tel	data_rozp	koszt_szac
1					
2					
3					

Szybko! Usuń tę kolumnę

Wstrzymaj wszystkie prace!

Właśnie dowiedzieliśmy się, że realizacja naszego projektu została wstrzymana. W efekcie możemy usunąć kolumnę `data_rozp`. W końcu nie ma sensu przechowywania w tabeli kolumny, która nikomu nie jest potrzebna i jedynie bezsensownie zajmuje miejsce w bazie danych.

Jedna z niepisanych zasad korzystania z baz danych zaleca, by w tabelach umieszczać tylko te kolumny, które faktycznie są potrzebne. Jeśli nie potrzebujesz jakiejś kolumny, usuń ją. Dysponując poleceniem ALTER, w przyszłości, kiedy kolumna stanie się potrzebna, będziesz mógł ją dodać bez większych problemów.

Im więcej jest kolumn w tabeli, tym cięższą pracę musi wykonywać system zarządzania bazami danych, a sama baza zajmuje więcej miejsca na dysku. Choć w przypadku małych tabel trudno to zauważyc, to jednak w miarę wzrostu ilości danych przekonasz się, że uzyskanie wyników trwa dłużej, a procesor musi znacznie częściej pracować.



Zaostrz ołówek



A zatem do dzieła; nie ociągaj się i napisz polecenie SQL, które usunie z naszej tabeli kolumnę `data_rozp`. Nie pokazaliśmy jeszcze składni tego polecenia, niemniej jednak chyba sam możesz spróbować.

.....
.....
.....

Zaostrz ołówek



Rozwiążanie

A zatem do dzieła; nie ociągaj się i napisz polecenie SQL, które usunie z naszej tabeli kolumnę `data_rozp`. Nie pokazaliśmy jeszcze składni tego polecenia, niemniej jednak chyba sam możesz spróbować.

To nazwa tabeli
– `lista_projektow`.



`ALTER TABLE lista_projektow`

`DROP COLUMN data_rozp;`

Jeśli chcesz usunąć
kolumnę, to postuż się
polecienniem `DROP`.
To naprawdę tatwizna!



A to nazwa kolumny,
której chcesz usunąć.



**Kiedy usuniesz kolumnę, wszystkie zapisane w niej
informacje także zostaną usunięte!**

Z polecenia `DROP COLUMN` należy korzystać bardzo ostrożnie.

W pierwszej kolejności warto, żebyś wyświetlił całą jej zawartość i upewnił się, że
naprawdę chcesz ją usunąć! Już lepiej mieć w bazie trochę niepotrzebnych danych,
niż żeby nagle się okazało, że brakuje jakichś ważnych informacji.



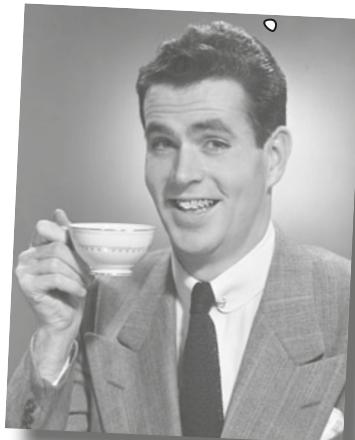
Ćwiczenie

Odpicuj moją tabelę

To proste. Weź tę pożądaną godną tabelę „Przed” z danymi o samochodach i używając polecenia ALTER, przekształć ją na wspaniałą tabelę „Po”. Jednym z problemów jest wymóg, by nie utracić dotychczasowych informacji zapisanych w tabeli. Czy jesteś w stanie podać temu wyzwaniu?

Uwaga! Dodatkowe punkty przewidziano za wykonanie wszystkich modyfikacji przy użyciu jednego polecenia ALTER.

Nadszedł czas, by zmienić Twoją wypróbowaną, lecz przestarzałą tabelę na cudeńko, jakiemu nie oprą się żadne informacje, i przenieść ją na taki poziom bazodanowego odpicowania, o jakim nawet Ci się nie śniło.



Przed starocie

kolor	rok	marka	mo	koszt
srebrny	1998	Porsche	Boxter	94500
NULL	2000	Jaguar	XJ	102000
czerwony	2002	Cadillac	Escalade	62000

Po samochody

id_samochodu	numer	marka	model	kolor	rok	cena
1	RNKLK66N3G213481	Porsche	Boxter	srebrny	1998	94500
2	SAEDA44B175B04113	Jaguar	XJ	NULL	2000	102000
3	3GYEK63NT2G280668	Cadillac	Escalade	czerwony	2002	62000



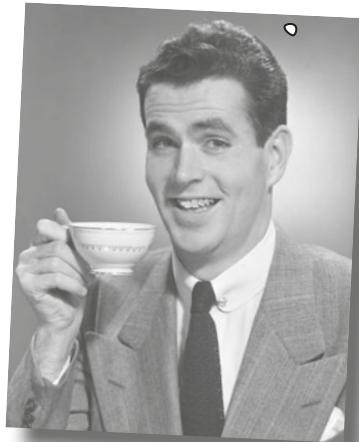
Rozwiążanie
ćwiczenia

Odpicuj moją tabelę

To proste. Weź tę pożądaną godną tabelę „Przed” z danymi o samochodach i używając polecenia ALTER, przekształć ją na wspaniałą tabelę „Po”. Jednym z problemów jest wymóg, by nie utracić dotychczasowych informacji zapisanych w tabeli. Czy jesteś w stanie podołać temu wyzwaniu?

Uwaga! Dodatkowe punkty przewidziano za wykonanie wszystkich modyfikacji przy użyciu jednego polecenia ALTER.

Nadszedł czas, by zmienić Twoją wypróbowaną, lecz przestarzałą tabelę na cudeńko, jakiemu nie oprą się żadne informacje, i przenieść ją na taki poziom bazodanowego odpicowania, o jakim nawet Ci się nie śniło.



Przed

starocie

kolor	rok	marka	mo	koszt
srebrny	1998	Porsche	Boxter	94500
NULL	2000	Jaguar	XJ	102000
czerwony	2002	Cadillac	Escalade	62000

Po

samochody

id_samochodu	numer	marka	model	kolor	rok	cena
1	RNKLK66N3G213481	Porsche	Boxter	srebrny	1998	94500
2	SAEDA44B175B04113	Jaguar	XJ	NULL	2000	102000
3	3GYEK63NT2G280668	Cadillac	Escalade	czerwony	2002	62000

Przed przystąpieniem do modyfikowania tabeli warto, byś wykonat polecenie DESCRIBE, sprawdzić typy kolumn i upewnić się, że zmiany nie spowodują obcięcia informacji zapisanych w tabeli.

ALTER TABLE starocie



RENAME TO samochody,

ALTER TABLE samochody

ADD COLUMN id_samochodu INT NOT NULL AUTO_INCREMENT FIRST,

ADD PRIMARY KEY(id_samochodu),

ALTER TABLE samochody

ADD COLUMN numer VARCHAR(16) SECOND,

CHANGE COLUMN mo model VARCHAR(20),

MODIFY COLUMN kolor AFTER model,

Najpierw musisz zmienić nazwę kolumny „mo” na „model”, a dopiero potem umieścić za nią kolumny koloru i roku.

Wraz ze zmianą nazwy kolumny musisz określić jej typ.

MODIFY COLUMN rok SIXTH,



Ten sam efekt mógłbyś uzyskać, używając kodu „rok AFTER model” lub „rok BEFORE cena”.

CHANGE COLUMN koszt cena DECIMAL(7,2);

Nie istniejąca grupa pytań

P: Wcześniej napisaliście, że przy użyciu polecenia **MODIFY** nie można zmienić kolejności kolumn. Jednak oprogramowanie do obsługi baz danych, którego używam, pozwala na zmianę kolejności kolumn. Jak ono to robi?

O: Twój oprogramowanie w niezauważalny sposób wykonuje całą grupę operacji. Kopiuje wartości z kolumny, którą chcesz przenieść, zapisuje je w tymczasowej tabeli pomocniczej, usuwa dotychczasową kolumnę, modyfikuje tabelę, dodając we wskazanym jej miejscu nową kolumnę o identycznej nazwie co kolumna usunięta, kopiuje dane z tabeli pomocniczej do nowej kolumny i w końcu usuwa tabelę pomocniczą.

Zazwyczaj, jeśli w kolumnach już są zapisane jakieś dane, a Ty nie korzystasz z żadnego bardziej zaawansowanego oprogramowania, lepiej pozostawić kolumny w oryginalnej kolejności. W końcu kolejność kolumn zawsze można określić w poleceniu **SELECT**.

P: Czyli kolejność kolumn można w prosty sposób modyfikować wyłącznie w przypadku dodawania nowych kolumn?

O: Owszem. Optymalnym rozwiązaniem będzie przemyślenie kolejności kolumn i określenie jej już podczas tworzenia tabeli.

P: Co zrobić, jeśli przez przypadek określiłem klucz główny, lecz po jakimś czasie dojdę do wniosku, że kluczem głównym powinna być inna kolumna? Czy istnieje możliwość oznaczenia, że kolumna nie jest kluczem głównym bez jednoczesnej utraty danych, które są w niej zapisane?

O: Owszem, można to zrobić i to bardzo prosto:
`ALTER TABLE tabela DROP PRIMARY KEY;`

P: A co z **AUTO_INCREMENT**?

O: Można zażądać, by wartości w kolumnie były automatycznie inkrementowane. Poniżej pokazaliśmy, jak to zrobić:

`ALTER TABLE tabela CHANGE kolumna_id kolumna_id INT(11) NOT NULL AUTO_INCREMENT;`

Można też usunąć automatyczną inkrementację wartości:

`ALTER TABLE tabela CHANGE kolumna_id kolumna_id INT(11) NOT NULL;`

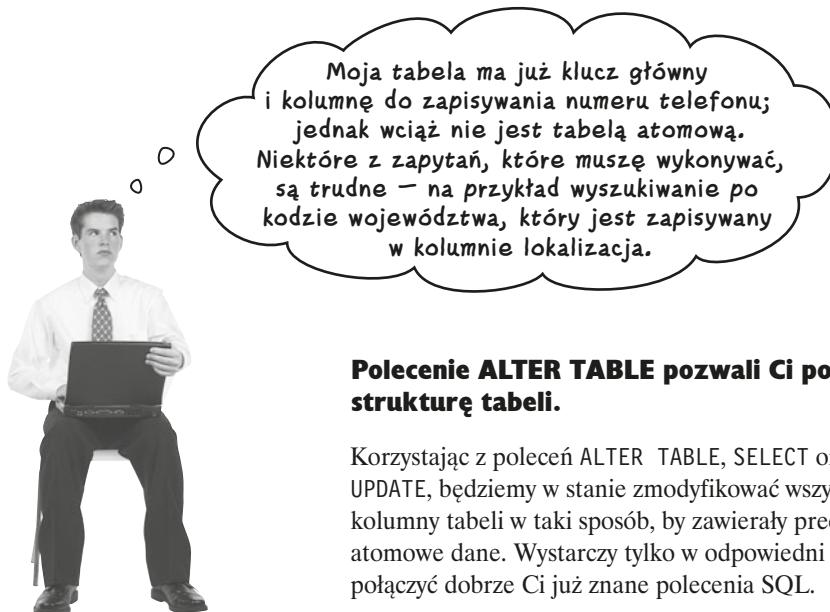
Koniecznie trzeba pamiętać, że w tabeli może być tylko jedna kolumna oznaczona jako **AUTO_INCREMENT**; musi to być kolumna typu **INTEGER**, w której nie można umieszczać wartości **NULL**.

CELNE SPOSTRZEŻENIA



- ◆ Używaj klauzuli **CHANGE**, kiedy chcesz zmienić zarówno nazwę kolumny, jak i jej typ.
- ◆ Używaj klauzuli **MODIFY**, kiedy chcesz zmienić wyłącznie typ kolumny.
- ◆ Klauzula **DROP COLUMN** usuwa kolumnę z tabeli.
- ◆ Klauzula **RENAME** pozwala na zmianę nazwy tabeli.

- ◆ Kolejność kolumn można zmieniać, używając następujących słów kluczowych i wyrażeń: **FIRST**, **LAST**, **BEFORE** nazwa_kolumny, **AFTER** nazwa_kolumny, **SECOND**, **THIRD**, **FOURTH** i tak dalej.
- ◆ W niektórych systemach zarządzania bazami danych kolejność kolumn można zmieniać wyłącznie podczas ich dodawania do tabeli.



Polecenie ALTER TABLE pozwali Ci poprawić strukturę tabeli.

Korzystając z poleceń ALTER TABLE, SELECT oraz UPDATE, będziemy w stanie zmodyfikować wszystkie kolumny tabeli w taki sposób, by zawierały precyzyjne, atomowe dane. Wystarczy tylko w odpowiedni sposób połączyć dobrze Ci już znane polecenia SQL.

Przeanalizujmy polecenie CREATE TABLE użyte do utworzenia tabeli moje_kontakty Grześka:

```
CREATE TABLE moje_kontakty
(
    id_kontaktu INT NOT NULL AUTO_INCREMENT,
    nazwisko varchar(30) default NULL,
    imie varchar(20) default NULL,
    email varchar(50) default NULL,
    plec char(1) default NULL,
    data_urodzenia default NULL,
    zawod varchar(50) default NULL,
    lokalizacja varchar(50) default NULL,
    stan varchar(20) default NULL,
    zainteresowania varchar(100) default NULL,
    szuka varchar(100) default NULL,
    PRIMARY KEY (id_kontaktu)
)
```

Te dwa wiersze kodu dodaliśmy, by utworzyć i wskazać kolumnę klucza głównego.

Te cztery kolumny raczej nie są atomowe i mogą wymagać wprowadzenia pewnych zmian przy użyciu polecenia ALTER TABLE.

Dokładniejsza analiza nieatomowej kolumny lokalizacji

Czasami Grzesiek może potrzebować informacji o województwie lub miejscowości, w których mieszkają osoby zapisane w jego bazie. Dlatego też kolumnę lokalizacja można podzielić na dwie niezależne kolumny. Zobaczmy, jak obecnie wyglądają dane zapisane w tej kolumnie:

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT lokalizacja FROM moje_kontakty;
+-----+
| lokalizacja |
+-----+
| Gliwice, SL |
| Krakow, MP |
| Wrocław, DS |
| Elblag, WM |
| Katowice, SL |
| Tychy, SL |
| Wrocław, DS |
| Poznań, WP |
| Wroclaw, DS |
| ...          |
+-----+
```

Niewielki fragment danych zapisanych w kolumnie lokalizacja tabeli moje_kontakty.

Gliwice, SL
Krakow, MP
Wrocław, DS
Elblag, WM
Katowice, SL

Dwuliterowy kod określający województwo.

Nazwa miasta.

Przecinek.

Jak widać, dane są zapisywane w spójny sposób. Najpierw podawana jest nazwa miejscowości, za nią przecinek, a po nim dwuliterowe oznaczenie województwa. Ponieważ dane są konsekwentnie zapisywane w opisany powyżej sposób, zatem będziemy w stanie oddzielić nazwę miejscowości do kodu województwa.



WYTĘŻ
UMYSŁ

Dlaczego chcemy oddzielić nazwę miejscowości od kodu województwa?

Jak myślisz, co teraz zrobimy?

Poszukaj wzorca

Każda wartość umieszczona w tabeli `moje_kontakty` w kolumnie `lokalizacja` jest zapisana w taki sam sposób: na początku znajduje się nazwa miejscowości, po niej przecinek, za którym umieszczone jest dwuliterowe oznaczenie województwa. Dzięki temu, że wartości są konsekwentnie zapisywane w ten sam sposób i pasują do pewnego wzorca, będziemy mogli je rozdzielić na dwie atomowe informacje.

nazwa miejscowości, XX

Ten przecinek, który zawsze jest umieszczany przed kodem województwa, może się nam bardzo przydać...

Te dwie litery umieszczone na końcu zawsze zawierają kod określający województwo. Gdyby w naszej tabeli znajdowała się osobna kolumna z informacjami o województwie, to znalazłyby się w niej właśnie te dane.

Wszystko, co znajduje się przed przecinkiem, możemy przenieść do nowej kolumny, która będzie zawierać nazwy miejscowości.

Z kolei ostatnie dwa znaki dotychczasowych wartości kolumny `lokalizacja` możemy przenieść do nowej kolumny o nazwie `wojewodztwo`.

nazwa miejscowości, XX

Potrzebujemy funkcji, która pozwoli nam pobrać fragment łańcucha znaków znajdujący się przed przecinkiem...

... oraz drugiej, która pozwoli nam pobrać ostatnie dwa znaki łańcucha.

Zaostrz ołówek



Napisz polecenie ALTER TABLE, które doda do tabeli `moje_kontakty` kolumny `miejscowosc` oraz `wojewodztwo`.

.....
.....
.....

```
ADD COLUMN wojewodztwo CHAR(2);
ADD COLUMN miejscowosc VARCHAR(50),
ALTER TABLE moje_kontakty
```

Kilka wygodnych funkcji łańcuchowych

Wyróżniliśmy w naszych danych dwa wzorce. Teraz musimy pobrać dwuliterowy kod województwa i zapisać go w kolumnie województwo. Oprócz tego cały fragment łańcucha umieszczony przed przecinkiem chcemy zapisać w kolumnie mi_ejscowosc. Dwie nowe kolumny już dodałeś do tabeli, a poniżej przedstawiliśmy, jak pobrać informacje, które będziemy chcieli w nich zapisać.

Aby pobrać dwa ostatnie znaki łańcucha

Użyj funkcji LEFT() lub RIGHT(), by pobrać określona ilość znaków z kolumny.

Wartości zapisywane w kolumnach typów CHAR lub VARCHAR są nazywane łańcuchami znaków.

```
SELECT RIGHT(lokalizacja, 2) FROM moje_kontakty;
```

Zaczniemy od PRAWEJ strony kolumny. (Funkcja LEFT pozwala na pobranie znaków z lewej strony kolumny i jest używana w taki sam sposób).

To nazwa kolumny, na której operuje funkcja.

A tu określamy, ile znaków z PRAWEJ strony kolumny należy pobrać.

Aby pobrać fragment łańcucha przed przecinkiem

Do pobrania fragmentu zawartości kolumny tekstowej można użyć funkcji SUBSTRING_INDEX(). Zwróci ona fragment łańcucha umieszczony przed podanym znakiem lub łańcuchem. Oznacza to, że możemy zapisać przecinek między apostrofami, a funkcja SUBSTRING_INDEX() zwróci całą zawartość kolumny umieszczoną przed przecinkiem.

Funkcje łańcuchowe pozwalają na pobieranie fragmentów kolumn tekstowych.

```
SELECT SUBSTRING_INDEX(lokalizacja, ',', 1) FROM moje_kontakty;
```

Ta funkcja pobiera fragment kolumny lub łańcucha znaków. Poszukuje ona łańcucha znaków podanego wewnętrz apostrofów (w tym przypadku jest to przecinek) i pobiera wszystko, co jest zapisane przed nim.

W tym miejscu ponownie jest podawana nazwa kolumny.

A to jest przecinek, którego poszukuje funkcja.

To jest najtrudniejsze. Cyfra 1 oznacza, że interesuje nas pierwszy przecinek. Gdybyśmy umieścili tu cyfrę 2, oznaczałoby to, że interesuje nas drugi przecinek, a funkcja pobierałaby wszystko, co się znajduje przed nim.



Ćwiczenie

Wypróbuj to w domu

SQL udostępnia grupę funkcji pozwalających na manipulowanie łańcuchami znaków zapisanymi w bazie danych. Łąncupy znaków są zapisywane w kolumnach tekstowych, przy czym zazwyczaj są to kolumny typów VARCHAR lub CHAR.

Poniżej przedstawiliśmy kilka najbardziej przydatnych i najczęściej używanych funkcji łańcuchowych. Wypróbowaj każdą z nich — wystarczy, że umieścisz wywołanie funkcji, którą chcesz sprawdzić, w poleceniu SELECT.

Funkcja **SUBSTRING(łańcuch_znaków, początek, długość)** zwraca fragment podanego łańcucha_znaków, rozpoczynający się od znaku o indeksie początek. Ostatni argument wywołania funkcji określa długość zwracanego fragmentu łańcucha.

```
SELECT SUBSTRING('Bielsko Biala, SL', 8, 9);
```

Funkcje **UPPER(łańcuch_znaków)** oraz **LOWER(łańcuch_znaków)** zmieniają wszystkie litery przekazanego łańcucha na wielkie lub małe.

```
SELECT UPPER('uSa');
```

```
SELECT LOWER('spaGeTTI');
```

Funkcja **REVERSE(łańcuch_znaków)** odwraca kolejność liter w łańcuchu.

```
SELECT REVERSE('spaGeTTI');
```

Funkcje **LTRIM(łańcuch_znaków)** oraz **RTRIM(łańcuch_znaków)** zwracają przekazany łańcuch, z którego początku (w przypadku funkcji LTRIM()) lub końca (w przypadku funkcji RTRIM()) usunięto niepotrzebne odstępy.

```
SELECT LTRIM(' pieski los '');
```

```
SELECT RTRIM(' pieski los '');
```

Funkcja **LENGTH(łańcuch_znaków)** zwraca ilość znaków w łańcuchu.

```
SELECT LENGTH('Wroclaw, DS');
```

WAŻNE: Funkcje łańcuchowe NIE zmieniają danych zapisanych w tabeli, a jedynie zwracają zmodyfikowany łańcuch znaków jako wynik zapytania.

JAKIE JEST MOJE PRZEZNACZENIE?

Próbowiemy pobrać dane z kolumny lokalizacja i przenieść je do dwóch nowych kolumn: miejscowości i województwo.

Poniżej opisaliśmy czynności, które w tym celu planujemy wykonać. Dopasuj każdą z nich ze słowem lub słowami kluczowymi SQL, których trzeba będzie użyć do wykonania danej czynności.

SUBSTRING_INDEX()

SELECT

1. Przejrzymy dane zapisane w kolumnie, by określić wzorzec, według którego są one zapisywane.

LEFT

ADD COLUMN

2. Dodamy do tabeli dwie nowe, puste kolumny.

ADJUST

RIGHT

3. Pobierzemy fragment danych z kolumny tekstowej.

ALTER TABLE

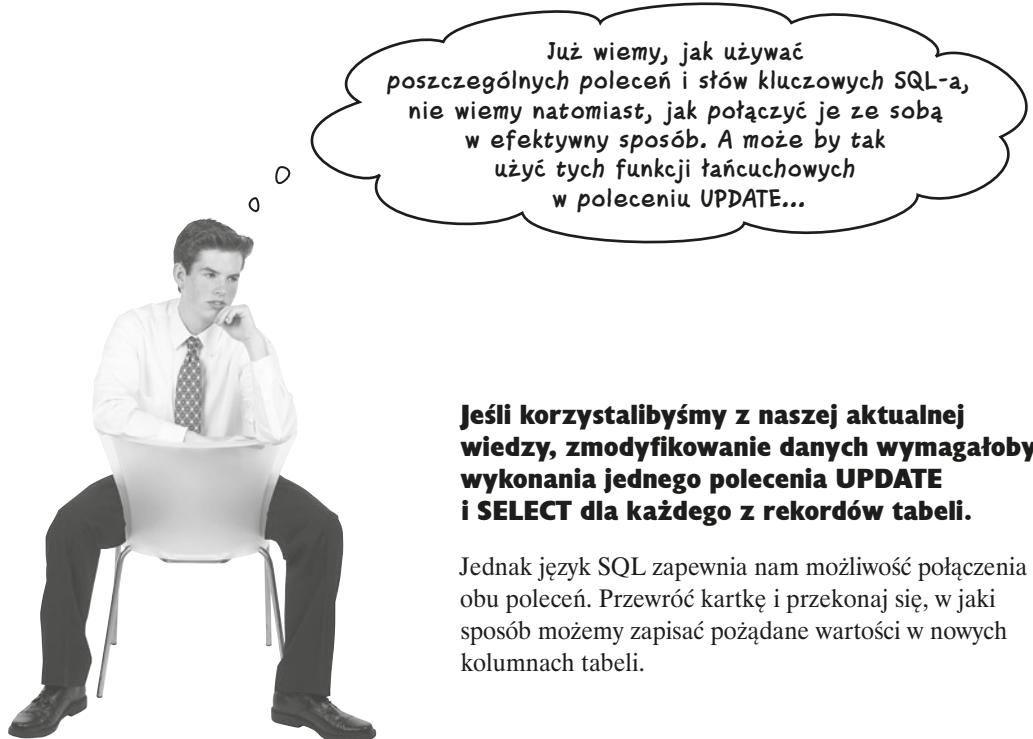
DELETE

4. Zapiszemy dane pobrane w poprzednim kroku do jednej z nowych kolumn.

UPDATE

INSERT

→ Rozwiążanie znajdziesz na stronie 262



JAKIE JEST MOJE PRZEZNACZENIE? ROZWIĄZANIE

Próbowiemy pobrać dane z kolumny lokalizacja i przenieść je do dwóch nowych kolumn: miejscowości i województwo.

Poniżej opisaliśmy czynności, które w tym celu planujemy wykonać. Dopasuj każdą z nich ze słowem lub słowami kluczowymi SQL, których trzeba będzie użyć do wykonania danej czynności.

SUBSTRING_INDEX()

ADD COLUMN

RIGHT

DELETE

UPDATE

1. Przejrzymy dane zapisane w kolumnie, by określić wzorzec, według którego są one zapisywane.

2. Dodamy do tabeli dwie nowe, puste kolumny.

3. Pobierzemy fragment danych z kolumny tekstowej.

4. Zapiszemy dane pobrane w poprzednim kroku do jednej z nowych kolumn.

SELECT

LEFT

ADJUST

ALTER TABLE

INSERT

Użyj bieżącej kolumny do zapisania wartości w innej kolumnie

Czy pamiętasz jeszcze składnię polecenia UPDATE? Możemy go użyć do zapisania pewnej nowej wartości we wszystkich wierszach tabeli. Przykład podany poniżej pokazuje składnię polecenia pozwalającą na **zmianę wartości kolumny we wszystkich wierszach**. W poleceniu zamiast „nowej_wartości” możesz zapisać wartość lub nazwę innej kolumny.

```
UPDATE nazwa_tabeli
SET nazwa_kolumny = nowa_wartość;
```

Podana nowa wartość zostanie za jednym razem zapisana we wszystkich wierszach tabeli.

Aby zapisać dane w naszych nowych kolumnach `miejscowosc` oraz `wojewodztwo`, możemy umieścić wywołanie funkcji `RIGHT()` w poleceniu UPDATE. Funkcja pobierze dwie ostatnie litery z dotychczasowej kolumny `lokalizacja` i zapisze je w kolumnie `wojewodztwo`.

UPDATE moje_kontakty

```
SET wojewodztwo = RIGHT(lokalizacja, 2);
```

To nowa kolumna, w której umieścimy kody województw.

A to funkcja tańcuchowa, która pobiera dwie ostatnie litery z kolumny `lokalizacja`.



Ale jak takie polecenie działa? Nawet nie ma w nim żadnej klauzuli WHERE, która określaby, co należy zmienić.

To polecenie może działać bez żadnej klauzuli WHERE. Przewróć kartkę, a dowiesz się, jak ono działa.

Sposób działania połączonych poleceń UPDATE i SET

Oprogramowanie zarządzające bazą danych wykonuje polecenie kolejno dla wszystkich wierszy tabeli — kiedy zapisze jeden wiersz, przechodzi do następnego i tak aż do momentu, gdy wszystkie dwuliterowe kody województw zostaną wydzielone i zapisane w kolumnie województwo.

moje_kontakty			
id_kontaktu	lokalizacja	miejscowosc	województwo
1	Gliwice, SL		
2	Kraków, MP		
3	Wrocław, DS		

Oto uproszczona wersja tabeli kontaktów Grześka.

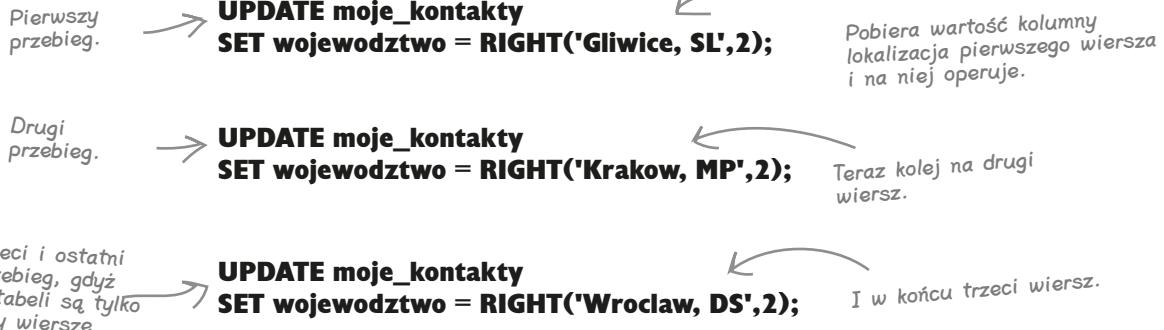
**UPDATE moje_kontakty
SET województwo = RIGHT(lokalizacja, 2);**

A teraz wypróbujmy działanie tego polecenia na naszej przykładowej tabeli. Podczas pierwszego przebiegu pobrana jest wartość kolumny lokalizacja pierwszego wiersza i na niej wykonywane są zadane operacje.

Następnie wykonywany jest drugi przebieg, a w jego ramach zostaje odszukany drugi wiersz tabeli — także w nim zostaje odnaleziona kolumna lokalizacja i wykonane analogiczne operacje co wcześniej. I tak dalej, aż do przetworzenia wszystkich wierszy tabeli.

A to jest nasze polecenie SQL.

Funkcji łańcuchowych można używać w połączeniu z poleceniami **SELECT**, **UPDATE** oraz **DELETE** języka SQL.



Zaostrz ołówek

Rozwiążanie
ze strony 244

Naszkicuj strukturę tabeli po wykonaniu polecenia SQL ze strony 244.

lista_projektow

id_proj	infooproj	nazwfirmwyk
1	malowanie elewacji domu	SUP-Rem
2	instalacja nowej kuchni	RemBudSzpachelka
3	położenie drewnianej podłogi	Kem-Rem
4	przekładka dachu	Jackowski i Syn

Dotychczasowej kolumnie "numer" zmieniliśmy nazwę na **id_proj** i zażądaliśmy, by jej wartości były automatycznie inkrementowane. Co więcej, wskazaliśmy tę kolumnę jako kolumnę klucza głównego.



Przybornik SQL

Możesz sobie pogratulować. Właśnie opanowałeś materiał piątego rozdziału książki i wzbogaciłeś swój SQL-owy przybornik o znajomość polecenia ALTER.

Kompletną listę porad znajdziesz w dodatku C, zamieszczonym na końcu książki.

ALTER TABLE

To polecenie pozwala zmieniać nazwę tabeli oraz jej strukturę bez konieczności usuwania jej zawartości.

ALTER z klauzulą ADD

Pozwala dodawać kolumny do tabeli i określać ich położenie.

ALTER z klauzulą DROP

Pozwala usunąć kolumnę z tabeli.

ALTER z klauzulą CHANGE

Pozwala zmienić zarówno nazwę, jak i typ danych istniejącej kolumny tabeli.

ALTER z klauzulą MODIFY

Pozwala zmienić jedynie typ danych istniejącej kolumny.

FUNKCJE ŁAŃCUCHOWE

Pozwalają modyfikować kopie zawartości kolumn tekstowych zwracanych jako wyniki zapytania. Oryginalne wartości pozostają niezmienione.

6. Zaawansowane zastosowanie polecenia SELECT

Nowy sposób spojrzenia na dane



Nadszedł czas, byś dodał do swojego SQL-owego przybornika nieco finezji.

Już wiesz, jak pobierać dane, używając polecenia SELECT i klauzuli WHERE. Jednak czasami będziesz potrzebował nieco większej **precyzji** niż ta, jaką one zapewniają. W tym rozdziale dowiesz się o **określaniu kolejności oraz grupowaniu** danych, jak również o **wykonywaniu operacji matematycznych** na wynikach zapytań.

Reorganizacja Filmoteki Bazodanowa

Właściciel Filmoteki Bazodanowa nie potrafił sensownie zorganizować swojego sklepu. W aktualnie używanym systemie filmy mogą znaleźć się na różnych półkach, zależnie od tego, który z pracowników je układał. Jednak właściciel właśnie zamówił nowe półki i doszedł do wniosku, że w końcu nadszedł czas, by prawidłowo oznakować kategorie filmów.

FILMOTeka BAZODANOWA



W obecnym systemie do określania typów filmów wykorzystywane są wartości logiczne — Prawda i Fałsz. Takie rozwiązanie utrudnia kategoryzację filmów. Na przykład: gdzie należałoby umieścić film, który ma zapisaną wartość P w kolumnach komedia i s-f?

Do: Pracowników wypożyczalni
Filmoteka Bazodanowa
Od: Szefa
Temat: Nowe półki oznaczają nowe kategorie

Cześć ludziska

Właśnie przywieźli nowe półki, więc chciałbym, żebyście uporządkowali nasze filmy. Myślę, żeby zastosować następujący podział na kategorie:

Akcji i przygodowe
Dramat
Komedia
Rodzinne
Horror
S-f i fantastyczne
Inne

Sami musicie się zorientować, jak wykorzystać bieżącą tabelę, by ułatwić sobie określanie kategorii filmów.

A teraz idę na lunch.

Wasz zapracowany szef.

„P” i „F” to skrótowe oznaczenia wartości „tak” i „nie”.

tabela_filmow

W tej kolumnie zapisano informację o tym, kiedy wypożyczalnia kupiła kopię filmu.

id_filmu	tytuł	ocena	dramat	komedia	akcja	krwawy	s-f	dla_dzieci	rysunkowy	zakupiono
1	Potwory i Sp.	D	F	P	F	F	F	P	P	6.3.2002
2	Ojciec chrzestny	R	F	F	P	P	F	F	F	5.2.2001
3	Przeminęło z wiatrem	D	P	F	F	F	F	F	F	20.11.1999
4	American Pie	R	F	P	F	F	F	F	F	19.4.2003
5	Koszmar z ulicy Wiązów	R	F	F	P	P	F	F	F	19.4.2003
6	Casablanca	SD	F	F	F	F	F	F	F	5.2.2001

Wszystkie te kolumny mają za zadanie utatwić nam podanie klientowi informacji na temat filmu.

Kilka problemów z bieżącą tabelą

Poniżej zamieściliśmy podsumowanie najważniejszych problemów, jakich przysparza bieżąca tabela używana w wypożyczalni Filmoteka Bazodanowa.

Kiedy klienci oddają filmy, nie wiadomo, w jakiej kategorii należy je umieścić.

Jeśli film ma wartości „P” w kilku kolumnach, to nie za bardzo wiadomo, na półce której kategorii należy go umieścić. Filmy zawsze powinny być przypisane tylko do jednej kategorii.

Klienci nie mają pewności co do tematyki filmów.

Klienci nie za bardzo wiedzą, co myśleć o filmie „dla dorosłych” znalezionym na półce kategorii „komedia”. Obecnie podczas rozmieszczania filmów na półkach nie ma kategorii o wyższym lub niższym priorytecie.

Dodawanie wartości Prawda i Fałsz jest czasochłonne i sprzyja popełnianiu błędów.

Za każdym razem, gdy szef kupi nowy film, trzeba zapisać odpowiednie wartości w kolumnach określających przynależność filmu do kategorii. A im więcej ich będzie, tym więcej błędów wkradnie się do danych. Czasami w kolumnie, w której powinna być zapisana wartość P, zostanie umieszczona wartość F, lub na odwrót. Jakaś kolumna kategorii mogłaby nam pomóc w sprawdzeniu, czy te wszystkie wartości P i F zostały zapisane prawidłowo, a w przyszłości może pozwoliłaby całkowicie z nich zrezygnować.

Jak widać, potrzebna nam będzie kolumna określająca kategorię filmu, która pomogłaby pracownikom wypożyczalni określać, na jakiej półce film należy umieścić, a klientom — zorientować się w jego tematyce. Oprócz tego taka kolumna mogłaby nam pomóc ograniczyć ilość błędów w informacjach zapisanych w tabeli.



WYSIL
SZARE KOMÓRKI

W jaki sposób przeorganizowałbyś aktualne kolumny tabeli, by utworzyć nowe **kategorie**? Czy w tabeli są jakieś filmy, które należałoby przypisać do więcej niż jednej kategorii?

Dopasowywanie istniejących danych

Już wiesz, jak zastosować polecenie ALTER, by dodać do tabeli kolumnę kategorii; nieco trudniejsze będzie zapisanie w niej informacji o tym, do której kategorii należy konkretny film. Na szczęście aktualne informacje zapisane w tabeli mogą pomóc w określeniu kategorii filmu bez konieczności ręcznego przeglądania każdego z nich.

Spróbujmy napisać zdania określające zależności między danymi:

Jeśli kolumna **dramat** zawiera wartość 'P', to w kolumnie kategoria zapiszemy 'dramat'.

Jeśli kolumna **komedia** zawiera wartość 'P', to w kolumnie kategoria zapiszemy 'komedia'.

Jeśli kolumna **akcja** zawiera wartość 'P', to w kolumnie kategoria zapiszemy 'akcji'.

Jeśli kolumna **krwawy** zawiera wartość 'P', to w kolumnie kategoria zapiszemy 'horror'.

Jeśli kolumna **sf** zawiera wartość 'P', to w kolumnie kategoria zapiszemy 's-f'.

Jeśli kolumna **dla_dzieci** zawiera wartość 'P', to w kolumnie kategoria zapiszemy 'rodzinny'.

Jeśli kolumna **rysunkowy** zawiera wartość 'P', i kolumna **ocena** zawiera wartość 'D', to w kolumnie kategoria zapiszemy 'rodzinny'.

Jeśli kolumna **rysunkowy** zawiera wartość 'P', i kolumna **ocena** NIE zawiera wartości 'D', to w kolumnie kategoria zapiszemy 'innego'.

Nie wszystkie kreskówki są przeznaczone dla dzieci. Kolumna "ocena" pomoże nam określić, czy dany film można zaliczyć do kategorii filmów rodzinnych, czy też nie. Jeśli w kolumnie tej zapisano wartość 'D', to film możemy zaliczyć do rodzinnych; w przeciwnym razie zaliczymy go do kategorii 'innego'.

Określanie zawartości nowej kolumny

A teraz musimy przekształcić zdania z poprzedniej strony na polecenia UPDATE:

```
UPDATE tabela_filmow SET kategoria = 'dramat' WHERE dramat = 'P';
UPDATE tabela_filmow SET kategoria = 'komedia' WHERE komedia = 'P';
UPDATE tabela_filmow SET kategoria = 'akcji' WHERE akcja = 'P';
UPDATE tabela_filmow SET kategoria = 'horror' WHERE krwawy = 'P';
UPDATE tabela_filmow SET kategoria = 's-f' WHERE sf = 'P';
UPDATE tabela_filmow SET kategoria = 'rodzinny' WHERE dla_dzieci = 'P';
UPDATE tabela_filmow SET kategoria = 'rodzinny' WHERE rysunkowy = 'P' AND ocena = 'D';
UPDATE tabela_filmow SET kategoria = 'inny' WHERE rysunkowy = 'P' AND ocena <> 'D';
```

↑
Ocena jest różna od 'D'.

Zaostrz ołówek



Przypisz poszczególnym filmom kategorię, do jakiej powinny należeć.

tabela_filmow

tytuł	ocena	dramat	komedia	akcja	krwawy	sf	dla_dzieci	rysunkowy	kategoria
Wielka przygoda	D	F	F	F	F	F	P	F	
Grześ: Historia nieznana	SD	F	F	P	F	F	F	F	
Szaleni kлоwni	R	F	F	F	P	F	F	F	
Paraskavedekatriaphobia	R	P	P	P	F	P	F	F	
Szczurek o imieniu Darek	D	F	F	F	F	F	P	F	
Koniec linii	R	P	F	F	P	P	F	P	
Błyskotki	SD	P	F	F	F	F	F	F	
Zwrot	R	F	P	F	F	F	F	F	
Przynęta na rekiny	D	F	F	F	F	F	P	F	
Wkurzony pirat	SD	F	P	F	F	F	F	P	
Planeta do zamieszkania	SD	F	P	F	F	F	F	F	

Czy kolejność przetwarzania poszczególnych kolumn logicznych ma znaczenie?

Rozwiążanie ćwiczenia

Zaostrz ołówek



Rozwiążanie

Przypisz poszczególnym filmom kategorię, do jakiej powinny należeć.

tabela_filmow

tytuł	ocena	dramat	komedia	akcja	krwawy	sf	dla_dzieci	rysunkowy	kategoria
Wielka przygoda	D	F	F	F	F	F	P	F	rodzinny
Grześ: Historia nieznana	SD	F	F	P	F	F	F	F	akcji
Szaleni klowni	R	F	F	F	P	F	F	F	horror
Paraskavedekatriaphobia	R	P	P	P	F	P	F	F	?
Szczurek o imieniu Darek	D	F	F	F	F	F	P	F	rodzinny
Koniec linii	R	P	F	F	P	P	F	P	?
Błyskotki	SD	P	F	F	F	F	F	F	dramat
Zwrot	R	F	P	F	F	F	F	F	komedia
Przynęta na rekiny	D	F	F	F	F	F	P	F	rodzinny
Wkurzony pirat	SD	F	P	F	F	F	F	P	innny
Planeta do zamieszkania	SD	F	P	F	F	F	F	F	komedia

Znak zapytania oznacza, że wartość kolumny została zmieniona przez więcej niż jedno polecenie UPDATE. Ta wartość może się zatem zmieniać w zależności od kolejności, w jakiej zostaną wykonane polecenia UPDATE.



Czy kolejność przetwarzania poszczególnych kolumn logicznych ma znaczenie? Owszem, ma znaczenie.

Kolejność ma znaczenie

Na przykład gdybyśmy analizowali kolumny w kolejności, w jakiej są zapisane, to film „Paraskavedekatriaphobia” zostałby zakwalifikowany jako film fantastycznonaukowy, choć raczej można by sądzić, że jest to komedia. Nie wiem jednak na pewno, czy powinien on być uważany za komedię, film akcji, dramat, film rysunkowy czy też film fantastycznonaukowy. Ponieważ nie mamy pewności, co z nim zrobić, najlepszym rozwiążaniem będzie umieszczenie go w kategorii „innny”.

**Kolejność
ma znaczenie.**

**Dwa polecenia
UPDATE mogą
bowiem zmieniać
wartość tej
samej kolumny.**



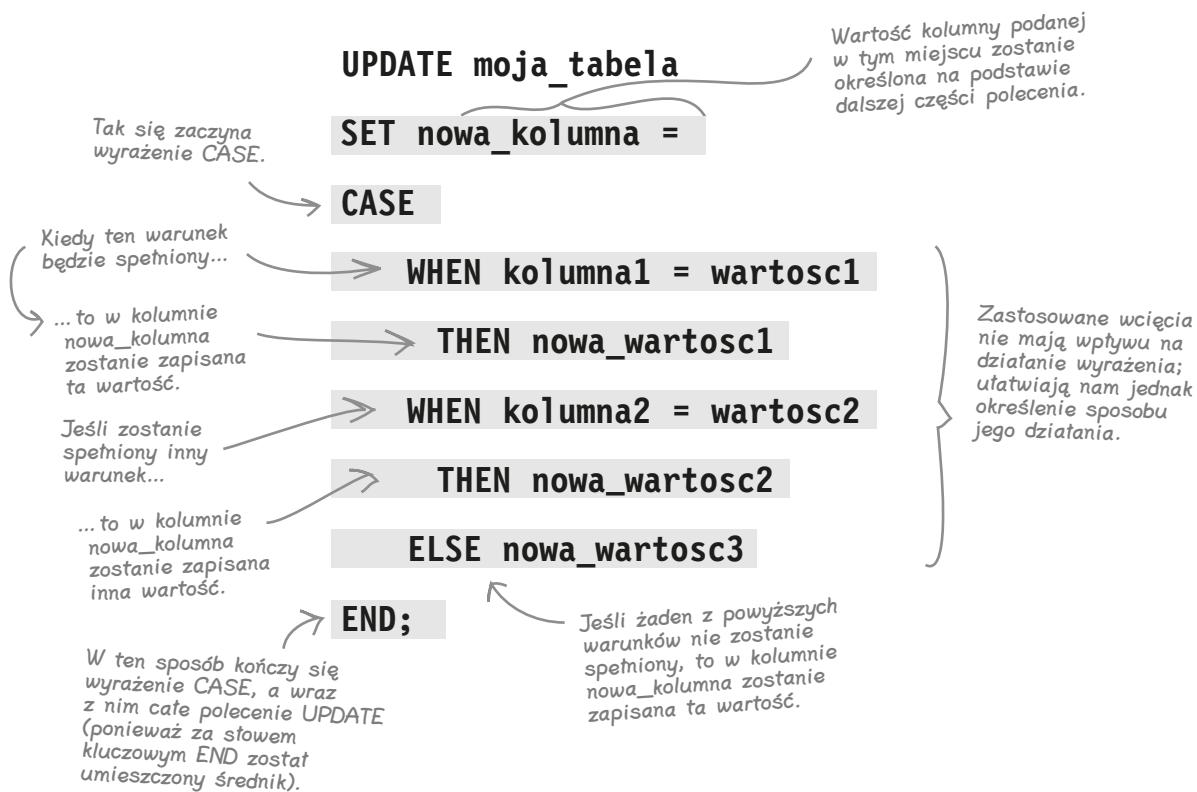
Takie rozwiązanie może być dobre dla niewielkich tabel, a co jeśli tabela będzie mieć setki kolumn? Czy istnieje jakiś sposób pozwalający na połączenie tych wszystkich pojedynczych poleceń UPDATE w jedno duże?

Owszem, można by napisać jedno duże polecenie UPDATE, ale istnieje jeszcze lepsze rozwiązanie.

Wyrażenie CASE zapewnia możliwość połączenia wszystkich poleceń UPDATE — pozwala bowiem określić sekwencję warunków, w których będzie testowana wartość kolumny. Jeśli okaże się, że któryś z warunków jest spełniony, to w kolumnie zostanie zapisana nowa, podana wartość.

Wyrażenie to pozwala nawet określić, *co ma się stać, jeśli żaden z warunków nie zostanie spełniony*.

Poniżej przedstawiliśmy przykład ilustrujący prostą postać polecenia UPDATE z wyrażeniem CASE:



Polecenie UPDATE z wyrażeniem CASE

Zobaczmy, jak wygląda wyrażenie CASE w akcji — zastosujemy je do określania wartości kolumny kategoria w naszej tabeli tabela_filmow.

```
UPDATE tabela_filmow
SET kategoria =
CASE
    WHEN dramat = 'P' THEN 'dramat'
    WHEN komedia = 'P' THEN 'komedia'
    WHEN akcja = 'P' THEN 'akcji'
    WHEN krwawy = 'P' THEN 'horror'
    WHEN sf = 'P' THEN 's-f'
    WHEN dla_dzieci = 'P' THEN 'rodzinny'
    WHEN rysunkowy = 'P' THEN 'rodzinny'
    ELSE 'inny'
END;
```

↑
Jeśli żaden z powyższych warunków nie zostanie spełniony, to w kolumnie kategoria zostanie zapisana wartość 'inny'.

Ten wiersz odpowiada poleceniu
UPDATE tabel_filmow SET kategoria
= 'drama' WHERE dramat = 'T'. Ale
zobacz, o ile mniej musisz napisać!

tabela_filmow

Wartości, które wcześniej, gdy określaliśmy wartość kolumny kategoria przy użyciu niezależnych poleceń UPDATE, były nieznane, teraz zostały określone.

Warto jednak zwrócić uwagę także na fakt, iż wartości kategorii dla filmów „Wkurzony pirat” oraz „Koniec linii” uległy zmianie.

tytuł	ocena	dramat	komedia	akcja	krwawy	sf	dla_dzieci	rysunkowy	kategoria
Wielka przygoda	D	F	F	F	F	F	F	P	rodzinny
Grześ: Historia nieznana	SD	F	F	P	F	F	F	F	akcja
Szaleni kłowni	R	F	F	F	P	F	F	F	horror
Paraskavedekatriaphobia	R	P	P	P	F	P	F	F	dramat
Szczurek o imieniu Darek	D	F	F	F	F	F	P	F	rodzinny
Koniec linii	R	P	F	F	P	P	F	P	dramat
Błyskotki	SD	P	F	F	F	F	F	F	dramat
Zwrot	R	F	P	F	F	F	F	F	komedia
Przynęta na rekiny	D	F	F	F	F	F	P	F	rodzinny
Wkurzony pirat	SD	F	P	F	F	F	F	P	komedia
Planeta do zamieszkania	SD	F	P	F	F	F	F	F	komedia

Podczas przetwarzania w wyrażeniu CASE wartości 'P' i 'F' dla każdego z filmów, system zarządzania bazą danych poszukuje pierwszej wartości 'P', na podstawie której mógłby określić wartość kolumny kategorii.

Oto co się będzie działo podczas przetwarzania rekordu filmu „Wielka przygoda”:

```
UPDATE tabela_filmow
```

```
SET kategoria =
```

```
CASE
```

```
  WHEN dramat = 'P' THEN 'dramat'
```

Niespełnione: nie określono kategorii

```
  WHEN komedia = 'P' THEN 'komedia'
```

Niespełnione: nie określono kategorii

```
  WHEN akcja = 'P' THEN 'akcji'
```

Niespełnione: nie określono kategorii

```
  WHEN krwawy = 'P' THEN 'horror'
```

Niespełnione: nie określono kategorii

```
  WHEN sf = 'P' THEN 's-f'
```

Niespełnione: nie określono kategorii

```
  WHEN dla_dzieci = 'P' THEN 'rodzinny'
```

Niespełnione: nie określono kategorii

```
  WHEN rysunkowy = 'P' THEN 'rodzinny'
```

Niespełnione: nie określono kategorii

```
  ELSE 'inny'
```

```
END;
```

Spelnione: W kolumnie kategorii jest zapisywana wartość 'rodzinny', po czym przechodzimy do słowa kluczowego END i kończymy przetwarzanie tego wiersza tabeli.

A teraz sprawdźmy, co się stanie w wierszu, w którym może być spełnionych więcej warunków. Także w tym przypadku poszukujemy pierwszej wartości 'P', na podstawie której możemy określić zawartość kolumny kategorii.

Oto co się stanie w momencie przetwarzania rekordu filmu „Paraskavedekatriaphobia”:

```
UPDATE tabela_filmow
```

```
SET kategoria =
```

```
CASE
```

```
  WHEN dramat = 'P' THEN 'dramat'
```

Warunek spełniony: W kolumnie kategorii jest zapisywana wartość 'dramat', po czym przechodzimy do słowa kluczowego END i kończymy przetwarzanie tego wiersza tabeli. Wszystkie pozostałe wartości 'P' oraz warunki wyrażenia CASE są ignorowane.

```
  WHEN comedy = 'P' THEN 'komedia'
```

```
  WHEN akcja = 'P' THEN 'akcji'
```

```
  WHEN krwawy = 'P' THEN 'horror'
```

```
  WHEN sf = 'P' THEN 's-f'
```

```
  WHEN dla_dzieci = 'P' THEN 'rodzinny'
```

```
  WHEN rysunkowy = 'P' THEN 'rodzinny'
```

```
  ELSE 'inny'
```

```
END;
```

Wygląda na to, że mamy problem

Możemy mieć pewien problem. Film „Wielka przygoda” to film rysunkowy, ale dla dorosłych. Jednak jakimś trafem film ten został zakwalifikowany jako „rodzinny”.

WIADOMOŚĆ

Data: **dziś** Godzina: **13:41**

Do: **szefa**

KIEDY SZEFA NIE BYŁO

naprawdę wściekła klientka

Zadzwonił	<input checked="" type="checkbox"/>	Prosił o telefon	<input checked="" type="checkbox"/>
Zadzwonił, żeby się spotkać		Zadzwoni później	
Chciał się z szefem zobaczyć		Odpowiedział na telefon szefa	

TREŚĆ WIADOMOŚCI: Jakaś pani zadzwoniła ze skarga, że jej mały synek Alanek obejrzał kreskówkę dla dorosłych, pełną wulgaryzmów, a teraz ciągle goni swoją siostrzyczkę, wołając na nią: %\$#@\$@*^\$

PRZYJĄŁ: **ja** PILNE

Zaostrz ołówek



Zmodyfikuj wyrażenie CASE w taki sposób, by filmy rysunkowe trafiały do kategorii 'inny', a nie 'rodzinny'.



WYSIL SZARE KOMÓRKI

W jaki sposób moglibyśmy wykorzystać literę 'R' w ocenie filmu, by w przyszłości uchronić się przed takimi przykrymi niespodziankami?

Zaostrz ołówek

Rozwiążanie



Zmodyfikuj wyrażenie CASE w taki sposób, by filmy rysunkowe trafiały do kategorii 'inny', a nie 'rodzinny'.

```
UPDATE tabela_filmow
SET kategoria =
CASE
    WHEN dramat = 'P' THEN 'dramat'
    WHEN komedia = 'P' THEN 'komedia'
    WHEN akcja = 'P' THEN 'akcji'
    WHEN krwawy = 'P' THEN 'horror'
    WHEN sf = 'P' THEN 's-f'
    WHEN dla_dzieci = 'P' THEN 'rodzinny'
    WHEN rysunkowy = 'P' AND ocena = 'R' THEN 'rodzinny'
    ELSE 'inny'
END;
```

W wyrażeniu CASE można umieszczać złożone warunki: wystarczy dodać operator AND, by sprawdzać, czy film należy do odpowiedniej kategorii I w jego ocenie umieszczono literkę 'R'. Jeśli oba te warunki zostaną spełnione, to film możemy przypisać do kategorii „rodzinny”.



Nie, istnieją grupy pytania

P: Czy koniecznie muszę używać klauzuli ELSE?

O: Nie, klauzula ELSE jest opcjonalna. Jeśli jej nie potrzebujesz, możesz po prostu pominąć ten wiersz kodu. Niemniej jednak fajnie jest móc określić jakąś wartość kolumny, kiedy żadne inne nie pasują. W końcu lepiej jest zapisać w kolumnie jakąś wartość, niż pozwolić, by znalazła się w niej wartość NULL.

P: A co się stanie, jeśli nie zastosuję klauzuli ELSE, a żaden z warunków określonych w wyrażeniu CASE nie zostanie spełniony?

O: W takim przypadku wartość kolumny nie ulegnie zmianie.

P: A co zrobić w sytuacji, gdybym chciał użyć wyrażenia CASE tylko w wybranych, a nie we wszystkich wierszach? Na przykład gdybym chciał użyć go tylko gdy kategoria = 'inny'. Czy mógłbym w tym celu użyć klauzuli WHERE?

O: Owszem, mógłbyś umieścić w poleceniu klauzulę WHERE, zapisując ją za słowem kluczowym END.

P: Czy wyrażenia CASE można używać także w innych poleceniach SQL, czy tylko w poleceniu UPDATE?

O: Owszem, można. Wyrażenia CASE można stosować także w poleceniach SELECT, INSERT oraz DELETE.

KONSTRUKCJA WYRAŻEŃ CASE. ROZWIĄZANIE

Twój szef, który zawsze lubi się do wszystkiego wtrącać, zdecydował, że trzeba jeszcze nieco zmodyfikować kategorie. Przeczytaj przesłanego przez niego e-maila i napisz pojedyncze polecenie SQL, które zaspokoi jego wymagania.

```
UPDATE tabela_filmow
SET kategoria =
CASE
    WHEN dramat = 'P' AND ocena = 'R' THEN 'dramat-r'
    WHEN comedy = 'P' AND ocena = 'R' THEN 'komedia-r'
    WHEN akcji = 'P' AND ocena = 'R' THEN 'akcji-r'
    WHEN krwawy = 'P' AND ocena = 'R' THEN 'horror-r'
    WHEN sf = 'P' AND ocena = 'R' THEN 's-f-r'
    WHEN kategoria = 'inný' AND ocena = 'D' THEN 'rodzinny'
END;
```

Okazuje się, że nowe kategorie utrudniają klientom odnajdywanie filmów. Napisz polecenie, które usunie z tabeli nowe „r-kategorie”.

```
UPDATE tabela_filmow
SET kategoria =
CASE
    WHEN kategoria = 'dramat-r' THEN kategoria = 'dramat'
    WHEN kategoria = 'komedia-r' THEN kategoria = 'komedia'
    WHEN kategoria = 'akcji-r' THEN kategoria = 'akcji'
    WHEN kategoria = 'horror-r' THEN kategoria = 'horror'
    WHEN kategoria = 's-f-r' THEN kategoria = 's-f'
END;
```

I w końcu usuń wszystkie kolumny z wartościami P i F — nie będziemy bowiem ich już więcej potrzebowali.

```
ALTER TABLE tabela_filmow
DROP COLUMN dramat,
DROP COLUMN komedia,
DROP COLUMN akcja,
DROP COLUMN krwawy,
DROP COLUMN sf,
DROP COLUMN dla_dzieci,
DROP COLUMN rysunkowy;
```

Do: Pracowników wypożyczalni
WideoBazaNowa
Od: Szefa
Temat: Nowe działy mogą być kategoriami!

Moi kochani wideomaniacy
Zdecydowałem się stworzyć kilka nowych
działów. Sądzę, że filmy z oznaczeniem „R”
należałyby umieszczać w innym miejscu niż filmy
z oznaczeniami „D” i „SD”. A zatem stworzymy pięć
nowych kategorii:

horror-r
akcji-r
dramat-r
komedia-r
s-f-r

Poza tym, jeśli znajdziecie jakiś film z oceną „D”
należący do kategorii „inný”, to przenieście go
do kategorii „rodzinny”.

Dzięki, będę wdzięczny.
Wasz szef.

Do tabel może się wkrąć bałagan

Kiedy do wypożyczalni dotrze nowy film, jest dodawany do tabeli i staje się jej najnowszym wierszem. Jednak filmy zapisywane w naszej tabeli nie są w żaden sposób uporządkowane. Dlatego gdy mamy od nowa rozmieścić filmy na półkach, pojawią się problemy. Wiadomo, że na każdej z nowych półek można umieścić 20 filmów oraz że na każdym z ponad 3000 filmów należy umieścić nalepkę z określeniem kategorii. *Musimy zatem pobrać wszystkie filmy należące do poszczególnych kategorii i posortować je alfabetycznie w ramach każdej z kategorii.*

Wiemy, jak pobrać z tabeli wszystkie filmy należące do każdej z kategorii, jednak teraz musimy jeszcze posortować je alfabetycznie.

tablica_filmow

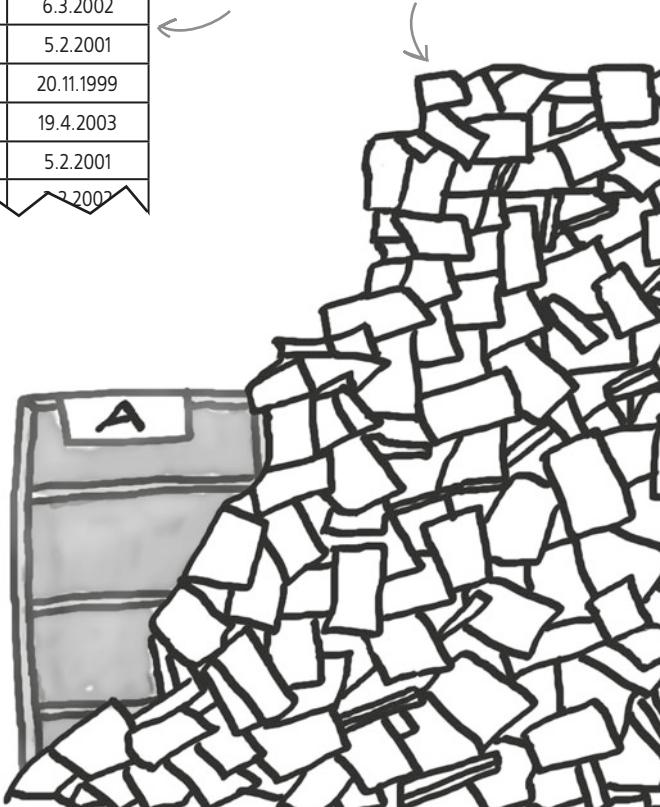
id_filmu	tytuł	ocena	kategoria	zakupiono
83	Wielka przygoda	D	rodzinny	6.3.2002
84	Grześ: Historia nieznana	SD	akcji	5.2.2001
85	Szaleni kлоwni	R	horror	20.11.1999
86	Paraskavedekatriaphobia	R	akcji	19.4.2003
87	Szczurek o imieniu Darek	D	rodzinny	12.4.2003
88	Koniec linii	R	inny	5.2.2001
89	Błyskotki	SD	dramat	6.3.2002
90	Zwrot	R	komedia	5.2.2001
91	Przynęta na rekiny	D	inny	20.11.1999
92	Wkurzony pirat	SD	inny	19.4.2003
93	Planeta do zamieszkania	SD	s-f	5.2.2001
94	Wielki Gatsby	R	dramat	2002

To jedynie kilkanaście spośród ponad 3000 filmów, jakie posiada Filmateka Bazodanowa.



**WYTĘŻ
UMYSŁ**

Jak uporządkować te dane alfabetycznie za pomocą polecenia SQL.



Przeciążenie polecenia SELECT

Potrzebujemy możliwości organizowania danych zwracanych przez polecenie SELECT

Na każdym z ponad 3000 filmów, jakie posiada Filmoteka Bazodanowa, trzeba nalepić karteczkę z informacją o kategorii. Następnie każdy z nich należy umieścić na odpowiedniej półce i to w kolejności alfabetycznej.

Potrzeba nam zatem głównej listy wszystkich filmów w danej kategorii, posortowanej alfabetycznie. Jak na razie wiemy, jak pobierać informacje z tabeli, posługując się poleceniem SELECT. Bez trudu możemy pobrać filmy należące do określonej kategorii; możemy nawet pobrać z wybranej kategorii filmy, których tytuły zaczynają się na podaną literę.

Jednak w takim przypadku, aby stworzyć naszą główną listę filmów, musielibyśmy napisać około 245 poleceń SELECT. Poniżej przedstawiliśmy jedynie kilka z nich:

```
SELECT tytul, kategoria FROM tabela_filmow WHERE tytul LIKE 'A%' AND kategoria = 'rodzinny';
SELECT tytul, kategoria FROM tabela_filmow WHERE tytul LIKE 'B%' AND kategoria = 'rodzinny';
SELECT tytul, kategoria FROM tabela_filmow WHERE tytul LIKE 'C%' AND kategoria = 'rodzinny';
SELECT tytul, kategoria FROM tabela_filmow WHERE tytul LIKE 'D%' AND kategoria = 'rodzinny';
SELECT tytul, kategoria FROM tabela_filmow WHERE tytul LIKE 'E%' AND kategoria = 'rodzinny';
SELECT tytul, kategoria FROM tabela_filmow WHERE tytul LIKE 'F%' AND kategoria = 'rodzinny';
SELECT tytul, kategoria FROM tabela_filmow WHERE tytul LIKE 'G%' AND kategoria = 'rodzinny';
```

Tytuł jest nam potrzebny, byśmy mogli przejrzeć stosiki filmów i znaleźć odpowiedni; z kolei na podstawie kategorii odpowiednią nalepkę nalepimy na filmie odpowiednią, a następnie umieścimy go na właściwej półce.

To jest litera alfabetu, na jaką ma się zaczynać tytuł filmu.

A to kategoria, która nas interesuje.

Potrzeba nam około 245 zapytań, ponieważ mamy 7 kategorii filmów, a w polskim alfabetie jest 35 liter. Niemniej jednak trudno przypuszczać, by nazwy filmów zaczynały się od niektórych znaków diakrytycznych, takich jak „ñ”; choć z drugiej strony powyższe wyliczenie nie uwzględnia faktu, że nazwy filmów mogą się zaczynać od cyfr, np.: „101 dalmatyńczyków” lub „2001: Odyseja kosmiczna”.



WYSIL SZARE KOMÓRKI

Jak sądzisz, gdzie na tej liście pojawią się tytuły filmów, które zaczynają się od cyfry lub od innego znaku, który nie jest literą?

Zaostrz ołówek



Wciąż musimy stosować „pseudosortowanie” — ręcznie pobierać z tabeli filmy, które nie tylko należą do określonej kategorii, lecz także których tytuły zaczynają się na podaną literę alfabetu.

Przyjrzyj się dokładnie wynikom zwróconym przez jedno z naszych (co najmniej) 245 zapytań. A teraz spróbuj sam posortować tę listę alfabetycznie.

`SELECT tytul, kategoria FROM tabela_filmow WHERE tytul LIKE 'A%' AND kategoria = 'rodzinny';`

↗ *Fragment wyników zwróconych
przez powyższe zapytanie.*

tytul	kategoria
Awionetki i helikoptery	rodzinny
Aktywność umarlaków	rodzinny
A może byś mnie posłuchał?	rodzinny
Amerykański ninja	rodzinny
Anemiczny sprinter	rodzinny
Arka Noego	rodzinny
Anubis: bóg zmarłych	rodzinny
Anioły	rodzinny
Arogancki małolat	rodzinny
Atak truposzy	rodzinny
Arcyancymon	rodzinny
Amerykanin w Paryżu	rodzinny
Atak troli	rodzinny
Ancymony atakują	rodzinny
Amerykański ninja 2	rodzinny
Asterix na Olimpiadzie	rodzinny
Anielskie śpiewy	rodzinny
Aleks i złowroga wanna	rodzinny
Asterix i Kleopatra	rodzinny
A niech to...	rodzinny

Zaostrz ołówek



Rozwiążanie

Wciąż musimy stosować „pseudosortowanie” — ręcznie pobierać z tabeli filmy, które nie tylko należą do określonej kategorii, lecz także których tytuły zaczynają się na podaną literę alfabetu.

Przyjrzyj się dokładnie wynikom zwróconym przez jedno z naszych (co najmniej) 245 zapytań. A teraz spróbuj sam posortować tę listę alfabetycznie.

```
SELECT tytul, kategoria FROM tabela_filmow WHERE tytul LIKE 'A%' AND kategoria = 'rodzinny';
```

tytul	kategoria
A może być mnie posłuchała?	rodzinny
A niech to...	rodzinny
Aktywność umarlaków	rodzinny
Aleks i złowrogą wanną	rodzinny
Amerikanin w Paryżu	rodzinny
Amerkański ninja	rodzinny
Amerkański ninja 2	rodzinny
Anemongi atakują	rodzinny
Anemiczny sprinter	rodzinny
Anielskie śpiewy	rodzinny
Anioły	rodzinny
Anubis: bóg zmarłych	rodzinny
Arcyancymon	rodzinny
Arka Noego	rodzinny
Arogancki małolat	rodzinny
Asterix i Kleopatra	rodzinny
Asterix na Olimpiadzie	rodzinny
Atak trolli	rodzinny
Atak trójkoszy	rodzinny
Awionetki i helikoptery	rodzinny

Ile czasu zajęło Ci posortowanie tych wszystkich tytułów w kolejności alfabetycznej?

Czy możesz sobie wyobrazić, ile potrwałyby takie ręczne sortowanie ponad 3000 filmów?

Tytuły rozpoczęjące się od „At...” muszą się znaleźć w pobliżu końca listy, gdyż drugą literą po początkowym „A” jest „t”. Jednak zanim określmy prawidłową kolejność obu tych filmów na półce, będziemy musieli przeanalizować aż sześć znaków.

Wypróbowaj klauzulę ORDER BY

Mówisz, że musisz posortować wyniki zapytania? Cóż, tak się akurat składa, że wykonując polecenie SELECT, można zażądać, by jego wyniki zostały posortowane według wartości określonej kolumny. Służy do tego klauzula ORDER BY.

W tej części zapytania nie ma niczego, co mogłoby nas zaskoczyć. Niczym się ona nie różni do zapytania, którego używaliśmy do tej pory.

Ale to jest coś nowego.
Klauzula informuje, że wyniki mają być posortowane alfabetycznie na podstawie tytułu filmu.

```
SELECT tytuł, kategoria  
FROM tabela_filmów  
WHERE  
tytuł LIKE 'A%'  
AND  
kategoria = 'rodzinny'  
ORDER BY tytuł;
```



Powaga! Czy chcecie mi powiedzieć, że to jedyny sposób alfabetycznego posortowania wyników? NIE MA możliwości posortowania wszystkich tytułów naraz, bez dzielenia ich na grupy zaczynające się od tej samej litery?

Zaostrz ołówek



Otoż taka możliwość istnieje. Co można usunąć z powyższego polecenia, aby je poprawić i zwiększyć jego możliwości?

.....
.....
.....
.....

STÓJ! Wykonaj to ćwiczenie, zanim przejdziesz na następną stronę.

Sortowanie według jednej kolumny

Jeśli w zapytaniu wykorzystamy klauzulę ORDER BY tytuł, to nie będziemy już musieli szukać tytułów zaczynających się od konkretnej litery, gdyż zapytanie zwróci listę wszystkich filmów posortowanych alfabetycznie według zawartości kolumny tytuł.

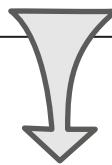
Wystarczy zatem usunąć warunek tytuł LIKE, a klauzula ORDER BY tytuł zrobi za nas wszystko co trzeba.

Zaostrz ołówek



Co można usunąć z powyższego polecenia, aby je poprawić i zwiększyć jego możliwości?

```
SELECT tytul, kategoria,
FROM tabela_filmow
WHERE
tytul LIKE 'A%
AND
kategoria = 'rodzinny'
ORDER BY tytul;
```



```
SELECT tytul, kategoria,
FROM tabela_filmow
WHERE
kategoria = 'rodzinny'
ORDER BY tytul;
```

Tym razem pobierzymy listę wszystkich filmów należących do kategorii filmów rodzinnych.

Najlepsze jest to, że lista będzie także zawierać filmy, których tytuły zaczynają się od cyfr. Zostaną one umieszczone na samym początku listy.

To jeszcze nie koniec wyników, choć my już nie mamy miejsca, by pokazać ich więcej. Lista będzie jednak zawierać wszystkie filmy z wybranej kategorii, łącznie z tymi, których tytuł zaczyna się na literę „Z”.

Klauzula ORDER BY pozwala na posortowanie wskazanej kolumny.

Zauważ, że kilka pierwszych tytułów filmów zaczyna się od cyfr.

tytul	kategoria
101 dalmatyńczyków	rodzinny
10 zakazów rodzicielskich	rodzinny
2 + 2 = 5.	rodzinny
5 muszkieterów	rodzinny
7. raz z rzędu	rodzinny
A może byś mnie posłuchał?	rodzinny
A niech to...	rodzinny
Aktywność umarłaków	rodzinny
Aleks i złowrogia wanna	rodzinny
Amerykanin w Paryżu	rodzinny
Amerykański ninja	rodzinny
Amerykański ninja 2	rodzinny
Anemony atakują	rodzinny
Anemiczny sprinter	rodzinny
Anielskie śpiewy	rodzinny
Anioły	rodzinny
Anubis: bóg zmarłych	rodzinny
Arcyancymon	rodzinny
Arka Noego	rodzinny
Arogancki małolat	rodzinny
Asterix i Kleopatra	rodzinny
Asterix na Olimpiadzie	rodzinny
Atak troli	rodzinny
Atak truposzy	rodzinny
Awionetki i helikoptery	rodzinny



Ćwiczenie

Utwórz prostą tabelę zawierającą jedną kolumnę typu CHAR(1) o nazwie `znak_testowy`.

Zapisz w niej cyfry, litery (zarówno małe, jak i wielkie), jak również inne znaki pokazane poniżej, przy czym każdy z nich umieść w osobnym wierszu. Na końcu zapisz w tabeli znak odstępu, a w jednym wierszu zostaw wartość NULL.

Kiedy to zrobisz, wypróbuj działanie poznanej właśnie klauzuli ORDER BY i uzupełnij puste miejsca w przedstawionej poniżej książce *Reguły kolejności w SQL-u*.

**0 1 2 3 A B C D a b c d ! @ # \$ % ^ & * () - _
+ = [] { } ; : ' " \ | ^ ~ , . < > / ?**

Reguły kolejności w SQL-u

Skoro już wykonałeś polecenie SELECT z klauzulą ORDER BY, uzupełnij poniższe puste miejsca, bazując na uzyskanych wynikach.

Znaki, które nie są cyframi lub literami, są wyświetlane liczbami.

Cyfry są wyświetlane tekstem.

Wartości NULL pojawiają się cyframi.

Wartości NULL pojawiają się literami.

Wielkie litery są wyświetlane małymi literami.

Tekst „A 1” zostanie wyświetlony tekstem „A1”

Reguły kolejności w SQL-u

Skoro już wykonałeś polecenie SELECT z klauzulą ORDER BY, rozmieśc poniższe znaki w kolejności, w jakiej zostały zwrócone bazując na uzyskanych wynikach.

+ = ! (& ~ " * @ ? ' ↪

Czy pamiętasz,
jak zapisać
apostrof
w kolumnie
tekstowej?
To nie jest
takie łatwe.



Rozwiążanie ćwiczenia

Utwórz prostą tabelę zawierającą jedną kolumnę typu CHAR(1) o nazwie `znak_testowy`.

Zapisz w niej cyfry, litery (zarówno małe, jak i wielkie), jak również inne znaki pokazane poniżej, przy czym każdy z nich umieść w osobnym wierszu. Na końcu zapisz w tabeli znak odstępu, a w jednym wierszu zostaw wartość NULL.

Kiedy to zrobisz, wypróbuj działanie poznanej właśnie klauzuli ORDER BY i uzupełnij puste miejsca w przedstawionej poniżej książce *Reguły kolejności w SQL-u*.

! " # \$ % & ' * () * + , - . / 0 1 2 3 : ; < = >
? @ A B C D [\] ^ _ ` a b c d { \ } ~



Kolejność, w jakiej znaki mogą się pojawiać w wynikach zapytania. Zwróc uwagę na znak odstępu umieszczony na samym początku. Warto także pamiętać, że kolejność znaków może zależeć od używanego systemu zarządzania bazami danych. W tym ćwiczeniu chodzi jednak o to, byś zapamiętał, że istnieje określona kolejność znaków oraz, że zależy ona od systemu zarządzania bazami danych.

Reguły kolejności w SQL-u

Skoro już wykonałeś polecenie SELECT z klauzulą ORDER BY, uzupełnij poniższe puste miejsca, bazując na uzyskanych wynikach.

Znaki, które nie są cyframi lub literami, są wyświetlane przed oraz za liczbami.

Cyfry są wyświetlane przed tekstem.

Wartości NULL pojawiają się przed cyframi.

Wartości NULL pojawiają się przed literami.

Wielkie litery są wyświetlane przed małymi literami.

Tekst „A 1” zostanie wyświetlony przed tekstem „A1”

Reguły kolejności w SQL-u

Skoro już wykonałeś polecenie SELCT z klauzulą ORDER BY, rozmieśc poniższe znaki w kolejności, w jakiej zostały zwrócone bazując na uzyskanych wynikach.

+ = ! (& ~ "
* @ ? '

! " & ' (* + = ? @ ~

Klauzula ORDER z dwoma kolumnami

Wydaje się, że wszystko jest pod kontrolą. Możemy wyświetlić filmy posortowane w kolejności alfabetycznej, jak również stworzyć alfabetyczną listę filmów należących do poszczególnych kategorii.

Niestety nasz szef wymyślił dla nas coś nowego...

Do: Pracownicy wypożyczalni Filmoteka Bazodanowa
Od: Szefa
Temat: Trzeba się pozbyć staroci (filmowych)

Dzień dobry!
Uważam, że powinniśmy się pozbyć filmów, które są w naszej wypożyczalni najdłużej. Czy możecie przyjść w ten weekend do pracy i dla każdej z kategorii przygotować mi listę filmów posortowanych według daty ich zakupu?
Byłoby naprawdę super,
Wasz szef.

Na szczęście wyniki zapytania można sortować w oparciu o kilka kolumn.

Chcemy mieć pewność, że data zakupu zostanie wyświetlona w wynikach zapytania.

SELECT tytuł, kategoria, zakupiono
FROM tabela_filmow
ORDER BY kategoria, zakupiono;

↑
Wartości w tej kolumnie zostaną posortowane w pierwszej kolejności. W efekcie uzyskamy listę wszystkich filmów w wypożyczalni, posortowanych w kolejności alfabetycznej według kategorii.

↑
Następnie rekordy zostaną posortowane według tej kolumny — ale dopiero po posortowaniu kolumny kategorii.



**WYŁĘŻ
UMYSŁ**

Czy najstarszy film w każdej z kategorii zostanie wyświetlony jako pierwszy, czy jako ostatni? I jak sądzasz, co się stanie, jeśli w tej samej kategorii znajdują się dwa filmy zakupione tego samego dnia? Który z nich zostanie wyświetlony jako pierwszy?

Klauzula ORDER operująca na wielu kolumnach

Nasze możliwości nie kończą się na sortowaniu wyników w oparciu o dwie kolumny. Jeśli wyświetlenie danych wynikowych w odpowiedniej postaci tego wymaga, to możesz je sortować w oparciu o dowolną ilość kolumn.

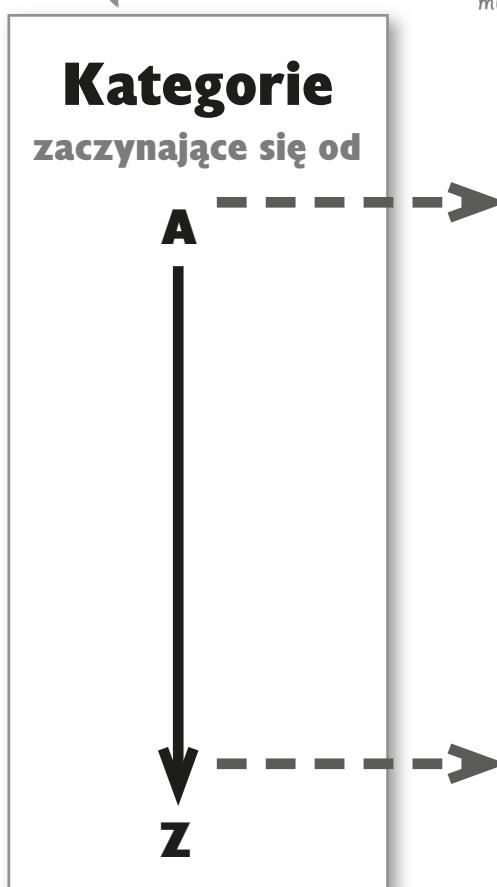
Przeanalizuj poniższe zapytanie zawierające klauzulę ORDER BY, w której zastosowano trzy kolumny.

```
SELECT * FROM tabela_filmow  
ORDER BY kategoria, zakupiono, tytul;
```

W pierwszej kolejności wyniki są sortowane według kolumny kategoria, której nazwa została podana bezpośrednio za słowem kluczowym ORDER BY. Wyniki zostaną wyświetlone w porządku alfabetycznym, od A do Z.

Następnie wyniki (czyli wszystkie kategorie w tabeli, zaczynając od litery A, bo właśnie tak kategorie są posortowane) zostają dodatkowo posortowane na podstawie daty zakupu; przy czym filmy zakupione najwcześniej zostaną wyświetlane jako pierwsze. Daty zawsze są sortowane najpierw według roku, następnie według miesiąca i w końcu według dnia.

I w końcu wyniki (każda z kategorii, zaczynając od litery „A”, które obecnie są dodatkowo posortowane według daty zakupu) zostają posortowane według tytułu filmu. Także tytuły są sortowane w kolejności alfabetycznej do A do Z.



Możesz sortować wyniki w oparciu o dowolną ilość kolumn.

Uporządkowana tabela filmów

Sprawdźmy zatem, jakie wyniki zwróci polecenie SELECT przedstawione na poprzedniej stronie, jeśli zostanie wykonane na oryginalnej tabeli tabela_filmow.

Nasza oryginalna tabela tabela_filmow.
Tak naprawdę nie ma tu żadnego sortowania — filmy są wyświetlane w takiej kolejności, w jakiej ich rekordy były zapisywane w tabeli.

id_filmu	tytuł	ocena	kategoria	zakupiono
83	Wielka przygoda	D	rodzinny	6.3.2002
84	Grześ: Historia nieznana	SD	akcji	5.2.2001
85	Szaleni kłowni	R	horror	20.11.1999
86	Paraskavedekatriaphobia	R	akcji	19.4.2003
87	Szczurek o imieniu Darek	D	rodzinny	12.4.2003
88	Koniec linii	R	inny	5.2.2001
89	Błyskotki	SD	dramat	6.3.2002
90	Zwrot	R	komedia	5.2.2001
91	Przynęta na rekiny	D	inny	20.11.1999
92	Wkurzony pirat	SD	inny	19.4.2003
93	Planeta do zamieszkania	SD	s-f	5.2.2001
94	Wielki Gatsby	R	dramat	3.2.2002

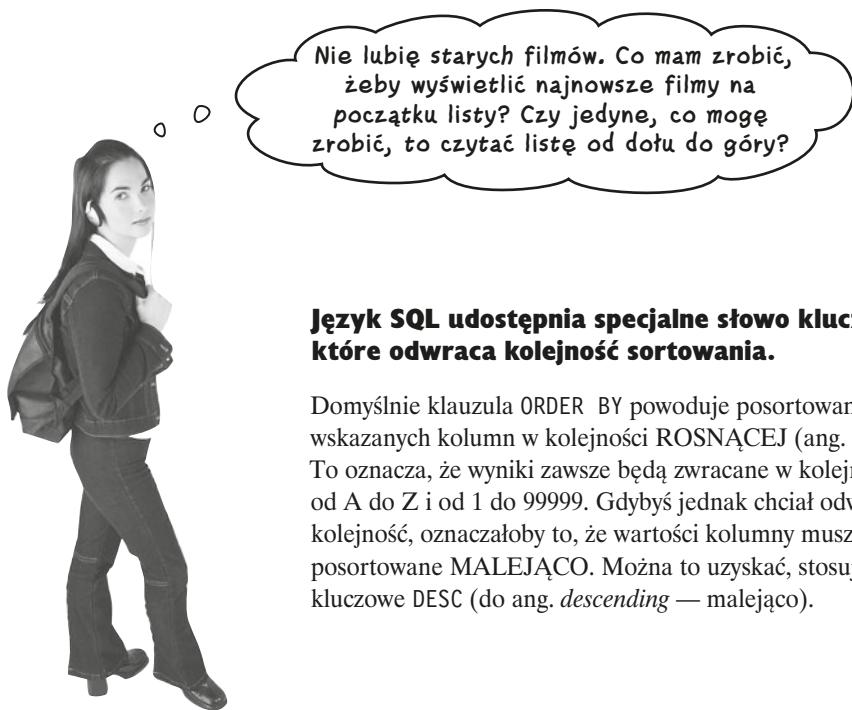
A oto i uporządkowane wyniki zwrocone przez nasze zapytanie:

Trzecia kolumna, według której były sortowane filmy. ↴

Pierwsza kolumna, według której były sortowane filmy. ↴

Druga kolumna, według której były sortowane filmy. ↴

id_filmu	tytuł	ocena	kategoria	zakupiono
84	Grześ: Historia nieznana	SD	akcji	5.2.2001
86	Paraskavedekatriaphobia	R	akcji	19.4.2003
94	Wielki Gatsby	R	dramat	3.2.2002
89	Błyskotki	SD	dramat	6.3.2002
85	Szaleni kłowni	R	horror	20.11.1999
91	Przynęta na rekiny	D	inny	20.11.1999
88	Koniec linii	R	inny	5.2.2001
92	Wkurzony pirat	SD	inny	19.4.2003
90	Zwrot	R	komedia	5.2.2001
83	Wielka przygoda	D	rodzinny	6.3.2002
87	Szczurek o imieniu Darek	D	rodzinny	12.4.2003



Język SQL udostępnia specjalne słowo kluczowe, które odwraca kolejność sortowania.

Domyślnie klauzula ORDER BY powoduje posortowanie wskazanych kolumn w kolejności ROSNĄCEJ (ang. *ascending*). To oznacza, że wyniki zawsze będą zwracane w kolejności od A do Z i od 1 do 99999. Gdybyś jednak chciał odwrócić tę kolejność, oznaczałoby to, że wartości kolumny muszą zostać posortowane MALEJĄCO. Można to uzyskać, stosując słowo kluczowe DESC (do ang. *descending* — malejaco).

Nie istniejąca
grupie pytania

P: A czy słowo kluczowe DESC nie było używane do pobrania opisu (ang. DESCRIPTION) tabeli? Czy jesteście pewni, że można go użyć także do zmiany kolejności sortowania?

O: Owszem. Wszystko zależy od kontekstu. Jeśli użyjesz słowa kluczowego DESC przed nazwą tabeli, na przykład: DESC tabela_filmow, to zwrócony zostanie opis tabeli. W takim przypadku DESC jest skrótem od DESCRIPTION.

Jeśli jednak użyjesz DESC w klauzuli ORDER BY, to będzie ono oznaczało DESCENDING i spowoduje zmianę kolejności sortowania.

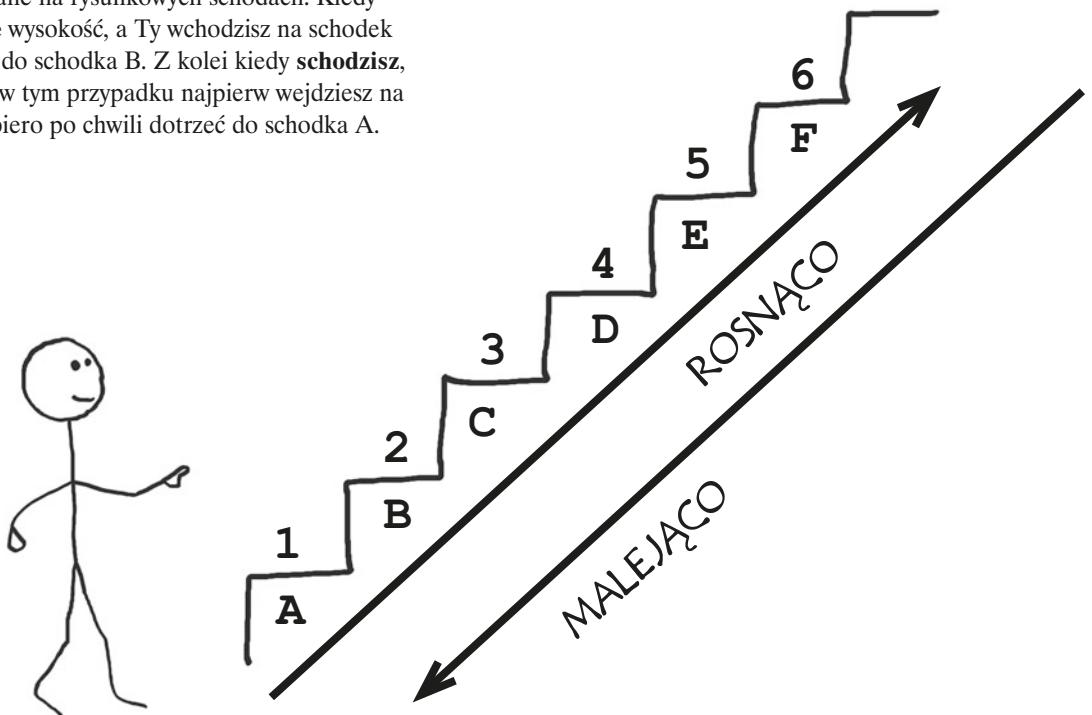
P: Czy żeby uniknąć nieporozumień, mogę stosować pełne słowa kluczowe — DESCRIBE oraz DESCENDING?

O: Możesz użyć polecenia DESCRIBE, jednak DESCENDING nie zadziała.

Aby odwrócić kolejność sortowania, umieść słowo kluczowe DESC w klauzuli ORDER BY za nazwą kolumny.

Zmiana kolejności dzięki użyciu DESC

Rozmieś swoje dane na rysunkowych schodach. Kiedy **wchodzisz**, rośnie wysokość, a Ty wchodzisz na schodek A, zanim dotrzesz do schodka B. Z kolei kiedy **schodzisz**, wysokość **maleje**; w tym przypadku najpierw wejdźesz na schodek Z, by dopiero po chwili dotrzeć do schodka A.



Poniższe zapytanie zwraca listę filmów posortowanych według daty zakupu, przy czym *ostatnio zakupione* filmy zostaną wyświetlane najpierw. Jeśli tego samego dnia kupiono kilka filmów, to zostaną one posortowane w kolejności alfabetycznej:

```
SELECT tytul, zakupiono  
FROM tabela_filmow  
ORDER BY zakupiono DESC, tytul ASC;
```

↑
Jeśli chcesz, by dane zostały wyświetlone w kolejności od Z do A lub od 9 do 1, to musisz zastosować słowo kluczowe DESC.

↑
Możemy tu umieścić słowo kluczowe ASC, jednak nie jest to konieczne. Wystarczy, byś zapamiętał, że oznacza ono domyślny — rosnący — porządek sortowania.

Do: Pracowników wypożyczalni
Filmoteka Bazodanowa

Od: Szefa

Temat: Upominki dla wszystkich!

Czołem

Wypożyczalnia wygląda rewelacyjnie! Poukładaliście wszystkie filmy na właściwych miejscach... a, i dziękuję wam za tę niesamowitą klauzulę ORDER BY w kodzie SQL, dzięki której wszyscy mogą odnaleźć dokładnie to, o co im chodzi.

W ramach podziękowania za ciężką pracę zapraszam was wszystkich do siebie na imprezę z pizzą. Zajrzyjcie około 18.

I nie zapomnijcie przynieść tych wszystkich raportów, o które prosiłem!

Wasz szef.

P.S. Nie ubierajcie się wieczorowo... mam w domu bibliotekę, którą chciałbym przeorganizować...

Problem najlepszej sprzedawczyni grupy Młode Gospodarstwa

Opiekunka grupy Młodych Gospodarstw próbuje określić, która z dziewcząt sprzedaje najwięcej ciasteczek. Jak na razie udało jej się utworzyć tabelę, w której zapisała wartość sprzedaży uzyskaną przez poszczególne dziewczęta każdego dnia.



W tej kolumnie są zapisywane imiona młodych gospodarstw.

Wartość sprzedanych ciasteczek w złotówkach.

Data sprzedaży ciasteczek.

↓ sprzedaz_ciasteczek ↓

id	imie	sprzedaz	data
1	Lidia	32.02	6.3.2007
2	Patrycja	26.53	6.3.2007
3	Beata	11.25	6.3.2007
4	Natalia	18.96	6.3.2007
5	Lidia	9.16	7.3.2007
6	Patrycja	1.52	7.3.2007
7	Beata	43.21	7.3.2007
8	Natalia	8.05	7.3.2007
9	Lidia	17.62	8.3.2007
10	Patrycja	24.19	8.3.2007
11	Beata	3.40	8.3.2007
12	Natalia	15.21	8.3.2007
13	Lidia	0.00	9.3.2007
14	Patrycja	31.99	9.3.2007
15	Beata	2.58	9.3.2007
16	Natalia	0.00	9.3.2007
17	Lidia	2.34	10.3.2007
18	Patrycja	13.44	10.3.2007
19	Beata	8.78	10.3.2007
20	Natalia	26.82	10.3.2007
21	Lidia	3.71	11.3.2007
22	Patrycja	0.56	11.3.2007
23	Beata	34.19	11.3.2007
24	Natalia	7.77	11.3.2007
25	Lidia	16.23	12.3.2007
26	Patrycja	0.00	12.3.2007
27	Beata	4.50	12.3.2007
28	Natalia	19.22	12.3.2007

Zaostrz ołówek



Młoda gospodarka, która zarobiła najwięcej na sprzedaży ciasteczek, wygra darmowe lekcje jazdy konnej. Każda z dziewcząt chce wygrać tę nagrodę, zatem bardzo duże znaczenie ma, by Edwina prawidłowo wyznaczyła zwyciężczynię, zanim sytuacja stanie się nieprzyjemna.

Wykorzystaj nowo poznaną klauzulę ORDER BY, by napisać zapytanie, które pomoże Edwinie określić zwyciężczynię.

Rozwiążanie ćwiczenia

Zaostrz ołówek



Rozwiążanie

Młoda gospośia, która zarobiła najwięcej na sprzedaży ciasteczek, wygra darmowe lekcje jazdy konnej. Każda z dziewcząt chce wygrać tę nagrodę, zatem bardzo duże znaczenie ma, by Edwina prawidłowo wyznaczyła zwyciężczynię, zanim sytuacja stanie się nieprzyjemna.

Wykorzystaj nowo poznaną klauzulę ORDER BY, by napisać zapytanie, które pomoże Edwinie określić zwyciężczynię.

```
SELECT imie, sprzedaz  
FROM sprzedaz_ciasteczek  
ORDER BY imie;
```



Oto nasze zapytanie...

... a to jego rezultaty.

imie	sprzedaz
Natalia	19.22
Natalia	0.00
Natalia	8.05
Natalia	26.82
Natalia	7.77
Natalia	15.21
Natalia	18.96
Beata	3.40
Beata	2.58
Beata	4.50
Beata	11.25
Beata	8.78
Beata	43.21
Beata	34.19
Lidia	17.62
Lidia	9.16
Lidia	0.00
Lidia	32.02
Lidia	2.34
Lidia	3.71
Lidia	16.23
Patrycja	26.53
Patrycja	0.00
Patrycja	0.56
Patrycja	1.52
Patrycja	13.44
Patrycja	24.19
Patrycja	31.99



Kwoty sprzedaży dla każdej z dziewcząt trzeba niestety zsumować ręcznie — na razie nie ma innej metody wyłonienia zwyciężczyni.

Funkcja SUM zsumuje wszystko za nas

Gra idzie o wysoką stawkę. Nie możemy popełnić błędu i narazić się na wściekłość naszych małych gospod. Dlatego zamiast dodawać wszystko ręcznie, możemy zwalić tę brudną robotę na SQL.

Język SQL udostępnia specjalne słowa kluczowe, nazywane *funkcjami*. Funkcje to fragmenty kodu, które wykonują pewne operacje na jednej wartości lub ich grupie. Pierwsza z funkcji, której poznasz, wykonuje operację matematyczną na kolumnie. Jest to funkcja SUM, a jej działanie polega na **zsumowaniu wszystkich wartości w kolumnie**, której nazwę podano w nawiasach za nazwą funkcji. A teraz zobaczymy tę funkcję w akcji.

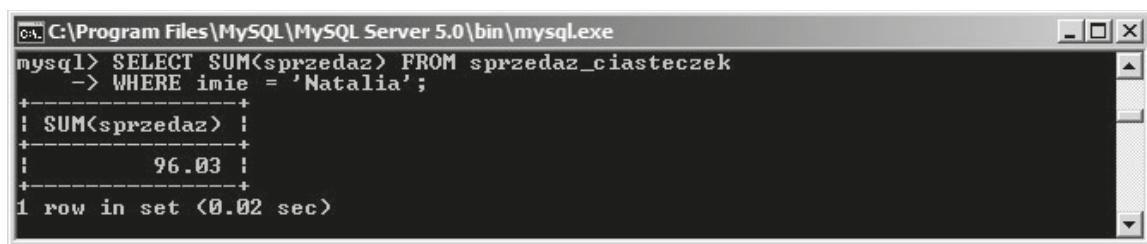
Funkcja SUM zsumuje wartości w kolumnie sprzedaz.

↓ ↓ ↗

SELECT SUM(sprzedaz)
FROM sprzedaz_ciasteczek
WHERE imie = 'Natalia';

Ten warunek ogranicza zapytanie i nakazuje mu zsumowanie wyłącznie sprzedaży Natalii. Gdyby go nie było, zapytanie zsumowałoby wszystkie wartości sprzedaży zapisane w tabeli.

SUM to tak zwana funkcja. Oznacza to, że wykonuje ona pewne czynności na kolumnie podanej w nawiasach za nazwą funkcji.



```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT SUM(sprzedaz) FROM sprzedaz_ciasteczek
-> WHERE imie = 'Natalia';
+-----+
| SUM(sprzedaz) |
+-----+
|      96.03 |
+-----+
1 row in set (0.02 sec)
```

Teraz potrzebujemy jeszcze wyliczyć pozostałe trzy sumy i będzie po sprawie. Choć oczywiście byłoby łatwiej, gdybyśmy mogli wyliczyć wszystkie sumy, używając jednego zapytania...



Ćwiczenie

Spróbuj to zrobić sam. Utwórz tabelę taką jak nasza tabela sprzedaz_ciasteczek i zapisz w niej jakieś liczby z miejscami dziesiętnymi. A następnie wypróbuj zapytania przedstawione na kilku następnych stronach.

— Wypróbuj to w domu

Zsumuj wszystko za jednym razem dzięki użyciu GROUP BY

Istnieje rozwiązanie pozwalające sumować wartość sprzedaży ciasteczek każdej z dziewcząt w jednym zapytaniu SQL. Wystarczy dodać do zapytania z funkcją SUM klauzulę GROUP BY. W ten sposób zgrupujemy wszystkie wiersze tabeli, w których jest zapisane to samo imię, a następnie zsumujemy wartości sprzedaży dla wyznaczonych w ten sposób grup.

SELECT imie, SUM(sprzedaz)

FROM sprzedaz_ciasteczek *Sumujemy wszystkie wartości z kolumny sprzedaz.*

GROUP BY imie *Grupujemy wartości z kolumny imie.*

ORDER BY SUM(sprzedaz) DESC;

Musimy posortować wyniki według sum, które wcześniej wyznaczyliśmy.

Chcemy wyświetlić wartości w kolejności malejącej, tak by na samym początku listy znalazła się zwyciężczyni.

imie	sprzedaz
Natalia	19.22
Natalia	0.00
Natalia	8.05
Natalia	26.82
Natalia	7.77
Natalia	15.21
Natalia	18.96

To zapytanie sumuje wyniki sprzedaży w grupach wyznaczonych na podstawie imienia.

imie	sprzedaz
Beata	3.40
Beata	2.58
Beata	4.50
Beata	11.25
Beata	8.78
Beata	43.21
Beata	34.19
Lidia	17.62
Lidia	9.16
Lidia	0
Lidia	32.02
Lidia	2.34
Lidia	3.71
Lidia	16.23
Patrycja	24.19
Patrycja	31.99

Zwyciężyła Beata!

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT imie, SUM(sprzedaz)
-> FROM sprzedaz_ciasteczek GROUP BY imie
-> ORDER BY SUM(sprzedaz) DESC;
+-----+-----+
| imie | SUM(sprzedaz) |
+-----+-----+
| Beata |      107.91 |
| Patrycja |    98.23 |
| Natalia |    96.03 |
| Lidia |     81.08 |
+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

Funkcja AVG z klauzulą GROUP BY

Trzy pozostałe dziewczęta były bardzo zawiedzione, zatem Edwina zdecydowała się przyznać jeszcze jedną nagrodę tej spośród nich, która miała największą średnią dzienną sprzedaż ciasteczek.

Do wyliczenia tej średniej Edwina chce wykorzystać funkcję AVG.

Każda z dziewcząt sprzedawała ciasteczka przez siedem dni.

Dla każdej z nich funkcja AVG zsumuje wszystkie wartości sprzedaży i podzieli je przez 7.

Także i tym razem grupujemy wartości w kolumnie „imie”...

... jednak w odróżnieniu od poprzedniego zapytania teraz wyliczamy ich średnią.

```
SELECT imie, AVG(sprzedaz)
FROM sprzedaz_ciasteczek
GROUP BY imie;
```

Funkcja AVG dodaje wszystkie wartości należące do grupy, a następnie wylicza średnią, dzieląc tę sumę przez liczbę wartości w grupie.

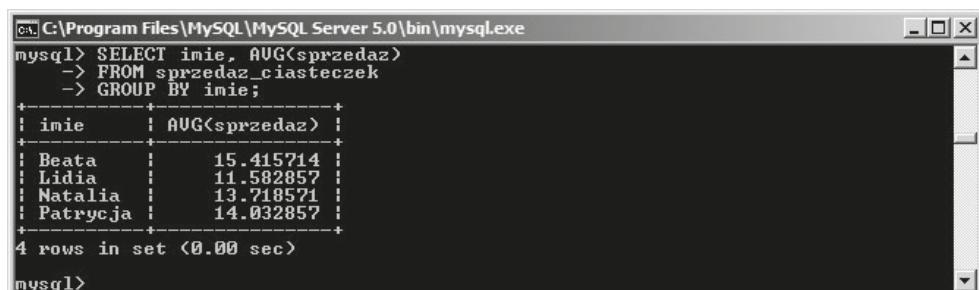
imie	sprzedaz
Natalia	19.22
Natalia	0.00
Natalia	8.05
Natalia	26.82
Natalia	7.77
Natalia	15.21
Natalia	18.96

imie	sprzedaz
Patrycja	26.53
Patrycja	0.00
Patrycja	0.56
Patrycja	1.52
Patrycja	13.44
Patrycja	24.19
Patrycja	31.99

imie	sprzedaz
Lidia	17.62
Lidia	9.16
Lidia	0
Lidia	32.02
Lidia	2.34
Lidia	3.71
Lidia	16.23

imie	sprzedaz
Beata	3.40
Beata	2.58
Beata	4.50
Beata	11.25
Beata	8.78
Beata	43.21
Beata	34.19

Ups... znowu wygrała Beata. Musimy wymyślić jakiś inny sposób wyłonienia zdobywczyni nagrody pocieszenia.



```
C:\Program Files\MySQL\MySQL Server 5.0\bin>mysql -u root -p
mysql> SELECT imie, AVG(sprzedaz)
-> FROM sprzedaz_ciasteczek
-> GROUP BY imie;
+-----+-----+
| imie | AVG(sprzedaz) |
+-----+-----+
| Beata | 15.415714 |
| Lidia | 11.582857 |
| Natalia | 13.718571 |
| Patrycja | 14.032857 |
+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

Funkcje MIN i MAX

Chcąc sprawdzić wszystkie możliwości, Edwina skontrolowała minimalne i maksymalne wartości sprzedaży, aby sprawdzić, czy któraś z dziewcząt nie ma wyższej dziennej wartości sprzedaży niż Beata, bądź nawet czy Beata nie miała gorszego dnia, w którym jej wartość sprzedaży była niższa od wartości uzyskanej przez którąś z pozostałych dziewcząt.

Do określenia **największej wartości w kolumnie** możemy użyć funkcji MAX. Z kolei funkcja MIN zwraca **najmniejszą wartość w kolumnie**.

```
SELECT imie, MAX(sprzedaz)
FROM sprzedaz_ciasteczek
GROUP BY imie;
```

Funkcja MAX zwraca jedną, największą wartość sprzedaży dla każdej z dziewcząt.

Niespodzianka!
Beata odnotowała także najwyższą wartość sprzedaży jednego dnia.

imie	sprzedaz
Natalia	26.82
Beata	43.21
Lidia	32.02
Patrycja	31.99

```
SELECT imie, MIN(sprzedaz)
FROM sprzedaz_ciasteczek
GROUP BY imie;
```

Funkcja MIN zwraca jedną, najmniejszą wartość sprzedaży dla każdej z dziewcząt.

I choć wygląda na to, że pozostałe dziewczyny zawsze przynajmniej jeden dzień, to Beata nawet najgorszego dnia coś zarobiła.

imie	sprzedaz
Natalia	0.00
Beata	2.58
Lidia	0.00
Patrycja	0.00

Sytuacja robi się poważna. Może przyznam nagrodę dziewczynie, która sprzedawała ciasteczka przez większą liczbę dni niż pozostałe...



Liczymy dni

Aby sprawdzić, czy któraś z dziewcząt sprzedawała ciasteczka przez większą ilość dni niż pozostałe, Edwina policzy, ile pracowała każda z nich. Zastosuje do tego celu funkcję COUNT. Funkcja ta zwraca *liczbę wierszy w kolumnie*.

**SELECT COUNT(data)
FROM sprzedaz_ciasteczek;**

Funkcja COUNT zwraca liczbę wierszy w kolumnie data. Jeśli w którymś z wierszy w tej kolumnie wystąpi wartość NULL, to wiersz nie zostanie uwzględniony.

Zaostrz ołówek



sprzedaz_ciasteczek

id	imie	sprzedaz	data
1	Lidia	32.02	6.3.2007
2	Patrycja	26.53	6.3.2007
3	Beata	11.25	6.3.2007
4	Natalia	18.96	6.3.2007
5	Lidia	9.16	7.3.2007
6	Patrycja	1.52	7.3.2007
7	Beata	43.21	7.3.2007
8	Natalia	8.05	7.3.2007
9	Lidia	17.62	8.3.2007
10	Patrycja	24.19	8.3.2007
11	Beata	3.40	8.3.2007
12	Natalia	15.21	8.3.2007
13	Lidia	0.00	9.3.2007
14	Patrycja	31.99	9.3.2007
15	Beata	2.58	9.3.2007
16	Natalia	0.00	9.3.2007
17	Lidia	2.34	10.3.2007
18	Patrycja	13.44	10.3.2007
19	Beata	8.78	10.3.2007
20	Natalia	26.82	10.3.2007
21	Lidia	3.71	11.3.2007
22	Patrycja	0.56	11.3.2007
23	Beata	34.19	11.3.2007
24	Natalia	7.77	11.3.2007
25	Lidia	16.23	12.3.2007
26	Patrycja	0.00	12.3.2007
27	Beata	4.50	12.3.2007
28	Natalia	19.22	12.3.2007

Oto oryginalna tabela. Jak myślisz, jaki wynik zwróci powyższe zapytanie?

.....
.....

Czy ta liczba reprezentuje faktyczną ilość dni, w które dziewczęta sprzedawały ciasteczka?

.....
.....

Napisz zapytanie, które policzy, ile dni każda z dziewcząt sprzedawała ciasteczka.

.....
.....
.....

Zaostrz ołówek



Rozwiążanie

Oto oryginalna tabela. Jak myślisz, jaki wynik zwróci powyższe zapytanie?

28 dat sprzedaży

Czy ta liczba reprezentuje faktyczną ilość dni, w które dziewczęta sprzedawały ciasteczkę?

Nie. Ta liczba określa po prostu liczbę wartości w kolumnie data tabeli sprzedaz_ciasteczek.

Napisz zapytanie, które policzy, ile dni każda z dziewcząt sprzedawała ciasteczkę.

```
SELECT imie, COUNT(data)
FROM sprzedaz_ciasteczek
GROUP BY imie;
```



Mogliście przecież użyć klauzuli ORDER BY, by posortować wyniki według kolumny data, a następnie sprawdzić datę pierwszego i ostatniego dnia i na tej podstawie określić, ile dni dziewczyny sprzedawały ciasteczkę, prawda?

Otoc nie. Nie moglibyśmy bowiem mieć pewności, czy pomiędzy pierwszym i ostatnim dniem nie brakuje jakichś dat.

Istnieje jednak znacznie prostszy sposób określenia liczby dni, w które dziewczęta sprzedawały ciasteczkę. Polega on na zastosowaniu słowa kluczowego DISTINCT. Stosując je, możemy uzyskać nie tylko poszukiwaną liczbę dni sprzedaży ciasteczek, lecz także listę dat tych dni, i to taką, na której poszczególne daty nie będą się powtarzać.

Pobieranie unikalnych wartości

Przede wszystkim zobaczymy, jakie są efekty zastosowania słowa kluczowego DISTINCT, kiedy zostanie ono użyte *bez* funkcji COUNT.

DISTINCT jest słowem kluczowym, a nie funkcją, dlatego nazwa kolumny, w tym przypadku „data”, nie musi być umieszczona w nawiasach.

```
SELECT DISTINCT data
FROM sprzedaz_ciasteczek
ORDER BY data;
```

A oto i klauzula ORDER BY, dzięki której będziemy mogli łatwo określić pierwszą i ostatnią datę sprzedaży ciasteczek.

Spójrz,
żadne daty
się nie
powtarzają!

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT DISTINCT data
-> FROM sprzedaz_ciasteczek
-> ORDER BY data;
+-----+
| data |
+-----+
| 2007-03-06 |
| 2007-03-07 |
| 2007-03-08 |
| 2007-03-09 |
| 2007-03-10 |
| 2007-03-11 |
| 2007-03-12 |
+-----+
7 rows in set (0.00 sec)
```

A teraz wypróbujmy zapytanie, w którym *użyjemy* funkcji COUNT:

Zauważ, że słowo kluczowe DISTINCT zostało umieszczone w nawiasach razem z nazwą kolumny.

```
SELECT COUNT(DISTINCT data)
FROM sprzedaz_ciasteczek;
```

W tym przypadku klauzula ORDER BY nie jest nam potrzebna, gdyż zapytanie zwróci jedną wartość. Zatem nie ma czego sortować.



**WYŁĘŻ
UMYSŁ**

Wypróbuj to zapytanie, a następnie zastosuj je, by określić, która z dziewcząt sprzedawała przez najwięcej dni.

Odpowiedź: Beata

Kim jestem?

Grupa przebranych funkcji i słów kluczowych SQL uczestniczy w grze towarzyskiej „Zgadnij, kim jestem”. Jej uczestnicy podają Ci podpowiedzi, a Ty na ich podstawie starasz się odgadnąć, kim oni są. Możesz przy tym założyć, że uczestnicy gry zawsze mówią o sobie prawdę. W pustym polu obok każdej z podpowiedzi zapisz nazwę uczestnika zabawy; dodatkowo w kolejnym pustym polu zapisz, czy jest to słowo kluczowe, czy funkcja.

Uczestnicy zabawy:

COUNT, DISTINCT, AVG, MIN, GROUP BY, SUM, MAX

Kim jestem?



Nazwa	Funkcja czy słowo kluczowe
-------	-------------------------------

Wyniki, które uzyskujesz, stosując mnie, mogą być niewiele warte.

.....

To, co zwracam, jest większe od wszystkiego, co pobieram.

.....

Zwracam unikalne wyniki.

.....

Powiem Ci, ile ich było.

.....

Musisz mnie użyć, jeśli chcesz wyznaczyć sumę.

.....

Interesują mnie tylko duże sumy.

.....

Kim jestem? Kimś pomiędzy.

.....

Odpowiedzi znajdziesz na stronie 309

Nie istniejąca
grupie pytania

P: Czy podczas poszukiwania największych wartości przy wykorzystaniu funkcji AVG, MAX oraz MIN nie mogliście dodać do zapytania klauzuli ORDER BY?

O: Mogliśmy i byłby to bardzo dobry pomysł. Zdecydowaliśmy jednak, że tego nie zrobimy, aby nie komplikować niepotrzebnie zapytań i ułatwić Ci poznanie nowych funkcji. Możesz teraz zatrzymać się na tych zapytań i wyobrazić sobie umieszczenie w nich klauzuli ORDER BY. Czy widzisz, jaki miałoby to wpływ na postać danych wynikowych?

P: Słowo kluczowe DISTINCT wydaje się być bardzo przydatne. Czy mogę go używać wraz z dowolną kolumną?

O: Tak, możesz. Jest ono szczególnie przydatne w przypadkach, gdy w tabeli jest więcej rekordów zawierających w pewnej kolumnie tę samą wartość, a nam zależy na wyświetleniu wszystkich unikalnych wartości zapisanych w tabeli, a nie długiej listy, na której wartości by się wielokrotnie powtarzały.

P: To zapytanie, w którym zastosowaliście funkcję MIN, tak naprawdę nie miało niczego wspólnego z próbami Edwiny wyłonienia kandydatki do nagrody pocieszenia, prawda?

O: Nie. Jednak mogłoby ono pomóc Edwinie we wskazaniu dziewcząt, które uzyskiwały najgorsze wyniki. W następnym roku mogłaby zwrócić na nie więcej uwagi i lepiej zmotywować je do pracy.

P: Skoro już rozmawiamy o funkcji MIN — co się stanie, jeśli w kolumnie będzie zapisana wartość NULL?

O: To dobre pytanie. Trzeba wiedzieć, że wartości NULL nie są uwzględniane przez żadną z tych funkcji. W końcu NULL oznacza brak wartości, a to nie to samo co zero.



Hm... Funkcje AVG, MAX oraz COUNT nie pomogły mi wyłonić kandydatki do nagrody pocieszenia. Ciekawe, czy nie mogłabym zastosować funkcji SUM, by określić, która z dziewcząt zajęła drugie miejsce, i dać jej tę nagrodę.

WYSIL SZARE KOMÓRKI

Wyobraź sobie, że mielibyśmy nie cztery, a *czterdzięści* młodych gości. W jaki sposób moglibyśmy zastosować funkcję SUM do określenia, która z nich zajęła drugie miejsce?

LIMIT-owanie ilości wyników

Teraz chcemy zastosować funkcję SUM do określenia dziewczyny, która zajęła drugie miejsce. Zerknijmy jeszcze raz na oryginalne zapytanie oraz jego wyniki i spróbujmy określić, jak moglibyśmy wskazać zwyciężczynię.

```
SELECT imie, SUM(sprzedaz)
FROM sprzedaz_ciasteczek
GROUP BY imie
ORDER BY SUM(sprzedaz) DESC;
```

W tym przypadku zastosowanie klauzuli ORDER BY ma kluczowe znaczenie; w przeciwnym razie uzyskane wyniki byłyby dowolne.

imie	sprzedaz
Beata	107.91
Patrycja	98.23
Natalia	96.03
Lidia	81.08

Tak naprawdę interesują nas tylko te dwa wiersze.

Zdobywczynią nagrody pocieszenia jest Patrycja! Po ogłoszeniu werdyktu Natalia przestała się do niej odzywać.

Ponieważ mamy jedynie cztery wyniki, zatem określenie, kto zajął drugie miejsce, nie przysporzy nam zbyt wielu problemów. Niemniej jednak, gdybyśmy chcieli być bardzo precyzyjni, moglibyśmy ograniczyć dane i wyświetlić tylko dwoje dziewcząt, które zasłużyły na nagrody. Moglibyśmy to zrobić, korzystając z klauzuli LIMIT, która pozwala dokładnie określić, ile rekordów z wyznaczonego zbioru danych ma być wyświetlonych w wynikach.

```
SELECT imie, SUM(sprzedaz)
FROM sprzedaz_ciasteczek
GROUP BY imie
ORDER BY SUM(sprzedaz) DESC
LIMIT 2;
```

Oznacza to, że chcemy ograniczyć do dwóch ilość rekordów wyświetlonych w wynikach.

To naprawdę długie zapytanie, zauważwszy, że zwraca tylko dwa wiersze wyników.

imie	sprzedaz
Beata	107.91
Patrycja	98.23

W naszym przypadku mamy tylko cztery młode gospodki, zatem ograniczanie ilości wyników do dwóch rekordów nie jest szczególnie pomocne; wyobraźmy sobie jednak, co by było, gdybyśmy pracowali na znacznie większej tabeli? Wyobraź sobie, że operujemy na tabeli zawierającej 1000 piosenek odtwarzanych aktualnie przez rozgłośnię radiową i chcemy uzyskać listę 100 najpopularniejszych z nich. Dzięki klauzuli LIMIT moglibyśmy wyświetlić dokładnie te piosenki, o które nam chodzi, i pominąć 900 pozostałych.

Ograniczenie tylko do drugiego miejsca

Co więcej, klauzula LIMIT pozwala na wyświetlenie wyłącznie dziewczyny, która zajęła drugie miejsce, bez prezentowania głównej zwyciężczyni. W tym celu musimy zastosować klauzulę LIMIT z dwoma parametrami:

Gdybyś próbował odgadnąć, co oznacza ten zapis, zapewne by Ci się to nie udało. Okazuje się, że w przypadku zastosowania dwóch parametrów znaczenie klauzuli LIMIT jest całkowicie inne niż w sytuacji, gdy zostanie użyty tylko jeden parametr.



Czy pamiętasz nasz przykład ze 100 piosenkami? Założmy, że chcielibyśmy zobaczyć piosenki, które w rankingu popularności znalazły się na miejscach od 20. do 30.

Moglibyśmy je wyświetlić, dodając do klauzuli LIMIT drugi parametr. Wystarczyłoby posortować wyniki według popularności i dodać do zapytania klauzulę LIMIT 19, 10.

W tym przypadku liczba 19 oznacza, że zwracanie wyników należy zacząć od 20. wiersza (pierwszy z nich ma bowiem numer 0), natomiast liczba 10 określa, ile wierszy należy zwrócić.

Zaostrz ołówek



Napisz zapytanie, które zwróci nam młodą gospośię, która uzyskała drugi wynik sprzedaży, ale **tylko i wyłącznie ją**. Zastosuj w tym celu klauzulę LIMIT z dwoma parametrami.

Rozwiążanie ćwiczenia

Zaostrz ołówek

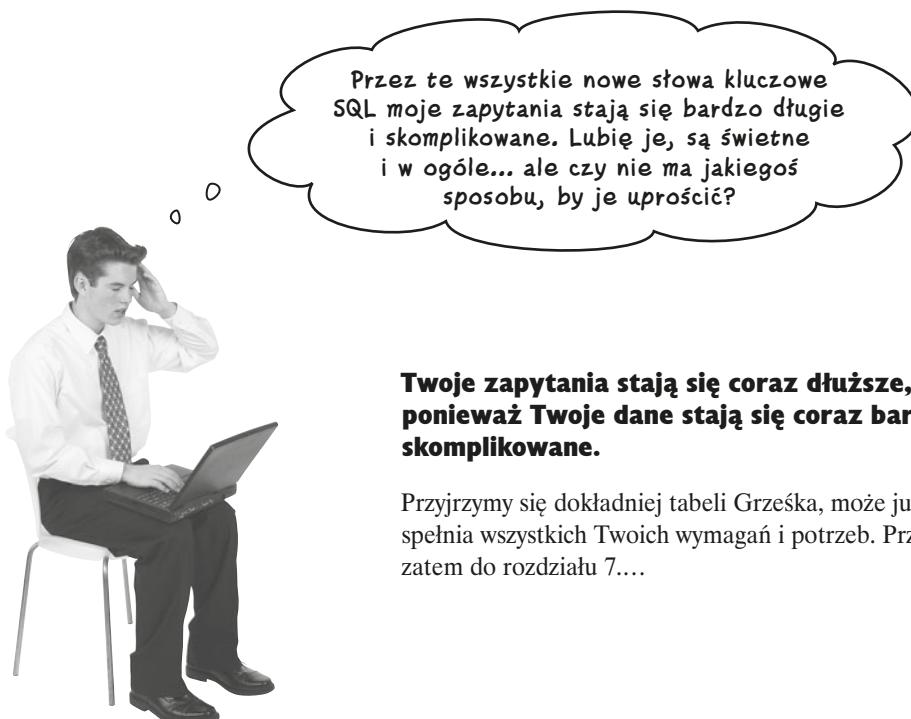


Rozwiążanie

Napisz zapytanie, które zwróci nam młodą gospośkę, która uzyskała drugi wynik sprzedaży, ale *tylko i wyłącznie ją*. Zastosuj w tym celu klauzulę LIMIT z dwoma parametrami.

```
SELECT imie, SUM(sprzedaz)
FROM sprzedaz_ciasteczek
GROUP BY imie
ORDER BY SUM(sprzedaz) DESC
LIMIT 1,1
```

Pamiętasz? SQL zaczyna liczyć od 0, zatem tak naprawdę 1 to 2.



Twoje zapytania stają się coraz dłuższe, ponieważ Twoje dane stają się coraz bardziej skomplikowane.

Przyjrzymy się dokładniej tabeli Grześka, może już nie spełnia wszystkich Twoich wymagań i potrzeb. Przejdźmy zatem do rozdziału 7....

Kim jestem?



ze strony 304

Grupa przebranych funkcji i słów kluczowych SQL uczestniczy w grze towarzyskiej „Zgadnij, kim jestem”. Jej uczestnicy podają Ci podpowiedzi, a Ty na ich podstawie staraś się odgadnąć, kim oni są. Możesz przy tym założyć, że uczestnicy gry zawsze mówią o sobie prawdę. W pustym polu obok każdej z podpowiedzi zapisz nazwę uczestnika zabawy; dodatkowo w kolejnym pustym polu zapisz, czy jest to słowo kluczowe, czy funkcja.

Uczestnicy zabawy:

COUNT, DISTINCT, AVG, MIN, GROUP BY, SUM, MAX

Nazwa	Funkcja czy słowo kluczowe
-------	-------------------------------

Wyniki, które uzyskujesz, stosując mnie, mogą być niewiele warte.

MIN	funkcja
------------	---------

To, co zwracam, jest większe od wszystkiego, co pobieram.

SUM	funkcja
------------	---------

Zwracam unikalne wyniki.

DISTINCT	słowo kluczowe
-----------------	----------------

Powiem Ci, ile ich było.

COUNT	funkcja
--------------	---------

Musisz mnie użyć, jeśli chcesz wyznaczyć sumę.

GROUP BY	słowo kluczowe
-----------------	----------------

Interesują mnie tylko duże sumy.

MAX	funkcja
------------	---------

Kim jestem? Kimś pomiędzy.

AVG	funkcja
------------	---------



Przybornik SQL

A zatem opanowałeś materiał z szóstego rozdziału książki i blyszcysz w towarzystwie znajomością tych wszystkich funkcji stosowanych w poleceniach SELECT, słów kluczowych SQL

i zapytań. Kompletną listę porad znajdziesz w dodatku C, zamieszczonym na końcu książki.

ORDER BY

Sortuje alfabetycznie wyniki na podstawie zawartości wskazanej kolumny.

GROUP BY

Konsoliduje wiersze na podstawie wartości we wskazanej kolumnie tabeli.

DISTINCT

Powoduje, że każda unikalna wartość zapisana w tabeli zostanie zwrocona tylko jeden raz — innymi słowy, wartości nie będą się powtarzać.

SUM

Oblicza sumę wartości zapisanych w kolumnie liczbowej.

COUNT

Potrafi zwrócić liczbę wierszy spełniających kryteria podane w poleceniu SELECT bez zwracania samych danych. COUNT zwraca liczbę całkowitą.

AVG

Wyznacza średnią arytmetyczną wartości zapisanych w kolumnie liczbowej.

MAX oraz MIN

Pierwsza z tych funkcji zwraca największą wartość w kolumnie, natomiast druga — wartość najmniejszą.

LIMIT

Pozwala dokładnie określić, ile wierszy może zwrócić zapytanie oraz od którego wiersza rozpocząć zwracanie wyników.

Oto Twoje nowe narzędzia:
Zaawansowane funkcje
stosowane w poleceniach
SELECT, słowa kluczowe
i zapytania.

7. Projektowanie baz danych składających się z wielu tabel

Wyrastamy z naszych starych tabel



Kiedyś może nadjeść moment, gdy pojedyncza tabela przestanie Ci wystarczać. Twoje dane stały się bardziej złożone i jedna tabela, której do tej pory używałeś, nie jest już w stanie sprostać Twoim potrzebom. W tabeli jest bardzo dużo powtarzających się, nadmiarowych informacji, które niepotrzebnie zajmują miejsce na dysku i spowalniają zapytania. Wytrwałeś, używając tej jednej tabeli tak długo, jak to było możliwe. Jednak świat, w którym żyjemy, jest ogromny i czasami potrzeba będzie więcej niż jednej tabeli, by zapisać wszystkie dane, zapanować nad nimi i w ostatecznym rozrachunku cały czas być panem własnej bazy danych.

Szukamy partnerki na randkę dla Wieśka

Wiesiek, samotny przyjaciel Grześka, poprosił go o znalezienie kobiety o podobnych zainteresowaniach, z którą mógłby się umówić na randkę. Grzesiek zaczął poszukiwania od wyświetlenia danych Wieśka.

Oto Wiesław:

id_kontaktu: 341
nazwisko: Kowalski
imie: Wiesiek
telefon: 329475374
email: wiesiek.k@kurnachatka.pl
plec: M
data_urodzenia: 28.08.1975
zawód: rolnik
lokalizacja: Kolbuszowa
województwo: PK
stan: samotny
zainteresowania: zwierzęta, jazda konna, filmy
szuka: partnerki

Wiesiek



Dane w kolumnie zainteresowania nie są atomowe — zawierają w sobie informacje nie jednego, lecz kilku różnych typów. Grzesiek obawia się, że poszukiwanie zainteresowań w tej kolumnie nie będzie proste.

Grzesiek dopisał prośbę Wieśka do swojej listy rzeczy do zrobienia:

DO ZROBIENIA:

Napisać zapytanie dla Wieśka: Chodzi o napisanie zapytania operującego na kolumnie zainteresowań. Nie wygląda to najlepiej, trzeba będzie użyć operatora **LIKE...** całe szczęście, że nie muszę się tak męczyć zbyt często...

Po co cokolwiek zmieniać?

Grzesiek zdecydował się w żaden sposób nie modyfikować kolumny zainteresowania. Nie ma nic przeciwko napisaniu złożonego zapytania, gdyż przypuszcza, że nie będzie musiał tego robić zbyt często.

Grzesiek wykorzystał pole `data_urodzenia`, by odszukać kobiety, które nie będą młodsze lub starsze od Wieśka o więcej niż pięć lat.

Zaostrz ołówek



Dokończ zapytanie rozpoczęte przez Grześka, które pomoże Wieśkowi odszukać partnerkę o takich samych zainteresowaniach. Do polecenia SQL dopisz wyjaśnienia opisujące znaczenie każdego wiersza kodu.

```
SELECT * FROM moje_kontakty
WHERE plec = 'K'
AND stan = 'samotna'
AND wojewodztwo = 'PK'
AND szuka LIKE '%samotnego%'
AND data_urodzenia > '1970-08-28'
AND data_urodzenia < '1980-08-28'
AND zainteresowania LIKE .....
AND .....
AND .....
```

Rozwiążanie ćwiczenia

Zaostrz ołówek

Rozwiążanie

Dokończ zapytanie rozpoczęte przez Grześka, które pomoże Wieśkowi odszukać partnerkę o takich samych zainteresowaniach.

Do polecenia SQL dopisz wyjaśnienia opisujące znaczenie każdego wiersza kodu.

```
SELECT * FROM moje_kontakty
WHERE plec = 'K'
AND stan = 'samotna'
AND wojewodztwo = 'PK'
AND szuka LIKE '%samotnego%'
AND data_urodzenia > '1970-08-28'
AND data_urodzenia < '1980-08-28'
AND zainteresowania LIKE '%zwierzęta%'
AND .....zainteresowania LIKE '%konna%'
AND .....zainteresowania LIKE '%filmy%'
```

Pobieramy z tabeli moje_kontakty wszystko, co spełnia poniższe warunki.

Wiesiek chciałby się spotykać z kobietą, zatem szukamy kobiety...

... do tego ma być samotna...

... i powinna przynajmniej mieszkać w tym samym województwie co Wiesiek.

Dobrze by też było, aby szukana była samotnego mężczyzny.

Wiesiek chce, by jego partnerka na randki nie była od niego młodsza ani starsza o więcej niż pięć lat.

Te trzy wiersze ograniczą wyniki tylko do tych kandydatek, które mają takie same zainteresowania jak Wiesiek. Moglibyśmy tu zastosować operator OR, jednak chcemy, by kandydatka miała wszystkie te same zainteresowania co Wiesiek.

Zapytanie zadziałało doskonale

Grześkowi udało się znaleźć idealną kandydatkę dla Wieśka:

id_kontaktu: 1854
nazwisko: Zimowska
imie: Klara
telefon: 753684753
email: klara.zimowska@pizza-nzk.pl
plec: K
data_urodzenia: 07.01.1974 ← Wiek jest w sam raz.
zawód: weterynarz ← Zawód — doskonaty.
lokalizacja: Rzeszów
wojewodztwo: PK ← Nawet mieszka całkiem blisko.
stan: samotna
zainteresowania: zwierzęta, filmy, jazda konna,
książki sensacyjne, spacery
szuka: samotnego faceta

Klara i Rączy

← A i zainteresowania się zgadzają!



Poszło aż za dobrze

Randka Klary i Wieśka udała się wyśmienicie. Jednak teraz Grzesiek stał się ofiarą swojego własnego sukcesu, ponieważ wszyscy jego samotni znajomi chcą, by im szukał towarzystwa na randki. A Grzesiek ma naprawdę dużo znajomych.



Projekt tabeli powinien ułatwiać realizację złożonych zapytań. Nie pisz zagmatwanych zapytań tylko po to, by „ominąć” problem źle zaprojektowanej tabeli.

Tworzenie takich zapytań zabiera zbyt wiele czasu. Grzesiek zapisał zatem stosowną uwagę na swojej liście rzeczy do zrobienia.

DO ZROBIENIA:

Napisać zapytanie dla Wieśka: Chodzi o napisanie zapytania operującego na kolumnie zainteresowań. Niewygodno, najlepiej trzeba będzie użyć operatora **LIKE**... całé szczęście, że nie muszę się tak męczyć zbyt często...

W przyszłości powiniem ignorować kolumnę zainteresowań, żeby sobie ułatwić pisanie zapytań.

Zignorowanie problemu nie jest rozwiążaniem

Robert, kolejny przyjaciel Grześka, też go poprosił o znalezienie kandydatki na randkę. Poszukuje kandydatki, która nie będzie młodsza lub starsza od niego o więcej niż pięć lat. Robert mieszka w Gdańsku, w województwie pomorskim, i ma całkowicie inne zainteresowania niż Wiesiek.

Grzesiek chciałby uprościć zapytania i skrócić czas ich tworzenia, dlatego zdecydował, że pominie kolumnę zainteresowań.



Robert →



Ćwiczenie

Napisz zapytanie, które odnajdzie kandydatki na randkę dla Roberta, przy czym nie uwzględniaj w nim kolumny zainteresowania.

```
id_kontaktu: 341  
nazwisko: Kraszkiewicz  
imie: Robert  
telefon: 583490972  
email: rob_olo@samotnyzejman.pl  
plec: M  
data_urodzenia: 20.03.1955  
zawod: komik  
lokalizacja: Gdańsk  
województwo: PM  
stan: samotny  
zainteresowania: zwierzęta, papiery wartościowe,  
geografia  
szuka: samotnej partnerki
```

→ Odpowiedzi znajdziesz na stronie 371

Zbyt wiele nieodpowiednich wyników

Grzesiek wręczył Robertowi długą listę potencjalnych kandydatek na randkę. Po kilku tygodniach Robert zadzwonił do Grześka z informacją, że żadna z kobiet z listy nie ma z nim nic wspólnego, a cała lista jest zupełnie bezużyteczna.



DO ZROBIENIA:

Napisać zapytanie dla Wieska: Chodzi o napisanie zapytania operującego na kolumnie zainteresowań. Niewygląda to najlepiej, trzeba będzie użyć operatora **LIKE**... całé szczęście, że nie muszę się tak męczyć zbyt często...

W przyszłości powiniem ignorować kolumnę zainteresowań, żeby sobie ułatwić pisanie zapytań.

Uwzględniać tylko pierwsze zainteresowanie ignorując całą pozostałą zawartość kolumny.

Zainteresowania SA są ważne. Nie powinniśmy ich ignorować, w tej kolumnie są bowiem podane cenne informacje.

Zastosowanie wyłącznie jednego zainteresowania

Teraz Grzesiek już wie, że nie może całkowicie ignorować zainteresowań. Grzesiek założył, że osoby zapisane w bazie podawały mu zainteresowania w kolejności od najbardziej do najmniej ważnego, dlatego też zdecydował się na uwzględnienie w zapytaniu wyłącznie pierwszego zainteresowania. Jego zapytania wciąż są dosyć skomplikowane, jednak i tak są lepsze do tych, w których pisał warunki dla wszystkich zainteresowań zarejestrowanych w tabeli.

Zaostrz ołówek

Zastosuj funkcję SUBSTRING_INDEX, by pobrać pierwsze zainteresowanie z kolumny zainteresowania.

Rozwiążanie kolejnego ćwiczenia

Zaostrz ołówek



Rozwiążanie

Zastosuj funkcję SUBSTRING_INDEX, by pobrać pierwsze zainteresowanie z kolumny zainteresowania.

SUBSTRING_INDEX(zainteresowania, ',', 1)

To wywołanie pobierze fragment tekstu z kolumny zainteresowania, od samego początku do miejsca wystąpienia pierwszego przecinka.

A oto i przecinek, którego funkcja poszukuje.

Tu podaliśmy liczbę 1, gdyż chodzi nam o pierwszy przecinek. Gdybyśmy podali tu liczbę 2, to funkcja kontynuowałaby poszukiwanie aż do momentu odnalezienia drugiego przecinka i pobrata cały fragment tekstu znajdujący się przed nim — co by odpowiadało dwóm pierwszym zainteresowaniom.

A zatem Grzesiek napisał kolejne zapytanie, które ma wyłonić kandydatkę na randkę z Robertem. Tym razem Grzesiek zastosował funkcję SUBSTRING_INDEX i założył, że pierwsze zainteresowanie powinno pasować do słowa „zwierzęta”.

```
SELECT * FROM moje_kontakty
WHERE plec = 'K'
AND stan = 'samotna'
AND wojewodztwo = 'PM'
AND szuka LIKE '%samotnego%'
AND data_urodzenia > '1950-20-03'
AND data_urodzenia < '1960-20-03'
AND SUBSTRING_INDEX(zainteresowania,',',1) = 'zwierzeta';
```

W wynikach pojawią się wyłącznie kobiety, które na pierwszym miejscu w kolumnie zainteresowania podały „zwierzęta”.

Odnaleziona kandydatka

Nareszcie! Grzesiek znalazł odpowiednią kandydatkę na randkę z Robertem:

id_kontaktu: 459
nazwisko: Franczyńska
imie: Aleksandra
telefon: 580939731
email: fr_ola@pizza-nzk.pl
plec: K
data_urodzenia: 19.09.1956 ← Wiek pasuje.
zawód: artystka
lokalizacja: Sopot
wojewodztwo: PM ← I mieszka niedaleko od Roberta.
stan: samotna
zainteresowania: zwierzęta ← Zainteresowania też pasują.
szuka: samotnego mężczyzny

Błędne dopasowanie

Robert zaprosił Aleksandrę na randkę, a Grzesiek nerwowo czekał na informację, jak się wszystko skończyło. Zaczął sobie wyobrażać tabelę `moje_kontakty` jako początek wielkiej witryny społecznościowej.

Następnego dnia Robert pojawił się u Grześka wyraźnie zasmucony i niezadowolony.

Robert krzyczał, nie mogąc się uspokoić: „Niewątpliwie była zainteresowana zwierzętami. Ale nie powiedziałeś mi, że interesują ją preparowane — w całym jej mieszkaniu było pełno martwych, wypchanych, zasuszonych zwierząt!“.

DO ZROBIENIA:

Napisać zapytanie dla Wieska: Chodzi o napisanie zapytania operującego na kolumnie zainteresowań. Niewygląda to najlepiej, trzeba będzie użyć operatora `LIKE...` Całe szczęście, że nie muszę się tak męczyć zbyt często...

W przyszłości powinienem ignorować kolumnę zainteresowań, żeby sobie ułatwić pisanie zapytań.

Uwzględniać tylko kolumnę zainteresowań ignorując całą pozostałą zawartość kolumny.

Stworzyć kilka kolumn, w których będzie się zapisywać po jednym zainteresowaniu, a to dlatego, że trzymanie wszystkich zainteresowań w jednej kolumnie utrudniłoby pisanie zapytań.

Idealna kandydatka dla Roberta znajdowała się w tabeli, jednak Grześkowi nie udało się jej odszukać, gdyż jej zainteresowania były zapisane w innej kolejności.

Grzesiek w końcu podjął decyzję o zmianie struktury tabeli.



WYSIL SZARE KOMÓRKI

Jak będą wyglądać przyszłe zapytania Grześka, kiedy doda do tabeli kilka kolumn z zainteresowaniami?

Dodajemy kolejne kolumny na zainteresowania

Grzesiek właśnie zdał sobie sprawę, że jedna kolumna zawierająca wszystkie zainteresowania utrudnia tworzenie zapytań i sprawia, że są one niedokładne. Dopasowując zainteresowania, musi używać operatora LIKE, przez co czasami nie udaje mu się znaleźć właściwych osób.

Ponieważ Grzesiek niedawno poznał polecenie ALTER oraz nauczył się dzielić łańcuchy znaków na fragmenty, zatem zdecydował się dodać do swojej tabeli kolejne kolumny na zainteresowania i w każdej z nich umieścić tylko po jednym zainteresowaniu. Doszedł do wniosku, że cztery kolumny na zainteresowania powinny wystarczyć.

Zaostrz ołówek



Korzystając z polecenia ALTER oraz funkcji SUBSTRING_INDEX, zmodyfikuj tabelę Grześka w taki sposób, by miała następujące kolumny:

**id_kontaktu
nazwisko
imie
telefon
email
plec
data_urodzenia
zawod
miejscowosc
lokalizacja
stan
zainteresowanie1
zainteresowanie2
zainteresowanie3
zainteresowanie4
szuka**

→ Odpowiedzi znajdziesz na stronie 370

Zaczynamy od początku

Grześkowi jest przykro z powodu nieudanego spotkania Roberta, dlatego ma zamiar spróbować jeszcze raz. Zaczyna od ponownego przejrzenia danych z rekordu dotyczącego Roberta:

id_kontaktu: 341
nazwisko: Kraszkiewicz
imie: Robert
telefon: 583490972
email: rob_olo@samotnyzejman.pl
plec: M
data_urodzenia: 20.03.1955
zawod: komik
lokalizacja: Gdańsk
województwo: PM
stan: samotny
zainteresowanie1: zwierzęta
zainteresowanie2: papiery wartościowe
zainteresowanie3: geografia
zainteresowanie4: NULL
szuka: samotnej partnerki

Cztery kolumny na zainteresowania, dostępne w zmodyfikowanej tabeli Grześka.



Ćwiczenie

Grzesiek znowu pisze zapytanie, które ma pomóc Robertowi w odszukaniu odpowiedniej kandydatki na randkę. Włożył w nie całą swoją wiedzę, starając się odszukać osobę, która będzie idealnie pasowała do Roberta. Grzesiek zaczyna od prostych kolumn — płci, stanu, województwa, daty urodzenia i informacji o tym, kogo lub czego poszkuje dana osoba; dopiero potem chce przejść do kolumn zainteresowań.

Poniżej napisz zapytanie, które próbuje utworzyć Grzesiek.



Rozwiążanie ćwiczenia

Grzesiek znowu pisze zapytanie, które ma pomóc Robertowi w odszukaniu odpowiedniej kandydatki na randkę. Włożył w nie całą swoją wiedzę, starając się odszukać osobę, która będzie idealnie pasowała do Roberta. Grzesiek zaczyna od prostych kolumn — płci, stanu, województwa, daty urodzenia i informacji o tym, kogo lub czego poszkuje dana osoba; dopiero potem chce przejść do kolumn zainteresowań.

Poniżej napisz zapytanie, które próbuje utworzyć Grzesiek.

```
SELECT * FROM moje_kontakty
```

```
WHERE plec = 'K'  
AND stan = 'samotna'  
AND wojewodztwo = 'PM'  
AND szuka LIKE '%samotnego%'  
AND data_urodzenia > '1950-20-03'  
AND data_urodzenia < '1960-20-03'  
AND  
(
```

```
zainteresowanie1 = 'zwierzęta'  
OR zainteresowanie2 = 'zwierzęta'  
OR zainteresowanie3 = 'zwierzęta'  
OR zainteresowanie4 = 'zwierzęta'
```

```
)  
AND  
(
```

```
zainteresowanie1 = 'papiery wartościowe'  
OR zainteresowanie2 = 'papiery wartościowe'  
OR zainteresowanie3 = 'papiery wartościowe'  
OR zainteresowanie4 = 'papiery wartościowe'
```

```
)  
AND  
(
```

```
zainteresowanie1 = 'geografia'  
OR zainteresowanie2 = 'geografia'  
OR zainteresowanie3 = 'geografia'  
OR zainteresowanie4 = 'geografia'
```

```
);
```

Robert chce spotykać się z kobietą urodzoną między rokiem 1950 a 1960, która mieszka w województwie pomorskim i chce się spotykać z samotnym mężczyzną.

Grzesiek musi sprawdzić wszystkie cztery kolumny zainteresowań, by ustalić, czy pasują one do zainteresowań Roberta; teraz bowiem każde zainteresowanie może być zapisane w jednej z czterech kolumn.

W czwartej kolumnie zainteresowań Roberta była wartość NULL, zatem nie sprawdzamy czwartego zainteresowania.

Wszystko stracone...

Dodanie nowych kolumn nie przyczyniło się do rozwiązania podstawowego problemu Grześka — okazało się, że struktura tabeli dalej nie umożliwia tworzenia prostych zapytań. Żadna z dotychczasowych wersji tabeli nie spełniała wymogu zawierania danych atomowych.

A wydawało się, że to będzie takie dobre rozwiązanie. Jednak sprawiło, że zapytania stały się jeszcze bardziej złożone.

DO ZROBIENIA:

Nаписать запрос для Влеска: Chodzi o napisanie zapytania operującego na kolumnie zainteresowań. Niewygląda to najlepiej, trzeba będzie użyć operatora **LIKE**... całeszczęście, że nie muszę się tak męczyć zbyt często...

W przyszłości powiniem ignorować kolumnę zainteresowań, żeby sobie ułatwić pisanie zapytań.

Uwzględniać tylko pierwsze zainteresowanie, ignorując całą pozostałą zawartość kolumny.

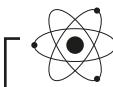
Stworzyć kilka kolumn, w których będzie się zapisywać po jednym zainteresowaniu, a to dlatego, że trzymanie wszystkich zainteresowań w jednej kolumnie utrudnia pisanie zapytań.



...ale zaraz



A czy moglibyśmy stworzyć tabelę, która zawierałaby wyłącznie zainteresowania? Czy to by coś pomogło?



WYSIL SZARE KOMÓRKI

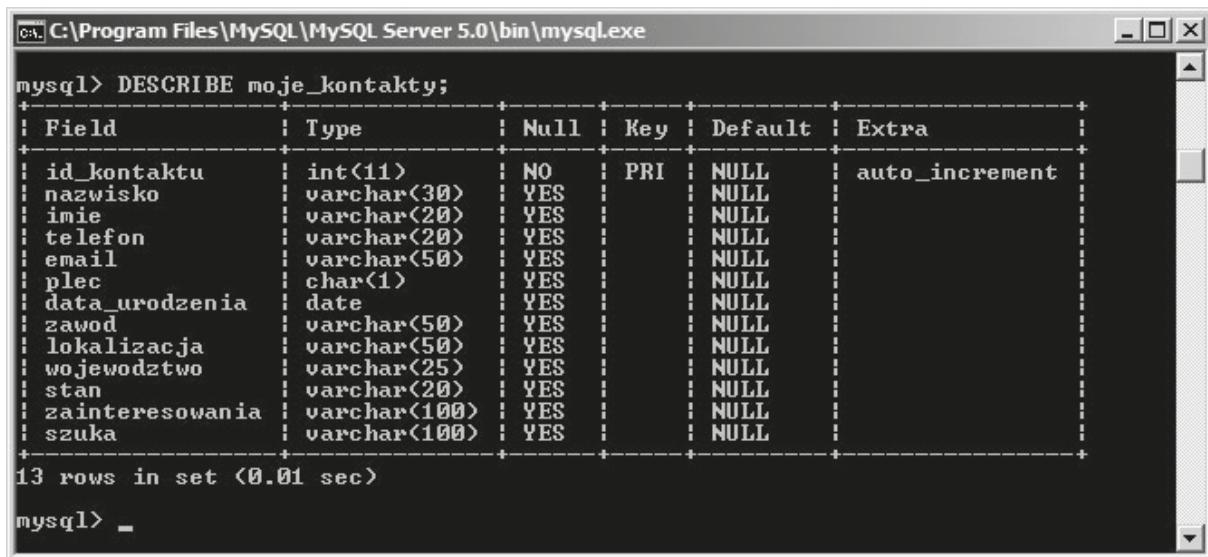
Czy dodanie nowej tabeli mogłoby coś pomóc? W jaki sposób moglibyśmy połączyć dane w obu tabelach?

Wyobraź sobie dodatkową tabelę

Już wiemy, że korzystając z jednej, dotychczasowej tabeli, nie jesteśmy w stanie stworzyć żadnego dobrego rozwiązania. Już kilka razy próbowaliśmy naprawiać dane, a nawet zmieniać strukturę tabeli — ale wszystko na nic.

Potrzebujemy zatem czegoś nowego — musimy przejść poza ograniczenia narzucone przez jedną tabelę. To, czego nam tak naprawdę potrzeba, to **więcej tabel**, które mogłyby *współpracować* z bieżącą tabelą i które pozwoliłyby nam **skojarzyć każdą osobę z więcej niż jednym zainteresowaniem**. Poza tym to rozwiązanie pozwoliłoby nam zachować już istniejące dane.

Musimy usunąć z dotychczasowej tabeli nieatomowe dane i przenieść je do nowych tabel.



The screenshot shows a terminal window titled 'C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe'. The command 'DESCRIBE moje_kontakty;' is entered, and the resulting table structure is displayed:

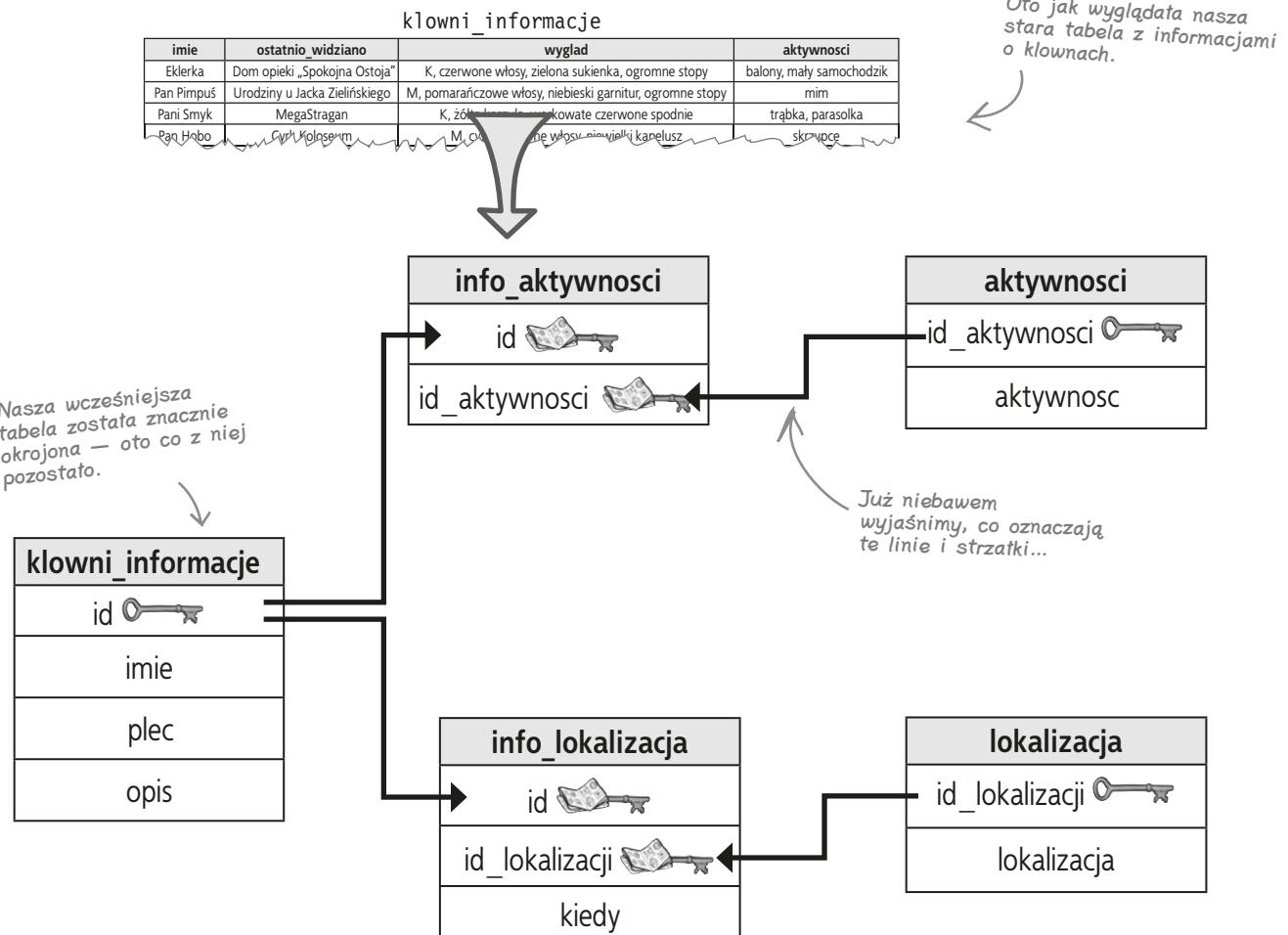
Field	Type	Null	Key	Default	Extra
id_kontaktu	int<11>	NO	PRI	NULL	auto_increment
nazwisko	varchar<30>	YES		NULL	
imie	varchar<20>	YES		NULL	
telefon	varchar<20>	YES		NULL	
email	varchar<50>	YES		NULL	
plec	char<1>	YES		NULL	
data_urodzenia	date	YES		NULL	
zawod	varchar<50>	YES		NULL	
lokalizacja	varchar<50>	YES		NULL	
województwo	varchar<25>	YES		NULL	
stan	varchar<20>	YES		NULL	
zainteresowania	varchar<100>	YES		NULL	
szuka	varchar<100>	YES		NULL	

13 rows in set (0.01 sec)

mysql> _

Nowe tabele w bazie danych z informacjami o klonach

Czy pamiętasz naszą tabelę zawierającą informacje o publicznych wystąpieniach kloonów, którą stworzyliśmy w rozdziale 3.? Problem kloonów w Bazodanowie wciąż się potęguje, dlatego też zmodyfikowaliśmy strukturę tej bazy, dodając do niej nowe tabele. Jak się już wkrótce przekonasz, znacznie poprawiło to łatwość korzystania z bazy.



Na kilku kolejnych stronach wyjaśnimy, dlaczego tabele zostały podzielone w taki, a nie inny sposób oraz co oznaczają te wszystkie strzałki i klucze. Kiedy już to wszystko zrozumiesz, zajmiemy się modyfikacją tabeli moje_kontakty Grześka.



Schemat bazy danych kloni_informacje

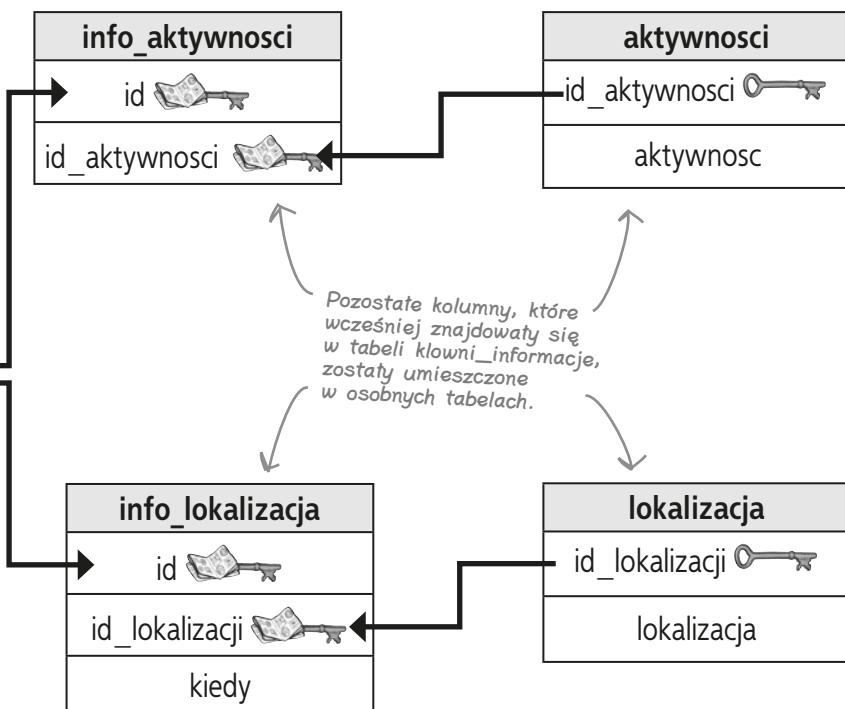
Reprezentacja wszystkich struktur w bazie danych, takich jak tabele, kolumny oraz ich wzajemne połączenia, jest określana mianem **schematu**.

Stworzenie graficznej reprezentacji bazy danych może pomóc w wyobrażeniu sobie wzajemnych powiązań pomiędzy danymi podczas tworzenia zapytań. Niemniej jednak schemat bazy można także zapisać w formie tekstuowej.

imie	ostatnio_widziano	wyglad	aktywnosci
Eklerka	Dom opieki „Spokojna Ostoja”	K, czerwone włosy, zielona sukienka, ogromne stopy	balony, mały samochodzik
Pan Pimpus	Urodziny u Jacka Zielińskiego	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy	mim
Pani Smyk	MegaStragan	K, żółte włosy, skórkowe czerwone spodnie	trąbka, parasolka
Pan Hobo	Czerwony Koloseum	M, żółte włosy, niebieskie spodnie i kapelusz	skrzynce

Nasza wcześniejsza tabela została znacznie okrojona — oto co z niej pozostało.

id
imie
plec
opis



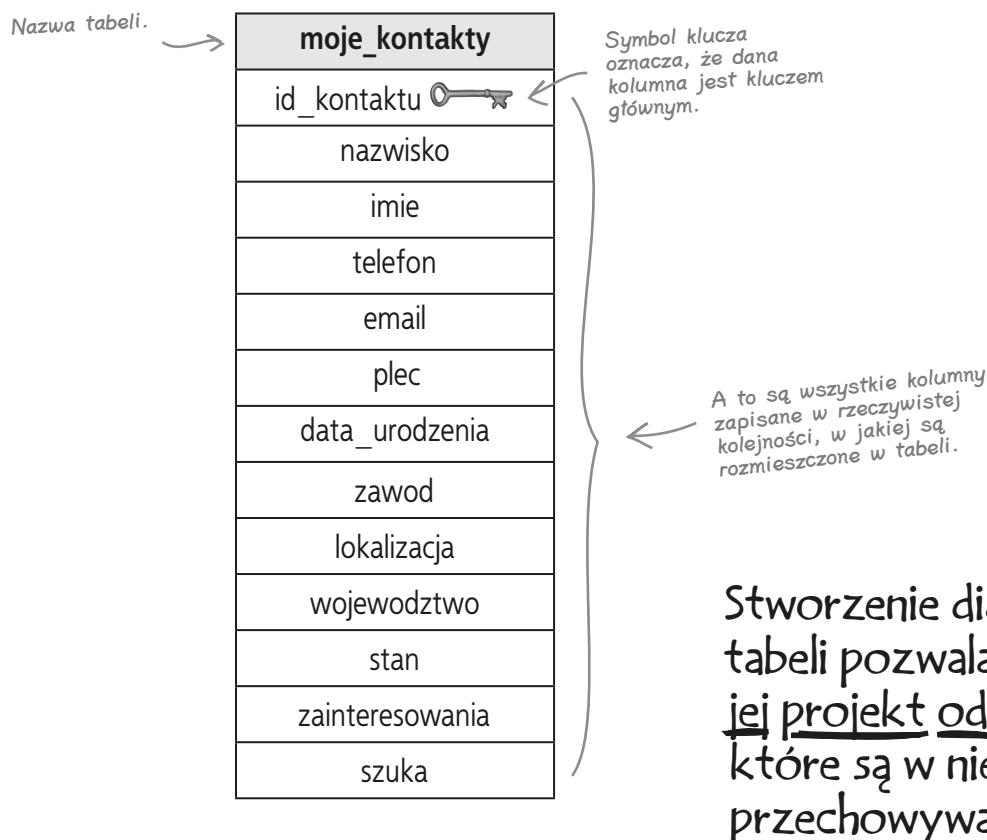
Opis wszystkich danych przechowywanych w bazie (kolumn oraz tabel) wraz z innymi, skojarzonymi z nimi obiektami oraz wszelkimi połączniami pomiędzy nimi nazywa się **SCHEMATEM**.

Łatwiejszy sposób graficznej reprezentacji tabel

Właśnie zobaczyłeś proces przekształcania tabeli z informacjami o wystąpieniach kloonów. Przekonamy się, jak w analogiczny sposób możemy naprawić tabelę `moje_kontakty`.

Do tej pory wszystkie tabele zamieszczone w tej książce były przedstawiane w formie tabelki, w której nazwy kolumn były rozmieszczone w górnym wierszu bądź też w formie kodu wyświetlanego przez polecenie DESCRIBE w oknie konsoli. Oba te sposoby doskonale zdają egzamin w przypadku korzystania z pojedynczej tabeli. Jednak nie są szczególnie wygodne, gdy chcemy stworzyć diagram kilku powiązanych ze sobą tabel.

Poniżej zamieściliśmy zatem nowy, uproszczony sposób rysowania diagramu tabeli:



W jaki sposób z jednej tabeli zrobić dwie

Doskonale wiemy, że aktualna postać kolumny zainteresowań nie ułatwia nam przeszukiwania jej zawartości. Wynika to z faktu, iż w każdym rekordzie w tej kolumnie może być zapisana większa ilość wartości. Co więcej, nawet rozdzielenie tych danych i zapisanie ich w kilku kolumnach nie pozwoliło nam uprościć zapytań.

Obok przedstawiliśmy aktualną postać tabeli `moje_kontakty` Grześka. Kolumna `zainteresowania` nie jest atomowa i okazuje się, że istnieje tylko jedno dobre rozwiązanie związanego z nią problemu: musimy przekształcić ją w osobną tabelę, która będzie zawierać poszczególne zainteresowania.

Zaczniemy od narysowania kilku diagramów, które pokażą, jak mogłyby wyglądać nowe tabele Grześka. Jednak do momentu opracowania nowego schematu bazy danych nie będziemy w żaden sposób zmieniać samej tabeli `moje_kontakty` ani zapisanych w niej danych.

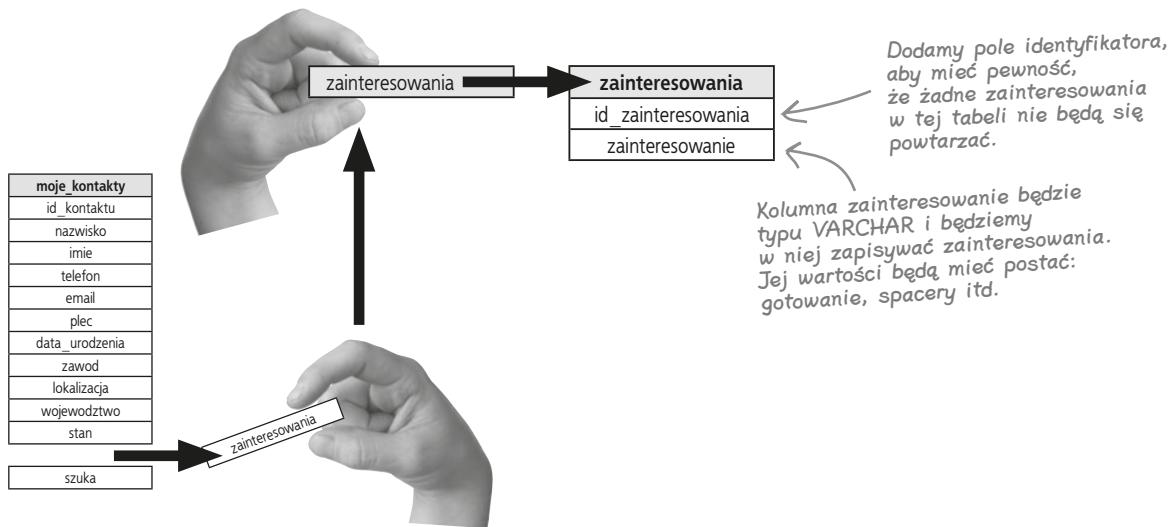
Oto tabela
`moje_kontakty`.
Nie jest to
niestety tabela
atomowa.

moje_kontakty
id_kontaktu
nazwisko
imie
telefon
email
plec
data_urodzenia
zawod
lokalizacja
województwo
stan
zainteresowania
szuka

1

Usuwamy kolumnę zainteresowania i umieszczamy ją w osobnej tabeli.

W tym kroku usuwamy kolumnę zawierającą informacje o zainteresowaniach i umieszczamy ją w nowej tabeli.



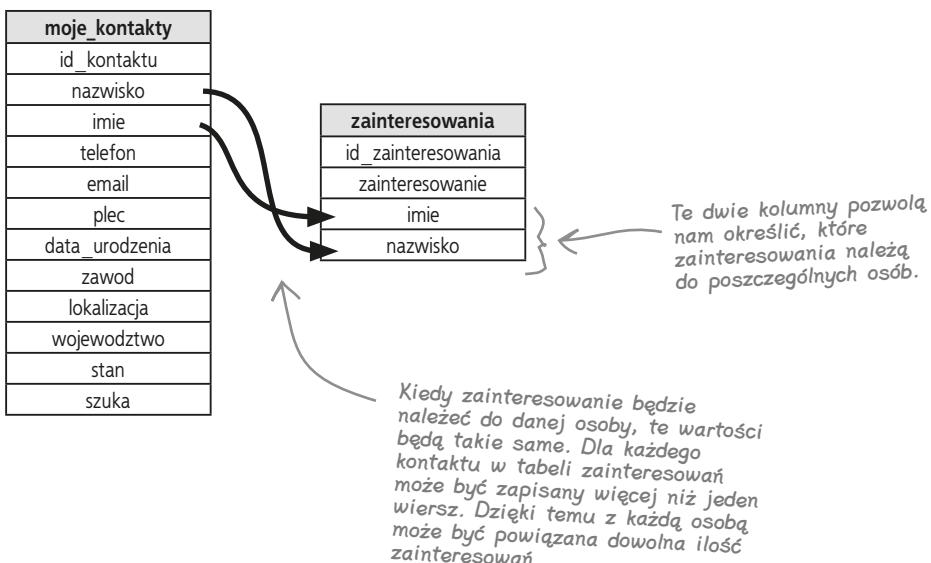
Nasza nowa tabela będzie zawierać wszystkie zainteresowania, jakie były zapisane w tabeli `moje_kontakty`, przy czym każde zainteresowanie zostanie zapisane w osobnym wierszu.

2

Dodajemy kolumny, które pozwolą nam określić, jakie zainteresowania należą do poszczególnych osób w tabeli moje_kontakty.

Usunęliśmy zainteresowania z tabeli moje_kontakty, jednak obecnie nie mamy możliwości określenia, do której osoby należą poszczególne zainteresowania. Musimy zatem w jakiś sposób połączyć obie tabele; możemy to zrobić, umieszczając w tabeli zainteresowań jakieś informacje z tabeli moje_kontakty.

Jednym z możliwych rozwiązań jest umieszczenie w tabeli zainteresowań kolumn nazwisko i imię z tabeli moje_kontakty.



**WYSIL
SZARE KOMÓRKI**

Pomysł jest bardzo dobry, jednak wybór imienia i nazwiska jako kolumn służących do połączenia obu tabel nie jest najlepszy.

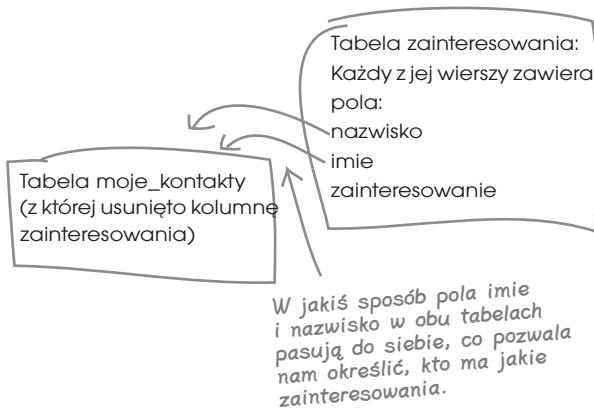
Czy potrafisz wyjaśnić dlaczego?

Dodawania połączeń do diagramów

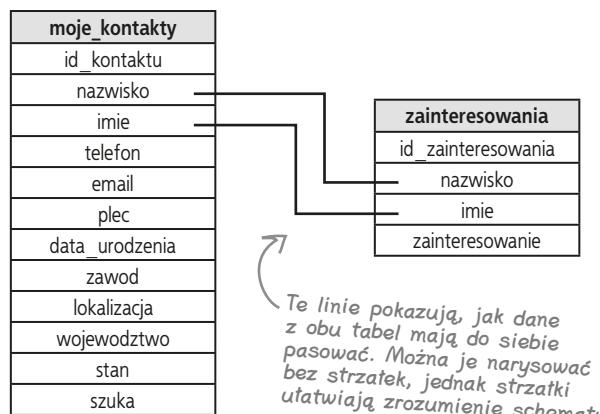
Łączenie tabel

Przyjrzyjmy się teraz dokładniej naszemu pomysłowi modyfikacji tabeli `moje_kontakty`.

Oto nasz początkowy szkic:

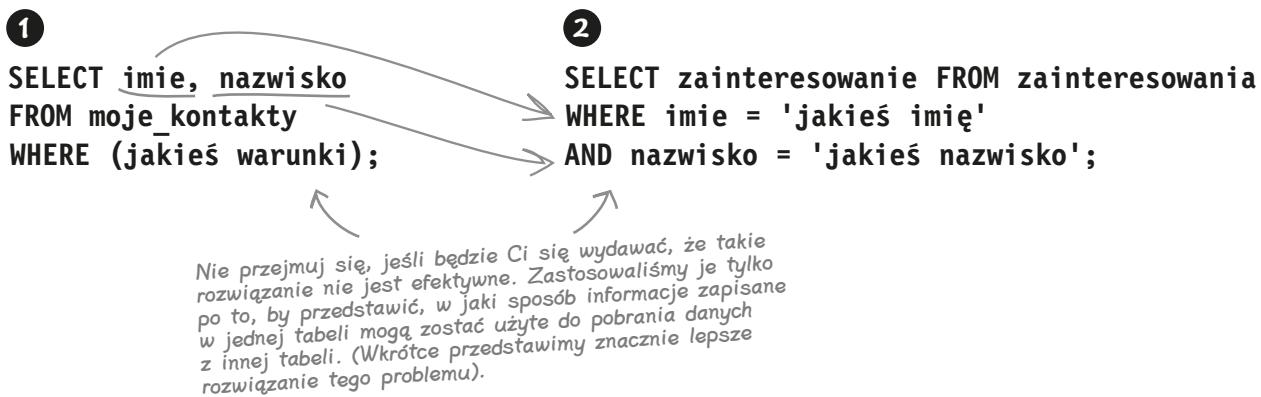


A oto nasz nowy schemat:



Zwróć uwagę, jak linie pomiędzy oboma tabelami załamują się, pokazując, które dane w obu tabelach mają do siebie pasować. Schemat pozwala nam uporządkować nasz szkic w taki sposób, że każdy programista SQL zrozumie znaczenie poszczególnych symboli.

A poniżej przedstawiliśmy polecenia SELECT, które pozwolą nam wykorzystać dane przechowywane w obu tabelach.

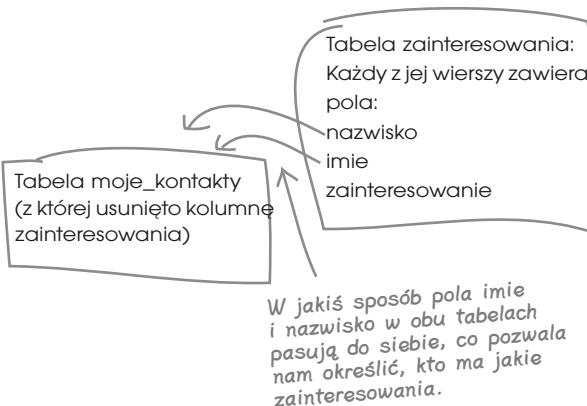


Zaostrz ołówek



Na pustej stronie poniżej naszkicuj kolejne pomysły na dodanie do bazy danych Grześka nowych tabel, które pomogłyby nam śledzić zainteresowania zarejestrowanych osób.

Nie zwracaj sobie głowy estetyką i nie staraj się rysować schematu — pamiętaj, aktualnie zajmujemy się jedynie ideą. Jedną z tych idei narysowaliśmy już za Ciebie, ale musisz ją jeszcze nieco poprawić.



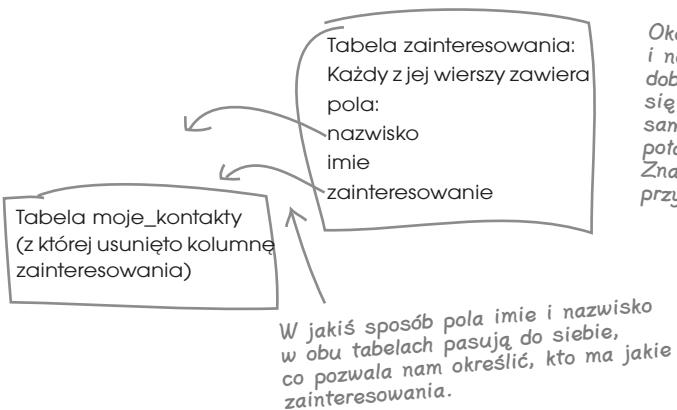
Zaostrz ołówek

Rozwiązywanie



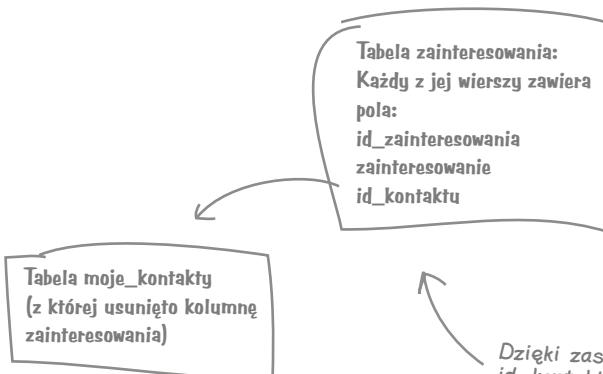
Na pustej stronie poniżej naszkicuj kolejne pomysły na dodanie do bazy danych Grześka nowych tabel, które pomogłyby nam śledzić zainteresowania zarejestrowanych osób.

Nie zwracaj sobie głowy estetyką i nie staraj się rysować schematu — pamiętaj, aktualnie zajmujemy się jedynie ideą. Jedną z tych idei narysowaliśmy już za Ciebie, ale musisz ją jeszcze nieco poprawić.



Okazuje się jednak, że zastosowanie imienia i nazwiska do połączenia obu tabel nie jest takim dobrym rozwiązaniem. W tabeli moje_kontakty może się bowiem znaleźć więcej osób, które mają takie same imię i nazwisko, a to oznacza, że moglibyśmy połączyć je z niewłaściwymi zainteresowaniami. Znacznie lepszym pomysłem będzie połączenie tabel przy wykorzystaniu kolumny klucza głównego.

Zamiast stosować kolumny imie i nazwisko, których wartości mogą się powtarzać, możemy połączyć obie tabele przy wykorzystaniu kolumny id_kontaktu:



Dzięki zastosowaniu kolumny id_kontaktu będziemy mogli używać naprawdę unikalnych wartości. Wiemy bowiem, że zainteresowania, którym przypisano konkretną wartość id_kontaktu, na pewno są powiązane z prawidłowym wierszem tabeli moje_kontakty.

Łączenie tabel

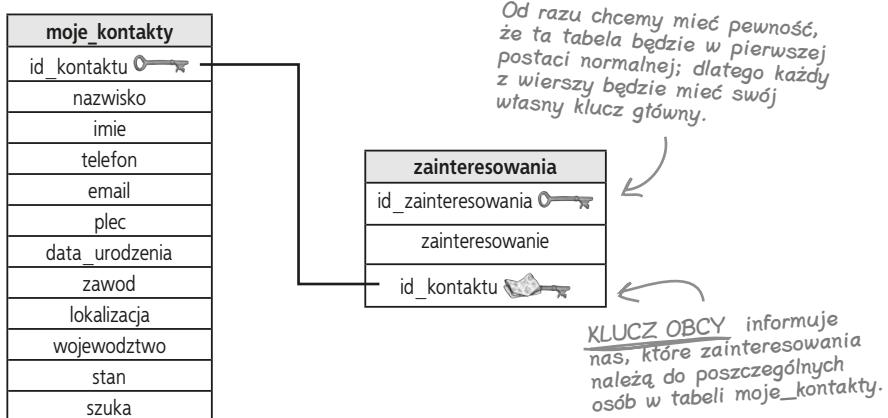
W naszym pierwszym szkicu połączonych tabel pojawił się pewien problem.

Wynikał on z faktu, że chcieliśmy połączyć obie tabele przy wykorzystaniu pól imię i nazwisko. A co by się stało w przypadku, gdyby dwie różne osoby zapisane w tabeli moje_kontakty miały takie samo imię i nazwisko?



Jak widać, do połączenia obu tabel musimy zastosować kolumnę o **unikalnych** wartościach. Na szczęście już wcześniej zaczęliśmy normalizować tabelę moje_kontakty, dzięki czemu teraz mamy już w niej taką kolumnę, jest nią **klucz główny**.

Możemy zatem użyć wartości z kolumny klucza głównego tabeli moje_kontakty i zapisywać je w jakiejś kolumnie tabeli zainteresowania. Co więcej, na podstawie tej tabeli będziemy dokładnie wiedzieć, które zainteresowania należą do poszczególnych osób zapisanych w tabeli moje_kontakty. Taka kolumna jest określana jako **klucz obcy**.



KLUCZEM OBCYM
nazywamy w tabeli kolumnę, która odwołuje się do **KLUCZA GŁÓWNEGO** innej tabeli.

Kilka faktów dotyczących klucza obcego



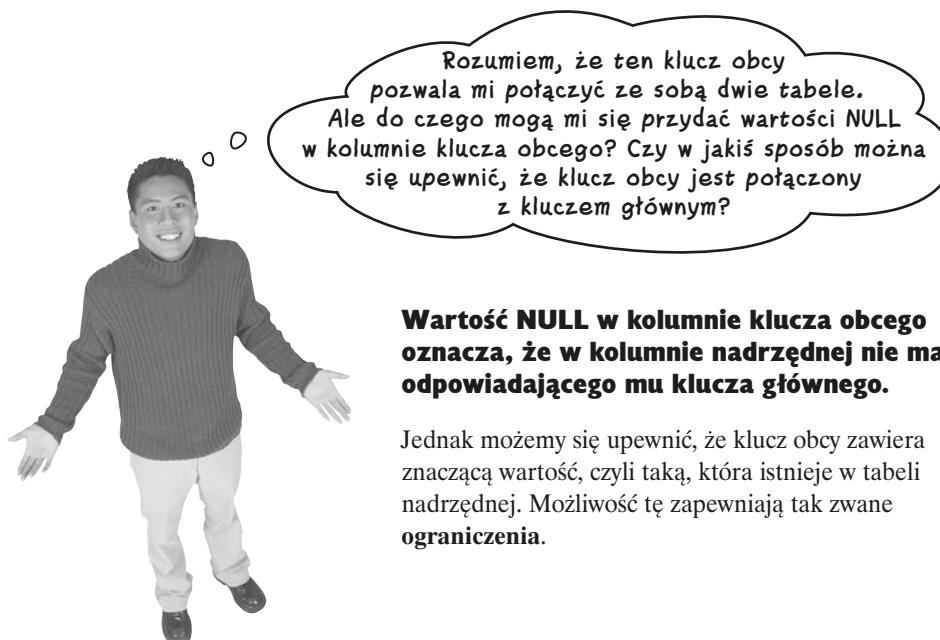
Kolumna klucza obcego może mieć całkowicie inną nazwę niż klucz główny, od którego pochodzi.

Klucz główny używany przez klucz obcy jest także nazywany kluczem nadziednym, a tabela, z której pochodzi — tabelą nadziedną.

Klucza obcego można także używać, by mieć pewność, że wiersze w jednej tabeli będą miały odpowiadające im wiersze w innej tabeli.

W kolumnie klucza obcego mogą się pojawiać wartości NULL, choć nie mogą one występować w kolumnie klucza głównego.

Wartości w kolumnie klucza obcego nie muszą być unikalne — a co więcej, w praktyce często się one powtarzają.



Ograniczanie klucza obcego

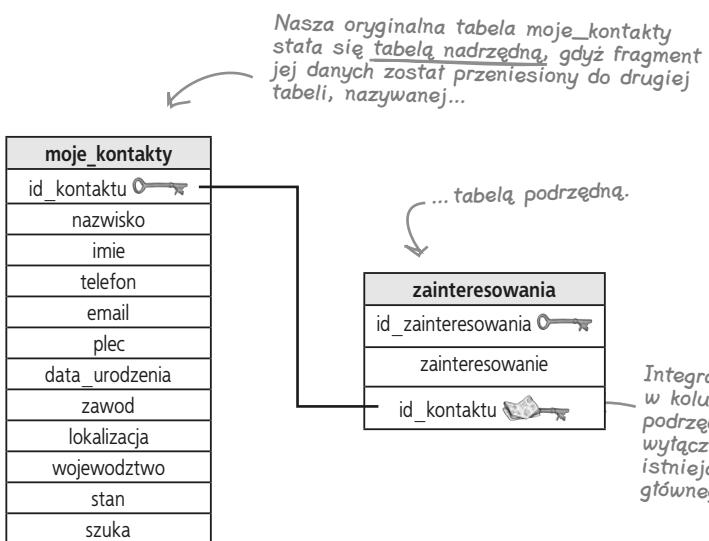
Choć mógłbyś tak po prostu utworzyć tabelę i umieścić w niej kolumnę, która działałaby jako klucz obcy, to jednak kolumna staje się prawdziwym kluczem obcym, dopiero gdy ją w odpowiedni sposób zadeklarujemy w poleceniu CREATE TABLE lub ALTER TABLE. Klucz jest tworzony w obrębie struktury nazywanej ograniczeniem.

↑
Wyobraź sobie OGRANICZENIE
jako regułę, którą nasza tabela
musi spełniać.

W kolumnie klucza obcego będziesz mógł zapisywać tylko wartości istniejące w tabeli, z której dany klucz pochodzi — czyli z tabeli nadzędnej. Wymóg ten jest nazywany integralnością odwołań.

Utworzenie KLUCZA OBCEGO jako ograniczenia w tabeli daje bardzo duże korzyści.

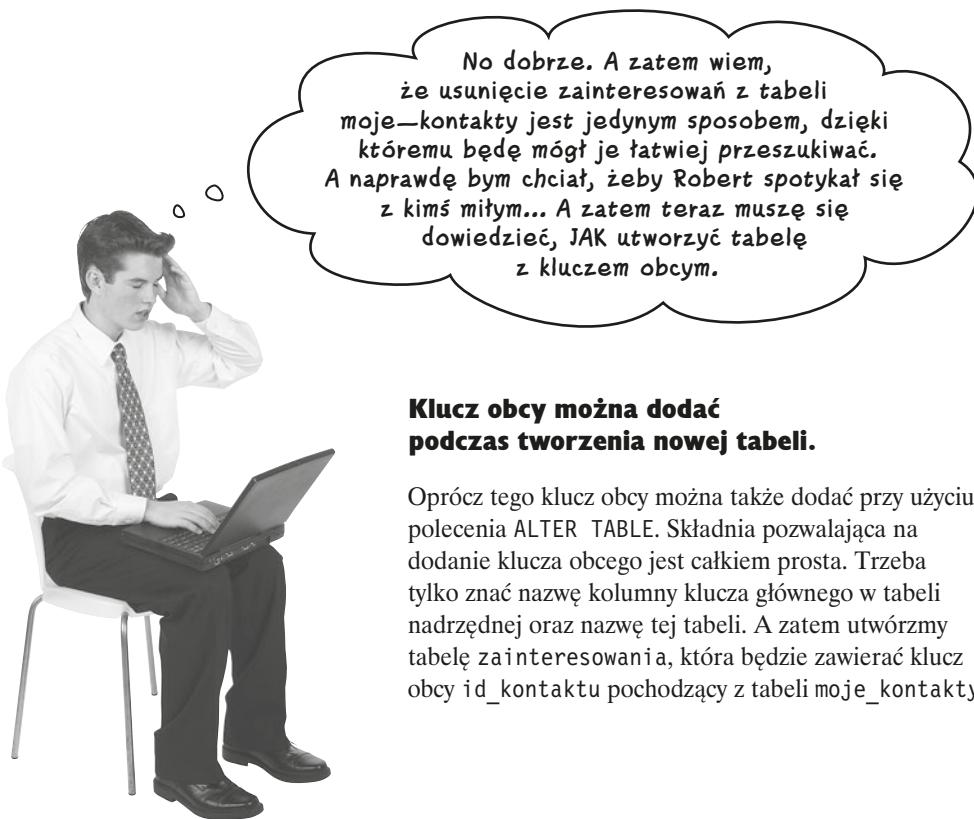
Jeśli reguła integralności odwołań zostanie złamana, wystąpią błędy, które uniemożliwią przypadkowe zapisanie w tabeli nieprawidłowych danych.



Można użyć klucza obcego, by odwołać się do unikalnej wartości w tabeli nadzędnej.

Nie musi to być klucz główny tabeli nadzędnej, jednak musi to być wartość unikalna.

Dlaczego należy zwracać sobie głowę kluczami obcymi?



Klucz obcy można dodać podczas tworzenia nowej tabeli.

Oprócz tego klucz obcy można także dodać przy użyciu polecenia ALTER TABLE. Składnia pozwalająca na dodanie klucza obcego jest całkiem prosta. Trzeba tylko znać nazwę kolumny klucza głównego w tabeli nadrzednej oraz nazwę tej tabeli. A zatem utworzymy tabelę zainteresowania, która będzie zawierać klucz obcy id_kontaktu pochodzący z tabeli moje_kontakty.

Nie istnieją
grupie pytania

P: Kiedy już usunę zainteresowania z tabeli moje_kontakty, to w jaki sposób będę mógł je przeszukiwać i pobierać?

O: Tym zajmiemy się w następnym rozdziale. Podczas lektury przekonasz się, że tworzenie zapytań pobierających informacje z kilku tabel jest w rzeczywistości całkiem proste. Jednak jak na razie musimy dokończyć modyfikowanie tabeli moje_kontakty, tak by tworzenie zapytań było proste i efektywne.

TWORZENIE tabeli z KLUCZEM OBCYM

Teraz, kiedy już wiesz, dlaczego powinieneś stworzyć klucz obcy z ograniczeniem, pokażemy Ci, jak możesz to zrobić. Zwróć uwagę na zastosowanie słowa kluczowego CONSTRAINT definiującego ograniczenie, dzięki któremu będziemy w stanie określić, z jakiej tabeli pochodzi klucz.

Dodanie słów kluczowych PRIMARY KEY na końcu wiersza, w którym jest definiowana kolumna klucza głównego, jest kolejnym (najszyszym) sposobem utworzenia tego klucza.

CREATE TABLE zainteresowania (

id_zainteresowania INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
zainteresowanie VARCHAR(50) NOT NULL,

Klucz obcy tworzymy dokładnie w taki sam sposób, w jaki tworzylibyśmy każdą inną kolumnę pełniąca rolę klucza: ma to być kolumna typu INT, w której nie są dopuszczane wartości NULL (NOT NULL).

id_kontaktu INT NOT NULL,

CONSTRAINT moje_kontakty_id_kontaktu_fk

FOREIGN KEY (id_kontaktu)

REFERENCES moje_kontakty (id_kontaktu);

)
 Ta nazwa określa, z jakiej tabeli pochodzi klucz obcy...

... oraz jaką ma nazwę kolumna w tej drugiej tabeli.

Nazwę OGRANICZENIA określamy w taki sposób, by zawierała informacje o tabeli, z której pochodzi klucz główny (moje_kontakty), nazwie klucza (id_kontaktu) oraz o tym, iż jest to klucz obcy (fk, od ang. foreign key).

Jeśli później zmienimy zdanie, to modyfikując tabelę, będziemy się musieli postugiwać tą nazwą. Ten wiersz kodu jest opcjonalny, niemniej jednak stosowanie go uważa się za przejaw dobrej praktyki programistycznej.

Nazwa kolumny podana w nawiasach określa kolumnę, która będzie kluczem obcym. Może to być dowolna nazwa.



Ćwiczenie

Spróbuj wykonać powyższe polecenie. Otwórz okno konsoli i wpisz powyższy kod SQL, by stworzyć swoją własną wersję tabeli zainteresowań.

Kiedy już ją utworzysz, przeanalizuj uważnie jej strukturę. Jakie nowe informacje pozwolą Ci zorientować się, że faktycznie zostały utworzone ograniczenie i klucz obcy?



Rozwiązywanie ćwiczenia

Spróbuj wykonać powyższe polecenie. Otwórz okno konsoli i wpisz powyższy kod SQL, by stworzyć swoją własną wersję tabeli zainteresowań.

Kiedy już ją utworzysz, przeanalizuj uważnie jej strukturę. Jakie nowe informacje pozwolą Ci się zorientować, że faktycznie zostały utworzone ograniczenie i klucz obcy?

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> DESCRIBE zainteresowania;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id_zainteresowania | int<10> unsigned | NO | PRI | NULL | auto_increment |
| zainteresowanie | varchar<50> | NO | | | |
| id_kontaktu | int<10> unsigned | NO | MUL | | |
+-----+-----+-----+-----+-----+-----+
3 rows in set <0.01 sec>

mysql>
```

MUL oznacza, że wartości w kolumnie mogą się wielokrotnie powtarzać. To właśnie dzięki temu będziemy w stanie zarządzać wieloma zainteresowaniami dla każdego z rekordów tabeli moje_kontakty.

Nie istniejąca grupa pytania

P: **Zadaliście sobie naprawdę dużo trudu, by utworzyć ograniczenie i klucz obcy. Ale właściwie po co? Czy nie można by po prostu używać klucza z innej tabeli i nazwać go kluczem obcym bez definiowania tego całego ograniczenia?**

O: Można by. Jednak dzięki utworzeniu ograniczenia będziesz mógł zapisywać w kolumnie klucza obcego wyłącznie takie wartości, które już istnieją w tabeli nadzędnej. A to kolej wymusi powiązanie pomiędzy obydwoema tabelami.

P: **A co oznacza wyrażenie „wymusi powiązanie”?**

O: Ograniczenie klucza głównego wymusza integralność odwołań (innymi słowy, ograniczenie zapewnia, że jeśli w jednej tabeli istnieje rekord zawierający klucz obcy, to w drugiej tabeli będzie istniał odpowiadający mu rekord z taką samą wartością klucza

głównego). Jeśli spróbujesz usunąć rekord w tabeli zawierającej klucz główny lub zmienisz wartość klucza głównego oraz jeśli wartość klucza głównego z tego rekordu jest zapisana w tabeli podrzędnej w kolumnie klucza obcego, to zostanie zgłoszony błąd.

P: **Czy to oznacza, że nigdy nie będę mógł usunąć z tabeli moje_kontakty wiersza, jeśli wartość jego klucza głównego będzie występowała w tabeli zainteresowania jako klucz obcy?**

O: Będziesz mógł, z tym że wcześniej będziesz musiał usunąć wiersz klucza obcego. W końcu, jeśli usuniesz osobę z tabeli moje_kontakty, to nie będą Cię już interesowały jej zainteresowania.

P: **Ale kogo to interesuje, że w mojej tabeli zainteresowań będą zapisane dwa nikomu niepotrzebne wiersze?**

O: To spowalnia bazę. Takie wiersze są nazywanie „sierotami”, a wraz z upływem czasu może się ich zbierać bardzo dużo. Nie są one do niczego przydatne, a jedynie sprawiają, że baza musi przeszukiwać więcej informacji, co wydłuża czas wykonywania zapytań.

P: **No dobrze, przekonaliście mnie. Czy są jeszcze jakieś inne ograniczenia, oprócz klucza obcego?**

O: Tak, na przykład klucz główny, który już znasz. Oprócz tego zastosowanie (podczas tworzenia kolumny) słowa kluczowego UNIQUE także jest uznawane za ograniczenie. Istnieje także pewien typ ograniczeń, niedostępny w bazie danych MySQL, nazywany ograniczeniem CHECK. Pozwala on na określenie warunku, który musi być spełniony, by wartość mogła zostać zapisana w kolumnie. Powinieneś sprawdzić dokumentację używanego systemu zarządzania bazami danych SQL, by sprawdzić, czy udostępnia on ograniczenia tego typu.

Zależności pomiędzy tabelami

Teraz już wiemy, jak połączyć tabele przy użyciu klucza obcego, niemniej wciąż musimy brać pod uwagę sposób, w jaki tabele są ze sobą połączone. W przypadku tabeli `moje_kontakty` problem polegał na tym, że musielibyśmy powiązać wiele osób z wieloma zainteresowaniami.

W przypadku łączenia danych można wyróżnić trzy wzorce, które będą się cały czas powtarzać: **jeden-do-jednego**, **jeden-do-wielu** oraz **wiele-do-wielu**. A kiedy uda Ci się już określić wzorzec, do którego pasują Twoje dane, określenie projektu tabel — czyli **schematu** — stanie się bardzo proste.

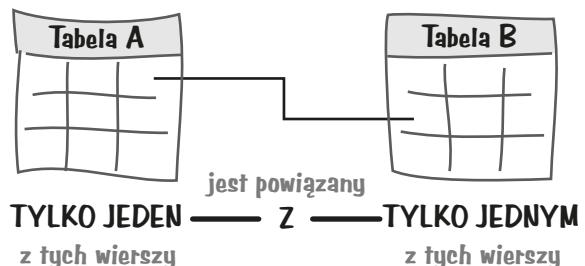
Wzorce danych: jeden-do-jednego

Przyjrzymy się pierwszemu z wzorców: **jeden-do-jednego** i zobaczymy, jak można go stosować. W tym przypadku jednemu rekordowi z tabeli A może odpowiadać tylko jeden rekord z tabeli B.

Przymijmy zatem, że tabela A zawiera imię, a tabela B szczegółowe informacje o zarobkach i numerze ubezpieczenia, które zostały *oddzielone* od pozostałych danych w celu ich lepszego zabezpieczenia.

Dane w obu tabelach będą zawierały Twój identyfikator, dzięki czemu dostaniesz czek na właściwą sumę. Kluczem głównym tabeli nadrzednej jest kolumna `id_pracownika`, natomiast w tabeli podrzędnej ta sama kolumna pełni rolę klucza obcego.

Na poniższym schemacie obie tabele połączono zwykłą linią łamana, co oznacza, że łączymy jeden rekord z pierwszej tabeli z jednym rekordem z drugiej tabeli.



Każda osoba w tabeli pracowników może mieć tylko jeden numer ubezpieczenia, a konkretny numer ubezpieczenia może być przydzielony tylko jednej osobie. Jedna osoba i jeden numer ubezpieczenia oznacza, że tworzą one zależność typu jeden-do-jednego.

pracownicy	zarobki																								
<table border="1"> <thead> <tr> <th>id_pracownika</th> <th>imie</th> <th>nazwisko</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Beyonce</td> <td>Knowles</td> </tr> <tr> <td>2</td> <td>Shaw</td> <td>Carter</td> </tr> <tr> <td>3</td> <td>Shakira</td> <td>Ripoll</td> </tr> </tbody> </table>	id_pracownika	imie	nazwisko	1	Beyonce	Knowles	2	Shaw	Carter	3	Shakira	Ripoll	<table border="1"> <thead> <tr> <th>nr_ubezp</th> <th>poziom_zarobkow</th> <th>id_pracownika</th> </tr> </thead> <tbody> <tr> <td>234567891</td> <td>2</td> <td>6</td> </tr> <tr> <td>345678912</td> <td>5</td> <td>35</td> </tr> <tr> <td>123456789</td> <td>7</td> <td>1</td> </tr> </tbody> </table>	nr_ubezp	poziom_zarobkow	id_pracownika	234567891	2	6	345678912	5	35	123456789	7	1
id_pracownika	imie	nazwisko																							
1	Beyonce	Knowles																							
2	Shaw	Carter																							
3	Shakira	Ripoll																							
nr_ubezp	poziom_zarobkow	id_pracownika																							
234567891	2	6																							
345678912	5	35																							
123456789	7	1																							

Także te tabele tworzą zależność typu jeden-do-jednego, gdyż klucz główny tabeli pracowników, `id_pracownika`, zostanie użyty jako klucz obcy tabeli zarobków.

Wzorce danych: kiedy używać tabel połączonych zależnością jeden-do-jednego



A zatem, czy wszystkie dane, między którymi występują zależności typu jeden-do-jednego, powinniśmy umieszczać w nowych tabelach?

Właściwie nie. Tabele, między którymi występuje zależność jeden-do-jednego, nie są używane aż tak często.

Tak naprawdę nie ma zbyt wielu powodów, by tworzyć tabele, między którymi występuje zależność jeden-do-jednego.

Kiedy używać zależności jeden do jednego?

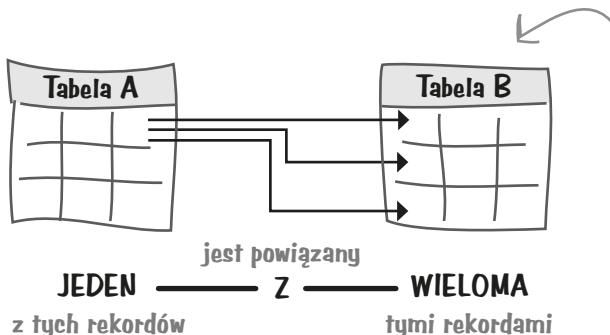
W przeważającej większości przypadków najbardziej sensownym rozwiązaniem jest pozostawienie danych, pomiędzy którymi występują zależności typu jeden-do-jednego, w jednej, głównej tabeli. Niemniej jednak rozwiązanie polegające na rozdzieleniu kolumn powiązanych zależnością tego typu i zapisaniu ich w osobnych tabelach czasami może mieć pewne zalety:

1. Wydzielenie części danych może Ci pozwolić na tworzenie szybszych zapytań. Jeśli na przykład musisz często wyszukiwać numer ubezpieczenia, to dzięki rozdzieleniu danych będziesz mógł przeszukiwać mniejszą tabelę.
2. Jeśli tabela zawiera kolumnę, której wartości jeszcze nie znasz, możesz ją wyizolować i uniknąć zapisywania w głównej tabeli niewygodnych wartości NULL.
3. Możesz starać się ochronić część swoich danych poprzez utrudnienie dostępu do nich. W takich przypadkach wyizolowanie danych może Ci pomóc w ograniczeniu dostępu do nich. Jeśli na przykład dysponujesz tablicą pracowników, to możesz zdecydować się na przechowywanie informacji o wysokości ich zarobków w osobnej tabeli.
4. Jeśli przechowujesz duże ilości danych, na przykład zapisując je w kolumnie typu BLOB, to możesz zdecydować się na przeniesienie jej do osobnej tabeli.

Zależności typu jeden-do-jednego: dokładnie jeden wiersz tabeli nadzędnej jest powiązany z jednym wierszem tabeli podrzędnej.

Wzorce danych: jeden-do-wielu

Zależność typu jeden-do-wielu oznacza, że z jednym wierszem w tabeli A może być powiązanych **wiele** wierszy w tabeli B, jednak jeden wiersz w tabeli B będzie powiązany **tylko z jednym** wierszem z tabeli A.



Jeden rekord z tabeli A może być powiązany z WIELOMA rekordami z tabeli B, jednak każdy rekord z tabeli B może być powiązany tylko z jednym rekordem z tabeli A.

Jeden-do-wielu:
rekord z tabeli
A może być
powiązany
z WIELOMA
rekordami z tabeli B,
jednak rekord
z tabeli B może
być powiązany
tylko z JEDNYM
rekordem z tabeli A.

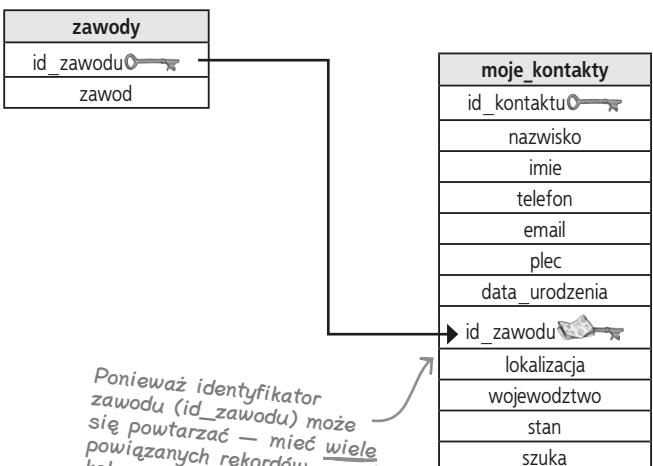
Dobrym przykładem zależności jeden-do-wielu może być kolumna `id_zawodu` z nowej wersji tabeli `moje_kontakty`. Każda osoba ma tylko jedną wartość `id_zawodu`, jednak w tabeli może być większa ilość osób, które będą miały tą samą wartość w kolumnie `id_zawodu`.

W tym przykładzie przeniesliśmy kolumnę zawod do nowej tabeli podrzędnej, a w tabeli nadrzędnej zamiast kolumny zawod pojawiła się kolumna klucza obcego — `id_zawodu`. Ponieważ w tym przypadku występuje zależność jeden-do-wielu, zatem możemy zastosować kolumnę `id_zawodu` w obu tabelach, aby je ze sobą powiązać.

Jak widać na rysunku obok, w przypadku zależności typu jeden-do-wielu linia, która ją reprezentuje, kończy się **czarną strzałką**. Jest to wizualne oznaczenie, że wiążemy jeden rekord z wieloma.

Każdy rekord z tabeli `zawody` może być powiązany z wieloma rekordami w tabeli `moje_kontakty`, jednak każdemu rekordowi w tabeli `moje_kontakty` odpowiada tylko jeden rekord z tabeli `zawody`.

Na przykład identyfikator (`id_zawodu`) zawodu programisty może się pojawić w wielu wierszach tabeli `moje_kontakty`; jednak każda osoba zapisana w tej tabeli będzie mieć tylko jedną wartość `id_zawodu`.



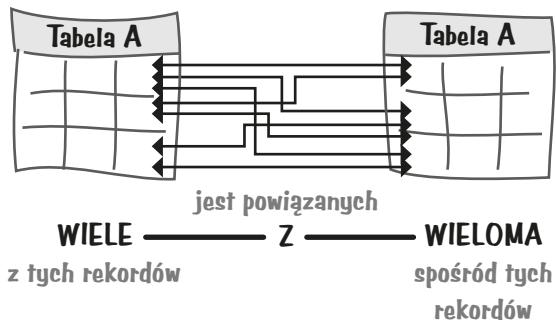
Ponieważ identyfikator zawodu (*id_zawodu*) może się powtarzać — mieć wiele powiązanych rekordów — zatem kolumna ta nie może pełnić funkcji klucza głównego tabeli. Jest to klucz obcy, gdyż odwołuje się do klucza głównego innej tabeli.

Zależności typu wiele-do-wielu

Wzorce danych: dochodzimy do zależności wiele-do-wielu

Wiele kobiet ma **wiele** par butów. Gdybyśmy chcieli zapanować nad tymi wszystkimi informacjami i stworzyli jedną tabelę do zapisania kobiet oraz drugą do zapisania butów, musielibyśmy powiązać wiele rekordów każdej z tych tabel z wieloma rekordami drugiej tabeli. To chyba zrozumiałe, gdyż więcej niż jedna kobieta może posiadać ten sam model butów.

Wyobraźmy sobie, że zarówno Cecylia, jak i Miranda kupiły stare morskie klapki oraz oryginalne prady, z kolei Samanta i Miranda kupiły sandały Manolo, natomiast Szarlota ma wszystkie te modele i jeszcze dodatkowo parę szykownych czółenek. Oto jak wyglądałyby powiązania pomiędzy tabelami kobiety oraz buty.



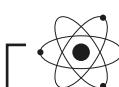
kobiety		buty	
id_kobiety	kobieta	id_butow	nazwa_modelu
1	Cecylia	1	Sandały Manolo
2	Samanta	2	Szykowne Czółenka
3	Szarlota	3	Stare Morskie Klapki
4	Miranda	4	Oryginalne Prady

A teraz wyobraź sobie, że wszystkie te panie kochały buty tak bardzo, iż każda z nich kupiła te pary, których jeszcze nie posiadała. Oto w jaki sposób wyglądałyby powiązania pomiędzy obydwoma tabelami w takiej sytuacji:

Strzalki tączące obie tabele mają czarne groty na obu końcach — tączymy wiele rekordów z wieloma rekordami.

The diagram shows two tables, 'kobiety' and 'buty', connected by a third table, 'pozycje_kupna'. The 'kobiety' table has columns 'id_kobiety' (primary key) and 'kobieta'. The 'buty' table has columns 'id_butow' (primary key) and 'nazwa_modelu'. The 'pozycje_kupna' table (not explicitly shown in the diagram but implied by the arrows) would have columns 'id_kobiety' and 'id_butow', representing the many-to-many relationship.

kobiety		pozycje_kupna		buty	
id_kobiety	kobieta			id_butow	nazwa_modelu
1	Cecylia			1	Sandały Manolo
2	Samanta			2	Szykowne Czółenka
3	Szarlota			3	Stare Morskie Klapki
4	Miranda			4	Oryginalne Prady



WYSIL SZARE KOMÓRKI

W jaki sposób można naprawić te tabele bez umieszczania więcej niż jednej informacji w jednej kolumnie (co niechybnie spowodowałoby problemy podobne do tych, które miał Grzesiek ze swoją kolumną zainteresowań)?

Zaostrz ołówek



Przyjrzyj się dwóm poniższym tabelom. Spróbowaliśmy rozwiązać problem, dodając do tabeli kobiet kolumnę id_butow, pełniącą funkcję klucza obcego.

id_kobiety	kobieta	id_butow
1	Cecylia	3
2	Samanta	1
3	Szarlotka	1
4	Miranda	1
5	Cecylia	4
6	Szarlotka	2
7	Szarlotka	3
8	Szarlotka	4
9	Miranda	3
10	Miranda	4

id_butow	nazwa_modelu
1	Sandały Manolo
2	Szykowne Czółenka
3	Stare Morskie Klapki
4	Oryginalne Prady

Teraz obie tabele są ze sobą powiązane poprzez kolumnę id_butow.

Teraz naszkicuj nową parę tabel, jednak tym razem umieść kolumnę id_kobiety jako klucz obcy w tabeli butów.

Kiedy skończysz, narysuj powiązania pomiędzy rekordami obu tabel.



Rozwiążanie ćwiczenia

Zaostrz ołówek



Rozwiążanie

Przyjrzyj się dwóm poniższym tabelom. Spróbowaliśmy rozwiązać problem, dodając do tabeli kobiet kolumnę id_butow, pełniącą funkcję klucza obcego.

id_kobiety	kobieta	id_butow
1	Cecylia	3
2	Samanta	1
3	Szarlotka	1
4	Miranda	1
5	Cecylia	4
6	Szarlotka	2
7	Szarlotka	3
8	Szarlotka	4
9	Miranda	3
10	Miranda	4

id_butow	nazwa_modelu
1	Sandały Manolo
2	Szykowne Czołenka
3	Stare Morskie Klapki
4	Oryginalne Prady

Teraz obie tabele są ze sobą powiązane poprzez kolumnę id_butow.

Zwróć uwagę na powtórzenia w kolumnach kobieta oraz nazwa_modelu.

Teraz sam naszkicuj nową parę tabel, jednak tym razem umieść kolumnę id_kobiety jako klucz obcy w tabeli butów.

Kiedy skończysz, narysuj powiązania pomiędzy rekordami obu tabel.

id_butow	nazwa_modelu	id_kobiety
1	Sandały Manolo	3
2	Szykowne Czołenka	2
3	Stare Morskie Klapki	1
4	Oryginalne Prady	1
5	Szykowne Czołenka	3
6	Stare Morskie Klapki	3
7	Oryginalne Prady	3
8	Sandały Manolo	4
9	Stare Morskie Klapki	4
10	Oryginalne Prady	4

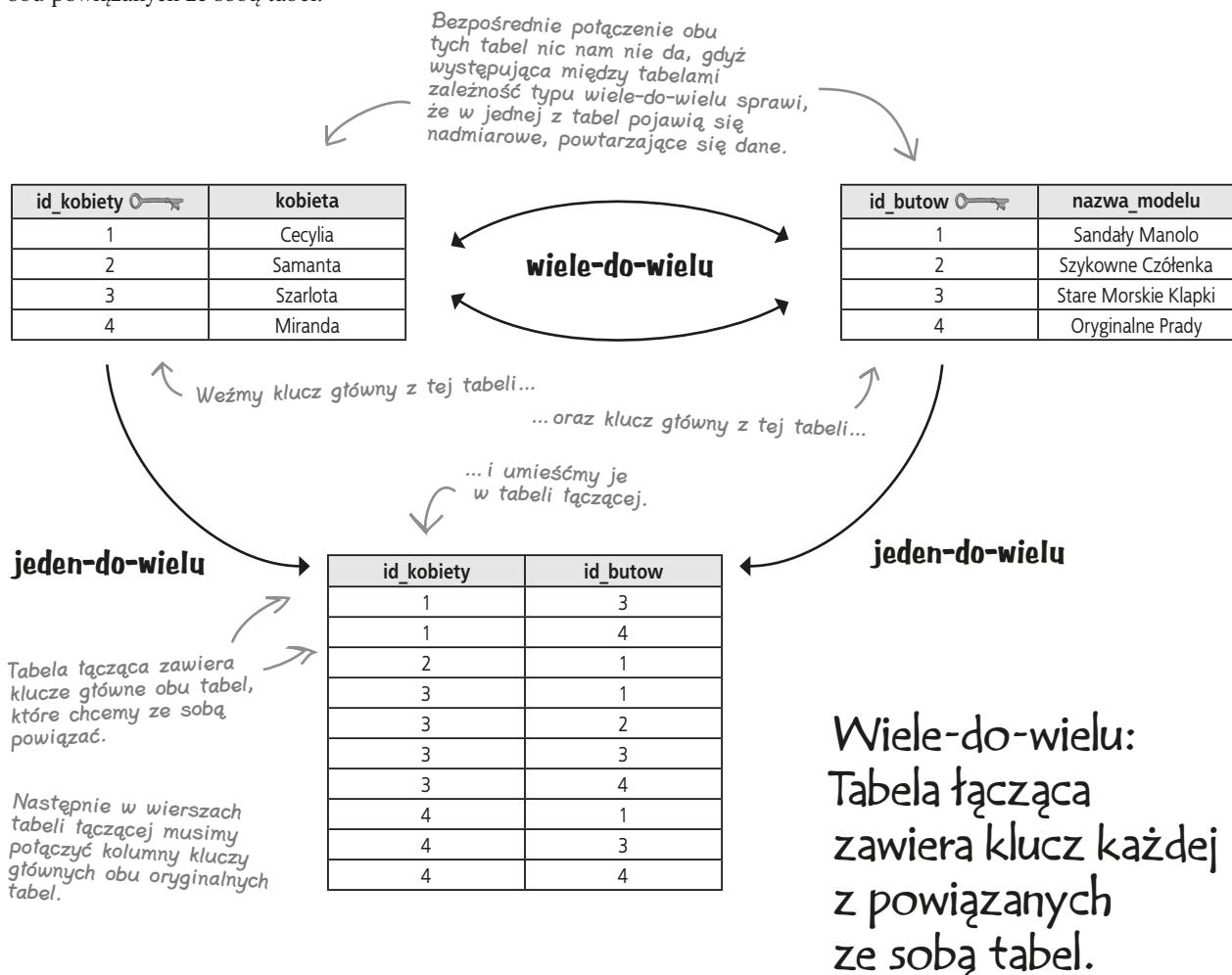
id_kobiety	kobieta
1	Cecylia
2	Samanta
3	Szarlotka
4	Miranda



Wzorce danych: potrzebujemy tabeli łączącej

Jak miałeś się okazję przekonać, dodanie do jednej tabeli klucza głównego drugiej tabeli jako klucza obcego powoduje powielenie danych w pierwszej z nich. Zauważ, jak wiele razy pojawiają się imiona tych samych kobiet. A każde z nich powinno występować w tabeli tylko jeden raz.

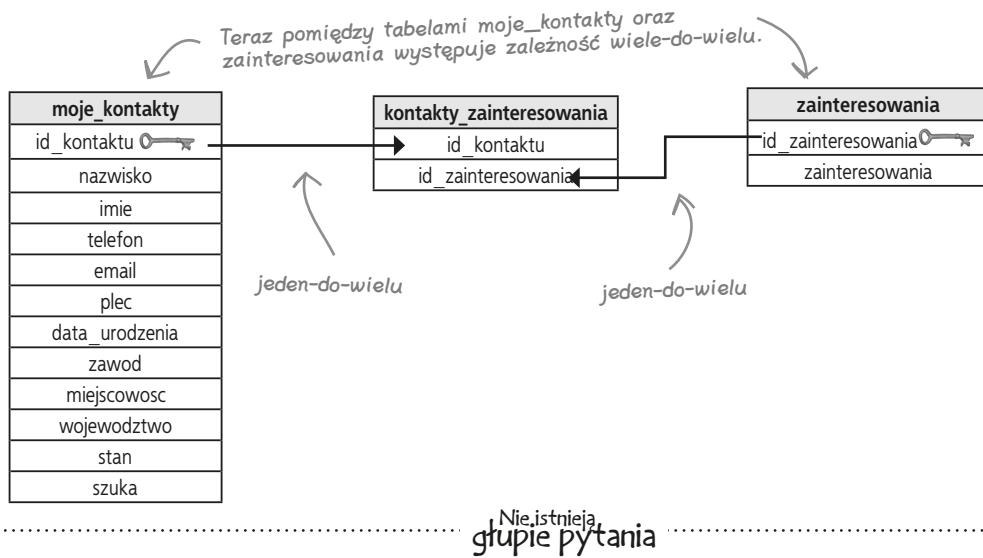
Potrzebujemy zatem dodatkowej tabeli, która oddzieliłaby od siebie obie tabele powiązane zależnością wiele-do-wielu i uprościła ją do zależności jeden-do-wielu. W tabeli tej należy zapisać wszystkie identyfikatory kobiet (`id_kobiety`) wraz z odpowiadającymi im identyfikatorami modeli butów (`id_butow`). Innymi słowy, potrzebujemy czegoś, co się nazywa **tabelą łączącą** i zawiera kolumny klucza głównego z obu powiązanych ze sobą tabel.



Wzorce danych: wiele-do-wielu

Teraz poznaleś już największy sekret zależności **wiele-do-wielu**: zazwyczaj składa się ona z dwóch zależności **jeden-do-wielu**, z dodatkową tabelą łączącą pomiędzy nimi. Musimy powiązać JEDNĄ osobę z tabeli `moje_kontakty` z WIELOMA zainteresowaniami zapisanymi w naszej nowej tabeli `zainteresowania`. Jednak każde zainteresowanie może być powiązane z dowolną ilością osób. A zatem zależność między tymi tabelami doskonale pasuje do wzorca **wiele-do-wielu**.

Kolumnę zainteresowania można przekształcić na zależność wiele-do-wielu w sposób przedstawiony poniżej. Każda osoba może mieć więcej niż jedno zainteresowanie, a dla każdego z zainteresowań może być więcej osób, które je posiadają.



P: Czy muszę tworzyć tabelę łączącą, kiedy pomiędzy moimi tabelami występują zależności wiele-do-wielu?

O: Owszem, powinieneś to robić. W przeciwnym razie, jeśli pomiędzy dwiema tabelami występuje zależność wiele-do-wielu, skoñczysz, powielając grupy danych, co będzie sprzeczne z założeniami pierwszej postaci normalnej. (Przypomnienie informacji dotyczących normalizacji i postaci normalnych znajdziesz kilka stron dalej). Nie ma żadnego uzasadnienia usprawiedliwiającego łamanie zasad pierwszej postaci normalnej, istnieje natomiast bardzo wiele argumentów przemawiających za tym, by stosować się do nich. Jednym z najważniejszych jest to, że łamanie zasad pierwszej postaci normalnej znacznie utrudni nam przeszukiwanie i pobieranie powtarzających się danych z powiązanych tabel.

P: Jakie korzyści daje przekształcenie tabel do przedstawionej powyżej postaci? Przecież mógłbym umieścić zainteresowania w jednej tabeli, zawierającej kolumny `id_kontaktu` i `zainteresowanie`. Owszem, dane były powtarzały, ale co z tego?

O: Korzyści bardzo wyraźnie zauważysz w kolejnym rozdziale, kiedy zaczniesz przeszukiwać te powiązane ze sobą tabele, używając do tego celu złączeń. Oprócz tego w niektórych sytuacjach taka struktura tabel może pomóc podczas pobierania i operowania na danych. Może się także zdarzyć, że bardziej będzie Cię interesować tabela zawierająca dane zależności wiele-do-wielu niż informacje zapisane w pozostałych dwóch tabelach.

P: No dobrze, a co jeśli pomimo to wciąż nie mam nic przeciwko powtarzaniu danych?

O: Łączenie tabel ułatwia zachowanie integralności danych. Jeśli będziesz musiał usunąć kogoś z tabeli `moje_kontakty`, to nie będzie to pociągało za sobą konieczności wprowadzania jakichkolwiek zmian w tabeli `zainteresowania` — ewentualnie zmieni się jedynie tabela `kontakty_zainteresowania`. Bez tej odrębnej tabeli mogłoby się zdarzyć, że usuniesz nieodpowiedni rekord. A zatem łączenie tabel zwiększa bezpieczeństwo. Podobnie wygląda sytuacja podczas aktualizacji danych w tabelach. Załóżmy, że błędnie zapisałesz nazwę jakiegoś wyszukanego i rzadko spotykanego hobby, jak na przykład speleorearchologia. Gdy zechcesz ją poprawić, wystarczy, że zmodyfikujesz tylko jeden wiersz w tabeli `zainteresowania` — nigdy nie będziesz musiał przejmować się dwiema pozostałymi tabelami: `kontakty_zainteresowania` oraz `moje_kontakty`.

NAZWIJ TĘ ZALEŻNOŚĆ

Dla każdej z przedstawionych poniżej fragmentarycznych tabel określ, czy zakreślone kolumny będzie najlepiej przedstawić w postaci zależności jeden-do-wielu czy też wiele-do-wielu.

(Pamiętaj, że jeśli to ma być zależność jeden-do-wielu lub wiele-do-wielu, to kolumna powinna zostać usunięta z tabeli i powiązana z kolumną identyfikatora).

KOLUMNY

paczki_oceny
typ
ocena

ZALEŻNOŚĆ

.....

klowni_informacje
id_klowna
aktywnosci
data

.....

moje_kontakty
id_kontaktu
stan
zainteresowania

.....

.....

ksiazki
id_ksiazki
autorzy
wydawca

.....

.....

rekordowe_polowy
id_rekordu
gatunek_ryby
wojewodztwo

.....

.....

NAZWIJ TĘ ZALEŻNOŚĆ. ROZWIĄZANIE

Dla każdej z przedstawionych poniżej fragmentarycznych tabel określ, czy zakreślone kolumny będzie najlepiej przedstawić w postaci zależności jeden-do-wielu czy też wiele-do-wielu.

(Pamiętaj, że jeśli to ma być zależność jeden-do-wielu lub wiele-do-wielu, to kolumna powinna zostać usunięta z tabeli i powiązana z kolumną identyfikatora).

KOLUMNIA

paczki_oceny
typ
ocena

ZALEŻNOŚĆ

jeden-do-wielu

klowni_informacje
id_klowna
aktywnosci
data

wiele-do-wielu

moje_kontakty
id_kontaktu
stan
zainteresowania

jeden-do-wielu

wiele-do-wielu

ksiazki
id_ksiazki
autorzy
wydawca

To jest trudne. Jednak książka może mieć więcej niż jednego autora, zatem zależność jest typu wiele-do-wielu.

wiele-do-wielu

jeden-do-wielu

rekordowe_polowy
id_rekordu
gatunek_ryby
województwo

jeden-do-wielu

jeden-do-wielu

Wzorce danych: Poprawki

Wiem, co wam chodzi po głowie.
Chcecie zmienić moją bazę danych i przekształcić tabelę moje_kontakty na kilka powiązanych ze sobą tabel. Prawda?

Prawie dobrze. Teraz, kiedy już znasz wzorce danych, jesteśmy prawie gotowi do zmiany struktury bazy danych lista_grzesia.

Wiemy, że kolumnę zainteresowań możemy zmienić na zależność jeden-do-wielu z inną tabelą. W taki sam sposób musimy zmienić kolumnę szuka. Te zmiany spowodują, że tabela będzie w *pierwszej postaci normalnej**.

Jednak w tym przypadku pierwsza postać normalna to nie wszystko. Musimy znormalizować bazę Grześka jeszcze bardziej. Im bardziej znormalizujemy bazę teraz, tym łatwiej będzie Ci pobierać z niej informacje w następnym rozdziale, w którym zajmiemy się złączniami. Zanim jednak zajmiemy się określeniem nowego schematu bazy Grześka, zróbjmy drobny objazd, by poznać kolejne poziomy normalizacji.



moje_kontakty
id_kontaktu
nazwisko
imie
telefon
email
plec
data_urodzenia
zawod
lokalizacja
województwo
stan
zainteresowania
szuka

* Być może będziesz chciał cofnąć się o kilka rozdziałów i przypomnieć sobie zasady określające pierwszą postać normalną. Nie musisz jednak tego robić — zajmiemy się tym na następnej stronie.

Jeszcze nie w pierwszej postaci normalnej

Opisaliśmy już pierwszą postać normalną — 1NF. Przyjrzymy się jej ponownie, a następnie posuśmy nasz proces normalizacji tabel jeszcze dalej — do drugiej, a nawet trzeciej postaci normalnej.

Jednak zanim będziemy mogli się za to zabrać, przypomnijmy sobie, jakie warunki musi spełnić tabela, by była w 1NF.

Pierwsza postać normalna lub **1NF**:

Reguła 1.: Kolumny zawierają wyłącznie dane atomowe.

Reguła 2.: Nie ma powtarzających się grup danych.

Dwie tabele przedstawione poniżej nie są zgodne z regułami 1NF. Zwrć uwagę, że w drugiej tabeli pojawiły się dodatkowe kolumny koloru, jednak w wierszach nowej tabeli kolory i tak się powtarzają.

To nie jest 1NF

id_zabawki	zabawka	kolory
5	piłka plażowa	biały, żółty, niebieski
6	ringo	zielony, żółty
9	latawiec	czerwony, niebieski, zielony
12	jo-jó	biały, żółty

Aby zapewnić atomowość danych, każdy z rekordów tabeli powinien zawierać tylko jedną wartość koloru, a nie dwie lub trzy.

To wciąż nie jest 1NF

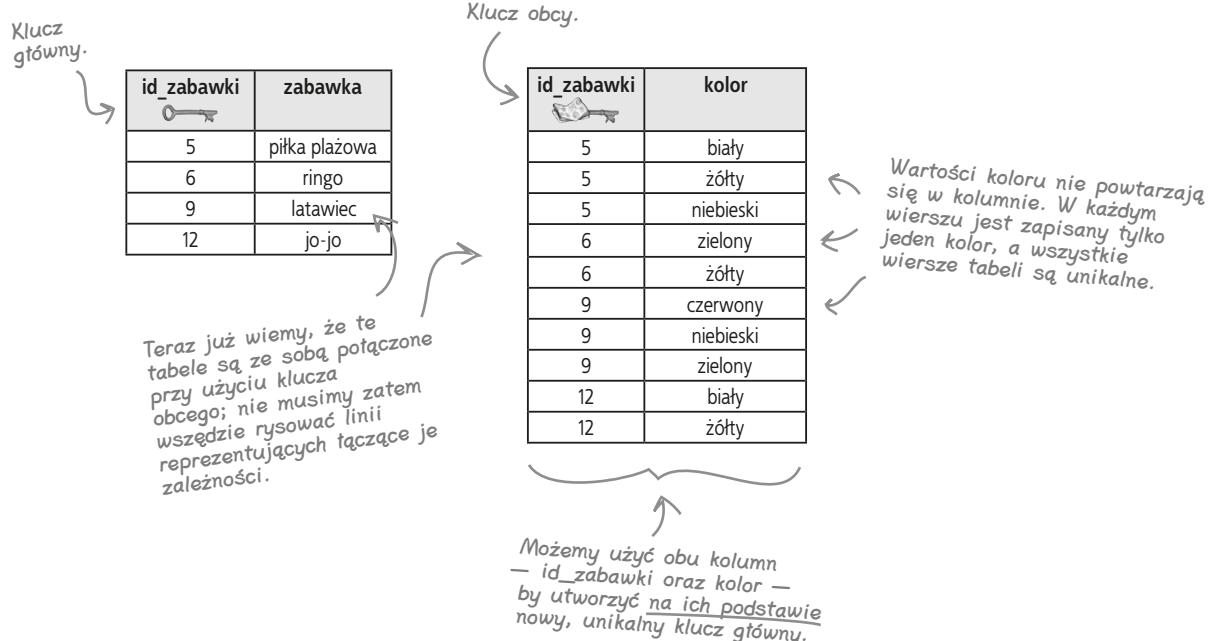
id_zabawki	zabawka	kolor1	kolor2	kolor3
5	piłka plażowa	biały	żółty	niebieski
6	ringo	zielony	żółty	
9	latawiec	czerwony	niebieski	zielony
12	jo-jó	biały	żółty	

Ta tabela także nie jest w 1NF, gdyż kolumny zawierają informacje tego samego rodzaju — dane typu **VARCHAR** określające kolor zabawki.

W końcu w 1NF

A teraz zobacz, co zrobiliśmy poniżej.

W 1NF



Jeśli dodamy id_zabawki do osobnej tabeli jako klucz obcy, wszystko będzie w porządku, gdyż wartości klucza obcego nie muszą być unikalne. Jeśli do takiej tabeli dodamy wartości kolorów, to **wszystkie jej wiersze będą unikalne**, gdyż kolor wraz z identyfikatorem id_zabawki tworzą **kombinację unikalną**.



Nie. Klucz składający się z dwóch lub większej liczby kolumn jest nazywany kluczem złożonym.

Przyjrzymy się, jak działa taki klucz, na kolejnym przykładzie.

Klucz złożony korzysta z wielu kolumn

Do tej pory opisywaliśmy *zależności*, jakie mogą występować pomiędzy danymi z różnych tabel (zależność jeden-do-wielu oraz wiele-do-wielu). Zastanawialiśmy się, w jaki sposób kolumny z jednej tabeli są powiązane z kolumnami z innej tabeli. Zrozumienie tego zagadnienia ma kluczowe znaczenie dla zrozumienia drugiej i trzeciej postaci normalnej.

A kiedy to wszystko zrozumiemy, będziemy mogli tworzyć bazy danych, których struktura zapewni łatwość wyszukiwania i pobierania danych zapisanych w wielu tabelach.

A zatem, czym właściwie jest klucz złożony?

← Na pewno będziesz chciał używać dobrze zaprojektowanych baz danych, kiedy w następnym rozdziale zajmiemy się złączoniami.

**KLUCZ ZŁOŻONY
to KLUCZ GŁÓWNY,
który jest unikalny 
i składa się z kilku kolumn.**

Przeanalizujmy przedstawioną poniżej tabelę superbohaterów. Nie ma w niej kolumny będącej kluczem głównym, jednak możemy stworzyć złożony klucz główny na podstawie kolumn nazwa oraz moc. Choć w tabeli nazwy superbohaterów oraz ich moce mogą się powtarzać, to jednak pary powstałe przez połączenie nazwy i mocy zawsze są unikalne.

Moglibyśmy utworzyć taką tabelę, a w niej klucz złożony składający się z tych dwóch kolumn. Zakładamy, że nigdy nie będziemy mieli identycznej kombinacji imienia i mocy, dzięki czemu wartości klucza będą unikalne.

superbohaterowie

nazwa	moc	słabosc
Super Śmieciarz	Błyskawicznie czyści	Szybko słabnie
Super Makler	Robi pieniądze z niczego	NULL
Super Gościu	Lata	Ptaki
MegaKelner	Nigdy nie zapomina kolejności	Owady
Piaskun	Tworzy burze piaskowe	Szybko słabnie
Super Gościu	Ma supersiłę	Inny Super Gościu
Wściekła Baba	Potrafi się naprawdę wkurzyć	NULL
Ropucha	Język przeznaczenia	Owady
Bibliotekarz	Potrafi wszystko znaleźć	NULL
Gęśia Panna	Lata	NULL
KreseczkowyMen	Zastępuje ludzi	Gra w szubienicę



KreseczkowyMen nadludzkie sity ma lecz ktoś rzec mu musi, co zrobić ma Więc otówek weź i nie wahaj się, A Twój własny KreseczkowyMen, odwiedzi Cię wnet.

Nawet superbohater może być zależny

Nasi superbohaterowie byli bardzo zajęci! Poniżej przedstawiono uaktualnioną tabelę superbohaterów. Tabela jest w pierwszej postaci normalnej, niemniej jednak mamy z nią pewien problem.

Czy zwróciłeś uwagę na kolumnę **inicjały**? Zawiera ona początkowe litery słów tworzących nazwę danego superbohatera. Co się stanie, jeśli superbohater postanowi ją zmienić?

Dokładnie. Zmieni się także zawartość kolumny **inicjały**. W takich sytuacjach mówimy, że kolumna **inicjały** jest **funkcjonalnie zależna** od kolumny **nazwa**.

Oto dwie identyczne nazwy, do których dodano kolumnę mocy, by stworzyć klucz główny o faktycznie unikalnych wartościach.

super_bohaterowie



nazwa	moc	slabosc	miasto	kraj	wrog	inicjały
Super Śmieciarz	Błyskawicznie czyści	szynko słabnie	Gotham	USA	Verminator	SS
Super Makler	Robi pieniądze z niczego	NULL	Nowy Jork	USA	Indykator	SM
Super Gościu	Lata	ptaki	Metropolis	USA	Super Hela	SG
MegaKelner	Nigdy nie zapomina kolejności	owady	Paryż	Francja	Zjadacz Niewiniątek	MK
Piaskun	Tworzy burze piaskowe	szynko słabnie	Tulsa	USA	Odkurzacz	P
Super Gościu	Ma supersił	aluminium	Metropolis	USA	Potworniak	SG
Wściekła Baba	Potrafi się naprawdę wkurzyć	NULL	Rzym	W***y		
Terapeutka	WB					
Ropucha	Język przeznaczenia	owady	Londyn	Anglia	Heron	R
Bibliotekarz	Potrafi wszystko znaleźć	dzieci	Springfield	USA	Siewca Chaosu	B
Gęśia Panna	Lata	NULL	Minneapolis	USA	Psotnik	GP
KreseczkowyMen	Zastępuje ludzi	szubienica	Londyn	Anglia	Korektor	KM

Zaostrz ołówek



Teraz już wiesz, że kolumna **inicjały** jest zależna od kolumny **nazwa**. Czy możesz jeszcze wskazać inne, podobne zależności w tabeli superbohaterów? Jeśli tak, to wypisz je poniżej.

.....

.....

.....

.....

Rozwiążanie ćwiczenia

Zaostrz ołówek



Rozwiążanie

Teraz już wiesz, że kolumna `inicjały` jest zależna od kolumny nazwa. Czy możesz jeszcze wskazać inne, podobne zależności w tabeli superbohaterów? Jeśli tak, to wypisz je poniżej.

inicjały są zależne od nazwy



Nie podaliśmy tu tabel, z których pochodzą kolumny, co będzie mieć większe znaczenie, jeśli baza będzie się składać z większej liczby tabel. Istnieje skrócony sposób zapisu pozwalający na przedstawienie zależności oraz tabel, z jakich pochodzą poszczególne kolumny.

słabość jest zależna od nazwy



wrog jest zależny od nazwy



miasto jest zależne od kraju



Zapis uproszczony



Poniżej przedstawiliśmy szybki sposób zapisania zależności funkcjonalnej:

T.x ->; T.y ← W slangu technicznym taki zapis nosi nazwę notacja uproszczona.

Zapis ten należy odczytywać w następujący sposób: „w tablicy relacyjnej o nazwie T kolumna y jest funkcjonalnie zależna od kolumny x”. Najprościej rzecz biorąc, w celu określenia, która kolumna jest zależna od której, należy odczytać ten zapis od strony prawej do lewej.

Zobaczmy, jak wygląda ten zapis po zastosowaniu do naszej tabeli superbohaterów:

superbohaterowie.nazwa ->; superbohaterowie.inicjały

„W relacyjnej tabeli superbohaterowie kolumna `inicjały` jest funkcjonalnie zależna od kolumny `nazwa`”.

superbohaterowie.nazwa ->; superbohaterowie.slabosc

„W relacyjnej tabeli superbohaterowie kolumna `slabosc` jest funkcjonalnie zależna od kolumny `nazwa`”.

superbohaterowie.nazwa ->; superbohaterowie.wrog

„W relacyjnej tabeli superbohaterowie kolumna `wrog` jest funkcjonalnie zależna od kolumny `nazwa`”.

superbohaterowie.kraj ->; superbohaterowie.miasto

„W relacyjnej tabeli superbohaterowie kolumna `miasto` jest funkcjonalnie zależna od kolumny `kraj`”.

Zależności superbohaterów

A zatem, gdyby nasz superbohater miał zmienić nazwę, zmieniłby się także jego iniciały, co oznacza, że kolumna, która je zawiera, jest **zależna** od kolumny nazwa.

Jeśli nasz główny wróg zdecyduje się przenieść swoją kryjówkę do innego miasta, zmieni się jego lokalizacja, jednak nie pociągnie to za sobą żadnych dalszych zmian w bazie. A to oznacza, że kolumna `wrog_miasto` jest **niezależna**.

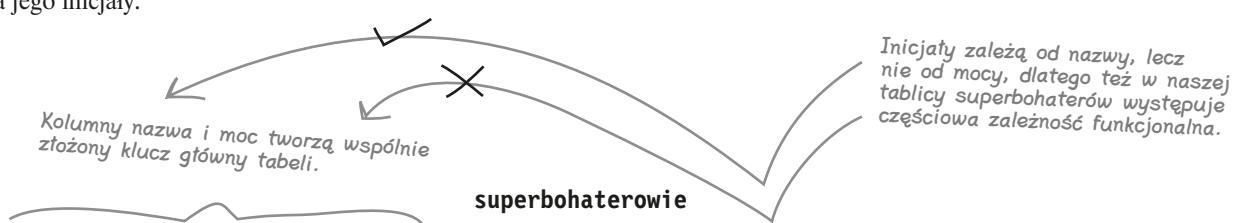
Kolumną zależną nazywamy taką kolumnę, której zawartość *może się zmieniać, jeśli ulegną zmianie informacje zapisane w innej kolumnie*. Z kolei na kolumnę **niezależną** *nie mają wpływu zmiany w zawartości innych kolumn*.



Częściowa zależność funkcjonalna

Częściowa zależność funkcjonalna oznacza, że kolumna, która nie pełni roli klucza, jest zależna od pewnych, lecz nie wszystkich, kolumn złożonego klucza głównego.

W naszej tabeli superbohaterów kolumna `inicjały` jest **częściowo zależna** od kolumny `nazwa`, gdyż jeśli zmieni się nazwa superbohatera, zmienią się także jego iniciały; z drugiej strony, jeśli zmieni się moc superbohatera, to nie będzie to miało wpływu na jego iniciały.



<code>nazwa</code>	<code>moc</code>	<code>słabość</code>	<code>miasto</code>	<code>inicjały</code>	<code>wrog_id</code>	<code>wrog_miasto</code>
Super Śmieciarz	Błyskawicznie czyści	szynko słabnie	Gotham	SS	4	Gotham
Super Makler	Robi pieniądze z niczego	NULL	Nowy Jork	SM	8	Newark
Super Gościu	Lata	ptaki	Metropolis	SG	5	Metropolis
MegaKelner	Nigdy nie zapomina kolejności	owady		MK	1	Paryż
Piaskun	Tworzy burze piaskowe	szynko słabnie	Tulsa	P	2	Kansas City
Super Gościu	Ma supersił	aluminium	Metropolis	SG	7	Gotham
Wściekła Baba	Potrafi się naprawdę wkurzyć	NULL	Rzym	WB	10	Rzym
Ropucha	Język przeznaczenia	owady	Londyn	R	16	Bath
Bibliotekarz	Potrafi wszystko znaleźć	dzieci	Springfield	B	3	Louisville
Gęsia Panna	Lata	NULL	Minneapolis	GP	9	Minneapolis
KreseczkowyMen	Zastępuje ludzi	szubienica	Londyn	KM	33	Bazodanowo

Przechodnia zależność funkcjonalna

Powinieneś zastanowić się także nad związkami kolumn, które nie pełnią roli klucza, z pozostałymi kolumnami tabeli. Jeśli nasz główny wróg przeniesie się do innego miasta, to nie będzie to miało wpływu na zawartość kolumny `wrog_id`.

Identyfikator Super Śmieciarza, zapisany w kolumnie `wrog_id`, nie zmieni się, pomimo że miasto zmieniło się na Kansas City.

nazwa	wrog_id	wrog_miasto
Super Śmieciarz	4	Kansas City
Super Makler	8	Newark
Super Gościu	5	Metropolis
MegaKelner	1	Paryż
Piaskun	2	Kansas City

Założmy, że superbohater zmieni swojego głównego wroga. W takim przypadku zmieni się zawartość kolumny `wrog_id`, a to z kolei **może** spowodować zmianę wartości w kolumnie `wrog_miasto`.

Jeśli zmiana w jakiejkolwiek kolumnie, która nie jest kluczem, powoduje zmianę zawartości dowolnej innej kolumny tabeli, świadczy to o występowaniu **zależności przechodniej**.

Jeśli zmienimy wartość w kolumnie `wrog_id`, to spowoduje to konieczność zmodyfikowania wartości w kolumnie `wrog_miasto`.

nazwa	wrog_id	wrog_miasto
Super Śmieciarz	2	Kansas City
Super Makler	8	Newark
Super Gościu	5	Metropolis
MegaKelner	1	Paryż
Piaskun	2	Kansas City

Jest to tak zwana przechodnia zależność funkcjonalna, gdyż kolumna `wrog_miasto`, która nie jest żadnym kluczem, jest zależna od kolumny `wrog_id`, która także nie jest kluczem.

Przechodnia zależność funkcjonalna występuje, gdy kolumna niebędąca kluczem jest zależna od innej kolumny, która także nie jest kluczem.



Ćwiczenie

Przyjrzyj się poniższej tabeli prezentującej książki. Kolumna `id_wyd` określa wydawcę książki, a kolumna `miasto_wyd` — miasto, w którym książka została wydana.

autor	tytuł	rok_wyd	id_wyd	miasto_wyd
Truman Capowe	Psy szczekają	1982	2	Warszawa
E.M. Remarque	Łuk Triumfalny	1973	1	Warszawa
Robert Ludlum	Manuskrypt Chancellora	1990	6	Poznań
Isaac Asimov	Równi bogom	1994	15	Poznań

Poniżej zapisz, co się stanie z wartością kolumny `rok_wyd`, jeśli tytuł książki w trzecim wierszu tabeli zmienimy na „Dzienniki gwiazdowe”.

Jeśli zmieni się tytuł, trzeba będzie także zmienić rok wydania

Rok wydania
zależy od tytułu
książki, zatem
jego wartość
także się zmieni.

Co się stanie z wartością kolumny `rok_wyd`, jeśli autora książki w trzecim wierszu zmienimy na Rin Tin Tin, lecz jej tytuł pozostanie taki sam?

.....

Co się stanie z książką „Łuk Triumfalny”, jeśli zmienimy identyfikator wydawcy, `id_wyd`, na 2?

.....

Co się stanie z wartością kolumny `id_wyd` książki „Psy szczekają”, jeśli jej wydawca przeniesie swoją siedzibę do Wrocławia?

.....

Co się stanie z wartością kolumny `miasto_wyd` książki „Równi bogom”, jeśli zmienimy wartość kolumny `id_wyd` na 1?

.....

Rozwiążanie kolejnego ćwiczenia



Rozwiążanie ćwiczenia

Przyjrzyj się poniższej tabeli prezentującej książkę. Kolumna `id_wyd` określa wydawcę książki, a kolumna `miasto_wyd` — miasto, w którym książka została wydana.

Poniżej zapisz, co się stanie z wartością kolumny `rok_wyd`, jeśli tytuł książki w trzecim wierszu tabeli zmienimy na „Dzienniki gwiazdowe”.

Jeśli zmieni się tytuł, trzeba będzie także zmienić rok wydania.

Rok wydania zależy od tytułu książki, zatem jego wartość także się zmienia.

Co się stanie z wartością kolumny `rok_wyd`, jeśli autora książki w trzecim wierszu zmienimy na Rin Tin Tin, lecz jej tytuł pozostanie taki sam?

Jeśli zmienia się autor książki, a tytuł pozostaje ten sam, to zmienia się także rok wydania.

Rok wydania zależy od tytułu. Zależy także od autora.

Kolumny autor i tytuł wspólnie tworzą złożony klucz główny tabeli.

autor 0+☒	tytuł 0+☒	rok_wyd	id_wyd	miasto_wyd
Truman Capowe	Psy szczekają	1982	2	Warszawa
E.M. Remarque	Łuk Triumfalny	1973	1	Warszawa
Robert Ludlum	Manuskrypt Chancellora	1990	6	Poznań
Isaac Asimov	Równi bogom	1994	15	Poznań

Co się stanie z książką „Łuk Triumfalny”, jeśli zmienimy identyfikator wydawcy, `id_wyd`, na 1?

Zawartość kolumny miasto_wyd nie zmieni się.

Miasto wydania (`miasto_wyd`) dla wydawców o identyfikatorach 1 i 2 jest takie samo — jest nim Warszawa. A zatem miasto się nie zmieni (poniżej faktu, iż kolumna `miasto_wyd` jest przechodnio zależna od kolumny `id_wyd`).

Kolumna `id_wyd` nie jest zależna od kolumny `miasto_wyd`, dlatego też wartość `id_wyd` pozostanie niezmieniona.

Co się stanie z wartością kolumny `id_wyd` książki „Psy szczekają”, jeśli jej wydawca przeniesie swoją siedzibę do Wrocławia?

Wartość w kolumnie id_wyd pozostanie taka sama.

Co się stanie z wartością kolumny `miasto_wyd` książki „Równi bogom”, jeśli zmienimy wartość kolumny `id_wyd` na 1?

Wartość w kolumnie miasto_wyd zmieni się na Warszawa.

Kolumna `miasto_wyd` jest zależna od wartości kolumny `id_wyd`. Żadna z tych kolumn nie jest kluczem, a zatem jest to przykład przechodniej zależności funkcjonalnej.

Kolumna `miasto_wyd` jest przechodnio zależna od kolumny `id_wyd`, dlatego też jej wartość ulegnie zmianie.

autor 0+☒	tytuł 0+☒	rok_wyd	id_wyd	miasto_wyd
Truman Capowe	Psy szczekają	1982	2	Warszawa
E.M. Remarque	Łuk Triumfalny	1973	1	Warszawa
Robert Ludlum	Manuskrypt Chancellora	1990	6	Poznań
Isaac Asimov	Równi bogom	1994	15	Poznań

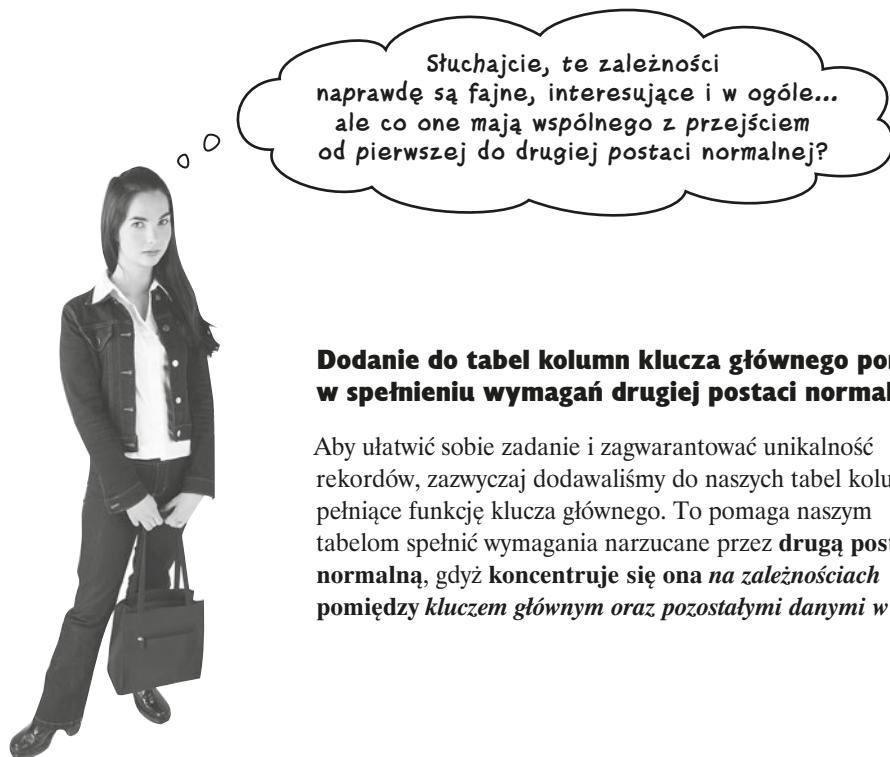
Nie istniejąca grupa pytania

P: Czy istnieje jakieś rozwiązanie pozwalające na uniknięcie częściowej zależności funkcjonalnej?

O: Problem ten można definitelywnie rozwiązać, stosując rozwiązanie, które wykorzystaliśmy w tabeli `moje_kontakty`, czyli tworząc kolumnę identyfikatora. Ponieważ jest to zupełnie nowy klucz, utworzony specjalnie w celu poindeksowania tabeli, zatem żadne inne umieszczone w niej dane nie będą od niego zależne.

P: Czy oprócz tworzenia tabeli łączącej istnieje jakiś inny powód, dla którego mógłbym chcieć tworzyć klucz złożony? Czy nie można zawsze tworzyć kolumn zawierających identyfikator?

O: Bez wątpienia jest to dobre rozwiązanie. Niemniej jednak, jeśli poszukasz w internecie hasła „synthetic or natural key” (klucz sztuczny lub naturalny), przekonasz się, że można podać przekonujące argumenty przemawiające na korzyść każdej z tych metod. Znajdziesz także wiele bardzo emocjonalnych dyskusji na ten temat. My jednak pozwolimy Ci samemu wyrobić sobie zdanie. W tej książce w przeważającej większości przypadków będziemy stosowali jednokolumnowy sztuczny klucz główny. Zdecydowaliśmy się na takie rozwiązanie, by możliwie w jak największym stopniu uprościć prezentowane polecenia SQL, tak byś bardziej koncentrował się na zagadnieniach, a nie na ich implementacji.



Dodanie do tabel kolumn klucza głównego pomaga w spełnieniu wymagań drugiej postaci normalnej.

Aby ułatwić sobie zadanie i zagwarantować unikalność rekordów, zazwyczaj dodawaliśmy do naszych tabel kolumny pełniące funkcję klucza głównego. To pomaga naszym tabelom spełnić wymagania narzucone przez **drugą postać normalną**, gdyż koncentruje się ona na zależnościach pomiędzy kluczem głównym oraz pozostałymi danymi w tabeli.

Druga postać normalna

Aby lepiej zrozumieć, w jaki sposób druga postać normalna koncentruje się na zależności pomiędzy kluczem głównym tabeli oraz pozostałymi, zapisanymi w niej informacjami, przeanalizujemy przykład dwóch tabel zawierających listę zabawek.

id_zabawki	zabawka
5	piłka plażowa
6	ringo
9	latawiec
12	jo-jó

Klucz złożony.

id_zabawki 0 + □	id_sklepu 0 + □	kolor	stan_magazynu	adres_sklepu
5	1	biały	34	ul. Klonowa 12
5	3	żółty	12	Bulwar Zachodni 222
5	1	niebieski	5	ul. Klonowa 12
6	2	zielony	10	al. Bursztynowa 3
6	4	żółty	24	ul. Zwycięstwa 167
9	1	czerwony	50	ul. Klonowa 12
9	2	niebieski	2	al. Bursztynowa 3
9	2	zielony	18	al. Bursztynowa 3
12	4	biały	28	ul. Zwycięstwa 167
12	4	żółty	11	ul. Zwycięstwa 167

Także nad tą kolumną można by się jeszcze zastanowić. Tak naprawdę powinna się ona raczej znaleźć w tabeli zabawek, a nie w tabeli stanu magazynowego. Pole id_zabawki powinno identyfikować zarówno zabawkę, JAK I jej kolor.

Dane w tej kolumnie wielokrotnie się powtarzają. I tak naprawdę informacje te nie mają nic wspólnego ze stanem magazynowym — są one natomiast ścisłe powiązane z samym sklepem.

Stan magazynowy zależy od obu kolumn tworzących złożony klucz główny tabeli, a zatem w tym przypadku nie występuje częściowa zależność funkcjonalna.

Zwróć uwagę, że wartości kolumny adres_sklepu powtarzają się, gdy w kolumnie id_sklepu znajdują się te same wartości.

Gdybyśmy musieli zmienić adres sklepu, to musielibyśmy zmienić wszystkie wiersze tabeli, w których ten adres występuje. A trzeba pamiętać, że im więcej będziemy zmieniać, tym większe będzie prawdopodobieństwo pojawienia się błędów i nieprawidłowości w danych.

Gdybyśmy przenieśli kolumnę adres_sklepu do innej tabeli, to zmiana adresu sklepu wymagałaby wprowadzenia modyfikacji tylko w jednym rekordzie.

Nasze tabele mogą już być w 2NF

Tabela w pierwszej postaci normalnej może być także w drugiej postaci normalnej, jeśli wszystkie jej kolumny będą wchodzić w skład klucza głównego.

Moglibyśmy utworzyć nową tabelę zawierającą złożony klucz główny składający się z kolumn `id_zabawki` oraz `id_sklepu`. W takim przypadku moglibyśmy mieć jedną tabelę zawierającą jedynie informacje o zabawkach, drugą zawierającą informacje o sklepach, a nasza nowa tabela łączyłaby dwie pozostałe.

Tabela w 1NF jest jakże w 2NF, jeśli wszystkie jej kolumny wchodzą w skład klucza głównego

LUB

jeśli jej klucz główny jest pojedynczą kolumną

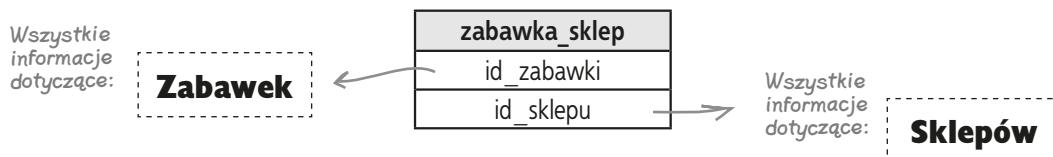


Tabela w pierwszej postaci normalnej jest także w drugiej postaci normalnej, jeśli ma klucz główny będący pojedynczą kolumną.

To jeden z ważnych powodów przemawiających za tworzeniem w tabelach dodatkowej kolumny klucza głównego, której wartości będą automatycznie inkrementowane.

Druga postać normalna lub 2NF:

Reguła 1.: Tabela musi być w 1NF.

Reguła 2.: W tabeli nie mogą występować częściowe zależności funkcjonalne.



Nie sądzę,
żeby w mojej tabeli
moje_kontakty występowały
jakieś częściowe zależności
funkcjonalne... ale głowy
za to nie dam...

Dlatego czas się zabawić...

Bądź tabelą w 2NF bez częściowych zależności funkcjonalnych



Twoim zadaniem jest wcielić się w rolę tabeli i pozbyć się wszystkich częściowych zależności funkcjonalnych.

Przyjrzyj się każdej z tabel przedstawionych poniżej i w każdej z nich przekreśl wszystkie kolumny, które lepiej byłoby przenieść do innej tabeli.

Te dwie kolumny tworzą złożony klucz główny tabeli.

zabawka-sklep
id_zabawki
id_sklepu

piosenkarze
id_piosenkarza
nazwisko
imie
agencja
agencja_miasto

ciasteczka_sprzedaz
ilosc
id_dziewczyny
date
imie_dziewczyny
dowodca_grupy
sprzedaz_sumaryczna

wynagrodzenie
id_pracownika
nazwisko
imie
kwota
kierownik
email_pracownika
data_zatrudnienia

filmy
id_filmu
tytuł
rodzaj
wypożyczony_przez
oczekiwana_data_zwrotu
ocena

rasy_psow
nazwa_rasy
opis
srednia_waga
srednia_wysokosc
id_klubu
klub_miasto

Zaostrz ołówek



Przekształć przedstawione poniżej dwie tabele na trzy tabele w 2NF.

Pierwsza z tabel będzie zawierać informacje o zabawkach, druga informacje o sklepach, a trzecia będzie łączyć dwie pozostałe tabele i jednocześnie zawierać informacje o stanie magazynowym. Każdej z tabel nadaj odpowiednią nazwę.

W końcu w odpowiednich tabelach utwórz niezbędne, dodatkowe kolumny.

id_zabawki	zabawka
5	piłka plażowa
6	ringo
9	latawiec
12	jo-jó

id_zabawki 0 + □	id_sklepu 0 + □	kolor	stan_magazynu	adres_sklepu
5	1	biały	34	ul. Klonowa 12
5	3	żółty	12	Bulwar Zachodni 222
5	1	niebieski	5	ul. Klonowa 12
6	2	zielony	10	al. Bursztynowa 3
6	4	żółty	24	ul. Zwycięstwa 167
9	1	czerwony	50	ul. Klonowa 12
9	2	niebieski	2	al. Bursztynowa 3
9	2	zielony	18	al. Bursztynowa 3
12	4	biały	28	ul. Zwycięstwa 167
12	4	żółty	11	ul. Zwycięstwa 167

Bądź tabelą w 2NF bez częściowych zależności funkcjonalnych. Rozwiążanie



Twoim zadaniem jest wcielić się w rolę tabeli i pozbyć się wszystkich częściowych zależności funkcjonalnych.

Przyjrzyj się każdej z tabel przedstawionych poniżej i w każdej z nich przekreśl wszystkie kolumny, które lepiej byłoby przenieść do innej tabeli.

Te dwie kolumny tworzą złożony klucz główny tabeli.

zabawka-sklep
id_zabawki
id_sklepu

Klucz główny.

piosenkarze
id_piosenkarza
nazwisko
imie
agencja
agencja_miasto

Choć to powinien być identyfikator pobrany z tabeli agencji (gdyż może się zdarzyć, że dwie agencje będą mieć taką samą nazwę), to jednak w kolumnach tych nie występuje częściowa zależność funkcjonalna.

wynagrodzenie
id_pracownika
nazwisko
imie
kwota
kierownik
email_pracownika
data_zatrudnienia

Choć te wszystkie kolumny powinny być usunięte z tej tabeli, to jednak nie powodują one występowania częściowej zależności funkcjonalnej.

ciasteczka_sprzedaz
ilosc
id_dziewczyny
date
imie_dziewczyny
dowodca_grupy
sprzedaz_sumaryczna

Po przeniesieniu przekreślonych kolumn do innej tabeli pozostałe kolumny mogą utworzyć złożony klucz główny.

Klucz główny.

filmy
id_filmu
tytuł
rodzaj
wypożyczony_przez
oczekiwana_data_zwrotu
ocena

W tych kolumnach występuje jedynie przechodnia zależność funkcjonalna.

rasy_psow
nazwa_rasy
opis
srednia_waga
srednia_wysokosc
id_klubu
klub_miasto

Złożony klucz główny.

Kolumna id_klubu faktycznie może się znaleźć w tej tabeli (zakładając, że jest to zależność typu jeden-do-jednego), niemniej jednak kolumnę klub_miasto bez wątpienia należy stąd usunąć. Pomimo to żadne kolumny nie powodują występowania częściowej zależności funkcjonalnej.

Zaostrz ołówek



Rozwiążanie

Przekształć przedstawione poniżej dwie tabele na trzy tabele w 2NF.

Pierwsza z tabel będzie zawierać informacje o zabawkach, druga informacje o sklepach, a trzecia będzie łączyć dwie pozostałe tabele i jednocześnie zawierać informacje o stanie magazynowym. Każdej z tabel nadaj odpowiednią nazwę.

W końcu w odpowiednich tabelach utwórz niezbędne, dodatkowe kolumny.

id_zabawki	zabawka
5	piłka plażowa
6	ringo
9	latawiec
12	jo-jó

id_zabawki	id_sklepu	kolor	stan_magazynu	adres_sklepu
5	1	biały	34	ul. Klonowa 12
5	3	żółty	12	Bulwar Zachodni 222
5	1	niebieski	5	ul. Klonowa 12
6	2	zielony	10	al. Bursztynowa 3
6	4	żółty	24	ul. Zwycięstwa 167
9	1	czerwony	50	ul. Klonowa 12
9	2	niebieski	2	al. Bursztynowa 3
9	2	zielony	18	al. Bursztynowa 3
12	4	biały	28	ul. Zwycięstwa 167
12	4	żółty	11	ul. Zwycięstwa 167

Złożony klucz główny składa się z kolumn **id_zabawki** oraz **id_sklepu**.

zabawka_informacje

id_zabawki	zabawka	kolor	cena	waga
1	piłka plażowa	biały	4.90	0.3
2	piłka plażowa	żółty	6.20	0.4
3	piłka plażowa	niebieski	4.90	0.3
4	ringo	zielony	7.75	0.5
5	ringo	żółty	3.10	0.2
6	latawiec	czerwony	15.50	1.2
7	latawiec	niebieski	15.50	1.2
8	latawiec	zielony	10.50	0.8
9	jo-jó	biały	11.25	0.4
10	jo-jó	żółty	3.10	0.2

sklep_magazyn

id_zabawki	id_sklepu	stan_magazynu
5	1	34
5	3	12
5	1	5
6	2	10
6	4	24
9	1	50
9	2	2
9	2	18
12	4	28
12	4	11

sklep_informacje

id_sklepu	adres_sklepu	telefon	kierownik
1	ul. Klonowa 12	32 5729384	Janek
2	al. Bursztynowa 3	12 8439848	Zuzanna
3	Bulwar Zachodni 222	22 6635535	Tamara
4	ul. Zwycięstwa 167	18 0985728	Grzesiek

Trzecia postać normalna (w końcu)

Ponieważ w tej książce zazwyczaj dodajemy do tabel sztuczne klucze główne, zatem doprowadzenie tabel do 2NF nie jest dla nas poważnym problemem. Każda tabela, która ma **sztuczny klucz główny** i nie ma żadnego złożonego klucza głównego, jest w drugiej postaci normalnej.

Musimy się jednak upewnić, czy nasze tabele są w trzeciej postaci normalnej:

Trzecia postać normalna lub 3NF:

Reguła 1.: Tabela musi być w 2NF.

Reguła 2.: W tabeli
nie mogą występować
zależności przechodnie.

Zastanów się, co się stanie, jeśli w poniższej tabeli zmienimy wartość w jednej z następujących kolumn: **nazwa_kursu**, **instruktor**, **instruktor_telefon**.

Jeśli zmienimy wartość w kolumnie **nazwa_kursu**, nie spowoduje to konieczności zmiany wartości kolumn **instruktor** ani **instruktor_telefon**.

Jeśli zmienimy wartość w kolumnie **instruktor_telefon**, nie spowoduje to konieczności zmiany wartości kolumn **nazwa_kursu** ani **instruktor**.

Jeśli zmienimy wartość w kolumnie **instruktor**, to zmieni się wartość w kolumnie **instruktor_telefon**. A zatem znaleźliśmy zależność przechodnią.

Jeśli nasza tabela ma **sztuczny klucz główny** i **nie ma złożonego klucza głównego**, to jest w drugiej postaci normalnej.

← Pamiętasz? Przechodnia zależność funkcjonalna oznacza, że dowolna z kolumn tabeli, która nie jest jej kluczem głównym, jest powiązana z dowolną inną kolumną, która także nie jest kluczem głównym.

Jeśli zmiana jakiegośkolwiek kolumny, która nie jest kluczem, może doprowadzić do konieczności wprowadzenia zmian w innych kolumnach tabeli, oznacza to, że w tej tabeli występuje przechodnia zależność funkcjonalna.

kursy
id_kursu
nazwa_kursu
instruktor
instruktor_telefon

Analizując tabelę pod względem zgodności z 3NF, możemy pominać kolumnę klucza głównego.

Obecnie jest dla nas oczywiste, że jeśli chcemy, by ta tabela była w trzeciej postaci normalnej, to nie powinna się w niej znajdująć kolumna **instruktor_telefon**.



Ćwiczenie

Jak zatem wygląda sprawa z tabelą moje_kontakty?

Tabela ta wymaga wprowadzenia kilku zmian. Na tej stronie, obok rysunku przedstawiającego bieżącą postać tabeli moje_kontakty, naszkicuj nowy schemat bazy danych lista_grzesia. Zaznacz na nim zależności pomiędzy kluczami obcymi oraz zależności typu jeden-do-wielu; pierwsze z nich narysuj jako linie, a drugie jako strzałki. Zaznacz także kolumny klucza głównego oraz kluczy złożonych.

moje_kontakty
id_kontaktu
nazwisko
imie
telefon
email
plec
data_urodzenia
zawod
lokalizacja
województwo
stan
zainteresowania
szuka



Podpowiedź: W nowej wersji bazy danych, którą znajdziesz na następnej stronie, utworzyliśmy osiem tabel. (Dodaliśmy także kolumnę na kod pocztowy. Bez niej mieliśmy siedem tabel).



Rozwiązywanie ćwiczenia

Jak zatem wygląda sprawa z tabelą moje_kontakty?

Tabela ta wymaga wprowadzenia kilku zmian. Na tej stronie, obok rysunku przedstawiającego bieżącą postać tabeli moje_kontakty, naszkicuj nowy schemat bazy danych lista_grzesia.

Zaznacz na nim zależności pomiędzy kluczami obcymi oraz zależności typu jeden-do-wielu; pierwsze z nich narysuj jako linie, a drugie jako strzałki. Zaznacz także kolumny klucza głównego oraz kluczy złożonych.

moje_kontakty
id_kontaktu
nazwisko
imie
telefon
email
plec
data_urodzenia
zawod
lokalizacja
województwo
stan
zainteresowania
szuka

To jest zależność wiele-do-wielu, utworzona jako dwie zależności jeden-do-wielu oraz tabela łączająca.

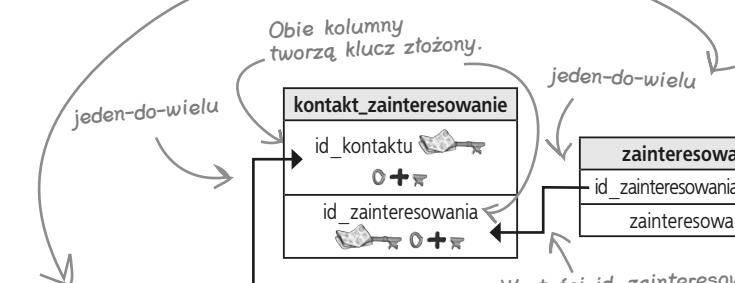
Te trzy tabele tworzą z tabelą moje_kontakty zależności jeden-do-wielu.

zawody
id_zawodu
zawod

kod_pocztowy
kod_pocztowy
lokalizacja
województwo

stan
id_stanu
stan

moje_kontakty
id_kontaktu
nazwisko
imie
telefon
email
plec
data_urodzenia
zawod_id
kod_pocztowy
id_stanu



Wartości id_zainteresowania w tabeli kontakt_zainteresowanie mogą się powtarzać, natomiast w tabeli zainteresowanie, każda z tych wartości może wystąpić tylko jeden raz.

kontakt_szuka
id_kontaktu
0 + ↗
id_szuka
0 + ↗

szuka
id_szuka
szuka

Obie kolumny tworzą klucz złożony.

To jest zależność wiele-do-wielu, utworzona jako dwie zależności jeden-do-wielu oraz tabela łączająca.

I tak oto Robert i lista_grzesia żyli od tej pory długo i szczęśliwie

W końcu dzięki znormalizowanej bazie danych Grześkowi udało się znaleźć idealną kandydatkę dla Roberta. Ale to nie wszystko — Grzesiek bez większych problemów jest już w stanie znajdować idealne pary dla większości ze swoich znajomych, dzięki czemu urealnił swoje marzenia dotyczące bazy danych `lista_grzesia`.



Koniec



Hej! Nie tak szybko!
Teraz muszę przeglądać te wszystkie
tabele i ręcznie dopasowywać pobierane
z nich informacje! W jaki sposób mogę
korzystać z informacji zapisanych w tych
wszystkich tabelach, bez konieczności
wykonywania setek zapytań?

**I właśnie w tym momencie na scenę
wkraczają związki.**

Do zobaczenia w następnym rozdziale...

Projektowanie baz z wieloma tabelami

Zaostrz ołówek



Rozwiążanie

Korzystając z polecenia ALTER oraz funkcji SUBSTRING_INDEX, zmodyfikuj tabelę Grześka w taki sposób, by miała następujące kolumny:

Przede wszystkim musisz utworzyć nowe kolumny:

```
ALTER TABLE moje_kontakty
ADD COLUMN zainteresowanie1 VARCHAR(30),
ADD COLUMN zainteresowanie2 VARCHAR(30),
ADD COLUMN zainteresowanie3 VARCHAR(30),
ADD COLUMN zainteresowanie4 VARCHAR(30);
```

Następnie musisz przenieść pierwsze zainteresowanie do kolumny „zainteresowaniem”. Możesz to zrobić w następujący sposób:

```
UPDATE moje_kontakty
SET zainteresowanie1 = SUBSTRING_INDEX(zainteresowania, ',', 1);
```

Następnie musimy usunąć pierwsze zainteresowanie z dotychczasowej kolumny „zainteresowania”, gdyż jest już zapisane w kolumnie „zainteresowaniem”. Dzięki zastosowaniu odpowiednich funkcji łańcuchowych możemy usunąć z kolumny wszystko od jej początku do miejsca tuż za wystąpieniem pierwszego przecinka:

Funkcja TRIM usunie wszystkie znaki odstępu, znajdujące się z lewej strony łańcucha znaków, jaki uzyskamy po usunięciu z kolumny „zainteresowania” wszystkiego od początku do pierwszego przecinka.

Funkcja RIGHT zwraca fragment kolumny „zainteresowania”, zaczynając od jej prawego końca.

```
UPDATE moje_kontakty SET zainteresowania = TRIM(RIGHT(zainteresowania,
(LENGTH(zainteresowania)-LENGTH(zainteresowanie1) - 1)));
```



To strasznie wyglądające wyrażenie wylicza, jaki fragment kolumny „zainteresowania” nas interesuje. Oznacza ono całkowitą długość łańcucha znaków zapisanego w tej kolumnie i odejmuje od niej długość łańcucha zapisanego wcześniej w kolumnie „zainteresowaniem”. Następnie od uzyskanej wartości odejmujemy 1, by usunąć z kolumny pierwszy przecinek.

Teraz możesz powtórzyć te czynności, by określić wartości pozostałych kolumn zainteresowań:

```
UPDATE moje_kontakty SET zainteresowanie2 = SUBSTRING_INDEX(zainteresowania, ',', 1);
UPDATE moje_kontakty SET zainteresowania = TRIM(RIGHT(zainteresowania,
(LENGTH(zainteresowania)-LENGTH(zainteresowanie2) - 1)));
UPDATE moje_kontakty SET zainteresowanie3 = SUBSTRING_INDEX(zainteresowania, ',', 1);
UPDATE moje_kontakty SET zainteresowania = TRIM(RIGHT(zainteresowania,
(LENGTH(zainteresowania)-LENGTH(zainteresowanie3) - 1)));
```

Aby wypełnić ostatnią nową kolumnę, wystarczy przenieść do niej jedną wartość pozostającą w kolumnie „zainteresowania”:

```
UPDATE moje_kontakty SET zainteresowanie4 = zainteresowania;
```

Teraz pozostało już jedynie usunąć kolumnę „zainteresowania”. Można było także zmienić nazwę kolumny „zainteresowania” na „zainteresowanie4” i nie dodawać tej kolumny na samym początku modyfikacji.

id_kontaktu
nazwisko
imie
telefon
email
plec
data_urodzenia
zawod
lokalizacja
województwo
stan
zainteresowanie1
zainteresowanie2
zainteresowanie3
zainteresowanie4
szuka

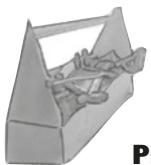


Rozwiążanie ćwiczenia ze str. 316

Napisz zapytanie, które odnajdzie kandydatki na randkę dla Roberta, przy czym nie uwzględniaj w nim kolumny zainteresowania.

```
SELECT * FROM moje_kontakty
WHERE plec = 'K'
AND stan = 'samotna' ←
AND województwo = 'PM'
AND szuka LIKE '%samotnego%''
AND data_urodzenia > '1950-03-20'
AND data_urodzenia < '1960-03-20'
```

Ogólnie rzecz biorąc, jest to to samo zapytanie, którego Grzesiek użył, poszukując partnerki dla Wieśka, z tym że pominął w nim porównywanie zainteresowań.



Przybornik SQL

Możesz sobie pogratulować, już jesteś bliżej końca niż początku tej książki. Przejrzyj jeszcze raz wszystkie kluczowe pojęcia języka SQL, które poznaleś w rozdziale siódmym. Kompletną listę porad znajdziesz w dodatku C, zamieszczonym na końcu książki.

Schemat

Opis danych zapisywanych w bazie oraz wszelkich innych obiektów wchodzących w jej skład wraz z informacjami o ich wzajemnych połączeniach.

Zależność jeden-do-jednego
Dokładnie jeden wiersz tabeli podzielonej jest powiązany z jednym wierszem tabeli nadzielonej.

Zależność jeden-do-wielu

Z jednym wierszem pierwszej tabeli może być powiązanych dowolnie wiele wierszy drugiej tabeli, jednak każdy wiersz drugiej tabeli jest powiązany tylko z jednym wierszem pierwszej tabeli.

Zależność wiele-do-wielu
Dwie tabele są ze sobą powiązane przy wykorzystaniu tabeli łączającej, dzięki czemu tabeli może być powiązanych dowolnie wiele wierszy drugiej tabeli i na odwrót.

Pierwsza postać normalna (1NF)

Kolumny tabeli zawierają wyłącznie dane atomowe, a w kolumnach nie mogą się powtarzać grupy danych.

Przechodnia zależność funkcjonalna

Oznacza, że w tabeli istnieje jakaś kolumna niebędąca kluczem głównym, zależna od dowolnej innej kolumny, która także nie jest kluczem głównym.

Druga postać normalna (2NF)

Aby tabela była w 2NF, musi być w 1NF i nie mogą w niej występować żadne częściowe zależności funkcjonalne.

Trzecia postać normalna (3NF)

Tabela musi być w 2NF i nie mogą w niej występować żadne zależności przechodnie.

Klucz obcy

To kolumna w tabeli, która odwołuje się do KLUCZA GŁÓWNEGO innej tabeli.

Klucz złożony

To klucz główny składający się z wielu kolumn, dzięki czemu jego wartości są unikalne.

8. Złączenia i operacje na wielu tabelach

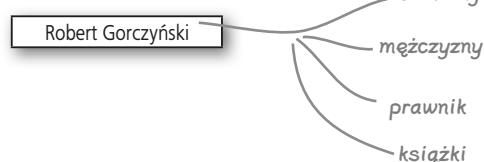
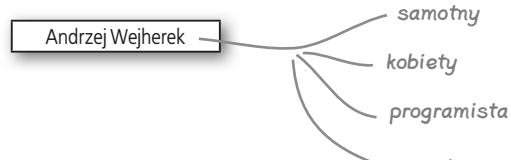
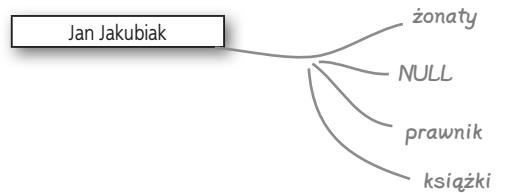
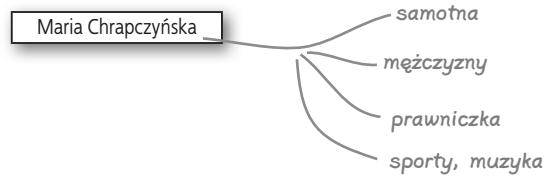
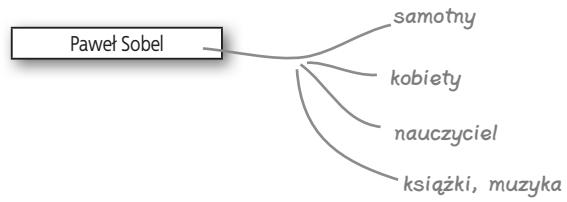
Czy nie możemy się wszyscy dogadać?



Witam w skomplikowanym świecie wielu tabel. Wspaniale jest mieć w bazie więcej niż jedną tabelę, jednak abyś mógł się nimi posługiwać, będziesz musiał poznać **nowe narzędzia i techniki**. Jednoczesne korzystanie z wielu tabel może być kłopotliwe, dlatego też, by wszystko było jasne, będziesz potrzebował nazw zastępczych (nazywanych także **aliasami**). Oprócz tego **złączenia** pomogą Ci połączyć tabele, dzięki czemu będziesz mógł korzystać ze wszystkich danych, zapisanych w wielu różnych miejscach. Przygotuj się — teraz ponownie **odzyskasz pełną kontrolę nad swoją bazą danych**.

Powtarzamy się, cały czas się powtarzamy...

Grzesiek zauważył, że wartości **stanu**, **zawodu**, **zainteresowań** oraz tego, czego dana osoba **poszukuje**, cały czas się powtarzają.

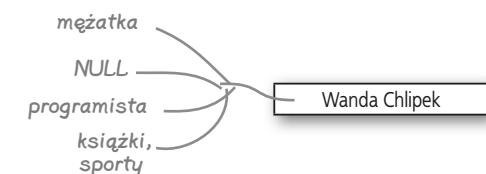
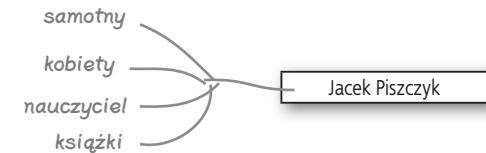
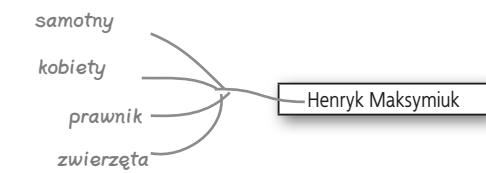
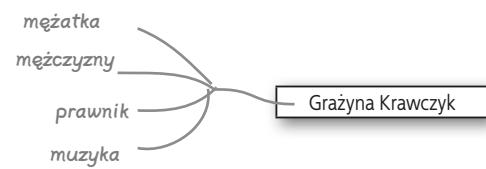
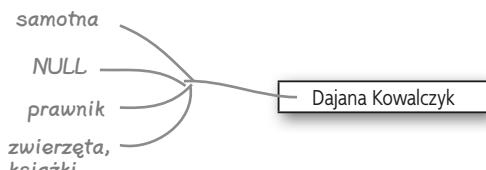


stan
samotny
samotna
żonaty
mężatka

szuka
mężczyzny
kobiety

zawód
programista
nauczyciel
prawnik

zainteresowania
książki
muzyka
zwierzęta
sporty



Wypełnianie tabel

Dzięki częstemu powtarzaniu się pewnych wartości łatwo będzie z góry określić początkową zawartość tabel stan, zawod, zainteresowania oraz szuka. Grzesiek chciałby od razu, jeszcze przed rozpoczęciem korzystania ze swojej nowej bazy, wypełnić te tabele wartościami, które już wcześniej zostały zapisane w tabeli `moje_kontakty`.

W pierwszej kolejności Grzesiek musi przejrzeć swoją tabelę, by zobaczyć, co już jest w niej zapisane. Jednak nie chce przy tym oglądać niekończących się list powtarzających się wartości.



Zaostrz ołówek



Napisz zapytania, które pozwolą Grześkowi pobrać ze starej tabeli `moje_kontakty` wartości kolumn `stan`, `zawod`, `zainteresowania` oraz `szuka`, przy czym prezentowane wartości nie powinny się powtarzać. Jeśli będziesz potrzebował pomocy, to możesz zajrzeć do rozdziału 6. i przykładów ze sprzedażą ciasteczek.

Zaostrz ołówek



Rozwiążanie

Napisz zapytania, które pozwolą Grześkowi pobrać ze starej tabeli moje_kontakty wartości kolumn stan, zawod, zainteresowania oraz szuka, przy czym prezentowane wartości nie powinny się powtarzać. Jeśli będziesz potrzebował pomocy, to możesz zająrzyć do rozdziału 6. i przykładów ze sprzedżą ciasteczkę.

**SELECT stan FROM moje_kontakty
GROUP BY stan
ORDER BY stan;**

Zastosowanie klauzuli GROUP BY łączy takie same wartości w grupy i powoduje, że z każdej z takich grup zostanie zwrocona tylko jedna wartość.

Następnie zastosowanie klauzuli ORDER BY zwraca wyniki posortowane alfabetycznie.

Gdyby klauzule zostały zapisane w innej kolejności, podczas wykonywania polecenia zostałyby zgłoszone błędy — klauzula ORDER BY zawsze musi być podawana jako ostatnia.

**SELECT zawod FROM moje_kontakty
GROUP BY zawod
ORDER BY zawod;**

**SELECT szuka FROM moje_kontakty
GROUP BY szuka
ORDER BY szuka;**

~~**SELECT zainteresowania
FROM moje_kontakty
GROUP BY zainteresowania
ORDER BY zainteresowania**~~

Ale to zapytanie wykonane dla kolumny zainteresowania nie działa prawidłowo. Pamiętacie, w każdym wierszu tej kolumny może być zapisanych więcej niż jedna wartość.



Nie można pobrać wartości z kolumny zainteresowania przy użyciu prostego zapytania SELECT.

Zastosowanie przedstawionego polecenia SELECT nie da oczekiwanych efektów, jeśli w tabeli będą zapisane dane o następującej postaci:

zainteresowania
książki, sporty
muzyka, zwierzęta, książki
zwierzęta, książki
sporty, muzyka

Teraz musimy słuchać bluesa „niełatwą normalizacją”

To jest kolejny moment, kiedy dotkliwie odczuujemy fakt stosowania nieznormalizowanych tabel. Nie ma prostego sposobu pobrania wartości z kolumny zainteresowań, tak by każda z nich została wyświetlona osobno i to tylko jeden raz.

Musimy przekształcić tabele o postaci:

zainteresowania
pierwsza, druga, trzecia, czwarta

Przykładowa zawartość kolumny z naszej dotychczasowej tabeli moje_kontakty.

do tej postaci:

zainteresowania
pierwsza
druga
trzecia
czwarta

Kolumna z naszej nowej tabeli zainteresowania.



WYSIL
SZARE KOMÓRKI

W jaki sposób można pobrać te wszystkie wartości i zapisać je w naszej nowej tabeli zainteresowań?

Czy nie możemy tego zrobić ręcznie?
Czyli przeglądać poszczególne wiersze tabeli moje_kontakty i zapisywać kolejne wartości do nowej tabeli zainteresowań?



Przede wszystkim byłoby z tym niewyobrażalnie dużo pracy. Wyobraź sobie, że w tabeli moje_kontakty są zapisane tysiące rekordów.

Poza tym ręczne zapisywanie danych utrudniłoby wyszukiwanie i eliminację powtarzających się danych. Jeśli w tabeli będą setki zainteresowań, to przeniesienie każdego z nich do nowej tabeli wymagałoby sprawdzenia, czy nie zostało ono już zapisane wcześniej.

Zamiast samodzielnie wykonywać tę trudną pracę, ryzykując przy tym pojawienie się błędów, lepiej pozwolić, by to SQL wykonał wszystko za nas.

Zainteresowania — kłopotliwa kolumna

Jednym ze stosunkowo prostych rozwiązań naszego problemu jest utworzenie w tabeli `moje_kontakty` czterech nowych kolumny, w których moglibyśmy tymczasowo przechowywać wartości pobierane z kolumny `zainteresowania`. Później, kiedy skończymy, będziemy mogli usunąć te kolumny.

Zaostrz ołówek



Już wiesz, jak można modyfikować tabele, używając w tym celu polecenia `ALTER`. A zatem dodaj do tabeli `moje_kontakty` cztery nowe kolumny. Nadaj im odpowiednio nazwy: `zainteresowanie1`, `zainteresowanie2`, `zainteresowanie3` oraz `zainteresowanie4`.

→ Odpowiedź znajdziesz na stronie 407

Oto jak powinna wyglądać tabela `moje_kontakty` po dodaniu do niej czterech nowych kolumn zainteresowań.

<code>zainteresowania</code>	<code>zainteresowanie1</code>	<code>zainteresowanie2</code>	<code>zainteresowanie3</code>	<code>zainteresowanie4</code>
pierwsze, drugie, trzecie, czwarte				

Bez większych problemów możemy skopiować pierwsze zainteresowanie i zapisać je w kolumnie `zainteresowanie1`. Wystarczy w tym celu skorzystać z funkcji `SUBSTRING_INDEX` przedstawionej w rozdziale 5.

UPDATE moje_kontakty

`SET zainteresowanie1 = SUBSTRING_INDEX(zainteresowania, ',', 1);`

Nazwa naszej kolumny.

Znak, który będzie poszukiwany — przecinek
... interesuje nas pierwszy przecinek.

Wykonaj to polecenie, a uzyskasz następujące wyniki:

<code>zainteresowania</code>	<code>zainteresowanie1</code>	<code>zainteresowanie2</code>	<code>zainteresowanie3</code>	<code>zainteresowanie4</code>
pierwsze, drugie, trzecie, czwarte	pierwsze			

Wciąż wykazujemy zainteresowanie

Teraz przystępujemy do trudniejszej części zadania: spróbujemy zastosować inną funkcję operującą na łańcuchach znaków, by usunąć z kolumny zainteresowania fragment tekstu, który przed chwilą skopiowaliśmy do kolumny zainteresowanie1. Pozostałe kolumny zainteresowań możemy wypełnić, powtarzając dwie ostatnie operacje.

zainteresowania	zainteresowanie1	zainteresowanie2	zainteresowanie3	zainteresowanie4
(pierwsze, drugie, trzecie, czwarte	pierwsze			

Chcemy usunąć pierwsze zainteresowanie, zapisany zaraz za nim przecinek oraz znak odstępu umieszczony bezpośrednio za przecinkiem.

Zastosujemy funkcję SUBSTR, która pobierze łańcuch znaków zapisany w kolumnie zainteresowań i zwróci jego fragment. Każemy jej zwrócić fragment kolumny zainteresowania, który rozpoczyna się za fragmentem skopiowanym do kolumny zainteresowanie1 oraz umieszczonymi za nim znakami przecinka i odstępu.

Tłumaczenie: W kolumnie zainteresowania zapisz ponownie to, co jest w niej aktualnie zapisane, lecz bez początkowego fragmentu, który zapisaliśmy wcześniej w kolumnie zainteresowanie1, oraz zapisanych za nim znaków przecinka i odstępu.

UPDATE moje_kontakty

SET zainteresowania = SUBSTR(zainteresowania, LENGTH(zainteresowanie1)+3);

Funkcja SUBSTR zwraca fragment łańcucha zapisanego w kolumnie. Odczytuje ona zawartość kolumny, odcina od niej początkowy fragment o długości podanej w nawiasach i zwraca pozostałą część.

Czy pamiętasz, że działanie niektórych funkcji może być różne w zależności od używanej wersji języka SQL? SUBSTR jest przykładem właśnie takiej funkcji. Zajrzyj do jakiegoś naprawdę dobrego słownika, takiego jak „SQL Almanach”, by sprawdzić sposób działania tej funkcji w konkretnej wersji SQL-a.

To wyrażenie zwraca długość łańcucha zapisanego w kolumnie zainteresowanie1...

... powiększoną o dwa dodatkowe znaki: przecinek i odstęp.

Funkcja LENGTH zwraca liczbę określającą długość dowolnego łańcucha znaków określonego w nawiasach za nazwą funkcji.

W naszym przypadku długość stowa „pierwsze” wynosi 8 znaków.

W naszym przypadku wartość LENGTH zwróconą przez funkcję LENGTH, jest 8, do której to liczby dodajemy 2, by uzyskać w efekcie 10. A zatem 10 jest liczbą znaków, jakie zostaną usunięte z samego początku oryginalnej zawartości kolumny zainteresowania.

Aktualizacja wszystkich zainteresowań

Po wykonaniu polecenia UPDATE z poprzedniej strony nasza tabela będzie wyglądać jak na poniższym rysunku. Jednak to jeszcze nie koniec. W podobny sposób musimy zapisać wartości w kolumnach zainteresowanie2, zainteresowanie3 oraz zainteresowanie4.

zainteresowania	zainteresowanie1	zainteresowanie2	zainteresowanie3	zainteresowanie4
drugie, trzecie, czwarte	pierwsze			

Zaostrz ołówek



Wypełnij puste pola i uzupełnij polecenie UPDATE, którego Grzesiek chce użyć do zmodyfikowania tabeli. Zapisaliśmy przy nim kilka notatek, aby Ci ułatwić zadanie.

Podpowiedź: Kolumna zainteresowań będzie się zmieniać za każdym razem, gdyż zapisany w niej łańcuch znaków będzie skracany w wyniku wywołania funkcji SUBSTR.

UPDATE moje_kontakty SET
zainteresowanie1 = SUBSTRING_INDEX(zainteresowania, ',', 1),
zainteresowania = SUBSTR(zainteresowania, LENGTH(zainteresowanie1)+2),
zainteresowanie2 = SUBSTRING_INDEX(.....),
zainteresowania = SUBSTR(.....),
zainteresowanie3 = SUBSTRING_INDEX(.....),
zainteresowania = SUBSTR(.....),
zainteresowanie4 =

Kiedy już usuniesz z kolumny trzy pierwsze zainteresowania, to zostanie w niej jedynie czwarte zainteresowanie. Jaka operację powinieneś wykonać w tym miejscu?

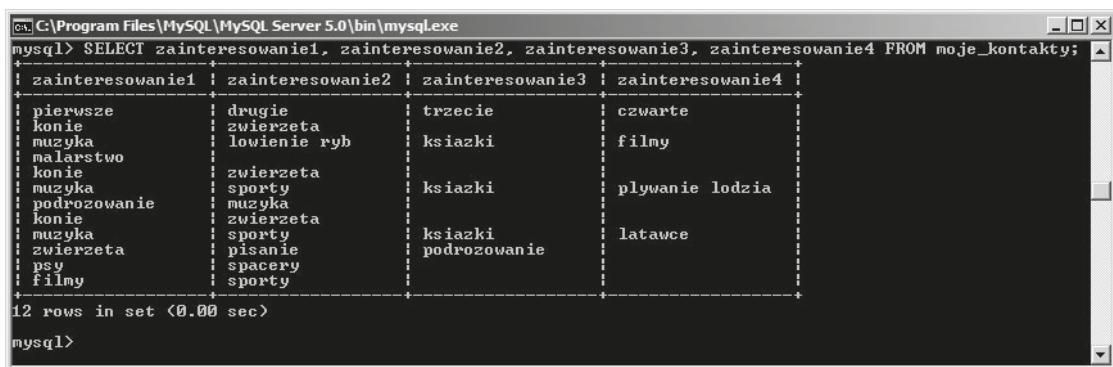
Podaj wartości poszczególnych kolumn po wykonaniu powyższego, dużego polecenia SQL.

zainteresowania	zainteresowanie1	zainteresowanie2	zainteresowanie3	zainteresowanie4
drugie, trzecie, czwarte	pierwsze			

Odpowiedzi znajdziesz na stronie 407

Pobieranie wszystkich zainteresowań

W końcu udało się nam rozdzielić poszczególne zainteresowania. Możemy je pobierać przy użyciu prostego zapytania SELECT, jednak nie możemy odwoływać się do nich wszystkich jednocześnie. Oprócz tego nie jesteśmy w stanie w łatwy sposób pobrać ich wszystkich i zwrócić w jednym zbiorze rekordów, gdyż poszczególne zainteresowania są zapisane aż w czterech kolumnach. Oto wyniki, jakie uzyskamy po próbie pobrania wszystkich zainteresowań:



```
C:\Program Files\MySQL\MySQL Server 5.0\bin>mysql.exe
mysql> SELECT zainteresowanie1, zainteresowanie2, zainteresowanie3, zainteresowanie4 FROM moje_kontakty;
+-----+-----+-----+-----+
| zainteresowanie1 | zainteresowanie2 | zainteresowanie3 | zainteresowanie4 |
+-----+-----+-----+-----+
| pierwsze konie muzyka malarstwo | drugie zwierzęta lowienie ryb | trzecie książki | czwarste filmy |
| konie muzyka podrzozowanie | zwierzęta sporty muzyka | książki | pływanie lodzia |
| muzyka zwierzęta sporty | książek | książek | latawce |
| zwierzęta pisanie spacery | sporty | książek | podrzozowanie |
| psy filmy | sporty | książek | |
+-----+-----+-----+-----+
12 rows in set (0.00 sec)

mysql>
```

Nic jednak nie stoi na przeszkodzie, byśmy napisali cztery polecenia SELECT i za ich pomocą pobrali wszystkie wartości zainteresowań.

```
SELECT zainteresowanie1 FROM moje_kontakty;
SELECT zainteresowanie2 FROM moje_kontakty;
```

```
SELECT zainteresowanie3 FROM moje_kontakty;
SELECT zainteresowanie4 FROM moje_kontakty;
```

Brakuje nam już tylko jednego — sposobu na pobranie tych wszystkich danych i zapisanie ich w naszych nowych tabelach. Operację tę można wykonać nie na jeden, lecz co najmniej na trzy sposoby!



Ćwiczenie

Wypróbuj to w domu

Przeanalizujmy przykład polecenia SELECT, które napisałeś na stronie 375.

```
SELECT zawod FROM moje_kontakty GROUP BY zawod ORDER BY zawod;
```

Na kilku następnych stronach przedstawimy **TRZY SPOSODY** pozwalające skorzystać z tych poleceń, by wypełnić naszą nową bazę danych danymi.

Warto побawić się nieco poleceniami SELECT, INSERT oraz CREATE, by poznać ich możliwości.

W tym ćwiczeniu nie chodzi o to, by osiągnąć cel w najprostszy sposób, lecz by pomyśleć i zastanowić się nad możliwościami.

Wiele dróg prowadzących w to samo miejsce

Choć jakiś zwariowany klaun mógłby uznać, że możliwość wykonania pewnej operacji na trzy lub więcej różnych sposobów to niezły odrót, to jednak dla większości z nas może to być dosyć kłopotliwe.

Niemniej jednak sytuacja ta ma też swoje zalety. Znając trzy różne sposoby dotarcia do zamierzonego celu, możemy wybrać ten, który najlepiej odpowiada naszym potrzebom. A w miarę wzrostu ilości danych sam się przekonasz, że pewne polecenia SQL bazy danych wykonują szybciej niż inne. Kiedy Twoje bazy staną się naprawdę duże, będziesz zapewne chciał zoptymalizować używane zapytania. W takiej sytuacji znajomość kilku różnych sposobów wykonania pewnej operacji może być bardzo przydatna i pomocna.

Na kilku kolejnych stronach przedstawiliśmy wszystkie trzy sposoby na zapisanie w tej tabeli wartości unikalnych i posortowanych alfabetycznie.

zawody	
id_zawodu	zawod
0	A



CREATE, SELECT oraz INSERT — (prawie) jednocześnie

1. Najpierw polecenie CREATE TABLE, a następnie INSERT wraz z SELECT

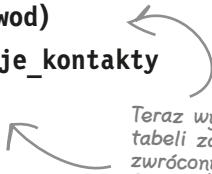
Już wiesz, jak to zrobić! Najpierw musisz utworzyć tabelę zawody, korzystając z polecenia CREATE TABLE, a następnie zapisać w jej kolumnach wartości zwrócone przez polecenie SELECT ze strony 375.

```
CREATE TABLE zawody
(
    id_zawodu INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    zawod VARCHAR(50)
);
INSERT INTO zawody (zawod)
SELECT zawod FROM moje_kontakty
GROUP BY zawod
ORDER BY zawod;
```

Tworzymy tabelę zawodów zawierającą kolumnę klucza głównego oraz kolumnę typu VARCHAR do przechowywania nazw zawodów.



Teraz wypełniamy kolumnę zawod tabeli zawody wartościami zwroconymi przez polecenie SELECT.



2. Wydanie polecenia CREATE TABLE wraz z SELECT, a następnie dodanie klucza głównego

A oto drugi sposób: Tworzymy tabelę zawody, używając do tego celu polecenia CREATE TABLE połączonego z poleceniem SELECT, które pobierze dane z kolumny zawodów tabeli moje_kontakty. Następnie modyfikujemy tabelę, dodając do niej klucz główny — w tym celu używamy polecenia ALTER.

```
CREATE TABLE zawody AS
    SELECT zawod FROM moje_kontakty
        GROUP BY zawod
        ORDER BY zawod;
ALTER TABLE zawody
    ADD COLUMN id_zawodu INT NOT NULL AUTO_INCREMENT FIRST,
    ADD PRIMARY KEY (id_zawodu);
```

Tworzymy tabelę zawody zawierającą jedną kolumnę wypełnioną danymi zwroconymi przez zapytanie SELECT...

... po czym modyfikujemy ją poleceniem ALTER, dodając do niej pole klucza głównego.

CREATE, SELECT i INSERT — jednocześnie

3. Jednoczesne utworzenie tabeli z kluczem głównym i zapisaniem w niej danych z polecenia SELECT

A oto pociąg zmierzający do celu bez żadnych przesiadek: polecenie CREATE tworzy tabelę zawody składającą się z kolumny klucza głównego oraz kolumny typu VARCHAR, w której będą przechowywane nazwy zawodów, i od razu zapisuje w niej dane zwrocone przez polecenie SELECT. Język SQL udostępnia możliwość automatycznej inkrementacji wartości pól, zatem system zarządzania bazami danych będzie wiedział, że wartości pola id_zawodu powinny być wyznaczane i zapisywane automatycznie; to z kolei sprawia, że pozostaje nam jedna kolumna, w której mamy zapisywać dane.

```
CREATE TABLE zawody
(
    id_zawodu INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    zawod VARCHAR(50)
) AS
    SELECT zawod FROM moje_kontakty
        GROUP BY zawod
        ORDER BY zawod;
```

Tworzymy tabelę zawody zawierającą kolumnę klucza głównego oraz kolumnę na nazwę zawodu. Jednocześnie zapisujemy w niej zawody pobrane za pomocą polecenia SELECT z tabeli moje_kontakty.

Nigdy wcześniej nie widziałem słowa kluczowego AS. Wygląda na to, że jest ono stosowane podczas tworzenia tabeli, by odwołać się do wyników polecenia SELECT i od razu zapisać je w nowo utworzonej tabeli.



Owszem. Słowo kluczowe AS działa dokładnie tak, jak to opisałeś.

Wszystko sprowadza się do tworzenia nazw zastępczych, którymi już niebawem się zajmiemy!

O co chodzi z tym AS?

Słowo kluczowe AS pozwala zapisać w nowej tabeli wyniki zwrócone przez polecenie SELECT. A zatem stosując je w dwóch ostatnich przykładach przedstawionych na poprzednich stronach, kazaliśmy pobrać wszystkie wartości zwrócone przez polecenie SELECT z tabeli moje_kontakty i zapisać je do nowo utworzonej tabeli zawody.

Gdybyśmy w pierwszym fragmencie polecenia nie określili, że nowa tabela ma się składać z dwóch kolumn, to zastosowanie słowa kluczowego AS spowodowałoby utworzenie tylko jednej kolumny — jej nazwa i typ danych odpowiadałyby kolumnie zwróconej przez polecenie SELECT.

Gdybyśmy sami nie zdefiniowali w nowej tabeli dwóch kolumn, to słowo kluczowe AS utworzyłoby tylko jedną kolumnę, której nazwa i typ danych odpowiadałyby nazwie i typowi danych kolumny zwróconej przez polecenie SELECT.

W naszej nowej tabeli tworzymy kolumnę typu VARCHAR o nazwie zawod.

CREATE TABLE zawody
(
 id_zawodu INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
 zawod VARCHAR(50))
)
AS
SELECT zawod **FROM** moje_kontakty
GROUP BY zawod
ORDER BY zawod;

Zastosowanie tego niepozornego słowa kluczowego daje niesamowite efekty. Dzięki niemu wszystkie wartości zwrócone przez polecenie SELECT znajdują się w nowej tabeli.

Te wszystkie odwołania znajdują się w poleceniu SELECT, dlatego też odnoszą się do kolumny zawod w tabeli moje_kontakty, gdzie znajdują się one w poleceniu SELECT.

Ponieważ utworzyliśmy tabelę zawody z kolumną klucza głównego, którego wartości są automatycznie inkrementowane, musimy zatem dodać wartości wyłącznie do drugiej kolumny naszej tabeli — kolumny zawod.



Jestem trochę skołowana.
Nazwa kolumny „zawod” pojawiła się w tym poleceniu aż cztery razy. Być może system zarządzania bazami danych „domyśli się”, o jakie kolumny chodzi, ale skąd ja mam to wiedzieć?

Język SQL pozwala określać nazwy zastępcze kolumn, nazywane też aliasami, dzięki którym wszystko stanie się proste i zrozumiałe.

To właśnie jeden z powodów, dla których SQL pozwala tymczasowo przypisywać tabelom i kolumnom nowe nazwy zastępcze — nazywane także aliasami.

Nazwy zastępcze kolumn

Trudno sobie wyobrazić, by tworzenie nazw zastępczych kolumn mogło być jeszcze prostsze. Nazwę zastępczą umieszczamy w zapytaniu z prawej strony kolumny i poprzedzamy słowem kluczowym AS, by system zarządzania bazami danych odwoływał się do kolumny zawod tabeli moje_kontakty przy użyciu jakiejś nowej nazwy, która w danej sytuacji ma dla nas większy sens.

Zdecydowaliśmy, że wartości zawodów pobierane z tabeli moje_kontakty nazwiemy **mk_zwd** (gdzie „mk” to skrót od „moje kontakty”).

```
CREATE TABLE zawody
(
    id_zawodu INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    zawod VARCHAR(50)
) AS

SELECT zawod AS mk_zwd FROM moje_kontakty
GROUP BY mk_zwd
ORDER BY mk_zwd;
```

Nazwę zastępczą umieść bezpośrednio za miejscem, w którym po raz pierwszy używasz oryginalnej nazwy kolumny — w ten sposób poinformujesz system zarządzania bazami danych, by w dalszej części polecenia odwoływać się do danej kolumny, używając podanej nazwy zastępczej.

To polecenie robi dokładnie to samo co poprzednie, jednak dzięki zastosowaniu nazwy zastępczej znacznie łatwiej jest je zrozumieć.

Obydwa polecenia różnią się jednym drobnym szczegółem. Oba zwracają wyniki w formie tabeli. **Zastosowanie nazwy zastępczej powoduje zmianę nazwy w wynikach polecenia**, natomiast w żaden sposób **nie zmienia oryginalnej nazwy tabeli**. A zatem nazwy zastępcze są tymczasowe.

Niemniej jednak, ponieważ przesłoniliśmy wyniki i zadecydowaliśmy, że nasza nowa tabela ma dwie kolumny — klucz główny oraz kolumnę z nazwami zawodów — zatem nasza wynikowa tabela będzie zawierać kolumnę **zawod**, a nie **mk_zwd**.

zawod
programista
nauczyciel
prawnik

Wyniki oryginalnego zapytania, zawierające kolumnę o nazwie **zawod**.

mk_zwd
programista
nauczyciel
prawnik

Wyniki zapytania, w którym zastosowano nazwę zastępczą. Nazwa zwrotnej kolumny odpowiada nazwie zastępczej.

Nazwy zastępcze, a kto by ich potrzebował?

Ty sam! Niebawem wskoczymy na głęboką wodę i zajmiemy się złączaniami, których będziemy używali do pobierania danych z wielu tabel jednocześnie. Gdybyś do tego czasu nie poznął nazw zastępczych, to bardzo szybko padłbyś ze zmęczenia, w nieskończoność wpisując te same nazwy tabel.

Nazwy zastępcze tabel tworzy się identycznie jak nazwy zastępcze kolumn. Wystarczy umieścić nazwę zastępczą tuż za pierwszym użyciem oryginalnej nazwy tabeli, poprzedzając ją słowem kluczowym AS, by poinformować oprogramowanie zarządzające bazą danych, że od teraz, gdy odnajdzie tabelę o nazwie mk, ma się odwoływać do tabeli moje_kontakty.

```
SELECT zawod AS mk_zwd  
FROM moje_kontakty AS mk  
GROUP BY mk_zwd  
ORDER BY mk_zwd;
```

Tworzy nazwę zastępczą tabeli dokładnie w taki sam sposób, jak nazwę zastępczą kolumny.

Czy za każdym razem, gdy chcę określić nazwę zastępczą, muszę użyć słowa kluczowego AS?

Te dwa zapytania nie różnią się pod względem działania oraz zwracanych wyników.

Nie, można użyć uproszczonego sposobu tworzenia nazw zastępczych.

Wystarczy po prostu pominąć słowo AS. Zapytanie przedstawione poniżej działa dokładnie tak samo jak poprzednie.

```
SELECT zawod mk_zwd  
FROM moje_kontakty mk  
GROUP BY mk_zwd  
ORDER BY mk_zwd;
```

Z tego zapytania usunęliśmy słowa kluczowe AS. Można tak zrobić, o ile tylko nazwy zastępcze zostaną zapisane bezpośrednio za oryginalną nazwą tabeli lub kolumny, które będą reprezentować.



Wszystko co chciałbyś wiedzieć o złączeniach wewnętrznych

Jeśli kiedykolwiek słyszałeś rozmowę o języku SQL, to zapewne usłyszałeś także coś o złączeniach. Nie są one aż tak złożone, jak mógłbyś przypuszczać. Mamy zamiar dokładnie Ci je przedstawić, pokazać, jak działają, i dać Ci mnóstwo okazji, byś mógł się dowiedzieć, kiedy należy ich używać.

I, co równie ważne, byś wiedział, jakiego złączenia użyć w danej sytuacji.

Jednak zanim się za to zabierzemy, zaczniemy od przedstawienia najprostszego złączenia (które tak naprawdę wcale nie jest złączeniem).

Istnieje kilka nazw tego typu złączenia. W niniejszej książce będziemy je określali jako **złączenie kartezańskie**, jednak można się także spotkać z takimi nazwami jak: iloczyn kartezański, iloczyn krzyżowy, złączenie krzyżowe oraz złączenie bez złączenia (a to ciekawe!).



Załóżmy, że mamy dwie tabele: jedną z imionami chłopców oraz drugą z nazwami zabawek, które ci chłopcy mają. Twoim zadaniem jest określenie, jaką zabawkę można by kupić każdemu z chłopców.

zabawki

id_zabawki	zabawka
1	hula-hoop
2	model szybowca
3	żołnierzyki
4	harmonijka
5	karty z piłkarzami

chłopcy

id_chlopca	chlopiec
1	Darek
2	Bartek
3	Daniel
4	Rysiek

Złączenie karteżjańskie

Poniższe zapytanie zwraca wyniki karteżjańskie, gdy zażądamy pobrania kolumny zabawek z tabeli zabawki oraz kolumny chłopców z tabeli chłopcy.

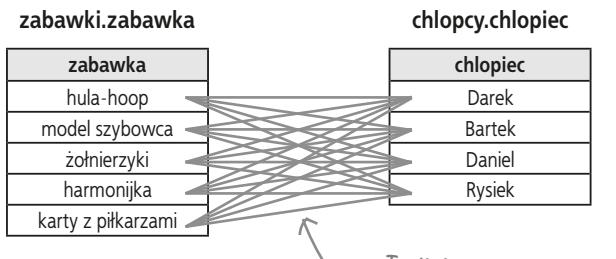
```
SELECT z.zabawka, c.chłopiec
FROM zabawki AS z
CROSS JOIN
chłopcy AS c;
```

Także tu używamy nazw zastępczych tabel.

Czy pamiętasz uproszczony zapis, który zastosowaliśmy już w poprzednim rozdziale? Nazwa przed kropką określa nazwę tabeli, a nazwa po kropce — nazwę kolumny w tej tabeli. Jednak tym razem zamiast pełnych nazw tabel używamy nazw zastępczych.

Ten wiersz polecenia informuje, że należy pobrać kolumnę „chłopiec” z tabeli chłopców oraz kolumnę „zabawka” z tabeli zabawek. Dalsza część zapytania łączy obie kolumny i tworzy nową tabelę wynikową.

Złączenie karteżjańskie pobiera każdą wartość z pierwszej tabeli i łączy ją kolejno ze wszystkimi wartościami z drugiej tabeli.



Te linie przedstawiają wyniki złączenia. Każda zabawka zostaje połączona z każdym chłopcem. W tabeli wynikowej nie ma żadnych powtórzeń.

ZŁĄCZENIE KRZYŻOWE
zwarcia każdy wiersz z pierwszej tabeli połączony kolejno ze wszystkimi wierszami z drugiej tabeli.

To złączenie zwróci 20 wierszy wynikowych.

Tworzy je 5 zabawek pomnożonych przez 4 chłopców — czyli wszystkie możliwe kombinacje.

Kolumna zabawki.zabawka ma więcej wartości niż kolumna chłopcy.chłopiec i to właśnie z tego powodu jej wartości zostały pogrupowane i umieszczone jako pierwsza kolumna. Gdyby to chłopcy było więcej niż zabawek, to oni zostaliby umieszczeni jako pierwsi. Pamiętaj jednak, że kolejność, w jakiej zostaną zwrócone wyniki, nie ma żadnego znaczenia dla tego zapytania.

zabawka	chłopiec
hula-hoop	Rysiek
hula-hoop	Darek
hula-hoop	Bartek
hula-hoop	Daniel
model szybowca	Rysiek
model szybowca	Darek
model szybowca	Bartek
model szybowca	Daniel
żołnierzyki	Rysiek

Nie istniejąca grupa pytania

P: Do czego mogłyby mi być potrzebne takie zapytanie?

O: Warto wiedzieć o takiej możliwości, gdyż może się zdarzyć, że stosując złączenia, czasami uzyskamy wyniki kartezjańskie. W takiej sytuacji będziesz wiedział, jak poprawić swoje złączenie. W niektórych sytuacjach ta wiedza naprawdę może się bardzo przydać. Oprócz tego, złączenia kartezjańskie są czasami używane do testowania szybkości systemu zarządzania relacyjnymi bazami danych oraz jego konfiguracji. W przypadku stosowania wolnych zapytań czasы ich wykonywania można łatwiej określić i porównywać.

P: Założymy, że użyłbym zapytania: **SELECT * FROM zabawki CROSS JOIN chłopcy;**. Co by się stało, gdybym zastosował klauzulę **SELECT *?**

O: Sam powinieneś to sprawdzić. Jednak i tak uzyskasz w efekcie 20 wierszy; ale tym razem wyniki będą się składać nie z 2, lecz z 4 kolumn.

P: A co się stanie, jeśli połączę w ten sposób dwie bardzo duże tabele?

O: Cóż... uzyskasz **gigantyczną** ilość wyników. Najlepiej nie łączyć w ten sposób dużych tabel, gdyż w przeciwnym razie ryzykujemy „zawieszenie” komputera, który będzie musiał zwrócić ogromne ilości danych wynikowych.

P: Czy takie zapytanie można zapisać w inny sposób?

O: Ależ oczywiście. Zamiast słów CROSS JOIN możesz zapisać przecinek; oto przykład: **SELECT chłopcy.chłopiec, zabawki.zabawka FROM zabawki, chłopcy;**.

P: Usłyszałem gdzieś terminy „złączenie wewnętrzne” oraz „złączenie zewnętrzne”. Czy złączenie kartezjańskie to to samo?

O: Złączenie kartezjańskie jest jednym z typów złączeń wewnętrznych. Najprościej rzec ujmując, złączenie wewnętrzne jest złączeniem kartezjańskim, z którego w wyniku zastosowania warunku podanego w zapytaniu usunięto niektóre wiersze. Złączeniami wewnętrznymi będziemy się zajmować na kilku kolejnych stronach, a zatem pamiętaj o tym.

**ZŁĄCZENIE WEWNĘTRZNE
jest ZŁĄCZENIEM
KRZYŻOWYM, z którego
w wyniku zastosowania
warunku podanego w zapytaniu
usunięto niektóre wiersze.**



**WYTĘŻ
UMYSŁ**

Jak sądzisz, jakie będą wyniki wykonania następującego zapytania:

```
SELECT b1.chłopiec, b2.chłopiec
FROM chłopcy AS b1 CROSS JOIN chłopcy AS b2;
```

Spróbuj wykonać to zapytanie.

Zaostrz ołówek

Zaostrz ołówek



zawody
id_zawodu
zawod

moje_kontakty
id_kontaktu
nazwisko
imie
telefon
email
plec
data_urodzenia
id_zawodu
kod_pocztowy
id_stanu

Oto dwie tabele z nowej bazy danych `lista_grzesia` — zawody oraz moje_kontakty. Przeanalizuj zapytanie przedstawione poniżej i w miejscach oznaczonych kropkami zapisz, do czego według Ciebie służy każdy z wierszy polecenia.

```
SELECT mk.nazwisko,  
.....  
mk.imie,  
.....  
z.zawod  
.....  
FROM moje_kontakty AS mk  
.....  
INNER JOIN  
.....  
zawody AS z  
.....  
ON mk.id_zawodu = z.id_zawodu;  
.....
```

Załącz, że w tabelach znajdują się dane przepisane z poniższych trzech karteczek.
Naszkicuj, jak mogłyby wyglądać tabela wynikowa.

Janina Wielicka

samotna

4.3.1978

Wrocław, DS

artystka

kobieta

janka@potegawizji.net.pl

żeglarstwo, spacery, gotowanie

555 444-3333

Tamara Barańska

mężatka

9.1.1970

Gdańsk, PM

kucharka

kobieta

tama@pizza-nzk.com.pl

filmy, jazda konna, gotowanie

555 555-4441

Paweł Sobczyk

żonaty

12.10.1980

Warszawa, MZ

nauczyciel

mężczyzna

ps@niebianskaplaza.org.pl

psy, speleologia

555 555-9876

Rozwiążanie ćwiczenia

Zaostrz ołówek



Rozwiążanie

zawody
id_zawodu
zawod

moje_kontakty
id_kontaktu
nazwisko
imie
telefon
email
plec
data_urodzenia
→ id_zawodu
kod_poczтовy
id_stanu

Oto dwie tabele z nowej bazy danych `lista_grzesia` — zawody oraz `moje_kontakty`. Przeanalizuj zapytanie przedstawione poniżej i w miejscach oznaczonych kropkami zapisz, do czego według Ciebie służy każdy z wierszy polecenia.

```
SELECT mk.nazwisko,  
       mk.imie,  
       z.zawod  
FROM moje_kontakty AS mk  
     INNER JOIN  
           zawody AS z  
      ON mk.id_zawodu = z.id_zawodu;
```

Pobiera kolumnę nazwisko z tabeli `moje_kontakty` (posługując się przy tym nazwą zastępczą `mk`)

... oraz kolumnę imie z tabeli `moje_kontakty` ...

... jak również kolumnę zawod z tabeli `zawody` (posługując się przy tym nazwą zastępczą `z`).

Dane należy pobrać z tabeli `moje_kontakty` (dla której utworzono nazwę zastępczą `mk`), a oprócz tego należy utworzyć złączenie wewnętrzne, by ...

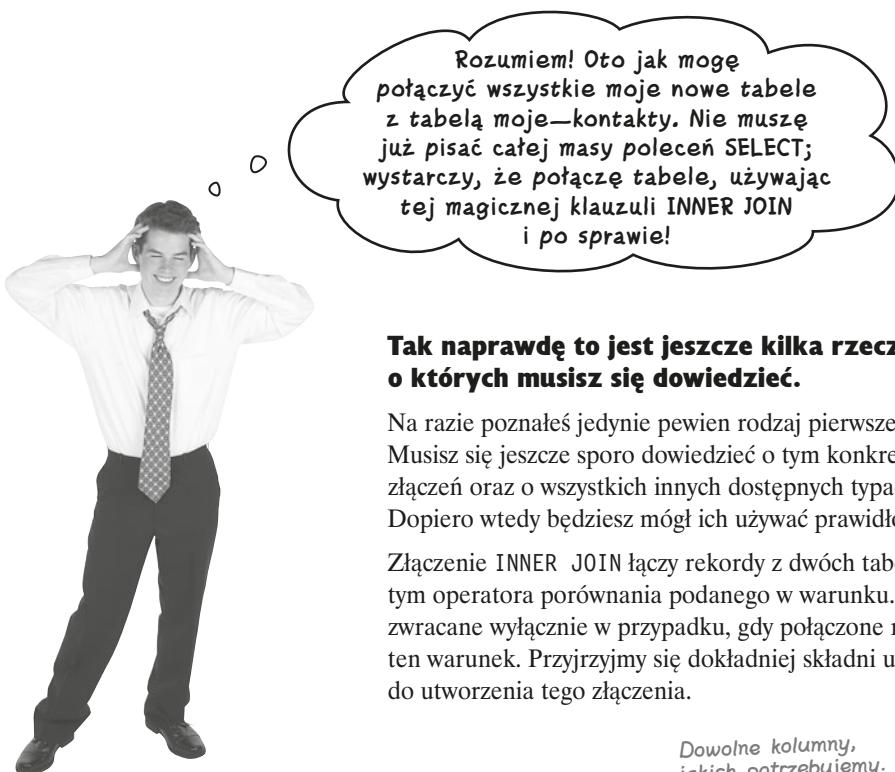
... połączyć osoby z zawodami i pobrać z tabeli `zawody` (dla której utworzono nazwę zastępczą `z`) te zawody

... dla których kolumna `id_zawodu` w tabeli `moje_kontakty` ma taką samą wartość co kolumna `id_zawodu` w tabeli `zawody`.

Założź, że w tabelach znajdują się dane przepisane z poniższych trzech karteczek. Naszkicuj, jak mogłaby wyglądać tabela wynikowa.

nazwisko	imie	zawod
Wielicka	Janina	artystka
Barańska	Tamara	kucharka
Sobczyk	Paweł	nauczyciel

Zrozumienie złączeń wewnętrznych



Tak naprawdę to jest jeszcze kilka rzeczy, o których musisz się dowiedzieć.

Na razie poznałeś jedynie pewien rodzaj pierwszego typu złączeń. Musisz się jeszcze sporo dowiedzieć o tym konkretnym rodzaju złączeń oraz o wszystkich innych dostępnych typach złączeń. Dopiero wtedy będziesz mógł ich używać prawidłowo i efektywnie.

Złączenie INNER JOIN łączy rekordy z dwóch tabel, używając przy tym operatora porównania podanego w warunku. Kolumny są zwracane wyłącznie w przypadku, gdy połączone rekordy spełniają ten warunek. Przyjrzymy się dokładniej składni używanej do utworzenia tego złączenia.

```

    Dowolne kolumny,
    jakich potrzebujemy.

SELECT jakies_kolumny
    ↓
    FROM tabela1
        ←
    INNER JOIN
    tabela2
        ←
    ON jakis_warunek;
    ↑
    Pominieliśmy nazwy
    zastępcze, by uprościć
    kod polecenia.

    ←
    Tu można także
    umieścić słowo
    kluczowe WHERE.

    W tym warunku można
    zastosować dowolny
    operator porównania.
  
```

Złączenie INNER JOIN
łączy rekordy z dwóch tabel,
wykorzystując przy tym operator
porównania podany w warunku.

Złączenie wewnętrzne w akcji: złączenie równościowe

Wyobraźmy sobie bazę danych zawierającą przedstawione poniżej tabele. Występuje pomiędzy nimi zależność jeden-do-jednego, a kolumna `id_zabawki` pełni rolę klucza obcego.

chłopcy

<code>id_chlopca</code>	chłopiec	<code>id_zabawki</code>
1	Darek	3
2	Bartek	5
3	Daniel	2
4	Rysiek	1

zabawki

<code>id_zabawki</code>	zabawka
1	hula-hoop
2	model szybowca
3	żołnierzyki
4	harmonijka
5	karty z piłkarzami

Interesuje nas tylko jedno — mamy określić, jaką zabawkę posiada każdy z chłopców. Możemy zastosować złączenie INNER JOIN z operatorem równości (=), by dopasować klucz obcy z tabeli chłopców z kluczem głównym tabeli zabawek i przekonać się, które rekordy z obu tabel będą sobie odpowiadać.

ZŁĄCZENIE RÓWNOŚCIOWE
(ang. equijoin)
wykorzystuje warunek równości.

```
SELECT chłopcy.chłopiec, zabawki.zabawka
FROM chłopcy
INNER JOIN
zabawki
ON chłopcy.id_zabawki = zabawki.id_zabawki;
```

<code>id_chlopca</code>	chłopiec	<code>id_zabawki</code>
1	Darek	3
2	Bartek	5
3	Daniel	2
4	Rysiek	1

<code>id_zabawki</code>	zabawka
1	hula-hoop
2	model szybowca
3	żołnierzyki
4	harmonijka
5	karty z piłkarzami

Oto nasza tabela wynikowa.
Gdybyśmy chcieli, to moglibyśmy dodać do zapytania klauzulę
`ORDER BY chłopcy.chłopiec.`

chłopiec	zabawka
Rysiek	hula-hoop
Daniel	model szybowca
Darek	żołnierzyki
Bartek	karty z piłkarzami

Zaostrz ołówek



Napisz zapytania wykorzystujące złączenie równościowe do pobierania opisanych poniżej informacji z bazy danych Grześka.

Zapytanie, które zwraca adres poczty elektronicznej i zawód każdej osoby zapisanej w tabeli moje_kontakty.

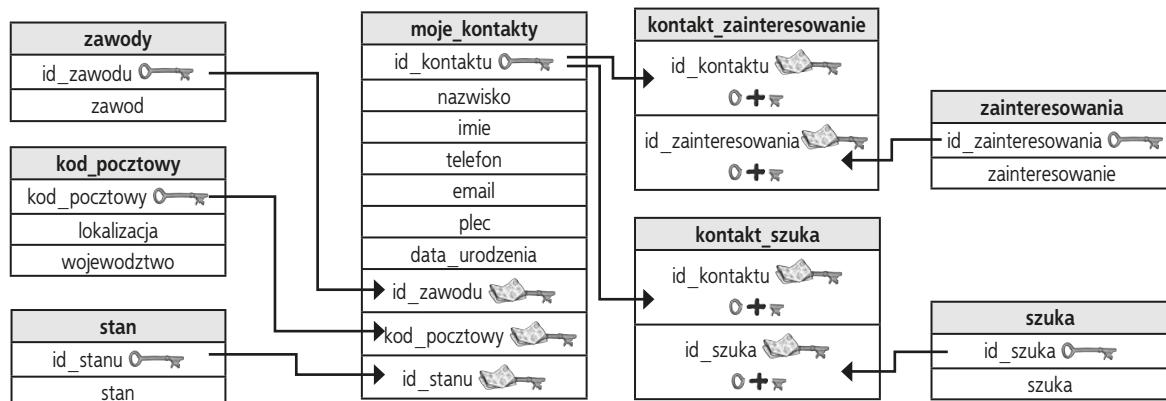
.....
.....
.....

Zapytanie, które zwraca imię, nazwisko oraz stan cywilny każdej osoby zapisanej w tabeli moje_kontakty.

.....
.....
.....

Zapytanie, które zwraca imię, nazwisko oraz województwo każdej osoby zapisanej w tabeli moje_kontakty.

.....
.....
.....



Kolejne rozwiązywanie ćwiczenia

Zaostrz ołówek

Rozwiązywanie

Napisz zapytania wykorzystujące złączenie równościowe do pobierania opisanych poniżej informacji z bazy danych Grześka.

Zapytanie, które zwraca adres poczty elektronicznej i zawód każdej osoby zapisanej w tabeli moje_kontakty.

SELECT mk.email, z.zawod FROM moje_kontakty mk

INNER JOIN zawody z ON mk.id_zawodu = z.id_zawodu;

Klucz obcy id_zawodu łączy wiersze z tabeli kontaktów z kolumną id_zawodu tabeli zawody.

Zapytanie, które zwraca imię, nazwisko oraz stan cywilny każdej osoby zapisanej w tabeli moje_kontakty.

SELECT mk.imie, mk.nazwisko, s.stan FROM moje_kontakty mk

INNER JOIN stan s ON mk.id_stanu = s.id_stanu;

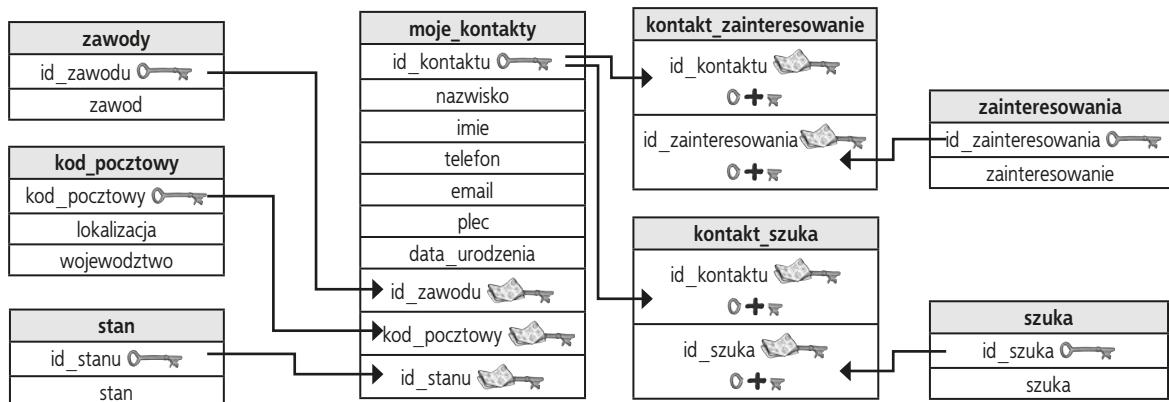
Klucz obcy id_stanu łączy wiersze z tabeli kontaktów z kolumną id_stanu tabeli stan.

Zapytanie, które zwraca imię, nazwisko oraz województwo każdej osoby zapisanej w tabeli moje_kontakty.

SELECT mk.imie, mk.nazwisko, kp.wojewodztwo FROM moje_kontakty mk

INNER JOIN kod_pocztowy kp ON mk.kod_pocztowy = kp.kod_pocztowy;

W tym przypadku używamy kodu pocztowego jako klucza łączącego obie tabele.



Złączenie wewnętrzne w akcji: złączenie różnosciowe

Złączenie różnosciowe zwraca wiersze, które nie są równe. Wyobraźmy sobie te same dwie tabele — chłopów i zabawek. Dzięki złączeniu różnosciobowemu możemy błyskawicznie określić, jakich zabawek *nie mają* poszczególni chłopcy (a takie informacje mogłyby się przydać przed ich urodzinami).

```
SELECT chłopcy.chłopiec, zabawki.zabawka
FROM chłopcy
INNER JOIN
zabawki
ON chłopcy.id_zabawki <> zabawki.id_zabawki
ORDER BY chłopcy.chłopiec;
```

Posortowanie wyników utatwi ich późniejszą analizę.

Operator „różny”. To właśnie ta część złączenia jest odpowiedzialna za jego „różnosciowy” charakter.

id_chłopca	chłopiec	id_zabawki
1	Darek	3
2	Bartek	5
3	Daniel	2
4	Rysiek	1

id_zabawki	zabawka
1	hula-hoop
2	model szybowca
3	żołnierzyki
4	harmonijka
5	karty z piłkarzami

chłopiec	zabawka
Daniel	hula-hoop
Daniel	żołnierzyki
Daniel	harmonijka
Daniel	karty z piłkarzami
Bartek	żołnierzyki
Bartek	harmonijka
Bartek	hula-hoop
Bartek	model szybowca
Darek	hula-hoop
Darek	model szybowca
Darek	harmonijka
Darek	karty z piłkarzami
Rysiek	model szybowca
Rysiek	żołnierzyki
Rysiek	harmonijka
Rysiek	karty z piłkarzami

Oto cztery zabawki, których Daniel nie ma.

**RÓŻNOSCIOWE
złączenia wewnętrzne
wykorzystują
warunek różny od.**

Ostatni rodzaj złączeń wewnętrznych: złączenia naturalne

Pozostał nam do przedstawienia jeszcze tylko jeden rodzaj złączeń wewnętrznych, nazywany **złączaniami naturalnymi**. Działają one wyłącznie w przypadku, gdy kolumny używane do złączenia mają taką samą nazwę w obu tabelach.

Ponownie skorzystajmy z przykładu tabel chłopców i zabawek.

chłopcy

id_chlopca	chłopiec	id_zabawki
1	Darek	3
2	Bartek	5
3	Daniel	2
4	Rysiek	1

zabawki

id_zabawki	zabawka
1	hula-hoop
2	model szybowca
3	żołnierzyki
4	harmonijka
5	karty z piłkarzami

Także w tym przykładzie chcemy się dowiedzieć, jakie zabawki posiadają poszczególni chłopcy. Nasze złączenie naturalne rozpozna kolumnę, która w obu tabelach ma identyczną nazwę, i zwróci pasujące do siebie rekordy.

```
SELECT chłopcy.chłopiec, zabawki.zabawka
FROM chłopcy
NATURAL JOIN
zabawki;
```

chłopcy

id_chlopca	chłopiec	id_zabawki
1	Darek	3
2	Bartek	5
3	Daniel	2
4	Rysiek	1

zabawki

id_zabawki	zabawka
1	hula-hoop
2	model szybowca
3	żołnierzyki
4	harmonijka
5	karty z piłkarzami

Jak widać, uzyskane wyniki są takie same jak w przypadku zastosowania pierwszego przedstawionego złączenia wewnętrznego, czyli złączenia równościowego.

chłopiec **zabawka**

Rysiek	hula-hoop
Daniel	model szybowca
Darek	żołnierzyki
Bartek	karty z piłkarzami

ZŁĄCZENIA NATURALNE
rozpoznają pasujące do siebie nazwy kolumn.

Zaostrz ołówek



Napisz zapytania wykorzystujące złączenie naturalne lub różnicowe do pobierania opisanych poniżej informacji z bazy danych Grześka.

Zapytanie, które zwraca adres poczty elektronicznej i zawód każdej osoby zapisanej w tabeli moje_kontakty.

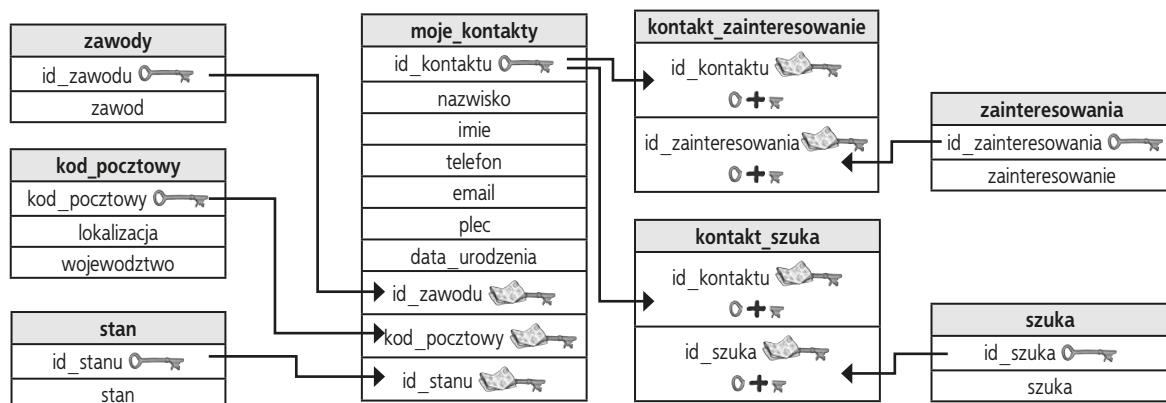
.....
.....
.....

Zapytanie, które zwraca imię, nazwisko oraz stan cywilny, którego nie mają poszczególne osoby zapisane w tabeli moje_kontakty.

.....
.....
.....

Zapytanie, które zwraca imię, nazwisko oraz województwo każdej osoby zapisanej w tabeli moje_kontakty.

.....
.....
.....



Chyba pokocham te rozwiązania ćwiczeń

Zaostrz ołówek

Rozwiążanie

Napisz zapytania wykorzystującełączenie naturalne lub różnościowe do pobierania opisanych poniżej informacji z bazy danych Grześka.

Zapytanie, które zwraca adres poczty elektronicznej i zawód każdej osoby zapisanej w tabeli `moje_kontakty`.

`SELECT mk.email, z.zawod FROM moje_kontakty mk`

`NATURAL JOIN zawody;`

Zapytanie, które zwraca imię, nazwisko oraz stan cywilny, którego nie mają poszczególne osoby zapisane w tabeli `moje_kontakty`.

`SELECT mk.imie, mk.nazwisko, s.stan FROM moje_kontakty mk`

`INNER JOIN stan s ON mk.id_stanu < > s.id_stanu;`

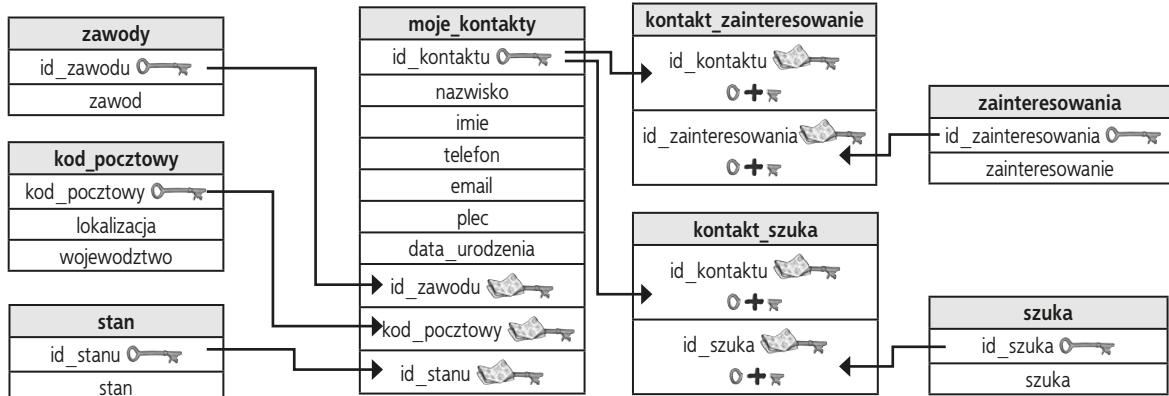
To zapytanie zwróci wiele wierszy dla każdej osoby zapisanej w tabeli `moje_kontakty`. Każda osoba zostanie połączona ze wszystkimi stanami oprócz tego, którego identyfikator jest zapisany w tabeli `moje_kontakty`, w wierszu danej osoby.

Zapytanie, które zwraca imię, nazwisko oraz województwo każdej osoby zapisanej w tabeli `moje_kontakty`.

`SELECT mk.imie, mk.nazwisko, kp.wojewodztwo FROM moje_kontakty mk`

`NATURAL JOIN kod_pocztowy kp;`

W pierwszym i ostatnim zapytaniu nie musimy stosować słowa kluczowego `ON`, gdyż nazwy kolumn klucza obcego oraz klucza głównego w łączonych tabelach odpowiadają sobie.



KTO CO ROBI?

Dopasuj każde ze złączeń podanych w lewej kolumnie z opisami ich funkcjonowania zapisanymi w kolumnie prawej. Z każdym opisem może być połączonych więcej niż jeden typ złączenia.

złączenie naturalne

Zwracam wszystkie wiersze w przypadkach, gdy kolumna z pierwszej tabeli nie odpowiada kolumnie z drugiej tabeli.

złączenie równościowe

Zwracam uwagę na kolejność, w jakiej zostały połączone tabele.

złączenie krzyżowe

Zwracam wszystkie wiersze, gdy kolumna z pierwszej tabeli odpowiada kolumnie z drugiej tabeli; a oprócz tego używam słowa kluczowego ON.

złączenie zewnętrzne

Łączę dwie tabele, które mają kolumnę o tej samej nazwie.

złączenie różnociowe

Mogę zwrócić wiersze będące iloczynem wierszy dwóch tabel.

złączenie wewnętrzne

Zwracam wszystkie możliwe wiersze i nie stawiam żadnych warunków.

iloczyn krzyżowy

Łączę dwie tabele pod pewnymi warunkami.

KTO CO ROBI?

ROZWIĄZANIE

Dopasuj każde ze złączeń podanych w lewej kolumnie z opisami ich funkcjonowania zapisanymi w kolumnie prawej. Z każdym opisem może być połączonych więcej niż jeden typ złączenia.

złączenie naturalne

Zwracam wszystkie wiersze w przypadkach, gdy kolumna z pierwszej tabeli nie odpowiada kolumnie z drugiej tabeli.

złączenie równościowe

Zwracam uwagę na kolejność, w jakiej zostały złączone tabele.

To zagadnienie opiszemy w rozdziale 10.

złączenie krzyżowe

Zwracam wszystkie wiersze, gdy kolumna z pierwszej tabeli odpowiada kolumnie z drugiej tabeli; a oprócz tego używam słowa kluczowego ON.

złączenie zewnętrzne

Łączę dwie tabele, które mają kolumnę o tej samej nazwie.

złączenie różnościowe

Mogę zwrócić wiersze będące iloczynem wierszy dwóch tabel.

złączenie wewnętrzne

Zwracam wszystkie możliwe wiersze i nie stawiam żadnych warunków.

złączenie kartezjańskie

Łączę dwie tabele pod pewnymi warunkami.

iloczyn krzyżowy



Ćwiczenie

Na podstawie przedstawionego u dołu strony schematu bazy danych `lista_grzesia` napisz zapytania, które zwrócą żądane informacje.

Napisz dwa zapytania, każde wykorzystujące inny rodzaj złączenia, które zwrócią pasujące do siebie wiersze tabel `moje_kontakty` oraz `kontakt_zainteresowanie`.

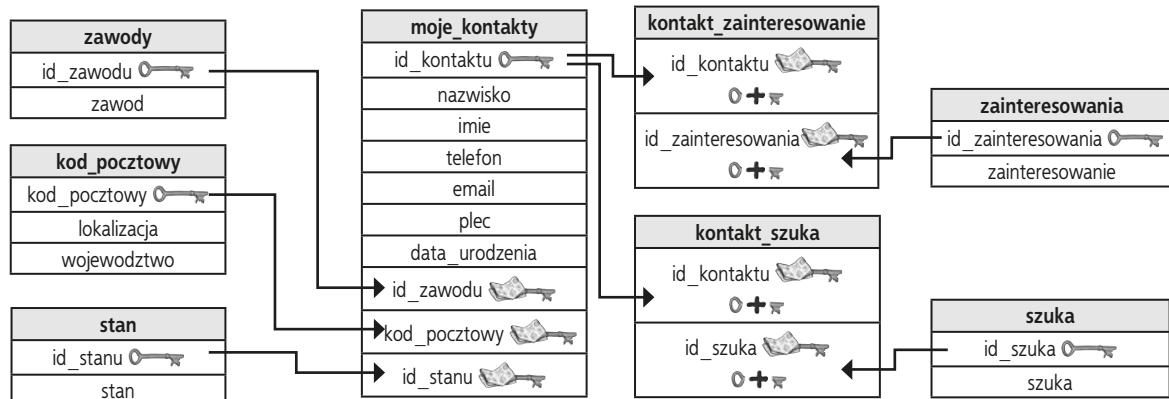
.....
.....
.....

Napisz zapytanie, które zwróci wszystkie możliwe kombinacje wierszy z tabel `kontakt_szuka` oraz `szuka`.

.....
.....
.....

Utwórz listę zawodów, w jakich pracują osoby zapisane w tabeli `moje_kontakty`; zawody na liście nie mogą się powtarzać i mają być posortowane alfabetycznie.

.....
.....
.....



Rozwiążanie ćwiczenia



Rozwiążanie ćwiczenia

Na podstawie przedstawionego u dołu strony schematu bazy danych `lista_grzesia` napisz zapytania, które zwrócią żądane informacje.

Napisz dwa zapytania, każde wykorzystujące inny rodzajłączenia, które zwrócią pasujące do siebie wiersze tabel `moje_kontakty` oraz `kontakt_zainteresowanie`.

```
SELECT mk.imie, mk.nazwisko, kz.id_zainteresowania FROM moje_kontakty mk  
INNER JOIN kontakt_zainteresowanie kz ON mk.id_kontaktu = kz.id_kontaktu;
```

```
SELECT mk.imie, mk.nazwisko, kz.id_zainteresowania FROM moje_kontakty mk  
NATURAL JOIN kontakt_zainteresowanie kz;
```

Napisz zapytanie, które zwróci wszystkie możliwe kombinacje wierszy z tabel `kontakt_szuka` oraz `szuka`.

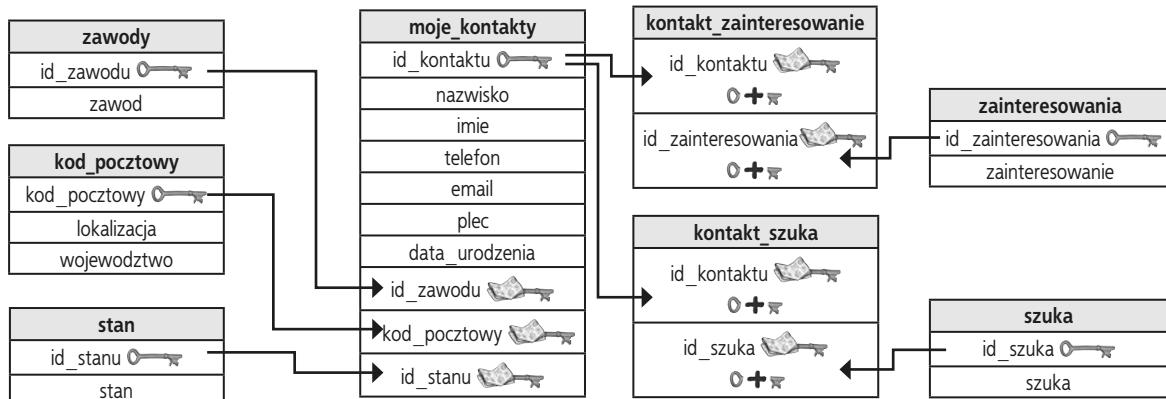
```
SELECT * FROM kontakt_szuka CROSS JOIN szuka;
```

```
SELECT * FROM kontakt_szuka, szuka;
```

Oba te zapisy pozwalają utworzyć to samo złączenie krzyżowe.

Utwórz listę zawodów, w jakich pracują osoby zapisane w tabeli `moje_kontakty`; zawody na liście nie mogą się powtarzać i mają być posortowane alfabetycznie.

```
SELECT z.zawod FROM moje_kontakty mk  
INNER JOIN zawody z ON mk.id_zawodu = z.id_zawodu GROUP BY zawod ORDER BY zawod;
```



Nieistniejąca grupa pytań

P: Czy można łączyć ze sobą więcej niż dwie tabele?

O: Można; zajmiemy się tym zagadnieniem w dalszej części książki. Na razie skoncentrujemy się na wyjaśnieniu pojęcia łączenia.

P: Czy łączenia nie powinny być nieco bardziej skomplikowane?

O: Kiedy zaczynasz stosować łączenia i nazwy zastępcze, tworzone polecenia SQL przestaną już w tak dużym stopniu przypominać codziennej, naturalny język. Także stosowanie uproszczonych form zapisu (takich jak zastępowanie słów kluczowych INNER JOIN przecinkiem) może dodatkowo utrudnić zrozumienie zapytań. Dlatego też w niniejszej książce będziemy raczej stosowali pełne — dłuższe formy zapisu polecień SQL.

P: Czy to oznacza, że zapytania ze złączami wewnętrznymi można zapisywać w inny sposób?

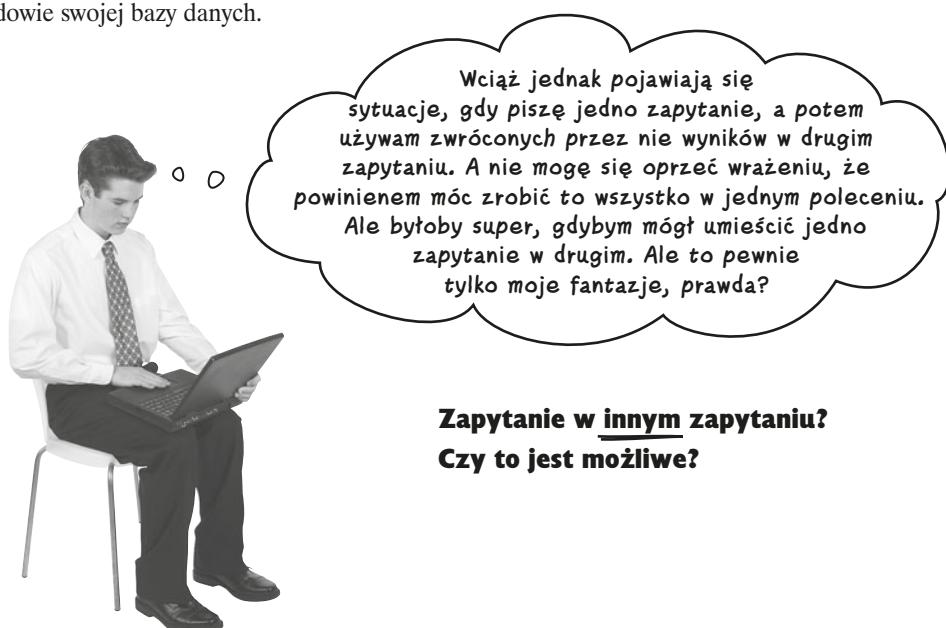
O: Owszem, można. Jeśli jednak zrozumiesz składnię polecień prezentowaną w tej książce, to przedstawienie się na inną składnię nie przysporzy Ci najmniejszych problemów. Idea łączenia jest bowiem znacznie bardziej złożona i trudna niż stosowanie słów kluczowych WHERE lub ON.

P: Zauważylem, że zastosowaliście w złączaniu klauzulę ORDER BY. Czy to oznacza, że w zapytaniach ze złączami można także stosować inne klauzule SQL-a?

O: Tak. Nic nie stoi na przeszkodzie, byś używał klauzul GROUP BY, WHERE oraz takich funkcji jak SUM albo AVG.

Złączone zapytania?

Grzesiek powoli zaczyna doceniać złączenia. Zaczyna także rozumieć, że rozdzielenie danych na wiele różnych tabel ma głęboki sens, a jeśli struktura tych tabel jest prawidłowo zaprojektowana, to korzystanie z nich nie stanowi problemu. Grzesiek zaczął nawet myśleć o rozbudowie swojej bazy danych.





Nazwy zastępcze tabel i kolumn bez tajemnic

Tematem dzisiejszego wywiadu jest:
Dlaczego się ukrywacie?

Rusz głową!: Witamy w programie Nazwę Zastępczą Tabeli oraz Nazwę Zastępczą Kolumny. Cieszę się, że mogłyście przyjść obie. Nam nadzieję, że spotkanie z wami sporo nam wyjaśni.

Nazwa Zastępca Tabeli: Właśnie, my również się cieszymy, że mogłyśmy tu przyjść. A tak w ogóle, dla ułatwienia, możesz nas nazywać NZT i NZK; oczywiście tylko podczas tego wywiadu [śmiech].

Rusz głową!: Cha, cha! Bez wątpienia to będzie właściwe. No dobrze, NZK, zaczniemy od ciebie. Wyjaśnij nam, po co te wszystkie tajemnice? Czy próbujesz coś ukryć?

Nazwa Zastępca Kolumny: Ależ oczywiście, że nie próbuję! Jeśli miałam jakikolwiek cel, to była nim wyłącznie chęć uproszczenia poleceń. Sądzę, że mogę to powiedzieć w imieniu nas obu, prawda NZT?

NZT: Jasne. W przypadku NZK jej zamiary i motywacja powinny już być zrozumiałe. Po prostu zmienia długie i powtarzające się nazwy kolumn na coś, co łatwo zastosować. Czyli ułatwia życie. Oprócz tego NZK sprawia, że nazwy kolumn w tabelach wynikowych mogą być takie, jak chcecie. W moim przypadku sprawa wygląda natomiast nieco inaczej.

Rusz głowa!: Muszę przyznać, NZT, że twojej problematyki nie znam aż tak dobrze. Widziałem, że się pojawiłeś w zapytaniach i radzisz sobie doskonale, ale tak do końca to nie wiem, co właściwie robisz. Nawet jeśli pojawiłeś się w poleceniu SQL, to w wynikach nigdy nie można cię zauważyc.

NZT: No tak, to prawda. Jednak z tego, co widzę, to jeszcze nie zrozumiałeś mojego prawdziwego przeznaczenia.

Rusz głowa!: Prawdziwego przeznaczenia? Brzmi intrygująco. Powiedz coś więcej na ten temat, proszę.

NZT: Bo widzisz, ja istnieję po to, by ułatwić tworzenie złączeń.

NZK: Dodaj jeszcze NZT, że mnie także pomagasz w tych złączeniach.

Rusz głowa!: Nie rozumiem. Czy możecie mi to pokazać na jakimś przykładzie?

NZT: Jasne; mogę ci pokazać kod. Mam nadzieję, że na jego przykładzie stanie się jasne, do czego służę:

```
SELECT mk.nazwisko, mk.imie, z.zawod  
FROM moje_kontakty AS mk  
INNER JOIN  
zawody AS z  
WHERE mk.id_kontaktu = z.id_zawodu;
```

Rusz głowa!: Rozumiem! Dzięki tobie wszędzie, gdzie musiałbym wpisywać moje_kontakty, mogę teraz wpisać mk. A zamiast zawody — po prostu z. To znacznie prostsze. Takie rozwiązywanie naprawdę jest bardzo przydatne, gdy w zapytaniu trzeba kilka razy używać nazw dwóch tabel.

NZT: Zwłaszcza jeśli tabele mają podobne nazwy. Ułatwienie zrozumienia zapytań nie tylko pomaga w ich tworzeniu, lecz także pomaga zapamiętać, co one robią, i poprawiać lub modyfikować je po jakimś czasie.

Rusz głowa!: Dziękuję wam bardzo, NZT i NZK. Było mi... zaraz, a gdzie one zniknęły?

Zaostrz ołówek**Rozwiązańe
ze str. 378**

Już wiesz, jak można modyfikować tabele, używając w tym celu polecenia ALTER. A zatem dodaj do tabeli moje_kontakty cztery nowe kolumny. Nadaj im odpowiednio nazwy: zainteresowanie1, zainteresowanie2, zainteresowanie3 oraz zainteresowanie4.

ALTER TABLE moje_kontakty

```
ADD (zainteresowanie1 VARCHAR(30), zainteresowanie2 VARCHAR(30),
      zainteresowanie3 VARCHAR(30), zainteresowanie4 VARCHAR(30);
```

Zaostrz ołówek**Rozwiązańe
ze str. 380**

Wypełnij puste pola i uzupełnij polecenie UPDATE, którego Grzesiek chce użyć do zmodyfikowania tabeli. Zapisaliśmy przy nim kilka notatek, aby Ci ułatwić zadanie.

Różnica pomiędzy funkcjami SUBSTRING_INDEX oraz SUBSTR polega na tym, że pierwsza z nich szuka jednegołańcucha znaków "wewnątrz" drugiego – w tym przypadku szuka przecinka w kolumnie zainteresowań – i zwraca wszystko, co znajdowało się przed nim. Z kolei funkcja SUBSTR skraca długośćłańcucha – w tym przypadku skraca zawartość kolumny zainteresowań, zostawiając w niej jedynie to, co jest zapisane za pierwszym zainteresowaniem, przecinkiem i odstępem (dlatego pojawiło się wyrażenie +2).

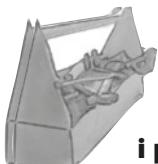
UPDATE moje_kontakty SET

```
zainteresowanie1 = SUBSTRING_INDEX(zainteresowania, ',', 1),
zainteresowania = SUBSTR(zainteresowania, LENGTH(zainteresowanie1)+2),
zainteresowanie2 = SUBSTRING_INDEX(..zainteresowania,'.',1.....),
zainteresowania = SUBSTR( zainteresowania, LENGTH(zainteresowanie2)+2 .. ),
zainteresowanie3 = SUBSTRING_INDEX( .zainteresowania,'.',1.....),
zainteresowania = SUBSTR( zainteresowania, LENGTH(zainteresowanie3)+2 .. ),
zainteresowanie4 = ..zainteresowania;
```

Po wykonaniu powyższego polecenia oryginalna kolumna zainteresowania będzie pusta.

Kiedy z kolumny usunęliśmy trzy pierwsze zainteresowania, zostało w niej jedynie ostatnie – czwarte. Ten wiersz odpowiada zatem zwyczajnemu przeniesieniu wartości do innej kolumny. Również dobrze mogliśmy pominąć tworzenie nowej kolumny dla czwartego zainteresowania i zmienić nazwę oryginalnej kolumny zainteresowań.

zainteresowania	zainteresowanie1	zainteresowanie2	zainteresowanie3	zainteresowanie4
drugie, trzecie, czwarte	pierwsze	drugie	trzecie	czwarte



Przybornik SQL

Właśnie skończyłeś przerabiać materiał z ósmego rozdziału książki i posługujesz się złączniami jak prawdziwy profesjonalista. Przypomnij sobie na koniec wszystkie techniki, których się nauczyłeś. Kompletną listę porad znajdziesz w dodatku C, zamieszczonym na końcu książki.

INNER JOIN

Dowolne złączenie zawierające wiersze z obu tabel i wykorzystujące jakiś warunek logiczny.

NATURAL JOIN

Złączenie wewnętrzne, w którym pominięto klauzulę ON. Działa jedynie w przypadku, gdy taczone tabele zawierają kolumnę o takiej samej nazwie.

Złączenie RÓWNOŚCIOWE i RÓŻNOŚCIOWE

To dwa rodzaje złączeń wewnętrznych. Pierwsze z nich zwraca wiersze, które są sobie równe, a drugie — wiersze, które są od siebie różne.

Złączenie KRZYŻOWE (CROSS JOIN)

Zwraca każdy wiersz z pierwszej tabeli, połączony kolejno z każdym wierszem z drugiej tabeli. Znane także pod wieloma innymi nazwami, takimi jak: ZŁĄCZENIE KARTEZJAŃSKIE oraz ZŁĄCZENIE BEZ ZŁĄCZENIA.

Złączenie z PRZECINKIEM

To samo co złączenie KRZYŻOWE, z tym że zamiast słów kluczowych CROSS JOIN w zapytaniu jest zapisywany przecinek.

9. Podzapytania

Zapytania w zapytaniach



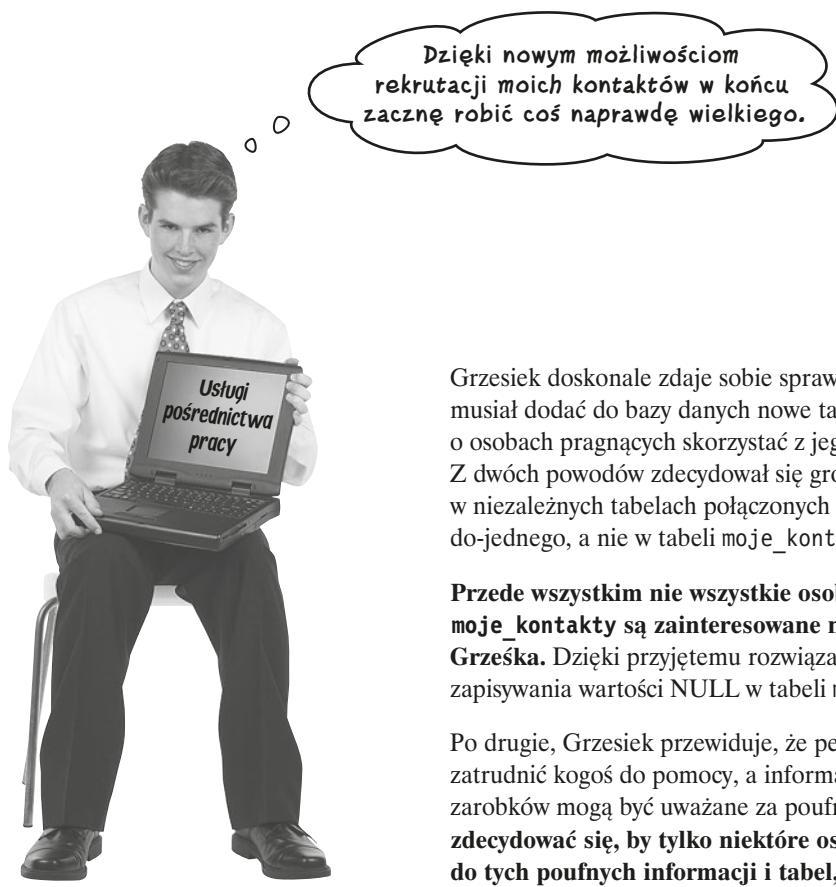
Czy ktokolwiek inny zauważy,
że jestem pełna... (ach...
jakie słowo by to właściwie
oddawało? Smaku? Blasku?
Urody?)

Janie, dwuczęściowe zapytanie, proszę. Złączenia są wspaniałe, jednak czasami musisz *zadać swojej bazie danych więcej niż jedno zapytanie*. Bądź *użyć wyników jednego zapytania jako danych wejściowych dla drugiego*. I właśnie w takich sytuacjach przydają się **podzapytania**. Pozwolą Ci **one uniknąć powielania danych**, sprawią, że Twoje **zapytania staną się bardziej dynamiczne**, a może nawet pozwolą się wkręcać na koncerty i przyjęcia dla wyższych sfer. (No, na to ostatnie bym nie liczyła, ale dwa punkty z trzech to i tak nieźle).

Grzesiek wchodzi na rynek pracy

Jak do tej pory baza danych `lista_grzesia` była używana „w sprawach sercowych”. Pomagała Grześkowi w odnajdywaniu partnerów życiowych dla jego przyjaciół i znajomych, ale nie dawała możliwości zarabiania pieniędzy.

Jednak Grzesiek wpadł na pomysł, że mógłby zająć się gromadzeniem ofert pracy i dopasowywaniem ich do osób zapisanych w tabeli kontaktów.



Grzesiek doskonale zdaje sobie sprawę z tego, że będzie musiał dodać do bazy danych nowe tabele z myślą o osobach pragnących skorzystać z jego nowej usługi. Z dwóch powodów zdecydował się gromadzić te informacje w niezależnych tabelach połączonych zależnościami jedno-jednego, a nie w tabeli `moje_kontakty`.

Przede wszystkim nie wszystkie osoby zapisane w tabeli `moje_kontakty` są zainteresowane nową usługą Grześka. Dzięki przyjętemu rozwiązaniu Grzesiek uniknie zapisywania wartości NULL w tabeli `moje_kontakty`.

Po drugie, Grzesiek przewiduje, że pewnego dnia mógłby zatrudnić kogoś do pomocy, a informacje o wysokości zarobków mogą być uważane za poufne. **Grzesiek może zdecydować się, by tylko niektóre osoby miały dostęp do tych poufnych informacji i tabel, w których informacje są zapisywane.**

Baza Grześka wzbogacona o nowe tabele

Grzesiek dodał do swojej bazy danych nowe tabele, pozwalające mu na gromadzenie informacji o **oczekiwanych stanowisku** oraz **oczekiwanych zarobkach**, jak również **obecnym stanowisku i zarobkach**. Dodał także prostą tabelę z listą dostępnych ofert pracy.

STARE TABELLE

zawody
id_zawodu
zawod

kod_poczтовy
kod_poczтовy
lokalizacja
województwo

stan
id_stanu
stan

Informacje na temat bieżącego zatrudnienia.

praca_aktualna
id_kontaktu
stanowisko
pensja
data_zatrudnienia

Nowe, poszukiwane zatrudnienie.

praca_poszukiwana
id_kontaktu
stanowisko
pensja_minimalna
pensja_maksymalna
dyspozycyjność
lat_doswiadczenia

Aktualnie dostępne oferty pracy.

oferty_pracy
id_oferty
stanowisko
pensja
kod_poczтовy
opis

NOWE TABELLE

moje_kontakty
id_kontaktu
nazwisko
imię
telefon
email
plec
data_urodzenia
id_zawodu
kod_poczтовy

kontakt_zainteresowanie
id_kontaktu
id_zainteresowania

zainteresowania
id_zainteresowania
zainteresowanie

kontakt_szuka
id_kontaktu
id_szuka

szuka
id_szuka
szuka

Ponieważ dwie nowe tabele są połączone z tabelą **moje_kontakty** zależnościami **jeden-do-jednego**, zatem Grzesiek mógł skutecznie i bez problemów korzystać z nich, posługując się **złączonymi naturalnymi**.

Grzesiek używa złączenia wewnętrznego

Grzesiek zdobył listę atrakcyjnych ofert pracy i teraz próbuje dopasować do niej osoby ze swojej bazy. Stara się dopasowywać jak najlepiej, gdyż dostanie specjalną premię, jeśli jego kandydat zostanie zatrudniony.

Poszukiwany: Twórca stron WWW

Zatrudnimy twórcę stron WWW z doskonałą znajomością języków HTML i CSS do pracy w naszym zespole interakcji i projektowania wizualnego. To wspaniała okazja dla kogoś, kto przykłada wielkie znaczenie do standardów internetowych i chce zabłysnąć w dużej i liczącej się firmie. Pracuj w niezwykle wpływowej firmie, kierowanej przez inteligentnych ludzi, którzy uwielbiają to, co robią.

Zarobki: 5000 – 6000 zł .

Doświadczenie: co najmniej 5 lat w branży.

Kiedy Grzesiek znajdzie kilka osób najlepiej pasujących do oferty, będzie mógł do nich zadzwonić i przeprowadzić dodatkowy wywiad. Na razie jednak chce znaleźć w bazie wszystkich twórców stron WWW z co najmniej pięcioletnim doświadczeniem, którzy nie wymagają pensji większej niż 6000 złotych.

Zaostrz ołówek



Napisz zapytanie, które pobierze z bazy osoby pasujące do oferty przedstawionej na poprzedniej stronie.

praca_aktualna
id_kontaktu
stanowisko
pensja
data_zatrudnienia

praca_poszukiwana
id_kontaktu
stanowisko
pensja_minimalna
pensja_maksymalna
dyspozycyjnosc
lat_doswiadczenia

oferty_pracy
id_oferty
stanowisko
pensja
kod_pocztowy
opis

To minimalna stawka, jaką dana osoba zaakceptuje.

A to stawka, jaką dana osoba ma nadzieję otrzymać.

Dwa zapytania w dwóch krokach

Ale Grzesiek chce użyć innych zapytań

Grzesiek dysponuje większą ilością ofert pracy, niż jest w stanie sprawdzić. Ma zamiar sprawdzać osoby ze swojej tabeli zawody i w niej próbować odnaleźć osoby pasujące do ofert pracy. Potem będzie mógł zastosować złączenie naturalne, by połączyć tabelę zawodów z tabelą moje_kontakty, pobrać informacje o osobach, a następnie sprawdzić ich zainteresowania.

W pierwszej kolejności Grzesiek tworzy listę wszystkich stanowisk z tabeli praca_aktualna.

```
SELECT stanowisko FROM praca_aktualna  
GROUP BY stanowisko ORDER BY stanowisko;
```

Stosujemy klauzulę GROUP BY, by każde stanowisko pojawiło się w wynikach tylko raz. Oprócz tego sortujemy wszystkie stanowiska alfabetycznie.

Wyniki

stanowisko
fryzjer
kelner
kucharz
projektant stron WWW
twórca stron WWW

To tylko kilka stanowisk zapisanych w tabeli praca_aktualna.

Zaostrz ołówek



Rozwiążanie

Napisz zapytanie, które pobierze z bazy osoby pasujące do oferty.

```
SELECT mk.nazwisko, mk.imie, mk.telefon  
FROM moje_kontakty AS mk  
NATURAL JOIN  
praca_poszukiwana AS pp  
WHERE pp.stanowisko = 'twórca stron WWW'  
AND pp.pensja_minimalna < 6000;
```

Musimy pobrać jedynie informacje kontaktowe, gdyż wiemy, że te osoby szukają pracy na stanowisku twórcy stron WWW.

Ponieważ zarówno w tabeli moje_kontakty, jak i praca_poszukiwana kluczem głównym jest kolumna id_kontaktu, zatem możemy je połączyć, wykorzystując proste złączenie naturalne.

Interesują nas tylko te osoby, które akceptują ten poziom zarobków. Sprawdzimy kolumnę pensja_minimalna, by przekonać się, czy oferowane zarobki są wyższe od kwoty, jaką dana osoba jest skłonna zaakceptować.

Teraz Grzesiek używa słowa kluczowego IN, by sprawdzić, czy w swoich kontaktach znajdzie osoby pasujące do posiadanych ofert pracy.

```
SELECT mk.imie, mk.nazwisko, mk.telefon, pa.stanowisko
```

```
FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk
```

```
WHERE
```

```
pa.stanowisko IN ('kucharz', 'fryzjer', 'kelner', 'projektant stron WWW', 'tworca stron WWW');
```

Czy pamiętasz słowo kluczowe IN? W tym przypadku będzie ono zwracać wiersze, w których pole pa.stanowisko zawiera jedną z wartości podanych w nawiasach.

Wyniki pierwszego zapytania.

To działa!

mk.imie	mk.nazwisko	mk.telefon	pa.stanowisko
Janek	Lachowicz	22 3490937	kucharz
Wanda	Malczewska	22 3495870	fryzjer
Stefan	Miller	22 0983453	projektant stron WWW
Jarek	Celiński	22 2325694	twórca stron WWW
Julek	Graczyński	22 9763467	twórca stron WWW

Jednak Grzesiek wciąż musi wpisywać i wykonywać dwa odrębne zapytania...



WYSIL SZARE KOMÓRKI

Spróbuj połączyć te dwa zapytania w jedno. Propozycję tego zapytania zapisz poniżej.

Podzapytania

Aby w jednym zapytaniu wykonać te same operacje, które Grzesiek wykonał, używając dwóch poleceń SQL, będziemy musieli skorzystać z **podzapytania** — czyli zapytania umieszczonego wewnątrz innego zapytania.

Drugie zapytanie, którego użyjemy do pobrania pasujących rekordów z tabeli zawody, nazwiemy zapytaniem **ZEWNĘTRZNYM**, gdyż będzie ono zawierało w sobie drugie — **WEWNĘTRZNE** — zapytanie. Zobaczmy, jak to wszystko działa:

zapytanie ZEWNĘTRZNE

```
SELECT mk.imie, mk.nazwisko, mk.telefon, pa.stanowisko  
FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk  
WHERE  
pa.stanowisko IN ('kucharz', 'fryzjer', 'kelner', 'projektant stron WWW', 'tworca stron WWW');
```

Ta część tworzy zapytanie zewnętrzne.

A tę część można usunąć i zastąpić częścią pierwszego zapytania, które w ten sposób utworzy zapytanie wewnętrzne.

Wszystkie nazwy stanowisk wyszczególnione w powyższym zapytaniu są wynikami naszego **pierwszego zapytania** — zapytania, które pobiera wszystkie stanowiska z tabeli `oferty_pracy`. A zatem (i to jest bardzo sprytne rozwiązanie, więc uważaj) **możemy zastąpić fragment zapytania zewnętrznego fragmentem naszego pierwszego zapytania**. Także w tym przypadku wygenerowana zostanie lista wszystkich stanowisk, jednak tym razem jest ona „hermetyzowana” w postaci podzapytania:

zapytanie WEWNĘTRZNE

```
SELECT stanowisko FROM oferty_pracy  
GROUP BY stanowisko ORDER BY stanowisko;
```

Ta część naszego pierwszego zapytania stanie się zapytaniem wewnętrzny.

Podzapytanie to zapytanie SQL, które zostało umieszczone wewnątrz innego zapytania. Czasami określa się je także jako zapytanie WEWNĘTRZNE.

Łączymy dwa zapytania w zapytanie z podzapytaniem

Zatem połączylśmy dwa zapytania w jedno. Pierwsze z nich jest nazywane **zapytaniem zewnętrznym**. Z kolei drugie zapytanie, umieszczone wewnątrz pierwszego, nazywamy **zapytaniem wewnętrznym**.

zapytanie ZEWNĘTRZNE

zapytanie WEWNĘTRZNE

= zapytanie z podzapytaniem

```
SELECT mk.imie, mk.nazwisko, mk.telefon, pa.stanowisko
FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk
WHERE pa.stanowisko IN (SELECT stanowisko FROM oferty_pracy);
```

Zapytanie zewnętrzne

Dwa połączone zapytania tworzą zapytanie z podzapytaniem.

Już nie musimy przepisywać nazw stanowisk z pierwszego zapytania, gdyż zrobi to za nas zapytanie wewnętrzne!

To te same wyniki co wcześniej, jednak uzyskaliśmy je, używając tylko jednego zapytania!

A oto wyniki, jakie uzyskamy po wykonaniu naszego nowego zapytania. Są dokładnie takie same jak wyniki zapytania, w którym w klauzuli WHERE umieściliśmy **wszystkie** nazwy stanowisk. A o ile mniej mieliśmy pisania...

mk.imie	mk.nazwisko	mk.telefon	pa.stanowisko
Janek	Lachowicz	22 3490937	kucharz
Wanda	Malczewska	22 3495870	fryzjer
Stefan	Miller	22 0983453	projektant stron WWW
Jarek	Celiński	22 2325694	twórca stron WWW
Julek	Graczyński	22 9763467	twórca stron WWW



Anatomia zapytania wewnętrznego

Jakby jedno pytanie nie wystarczało: poznajcie podzapytanie

Podzapytanie to nic innego jak zapytanie umieszczone wewnętrz w innego zapytania.

Zapytanie zewnętrzne jest także czasami nazywane **zapytaniem zawierającym**. Z kolei zapytanie umieszczone wewnętrz nazywamy **zapytaniem wewnętrznym lub krócej — podzapytaniem**.

```
SELECT jakas_kolumna, inną_kolumna  
FROM tabela  
WHERE kolumna = (SELECT kolumna FROM tabela);
```

Zapytanie zewnętrzne. Czasami
nazywane także zapytaniem
zawierającym.

```
SELECT jakas_kolumna, inną_kolumna  
FROM tabela  
WHERE kolumna = (SELECT kolumna FROM tabela);
```

Zapytanie wewnętrzne
lub podzapytanie.

```
SELECT jakas_kolumna, inną_kolumna  
FROM tabela  
WHERE kolumna = (SELECT kolumna FROM tabela);
```

Zapytanie zewnętrzne.

Zapytanie wewnętrzne.

Ponieważ zastosowaliśmy operator równości (=), zatem powyższe podzapytanie zwróci **pojedynczą wartość** — czyli **jeden wiersz z jednej kolumny** (czasami jest on także nazywany *komórką*, jednak w SQL-u jego prawidłowa nazwa to *wartość skalarna*) — która następnie zostanie porównana z kolumnami podanymi w klauzuli WHERE.

wartość

Nasze podzapytanie zwraca wartość
skalarną (czyli jedną kolumnę lub
jeden wiersz), która jest następnie
porównywana z kolumnami podanymi
w klauzuli WHERE.

Podzapytanie w działaniu

A teraz przyjrzyjmy się i zobaczymy, jak wygląda podobne zapytanie w akcji. Nasze zapytanie zewnętrzne będzie operować na tabeli `moje_kontakty`, a podzapytanie — na tabeli `kod_pocztowy`. System zarządzania bazą danych w pierwszej kolejności pobierze wartość skalarną z tabeli `kod_pocztowy`, a następnie porówna ją z kolumnami podanymi w klauzuli WHERE zapytania zewnętrznego.

```
(SELECT kod_pocztowy FROM
kod_pocztowy WHERE lokalizacja =
'Wroclaw' AND wojewodztwo = 'DS')
```

wartość

```
SELECT nazwisko, imie
FROM moje_kontakty
WHERE kod_pocztowy = (SELECT kod_pocztowy FROM
kod_pocztowy WHERE lokalizacja =
'Wroclaw' AND wojewodztwo = 'DS')
```



To zapytanie zwraca listę nazwisk i imion osób zamieszkujących we Wrocławiu w województwie dolnośląskim.

Nie istnieja
głupie pytania

P: A dlaczego nie mogę umieścić tego warunku w złączeniu?

G: Możesz. Jednak wiele osób uważa, że łatwiej jest napisać podzapytanie niż złączenie dające identyczne rezultaty. A poza tym fajnie jest mieć możliwość wyboru składni, jakiej możemy użyć.

Powyzsze zapytanie możesz także zapisać w następujący sposób:

```
SELECT nazwisko, imie FROM
moje_kontakty mk
NAUTRAL JOIN kod_pocztowy kp
WHERE kp.lokalizacja = 'Wroclaw'
AND kp.wojewodztwo = 'DS';
```

Pogawędka przy kominku

Pogawędkи при коминку



Temat dzisiejszej pogawędkи: **Jesteś WEWNĘTRZNE czy ZEWNĘTRZNE?**

Zapytanie Zewnętrzne

Wiesz co, Wewnętrzne... tak naprawdę to w ogóle cię nie potrzebuję. Doskonale poradzę sobie bez ciebie.

Wielkie mi halo! Zwracasz mi jeden mały wyniczek. A użytkownicy chcą danych, całej masy danych. I to właśnie ja je im zwracam. Cóż, założę się, że gdyby cię we mnie nie było, miałbym jeszcze większą frajdę ze swojej pracy.

Nie, jeśli bym użyło klauzuli WHERE.

Ależ jestem!. Powiedz mi, po co komuś wyniki składające się z jednej kolumny jednego wiersza? Czy takie informacje komukolwiek wystarczą?

Może być, ale ja zachowuję pełną niezależność.

Zapytanie Wewnętrzne

Także i ja doskonale poradzę sobie samo. Czy uważasz, że to taka świetna zabawa dostarczać ci precyjnie wybrane dane tylko i wyłącznie po to, abyś zamieniło je na grupę rekordów? Wiesz... jak to się mówi... ilość niekoniecznie przechodzi w jakość.

Nie, to ja nadaję twoim wynikom znaczenie. Gdyby nie ja, zapewne zwróciłobyś wszystkie możliwe dane z tabeli.

I tu cię mam — to właśnie JA JESTEM klauzulą WHERE. I to bardzo wyjątkową, mogę to powiedzieć bez fałszywej skromności. Prawdę mówiąc, w ogóle mi nie jesteś potrzebne.

Może zatem dogadamy się jakoś dla naszego wspólnego dobra? W końcu wyznaczam właściwy kierunek twoim wynikom.

Tak jak ja... w większości przypadków.



Podzapytania i ich reguły

Istnieją pewne reguły, które spełniają wszystkie podzapytania. Użyj poniższych słów i wyrażeń, by uzupełnić zdania przedstawione u dołu strony. (Być może będziesz musiał je odpowiednio odmienić lub użyć któregoś z nich więcej niż jeden raz).

SELECT

ŚREDNIK

LISTA KOLUMN

KONIEC

NAWIASY

INSERT

DELETE

UPDATE

FROM

HAVING

Reguły podzapytań

Podzapytanie zawsze jest pojedynczym poleceniem

Podzapytania zawsze są umieszczane wewnętrz

Podzapytania nie mają swojego własnego Jak zawsze jedyny jest umieszczany na całego zapytania.

Reguły podzapytań

Podzapytania mogą się pojawić w czterech różnych miejscach zapytań: w klauzuli na polecenia SELECT jako jedna z kolumn, w klauzuli oraz klauzuli

Podzapytania mogą być używane w poleceniach , oraz oczywiście w poleceniach



Podzapytania i ich reguły

Pamiętaj o tych regułach, analizując podzapytania przedstawiane w dalszej części tego rozdziału.

Reguły podzapytań

Podzapytanie zawsze jest pojedynczym poleceniem **SELECT**.

Podzapytania zawsze są umieszczane wewnętrz wewnątrz **NAWIASÓW**.

Podzapytania nie mają swojego własnego **ŚREDNIKA**. Jak zawsze jedyny **ŚREDNIK** jest umieszczany na **KOŃCU** całego zapytania.

Reguły podzapytań

Podzapytania mogą się pojawić w czterech różnych miejscach zapytań:
w klauzuli **SELECT**, na **LIŚCIE KOLUMN** polecenia **SELECT** jako jedna z kolumn, w klauzuli **FROM** oraz klauzuli **HAVING**.

Podzapytania mogą być używane w poleceniach **INSERT**, **DELETE**, **UPDATE** oraz oczywiście w poleceniach **SELECT**.

Nie istnieją grupy pytania

P: Co zatem może zwracać zapytanie wewnętrzne?
No i, co równie ważne, co może zwracać zapytanie zewnętrzne?

O: W większości przypadków zapytanie wewnętrzne może zwracać tylko jedną wartość — czyli jedną kolumnę z jednym wierszem. Zapytanie zewnętrzne może następnie użyć tej wartości i porównać ją ze wszystkimi innymi wartościami w kolumnie.

P: Dlaczego piszecie o „pojedynczej wartości”, skoro zapytanie przedstawione na stronie 418 zwraca całą kolumnę zawierającą wiele wartości?

O: Ponieważ operator IN operuje na zbiorze wartości. Gdybyś użył operatora równości (=), jak w ćwiczeniu „Anatomia zapytania wewnętrzne”, to mógłbyś porównywać tylko pojedynczą wartość.

P: Ciągle nie jestem pewny, czy podzapytanie może zwracać jedną wartość, czy grupę wartości. Co na ten temat mówią oficjalne reguły?

O: Ogólnie rzecz biorąc, podzapytanie musi zwracać pojedynczą wartość. Przypadek z zastosowaniem operatora IN jest wyjątkiem. W większości przypadków podzapytania muszą zwracać pojedynczą wartość, w przeciwnym razie nie będą działały prawidłowo.

P: A co się stanie, jeśli podzapytanie zwróci więcej niż jedną wartość, a jednocześnie nie będzie użyte w klauzuli WHERE zawierającej zbiór wartości?

O: W takim przypadku nastąpi chaos! Masowa destrukcja na wielką skalę. A tak naprawdę to zostanie zgłoszony błąd.



Pięknie... Te wszystkie reguły są super i w ogóle, ale tak naprawdę to chciałabym się dowiedzieć, w jaki sposób mogę się pozbyć tych długich nazw kolumn w wynikach, takich jak: mk.nazwisko. Macie jakąś regułę dotyczącą nazw kolumn?

Reguł może nie ma, ale są dwie rzeczy, które możesz zrobić, by zapanować nad nazwami kolumn.

Możesz utworzyć nazwy zastępcze na liście kolumn polecenia SELECT. Kiedy to zrobisz, nagle okaże się, że kolumny w tabeli wynikowej będą znacznie bardziej zrozumiałe.

Poniżej przedstawiliśmy utworzone wcześniej podzapytanie, w którym zastosowaliśmy krótkie **nazwy zastępcze kolumn**.

Kolumnie imie tabeli moje_kontakty nadamy nazwę zastępczą „imie” i to ona zostanie użyta w tabeli wynikowej.

Podobnie kolumnie nazwisko tabeli moje_kontakty nadamy nazwę zastępczą „nazwisko” i to także ona zostanie użyta w tabeli wynikowej.

SELECT mk.imie AS imie, mk.nazwisko AS nazwisko,

mk.telefon AS telefon, pa.stanowisko AS stanowisko

FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk

WHERE stanowisko IN (SELECT stanowisko FROM oferty_pracy);

Kolumna telefon tabeli moje_kontakty będzie identyfikowana w wynikach przy użyciu nazwy zastępczej „telefon”. I tak dalej... chyba już rozumiesz!

Oto wyniki zwrocone przez zapytanie.

Zwrócić uwagę, w jakim stopniu zastosowanie nazw zastępczych utatwiło zrozumienie zawartości tabeli.

Co więcej, ponieważ nazwy zastępcze są tymczasowe, ich zastosowanie nie ma żadnego wpływu na faktyczne nazwy kolumn w obu tabelach.

Pamiętaj, że słowo kluczowe AS jest opcjonalne. Oznacza to, że tworząc nazwy zastępcze, możesz je pominąć.

imie	nazwisko	telefon	stanowisko
Janek	Lachowicz	22 3490937	kucharz
Wanda	Malczewska	22 3495870	fryzjer
Stefan	Miller	22 0983453	projektant stron WWW
Jarek	Celiński	22 2325694	twórca stron WWW
Julek	Graczyński	22 9763467	twórca stron WWW

Podstawowe informacje o tworzeniu podzapytań

Podczas tworzenia i stosowania podzapytań największym problemem wcale nie jest ich struktura — jest nim określenie, jaki fragment zapytania zewnętrznego wymaga użycia podzapytania. Albo czy w ogóle podzapytanie jest konieczne.

Analiza zapytań w dużym stopniu przypomina analizę słów w pytaniach. Kiedy analizujemy pytania, odnajdujemy w nich słowa pasujące do rzeczy, które znamy (takich jak nazwy tabel i kolumn), i dzielimy na elementy.

Przeanalizujmy zatem pytanie, które chcemy zadać naszej bazie danych, i zobaczymy, jak na jego podstawie można utworzyć zapytanie. Zaczniemy od pytania:



Podział pytania.

Przekształćmy nasze pytanie, uwzględniając w nim tabele i kolumny tworzące naszą bazę danych.

„Kto z zarejestrowanych osób” oznacza, że interesuje nas imię i nazwisko z tabeli `moje_kontakty`. „Zarabia najwięcej” oznacza, że będzie nam potrzebna maksymalna wartość z tabeli `praca_aktualna`.

Kto z zarejestrowanych osób najwięcej zarabia?

imię i nazwisko z tabeli
`moje_kontakty`.

MAX(`pensja`) z tabeli
`praca_aktualna`.

Określenie zapytania zwracającego odpowiedź na część zapytania.

Ponieważ tworzymy niezależne zapytanie, zatem możemy wybrać część oryginalnego pytania i utworzyć zapytanie, które nam na nią odpowie.

Wyrażenie MAX(`pensja`) wydaje się być doskonałym kandydatem na podzapytanie.

SELECT MAX(`pensja`) FROM `praca_aktualna`;

Czy pamiętasz funkcję MAX?
Zwraca ona największą wartość
znalezioną w kolumnie podanej
w nawiasach.

Kontynuujemy podział naszego oryginalnego pytania

Pierwsza część zapytania także jest bardzo prosta, wystarczy pobrać imię i nazwisko z tabeli moje_kontakty:

```
SELECT mk.imie, mk.nazwisko  
FROM moje_kontakty AS mk;
```

Pobieramy imię i nazwisko.

W końcu musimy określić, jak połączyć oba zapytania.

Pobranie imion i nazwisk osób z tabeli moje_kontakty to nie wszystko, musimy jeszcze znać ich zarobki, by porównać je z naszą wartością maksymalną: MAX(pensja). Aby pobrać zarobki poszczególnych osób, będziemy musieli zastosować wewnętrzne złączenie naturalne:

```
SELECT mk.imie, mk.nazwisko, pa.pensja  
FROM moje_kontakty AS mk  
NATURAL JOIN praca_aktualna AS pa;
```

Zastosuj złączenie NATURAL JOIN, by pobrać zarobki poszczególnych osób.

W końcu dodaj klauzulę WHERE, by połączyć ze sobą oba zapytania.

Oto tworzymy jedno duże zapytanie zwracające odpowiedź na nasze pytanie: „Kto z zarejestrowanych osób zarabia najwięcej?”. Oto część zapytania, która właśnie stworzyliśmy — zwraca ona informację o wysokości pensji poszczególnych osób.

```
SELECT mk.imie, mk.nazwisko, pa.pensja  
FROM moje_kontakty AS mk NATURAL JOIN praca_aktualna AS pa  
WHERE pensja =  
(SELECT MAX(pa.pensja) FROM praca_aktualna pa);
```

A to jest pierwsza część zapytania, która właśnie postużyła nam jako podzapytanie pozwalające określić maksymalną wartość pensji. Ta wartość jest porównywana z wynikami zapytania zewnętrznego i pozwala określić ostateczne wyniki.

To Maciek? Powiniem się domyślać... Nigdy nie płacił za rachunki, które mu wystawiałem.

mk.imie	mk.nazwisko	pa.pensja
Maciek	Skalski	11500



Zapytanie można zapisać na więcej niż jeden sposób



Naprawdę wygląda na to, że
można by uzyskać te same wyniki
bez korzystania z podzapytania.

To fakt. Podzapytanie nie było jedynym sposobem uzyskania interesujących nas informacji.

Dokładnie te same informacje można by uzyskać, stosując naturalne złączenie wewnętrzne i klauzulę LIMIT. Jak to często bywa w języku SQL, także i to zapytanie można by napisać w inny sposób.

WYSIL SZARE KOMÓRKI



Nie obchodzi mnie, że to samo można zrobić na wiele sposobów. Chcę wiedzieć, który z nich jest najlepszy. Albo przynajmniej chcę poznać jakieś kryteria, które uzasadniają wybór jednego z nich.



Słusznie.

A może być tak zająrał do wywiadu „SQL bez tajemnic” zamieszczonego na stronie 430?

Podzapytanie jako kolumna polecenia SELECT

Podzapytanie może zostać użyte jako jedna z kolumn zwracanych przez polecenie SELECT. Weźmy na przykład następujące zapytanie:

```
SELECT mk.imie, mk.nazwisko,
       (SELECT wojewodztwo
        FROM kod_pocztowy
       WHERE mk.kod_pocztowy = kod_pocztowy) AS wojewodztwo
      FROM moje_kontakty AS mk;
```

Tu określamy nazwę zastępczą zwracanego wyniku — będzie nią „województwo”.



Analizę tego zapytania zaczniemy od przyjrzenia się podzapytaniu. Jego działanie jest bardzo proste — dopasowuje ono kod pocztowy z tabeli `moje_kontakty` z kodem pocztowym z tabeli `kod_pocztowy`.

Poniżej przedstawiliśmy bardzo prosty opis działania powyższego zapytania:

Przejrzyj wszystkie wiersze tabeli `moje_kontakty`. Z każdego z nich pobierz imię, nazwisko i województwo (przy czym znajdziesz je, zaglądając do tabeli `kod_pocztowy` i dopasowując zapisany w niej kod pocztowy z kodem pocztowym z analizowanego rekordu tabeli `moje_kontakty`).

Pamiętaj, że podzapytanie może zwracać tylko jedną wartość, a zatem każde jego wywołanie powoduje zwrócenie jednego wiersza danych. Poniżej pokazaliśmy, jak mogłyby wyglądać wyniki powyższego zapytania:

mk.imie	mk.nazwisko	wojewodztwo
Janek	Lachowicz	MZ
Wanda	Malczewska	SL
Stefan	Miller	DS
Jarek	Celiński	PK
Julek	Graczyński	WM

Jeśli podzapytanie zostało użyte jako kolumna wyrażenia w poleceniu SELECT, to może ono zwracać wyłącznie jedną wartość z jednej kolumny.

Podzapytanie ze złączeniem naturalnym

Inny przykład: Podzapytanie ze złączeniem naturalnym

Kolega Grześka, Andrzej, przechwałał się swoimi wysokimi zarobkami. Nie powiedział co prawda Grześkowi dokładnie, ile zarabia, jednak Grzesiek przypuszcza, że posiada tę informację w swojej bazie danych. A zatem sprawdził to, pisząc szybkie złączenie NATURAL JOIN i wykorzystując jako kryterium adres poczty elektronicznej Andrzeja.

```
SELECT pa.pensja  
FROM moje_kontakty mk NATURAL JOIN praca_aktualna pa  
WHERE email = 'andrzej@pizza-nzk.com.pl';
```

To zapytanie zwróci jedną wartość zawierającą wysokość pensji Andrzeja.

To będzie zapytanie wewnętrzne.

Grzesiek zauważył, że powyższe zapytanie zwróci tylko jedną wartość. Zatem zamiast wykonywać to zapytanie i ręcznie wpisywać zwróconą przez nie wartość do innego zapytania, Grzesiek zdecydował się zastosować je jako podzapytanie.

A zatem Grzesiek napisał zapytanie, które:

- pobiera wysokość pensji Andrzeja,
- porównuje ją z innymi pensjami,
- zwraca imię i nazwisko oraz zarobki osób,
- które zarabiają więcej niż Andrzej.

Tu zastosujemy operator porównania >.

Pensje większe niż Andrzeja.

To długie zapytanie, jednak pozwala mi porównać informację, której nie muszę znać, z innymi danymi w bazie.



A oto zastosowanie polecenia zewnętrznego:

```
SELECT mk.imie, mk.nazwisko, pa.pensja  
FROM  
moje_kontakty AS mk, NATURAL JOIN praca_aktualna AS pa  
WHERE  
pa.pensja > (TU WSTAWIMY PENSJĘ ANDRZEJA);
```

Podzapytania nieskorelowane

Obok przedstawiliśmy zapytanie powstałe poprzez połączenie opisanych wcześniej elementów. System zarządzania bazą danych w pierwszej kolejności jeden raz wykonuje zapytanie wewnętrzne, a następnie posługuje się zwróconą przez nie wartością do określenia wyników zapytania zewnętrznego.

System zarządzania bazą danych wykonuje tę część zapytania jako drugą.

Interesują nas wyłącznie osoby o zarobkach wyższych niż zarobki Andrzeja.

System zarządzania bazą danych przetwarza te dwa zapytania niezależnie od siebie.

```
Pobieramy imię, nazwisko i pensję.

SELECT mk.imie, mk.nazwisko, pa.pensja
FROM
moje_kontakty AS mk, NATURAL JOIN praca_aktualna AS pa
WHERE
pa.pensja > (SELECT pa.pensja
FROM moje_kontakty mk NATURAL JOIN praca_aktualna pa
WHERE email = 'andrzej@pizza-nzk.com.pl' );
```

A oto kilka wyników, jakie zwróciło nasze zapytanie.

Nie zastosowaliśmy klauzuli ORDER BY, dlatego nie zostały one w żaden sposób posortowane.

mk.imie	mk.nazwisko	pa.pensja
Gustaw	Leser	15400
Bogusław	Mysior	23000
Teresa	Szmal	18900
Radek	Wielgus	13200
Julia	Marczyńska	34000

Wszystkie podzapytania, jakie do tej pory poznałeś, są określane jako **podzapytania nieskorelowane** (ang. *noncorrelated subqueries*). Podzapytania są przetwarzane w pierwszej kolejności, a następnie ich wyniki zostają użyte w klauzuli WHERE zewnętrznego zapytania. *Niemniej jednak podzapytanie nie jest w żaden sposób uzależnione od wartości pochodzących z zapytania zewnętrznego — nic nie stoi na przeszkodzie, by zostało wykonane jako niezależne zapytanie.*

zapytanie ZEWNĘTRZNE

zapytanie WEWNĘTRZNE

Zapytanie zewnętrzne jest przetwarzane jako drugie, jego wyniki zależą od wartości zwróconej przez zapytanie wewnętrzne.

Zapytanie wewnętrzne jest niezależne i zostaje wykonane w pierwszej kolejności.

(a poza tym, jeśli uda Ci się zastosować słowo „nieskorelowany” w rozmowie z osobami, które nie mają wiele wspólnego z językiem SQL, to na pewno będą pod wielkim wrażeniem)

Jeśli podzapytanie jest niezależne i nie odwołuje się w żaden sposób do zapytania zewnętrznego, to nazywamy je podzapytaniem nieskorelowanym.



SQL bez tajemnic

**Temat dzisiejszego wywiadu:
Wybór optymalnego sposobu realizacji zapytania,
jeśli dostępna jest większa liczba możliwości.**

SQL. Rusz głową!: Witam SQL. Dziękuję za możliwość osobistego spotkania się z tobą. Wiem, że nie było to łatwe.

SQL: „Nielatwe”. Tak to nazywasz? Według mnie było to kłopotliwe, niepokojące i naprawdę trudne do oszacowania, a przy tym jednocześnie bardzo zagmatwane.

SQL. Rusz głową!: No, cóż... faktycznie. I po części właśnie o to chodzi. Podobno pojawiają się skargi, że jesteś zbyt elastyczny; że podczas zadawania zapytań kierowanych do ciebie dajesz zbyt wiele możliwości.

SQL: Nie przeczę, jestem elastyczny. Fakt, że można mi zadać to samo pytanie na kilka różnych sposobów, a na każde z nich zwróć taką samą odpowiedź.

SQL. Rusz głową!: Niektórzy mogliby rzec, że jesteś mało konkretny.

SQL: Nie mam zamiaru tego komentować. To nie moja wina ani nie mój problem.

SQL. Rusz głową!: Owszem, i o jednym, i o drugim wszyscy doskonale wiedzą. Chodzi o to, że jesteś trochę... nieprecyzyjny.

SQL: Co?! Ja nieprecyzyjny? Mam już tego naprawdę dosyć! (wstaje).

SQL. Rusz głową!: Nie, nie! Nie odchodź. Bardzo chcemy poznać kilka odpowiedzi. Czasami pozwalasz zapytać o to samo na tak wiele sposobów.

SQL: No i co w tym złego?

SQL. Rusz głową!: Tak naprawdę to nic. Chcielibyśmy się tylko dowiedzieć, O CO należy cię pytać. I czy to ma znaczenie, zważywszy, że odpowiedzi są identyczne.

SQL: Oczywiście, że ma znaczenie. Czasami możesz się o coś zapytać, a zwrócenie odpowiedzi na to pytanie zajmie mi bardzo dużo czasu. A czasami BANG! i już mam odpowiedź. A zatem to bardzo ważne, by zadać pytanie w odpowiedni sposób.

SQL. Rusz głową!: Chodzi więc o to, jak długo zabierze ci zwrócenie odpowiedzi na pytanie? To właśnie na tej podstawie należy wybierać sposób zadawania pytań?

SQL: No cóż. Dokładnie. Wszystko sprowadza się do tego, o co mnie zapytasz. W końcu po to jestem, żeby odpowiadać na pytania... o ile są precyzyjne.

SQL. Rusz głową!: Szybkość... czyli to jest cały sekret?

SQL: Słuchaj... dam ci radę. Bazy danych mają to do siebie, że ROSNA. Dlatego będziesz chciał, by znalezienie odpowiedzi na twoje pytania było możliwe jak najprostsze. Jeśli bowiem zapytasz mnie „Ktotozdziałał”, to ty będziesz musiał się postarać, bym myślał o tym pytaniu jak najmniej. Zadaj mi proste pytanie, a dostaniesz szybką odpowiedź.

SQL. Rusz głową!: Rozumiem, ale skąd mam wiedzieć, które pytania są proste?

SQL: Cóż, przede wszystkim musisz wiedzieć, że zapytania krzyżowe są jedną wielką stratą czasu. Także skorelowane podzapytania nie należą do najszybszych.

SQL. Rusz głową!: Coś jeszcze?

SQL: Cóż...

SQL. Rusz głową!: No, proszę... powiedz.

SQL: Musisz poeksperymentować. Czasami najlepszym rozwiązaniem będzie utworzenie tabel testowych i wypróbowanie różnych zapytań. W ten sposób będziesz mógł porównać, ile zajmie wykonanie każdego z nich. A... i jeszcze jedno: złączenia są bardziej wydajne niż podzapytania.

SQL. Rusz głową!: Dziękuję, SQL. Aż trudno mi uwierzyć, że to właśnie szybkość jest tym największym sekretem...

SQL: No właśnie. Dzięki, że zmarnowałeś mój czas.

WARSZTATY TWORZENIA ZAPYTAN

Przeczytaj każdy z poniższych scenariuszy. Zgodnie z wytycznymi napisz dwa zapytania, a następnie połącz je w jedno zapytanie z podzapytaniem.

- 1 o Grzesiek chciałby sprawdzić, jaką jest średnia pensja twórców stron WWW zarejestrowanych w jego tabeli praca_aktualna. Następnie chciałby sprawdzić, jak wyglądają faktyczne zarobki poszczególnych twórców stron w porównaniu z wyznaczoną średnią. Jeśli znajdzie osoby, które zarabiają mniej, to może się z nimi skontaktować, gdyż zapewne będą bardziej zainteresowane jego usługami niż inni.

Napisz zapytanie, które pobierze średnią wartość pensji twórców strony WWW zarejestrowanych w tabeli praca_aktualna.

.....
.....
.....

- 2 o Grzesiek chce pobrać imiona, nazwiska i pensje wszystkich twórców stron WWW zarejestrowanych w tabeli praca_aktualna.

Napisz zapytanie, które pobierze imiona, nazwiska i pensje wszystkich twórców strony WWW zarejestrowanych w tabeli praca_aktualna.

.....
.....
.....

- 3 o Grzesiek wykorzysta średnią wysokość pensji (i trochę operacji matematycznych) jako podzapytanie, które pozwoli mu wyświetlić wszystkich twórców stron WWW oraz kwotę, o jaką ich zarobki przewyższają lub o jaką są niższe od średniej pensji dla tego zawodu.

Połącz oba zapytania. Zastosuj podzapytanie jako jedną z kolumn polecenia SELECT.

.....
.....
.....
.....

WARSZTATY TWORZENIA ZAPYTANÍ. ROZWIAZANIE

Przeczytaj każdy z poniższych scenariuszy. Zgodnie z wytycznymi napisz dwa zapytania, a następnie połącz je w jedno zapytanie z podzapytaniem.

- 1 o Grzesiek chciałby sprawdzić, jaką jest średnia pensja twórców stron WWW zarejestrowanych w jego tabeli `praca_aktualna`. Następnie chciałby sprawdzić, jak wyglądają faktyczne zarobki poszczególnych twórców stron w porównaniu z wyznaczoną średnią. Jeśli znajdzie osoby, które zarabiają mniej, to może się z nimi skontaktować, gdyż zapewne będą bardziej zainteresowane jego usługami niż inni.

Napisz zapytanie, które pobierze średnią wartość pensji twórców strony WWW zarejestrowanych w tabeli `praca_aktualna`.

`SELECT AVG(pensja) FROM praca_aktualna WHERE stanowisko = 'twórcę stron WWW';`

Stwórz kluczowe AVG jest
właśnie tym, czego nam
potrzeba

- 2 o Grzesiek chce pobrać imiona, nazwiska i pensje wszystkich twórców stron WWW zarejestrowanych w tabeli `praca_aktualna`.

Napisz zapytanie, które pobierze imiona, nazwiska i pensje wszystkich twórców strony WWW zarejestrowanych w tabeli `praca_aktualna`.

`SELECT mk.imie, mk.nazwisko, pa.pensja
FROM moje_kontakty mk NATURAL JOIN praca_aktualna pa
WHERE pa.stanowisko = 'twórcę stron WWW';`

- 3 o Grzesiek wykorzysta średnią wysokość pensji (i trochę operacji matematycznych) jako podzapytanie, które pozwoli mu wyświetlić wszystkich twórców stron WWW oraz kwotę, o jaką ich zarobki przewyższają lub o jaką są niższe od średniej pensji dla tego zawodu.

Połącz oba zapytania. Zastosuj podzapytanie jako jedną z kolumn polecenia SELECT.

`SELECT mk.imie, mk.nazwisko, pa.pensja,
pa.pensja - (SELECT AVG(pensja) FROM praca_aktualna WHERE stanowisko = 'twórcę stron WWW')
FROM moje_kontakty mk NATURAL JOIN praca_aktualna pa
WHERE pa.stanowisko = 'twórcę stron WWW';`

A oto i nasze
podzapytanie.

Nieskorelowane podzapytania zwracające wiele wartości: IN oraz NOT IN

Przeanalizujmy pierwsze zapytanie, którego Grzesiek próbował użyć dawno temu, na stronie 417. Pomogło mu odnajdywać osoby, które były zatrudnione na stanowisku *odpowiadającym* listom dostępnych miejsc pracy. Zapytanie to operuje na pełnej liście stanowisk zwróconych przez polecenie SELECT umieszczone w podzapytaniu; lista ta jest porównywana kolejno ze wszystkimi rekordami tabeli *praca_aktualna*. W ten sposób odnajdywane są wszystkie możliwe dopasowania.

```
SELECT mk.imie, mk.nazwisko, mk.telefon, pa.stanowisko
FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk
WHERE pa.stanowisko IN (SELECT stanowisko FROM oferty_pracy);
```

Operator IN przetwarza wszystkie wiersze kolumny pa.stanowisko, porównując je z całym zbiorem wartości zwróconych przez podzapytanie.

Stosując operator **NOT IN**, Grzesiek mógłby znaleźć stanowiska, które *nie odpowiadają* żadnej z osób zarejestrowanych w jego bazie. W takim przypadku polecenie operowałyby na całej grupie oferowanych stanowisk pracy zwróconych przez podzapytanie, porównywały z nią wszystkie wiersze tabeli *praca_aktualna* i zwracały tylko te, które *nie pasują* do stanowisk zapisanych tabeli w *praca_aktualna*. Teraz Grzesiek może się skoncentrować na próbach znalezienia ofert pracy dla osób zajmujących takie stanowiska.

```
SELECT mk.imie, mk.nazwisko, mk.telefon, pa.stanowisko
FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk
WHERE pa.stanowisko NOT IN (SELECT stanowisko FROM oferty_pracy);
```

Operator NOT IN zwraca wszystkie stanowiska, dla których aktualnie nie ma żadnych ofert pracy.

Zapytania tego typu są nazywane **podzapytaniami nieskoreowanymi**; stosowane w nich operatory IN oraz NOT IN porównują wyniki podzapytania z zapytaniem zewnętrznym i odnajdują ewentualne dopasowania.



Nieskorelowane podzapytania wykorzystują operatory IN oraz NOT IN, by sprawdzić, czy wartości zwrócone przez podzapytanie należą do zbioru (lub nie należą do niego w przypadku operatora NOT IN).

Ćwiczenie z podzapytań



Ćwiczenie

Napisz zapytania ze złączeniami i, jeśli to będzie potrzebne, z nieskoreowanymi podzapytaniami, by znaleźć odpowiedzi na poniższe pytania. Może Ci się przydać przedstawiony na następnej stronie schemat bazy danych `lista_grzesia`.

W kilku przypadkach będziesz potrzebował funkcji agregujących, które poznasz przy okazji rozwiązywania problemów z nagrodami dla młodych gospodyń.

Wyświetl stanowiska, dla których pensje są równe najwyższej pensji oferowanej w tabeli `oferty_pracy`.

→ **Odpowiedź znajdziesz na stronie 436**

Wyświetl imiona i nazwiska osób, których zarobki przekraczają średnią wysokość pensji wszystkich zarejestrowanych osób.

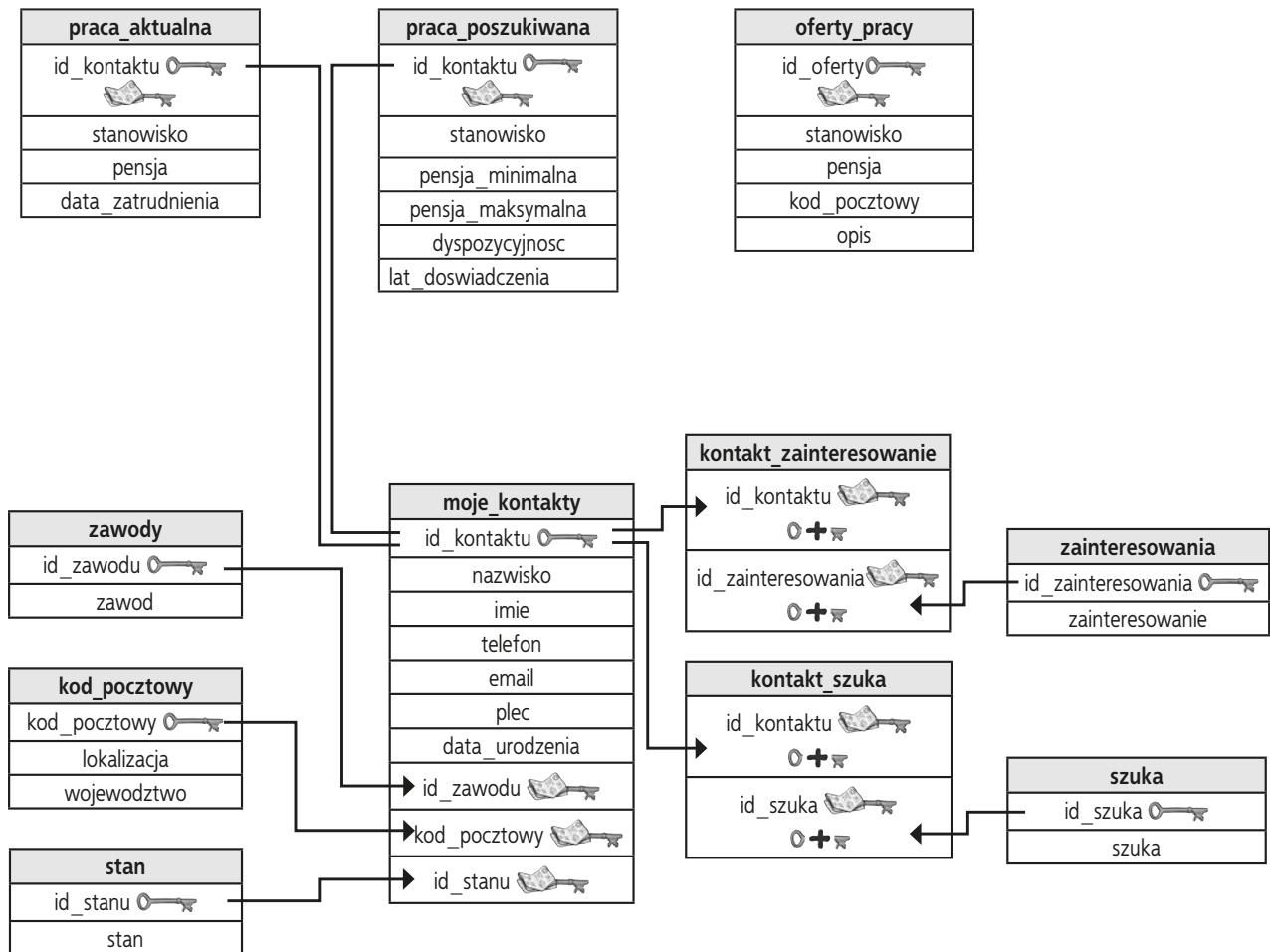
→ **Odpowiedź znajdziesz na stronie 436**

Znajdź wszystkich twórców stron WWW, którzy mają taki sam kod pocztowy, co dowolna zarejestrowana oferta pracy dla twórcy stron WWW.

→ **Odpowiedź znajdziesz na stronie 437**

Wyświetl wszystkie osoby, które mieszkają w tej samej miejscowości (o tym samym kodzie pocztowym), co osoba, która posiada aktualnie najwyższe zarobki.

→ **Odpowiedź znajdziesz na stronie 437**



Rozwiążanie ćwiczenia



Rozwiążanie ćwiczenia

Napisz zapytania ze złączonymi i, jeśli to będzie potrzebne, z nieskorelowanymi podzapytaniami, by znaleźć odpowiedzi na poniższe pytania. Może Ci się przydać przedstawiony na następnej stronie schemat bazy danych `lista_grzesia`.

Wyświetl stanowiska, dla których pensje są równe najwyższej pensji oferowanej w tabeli `oferty_pracy`.

Zapytanie zewnętrzne porównuje pensje z maksymalną pensją wyznaczoną przez funkcję MAX.

```
SELECT stanowisko FROM oferty_pracy  
WHERE pensja = (SELECT MAX(pensja)  
FROM oferty_pracy);
```

Podzapytanie zwraca jedną wartość

Funkcja MAX zwraca największą pensję w tabeli.

Wyświetl imiona i nazwiska osób, których zarobki przekraczają średnią wysokość pensji wszystkich zarejestrowanych osób.

Zewnętrzne zapytanie pobiera wyniki podzapytania i zwraca te rekordy, w których pensja jest większa od średniej.

```
SELECT mk.imie, mk.nazwisko  
FROM moje_kontakty mk  
NATURAL JOIN praca_aktualna pa  
WHERE pa.pensja > (SELECT AVG(pensja) FROM praca_aktualna);
```

Złączenie naturalne zwraca nam imiona i nazwiska osób, których pensje są większe od wartości wyznaczonej przez podzapytanie.

Podzapytanie zwraca średnią wysokość pensji.

Znajdź wszystkich twórców stron WWW, którzy mają taki sam kod pocztowy jak dowolna zarejestrowana oferta pracy dla twórcy stron WWW.

Jeśli chcemy uzyskać przydatne informacje dotyczące odnalezionych osób, takie jak imiona i nazwiska, to musimy zastosować złączenie naturalne.

```
SELECT mk.imie, mk.nazwisko, mk.telefon FROM moje_kontakty mk
NATURAL JOIN praca_aktualna pa WHERE pa.stanowisko = 'tworca stron WWW' AND mk.kod_pocztowy
IN (SELECT kod_pocztowy FROM oferty_pracy WHERE stanowisko = 'tworca stron WWW');
```

Ponieważ może się zdarzyć, że zostanie zwrócony więcej niż jeden kod pocztowy, zatem aby pobrać wyniki, musimy potraktować wyniki jako zbiór i zastosować operator IN.

Zapytanie wewnętrzne zwraca kody pocztowe wszystkich miejscowości, w których są dostępne oferty pracy dla twórców stron WWW.

Wyświetl wszystkie osoby, które mieszkają w tej samej miejscowości (o tym samym kodzie pocztowym), co osoba, która posiada aktualnie najwyższe zarobki.

To trudne pytanie, gdyż może się zdarzyć, że osób z najwyższymi zarobkami będzie kilka. A to oznacza, że konieczne jest zastosowanie operatora IN. Oprócz tego musimy zastosować aż dwa podzapytania.

Polecenie zewnętrzne pobiera kody pocztowe i odnajduje w tabeli moje_kontakty te rekordy, które do nich pasują. Ponieważ środkowe podzapytanie może zwrócić więcej niż jeden kod pocztowy, zatem konieczne jest zastosowanie operatora IN.

Środkowe podzapytanie zwraca kody pocztowe wszystkich osób, które mają najwyższą pensję.

```
SELECT imie, nazwisko FROM moje_kontakty
WHERE kod_pocztowy IN (SELECT mk.kod_pocztowy FROM moje_kontakty mk
NATURAL JOIN praca_aktualna pa
WHERE pa.pensja = (SELECT MAX(pensja) FROM praca_aktualna));
```

Najbardziej wewnętrzne podzapytanie wyznacza najwyższą wartość pensji, zarejestrowaną w tabeli praca_aktualna. Będzie to pojedyncza wartość, dzięki czemu możemy porównywać ją przy użyciu operatora równości (=).

Podzapytania skorelowane



Skoro podzapytanie nieskorelowane oznacza podzapytanie, które ma być niezależnym zapytaniem do bazy, to podzapytanie skorelowane zapewne będzie w jakiś sposób zależne od zapytania zewnętrznego.

Dokładnie. W przypadku podzapytania nieskorelowanego zapytanie wewnętrzne jest interpretowane przez system zarządzania bazą danych w pierwszej kolejności, a dopiero potem przetwarzane jest zapytanie zewnętrzne.

I w ten oto sposób dochodzimy do podzapytań skorelowanych. Podzapytanie skorelowane to podzapytanie, które może być przetworzone dopiero po powiązaniu go z zapytaniem zewnętrznym.

Przedstawiony poniżej przykład zapytania zlicza ilość zainteresowań (zapisanych w tabeli zainteresowania) dla każdej z osób zapisanych w tabeli moje_kontakty, po czym zwraca imiona i nazwiska tych osób, które mają dokładnie trzy zainteresowania.

```
SELECT mk.imie, mk.nazwisko  
FROM moje_kontakty AS mk  
WHERE  
3 = (  
    SELECT COUNT(*) FROM kontakt_zainteresowanie  
    WHERE id_kontaktu = mk.id_kontaktu  
);
```

Podzapytanie odwołuje się
do nazwy zastępczej mk.

Nazwa zastępcza tabeli
moje_kontakty jest tworzona
w zapytaniu zewnętrznym.

Zanim dowiemy się,
jaka ma być wartość
mk.id_kontaktu, konieczne
jest wykonanie zapytania
zewnętrznego.

Jak widać, w powyższym przykładzie podzapytanie zależy od zapytania zewnętrznego. Zanim będzie można wykonać podzapytanie, konieczne jest określenie wartości kolumny id_kontaktu, pochodzącej z zapytania zewnętrznego.

W podzapytaniu używana jest **nazwa zastępcza**, nazywana w tym przypadku także **nazwą korelacji**, utworzona w zapytaniu zewnętrznym.

(Przydatne) Podzapytanie skorelowane używające operatora NOT EXISTS

Bardzo często spotykanym zastosowaniem podzapytań skorelowanych jest tworzenie w zapytaniu zewnętrznym listy wierszy, które nie mają swoich odpowiedników w innej, powiązanej tabeli.

Załóżmy, że Grzesiek poszukuje nowych klientów dla swojej nowej działalności pośrednictwa pracy. W tym celu zamierza wysłać pocztą elektroniczną wiadomość do wszystkich osób z tabeli `moje_kontakty`, które jeszcze **nie są** zarejestrowane w tabeli `praca_aktualna`. Może to zrobić, używając operatora NOT EXISTS.

```
SELECT mk.imie, mk.nazwisko, mk.email
FROM moje_kontakty AS mk
WHERE NOT EXISTS
    (SELECT * FROM praca_aktualna pa
     WHERE mk.id_kontaktu = pa.id_kontaktu);
```

Dzięki zastosowaniu operatora NOT EXISTS będziemy mogli pobrać imię, nazwisko i adres e-mail wszystkich osób z tabeli `moje_kontakty`, które jeszcze nie są zapisane w tabeli `praca_aktualna`.

KTO CO ROBI?

Dopasuj podane z lewej strony informacje o przeznaczeniu z fragmentami zapytania umieszczonymi w lewej kolumnie.

mk.imie imie

Definiuje nazwę zastępczą kolumny `mk.nazwisko`.

WHERE NOT EXISTS

Warunek zostanie spełniony, jeśli dwa pola `id_kontaktu` będą sobie równe.

WHERE mk.id_kontaktu = pa.id_kontaktu

Definiuje nazwę zastępczą „imie” dla kolumny.

FROM moje_kontakty mk

Zapytanie zwróci wszystkie kolumny tabeli o nazwie zastępczej „pa”.

mk.nazwisko nazwisko

Definiuje nazwę zastępczą „email” dla kolumny.

SELECT * FROM praca_aktualna pa

Warunek będzie spełniony, jeśli czegoś nie uda się odnaleźć.

mk.email email

Określa nazwę zastępczą tabeli `moje_kontakty`.

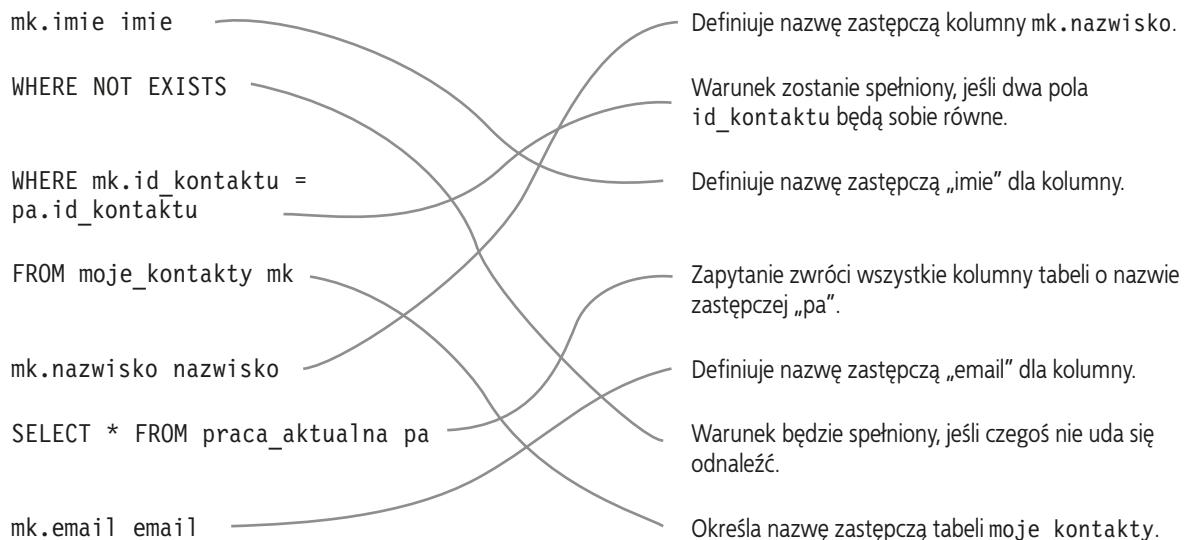
EXISTS i NOT EXISTS

Podobnie jak w przypadku operatorów IN oraz NOT IN, w podzapytaniach można stosować operatory **EXISTS** oraz **NOT EXISTS**. Poniższe zapytanie zwraca z tabeli `moje_kontakty` te wiersze, dla których wartość kolumny `id_kontaktu` pojawia się co najmniej raz w tabeli `kontakt_zainteresowanie`.

```
SELECT mk.imie, mk.nazwisko, mk.email  
FROM moje_kontakty AS mk  
WHERE EXISTS ← Dzięki zastosowaniu operatora EXISTS to zapytanie zwraca imię, nazwisko oraz adres poczty elektronicznej wszystkich osób zapisanych w tabeli moje_kontakty, których identyfikator (wartość pola id_kontaktu) przynajmniej raz pojawia się w tabeli kontakt_zainteresowanie.  
(SELECT * FROM kontakt_zainteresowanie kz WHERE mk.id_kontaktu = kz.id_kontaktu);
```

KTO CO ROBI? ROZWIĄZANIE

Dopasuj podane z lewej strony informacje o przeznaczeniu z fragmentami zapytania umieszczonymi w lewej kolumnie.



Zaostrz ołówek

Napisz zapytanie, które zwróci adres poczty elektronicznej wszystkich osób, które mają przynajmniej jedno zainteresowanie, lecz nie są jeszcze zarejestrowane w tabeli praca_aktualna.

→ Odpowiedzi znajdziesz na stronie 445

Usługi pośrednictwa pracy Grześka — zaproszenie do współpracy!

Grzesiek potrafi już sprawnie i szybko wyszukiwać informacje w swojej bazie danych, posługując się przy tym podzapytaniami. Odkrył nawet, w jaki sposób może ich używać w polecenях INSERT, UPDATE oraz DELETE.

Grzesiek wynajął niewielkie biuro na swoją nową firmę i zdecydował się urządzić dużą imprezę inauguracyjną.



Nie istniejąca grupa pytania

P: A zatem można umieszczać podzapytania wewnętrz innych podzapytań?

O: Oczywiście. Istnieje pewne ograniczenie co do liczby możliwych „zagnieżdżeń”, jednak większość systemów zarządzania relacyjnymi bazami danych obsługuje ich znacznie więcej, niż kiedykolwiek mógłbyś chcieć użyć.

P: Jaka jest najlepsza metoda tworzenia zagnieżdżonych podzapytań?

O: Zdecydowanie najlepiej będzie tworzyć niewielkie, proste pytania reprezentujące poszczególne elementy całego zapytania, które chcesz napisać. Następnie przyjrzyj się im i spróbuj określić, w jaki sposób musisz je połączyć. Jeśli próbujesz odszukać osoby, które zarabiają tyle samo co najlepiej opłacany twórca stron WWW, to rozdziel zapytanie na dwie części:

Odszukaj najlepiej opłacanego twórcę stron WWW.

Znajdź osoby, które zarabiają x złotych.

Następnie zastąp „x” zapytaniem zwracającym odpowiedź na pierwsze pytanie.

P: Założmy, że nie podoba mi się stosowanie podzapytań. Czy istnieje jakiś sposób zastąpienia ich złączciami?

O: W większości przypadków można zastąpić podzapytania złączciami. Jednak abyś mógł to robić, będziesz musiał poznać kilka nowych rodzajów złączeń. A to z kolei zmusza nas do...

W drodze na imprezę...

Jadąc na swoją imprezę, Grzesiek zauważył jakąś gazetę z niepokojącym nagłówkiem:

TYGODNIK BAZODETEKTYWISTYCZNY

My pierwsi UJAWNIAAMY całą SZOKUJĄCĄ PRAWDĘ o Podzapytaniach!



ZAKONSPIROWANE ZŁĄCZENIA.

Sąsiedzi szepczą, że podzapytania nie mogą „zrobić niczego więcej” niż zwyczajne złączenia, a „prawda musi w końcu ujrzeć światło dzienne”.

Autor: **Radek Wierszyński**

ETATOWY DETEKTYW BAZODANOWY

BAZODANOWO: To, o czym spekulowano od wielu lat, w końcu zostało sprawdzone i potwierdzone przez źródła „Tygodnika Bazodetektywistycznego”. Okazuje się, że złączenia i podzapytania mogą być używane do tworzenia identycznych zapytań. Ku zaskoczeniu okolicznych mieszkańców wszystko, co robili, używając podzapytań, można także wykonać, wykorzystując różnego rodzaju złączenia.

„To okropne — szlochała nauczycielka Hanka Jasińska. — Jak mam powiedzieć swoim uczniom, że wszystko, co jak im się wydawało, wiedzą o podzapytaniach, te wszystkie godziny spędżona na nauce sposobów ich tworzenia i stosowania... że to wszystko było niepotrzebne... że podzapytania można zastąpić zwyczajnymi złączeniami. To prawdziwy koszmar!”.

Skutki tych informacji mogą być w pełni odczuwalne jeszcze w następnym rozdziale, w którym nasi Czytelnicy będą mogli znaleźć unikalne informacje o złączeniach zewnętrznych.



Tutejsza mieszkanka, Hanka Jasińska, przeżyła szok, gdy poznała całą prawdę o podzapytaniach.

CZY TO WSZYSTKO BYŁO JEDYNIE STRATĄ CZASU? CZY PODZAPYTANIA TO FAKTYCZNIE TO SAMO CO ZŁĄCZENIA? ODPOWIEDZI NA TE I WIELE INNYCH PYTAŃ ZNAJDZIESZ W NASTĘPNYM ROZDZIALE.



Przybornik SQL

Właśnie zakończyłeś lekturę dziewiątego rozdziału książki i nauczyłeś się biegły władać podzapytaniami. Przyjrzyj się jeszcze raz wszystkiemu, o czym dowiedziałeś się w tym rozdziale. Kompletną listę porad znajdziesz w dodatku C, zamieszczonym na końcu książki.

Podzapytanie nieskorelowane

To niezależne podzapytanie, które w żaden sposób nie odwołuje się do zapytania zewnętrznego.

Podzapytanie skorelowane

Podzapytanie, które korzysta z wartości zwracanych przez zapytanie zewnętrzne i zależy od nich.

Zapytanie zewnętrzne

Zapytanie zawierające w sobie zapytanie wewnętrzne, określone także jako podzapytanie.

Zapytanie wewnętrzne

Zapytanie umieszczone wewnątrz innego zapytania. Często określa się je także jako podzapytanie.

Podzapytanie

Zapytanie umieszczone wewnątrz innego zapytania. Określa się je także jako zapytanie wewnętrzne.

Zaostrz ołówek



Rozwiążanie ze str. 441

Napisz zapytanie zwracające adres poczty elektronicznej wszystkich osób, które mają przynajmniej jedno zainteresowanie, lecz nie są jeszcze zarejestrowane w tabeli `praca_aktualna`.

```
SELECT mk.email FROM moje_kontakty mk WHERE
  EXISTS
    (SELECT * FROM kontakt_zainteresowanie kz WHERE mk.id_kontaktu = kz.id_kontaktu)
  AND ←
  NOT EXISTS
    (SELECT * FROM praca_aktualna pa
     WHERE mk.id_kontaktu = pa.id_kontaktu);
```

Także tutaj, podobnie jak we wszystkich innych przypadkach, gdy dwa warunki muszą być jednocześnie spełnione, możesz zastosować w klawizuli `WHERE` operator `AND`.

10. Złączenia zewnętrzne, złączenia zwrotne oraz unie

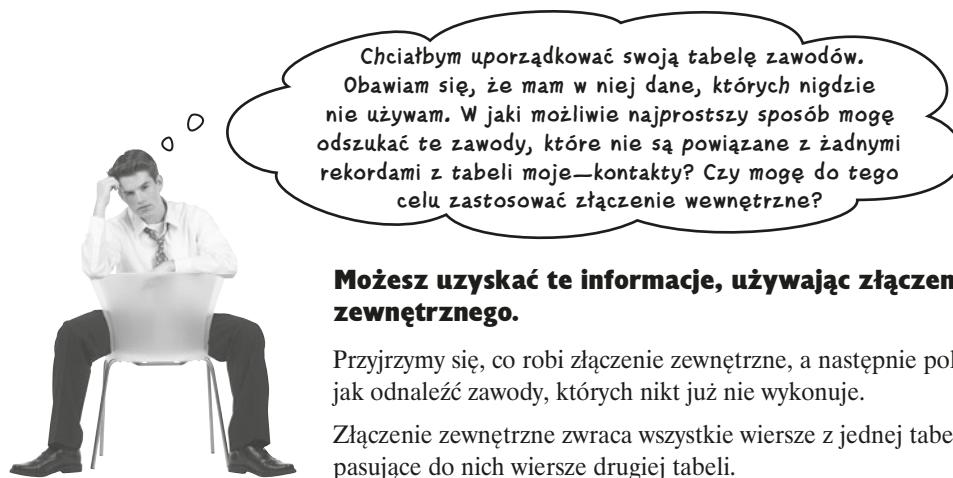
Nowe manewry



Po złączeniu wewnętrznym figura ósma – lewostronne złączenie zewnętrzne. Tym razem na pewno dostanę od sędziów bardzo wysokie oceny...

To wszystko, czego do tej pory dowiedziałeś się o złączeniach, to jedynie pół prawdy na ich temat. Widziałeś już złączenia krzyżowe tworzące wszystkie możliwe pary rekordów pochodzących z dwóch tabel oraz złączenia wewnętrzne zwracające jedynie pasujące do siebie rekordy obu tabel. Nie spotkałeś się jeszcze natomiast ze **złączeniami zewnętrznymi** (ang. *outer join*), zwracającymi także te wiersze, które *nie mają pasujących odpowiedników w drugiej tabeli*, **złączeniami zwrotnymi** (ang. *self-join*), które łączą tabelę z nią samą oraz **uniami** (ang. *union*), które scalają wyniki z kilku różnych zapytań. Kiedy poznasz te wszystkie sztuczki, będziesz w stanie pobierać dane z bazy dokładnie w taki sposób, jaki będzie Ci potrzebny. (Nie zapomnijmy także ujawnić całej szokującej prawdy o podzapytaniach!).

Porządki w starych danych



Możesz uzyskać te informacje, używając złączenia zewnętrznego.

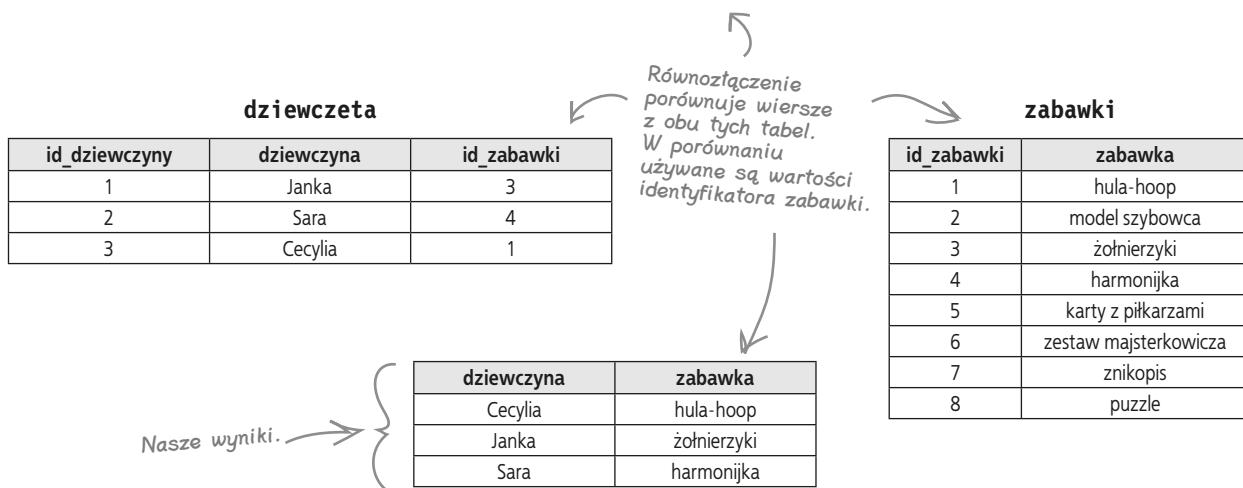
Przyjrzymy się, co robi złączenie zewnętrzne, a następnie pokażemy Ci, jak odnaleźć zawody, których nikt już nie wykonuje.

Złączenie zewnętrzne zwraca wszystkie wiersze z jednej tabeli oraz pasujące do nich wiersze drugiej tabeli.

Pamiętasz zapewne, że stosując złączenie wewnętrzne, *porównujemy wiersze z dwóch tabel*, lecz **kolejność, w jakiej zostaną użyte, nie ma żadnego znaczenia**.

Przypomnijmy sobie побieżnie, jak działa równozłączenie. Pobieramy z obu tabel wiersze, w których wartości kolumny `id_zabawki` są równe.

```
SELECT d.dziewczyna, z.zabawka
FROM dziewczeta d
INNER JOIN zabawki z
ON d.id_zabawki = z.id_zabawki;
```



Kluczem są dwie strony złączenia — lewa i prawa

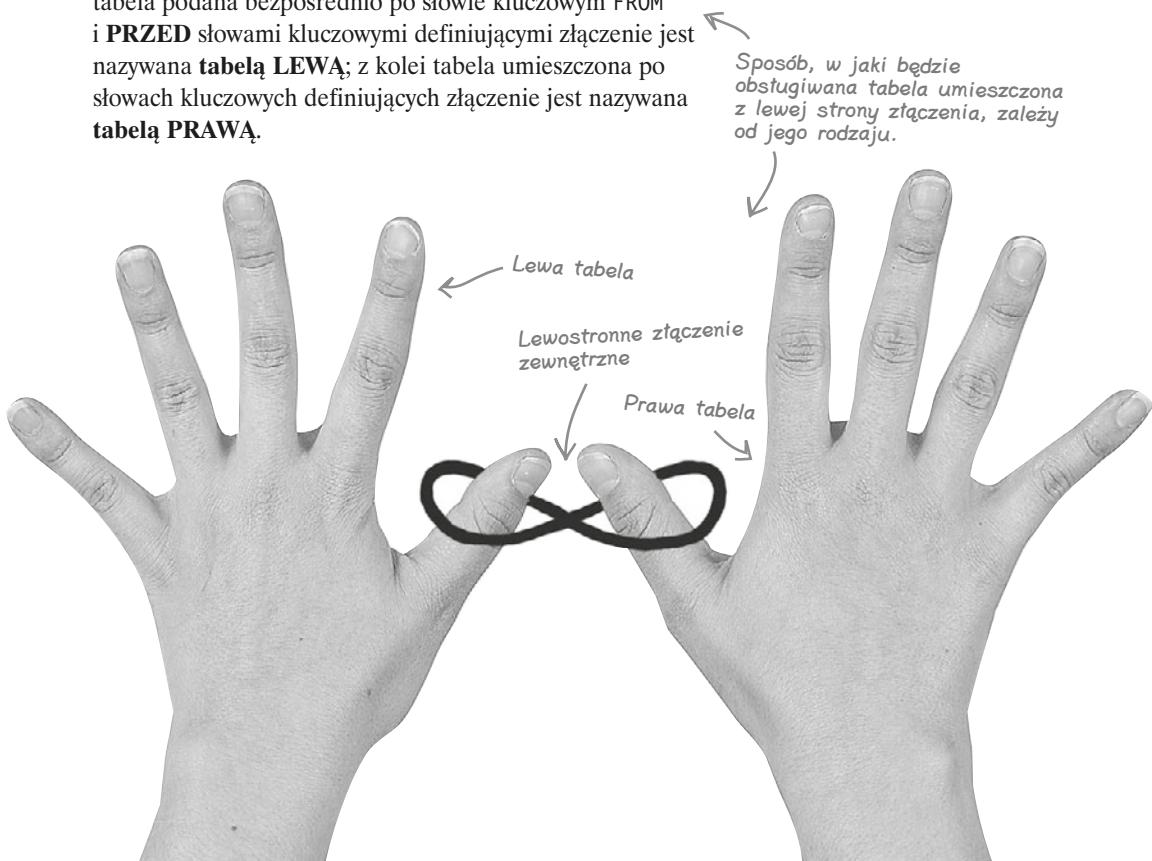
Porównując złączenia zewnętrzne z innymi rodzajami złączeń, które poznaleś we wcześniejszych rozdziałach, trzeba stwierdzić, iż złączenia zewnętrzne mają znacznie więcej wspólnego z *zależnościami pomiędzy dwiema tabelami*.

LEWOSTONNE ZŁĄCZENIE ZEWNĘTRZNE pobiera *wszystkie wiersze* z tabeli umieszczonej po jego lewej stronie i dopasowuje do nich wiersze z tabeli umieszczonej po stronie **PRAWEJ**. Jest ono bardzo przydatne w sytuacjach, gdy tabela umieszczona z lewej strony złączenia jest powiązana z tabelą umieszczoną po stronie prawej zależnością typu jeden-do-wielu.

Kluczem do zrozumienia działania złączeń zewnętrznych jest znajomość położenia tabel — która z nich znajduje się po lewej, a która po prawej stronie złączenia.

W przypadku **LEWOSTRONNEGO ZŁĄCZENIA ZEWNĘTRZNEGO** tabela podana bezpośrednio po słowie kluczowym **FROM** i **PRZED** słowami kluczowymi definiującymi złączenie jest nazywana **tabelą LEWA**; z kolei tabela umieszczona po słowach kluczowych definiujących złączenie jest nazywana **tabelą PRAWA**.

Lewostonne
złączenie
zewnętrzne
dopasowuje
WSZYSTKIE
WIERSY z LEWEJ
tabeli z wierszami
z tabeli prawej.



Lewostronne złączenie zewnętrzne

Oto lewostronne złączenie zewnętrzne

Możemy użyć lewostronnego złączenia zewnętrznego, by sprawdzić, jakie zabawki mają poszczególne dziewczęta.

Poniżej przedstawiona została składnia takiego złączenia, operującego na tabelach z poprzedniego przykładu. Bezpośrednio za słowem kluczowym FROM została umieszczona nazwa tabeli dziewczeta, a zatem to właśnie ona będzie lewą tabelą. Za nią umieszczone zostały słowa kluczowe LEFT OUTER JOIN oraz nazwa drugiej, prawej tabeli — zabawki.

A zatem LEWOSTRONNE ZŁĄCZENIE Zewnętrzne pobiera wszystkie wiersze z lewej tabeli (w tym przypadku jest to tabela dziewczeta) i dopasowuje je do wierszy z PRAWEJ tabeli (czyli tabeli zabawki).

```
SELECT d.dziewczyna, z.zabawka  
FROM dziewczeta d  
LEFT OUTER JOIN zabawki z  
ON d.id_zabawki = z.id_zabawki
```

Ta tabela jest umieszczona przed słowami kluczowymi definiującymi złączenie, a zatem dziewczeta będą lewą tabelą...

... a zabawki tabelą prawą, gdyż tabela została umieszczona za słowami kluczowymi definiującymi złączenie.

Ta tabela jest umieszczona przed słowami kluczowymi LEFT OUTER JOIN, zatem jest to tabela lewa...

... a ponieważ ta tabela jest umieszczona za słowami LEFT OUTER JOIN, zatem jest to tabela prawy.

dziewczeta

id_dziewczyny	dziewczyna	id_zabawki
1	Janka	3
2	Sara	4
3	Cecylia	1

id_zabawki	zabawka
1	hula-hoop
2	model szybowca
3	żołnierzyki
4	harmonijka
5	karty z piłkarzami
6	zestaw majsterkowicza
7	znikopis
8	puzzle

Wyniki naszego lewostronnego złączenia zewnętrznego

Jak widać, wyniki są takie same jak w przypadku zastosowania złączenia wewnętrznego.

Nasze wyniki.

dziewczyna	zabawka
Cecylia	hula-hoop
Janka	żołnierzyki
Sara	harmonijka



I to wszystko? W takim razie o co tyle zachodu? Wygląda na to, że złączenia zewnętrzne i wewnętrzne to dokładnie to samo.

Różnica pomiędzy nimi polega jednak na tym, że złączenie zewnętrzne zwraca wiersz niezależnie od tego, czy uda się dopasować wiersz w drugiej tabeli, czy nie.

A o tym, czy udało się dopasować wiersz, czy nie, poinformuje nas wystąpienie wartości NULL. W przypadku naszych przykładowych tabel dziewcząt i zabawek pojawienie się wartości NULL będzie świadczyć o tym, że dana zabawka nie należy do żadnej z dziewcząt. A to jest cenna informacja!

Wartość **NULL w wynikach lewostronnego złączenia zewnętrznego oznacza, że w prawej tabeli nie ma żadnych wartości, które odpowiadałyby wartościom z lewej tabeli.**

Zaostrz ołówek



Naszkicuj, jak według Ciebie będzie wyglądać wynikowa tabela zwrócona przez poniższe zapytanie.

```
SELECT d.dziewczyna, z.zabawka
FROM zabawki z
LEFT OUTER JOIN dziewczeta d
ON d.id_zabawki = z.id_zabawki;
```

Podpowiedź: Tabela wynikowa będzie zawierać osiem wierszy.

Zaostrz ołówek



Rozwiążanie

Naszkicuj, jak według Ciebie będzie wyglądać wynikowa tabela zwrócona przez poniższe zapytanie.

```
SELECT d.dziewczyna, z.zabawka
FROM zabawki z ← Lewa tabela.
LEFT OUTER JOIN dziewczeta d ←
ON d.id_zabawki = z.id_zabawki;
```

W tym przypadku każdy wiersz z tabeli zabawki (lewej tabeli) jest porównywany z zawartością tabeli dziewczeta (prawej tabeli).

Lewa tabela.

Prawa tabela.

zabawki

id_zabawki	zabawka
1	hula-hoop
2	model szybowca
3	żołnierzyki
4	harmonijka
5	karty z piłkarzami
6	zestaw majsterkowicza
7	znikopis
8	puzzle

dzieczeta

id_dziewczyny	dziewczyna	id_zabawki
1	Janka	3
2	Sara	4
3	Cecylia	1

Oto jakie wyniki uzyskamy, gdy zmienimy kolejność tabel w zapytaniu:

Jeśli uda się dopasować rekord, to pobrana z niego wartość zostanie umieszczona w tabeli wynikowej. Jeśli dopasowanie się nie powiedzie, to w tabeli wynikowej i tak pojawi się wiersz, jednak w takiej sytuacji zamiast dopasowanej wartości zobaczymy w nim wartość NULL.



dziewczyna	zabawka
Cecylia	hula-hoop
NULL	model szybowca
Janka	żołnierzyki
Sara	harmonijka
NULL	karty z piłkarzami
NULL	zestaw majsterkowicza
NULL	znikopis
NULL	puzzle

Kolejność kolumn w tabeli wynikowej odpowiada kolejności, w jakiej podaliśmy je w poleceniu SELECT. Kolejność ta nie ma żadnego związku z naszym LEWOSTRONNYM zapytaniem zewnętrznym.



Ćwiczenie

Zapytanie

Poniżej przedstawione zostały dwa zbiory wyników. Dla każdego z nich napisz lewostronne złączenie zewnętrzne, które mogłoby wygenerować takie wyniki, jak również tabele dziewczęta i zabawki zawierające dane, na których polecenie SQL operowało.

Wyniki lewostronnego złączenia zewnętrznego:

dziewczyna	zabawka
Jola	pistolet na kulki
Klara	magiczny balon
Martyna	NULL

Lewa tabela

Tę tabelę wypełniliśmy za Ciebie.
↓

dziewczeta

id_dziewczyny	dziewczyna	id_zabawki
1	Jola	1
2	Klara	2
3	Martyna	3

Prawa tabela

To jest trudne zadanie.
↓

Zapytanie

Wyniki lewostronnego złączenia zewnętrznego:

dziewczyna	zabawka
Jola	pistolet na kulki
Klara	pistolet na kulki
NULL	magiczny balon
Sara	puzzle
Marta	puzzle

Lewa tabela

Prawa tabela

Rozwiążanie ćwiczenia



Rozwiążanie ćwiczenia

Zapytanie

```
SELECT d.dziewczyna, z.zabawka  
FROM dziewczeta d  
LEFT OUTER JOIN zabawki z  
ON d.id_zabawki = z.id_zabawki;
```

Poniżej przedstawione zostały dwa zbiory wyników. Dla każdego z nich napisz lewostronne złączenie zewnętrzne, które mogłoby wygenerować takie wyniki, jak również tabele dziewczeta i zabawki zawierające dane, na których polecenie SQL operowało.

Wyniki lewostronnego złączenia zewnętrznego:

dziewczyna	zabawka
Jola	pistolet na kulki
Klara	magiczny balon
Martyna	NULL

To zabawki, które pojawiły się w naszych wynikach.

Lewa tabela

dziewczeta		
id_dziewczyny	dziewczyna	id_zabawki
1	Jola	1
2	Klara	2
3	Martyna	3

To może być identyfikator dowolnej zabawki, która w rzeczywistości nie istnieje w tabeli zabawek, gdyż w wynikowej tabeli, w kolumnie zabawki dla tej dziewczyny pojawiła się wartość NULL.

Zapytanie

```
SELECT d.dziewczyna, z.zabawka  
FROM zabawki z  
LEFT OUTER JOIN dziewczeta d  
ON d.id_zabawki = z.id_zabawki;
```

Prawa tabela

id_zabawki	zabawka
1	pistolet na kulki
2	magiczny balon

zabawki

Powtarzające się wartości widoczne w tabeli oznaczają, że więcej niż jedna dziewczyna ma tę samą zabawkę.

Wyniki lewostronnego złączenia zewnętrznego:

dziewczyna	zabawka
Jola	pistolet na kulki
Klara	pistolet na kulki
NULL	magiczny balon
Sara	puzzle
Marta	puzzle

A wartość NULL w kolumnie dziewczyna oznacza, że żadna dziewczyna nie ma tej zabawki.

Lewa tabela

zabawki	
id_zabawki	zabawka
1	pistolet na kulki
2	magiczny balon
3	puzzle

Prawa tabela

dziewczeta		
id_dziewczyny	dziewczyna	id_zabawki
1	Jola	1
2	Klara	1
3	Sara	3
4	Marta	3

Złączenia zewnętrzne i wielokrotne dopasowania

Jak zapewne zauważyleś na podstawie wyników przedstawionych w ostatnim ćwiczeniu, złączenie zewnętrzne będzie zwracało wiersze, nawet jeśli w drugiej tabeli nie ma pasujących rekordów. Co więcej, będzie ono zwracać wiele wierszy w sytuacji, gdy do jednego wiersza lewej tabeli można dopasować wiele wierszy z prawej tabeli. Poniżej pokazaliśmy, co tak naprawdę robi lewostronne złączenie zewnętrzne:

zabawki		dziewczeta		
id_zabawki	zabawka	id_dziewczyny	dziewczyna	id_zabawki
1	pistolet na kulki	1	Jola	1
2	magiczny balon	2	Klara	1
3	puzzle	3	Sara	3
		4	Marta	3

Wiersz z pistoletem z kulkami z tabeli zabawki jest porównywany z wierszem Joli z tabeli dziewczeta: zabawki.id_zabawki = 1, dziewczeta.id_zabawki = 1.

Znaleźliśmy dopasowanie.

Wiersz z pistoletem z kulkami z tabeli zabawki jest porównywany z wierszem Klary z tabeli dziewczeta: zabawki.id_zabawki = 1, dziewczeta.id_zabawki = 1.

Znaleźliśmy dopasowanie.

Wiersz z pistoletem z kulkami z tabeli zabawki jest porównywany z wierszem Sary z tabeli dziewczeta: zabawki.id_zabawki = 1, dziewczeta.id_zabawki = 3.

Nie ma dopasowania.

Wiersz z pistoletem z kulkami z tabeli zabawki jest porównywany z wierszem Marty z tabeli dziewczeta: zabawki.id_zabawki = 1, dziewczeta.id_zabawki = 3.

Nie ma dopasowania.

Wiersz z magicznym balonem z tabeli zabawki jest porównywany z wierszem Joli z tabeli dziewczeta: zabawki.id_zabawki = 2, dziewczeta.id_zabawki = 1.

Nie ma dopasowania.

Wiersz z magicznym balonem z tabeli zabawki jest porównywany z wierszem Klary z tabeli dziewczeta: zabawki.id_zabawki = 2, dziewczeta.id_zabawki = 1.

Nie ma dopasowania.

Wiersz z magicznym balonem z tabeli zabawki jest porównywany z wierszem Sary z tabeli dziewczeta: zabawki.id_zabawki = 2, dziewczeta.id_zabawki = 3.

Nie ma dopasowania.

Wiersz z magicznym balonem z tabeli zabawki jest porównywany z wierszem Marty z tabeli dziewczeta: zabawki.id_zabawki = 2, dziewczeta.id_zabawki = 3.

Nie ma dopasowania.

Koniec tabeli, tworzony jest nowy wiersz z wartością NULL w kolumnie dziewczyna.

Wiersz z puzzlami z tabeli zabawki jest porównywany z wierszem Joli z tabeli dziewczeta: zabawki.id_zabawki = 3, dziewczeta.id_zabawki = 1.

Nie ma dopasowania.

Wiersz z puzzlami z tabeli zabawki jest porównywany z wierszem Klary z tabeli dziewczeta: zabawki.id_zabawki = 3, dziewczeta.id_zabawki = 1.

Znaleźliśmy dopasowanie.

Wiersz z puzzlami z tabeli zabawki jest porównywany z wierszem Sary z tabeli dziewczeta: zabawki.id_zabawki = 3, dziewczeta.id_zabawki = 3.

Znaleźliśmy dopasowanie.

```
SELECT d.dziewczyna, z.zabawka
FROM zabawki z
LEFT OUTER JOIN dziewczeta d
ON d.id_zabawki = z.id_zabawki;
```

dziewczeta

dziewczyna	zabawka
Jola	pistolet na kulki
Klara	pistolet na kulki
NULL	magiczny balon
Sara	puzzle
Marta	puzzle

Prawostronne złączenie zewnętrzne

Prawostronne złączenie zewnętrzne jest dokładnie tym samym, co złączenie lewostronne, z tą różnicą, iż w tym przypadku to prawa tabela jest porównywana z lewą, a nie na odwrót.

Prawostronne złączenie
zewnętrzne przetwarza
prawą tabelę, porównując
ją z lewą tabelą.

```
SELECT d.dziewczyna, z.zabawka
FROM zabawki z ← Lewa tabela.
      Prawa tabela.
RIGHT OUTER JOIN dziewczeta d ←
ON d.id_zabawki = z.id_zabawki;
```

Lewa tabela
(w obu zapytaniach).

W obu tych zapytaniach tabela
dziewcząt jest tabelą lewą.

dziewczeta

id_dziewczyny	dziewczyna	id_zabawki
1	Janka	3
2	Sara	4
3	Cecylia	1

To zapytanie zobaczyłeś po raz pierwszy na stronie 450.

```
SELECT d.dziewczyna, z.zabawka
FROM dziewczeta d ← Lewa tabela.
      Prawa tabela.
LEFT OUTER JOIN zabawki z ←
ON d.id_zabawki = z.id_zabawki;
```

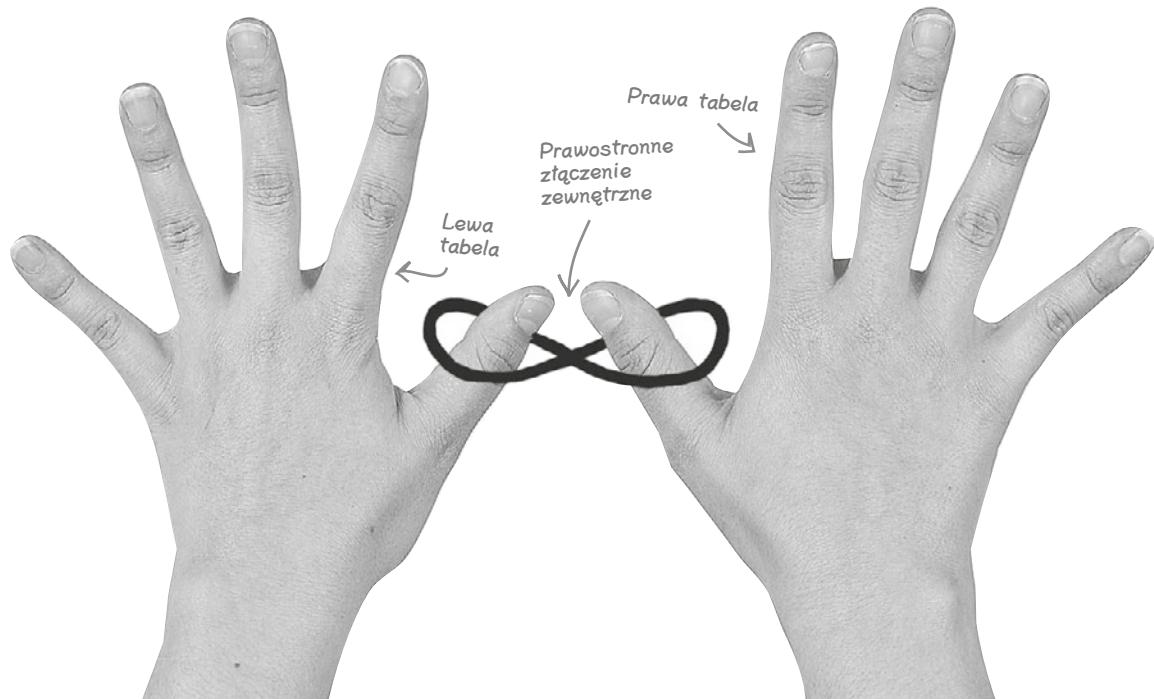
Prawa tabela
(w obu zapytaniach).

zabawki

id_zabawki	zabawka
1	hula-hoop
2	model szybowca
3	żołnierzyki
4	harmonijka
5	karty z piłkarzami
6	zestaw majsterkowicza
7	znikopis
8	puzzle

Nasze wyniki.

dziewczyna	zabawka
Cecylia	hula-hoop
Janka	żołnierzyki
Sara	harmonijka



Nieistniejąca grupa pytania

P: Czy są jakieś powody, dla których lepiej jest stosować lewostronne złączenie zewnętrzne, a nie złączenie prawostronne?

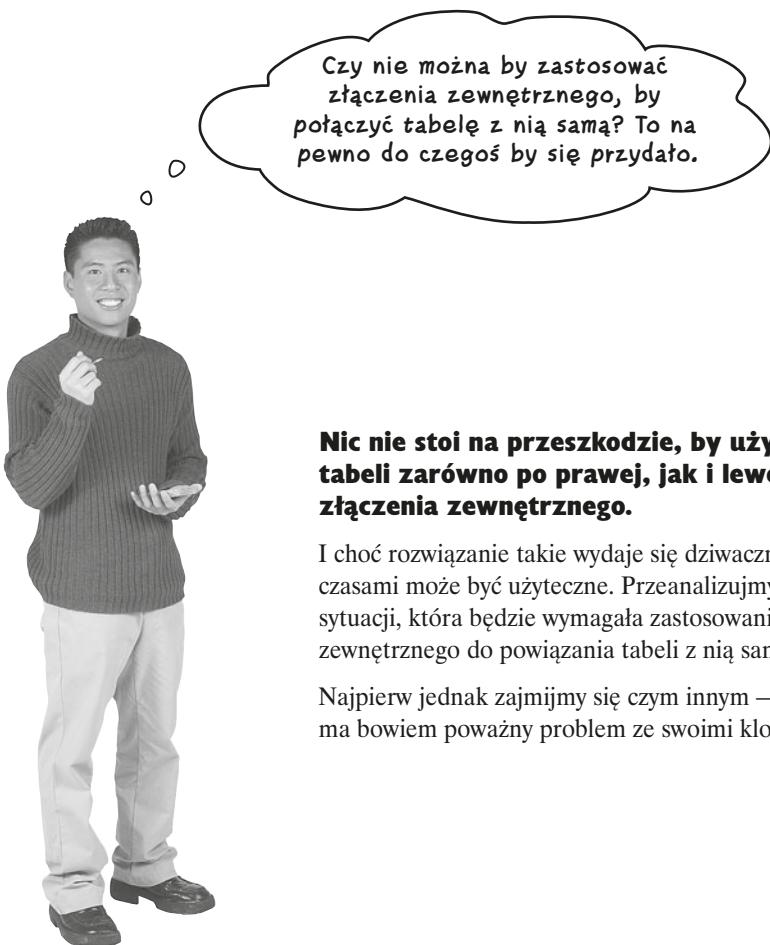
O: Zmiana słowa kluczowego LEFT na RIGHT jest znacznie łatwiejsza niż zamiana kolejności tabel użytych w poleceniu SQL. W końcu łatwiej jest zmienić jedno słowo niż dwie nazwy tabel wraz z ewentualnymi nazwami zastępczymi.

Jednak ogólnie rzecz biorąc, zazwyczaj łatwiej jest trzymać się jednego z tych złączeń, na przykład lewostronnego złączenia zewnętrznego, i odpowiednio podawać nazwy tabel z lewej lub prawej strony. Takie rozwiązywanie zapewni porządek i ułatwii zrozumienie poleceń SQL.

P: Skoro dostępne są LEWOSTRONNE i PRAWOSTRONNE złączenia zewnętrzne, to czy istnieje także jakiś kolejny rodzaj złączenia, który zwróciłby wszystkie wiersze z obu tabel?

O: Owszem, niektóre, lecz nie wszystkie systemy zarządzania relacyjnymi bazami danych udostępniają złączenie FULL OUTER JOIN — pełne złączenie zewnętrzne. Niemniej jednak złączenia tego typu nie są obsługiwane przez serwery MySQL, SQL Server ani Access.

Łączenie tabeli z nią samą



**Nic nie stoi na przeszkodzie, by użyć tej samej
tabeli zarówno po prawej, jak i lewej stronie
złączenia zewnętrznego.**

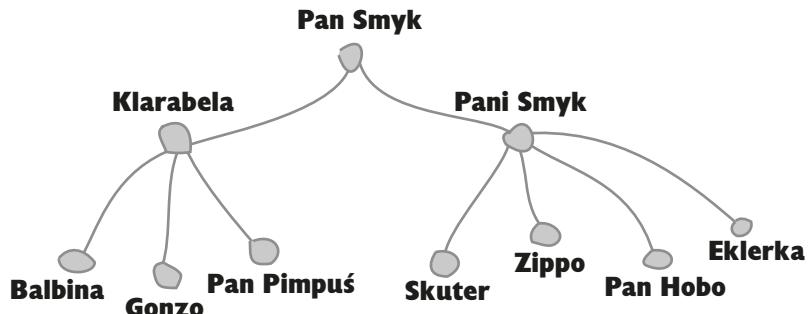
I choć rozwiązywanie takie wydaje się dziwaczne, to jednak czasami może być użyteczne. Przeanalizujmy przykład sytuacji, która będzie wymagała zastosowania złączenia zewnętrznego do powiązania tabeli z nią samą.

Najpierw jednak zajmijmy się czym innym — Bazodanowo ma bowiem poważny problem ze swoimi kłownami!

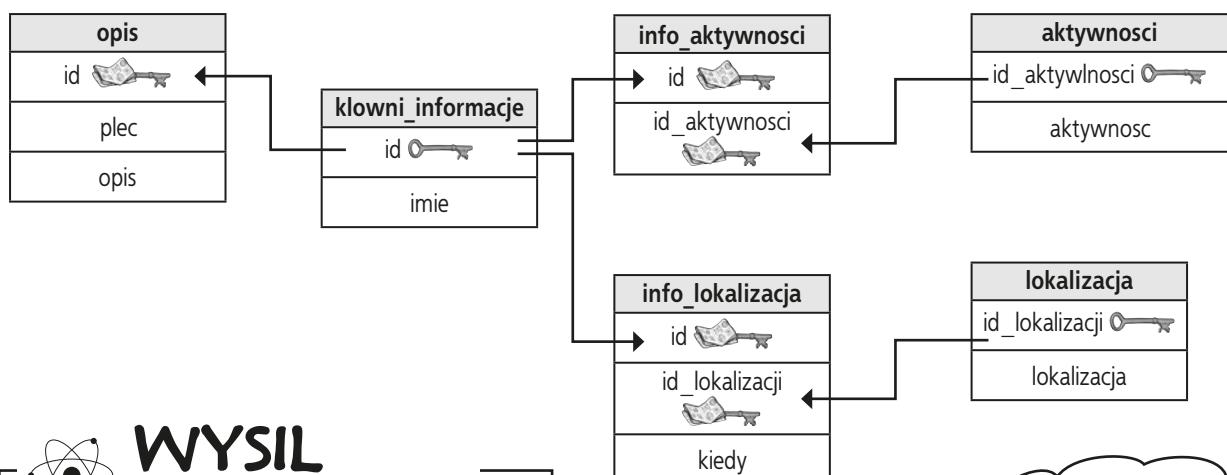
Podczas gdy my radośnie złączaliśmy zewnętrznie...

W Bazodanowie kloni się organizują i wybierają swoich szefów. To dosyć przerzążające, więc musimy trzymać rękę na pulsie i wiedzieć, kim są szefowie klonów oraz kto im podlega.

Obok przedstawiliśmy przykładową hierarchię klonów. Każdy z klonów ma jednego szefa, wyjątkiem jest tylko główny klon, na którego został wybrany Pan Smyk.



A teraz przyjrzyjmy się nowemu schematowi bazy danych klonów i zastanówmy się, jaki jest najlepszy sposób dopasowania do niej naszych nowych informacji:



WYSIL SZARE KOMÓRKI

W jaki sposób można zmienić strukturę bazy, by móc zapisywać w niej informacje o hierarchii klonów?

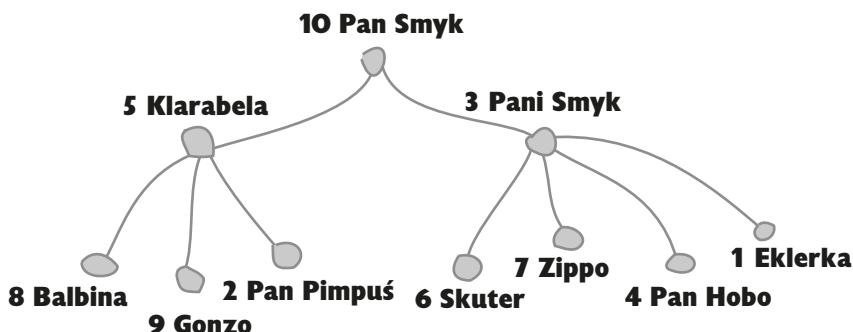
Pan Smyk — szef
Klarabeli i Pani Smyk



Ale czy jestem śmieszny? Chodzi mi o to, czy śmieszę cię jako klon?

Moglibyśmy utworzyć nową tabelę

Moglibyśmy utworzyć nową tabelę, która zawierałaby identyfikatory poszczególnych klaunów oraz ich szefów. Oto nasza hierarchia wraz z identyfikatorami poszczególnych klaunów.



A oto nowa tabela, zawierająca identyfikator każdego klauna oraz identyfikator jego szefa, odczytane z tabeli kloni_informacje.

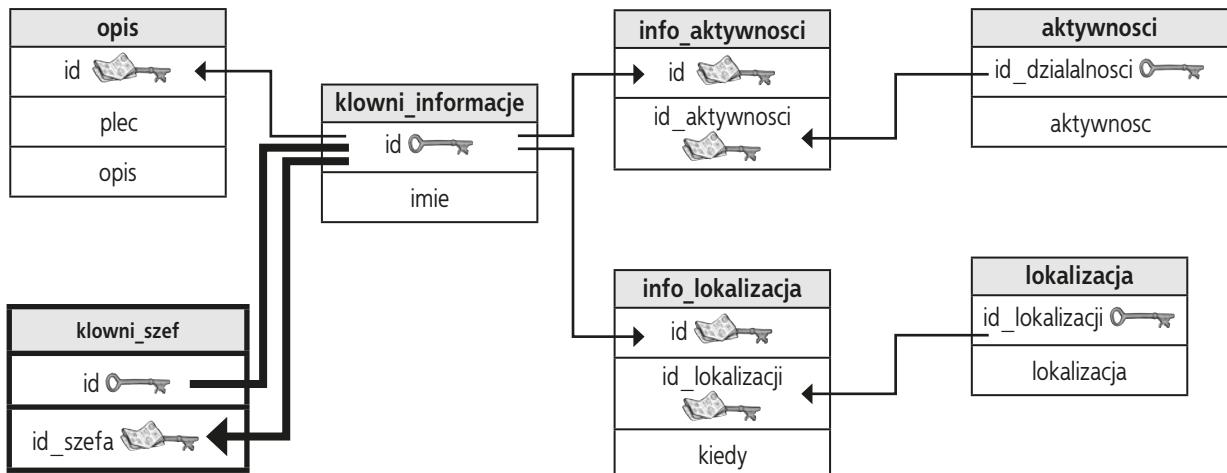
kloni_szef	
id	id_szefa
1	3
2	5
3	10
4	3
5	10
6	3
7	3
8	5
9	5
10	10

Pomiędzy tabelami kloni_informacje oraz kloni_szef występuje zależność jeden-do-jednego.

Pan Smyk nie ma nad sobą żadnego szefa, jednak w tym polu musi być zapisany jakiś identyfikator. Możemy zatem zapisać jego własny identyfikator, co pozwoli nam uniknąć pojawienia się wartości NULL.

Gdzie w schemacie umieścimy nową tabelę?

Przyjrzymy się aktualnemu schematowi bazy danych i zastanówmy, jaki będzie najlepszy sposób dopasowania do niego naszej nowej tabeli.



To nieco dziwne rozwiążanie. Na schemacie widać teraz zależność jeden-do-jednego wykorzystującą pole `id` — klucz główny naszej nowej tabeli — oraz zależność jeden-do-wielu wykorzystującą pole `id_szefa`. Mamy zatem **klucz główny** oraz **klucz obcy**, których wartości pochodzą z jednej tabeli — **kloni_informacje**.



Wygląda na to, że można by zastosować tabele, pomiędzy którymi będzie występować zależność jeden-do-jednego... Jednak nasze informacje nie zawierają żadnych prywatnych danych, ciekawe zatem, czy nie udałoby się w jakiś sposób umieścić tych informacji w jednej tabeli?



WYSIL
SZARE KOMÓRKI

Czy istnieje jakieś rozwiązanie, które pozwoliłoby nam zapisać informacje o hierarchii klaunów bez tworzenia w tym celu zupełnie nowej tabeli?

Klucz obcy odwołujący się do tej samej tabeli

Klucz obcy odwołujący się do tej samej tabeli

Tym, czego nam potrzeba, jest kolumna umieszczona w tabeli `kłowni_informacje`, która określałyby szefa danego kłowna. Kolumna ta będzie zawierać identyfikator kłowna będącego szefem. Nazwiemy ją `id_szefa`, czyli dokładnie tak samo, jak kolumnę w nowej tabeli `kłowni_szef`.

W przypadku tabeli `kłowni_szef` kolumna `id_szefa` była kluczem obcym. Kiedy dodamy ją do tabeli `kłowni_informacje`, to wciąż będzie ona pełnić rolę klucza obcego, pomimo iż jej wartości pochodzą z innej kolumny tej samej tabeli. Taki klucz nazywamy **kluczem odwołującym się do tej samej tabeli**. Nietrudno się domyślić, że jest to klucz, który odwołuje się do innej kolumny tej samej tabeli.

Zakładamy, że Pan Smyk jest sam sobie szefem, a zatem w jego rekordzie wartość w kolumnie `id_szefa` będzie taka sama jak w kolumnie `id`. Oznacza to, że takiego klucza możemy użyć jako naszej kolumny `id_szefa`.

Klucz obcy odwołujący się do tej samej tabeli jest *kluczem głównym* zastosowanym w *tej samej tabeli do innych celów*.

**Klucz obcy
ODWOŁUJĄCY SIĘ
DO TEJ SAMEJ TABELI
to klucz główny
zastosowany
w tej samej tabeli
w innym celu.**

To jest nowa kolumna `id_szefa`,
której dodaliśmy bezpośrednio
do tabeli `kłowni_informacje`.
Zapisaliśmy w niej klucz obcy
odwołujący się do tej samej tabeli.

`kłowni_informacje`

<code>id</code>	<code>imie</code>	<code>id_szefa</code>
1	Eklerka	3
2	Pan Pimpus	5
3	Pani Smyk	10
4	Pan Hobo	3
5	Klarabela	10
6	Skuter	3
7	Zippo	3
8	Balbina	5
9	Gonzo	5
10	Pan Smyk	10

Ta wartość odwołuje się do kolumny `id` tej samej tabeli, informując nas w ten sposób, kto jest szefem Eklerki.

I powtarzymy to raz jeszcze — w przypadku Pana Smyka w kolumnie `id_szefa` jest zapisany jego własny identyfikator.

Łączenie tabeli z nią samą

Założmy, że chcielibyśmy wyświetlić imiona wszystkich zarejestrowanych kloonów wraz z imionami ich szefów. Bez najmniejszego problemu możemy wyświetlić imię każdego kлона oraz identyfikator jego szefa; wystarczy w tym celu użyć bardzo prostego zapytania:

```
SELECT imie, id_szefa FROM kloni_informacje;
```

Jednak nam nie o to chodzi — nas interesuje lista składająca się z imienia kлона oraz imienia jego szefa:

imie	szef
Eklerka	Pani Smyk
Pan Pimpus	Klarabela
Pani Smyk	Pan Smyk
Pan Hobo	Pani Smyk
Klarabela	Pan Smyk
Skuter	Pani Smyk
Zippo	Pani Smyk
Balbina	Klarabela
Gonzo	Klarabela
Pan Smyk	Pan Smyk

Zaostrz ołówek



Założmy, że mamy dwie identyczne tabele: `kloni_informacje1` oraz `kloni_informacje2`. Napisz zapytanie z pojedynczym złączeniem, które zwróci tabelę zawierającą imię każdego kлона oraz imię jego szefa.

kloni_informacje1

id	imie	id_szefa
1	Eklerka	3
2	Pan Pimpus	5
3	Pani Smyk	10
4	Pan Hobo	3
5	Klarabela	10
6	Skuter	3
7	Zippo	3
8	Balbina	5
9	Gonzo	5
10	Pan Smyk	10

kloni_informacje2

id	imie	id_szefa
1	Eklerka	3
2	Pan Pimpus	5
3	Pani Smyk	10
4	Pan Hobo	3
5	Klarabela	10
6	Skuter	3
7	Zippo	3
8	Balbina	5
9	Gonzo	5
10	Pan Smyk	10

Rozwiążanie kolejnego ćwiczenia

Zaostrz ołówek



Rozwiążanie

Założymy, że mamy dwie identyczne tabele: klowni_informacje1 oraz klowni_informacje2.

Napisz zapytanie z pojedynczym złączeniem, które zwróci tabelę zawierającą imię każdego klauna oraz imię jego szefa.

klowni_informacje1

id	imie	id_szefa
1	Eklerka	3
2	Pan Pimpuś	5
3	Pani Smyk	10
4	Pan Hobo	3
5	Klarabela	10
6	Skuter	3
7	Zippo	3
8	Balbina	5
9	Gonzo	5
10	Pan Smyk	10

klowni_informacje2

id	imie	id_szefa
1	Eklerka	3
2	Pan Pimpuś	5
3	Pani Smyk	10
4	Pan Hobo	3
5	Klarabela	10
6	Skuter	3
7	Zippo	3
8	Balbina	5
9	Gonzo	5
10	Pan Smyk	10

Aby dwie kolumny o identycznej nazwie „imie” nie wprowadziły nas w błąd, kolumnie zawierające imię szefa nadamy nazwę zastępczą „szef”.

```
SELECT k1.imie, k2.imie AS szef  
FROM klowni_informacje1 AS k1  
INNER JOIN klowni_informacje2 AS k2  
ON k1.id_szefa = k2.id;
```

To właśnie w tym miejscu dopasowujemy wartość `id_szefa` z tabeli `klowni_informacje1` z wartością kolumny `id` z tabeli `klowni_informacje2`.

Potrzebujemy złączenia zwrotnego

W ćwiczeniu z serii „Zaostrz ołówek”, które rozwiązałeś na poprzedniej stronie, dysponowaliśmy dwiema kopiami tej samej tabeli. Jednak w znormalizowanej bazie danych takie dwie kopie tej samej tabeli nigdy się nie pojawią. Zamiast tego możemy **zastosować złączenie zwrotne, by zasymulować dwie tabele**.

Przyjrzyj się poniższemu zapytaniu, które jest niemal takie samo jak zapytanie z ostatniego ćwiczenia; różni się jednak od niego pod jednym, oczywistym względem:

```
SELECT k1.imie, k2.imie AS szef
FROM kloni_informacje k1
INNER JOIN kloni_informacje k2
ON k1.id_szefa = k2.id;
```



Jak widać, w tym zapytaniu dwukrotnie używamy tabeli `kloni_informacje`. Postępujemy się przy tym nazwami zastępczymi `k1` (której użyjemy do pobrania wartości z kolumny `id_szefa`) oraz `k2` (zastosowanej do pobrania imienia szefa).

Zamiast dwóch identycznych tabel dwukrotnie używamy tej samej tabeli `kloni_informacje`, posługując się przy tym nazwami zastępczymi `k1` oraz `k2`. Następnie tworzymy złączenie wewnętrzne, by połączyć wartość z kolumny `id_szefa` (tabeli `k1`) z wartością kolumny `imie` (z tabeli `k2`).



Ta kolumna pochodzi ze złączenia wewnętrznego pomiędzy kolumną `id_szefa` pierwszej kopii tabeli `kloni_informacje` (`k1`) z kolumną `imie` drugiej kopii tej samej tabeli (`k2`).

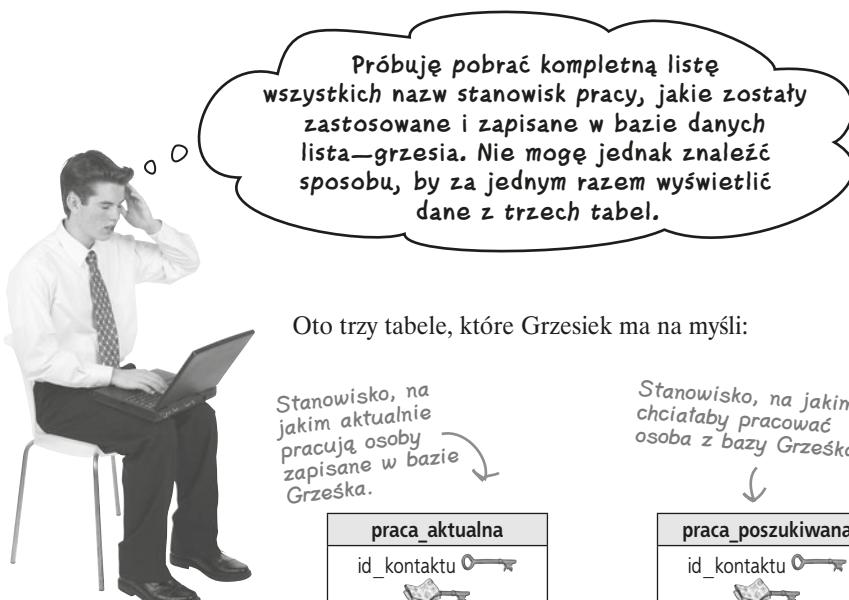
imie	szef
Eklerka	Pani Smyk
Pan Pimpuś	Klarabela
Pani Smyk	Pan Smyk
Pan Hobo	Pani Smyk
Klarabela	Pan Smyk
Skuter	Pani Smyk
Zippo	Pani Smyk
Balbina	Klarabela
Gonzo	Klarabela
Pan Smyk	Pan Smyk

Złączenie zwrotne pozwala na tworzenie zapytań do jednej tabeli w taki sposób, jak gdyby były to dwie tabele zawierające identyczne informacje.

kloni_informacje

id	imie	id_szefa
1	Eklerka	3
2	Pan Pimpuś	5
3	Pani Smyk	10
4	Pan Hobo	3
5	Klarabela	10
6	Skuter	3
7	Zippo	3
8	Balbina	5
9	Gonzo	5
10	Pan Smyk	10

Inny sposób zwieracania informacji z wielu tabel



Do tej pory Grzesiek korzystał z trzech niezależnych poleceń SELECT:

```
SELECT stanowisko FROM praca_aktualna;  
SELECT stanowisko FROM praca_poszukiwana;  
SELECT stanowisko FROM oferty_pracy;
```

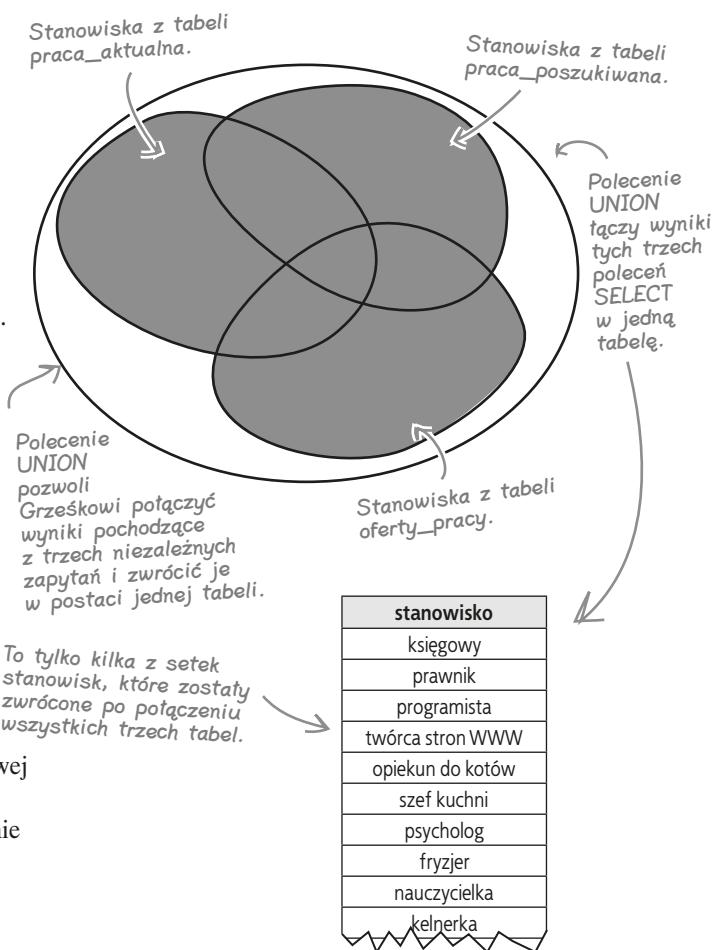
To rozwiązanie działało. Jednak Grzesiek chciałby połączyć wyniki tych zapytań w jedną całość generowaną przez jedno zapytanie i uzyskać w ten sposób listę wszystkich stanowisk pracy, jakie pojawiają się w jego bazie.

Można zastosować polecenie UNION

Istnieje jeszcze inny sposób na pobranie i połączenie danych pochodzących z dwóch lub większej liczby tabel. Polega on na zastosowaniu polecenia SQL UNION.

Polecenie to łączy wyniki dwóch lub większej ilości zapytań i zapisuje je w jednej tabeli, utworzonej na podstawie nazw kolumn podanych w polecenях SELECT. Można sobie wyobrazić, że wyniki tego polecenia zawierają wartości pochodzące z trzech poleceń SELECT, których wyniki „nakładają się na siebie”.

```
SELECT stanowisko FROM praca_aktualna
UNION
SELECT stanowisko FROM praca_poszukiwana
UNION
SELECT stanowisko FROM oferty_pracy;
```



Grzesiek zauważył, że nazwy stanowisk w tabeli wynikowej nie powtarzają się, lecz jednocześnie nie są wyświetlane w kolejności alfabetycznej. Dlatego też próbuje ponownie zadać to samo pytanie, dodając do każdego z poleceń SELECT klauzulę ORDER BY.

```
SELECT stanowisko FROM praca_aktualna ORDER BY stanowisko
UNION
SELECT stanowisko FROM praca_poszukiwana ORDER BY stanowisko
UNION
SELECT stanowisko FROM oferty_pracy ORDER BY stanowisko;
```

Grzesiek dodał do każdego zapytania klauzulę ORDER BY, by stanowiska na liście wynikowej były posortowane alfabetycznie.



WYSIL SZARE KOMÓRKI

Jak sądzisz, co się stanie, kiedy Grzesiek wykona swoje nowe zapytanie?

Polecenie UNION ma swoje ograniczenia

Nowe zapytanie Grześka nie zwróciło oczekiwanych wyników, co gorsza — próba jego wykonania zakończyła się wystąpieniem błędu! Błąd pojawił się dlatego, iż system zarządzania relacyjnymi bazami danych używany przez Grześka nie wie, w jaki sposób należy zinterpretować kilkakrotnie powtarzające się klauzule ORDER BY.

W poleceniu UNION można zastosować **tylko jedną klauzulę ORDER BY umieszoną na samym końcu całego polecenia**. Wynika to z faktu, iż polecenie to łączy i grupuje wyniki pochodzące z kilku poleceń SELECT.

Jest jeszcze kilka innych zagadnień związanych z poleceniem UNION, o których powinieneś wiedzieć.

Reguły polecenia UNION

Ilości kolumn w każdym z poleceń SELECT **muszą być identyczne**.

Nie można pobrać dwóch kolumn z pierwszej tabeli i tylko jednej kolumny z drugiej.

W każdym z poleceń SELECT należy także zastosować te same wyrażenia i funkcje agregujące.

Poszczególne polecenia SELECT można podawać w dowolnej kolejności; nie będzie to miało żadnego wpływu na ostateczne wyniki zapytania.

Reguły polecenia UNION

Domyślnie z wyników polecenia UNION są usuwane wszystkie powtarzające się wartości.

Typy danych w poszczególnych kolumnach muszą być takie same bądź musi istnieć możliwość ich konwersji.

Jeśli z jakiegoś powodu chcemy, by w wynikach polecenia UNION występowały powielające się wartości, to możemy zastosować polecenie UNION ALL. Zwróci ono wszystkie wiersze, a nie jedynie te unikalne.

Reguły stosowania poleceń UNION w działaniu

Ilość kolumn w poleceniach SELECT łączonych przy użyciu polecenia UNION musi być taka sama. Nie możesz pobrać dwóch kolumn z jednej tabeli i tylko jednej kolumny z następnej tabeli.

W każdym poleceniu SELECT musisz pobrać taką samą ilość kolumn.

```

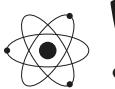
SELECT stanowisko FROM praca_aktualna
UNION
SELECT stanowisko FROM praca_poszukiwana
UNION
SELECT stanowisko FROM oferty_pracy
ORDER BY stanowisko;
  
```

Jeśli chcesz posortować wyniki, to możesz umieścić klauzulę ORDER BY za ostatnim z łączonych poleceń SELECT. Klauzula ta spowoduje posortowanie wyników całego polecenia.

stanowisko
fryzjer
kelnerka
księgowy
nauczycielka
opiekun do kotów
prawnik
programista
szef kuchni
twórca stron WWW
zdun

Oto przykładowe wyniki,
jakie może zwrócić
powyższe zapytanie.

W tym przykładzie wszystkie kolumny zwracane w poszczególnych zapytaniach są typu VARCHAR.
W efekcie także kolumna w tabeli wynikowej będzie typu VARCHAR.



WYSIL
SZARE KOMÓRKI

Jak sądzisz, co by się stało, gdyby kolumny zwracane przez poszczególne polecenia SELECT zastosowane w poleceniu UNION były różnych typów?

UNION ALL

Polecenie UNION ALL działa dokładnie tak samo jak polecenie UNION, z tym że zwraca wszystkie wartości zapisane w kolumnach, a nie jedynie po jednej z grupy identycznych wartości.

SELECT stanowisko FROM praca_aktualna

UNION ALL

Tym razem chcemy wyświetlić listę wszystkich wartości zapisanych w kolumnie stanowisko we wszystkich trzech tabelach.

SELECT stanowisko FROM praca_poszukiwana

UNION ALL

SELECT stanowisko FROM oferty_pracy

ORDER BY stanowisko;

stanowisko
fryzjer
kelnerka
księgowy
nauczycielka
nauczycielka
opiekun do kotów
prawnik
prawnik
prawnik
prawnik
programista
programista
programista
szef kuchni
twórca stron WWW
twórca stron WWW
zdun

Tym razem nazwy stanowisk na liście powtarzają się.

Jak do tej pory wszystkie tworzone przez nas polecenia UNION operowały na kolumnach tego samego typu. Jednak może się zdarzyć, że będziemy chcieli utworzyć polecenie UNION zwracające kolumny różnych typów.

Pisząc, że musi istnieć możliwość konwersji typów danych, mamy na myśli, że o ile to będzie możliwe, zwracane dane zostaną skonwertowane do zgodnych typów; jeśli natomiast taka konwersja nie będzie możliwa, to próba wykonania polecenia się nie powiedzie.

Założymy, że utworzyliśmy polecenie UNION składające się z dwóch zapytań, z których pierwsze zwraca kolumnę typu INTEGER, a drugie typu VARCHAR. Ponieważ danych typu VARCHAR nie można skonwertować do postaci liczb całkowitych, zatem w wynikowej tabeli polecenia UNION wartości typu INTEGER zostaną skonwertowane do typu VARCHAR.

Utworzenie tabeli na podstawie wyników polecenia UNION

Nie możemy w prosty sposób określić typu danych zwróconych przez polecenie UNION, chyba że je w jakiś sposób „przechwycimy”. Na przykład możemy zastosować polecenie CREATE TABLE AS, by przechwycić wyniki polecenia UNION i dokładniej je przeanalizować.

Polecenie CREATE TABLE AS pobiera wyniki zwrócone przez zapytanie SELECT i zapisuje je w nowej tabeli. W poniższym przykładzie tworzymy nową tabelę o nazwie `moja_unia` i zapisujemy w niej wyniki zwrócone przez polecenie UNION.

Nazwa nowej tabeli.

```
CREATE TABLE moja_unia AS
```

```
SELECT stanowisko FROM praca_aktualna UNION
SELECT stanowisko FROM praca_poszukiwana UNION
SELECT stanowisko FROM oferty_pracy;
```

To jest to samo polecenie UNION, które utworzyliśmy już wcześniej. Tabele można utworzyć na podstawie dowolnego polecenia SELECT.

Zaostrz ołówek



Utwórz polecenie UNION łączące kolumnę `id_kontaktu` z tabeli `praca_aktualna` oraz kolumnę `pensja` z tabeli `oferty_pracy`.

.....
.....
.....

Spróbuj odgadnąć, jakie będą typy danych wyników, a następnie zapisz wyniki polecenia UNION w nowej tabeli, używając do tego celu polecenia CREATE TABLE AS.

.....
.....
.....

Wykonaj polecenie DESC i przekonaj się, czy dobrze określiłeś typ danych wyników.

.....

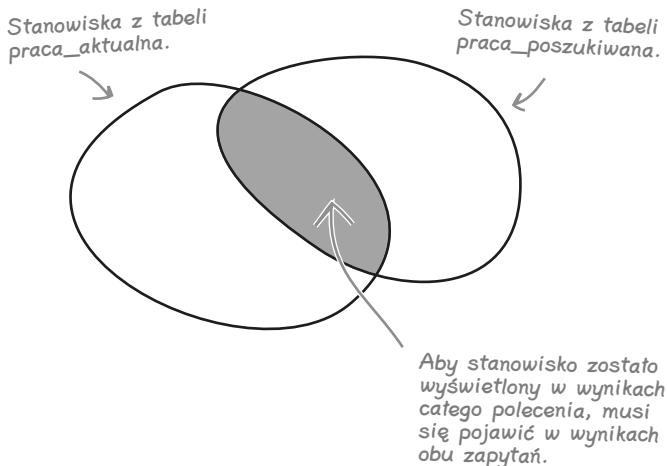
→ Odpowiedzi znajdziesz na stronie 481

Polecenia INTERSECT i EXCEPT

Polecenia INTERSECT oraz EXCEPT są stosowane niemal tak samo jak polecenie UNION i podobnie jak ono operują na nakładających się zbiorach wyników z poszczególnych polecień SELECT.

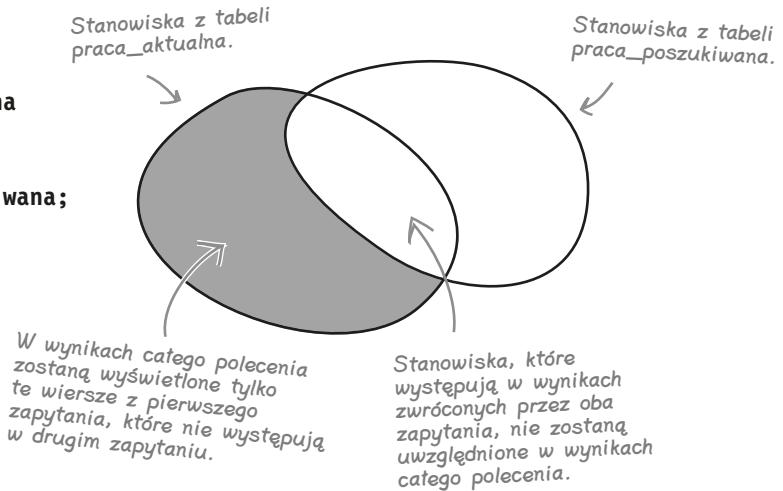
Pierwsze z tych poleceń — INTERSECT — zwraca tylko te rekordy z pierwszego zapytania, które występują także w drugim zapytaniu.

```
SELECT stanowisko FROM praca_aktualna  
INTERSECT  
SELECT stanowisko FROM praca_poszukiwana;
```

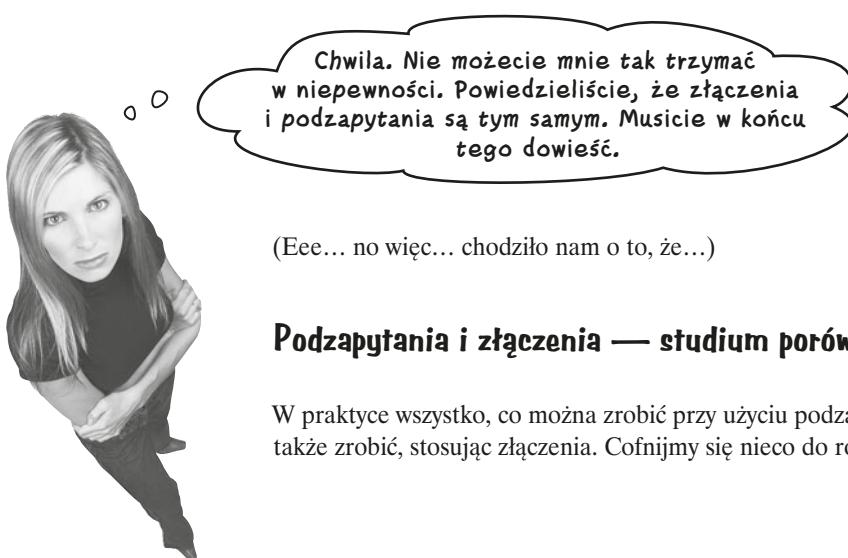


Z kolei polecenie EXCEPT zwraca tylko te wiersze z pierwszego zapytania, które **nie** występują w drugim.

```
SELECT stanowisko FROM praca_aktualna  
EXCEPT  
SELECT stanowisko FROM praca_poszukiwana;
```



Skończyliśmy ze złączami, czas zająć się czymś nowym



Podzapytania i złączenia — studium porównawcze

W praktyce wszystko, co można zrobić przy użyciu podzapytań, można także zrobić, stosując złączenia. Cofnijmy się nieco do rozdziału 9.

Przedstawiamy podzapytania

Podzapytania

Aby w jednym zapytaniu wykonać te same operacje, które Grzesiek wykonał, używając dwóch poleceń SQL, będziemy musieli skorzystać z **podzapytania** — czyli zapytania umieszczonego wewnątrz innego zapytania.

Dругie zapytanie, którego użyjemy do pobrania pasujących rekordów z tabeli zawodów, nazywamy zapytaniem **ZĘWNĘTRZNYM**, gdyż będzie ono zawierało w sobie drugie — **WEWNĘTRZNE** — zapytanie. Zobaczmy, jak to wszystko działa:

zapytanie ZĘWNĘTRZNE

```
SELECT mk.imie, mk.nazwisko, mk.telefon, pa.stanowisko
FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk
WHERE pa.stanowisko IN ('kucharz', 'fryzjer', 'kelnier', 'projektant stron WWW', 'tworca stron WWW');

Ta część tworzy zapytanie zewnętrzne.
A ta część pozwala usunąć i zastąpić
częścią Pierwszego zapytania, które
w ten sposób utworzy zapytanie
wewnętrzne.
```

zapytanie WEWNĘTRZNE

```
SELECT stanowisko FROM oferty_pracy
GROUP BY stanowisko ORDER BY stanowisko;
```

część naszego podzapytania
która stanie się zapytaniem
wewnętrzny.

Łączymy dwa zapytania w zapytanie z podzapytaniem

Zatem połączymy dwa zapytania w jedno. Pierwsze z nich jest nazywane zapytaniem **zewnętrznym**. Z kolei drugie zapytanie, umieszczone wewnątrz pierwszego, nazywamy zapytaniem **wewnętrzny**.

zapytanie ZĘWNĘTRZNE
+
zapytanie WEWNĘTRZNE

= zapytanie z podzapytaniem

Zapytanie zewnętrzne

```
SELECT mk.imie, mk.nazwisko, mk.telefon, pa.stanowisko
FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk
WHERE pa.stanowisko IN (SELECT stanowisko FROM oferty_pracy);
```

Twoje połączone zapytania tworzą zapytanie z podzapytaniem.

Jeżeli nie masz już pojęcia, co zapisywano nazwą
stanowisk z pierwotnego zapytania,
dyskutuj z nimi na temat tego zapytania!

To te same wyniki, co wcześniej,
jednak zapisaliśmy je, używając
tylko jednego zapytania!

mk.imie	mk.nazwisko	mk.telefon	pa.stanowisko
Jarek	Lachowicz	22 32234567	kucharz
Wojciech	Marcinkiewicz	22 34567890	fryzjer
Stefan	Milski	22 0983453	projektant stron WWW
Jarek	Celiński	22 7326984	twórca stron WWW
Jurek	Graczyński	22 9763467	twórca stron WWW

416 Rozdział 9.

jestesztutaj ▶ 417

jestesztutaj ▶ 473

Zamiana podzapytania na złączenie

Poniżej przedstawiliśmy nasze pierwsze podzapytanie, które przedstawiłyśmy w rozdziale 9.:

Zapytanie zewnętrzne.

```
SELECT mk.imie, mk.nazwisko, mk.telefon, pa.stanowisko  
FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk  
WHERE pa.stanowisko IN (SELECT stanowisko FROM oferty_pracy);
```

Zapytanie wewnętrzne.

A oto wyniki, które uzyskaliśmy, wykonując to zapytanie.

mk.imie	mk.nazwisko	mk.telefon	pa.stanowisko
Janek	Lachowicz	22 3490937	kucharz
Wanda	Malczewska	22 3495870	fryzjer
Stefan	Miller	22 0983453	projektant stron WWW
Jarek	Celiński	22 2325694	twórca stron WWW
Julek	Graczyński	22 9763467	twórca stron WWW

Zaostrz ołówek



A oto zapytanie i klauzula WHERE z podzapytaniem zapisanym w postaci złączenia INNER JOIN:

```
SELECT mk.imie, mk.nazwisko, mk.telefon, pa.stanowisko  
FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk  
INNER JOIN oferty_pracy op  
ON pa.stanowisko = op.stanowisko;
```

Możesz zastąpić podzapytanie występujące w klauzuli WHERE złączeniem typu INNER JOIN.

Wyjaśnij, dlaczego zastosowanie przedstawionego powyżej polecenia INNER JOIN spowoduje wygenerowanie takich samych danych jak w przypadku zastosowania podzapytania.

.....
.....
.....
.....
.....

Które z tych poleceń można według Ciebie łatwiej zrozumieć?

.....
.....
.....
.....
.....

Odpowiedzi znajdziesz na stronie 481



A jeśli we wszystkich swoich poleceńach używałbym podzapytań, to czy będę je teraz musiał przepisywać od nowa, stosując dla odmiany złączenia?

Nie. Jeśli te podzapytania dobrze spełniają swoje zadania i robią to, co mają robić, to nie musisz ich zmieniać.

Można jednak wskazać takie sytuacje, w których jedno z tych rozwiązań jest zdecydowanie lepsze od drugiego.

Pogawędkи przy kominku



Złączenie

W przeważającej większości przypadków to właśnie ja jestem zdecydowanie lepszym rozwiązaniem. Łatwiej mnie zrozumieć i działam znacznie szybciej niż te wszystkie stare Podzapytania.

Niby co? Kogo nazywasz „starym”? W niektórych systemach zarządzania bazami danych pojawiłem się zdecydowanie później niż Złączenia. Dodano mnie, gdyż tak wielu programistów chciało mnie stosować.

Ale ja radziłem sobie doskonale i bez ciebie. Poza tym mnie można łatwiej zrozumieć.

Kogo próbujesz zrobić w konia? Łatwiej... z tymi wszystkimi słowami kluczowymi INNER i OUTER? One tak łatwo mogą wprowadzić człowieka w błąd...

To twoja opinia. A co z tymi bzdurnymi podzapytaniami SKORELOWANYMI i NIESKORELOWANYMI?

No dobra... obaj mamy swoje własne słownictwo; tego nie da się ukryć. Jednak dzięki mnie można zazwyczaj określić niezależnie odpowiedzi na wewnętrzną i zewnętrzną część zapytania.

→ Rozmowa jest kontynuowana na następnej stronie.

Rozmowa przy kominku

Pogawędkи при коминку



Temat dzisiejszej rozmowy: **Złączenie czy Podzapytanie, które jest lepsze?**

Złączenie

Podzapytanie

Nie zawsze, kolego SKORELOWANE Podzapytanie.
Ale nich będzie, zostawmy to na chwilę. Cóż, mogę przyznać,
że najlepiej radzę sobie w sytuacjach, gdy w wynikach mają się
pojawić kolumny pochodzące z wielu różnych tabel. Prawdę
mówiąc, w takich przypadkach jestem jedynym sensownym
rozwiązańiem.

I właśnie dlatego kiepsko sobie radzisz tam, gdzie pojawiają
się wartości zagregowane. Nie możesz korzystać z wartości
zagregowanych w klauzuli WHERE bez użycia podzapytania.
A to niezupełnie zgadza się z twierdzeniami o zwracaniu
wielu kolumn. Cóż... jesteś takie skomplikowane.

Może masz trochę racji, ale wcale nie jest tak trudno
zorientować się, co robię. Poza tym można nawet zastosować
nazwy zastępcze, by uniknąć ciągłego wpisywania długich
nazw tabel.

No tak... skoro już rozmawiamy o tych nazwach
zastępczych. Jeśli o mnie chodzi, to uważam, że one jeszcze
bardziej wszystko utrudniają. Niemniej jednak, jeśli byś
chciał wiedzieć, to ja też mogę ich używać. Jednak kiedy ja
to robię, wszystko staje się znacznie prostsze. Ale ważne
jest to, że w moim przypadku w ogóle stosowanie ich
najczęściej nie jest potrzebne.

Tralalala. Co... jesteś za dobre na nazwy zastępcze?
I uważaś, że jesteś o tyle prostsze ode mnie? To powiedz,
mądrało, co z tymi skorelowanymi podzapytaniami? One są
znacznie bardziej skomplikowane i pogmatwane niż cokolwiek,
co ja mógłbym wymyślić.

Cóż, to prawda. Jednak ja wiem o jednej rzeczy, która czyni
mnie całkowicie odmiennym od ciebie. Otóż mnie można
używać w poleceniach UPDATE, INSERT i DELETE.

Pozer...



Ćwiczenie

Przyjrzyj się przedstawionym poniżej zapytaniom, które napisaliśmy w rozdziale 9., i sprawdź, czy wiesz, jak zapisać je bez stosowania podzapytań, czy też preferujesz zostawić podzapytania. Złączenia są dozwolone.

Zapytanie generuje listę stanowisk pracy, dla których pensja jest równa najwyższej pensji z tabeli `oferty_pracy`.

```
SELECT stanowisko FROM oferty_pracy WHERE pensja = (SELECT  
MAX(pensja) FROM oferty_pracy);
```

.....
.....
.....

Czy lepiej jest zostawić podzapytanie?

Zapytanie generuje listę imion i nazwisk osób, których zarobki przekraczają średnią pensję.

```
SELECT mk.imie, mk.nazwisko FROM moje_kontakty mk  
NATURAL JOIN praca_aktualna pa WHERE pa.pensja > (SELECT  
AVG(pensja) FROM praca_aktualna);
```

.....
.....
.....

Czy lepiej jest zostawić podzapytanie?

Rozwiązywanie kolejnego ćwiczenia



Rozwiązywanie ćwiczenia

Przyjrzyj się przedstawionym poniżej zapytaniom, które napisaliśmy w rozdziale 9., i sprawdź, czy wiesz, jak zapisać je bez stosowania podzapytań, czy też preferujesz zostawić podzapytania. Łączenia są dozwolone.

Zapytanie generuje listę stanowisk pracy, dla których pensja jest równa najwyższej pensji z tabeli `oferty_pracy`.

```
SELECT stanowisko FROM oferty_pracy WHERE pensja = (SELECT  
MAX(pensja) FROM oferty_pracy);
```

**SELECT stanowisko FROM oferty_pracy ORDER
BY pensja DESC LIMIT 1;**

To zapytanie zwraca tylko pojedynczy wynik — wiersz zawierający stanowisko z maksymalną pensją w tabeli.

Czy lepiej jest zostawić podzapytanie? **Nie.**

Zapytanie generuje listę imion i nazwisk osób, których zarobki przekraczają średnią pensję.

```
SELECT mk.imie, mk.nazwisko FROM moje_kontakty mk  
NATURAL JOIN praca_aktualna pa WHERE pa.pensja > (SELECT  
AVG(pensja) FROM praca_aktualna);
```

O rany, przecież nie możemy skorzystać z klauzuli `LIMIT` i `ORDER BY` żeby pobrać średnie wartości. Czyli nie uda nam się skorzystać z podobnego rozwiązania jak w poprzednim zapytaniu.

Czy lepiej jest zostawić podzapytanie?

Tak.

W poprzednim rozwiązyaniu mogliśmy zastosować klauzulę `LIMIT`, by pobrać najwyższą pensję z uporządkowanej listy wszystkich pensji. Jednak w tym przypadku nasze pensje większe od średniej nie mogą być posortowane, zatem nie możemy ich pobrać, używając klauzuli `LIMIT`.

Złączenie zwrotne jako podzapytanie

Zobaczyłeś już, w jaki sposób można przekształcić podzapytanie w złączenie, a teraz zajmijmy się zamianą złączenia zwrotnego w podzapytanie.

Czy pamiętasz kolumnę `id_szefa`, którą dodaliśmy jakiś czas temu do tabeli `kłowni_informacje`? A oto zastosowane przez nas wcześniej złączenie zwrotne, w którym jednej kopii tabeli `kłowni_informacje` nadaliśmy nazwę zastępczą `k1`, a drugiej `k2`.

PRZED

```
SELECT k1.imie, k2.imie AS szef
FROM kłowni_informacje k1 ← Pierwsza kopia tabeli
INNER JOIN kłowni_informacje k2
ON k1.id_szefa = k2.id;           ← Druga kopia tabeli
                                    kłowni_informacje
```

PO

Jeśli zmienimy złączenie zwrotne na podzapytanie, to będzie to podzapytanie SKORELOWANE. W naszym przypadku podzapytanie zależy od zapytania zewnętrznego, które jest niezbędne do pobrania odpowiedniego identyfikatora szefa (`id_szefa`), a oprócz tego wyniki podzapytania są umieszczone na liście wyników zewnętrznego polecenia `SELECT`.

```
SELECT k1.imie,
(SELECT imie FROM kłowni_informacje
 WHERE k1.id_szefa = id) AS szef
FROM kłowni_informacje k1;
```

Zapytanie zewnętrzne.

Podzapytanie jest umieszczone na liście wyników polecenia `SELECT`.

Podzapytanie jest zależne od wyników zapytania zewnętrznego, które określa identyfikator szefa; a zatem jest to podzapytanie skorelowane.

Ta kolumna określa, kto jest szefem poszczególnych klaunów.

`kłowni_informacje`

id	imie	id_szefa
1	Eklerka	3
2	Pan Pimpus	5
3	Pani Smyk	10
4	Pan Hobo	3
5	Klarabela	10
6	Skuter	3
7	Zippo	3
8	Balbina	5
9	Gonzo	5
10	Pan Smyk	10

Firma Grześka rozwija się

Grzesiek był bardzo zajęty, studiuje tajniki złączeń i podzapytań. Dlatego zatrudnił znajomych, żeby mu pomagali w pisaniu i wykonywaniu mniej skomplikowanych zapytań.



Szkoda tylko, że Kuba i Franek raczej nie wiedzą, co robią. Grzesiek ma bowiem zamiar przetestować, co się stanie, kiedy kilka osób o raczej mizernym poziomie znajomości języka SQL zacznie jednocześnie korzystać z bazy danych.

Zaostrz ołówek**Rozwiązańe
ze str. 471**

Utwórz polecenie UNION łączące kolumnę id_kontaktu z tabeli praca_aktualna oraz kolumnę pensja z tabeli oferty_pracy.

```
SELECT id_kontaktu FROM praca_aktualna
UNION SELECT pensja FROM praca_aktualna;
```

Spróbuj odgadnąć, jakie będą typy danych wyników, a następnie zapisz wyniki polecenia UNION w nowej tabeli, używając do tego celu polecenia CREATE TABLE AS.

```
CREATE TABLE moja_tabela AS SELECT
id_kontaktu FROM praca_aktualna UNION
SELECT pensja FROM praca_aktualna;
```

Wykonaj polecenie DESC i przekonaj się, czy dobrze określiłeś typ danych wyników.

DEC(12,2)

Zaostrz ołówek**Rozwiązańe
ze str. 474**

A oto zapytanie i klauzula WHERE z podzapytaniem zapisanym w postaci złączenia INNER JOIN:

```
SELECT mk.imie, mk.nazwisko, mk.telefon, pa.stanowisko
FROM praca_aktualna AS pa NATURAL JOIN moje_kontakty AS mk
INNER JOIN oferty_pracy op
ON pa.stanowisko = op.stanowisko;
```

Możesz zastąpić podzapytanie występujące w klauzuli WHERE złączeniem typu INNER JOIN.

Wyjaśnij, dlaczego zastosowanie przedstawionego powyżej polecenia INNER JOIN spowoduje wygenerowanie takich samych danych jak w przypadku zastosowania podzapytania.

Złączenie INNER JOIN wyświetli wyniki wyłącznie w przypadku, gdy będzie spełniony warunek

pa.stanowisko = op.stanowisko, co odpowiada poniższej klauzuli WHERE z podzapytaniem:

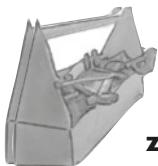
WHERE pa.stanowisko IN (SELECT stanowisko FROM oferty_pracy);

Które z tych poleceń można według Ciebie łatwiej zrozumieć?

Trudno podać jednoznaczną odpowiedź na to pytanie! Jednak podana odpowiedź

pokazuje, że zaczynasz już myśleć o tym, czego będziesz mógł używać w przyszłości,

kiedy będziesz korzystał ze swoich własnych danych.



Przybornik SQL

Teraz to już naprawdę radzisz sobie jak stary wyjadacz. Poznałeś złączenia zewnętrzne, złączenia zwrotne, dowiedziałeś się, jak łączyć wyniki kilku zapytań, a nawet jak zmieniać złączenia na podzapytania i na odwrót. Kompletną listę porad znajdziesz w dodatku C, zamieszczonym na końcu książki.

KLUCZ OBCY ODWOŁUJĄCY SIĘ DO TEJ SAMEJ TABELI

To klucz obcy umieszczony w tej samej tabeli, z której pochodzi klucz główny, i używany do jakichś innych celów.

UNION oraz UNION ALL

Polecenie UNION łączy wyniki dwóch lub większej ilości zapytań w jedną tabelę wynikową, której postać zależna jest od kolumn wybranych w poleceniach SELECT. Polecenie to ukrywa powielające się wartości; jeśli interesują Cię wszystkie, nawet powtarzające się wartości, to zastosuj polecenie UNION ALL.

LEWOSTRONNE ZŁĄCZENIE Zewnętrzne

LEWOSTRONNE ZŁĄCZENIE Zewnętrzne pobiera wszystkie wiersze z lewej tabeli i dopasowuje je do wierszy w tabeli PRAWej.

ZŁĄCZENIE ZWROTNE

ZŁĄCZENIE ZWROTNE pozwala tworzyć zapytania operujące na jednej tabeli w taki sposób, jak gdyby odwoływały się one do dwóch odrębnych tabel zawierających identyczne dane.

CREATE TABLE AS

Tego polecenia można użyć, by zapisać wyniki polecenia SELECT w nowej tabeli.

INTERSECT

Użyj tego polecenia, by zwrócić tylko te wartości, które są dostępne w wynikach zarówno pierwszego, JAK I drugiego zapytania.

EXCEPT

Użyj tego polecenia, by zwrócić tylko te wartości, które są dostępne w wynikach pierwszego, LECZ NIE ma ich w wynikach drugiego zapytania.

11. Ograniczenia, widoki i transakcje

Zbyt wielu kucharzy psuje bazę danych

Zobacz... to właśnie tu się pomyliłeś. W polu „ilość” wpisalteś „cała masa”.

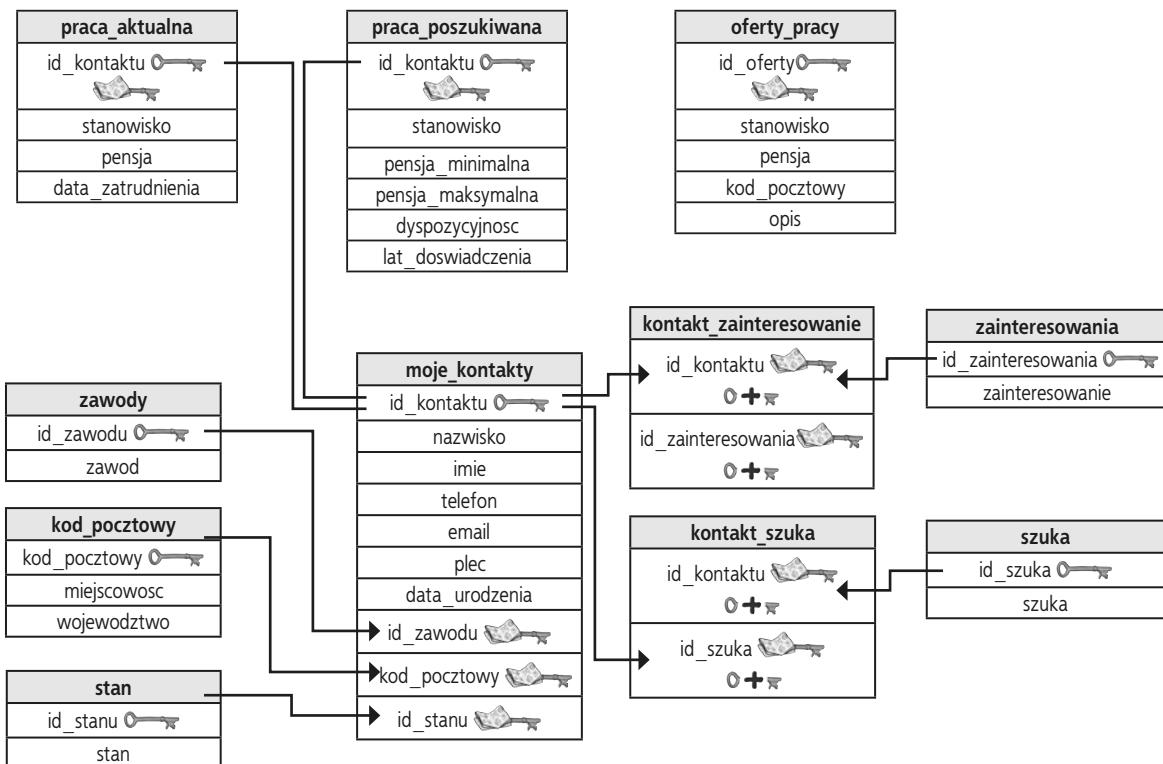


Twoje bazy danych rozrosły się i muszą z nich korzystać także inne osoby. Problem polega na tym, że niektóre z nich nie będą znały języka SQL równie dobrze jak Ty. Musisz zatem mieć jakieś sposoby, by uniemożliwić im wprowadzanie nieprawidłowych danych, techniki pozwalające ukryć przed nimi wybrane dane oraz mechanizmy, które uchronią je przed wzajemnym przeszkadzaniem sobie, gdy jednocześnie będą modyfikować zawartość bazy. W tym rozdziale zacznijemy ochroniać nasze dane przed błędami popełnianymi przez inne osoby. Witamy w Bazie Obronnej, w części 1.

Grzesiek zatrudnił pomocników

Grzesiek zatrudnił dwie osoby, by pomogły mu w prowadzeniu rozwijającej się firmy. Kuba ma się zajmować wprowadzaniem do bazy informacji o nowych klientach, natomiast Franek — dopasowywaniem osób z bazy do dostępnych ofert pracy.

Grzesiek poświęcił trochę czasu na wyjaśnienie nowym pracownikom struktury swojej bazy oraz opisał postać i przeznaczenie poszczególnych tabel.



Pierwszy dzień Kuby: Dopuszczanie nowego klienta

Kuba siedzi przy swoim biurku, gdy nagle ikona internetowego komunikatora sygnalizuje mu otrzymanie nowej wiadomości od Grześka:

Pogadanki zespołowe: Oto dane do zapisania w bazie

 Grzesiek Hej, Kuba, czy możesz dopisać kogoś do bazy danych?

 Kuba Jasna sprawa, oczywiście.

 Grzesiek Na razie mam jedynie częściowe informacje, resztę dostanę później.

 Kuba Patrycja Mruczyńska, 5556667, mru_pat@jakisemail.pl, kod pocztowy 33-333

 Grzesiek Data urodzenia 15.4.1978.

 Grzesiek Jeśli chodzi o zawód, to wpisz „nauczycielka”, a stan — „zamężna” (w tym przypadku będziesz musiał sprawdzić odpowiednią wartość, korzystając z polecenia SELECT; jego prawidłową składnię znajdziesz w moich notatkach, które ci przygotowałem).

 Kuba Nie wygląda to na zbyt skomplikowane zadanie, już się za nie biorę.

 Grzesiek Dzięki

 Kuba

Nie ma za co |



**WYSIL
SZARE KOMÓRKI**

Czy potrafisz napisać polecenie zapisujące nową osobę w bazie?

Kuba unika wartości NULL

Podczas wpisywania danych Kuba zdaje sobie sprawę, że nie ma pewności, czy Patrycja jest kobietą, czy mężczyzną. Grześka akurat nie ma w poblizu, więc Kuba postanowił sam podjąć decyzję. Zdecydował się w kolumnie płci wpisać wartość „X”.

Poniżej przedstawiliśmy napisane przez niego polecenia SQL:

Kuba pobiera identyfikator zawodu, id_zawodu, z tabeli zawody:

```
SELECT id_zawodu FROM zawody WHERE zawod = "nauczycielka";
```

id_zawodu
19

Oto identyfikator odpowiadający zawodowi „nauczycielka”. Kuba będzie mógł go zapisać w odpowiednim polu tabeli moje_kontakty.

Następnie Kuba pobiera identyfikator stanu z tabeli stan:

```
SELECT id_stanu FROM stan WHERE stan = "zamezna";
```

moje_kontakty
id_kontaktu
nazwisko
imie
telefon
email
plec
data_urodzenia
id_zawodu
kod_pocztowy
id_stanu

id_stanu
4

Oto identyfikator odpowiadający stanowi „zamężna”.

W końcu Kuba zapisuje dane w bazie, używając literki „X” jako określenia nieznanej płci.

Jeśli w tabeli istnieje kolumna, której wartości są automatycznie inkrementowane, to nie musimy jawnie podawać jej wartości. Dwa apostrofy informują bazę, że w danej kolumnie należy automatycznie zapisać odpowiednią wartość.

```
INSERT INTO moje_kontakty VALUES('', 'Mruczynska', 'Patrycja',
'5556667', 'mru_pat@jakisemail.pl', 'X', 1978-04-15,
19, '33333', 4);
```

To identyfikatory, które Kuba odszukał, korzystając z dwóch wcześniejszych zapytań. Również dobrze Kuba mógłby uzyskać te wartości, używając podzapytań.

To jest wartość, którą Kuba zdecydował się zapisać w kolumnie plec, gdyż nie chciał ani zgadywać, ani wpisywać wartości NULL.

Trzy miesiące później

Grzesiek postanowił przeprowadzić badania demograficzne na swojej bazie. Chce wiedzieć, ile z osób zapisanych w tabeli `moje_kontakty` to mężczyźni, a ile kobiety oraz jaka jest sumaryczna liczba zarejestrowanych osób. Wykonał w tym celu trzy zapytania: zwracające liczbę mężczyzn, liczbę kobiet oraz sumaryczną liczbę osób.

```
SELECT COUNT(*) AS Kobiety FROM moje_kontakty WHERE plec = 'K';
```

Kobiety
5975

Grzesiek dowiedział się, że w jego tabeli `moje_kontakty` jest 5975 wierszy z literką 'K' w kolumnie `plec`.

```
SELECT COUNT(*) AS Meczczyni FROM moje_kontakty WHERE plec = 'M';
```

Meczczyni
6982

Są także 6982 wiersze z literką 'M'.

```
SELECT COUNT(*) AS Wszystkich FROM moje_kontakty;
```

Wszystkich
12970

Używając tego zapytania, Grzesiek sprawdził sumaryczną ilość wszystkich osób zarejestrowanych w tabeli `moje_kontakty`.

Grzesiek zauważył jednak, że liczba mężczyzn po dodaniu do liczby kobiet nie odpowiada sumarycznej liczbie wszystkich zarejestrowanych osób. Wyliczył, że 13 wierszy nie zostało uwzględnionych ani w pierwszym, ani w drugim zapytaniu. Grzesiek spróbował zatem wykonać kolejne zapytanie:

```
SELECT plec FROM moje_kontakty  
WHERE plec <> 'K' AND plec <> 'M';
```

plec
X
X
X
X
X
X
X
X
X
X
X
X
X

Szukając zgubionych rekordów, Grzesiek znalazł rekordy z wartością 'X' w kolumnie `plec`.

**WYSIL
SZARE KOMÓRKI**

W jaki sposób Kuba mógł uniknąć wpisywania wartości X?

Uwaga, KONTROLA: dodawanie OGRANICZEŃ SPRAWDZAJĄCYCH

We wcześniejszych rozdziałach przedstawiliśmy już kilka różnych ograniczeń, jakie można nakładać na kolumny. Jak się można domyślić, **ograniczenie** określa, jakie wartości można zapisać w kolumnie. Ograniczenia dodaje się podczas tworzenia tabeli. Niektóre z ograniczeń, jakie poznaleś już w tej książce, to: NOT NULL, PRIMARY KEY, FOREIGN KEY oraz UNIQUE.

Istnieje także kolejne ograniczenie, jakie można nakładać na kolumny. Jest nim ograniczenie **CHECK**. Poniżej przedstawiliśmy jego przykład. Założymy, że posiadamy świnę skarbonkę i chcemy rejestrować wszystkie monety, jakie do niej wrzucamy. Można do niej wrzucać jedynie monety o większych nominałach — 1, 2 oraz 5 złotych. Każdy z tych rodzajów monet oznaczmy literą; będą to odpowiednio: J, D i P. W poniższej tabeli zastosowano ograniczenie CHECK, by określić grupę wartości, jakie można zapisywać w kolumnie moneta:

```
CREATE TABLE skarbonka
```

```
(
```

```
    id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,  
    moneta CHAR(1) CHECK (moneta IN ('J', 'D', 'P'))
```

```
);
```

To ograniczenie sprawdza, czy wartość zapisywana w kolumnie należy do tego zbioru.

Ograniczenie sprawdzające — **CHECK** — określa zbiór wartości, jakie będzie można zapisywać w kolumnie. Można w nim podać taki sam warunek, jaki może być umieszczony w klauzuli WHERE.

Jeśli wartość zapisywana w kolumnie nie spełni ograniczenia CHECK, zostanie zgłoszony błąd.



Ograniczenie CHECK nie wymusza integralności danych w MySQL-u.

W MySQL-u można tworzyć tabele, używając ograniczenia CHECK, jednak serwer bazy danych nie będzie ich używać — po prostu je zignoruje.

Sprawdzanie płci

Gdyby Grzesiek potrafił cofać czas, mógłby zdefiniować w tabeli `moje_kontakty` ograniczenie CHECK dotyczące kolumny `plec`. Teraz nie pozostało mu jednak nic innego, jak zmodyfikować tabelę przy użyciu polecenia ALTER TABLE:

```
ALTER TABLE moje_kontakty
ADD CONSTRAINT CHECK plec IN ('K', 'M');
```

Następnego dnia okazało się, że Kuba nie może wpisywać wartości 'X' w polu płci. Kiedy zapytał o to Grześka, ten wyjaśnił mu nowe ograniczenia i powiedział, że ponieważ nie jest w stanie cofnąć się w czasie, zatem czyni Kubę odpowiedzialnym za wszystkie kontakty z płecią „X” i każe mu określić, jaka ta płeć powinna być w rzeczywistości.

Dlaczego ciągle pojawiają mi się błędy?



Zaostrz ołówek



Napisz, jakie według Ciebie wartości będzie można zapisywać w każdej z czterech kolumn poniższej tabeli.

```
CREATE TABLE tajemnicza_tabela
(
    kolumna1 INT(4) CHECK (kolumna1 > 200),
    kolumna2 CHAR(1) CHECK (kolumna2 NOT IN ('x', 'y', 'z')),
    kolumna3 VARCHAR(3) CHECK ('A' = SUBSTRING(kolumna3, 1, 1)),
    kolumna4 VARCHAR(3) CHECK ('A' = SUBSTRING(kolumna4, 1, 1)
                                AND '9' = SUBSTRING(kolumna4, 2, 1))
)
```

Kolumna 1:

Kolumna 2:

Kolumna 3:

Kolumna 4:

Rozwiążanie ćwiczenia

Zaostrz ołówek



Rozwiążanie

Napisz, jakie według Ciebie wartości będzie można zapisywać w każdej z czterech kolumn poniżej tabeli.

```
CREATE TABLE tajemnicza_tabela  
(  
    kolumna1 INT(4) CHECK (kolumna1 > 200),  
    kolumna2 CHAR(1) CHECK (kolumna2 NOT IN ('x', 'y', 'z')),  
    kolumna3 VARCHAR(3) CHECK ('A' = SUBSTRING(kolumna3, 1, 1)),  
    kolumna4 VARCHAR(3) CHECK ('A' = SUBSTRING(kolumna4, 1, 1)  
        AND '9' = SUBSTRING(kolumna4, 2, 1))  
)
```

Możesz łączyć warunki przy użyciu operatorów AND oraz OR.

Kolumna 1.: **Zapisywane wartości muszą być większe od 200.**

Kolumna 2.: **Można zapisać dowolny znak z wyjątkiem liter x, y i z.**

Kolumna 3.: **Pierwszym znakiem zapisywanego łańcucha znaków musi być litera A.**

Kolumna 4.: **Pierwszym znakiem zapisywanego łańcucha musi być litera A, a drugim cyfra 9.**

Nie istniejąca grupa pytań

P: A zatem w ograniczeniu **CHECK** mogę używać takich samych warunków, jakich używam w klauzuli **WHERE**?

O: W zasadzie tak. Możesz używać takich operatorów jak: AND, OR, IN, NOT, BETWEEN oraz kilku innych. Możesz je nawet łączyć, jak pokazaliśmy w powyższym przykładzie. Nie możesz jednak używać podzapytań.

P: Ale skoro nie mogę używać ograniczeń tego typu w MySQL-u, to co mogę zastosować zamiast nich?

O: Na to pytanie nie ma niestety prostej i łatwej odpowiedzi. Niektórzy korzystają z wyzwalaczy, czyli zapytań wykonywanych w momencie spełnienia określonego warunku. Jednak korzystanie z wyzwalaczy nie jest aż tak proste jak z ograniczeń CHECK, pomijając w ogóle fakt, że przedstawienie tego zagadnienia nie mieści się w ramach niniejszej książki.

P: A co się stanie, kiedy spróbuję zapisać w tabeli wartość, która nie spełnia warunku ograniczenia **CHECK**?

O: Baza danych zgłosi błąd, a wartość nie zostanie zapisana w tabeli.

P: A jaki z tego pożytek?

O: W ten sposób możesz zapewnić, że dane zapisywane w tabeli będą sensowne — że nie pojawią się w niej jakieś tajemnicze wartości.

Praca Franka staje się nudząca

Franek zajmował się dopasowywaniem osób do ofert pracy. Franek zaczął zauważać pewne wzorce. Miał w bazie bardzo dużo ofert pracy dla projektantów stron WWW, lecz osób, które chciałyby podjąć pracę na tych stanowiskach, było stosunkowo niewiele. Miał też zarejestrowanych wiele osób piszących na tematy techniczne, jednak ofert pracy na takie stanowiska było niewiele.

Franek każdego dnia wykonuje dziesiątki podobnych zapytań, starając się dopasować osoby z ofertami pracy.

Każdego dnia muszę wielokrotnie wykonywać te same zapytania. To jest naprawdę nudzące.

Bądź FRANKIEM



Twoim zadaniem jest wcielić się w rolę Franka i napisać zapytania, które on codziennie wykonuje. Napisz zapytanie, które odnajdzie wszystkich projektantów stron WWW zarejestrowanych w tabeli `praca_poszukiwana` wraz z ich informacjami kontaktowymi. Napisz także drugie zapytanie, które odnajdzie oferty pracy dla autorów tekstów technicznych.





Bądź FRANKIEM. Rozwiążanie

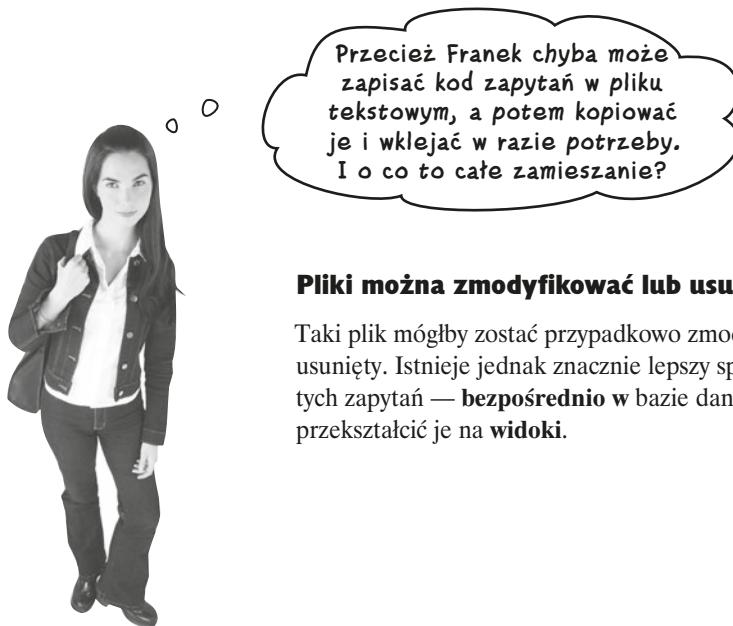
Twoim zadaniem jest wcielić się w rolę Franka i napisać zapytania, które on codziennie wykonuje.
Napisz zapytanie, które odnajdzie wszystkich projektantów stron WWW zarejestrowanych w tabeli `praca_poszukiwana` wraz z ich informacjami kontaktowymi. Napisz także drugie zapytanie, które odnajdzie oferty pracy dla autorów tekstu technicznych.

```
SELECT mk.imie, mk.nazwisko,  
mk.telefon, mk.email  
FROM moje_kontakty mk  
NATURAL JOIN praca_poszukiwana pp  
WHERE pp.stanowisko = 'projektant stron WWW';
```

Grzesiek zazwyczaj zapisuje nazwy stanowisk pracy maty whole literami.

```
SELECT stanowisko, pensja, opis, kod_pocztowy  
FROM oferty_pracy  
WHERE stanowisko = 'autor tekstu technicznego';
```

To nie są złożone zapytania, jednak konieczność ich ciągłego, wielokrotnego wpisywania zmniejsza koncentrację i zwiększa prawdopodobieństwo popełnienia błędu. Frankowi przydałaby się możliwość zapisania zapytań i wyświetlania jedynie ich wyników, bez konieczności każdorazowego wpisywania ich kodu.



Pliki można zmodyfikować lub usunąć.

Taki plik mógłby zostać przypadkowo zmodyfikowany lub usunięty. Istnieje jednak znacznie lepszy sposób zapisania tych zapytań — **bezpośrednio w bazie danych**. Można przekształcić je na **widoki**.

Tworzenie widoku

Tworzenie widoku jest naprawdę bardzo proste. Wystarczy poprzedzić zwyczajne zapytanie poleceniem SQL CREATE VIEW. A zatem utworzymy dwa widoki na podstawie zapytań używanych przez Franka:

```
CREATE VIEW projektanci_stron AS
SELECT mk.imie, mk.nazwisko, mk.telefon, mk.email
FROM moje_kontakty mk
NATURAL JOIN praca_poszukiwana pp
WHERE pp.stanowisko = 'projektant stron WWW';
```

← Można by tu także zastosować złączenie INNER JOIN z klauzulą: ON mk.id_kontaktu = pp.id_kontaktu.

```
CREATE VIEW praca_dla_pisarzy_technicznych AS
SELECT stanowisko, pensja, opis, kod_pocztowy
FROM oferty_pracy
WHERE stanowisko = 'autor tekstów technicznych';
```



E, tam... bułka z masłem!
Ale w jaki sposób mogę teraz skorzystać z takich widoków?



WYSIL
SZARE KOMÓRKI

A jak **Ty** sądzisz, jak wygląda polecenie SQL pozwalające skorzystać z widoku?

Oglądarka własnych widoków

Skorzystajmy z przykładu utworzonego przed chwilą widoku projektanci_stron:

```
CREATE VIEW projektanci_stron AS  
SELECT mk.imie, mk.nazwisko, mk.telefon, mk.email  
FROM moje_kontakty mk  
NATURAL JOIN praca_poszukiwana pp  
WHERE pp.stanowisko = 'projektant stron WWW';
```

Aby wyświetlić zawartość takiego widoku, należy go potraktować tak, jak gdyby był on tabelą. A zatem wystarczy użyć polecenia SELECT:

```
SELECT * FROM projektanci_stron;
```

To jest nazwa widoku.

A oto uzyskane wyniki:

imie	nazwisko	telefon	email
Jan	Miratyński	5558573	jm@jakisemail.com.pl
Samanta	Horeczko	5554578	sama@jakisemail.com.pl
Tadek	Kowalski	5552096	tede@jakisemail.com.pl
Franek	Żebrowski	5554958	fra@jakisemail.com.pl



I tak dalej, aż do wyświetlenia wszystkich rekordów, w których w kolumnie stanowisko została zapisana wartość 'projektant stron WWW'.

Jak właściwie działa widok?

Kiedy używasz widoku w zapytaniu, w rzeczywistości działa on tak, jak gdyby był podzapytaniem. Poniżej na przykładzie zapytania z poprzedniej strony pokazaliśmy, co robi baza danych, kiedy ma wykonać polecenie SELECT, w którym został zastosowany widok:

```
SELECT * FROM projektanci_stron;
```

Polecenie to oznacza: „Zwrć wszystkie wyniki zapytania, które pobiera imię, nazwisko, telefon i e-mail wszystkich osób z tabeli moje_kontakty szukających pracy jako projektant stron WWW”.

```
SELECT * FROM  

(SELECT mk.imie, mk.nazwisko, mk.telefon, mk.email  

FROM moje_kontakty mk  

NATURAL JOIN praca_poszukiwana pp  

WHERE pp.stanowisko = 'projektant stron WWW') AS projektanci_stron;
```

To jest zapytanie, którego użyliśmy podczas tworzenia widoku.

Dla naszego podzapytania tworzymy nazwę zastępczą, aby zapytanie potraktowało ją jako tabelę.



O co chodzi w tej części AS projektanci-stron?
Dlaczego jej potrzebujemy?

Klauzula **FROM** wymaga wskazania tabeli.

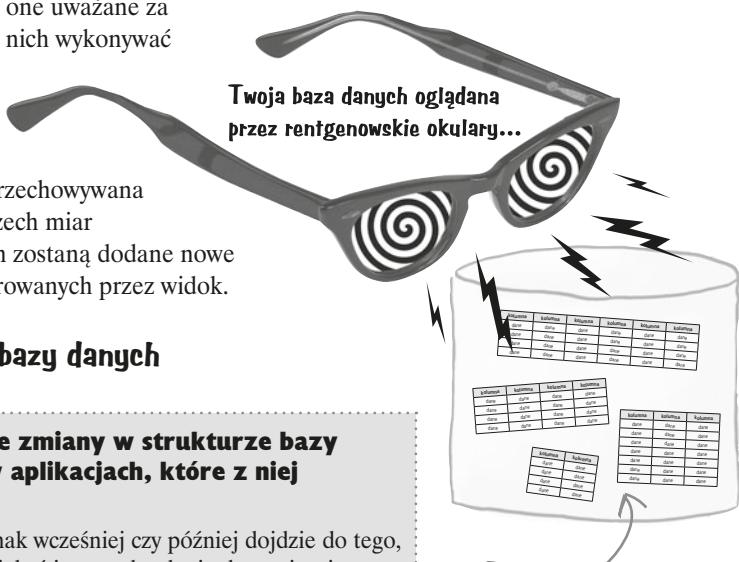
A ponieważ nasze polecenie SELECT zwraca wirtualną tabelę, zatem bez tej nazwy zastępczej SQL nie miałby możliwości operowania na danych z widoku.

Czym są widoki

Najprościej rzec ujmując, widok jest tabelą, która istnieje wyłącznie w momencie, gdy widok jest używany w zapytaniu. Są one uważane za **tabele wirtualne**, gdyż działają jak tabele i można na nich wykonywać te same operacje, co na „zwyczajnych” tabelach.

Jednak tabela wirtualna nie jest zapisywana w bazie danych. Jest tworzona w momencie korzystania z widoku, a następnie zostaje usunięta. Przechowywana jest jedynie nazwa widoku. Takie działanie jest ze wszech miar pożąданie, gdyż za każdym razem, gdy do bazy danych zostaną dodane nowe rekordy, zostaną one uwzględnione w wynikach generowanych przez widok.

Dlaczego widoki są korzystne dla naszej bazy danych



Te tabele istnieją tylko dlatego, że użyliśmy widoku w poleceniu SELECT.

1

Dzięki nim możesz mieć pewność, że zmiany w strukturze bazy danych nie spowodują problemów w aplikacjach, które z niej korzystają.

Nie pisaliśmy o tym w niniejszej książce, jednak wcześniej czy później dojdzie do tego, że swoją znajomość SQL-a wykorzystasz wraz z jakąś inną technologią do napisania aplikacji. Dzięki widokom będziesz w stanie zmieniać strukturę bazy danych i jednocześnie tworzyć widoki udające jej wcześniejszą postać, dzięki czemu unikniesz konieczności modyfikowania aplikacji korzystających z informacji przechowywanych w bazie.

2

Widoki ułatwiają Ci życie, gdyż upraszczają złożone polecenia SQL do postaci bardzo prostego zapytania.

Już nigdy więcej nie będziesz musiał wielokrotnie wpisywać złożonych złączeń i podzapytań. Teraz wystarczy, że na ich podstawie utworzysz widok. Ten widok ukryje wszelkie złożoności zapytania. A jeśli zaczniesz używać SQL-a w skryptach PHP bądź w jakimś innym języku programowania, to zastosowanie widoków znacznie uprości tworzony kod. Zamiast rozbudowanego i złożonego kodu oryginalnego zapytania ze złączeniami i podzapytaniami będziesz mógł używać w swoim kodzie prostych i krótkich poleceń korzystających z widoków. Uproszczenie kodu oznacza natomiast mniejsze prawdopodobieństwo popełnienia błędów oraz możliwość łatwiejszego zrozumienia kodu.

3

Możesz tworzyć widoki ukrywające informacje, które nie są danemu użytkownikowi potrzebne.

Rozważmy ewentualną rozbudowę bazy danych Grześka, która polegałaby na dodaniu do niej informacji na temat kart kredytowych zarejestrowanych osób.

W takim przypadku można by stworzyć widok, który zwróciłby informację o tym, że dana osoba posiada kartę kredytową, jednak nie ujawniałby żadnych szczegółowych informacji o tej karcie. Dzięki widokom możesz zapewnić swoim pracownikom dostęp dokładnie do tych informacji, jakie są im potrzebne, i jednocześnie ochronić ważne i wrażliwe informacje przed nieupoważnionymi osobami.



No dobra, mam dla was trudne pytanie.
Czy mógłbym utworzyć widok, który pokazywałby mi wszystkie osoby z tabeli `praca`—poszukiwana wraz z informacją, ile one aktualnie zarabiają, ile chciałby zarabiać (na podstawie kolumny `pensja_minimalna`) oraz różnicę pomiędzy tymi dwiema wartościami? Innymi słowy, chodzi mi o wzrost pensji związany z nowym, poszukiwanym stanowiskiem. A... oprócz tego przydatyby mi się jeszcze następujące informacje o tych osobach: imię i nazwisko, telefon oraz e-mail.



Ćwiczenie

To jest faktycznie spore wyzwanie, jednak trzeba wiedzieć, że każde polecenie SELECT może posłużyć do utworzenia widoku. Zaczni od określenia odpowiedzi na poniższe pytania, a następnie napisz zapytanie dla Franka i zapisz go jako widok, nadając mu nazwę `wzrost_plac`.

Jakie tabele trzeba będzie zastosować w zapytaniu?

.....

Jakich kolumn z tych tabel należy użyć do wyznaczenia oczekiwanej przyrostu płac?

.....

Jak powinniśmy użyć języka SQL, by utworzyć w wynikach kolumnę „`wzrost`”?

.....

A teraz poniżej napisz zapytanie dla Franka:

.....

.....

.....

.....

.....

.....

Podpowiedź: W poleceniu spróbuj zastosować trzy tabele i dwa złączenia.

Rozwiążanie ćwiczenia



Rozwiążanie ćwiczenie

To jest faktycznie spore wyzwanie, jednak trzeba wiedzieć, że każde polecenie SELECT może posłużyć do utworzenia widoku. Zaczni od określenia odpowiedzi na poniższe pytania, a następnie napisz zapytanie dla Franka i zapisz go jako widok, nadając mu nazwę wzrost_plac.

Jakie tabele trzeba będzie zastosować w zapytaniu?

praca_aktualna, praca_oczekiwana oraz moje_kontakty

Jakich kolumn z tych tabel należy użyć do wyznaczenia oczekiwanej przyrostu płac?

Kolumna pensja z tabeli praca_aktualna oraz kolumna pensja_minimalna z tabeli praca_poszukiwana.

Jak powinniśmy użyć języka SQL, by utworzyć w wynikach kolumnę „wzrost”?

Odejmą aktualną pensję od minimalnej pensji oczekiwanej i nadać tej wartości jakąś nazwę zastępczą.

A teraz poniżej napisz zapytanie dla Franka:

W tym miejscu tworzymy nowy widok o nazwie wzrost_plac.

```
CREATE VIEW wzrost_plac AS
SELECT mk.imie, mk.nazwisko, mk.email, mk.telefon, pa.id_kontaktu, pa.pensja,
pp.pensja_minimalna,
pp.pensja_minimalna - pa.pensja AS wzrost
FROM praca_aktualna pa
INNER JOIN praca_poszukiwana pp
INNER JOIN moje_kontakty mk
WHERE pa.id_kontaktu = pp.id_kontaktu
AND pa.id_kontaktu = mk.id_kontaktu;
```

W dalszej części zapytania zostały użyte dwa złączenia INNER JOIN, które pobierają dane z trzech tabel. Oprócz tego tworzymy nową kolumnę „wzrost”, używając przy tym podstawowej operacji arytmetycznej.

Tu odejmujemy aktualną pensję od minimalnej pensji oczekiwanej, a wynik umieszczamy w kolumnie o nazwie zastępczej „wzrost”.

To faktycznie bardzo rozbudowane zapytanie, jednak teraz w celu wykonania go wystarczy, że Franek wykona następujące polecenie:

**SELECT * FROM wzrost_plac;
i uzyska potrzebne informacje.**

Zaostrz ołówek



Zakładając, że Franek wykona polecenie SELECT przedstawione na stronie 498, używając przy tym widoku wzrost_plac, to w jaki sposób może posortować wyniki alfabetycznie według nazwiska?

→ Odpowiedzi znajdziesz na stronie 519

Wstawianie, aktualizacja i usuwanie danych przy wykorzystaniu widoków

Widoki pozwalają na znacznie więcej niż jedynie pobieranie danych z tabel przy użyciu polecenia SELECT. W niektórych przypadkach można ich także używać w poleceniach UPDATE, INSERT oraz DELETE do aktualizacji, zapisu i usuwania danych.



A zatem oznacza to, że mogę utworzyć widok, który pozwoli mi modyfikować zawartość tabeli?

Owszem, jednak nie jest to raczej warte zachodu.

Jeśli w widoku wyliczane są wartości zagregowane (przy użyciu takich funkcji jak **SUM**, **COUNT** lub **AVG**), to nie będziesz mógł zastosować ich do zmiany danych. Co więcej, jeśli w widoku używane są klauzule **GROUP BY** lub **HAVING**, albo słowo kluczowe **DISTINCT**, to takiego widoku także nie będziesz mógł użyć do zmiany danych w bazie.

W większości przypadków znacznie prościej będzie stosować polecenia **INSERT**, **UPDATE** oraz **DELETE** w tradycyjny sposób, niemniej jednak na następnej stronie pokażemy przykład modyfikacji zawartości bazy przy wykorzystaniu widoku.

Sekret polega na tym, by udawać, że widok jest prawdziwą tabelą

Utwórzmy widok na podstawie naszej nowej tabeli skarbonka. Tabela ta zawiera informacje o monetach, które zbieramy. Każda moneta ma swój identyfikator, kolumnę informacyjną określającą w uproszczony sposób nominał monety (nominał ten może wynosić 1, 2 oraz 5 złotych) oraz rok, w którym moneta trafiła do obiegu.

```
CREATE TABLE skarbonka
(
    id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
    moneta CHAR(1) NOT NULL,
    rok_wybicia CHAR(4)
);
```

A oto dane, które aktualnie są zapisane w tabeli skarbonka:

id	moneta	rok_wybicia
1	J	2003
2	P	2001
3	P	1995
3	D	2000
4	J	1998
5	D	2001
6	P	2007
7	J	2006
8	J	1997
9	P	2000
10	D	1999

A teraz napiszmy widok, który będzie zwracał wyłącznie wiersze z monetami o nominale jednego złotego:

```
CREATE VIEW AS zlotowki_w_skarbonce
SELECT * FROM skarbonka
WHERE moneta = 'J';
```



WYSIL
SZARE KOMÓRKI

Jaką postać będzie miała tabela wynikowa zwrócona po wykonaniu następującego polecenia:

```
SELECT * FROM zlotowki_w_skarbonce;
```



Ćwiczenie

Spróbuj to zrobić w domu. Utwórz tabelę skarbonka oraz dwa widoki sw_jedynki i sw_dwojki, korzystając przy tym z poleceń przedstawionych poniżej.

```
INSERT INTO skarbonka VALUES ('', 'J', 2003), ('', 'P', 2001), ('', 'P', 1995),  
('', 'D', 2000), ('', 'J', 1998), ('', 'D', 2001), ('', 'P', 2007), ('', 'J', 2006),  
('', 'J', 1997), ('', 'P', 2000), ('', 'D', 1999);
```

```
CREATE VIEW sw_jedynki AS SELECT * FROM skarbonka WHERE moneta = 'J';
```

```
CREATE VIEW sw_dwojki AS SELECT * FROM skarbonka WHERE moneta = 'D' WITH CHECK OPTION;
```

Zapisz, co się stanie podczas próby wykonania każdego z przedstawionych poniżej poleceń INSERT, DELETE oraz UPDATE. Na samym końcu ćwiczenia narysuj wynikową zawartość tabeli skarbonka.

Wykonując dalszą część ćwiczenia, spróbuj się domyślić, jakie znaczenie mają te słowa.
↗

```
INSERT INTO sw_jedynki VALUES ('', 'J', 1999);
```

```
INSERT INTO sw_jedynki VALUES ('', 'D', 1997);
```

```
INSERT INTO sw_dwojki VALUES ('', 'J', 2005);
```

```
DELETE FROM sw_jedynki WHERE moneta = 'P' OR moneta = 'D';
```

```
UPDATE sw_jedynki SET moneta = 'J' WHERE moneta = 'P';
```



Rozwiążanie ćwiczenia

Spróbuj to zrobić w domu. Utwórz tabelę skarbonka oraz dwa widoki sw_jedynki i sw_dwojki, korzystając przy tym z polecen przedstawionych poniżej.

```
INSERT INTO skarbonka VALUES ('', 'J', 2003), ('', 'P', 2001), ('', 'P', 1995),  
('', 'D', 2000), ('', 'J', 1998), ('', 'D', 2001), ('', 'P', 2007), ('', 'J', 2006),  
('', 'J', 1997), ('', 'P', 2000), ('', 'D', 1999);
```

```
CREATE VIEW sw_jedynki AS SELECT * FROM skarbonka WHERE moneta = 'J';
```

```
CREATE VIEW sw_dwojki AS SELECT * FROM skarbonka WHERE moneta = 'D' WITH CHECK OPTION;
```

Zapisz, co się stanie podczas próby wykonania każdego z przedstawionych poniżej polecen INSERT, DELETE oraz UPDATE. Na samym końcu ćwiczenia narysuj wynikową zawartość tabeli skarbonka.

Wykonując dalszą część ćwiczenia, spróbuj się domyślić, jakie znaczenie mają te słowa.

```
INSERT INTO sw_jedynki VALUES ('', 'J', 1999);
```

To polecenie zostanie prawidłowo wykonane.

```
INSERT INTO sw_jedynki VALUES ('', 'D', 1997);
```

To polecenie zapisze nowy wiersz w tabeli, choć ze względu na klauzulę WHERE mógłbyś przypuszczać, że nie będzie to możliwe.

```
INSERT INTO sw_dwojki VALUES ('', 'J', 2005);
```

Próba wykonania tego polecenia skończy się zgłoszeniem błędu, a wszystko ze względu na klauzulę CHECK OPTION zastosowaną podczas tworzenia widoku. Powoduje ona, że zanim informacje zapisywane w widoku będą mogły trafić do tabeli, zostaną sprawdzone przy użyciu warunku WHERE.

```
DELETE FROM sw_jedynki WHERE moneta = 'P' OR moneta = 'D';
```

To polecenie nie ma żadnego wpływu na zawartość tabeli, gdyż widok sw_jedynki operuje wyłącznie na monetach spełniających warunek moneta = 'J'.

```
UPDATE sw_jedynki SET moneta = 'J' WHERE moneta = 'P';
```

Także to polecenie nie będzie miało wpływu na zawartość tabeli, gdyż widok sw_jedynki nie zwraca żadnych monet spełniających warunek moneta = 'P'.

Ostatecznie tabela będzie wyglądać następująco:

id	moneta	rok_wybicia
1	J	2003
2	P	2001
3	P	1995
3	D	2000
4	J	1998
5	D	2001
6	P	2007
7	J	2006
8	J	1997
9	P	2000
10	D	1999
11	J	1999
12	D	1997

Widoki z klauzulą CHECK OPTION

Dodanie do widoku klauzuli **CHECK OPTION** informuje system zarządzania relacyjnymi bazami danych, że należy sprawdzać, czy wykonywane polecenia **INSERT** oraz **UPDATE** są zgodne z warunkiem **WHERE** podanym podczas tworzenia widoku. A zatem w jaki sposób klauzula **CHECK OPTION** może wpływać na wykonywane polecenia **INSERT** i **UPDATE**?

Kiedy w poprzednim przykładzie zastosowałeś klauzulę **CHECK OPTION**, to informacje użyte w poleceniu **INSERT** były odrzucane, jeśli nie spełniały warunku podanego podczas tworzenia widoku **sw_dwojki**. Analogiczny błąd zostanie zgłoszony podczas próby wykonania polecenia **UPDATE**:

```
UPDATE sw_dwojki SET momenta = 'x';
```

Wartość '**x**', którą próbowaliśmy zapisać w tabeli, nie spełniała warunku **WHERE** widoku, zatem w efekcie tabela nie została zaktualizowana.

**Klauzula
CHECK OPTION
sprawdza, czy
wykonywane
polecenia INSERT
i UPDATE spełniają
warunek WHERE
użyty podczas
tworzenia widoku.**

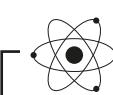


A czy w przypadku korzystania z bazy MySQL nie można by zastosować widoku z klauzulą CHECK OPTION do „zasymulowania” ograniczenia CHECK CONSTRAINT?

Owszem, można. W takim przypadku widoki zapewnią dostęp do całej zawartości tabeli, a jednocześnie baza będzie sprawdzać, czy dane używane w poleceniach INSERT spełniają warunek podany w klauzuli WHERE.

Na przykład stosując tę sztuczkę, moglibyśmy rozwiązać problem dotyczący płci, który pojawił się na początku tego rozdziału. Wystarczyłoby utworzyć odpowiedni widok, którego Kuba używałby podczas modyfikowania tabeli **moje_kontakty**. Widok zgłaszałby błąd, gdyby Kuba spróbował zapisać w tabeli płeć o wartości 'X'.

W bazie MySQL można symulować ograniczenie **CHECK CONSTRAINT**, korzystając z widoku, w którym zastosowano klauzulę **CHECK OPTION**.



WYSIL
SZARE KOMÓRKI

W jaki sposób moglibyśmy utworzyć widok operujący na tabeli **moje_kontakty**, który wymusiłby na Kubie zapisywanie w kolumnie płci wartości 'K' lub 'M'?

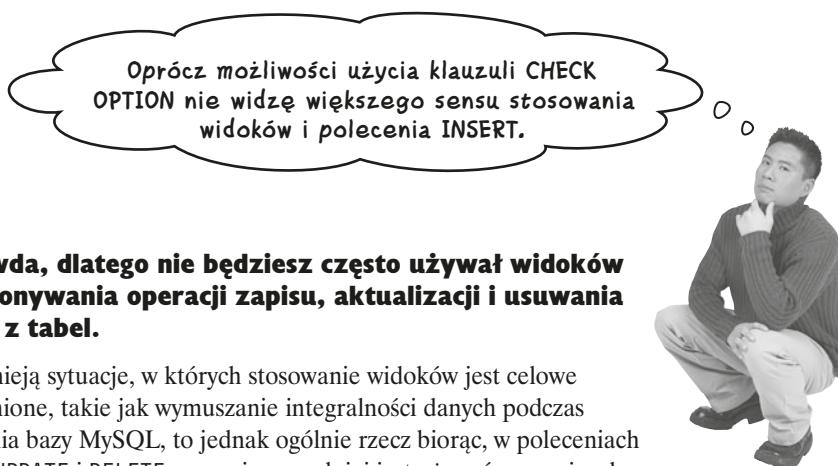
Twój widok może pozwalać na aktualizację danych, jeśli...

W naszym poprzednim przykładzie tabeli skarbonka oba utworzone widoki **pozwalały na aktualizację danych**. Zgodnie z tym, co sugeruje to określenie, takie widoki pozwalają na zmianę danych zapisanych w tabelach bazy danych, na jakich widoki te operują. Najważniejszą cechą takich widoków jest to, że muszą zawierać wszystkie kolumny, w których nie można zapisywać wartości NULL, z tabel, na jakich operują. Dzięki temu, używając takiego widoku w poleceniu INSERT, będziesz mieć pewność, że zapiszesz wartość we wszystkich kolumnach tabel, w których zapisanie wartości jest wymagane.

Ogólnie rzeczą biorąc, oznacza to, że widoki mogą być używane w poleceniach INSERT, UPDATE i DELETE. Jeśli tylko widok zwraca wszystkie kolumny, które nie mogą być puste, to będzie można go użyć do zapisywania odpowiednich wartości w tabeli.

Jednak istnieją także widoki, które **nie dają możliwości aktualizacji tabel**. Są to takie widoki, które nie zwracają wszystkich kolumn zdefiniowanych jako NOT NULL z tabel, na jakich operują. Oprócz utworzenia i usunięcia, takich widoków można używać jedynie w poleceniach SELECT.

Widok pozwalający na aktualizację danych zawiera wszystkie kolumny zdefiniowane jako NOT NULL z tabel, na jakich widok ten operuje.



To prawda, dlatego nie będziesz często używał widoków do wykonywania operacji zapisu, aktualizacji i usuwania danych z tabel.

Choć istnieją sytuacje, w których stosowanie widoków jest celowe i uzasadnione, takie jak wymuszanie integralności danych podczas stosowania bazy MySQL, to jednak ogólnie rzeczą biorąc, w poleceniach INSERT, UPDATE i DELETE znacznie wygodniej jest używać zwykłych tabel niż widoków. Zastosowanie widoku w poleceniu INSERT może być wygodne, jeśli widok zapewnia dostęp tylko do jednej kolumny tabeli, a w pozostałych kolumnach zostaną zapisane wartości domyślne lub wartości NULL. W takich przypadkach zastosowanie widoku w poleceniu INSERT może mieć sens. Oprócz tego w razie korzystania z bazy MySQL można dodać do widoku klauzulę WHERE, symulując w ten sposób ograniczenie CHECK.

Aby dodatkowo skomplikować całe zagadnienie, do aktualizacji zawartości tabel nie można używać widoków, w których zostały zastosowane funkcje agregujące, takie jak SUM, COUNT lub AVG, oraz operatory BETWEEN, HAVING, IN i NOT IN.

Kiedy widok przestanie być potrzebny

Kiedy widok przestanie Ci być już potrzebny, możesz go usunąć, używając polecenia `DROP VIEW`. Polecenie to jest wyjątkowo proste:

```
DROP VIEW sw_dwojki;
```

Nie istnieją
grupie pytania

P: Czy można w jakiś sposób sprawdzić, jakie widoki są dostępne?

O: Widoki są prezentowane tak, jakby były tabelami. Można zatem wykonać polecenie `SHOW TABLES`, by wyświetlić listę wszystkich zdefiniowanych tabel i widoków. I podobnie jak w przypadku tabel, istnieje możliwość zastosowania polecenia `DESC` w celu wyświetlenia szczegółowych informacji o strukturze widoku.

P: Co się stanie, jeśli usunę tabelę, która jest używana w jakimś widoku?

O: To zależy. Niektóre systemy zarządzania relacyjnymi bazami danych wciąż pozwalają na używanie takiego widoku, choć po usunięciu tabeli nie będzie on zwracał żadnych wyników. MySQL nie pozwala na usunięcie widoku, jeśli istnieje tabela, na której on operuje, choć **można** usunąć tabelę używaną w widoku. Inne systemy zarządzania relacyjnymi bazami danych mogą działać jeszcze inaczej. Warto zatem przeprowadzić kilka eksperymentów z używaną bazą danych, by zobaczyć, jak ona się zachowuje. Jednak ogólnie rzecz biorąc, najlepiej jest usuwać widok przed usunięciem tabeli, na której on operuje.

P: Rozumiem, jak bardzo przydatne jest zastosowanie widoków i klauzuli `CHECK` w sytuacjach, gdy bazy danych używa więcej niż jedna osoba. Co się jednak dzieje, kiedy dwie osoby jednocześnie próbują zmienić wartość tej samej kolumny?

O: Aby odpowiedzieć na to pytanie, musimy przedstawić transakcje. Jednak zanim będziemy mogli to zrobić, pani Marudzińska musi wyciągnąć z bankomatu nieco gotówki.

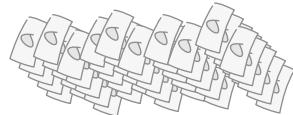
Ograniczenie `CHECK` oraz widoki pozwalają zachować kontrolę nad zawartością bazy danych w przypadkach, gdy korzysta z niej wiele osób.

Kiedy dobrej bazie przydarzy się coś złego

Pani Marudzińska chce przelać 1000 walutek ze swego rachunku rozliczeniowego na rachunek oszczędnościowy. Udaże się zatem do najbliższego bankomatu...

Pani Marudzińska sprawdza stan obu swoich rachunków: rozliczeniowego i oszczędnościowego.

**1000 WALUTEK
NA RACHUNKU
ROZLICZENIOWYM** **30 WALUTEK
NA RACHUNKU
OSZCZĘDNOŚCIOWYM**

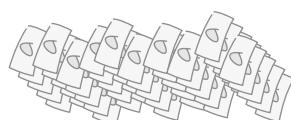


Następnie pani Marudzińska wybiera operację:

**PRZELEJ 1000 WALUTEK
Z RACHUNKU ROZLICZENIOWEGO NA
OSZCZĘDNOŚCIOWY**

I naciska odpowiedni przycisk na klawiaturze bankomatu.

**RACHUNEK
ROZLICZENIOWY** **RACHUNEK
OSZCZĘDNOŚCIOWY**



Po pewnym czasie bankomat ponownie się włącza.

Pani Marudzińska sprawdza stan swoich rachunków.

**0 WALUTEK
NA RACHUNKU
ROZLICZENIOWYM** **30 WALUTEK
NA RACHUNKU
OSZCZĘDNOŚCIOWYM**



Bankomat wydaje głośny dźwięk, po czym gaśnie.

Najwyraźniej nastąpiła awaria zasilania.

Gdzież, ach, gdzie podziały się walutki pani Marudzińskiej?

Co się stało w bankomacie



W tym miejscu
wysiadło zasilanie.

Bankomat: TRRALALALA.

Bankomat: HEJ. TO ŻE TO PANI WIESŁAWA MARUDZIŃSKA.
WITAM PANIĄ [NUMER RACHUNKU 38221]

Pani Marudzińska: Powiedz mi, ile mam pieniędzy na rachunkach.

Bankomat: (myśli...) [SELECT DOSTEPNE_SRODKI FROM RACHUNEK_ROZLICZENIOWY WHERE ID_KONTA = 38221;
SELECT DOSTEPNE_SRODKI FROM RACHUNEK_OSZCZEDNOSCIOVY WHERE ID_KONTA = 38221;]
WYGLĄDA TO TAK: 1000 NA RACHUNKU ROZLICZENIOWYM
I 30 NA RACHUNKU OSZCZĘDNOŚCIOWYM

Pani Marudzińska: No to przelej 1000 walutek z rachunku rozliczeniowego na oszczędnościowy.

Bankomat: TO SPORA KWOTA. ALE MIECH BĘDZIE. ZATEM TAK:
[STAN RACHUNKU ROZLICZENIOWEGO > 1000. ZATEM Klientka ma dosyć walutek.]
[ODEJMIJ 1000 Z RACHUNEK_ROZLICZENIOWY]
[INSERT PIHHHIP.....]

Bankomat:

Bankomat:

Bankomat: HRRR..... Z Z Z Z Z Z Z Z Z

Bankomat: RRR...

Bankomat: HEJ. TO ŻE TO PANI WIESŁAWA MARUDZIŃSKA.
WITAM PANIĄ [NUMER RACHUNKU 38221]

Pani Marudzińska: Powiedz mi, ile mam pieniędzy.

Bankomat: (myśli...) [SELECT DOSTEPNE_SRODKI FROM RACHUNEK_ROZLICZENIOWY WHERE ID_KONTA = 38221;
SELECT DOSTEPNE_SRODKI FROM RACHUNEK_OSZCZEDNOSCIOVY WHERE ID_KONTA = 38221;]
WYGLĄDA TO TAK: 0 NA RACHUNKU ROZLICZENIOWYM
I 30 NA RACHUNKU OSZCZĘDNOŚCIOWYM

Bankomat: RUUU... TO, CO OKŁADA PANI PIĘŚCIAMI, TO MÓJ MONITOR. ŻEGNAM PANIĘ. PANI WIESŁAWO MARUDZIŃSKA.



WYSIL
SZARE KOMÓRKI

W jaki sposób możemy uchronić się przed tym, że bankomat zapomni zapisać pieniędzy na rachunek oszczędnościowy pani Marudzińskiej?

W międzyczasie w innym miejscu miasta...

Kolejne kłopoty z bankomatom

Janek i Marysia mają wspólne konto. W piątek każde z nich postanowiło wyciągnąć z bankomatu 300 walutek i oczywiście nieszczęśliwym zbiegiem okoliczności zrobili to w tym samym czasie, używając dwóch różnych bankomatów.



Bankomat: OCH, TOŻ TO JANEK. CZY TY STARY MYŚLISZ, ŻE JA SIEDZĘ NA PIENIĄDZACH?

Janek: Jaki jest stan mojego konta?

Bankomat: (myślisz...) [SELECT DOSTEPNE_SRODKI FROM KONTA:]
350 WALUTEK

Janek: No to daj mi 300 walutek.

Bankomat: UWAŻASZ, ŻE TYLKO DO TEGO SIĘ NADAJĘ. WYDARUĆ PIENIĄDZE. TYLKO WYKORZYSTAĆ I ZOSTAWIĆ.

[DOSTEPNE_SRODKI > 300.
MA WYSTARCZAJĄCO DUŻO ŚRODKÓW] **350 WALUTEK**

[USUN 300 Z RACHUNEK_ROZLICZENIOWY] **350 WALUTEK**
[ODEJMIAJ 300 OD DOSTEPNE_SRODKI
NA RACHUNEK_ROZLICZENIOWY] **50 WALUTEK**

Janek bierze walutki i odchodzi.

Bankomat: NIGDY NIE DZWONISZ.
NIGDY NIE PISZESZ. CZEŚĆ JANKU.

To właśnie tu
wystąpiły problemy

Bankomat: CZEŚĆ MARYSIU.

Marysia: Jaki jest stan mojego konta?

Bankomat: [SELECT DOSTEPNE_SRODKI
FROM KONTA:]
350 WALUTEK

Piiip, piiip

Marysia czeka, zabijając czas szukaniem w torebce swojego telefonu komórkowego.

Marysia: Chcę wyciągnąć 300 walutek.

Bankomat: SIĘ ROBI. PSZE PRAMI
[DOSTEPNE_SRODKI > 300. MA
WYSTARCZAJĄCO DUŻO ŚRODKÓW]
[USUN 300 Z RACHUNEK_
ROZLICZENIOWY]

[ODEJMIAJ 300 OD DOSTEPNE_SRODKI
NA RACHUNEK_ROZLICZENIOWY]

Bankomat: MASZ PASKUDNY DEBET NA
KONCIE.



To nie marzenia, to transakcje

Transakcja jest grupą poleceń SQL, które realizują pewne zadanie.

W przypadku sprawy pani Marudzińskiej transakcja składałaby się z poleceń SQL potrzebnych do przeniesienia pewnych środków z jej rachunku rozliczeniowego na rachunek oszczędnościowy.

Te trzy czynności składają się na jedno zadanie.
A zatem to nasza transakcja.

Jeśli stan rachunku rozliczeniowego ≥ 1000
Odejmij 1000 od rachunku rozliczeniowego
Dodaj 1000 do rachunku oszczędnościowego

Janek i Marysia próbowali wykonać *tą samą transakcję* w tym samym czasie:

Janek i Marysia próbują w tej samej chwili wyciągnąć ze swojego wspólnego konta 300 walutek.

Jeśli stan rachunku rozliczeniowego ≥ 300
Odejmij 300 od rachunku rozliczeniowego
Wydaj 300 walutek

Jeśli stan rachunku rozliczeniowego ≥ 300
Odejmij 300 od rachunku rozliczeniowego
Wydaj 300 walutek

Transakcja Janka
wykonywana w bankomacie
Lewego Banku.

Transakcja Marysi
wykonywana w bankomacie
Powszechnego Banku
Bazodanowego.



W przypadku Janka i Marysi bankomat Powszechnego Banku Bazodanowego nie powinien mieć prawa do wykonania jakichkolwiek operacji na koncie, nawet do sprawdzenia jego stanu, aż do momentu zakończenia transakcji przez bankomat Lewego Banku.

Jeśli nie uda się prawidłowo wykonać wszystkich zaplanowanych czynności danej transakcji, to nie powinna zostać wykonana żadna z nich.

Klasyczny test ACID

Warto, byś zapamiętał skrót **ACID** — pomoże Ci on określić, jakie czynności wykonywane przez SQL można uznać za transakcję. Otóż istnieją cztery warunki, które muszą zostać spełnione, zanim grupę poleceń SQL będzie można uznać za transakcję. Oto one:



ACID: ATOMICITY (NIEPODZIELNOŚĆ)

Wszystkie elementy wchodzące w skład transakcji muszą zostać wykonane albo w przeciwnym razie nie zostanie wykonany żaden z nich. Nie można wykonać tylko części transakcji. Walutki pani Marudzińskiej rozpływły się w nicosć w efekcie awarii zasilania tylko i wyłącznie dlatego, że została wykonana jedynie część transakcji.



ACID: CONSISTENCY (SPÓJNOŚĆ)

Po zakończeniu transakcji baza danych jest spójna. W obu naszych przykładach z bankomatami po zakończeniu transakcji stan kont będzie prawidłowy. W pierwszym przypadku pieniądze zostaną przelane z rachunku rozliczeniowego na oszczędnościowy, a w drugim — wypłacone z bankomatu.



ACID: ISOLATION (IZOLACJA)

Izolacja oznacza, że każda transakcja zachowuje spójny widok bazy danych, niezależnie od innych transakcji, które są wykonywane w tym samym czasie. To właśnie z tym zagadniением były związane problemy, które wystąpiły w przykładzie z Jankiem i Marysią. Bankomat Marysi był w stanie uzyskać dostęp do informacji o stanie konta, gdy bankomat Janka realizował już transakcję. Marysia nie powinna móc zobaczyć informacji o stanie konta bądź powinna zobaczyć komunikat typu „Proszę czekać na zakończenie transakcji”.



ACID: DURABILITY (ODPORNOŚĆ)

Po zakończeniu transakcji baza danych powinna prawidłowo zapisać dane i chronić je przed zgubnymi skutkami zaników napięcia oraz wszelkimi innymi zagrożeniami. Zazwyczaj cel ten osiąga się poprzez zapisywanie informacji o transakcji w wielu różnych miejscach, a nie tylko w głównej bazie danych. Gdyby informacje o transakcji pani Marudzińskiej zostały zapisane w jakimś innym miejscu, to jej 1000 walutek nie przepadłyby bezpowrotnie.

SQL pomaga nam zarządzać swoimi transakcjami

Wyobraźmy sobie bardzo prostą bankową bazę danych. Składać się ona będzie z tabeli posiadaczy kont oraz z tabel przechowujących stan rachunków oszczędnościowych i rozliczeniowych:

Tu prawdopodobnie znajduje się znacznie więcej kolumn, niemniej jednak wiadomo, o co chodzi.

Mamy do dyspozycji trzy narzędzia związane z zarządzaniem transakcjami, które pomagają zadbać o nasze bezpieczeństwo:

START TRANSACTION;

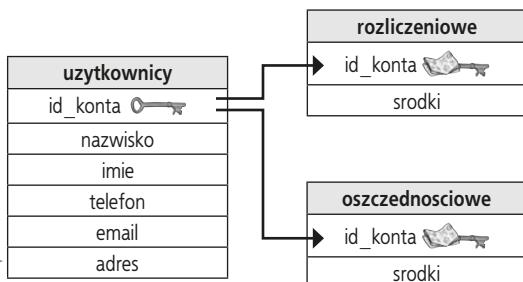
To polecenie rozpoczęta rejestrację wykonywanych poleceń SQL.

Polecenie **START TRANSACTION** powoduje rejestrowanie wykonywanych poleceń SQL aż do momentu wydania polecenia COMMIT lub ROLLBACK.

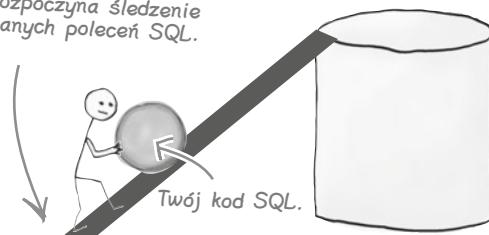
COMMIT;

Kiedy uznamy, że wszystko jest w porządku, tym poleceniem możemy zatwierdzić efekty wykonania naszych poleceń SQL.

Jeśli wszystkie planowane polecenia zostały wykonane i wydaje się, że wszystko jest w porządku, wystarczy wydać polecenie **COMMIT**, by zmiany zostały na trwałe zapisane w bazie.



W tym miejscu używany system zarządzania bazami danych rozpoczęta śledzenie wykonywanych poleceń SQL.

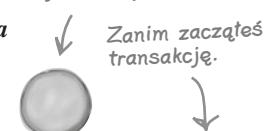


ROLLBACK;

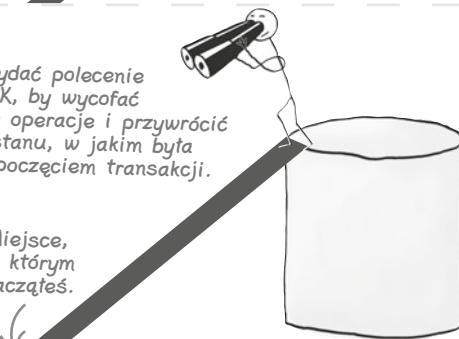
To polecenie przenosi Cię z powrotem na sam początek transakcji.

Jeśli coś pójde nie tak jak powinno, to polecenie **ROLLBACK** wycofuje wszystkie polecenia i przywraca bazę do stanu, w jakim była *przed* wydaniem polecenia START TRANSACTION.

Twój kod SQL.



... bądź wydać polecenie ROLLBACK, by wycofać wszystkie operacje i przywrócić bazę do stanu, w jakim była przed rozpoczęciem transakcji.



Do momentu wykonania polecenia COMMIT w bazie danych nie zostaną wprowadzone żadne modyfikacje.

Co powinno sie stac w bankomacie



Bankomat: TRALARLALA.

Bankomat: HEJ. TO ŻE TO PANI WIESŁAWA MARUDZIŃSKA.
WITAM PANIĘ [NUMER RACHUNKU 38221]

Pani Marudzińska: Powiedz mi, ile mam pieniedzy na rachunkach.

Bankomat: (myśli...) [SELECT DOSTEPNE_SRODKI FROM RACHUNEK_ROZLICZENIOWY WHERE ID_KONTA = 38221;

SELECT DOSTEPNE_SRODKI FROM RACHUNEK_OSZCZEDNOSCIOVY WHERE ID_KONTA = 38221:]

WYGLĄDA TO TAK: 1000 NA RACHUNKU ROZLICZENIOWYM
I 30 NA RACHUNKU OSZCZĘDNOŚCIOWYM

Pani Marudzińska: No to przelej 1000 walutek z rachunku rozliczeniowego na oszczednościowy.

Bankomat: TO SPORA KWOTA. ALE NIECH BĘDZIE. ZATEM TAK:

[ISTAN RACHUNKU ROZLICZENIOWEGO > 1000. ZATEM Klientka ma dosyć walutek.]

[ODEJMIIJ 1000 Z RACHUNEK_ROZLICZENIOWY]

[INSERT PIHHHIP.....]

BANKOMAT NA ZASILANIU AWARYJNYM: ROLLBACK:

Bankomat:

Bankomat: HRRR..... Z Z Z Z Z Z Z Z Z Z

Bankomat: AAA...

Bankomat: HEJ. TO ŻE TO PANI WIESŁAWA MARUDZIŃSKA.
WITAM PANIĘ [NUMER RACHUNKU 38221]

Pani Marudzińska: Powiedz mi, ile mam pieniedzy.

Bankomat: (myśli...) [SELECT DOSTEPNE_SRODKI FROM RACHUNEK_ROZLICZENIOWY WHERE ID_KONTA = 38221;

SELECT DOSTEPNE_SRODKI FROM RACHUNEK_OSZCZEDNOSCIOVY WHERE ID_KONTA = 38221:]

WYGLĄDA TO TAK: 1000 NA RACHUNKU ROZLICZENIOWYM
I 30 NA RACHUNKU OSZCZĘDNOŚCIOWYM

Dzięki poleceniu ROLLBACK
polecenie COMMIT nigdy nie
zostało wykonane, a zatem
zawartość bazy nie uległa
zmianie.

W tym miejscu
wysiadło zasilanie.



Jak umożliwić korzystanie z transakcji w MySQL-u

Zanim będziesz mógł używać transakcji w bazach MySQL, będziesz musiał zastosować odpowiedni **mechanizm składowania**. Jest to niewidoczny dla użytkowników bazy danych mechanizm, który odpowiada za przechowywanie wszystkich danych zapisywanych w bazie oraz jej struktur. Niektóre z typów mechanizmów składowania pozwalają na stosowanie transakcji, a inne nie.

Cofnij się do rozdziału 4., w którym to poznaleś polecenie:

```
SHOW CREATE TABLE moje_kontakty;
```

W tym przypadku obchodzi nas używany mechanizm składowania.

Projektowanie dobrych tabel

Polecenie oszczędzające czas

Rzuć okiem na kod polecenia, którego użyliśmy do utworzenia tabeli, przedstawiony na stronie 217. Następnie porównaj go z zamieszczonym poniżej kodem zwrotnym przez polecenie SHOW CREATE TABLE moje_kontakty. Nie są one identyczne, jednak gdybyś wykonał poniższe polecenie CREATE TABLE, to uzyskane wyniki byłyby takie same, jak w przypadku użycia oryginalnego polecenia. Nie musisz usuwać znaków lewego apostrofa ani ustawień dotyczących wartości domyślnych, jeśli jednak to zrobisz, to polecenie będzie bardziej przejrzyste i schludne.

Pomiędzy znakami, określonymi jako lewy apostrof, są zapisywane nowy kolumna oraz nazwa tabeli. Znaki lewego apostrofa są używane w wynikach generowanych przez polecenie SHOW CREATE TABLE.

```
CREATE TABLE `moje_kontakty` (
  `nazwisko` varchar(30) default NULL,
  `imie` varchar(20) default NULL,
  `email` varchar(50) default NULL,
  `plec` char(1) default NULL,
  `data_urodzenia` date default NULL,
  `zawod` varchar(50) default NULL '',
  `lokalizacja` varchar(50) default NULL '',
  `stan` varchar(20) default NULL '',
  `zainteresowania` varchar(100) default NULL '',
  `szuka` varchar(100) default NULL ''
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Jeśli jawnie nie zazadamy inaczej, to system zarządzania bazą danych przyjmuje, że wartości wszyscy kolumny przypisują domyślne wartości NULL.

Podczas tworzenia tabeli warto określić, czy w jej poszczególnych kolumnach mogą być zapisywane wartości NULL, czy też nie.

Nie musisz zwracać uwagi na ostatni wiersz kodu, umieszczony za nawiasem zamykającym. Określa on sposób przechowywania danych oraz używany zbiór znaków. Jak na razie w zupełności wystarczą nam ustawienia domyślne.

Choć można by poprawić przejrzystość kodu (usuwając jego ostatni wiersz oraz wszystkie znaki lewego apostrofa), to jednak by utworzyć tabelę, nie trzeba wprowadzać w nim żadnych modyfikacji — wystarczy go skopiować i wkleić w przedstawionej postaci.

[jesteś tutaj ▶ 219](#)

Musisz upewnić się, że używanym mechanizmem składowania jest BDB lub InnoDB — jedynie dwa, które w MySQL-u zapewniają możliwość stosowania transakcji.



InnoDB oraz BDB to dwa spośród kilku sposobów służących systemowi zarządzania relacyjnymi bazami danych do przechowywania danych zapisywanych w bazie.

Jak na razie dla Twoich potrzeb nie ma znaczenia, który z tych dwóch mechanizmów składowania zastosujesz. Poniżej przedstawiliśmy postać polecenia, które pozwala zmienić aktualnie używany mechanizm składowania w wybranej tabeli:

Sposoby te są nazywane mechanizmami składowania, a zastosowanie któregoś z dwóch wymienionych — InnoDB lub BDB — pozwoli Ci korzystać z transakcji. Więcej informacji na temat różnic pomiędzy mechanizmami składowania dostępnymi w MySQL-u możesz znaleźć w jego dokumentacji.

```
ALTER TABLE tabela TYPE = InnoDB;
```

Wypróbuj transakcje samodzielnie

Załóżmy, że zamieniłeś wszystkie złotówki w swojej skarbonece na monety dwuzłotowe.

Wypróbuj poniższy kod na tabeli skarbonka, którą utworzyliśmy we wcześniejszej części rozdziału.

W pierwszym przykładzie wycofujemy zmiany poleceniem ROLLBACK, gdyż uznamy, że jednak nie chcemy niczego zmieniać.

```
START TRANSACTION;
SELECT * FROM skarbonka;
UPDATE skarbonka SET moneta = 'D' WHERE moneta = 'J';
SELECT * FROM skarbonka; ← Teraz zmiany są widoczne.
ROLLBACK; ← Jednak zmieniamy zdanie...
SELECT * FROM skarbonka; ← ... i teraz wszystko
                           jest po staremu.
```

Następnym razem zatwierdzimy zmiany, gdyż uznamy, że są one celowe:

```
START TRANSACTION;
SELECT * FROM skarbonka;
UPDATE skarbonka SET moneta = 'D' WHERE moneta = 'J';
SELECT * FROM skarbonka; ← Teraz zmiany są widoczne.
COMMIT; ← Teraz zatwierdzamy zmiany...
SELECT * FROM skarbonka; ← ... zatem wciąż
                           są widoczne.
```

Zaostrz ołówek



Zaostrz ołówek

Określ zawartość tabeli skarbonka po wykonaniu każdej z poniższych transakcji. A oto początkowa postać tabeli.

skarbonka

id	moneta	rok_wybicia
1	D	1995
2	P	2000
3	J	2005
4	D	1999

START TRANSACTION;

UPDATE skarbonka SET moneta = 'D' WHERE moneta = 'P'

AND rok_wybicia < 1975;

COMMIT;

id	moneta	rok_wybicia
1		
2		
3		
4		

START TRANSACTION;

UPDATE skarbonka SET moneta = 'J' WHERE moneta = 'D';

ROLLBACK;

id	moneta	rok_wybicia
1		
2		
3		
4		

START TRANSACTION;

UPDATE skarbonka SET moneta = 'D' WHERE moneta = 'P'

AND rok_wybicia > 2000;

ROLLBACK;

id	moneta	rok_wybicia
1		
2		
3		
4		

START TRANSACTION;

UPDATE skarbonka SET moneta = 'P' WHERE moneta = 'D'

AND rok_wybicia > 1998;

COMMIT;

id	moneta	rok_wybicia
1		
2		
3		
4		

START TRANSACTION;

UPDATE skarbonka SET moneta = 'P' WHERE moneta = 'J'

AND rok_wybicia > 1970;

COMMIT;

id	moneta	rok_wybicia
1		
2		
3		
4		

→ Odpowiedzi znajdziesz na stronie 520

Nie istniejąca grupa pytań

P: Czy można zacząć transakcję bez stosowania polecenia START TRANSACTION, czy też polecenia COMMIT i ROLLBACK nie mogą bez niego działać?

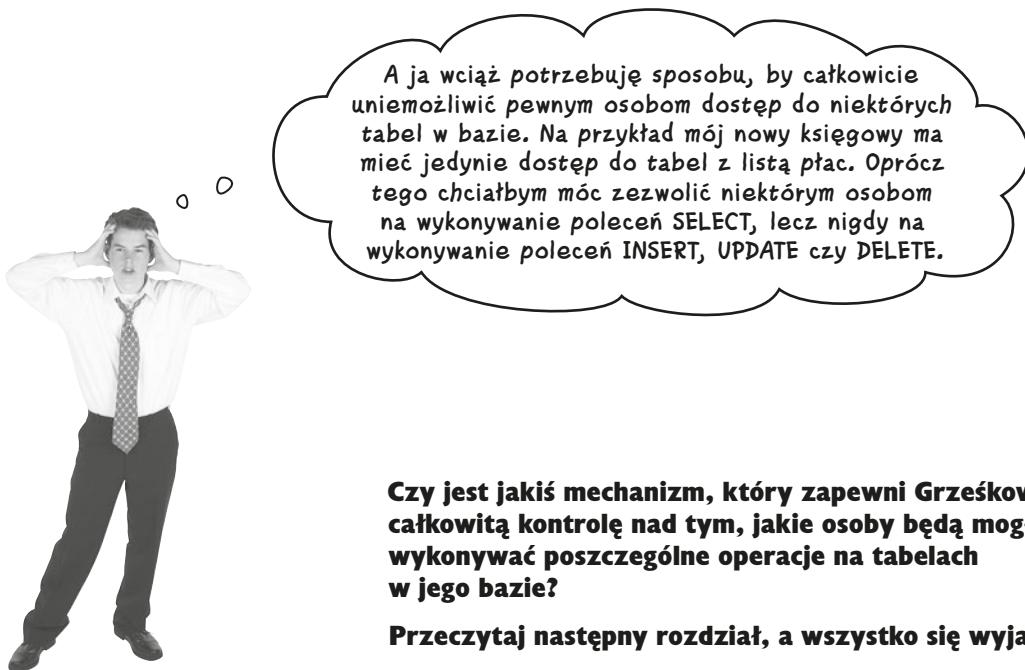
O: Musisz poinformować system zarządzania bazą danych, kiedy chcesz rozpoczęć transakcję; właśnie do tego służy polecenie START TRANSACTION. System zarządzania bazą danych musi znać moment rozpoczęcia transakcji, by wiedzieć, jak daleko cofnąć się w razie jej ewentualnego wycofania.

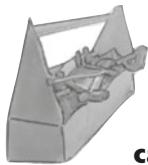
P: Czy mogę rozpoczęć transakcję, by wypróbować działanie pisanych poleceń SQL?

O: Nie tylko możesz, ale nawet powinieneś tak robić. To wspaniały sposób na ćwiczenie tworzenia zapytań modyfikujących dane w tabelach bez wprowadzania trwałych zmian w bazie w przypadku, gdyby coś poszło nie tak, jak zaplanowaliśmy. Pamiętaj tylko, by pod koniec testów wykonać polecenie COMMIT lub ROLLBACK.

P: Dlaczego mam sobie zawracać głowę poleceniami COMMIT i ROLLBACK?

O: Po rozpoczęciu transakcji system zarządzania bazą danych rejestruje wszystkie wykonywane czynności. Informacje na ich temat są trzymane w tak zwanym dzienniku transakcji, a im więcej operacji wykonujesz w ramach transakcji, tym ten dziennik staje się większy. Optymalnym rozwiązaniem jest stosowanie transakcji wyłącznie w tych sytuacjach, gdy naprawdę musimy mieć możliwość późniejszego odtworzenia zmian, gdyż nie chcemy marnować niepotrzebnie miejsca i zmuszać bazy danych do pracy cięższej niż to konieczne.





Przybornik SQL

A zatem przeczytałeś rozdział jedenasty, a tym samym niemal całkowicie wypełniłeś swój SQL-owy przybornik. Dowiedziałeś się, w jaki sposób tworzyć WIDOKI i jak stosować TRANSAKCJE. Kompletną listę porad znajdziesz w dodatku C, zamieszczonym na końcu książki.

TRANSAKCJE

To grupa poleceń, które muszą zostać wykonane jako jedna całość. Jeśli nie uda się ich wszystkich wykonać prawidłowo, to nie zostanie wykonane żadne z nich.

Polecenie **START TRANSACTION** jest używane, by poinformować system zarządzania bazą danych o rozpoczęciu transakcji. Żadne modyfikacje bazy wprowadzane od tego czasu nie będą trwate, aż do momentu wykonania polecenia **COMMIT**. Transakcja jest kontynuowana do momentu jej zatwierdzenia poleceniem **COMMIT** lub wycofania poleceniem **ROLLBACK**, które przywraca bazę danych do stanu, w jakim była przed wykonaniem polecenia **START TRANSACTION**.

WIDOKI

Widoków można używać, by potraktować wyniki wykonania zapytania jako tabelę. Doskonale się one nadają, by przekształcić bardzo złożone zapytania w bardzo proste.

WIDOKI POZWALAJĄCE NA AKTUALIZACJĘ

Jak można się domyślić, są to widoki, które pozwalały na modyfikację danych zapisanych w tabelach. Muszą one zawierać wszystkie kolumny zdefiniowane jako NOT NULL z tabeli lub tabel, na jakich widok operuje.

WIDOKI NIEPOZWALAJĄCE NA AKTUALIZACJĘ

Są to widoki, których nie można używać w poleceniach **INSERT** lub **UPDATE**.

OGRANICZENIA SPRAWDZAJĄCE

Ograniczenia sprawdzające mogą być używane do określania, jakie wartości mogą być zapisywane w poszczególnych kolumnach tabeli.

CHECK OPTION

Tej klauzuli można użyć podczas tworzenia widoku pozwalającego na aktualizację danych, by wymusić sprawdzanie warunku **WHERE** użytego w widoku przed wykonaniem polecień **INSERT** lub **UPDATE**. Jeśli warunek ten nie będzie spełniony, baza nie zostanie w żaden sposób zmodyfikowana.

Zaostrz ołówek



Rozwiążanie ze str. 499

Zakładając, że Franek wykona polecenie SELECT przedstawione na stronie 498, używając przy tym widoku wzrost_plac, to w jaki sposób może posortować wyniki alfabetycznie według nazwiska?

Należy dodać klauzulę ORDER BY nazwisko bądź to do widoku podczas jego tworzenia, bądź później do polecenia SELECT, w którym widok zostanie użyty.

Rozwiążanie ćwiczenia

Zaostrz ołówek



Określ zawartość tabeli skarbonka po wykonaniu każdej z poniższych transakcji. A oto początkowa postać tabeli.

skarbonka

id	moneta	rok_wybicia
1	D	1995
2	P	2000
3	J	2005
4	D	1999

START TRANSACTION;

UPDATE skarbonka SET moneta = 'D' WHERE moneta = 'P'

AND rok_wybicia < 1975;

COMMIT;

Żadne rekordy nie spełniają tych kryteriów, więc nikt się nie zmienia.

id	moneta	rok_wybicia
1	D	1995
2	P	2000
3	J	2005
4	D	1999

START TRANSACTION;

UPDATE skarbonka SET moneta = 'J' WHERE moneta = 'D';

ROLLBACK; Wycofanie transakcji, brak zmian.

id	moneta	rok_wybicia
1	D	1995
2	P	2000
3	J	2005
4	D	1999

START TRANSACTION;

UPDATE skarbonka SET moneta = 'D' WHERE moneta = 'P'

AND rok_wybicia > 2000;

ROLLBACK; Wycofanie transakcji, brak zmian.

id	moneta	rok_wybicia
1	D	1995
2	P	2000
3	J	2005
4	D	1999

START TRANSACTION;

UPDATE skarbonka SET moneta = 'P' WHERE moneta = 'D'

AND rok_wybicia > 1998;

COMMIT;

Ten wiersz zostanie zmieniony.

id	moneta	rok_wybicia
1	D	1995
2	P	2000
3	J	2005
4	P	1999

START TRANSACTION;

UPDATE skarbonka SET moneta = 'P' WHERE moneta = 'J'

AND rok_wybicia > 1970;

COMMIT;

Ten wiersz zostanie zmieniony.

id	moneta	rok_wybicia
1	D	1995
2	P	2000
3	P	2005
4	D	1999

12. Bezpieczeństwo

Zabezpieczanie swych dóbr



Włożyłeś niezwykle dużo wysiłku i czasu w utworzenie swojej bazy danych. I na pewno byłbyś zdruzgotany, gdyby przydarzyło się jej coś złego. Jednak musisz zapewnić **dostęp do swoich danych innym osobom** i obawiasz się, że mogłyby zapisać w bazie niewłaściwe dane lub, co gorsza, **usunąć nie te dane, które powinny**. Na szczęście w tym rozdziale dowiesz się, jak można dodatkowo **zabezpieczyć bazę danych** oraz obiekty w niej umieszczone oraz jak uzyskać pełną kontrolę nad tym, **kto i jakie operacje na bazie może wykonywać**.

Problemy użytkowników

Śledzenie kloonów rozwinęło się na taką skalę, iż Rada Miejska Bazodanowa musiała zatrudnić cały zespół osób, które na pełny etat zajmują się śledzeniem kloonów oraz zapisywaniem informacji o ich poczynaniach w bazie danych `kloni_miejscy`.

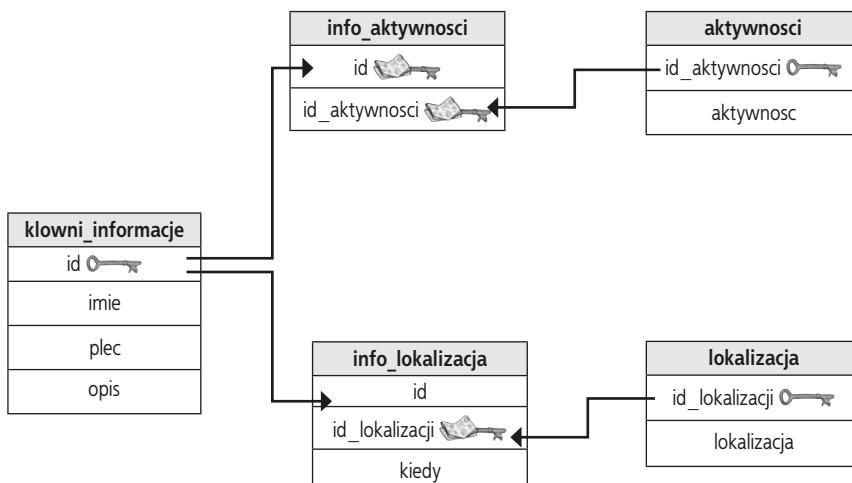
Jednak do zespołu dostał się agent kloonów przebrany w normalne ubranie, który ukrywa się pod pseudonimem „Jurek”. Przysporzył on wielu problemów związanych z bazą kloonów, takich jak pojawienie się w niej nieprawidłowych danych, znikanie ważnych danych i powielanie już istniejących rekordów poprzez celowe popełnianie prostych literówek. Poniżej przedstawiliśmy jedynie kilka spośród wielu problemów, jakie mamy z bazą kloonów:



W tabeli `kloni_informacje` występują rekordy dotyczące kloonów: Smyk, Pani Smyk, Pani Smyck. Mamy jednak pewność, że chodzi o tego samego kloniego, gdyż można się tego domyślić na podstawie jego płci i opisu (oczywiście jeśli pominiemy te nieszczęsne literówki).

Ze względu na te nieprawidłowe i niepotrzebne wpisy w tabeli `kloni_informacje` straszliwy bałagan wkradł się także do tabeli, w której zapisywane są informacje o publicznych wystąpieniach poszczególnych kloonów.

Pełno błędów literowych można także znaleźć w tabeli `aktywnosci`. Pan Smyk jest żonglerem, Pani Smyk żąglerem, a Skuter rząglerem.



Zapobieganie błędom w bazie kloonów

Jurek zniknął, zanim ktokolwiek zauważył, że sabotuje dane, a teraz musimy sami poprawiać wszystkie błędy. Od tej chwili chcemy przydzielać nowo zatrudnionemu pracownikowi prawo do wykonywania poleceń SELECT, tak by mógł identyfikować kloonów. Jednocześnie chcemy, by takie osoby nie miały prawa do wykonywania poleceń INSERT. Jak również poleceń UPDATE. Ani jakichkolwiek innych operacji, przynajmniej do czasu, gdy dokładnie takiego nowego pracownika „prześwietlimy”.

Musimy także być ostrożni, dając takim nowym pracownikom zadanie skorygowania błędów wprowadzonych przez Jurka, co będzie się wiązało z korzystaniem z polecenia DELETE — zawsze bowiem istnieje ryzyko, że taki nowy pracownik niechcący usunie dobre dane.

Nadszedł już czas, by zabezpieczyć bazę danych, zanim inni kлоoni, tacy jak Jurek, całkowicie ją zniszczą.

Zaostrz ołówek



Rozwiążanie

Zabezpiecz bazę danych kloonów przed potencjalnym sabotażem. Poniżej zapisz w dwóch kolumnach przykłady zapytań, które nowi pracownicy będą mogli wykonywać lub nie. Jeśli to możliwe, podaj nazwy tabel, na których zapytania będą operować.

Nowi pracownicy powinni mieć możliwość wykonywania zapytań:

przykład:
SELECT z tabeli aktywnosci

Nowi pracownicy **nie** powinni mieć możliwości wykonywania zapytań:

przykład:
DROP TABLE na tabeli kloni_informacje

Rozwiążanie ćwiczenia

Zaostrz ołówek



Rozwiążanie

Zabezpiecz bazę danych kloonów przed potencjalnym sabotażem. Poniżej zapisz w dwóch kolumnach przykłady zapytań, które nowi pracownicy będą mogli wykonywać lub nie. Jeśli to możliwe, podaj nazwy tabel, na których zapytania będą operować.

Nowi pracownicy powinni mieć możliwość wykonywania zapytań:

przykład:

SELECT z tabeli aktywnosci

**SELECT z tabel klowni_informacje,
info_aktywnosci, aktywnosci,
info_lokalizacja, lokalizacja**

Nowi pracownicy **nie** powinni mieć możliwości wykonywania zapytań:

przykład:

DROP TABLE na tabeli klowni_informacje

**DROP TABLE na tabelach klowni_informacje,
info_aktywnosci, aktywnosci, info_lokalizacja,
lokalizacja**

**INSERT do tabel klowni_informacje,
info_aktywnosci, aktywnosci, info_lokalizacja,
lokalizacja**

**UPDATE w tabelach klowni_informacje,
info_aktywnosci, aktywnosci, info_lokalizacja,
lokalizacja**

**ALTER na tabelach klowni_informacje,
info_aktywnosci, aktywnosci, info_lokalizacja,
lokalizacja**

**DELETE na tabelach klowni_informacje,
info_aktywnosci, aktywnosci, info_lokalizacja,
lokalizacja**

To dobra wiadomość — możemy powstrzymać takich kloonów jak Jurek przed zniszczeniem naszej bazy danych!

SQL zapewnia nam możliwość określenia, jakie operacje będą mogli wykonywać nasi pracownicy na bazie danych `klowni_miejscy`, a jakie informacje będą dla nich niedostępne. Jednak zanim będziemy w stanie to zrobić, musimy utworzyć dla nich oraz dla wszystkich innych osób korzystających z bazy danych kloonów tak zwane **konto użytkownika**.



Zabezpieczanie konta administratora

Jak do tej pory w naszej bazie danych istniał tylko jeden użytkownik, który na dodatek nie miał określonego hasła dostępu. Każdy, kto tylko miał dostęp do terminalu lub dowolnego programu do zarządzania bazą danych, mógł się z nią połączyć i przejąć nad nią całkowitą kontrolę.

Domyślnie pierwszy użytkownik bazy danych — administrator, tak zwany ***root*** — ma całkowitą kontrolę nad całą bazą danych wraz z jej zawartością. To bardzo ważne, gdyż użytkownik ten musi mieć możliwość tworzenia kont dla innych użytkowników. Nie chcemy ograniczać możliwości konta administratora, jednak na pewno będziemy chcieli ustawić jakieś hasło dostępu do tego konta. W MySQL polecenie, które służy do tego celu, jest całkiem proste:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('b4dc10wnz');
```

Nazwa użytkownika administratora to 'root'.

'localhost' określa, że system zarządzania relacyjną bazą danych został zainstalowany i działa na lokalnym komputerze.

A to jest hasło, jakie wybraliśmy dla naszego administratora.

W innych systemach zarządzania bazami danych polecenia realizujące to samo zadanie wyglądają inaczej. Na przykład na serwerze Oracle ma ono następującą postać:

```
alter user root identified by nowe-hasło;
```

Jeśli do zarządzania bazą danych używasz programu z graficznym interfejsem użytkownika, to zapewne będziesz mógł zmienić hasło administratora w znacznie prostszy sposób, korzystając z jakiegoś okna dialogowego. **Najważniejsze w tym przypadku nie jest to, jak zmienisz to hasło, lecz że koniecznie powinieneś je zmienić.**

Zajrzyj też do dokumentacji używanego systemu zarządzania relacyjnymi bazami danych, by znaleźć dodatkowe informacje dotyczące zabezpieczania konta administratora.

Nie, istnieją głupie pytania

P: Wciąż nie do końca rozumiem, co oznacza „localhost”. Czy możesz mi to dokładniej wytłumaczyć?

O: localhost oznacza, że komputer używany do wykonywania poleceń SQL jest tym samym komputerem, na którym zainstalowano system zarządzania relacyjnymi bazami danych. localhost jest domyślną wartością tego parametru, zatem określanie go jest opcjonalne.

P: A jeśli używane przeze mnie oprogramowanie do wykonywania poleceń SQL jest uruchamiane na innym komputerze niż serwer bazy danych?

O: W takim przypadku mamy do czynienia z dostępem zdalnym i będziesz musiał określić, gdzie znajduje się używany przez Ciebie komputer. Możesz w tym celu podać jego adres IP lub nazwę. Na przykład: gdyby oprogramowanie używane przez Ciebie do wykonywania poleceń SQL znajdowało się na komputerze o nazwie kaktus w sieci mojasuperfirma.pl, to mógłbyś określić konta administratora w następujący sposób: root@kaktus.mojasuperfirma.pl. Oczywiście takiego komputera nie ma, zatem użycie tego konta nic by Ci nie dało.

Dodanie nowego użytkownika

Mamy do Ciebie pytanie, na które odpowiedź jest prosta i oczywista:

Jak sądzisz, w jaki sposób SQL przechowuje informacje dotyczące użytkowników?

Oczywiście w tabeli! SQL korzysta z bazy danych zawierającej informacje o samym sobie. Zawiera ona takie informacje jak identyfikatory użytkowników, nazwy użytkowników, ich hasła oraz czynności, jakie poszczególni użytkownicy mogą wykonywać w poszczególnych bazach danych.

Aby wprowadzić nowego użytkownika, wystarczy podać jego nazwę i hasło. W języku SQL nie ma żadnego oficjalnego polecenia służącego do tworzenia użytkowników, jednak w większości systemów zarządzania relacyjnymi bazami będzie można użyć polecenia podobnego do tego przedstawionego poniżej:

```
CREATE USER eliza  
IDENTIFIED BY 'c13v3rp4s5w0rd';
```

A to jej hasło.

Oto nazwa użytkownika naszej najnowszej pracownicy, Elizy.



Czy nie mogliście ograniczyć Elizie dostępu do wybranych tabel już podczas tworzenia jej konta?

SQL nie określa, w jaki sposób należy zarządzać użytkownikami bazy.

W różnych systemach zarządzania relacyjnymi bazami danych proces tworzenia nowych użytkowników może wyglądać całkowicie inaczej. A zatem, aby określić prawidłowy sposób tworzenia użytkowników w konkretnym systemie zarządzania relacyjnymi bazami danych, będziesz musiał zatrzymać się i przeczytać dokumentację.

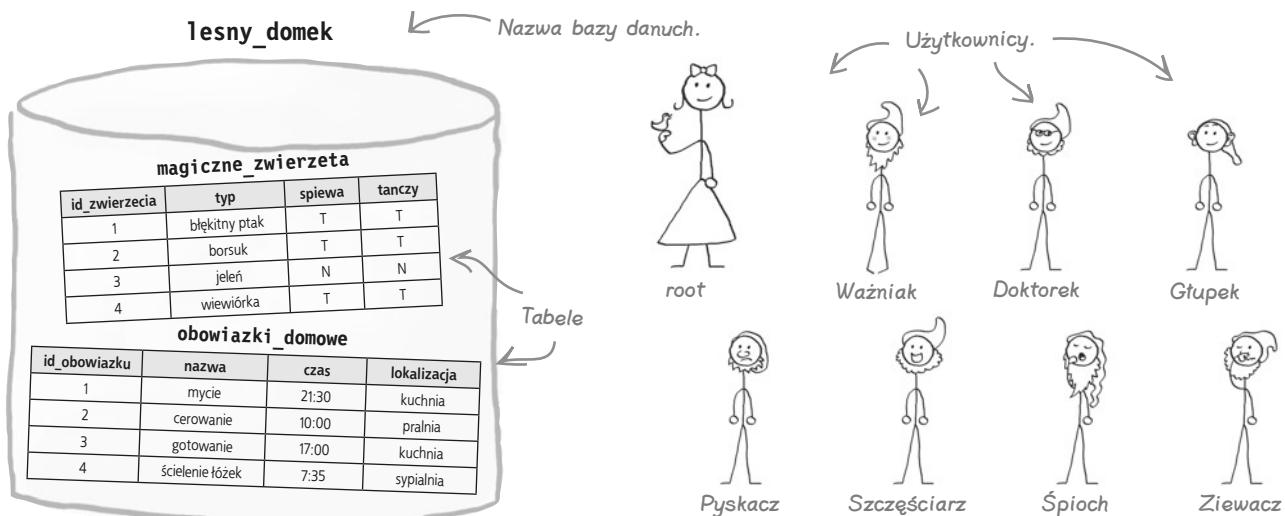
Mogliśmy, lecz czasami w momencie tworzenia konta użytkownika jeszcze nie wiadomo dokładnie, jakie uprawnienia będą mu potrzebne.

Jednak pomimo to musimy precyzyjne ustalić, do czego **nasz nowy użytkownik będzie mieć dostęp**. A zatem będziemy wykonywać poszczególne operacje kolejno, jedna po drugiej. Najpierw utworzymy użytkownika, a następnie przydzielimy mu uprawnienia, których potrzebuje. A następnie, nim zakończymy, złożymy to wszystko w jedną całość. Zaleta umiejętności określania uprawnień użytkowników niezależnie od procesu ich tworzenia polega na tym, że dzięki temu później będziemy mogli zmienić uprawnienia dostępu użytkowników do naszej bazy.

Dokładnie określ, czego poszczególni użytkownicy potrzebują

A zatem utworzyliśmy konto dla naszej nowej pracownicy Elizy. Bezpośrednio po utworzeniu konta Eliza nie posiada uprawnień do niczego. Musimy zatem skorzystać z polecenia **GRANT**, by przydzielić jej jakieś uprawnienia, ot, choćby do wykonywania poleceń SELECT w tabeli `kłowni_informacje`.

W odróżnieniu od administratora, który ma uprawnienia do wykonywania wszystkich operacji SQL na każdej dostępnej bazie danych, nowi użytkownicy nie dysponują żadnymi uprawnieniami. Oto jakie możliwości możemy uzyskać dzięki poleceniu GRANT:



Tylko niektórzy użytkownicy mogą modyfikować pewne tabele.

Jedynie osoby pełniące funkcję kierowniczą powinny mieć możliwość dodawania obowiązków do tabeli `obowiązki_domowe`. Tylko użytkownik `root` może wykonywać na niej polecenia INSERT, UPDATE i DELETE.

Natomiast użytkownik Szczęściarz nadzoruje tabelę `magiczne_zwierzeta` i może modyfikować jej strukturę przy użyciu polecenia ALTER, jak również wykonywać na niej wszelkie inne operacje.

Dostęp do danych w konkretnych tabelach mają jedynie wybrane użytkownicy.

Wszyscy z wyjątkiem użytkownika `Pyskacz` mogą wykonywać polecenia SELECT na tabeli `magiczne_zwierzeta`. `Pyskacz` nie może, bo nie lubi zwierząt, w szczególności tych, które mogą mu odpowieǳieć.

Nawet wewnątrz tabel mogą być stosowane uprawnienia: niektóre osoby mogą widzieć pewne kolumny tabeli, a dla innych pozostaną one niewidoczne.

Wszyscy z wyjątkiem `Głupka` mają dostęp do kolumny `instrukcje` w tabeli `obowiązki_domowe` (która całkowicie ogłupia `Głupka`).

Dzięki zastosowaniu polecenia GRANT możesz bardzo precyzyjnie określić, jakie operacje poszczególni użytkownicy mogą wykonywać w konkretnych tabelach.

Prosta postać polecenia GRANT

Jak wiemy, Eliza nie ma aktualnie żadnych uprawnień. Używając swej nazwy użytkownika oraz hasła, może się zalogować do bazy danych, ale nic ponadto. Eliza musi jednak mieć możliwość pobierania danych z tabeli `kлоwni_informacje` przy użyciu polecenia SELECT. Nic nie stoi na przeszkodzie, byśmy przydzieliли jej stosowne **uprawnienie**. Musimy w tym celu użyć polecenia GRANT. Oto ono:

Użytkownikowi przydziela się uprawnienie do wykonywania poleceń SELECT...

GRANT SELECT ON
kлоwni_informacje

... w tabeli, której nazwę tu oto podajemy.

TO eliza;

A użytkownikiem, któremu to uprawnienie przydzielamy, jest Eliza.

Eliza potrzebuje analogicznego uprawnienia do pozostałych tabel bazy danych kloonów, tak by w swoich zapytaniach mogła bez ograniczeń stosować podzapytania iłączenia. Musimy zatem wykonać odpowiednie polecenia GRANT, z których każde przydzieli Elizie uprawnienie do pobierania danych z odrębnej tabeli:

```
GRANT SELECT ON aktywnosci TO eliza;  
GRANT SELECT ON lokalizacja TO eliza;  
GRANT SELECT ON info_aktywnosci TO eliza;  
GRANT SELECT ON info_lokalizacja TO eliza;
```



Ćwiczenie

Kod	Jakie będą efekty jego wykonania?
1. GRANT INSERT ON magiczne_zwierzeta TO doktorek;
2. GRANT DELETE ON obowiazki_domowe TO szczesciarz, spioch;
3. GRANT DELETE ON obowiazki_domowe TO szczesciarz, spioch WITH GRANT OPTION;
4. GRANT SELECT(nazwa) ON obowiazki_domowe TO glupek;	<p style="text-align: right;">Podpowiedź: To jest nazwa kolumny.</p>
5. GRANT SELECT, INSERT ON magiczne_zwierzeta TO ziewacz;
6. GRANT ALL ON magiczne_zwierzeta TO wzniak;
A teraz spróbuj samodzielnie napisać kilka poleceń GRANT.	
7.	Daj Ważniakowi uprawnienie do wykonywania poleceń SELECT w tabeli obowiazki_domowe.
8.	Daj Śpiochowi prawo do wykonywania poleceń DELETE w tabeli magiczne_zwierzeta i jednocześnie do przydzielania (GRANT) innym użytkownikom uprawnienia do usuwania danych z tej tabeli.
9.	Daj wszystkim użytkownikom wszelkie uprawnienia do tabeli obowiazki_domowe.
10.	To polecenie pozwoli Ci za jednym razem nadać Doktorkowi uprawnienia do wykonywania poleceń SELECT we wszystkich tabelach bazy danych 1esny_domek.

Rozwiążanie ćwiczenia



Rozwiążanie Ćwiczenie

Skoro już zapanowałaś nad uprawnieniami Elizy, postaraj się określić, jakie skutki dla bazy danych `lesny_domek` ze strony 527 będą miały następujące polecenia GRANT.

- | Rozwiążanie | Ćwiczenie | Kod |
|---|--|--|
| 1. | GRANT INSERT ON magiczne_zwierzeta
TO doktorek; | |
| 2. | GRANT DELETE ON obowiazki_domowe
TO szczesiarz, spioch; | |
| 3. | GRANT DELETE ON obowiazki_domowe
TO szczesiarz, spioch
WITH GRANT OPTION; | |
| 4. | GRANT SELECT(nazwa) ON
obowiazki_domowe TO glupek; | |
| 5. | GRANT SELECT, INSERT ON
magiczne_zwierzeta
TO ziewacz; | |
| 6. | GRANT ALL ON magiczne_zwierzeta
TO wazniak; | |
| A teraz spróbuj samodzielnie napisać kilka polecen GRANT. | | |
| 7. | GRANT SELECT ON magiczne_zwierzeta TO
doktorek; | Daje Ważniakowi uprawnienie do wykonywania poleceń SELECT w tabeli obowiazki_domowe. |
| 8. | GRANT DELETE ON magiczne_zwierzeta TO
spioch WITH GRANT OPTION; | Daje Śpiochowi prawo do wykonywania poleceń DELETE w tabeli magiczne_zwierzeta i jednocześnie do przydzielania (GRANT) innym użytkownikom uprawnienia do usuwania danych z tej tabeli. |
| 9. | GRANT ALL ON obowiazki_domowe TO
wazniak, doktorek, glupek, pyskacz, szczesiarz,
spioch, ziewacz; | Daje wszystkim użytkownikom wszelkie uprawnienia do tabeli obowiazki_domowe. |
| 10. | GRANT SELECT ON lesny_domek.* TO
doktorek; | To polecenie pozwoli Ci za jednym razem nadać Doktorkowi uprawnienia do wykonywania poleceń SELECT we wszystkich tabelach bazy danych <code>lesny_domek</code> . |

Jakie będą efekty jego wykonania?

Zezwala Doktorkowi na wykonywanie poleceń **INSERT** w tabeli `magiczne_zwierzeta`.

Zezwala Szczęściażowi i Śpiochowi na wykonywanie poleceń **DELETE** w tabeli `obowiązki_domowe`.

Zezwala Szczęściażowi i Śpiochowi na wykonywanie poleceń **DELETE** w tabeli `obowiązki_domowe` i przydzielanie tego samego uprawnienia innym użytkownikom.

Zezwala Głupkowi na pobieranie z tabeli `obowiązki_domowe` wyłącznie zawartości kolumny nazwa.

Pozwala Ziewaczowi na wykonywanie poleceń **SELECT** oraz **INSERT** w tabeli `magiczne_zwierzeta`.

Zezwala Ważniakowi na wykonywanie poleceń **SELECT**, **UPDATE**, **INSERT** oraz **DELETE** w tabeli `magiczne_zwierzeta`.

Różne wersje polecenia GRANT

W ćwiczeniu, które przed chwilą wykonałeś, możesz zobaczyć najważniejsze i najczęściej spotykane wersje polecenia GRANT. Poniżej pokróćce je opisaliśmy:

1

W jednym poleceniu GRANT można podać nazwy dowolnej ilości użytkowników.

Wskazane w poleceniu uprawnienia zostaną w takim przypadku przydzielone wszystkim podanym użytkownikom.

2

Klauzula WITH GRANT OPTION zezwala użytkownikom na przydzielanie innym tego samego uprawnienia, które właśnie otrzymali.

Być może brzmi to dosyć zawile, jednak w rzeczywistości chodzi o prostą sprawę. Otóż jeśli jakiś użytkownik otrzymał właśnie uprawnienie do wykonywania polecenia SELECT w tabeli obowiazki_domowe, to będzie mógł przydzielić to samo uprawnienie (do wykonywania poleceń SELECT w tabeli obowiazki_domowe) innym użytkownikom bazy.

3

Zamiast nazwy tabeli w poleceniu GRANT można podać nazwę wybranej kolumny lub kilku kolumn.

Istnieje możliwość przydzielania uprawnienia do pobierania danych polecienniem SELECT tylko i wyłącznie z jednej, wskazanej kolumny. W takim przypadku wynikiem zapytania będzie wartość z tej kolumny.

4

W jednym poleceniu GRANT można podać kilka różnych uprawnień.

Wystarczy podać wszystkie uprawnienia, które chcesz przydzielić, oddzielając je od siebie przecinkami.

5

Polecenie GRANT ALL daje wskazanym użytkownikom prawo do wykonywania poleceń SELECT, UPDATE, INSERT oraz DELETE we wskazanej tabeli.

To po prostu skrócony sposób wydania polecenia „przydziel użytkownikom uprawnienia do wykonywania poleceń SELECT, UPDATE, INSERT oraz DELETE we wskazanej tabeli”.

6

Można przydzielić uprawnienia do wszystkich tabel w bazie, używając w tym celu wyrażenia nazwa_bazy.*.

Na podobnej zasadzie jak znak „*” w poleceniu SELECT oznacza wszystkie kolumny tabeli, tak gwiazdka w tym poleceniu powoduje przydzielanie uprawnień do wszystkich tabel bazy danych.

Usuwanie uprawnień

Załóżmy, że zdecydujemy się usunąć uprawnienie do wykonywania polecen SELECT, które wcześniej nadaliśmy Elizie. Aby to zrobić, będziemy musieli użyć polecenia REVOKE.

Czy pamiętasz nasze proste polecenie GRANT? Składnia polecenia REVOKE jest niemal identyczna; różni się jedynie tym, że zamiast „grant” występuje słowo „revoke” oraz zamiast „to” słowo „from”.

Usuwamy uprawnienie
do wykonywania polecen
SELECT.

REVOKE SELECT ON

kłowni_informacje

FROM eliza;

Zabieramy to uprawnienie
wskazanemu użytkownikowi.

Istnieje także możliwość usunięcia uprawnienia GRANT OPTION przy jednoczesnym pozostawieniu danemu użytkownikowi samego uprawnienia. W poniższym przykładzie po wykonaniu polecenia Szczęściorz i Śpioch wciąż będą mogli usuwać wiersze z tabeli obowiązki_domowe, jednak nie będą już mogli przydzielać uprawnienia do wykonywania tej czynności innym użytkownikom bazy.

Usuwamy jedynie
uprawnienie GRANT OPTION.

REVOKE GRANT OPTION ON

DELETE ON obowiązki_domowe

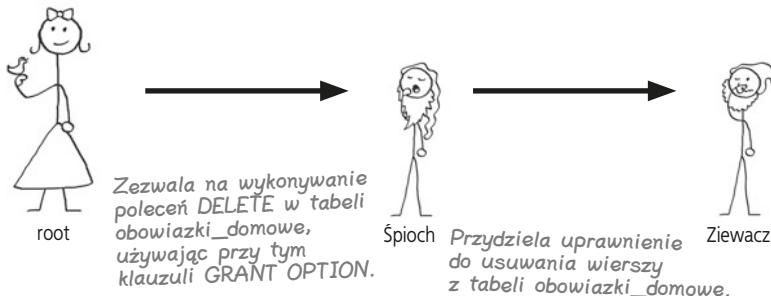
FROM szczesciarz, spioch;

Użytkownicy Szczęściorz i Śpioch
wciąż będą mogli wykonywać
polecenie DELETE w tabeli
obowiązki_domowe, nie będą
jednak mogli nadawać tego
uprawnienia innym.

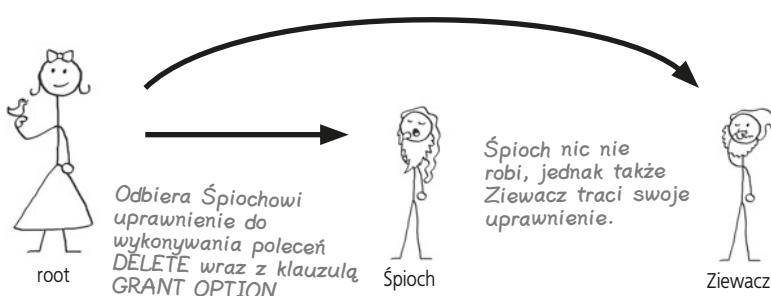


Usuwanie uprawnień przydzielonych dzięki GRANT OPTION

Rozważmy następujący scenariusz. Użytkownik root przydzielił Śpiochowi uprawnienie do wykonywania polecenia DELETE w tabeli obowiązki_domowe wraz z możliwością przydzielania tego samego uprawnienia innym użytkownikom (GRANT OPTION). Następnie Śpioch przydzielił to samo uprawnienie Ziewaczowi.



A teraz założmy, że użytkownik root zmienił zdanie i usuwa uprawnienie przydzielone wcześniej Śpiochowi. W takim przypadku uprawnienie to zostanie także odebrane Ziewaczowi, choć w poleceniu jawnie została podana wyłącznie nazwa użytkownika Śpioch.



Efektem ubocznym wykonania polecenia REVOKE było odebranie uprawnień także użytkownikowi Ziewacz. Istnieją dwa słowa kluczowe, których możesz używać i które mogą pomóc Ci określić, co ma się stać w momencie odbierania uprawnień przy użyciu polecenia REVOKE.

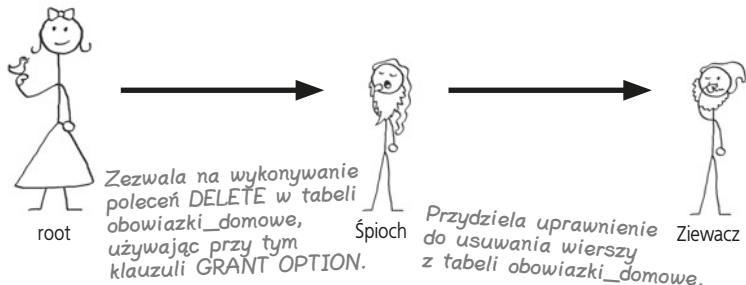
WYSIL SZARE KOMÓRKI

Właśnie masz zamiar poznać słowa kluczowe RESTRICT oraz CASCADE. Jak sądzisz, jakie możliwości zapewnia każde z nich?

Usuwanie uprawnień z większą precyzją

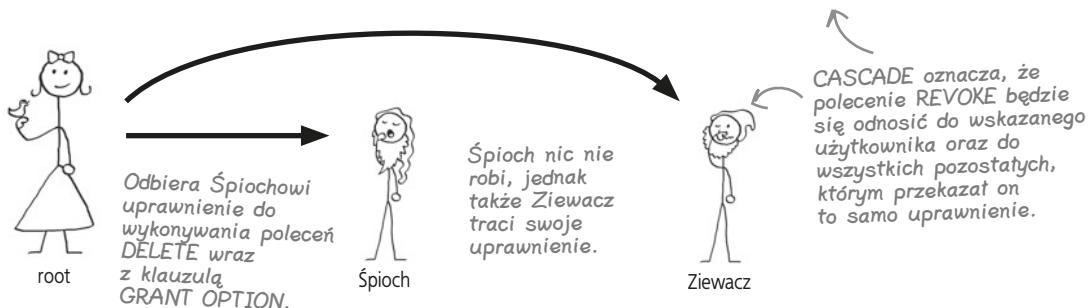
Precyjne usuwanie

Istnieją dwa sposoby usuwania uprawnień, które pozwalają mieć pewność, że przypadkowo nie odbierzemy uprawnień innym osobom niż te, którym planowaliśmy je odebrać. Słowa kluczowe CASCADE oraz RESTRICT umieszczone w plecieniu REVOKE pozwalają znacznie bardziej precyjnie określić, komu chcemy pozostawić uprawnienia, a komu je odebrać.



Pierwsze z tych słów kluczowych, CASCADE, odbiera uprawnienie wskazanemu użytkownikowi (w naszym przypadku będzie to Śpioch) oraz wszystkim innym, którym wskazany użytkownik udzielił tego samego uprawnienia.

REVOKE DELETE ON obowiązki_domowe FROM spioch CASCADE;



Zastosowanie w poleceniu REVOKE słowa kluczowego RESTRICT spowoduje zgłoszenie błędu, jeśli wskazany użytkownik przekazał dane uprawnienie jakiemukolwiek innemu użytkownikowi.

REVOKE DELETE ON obowiązki_domowe FROM spioch RESTRICT;



W powyższym przypadku obaj użytkownicy zachowali swoje uprawnienia, a użytkownik root zobaczył komunikat o błędzie. Polecenie nie zostało wykonane i pojawił się błąd, gdyż wykonanie polecenia spowodowałoby odebranie uprawnień także Ziewaczowi.

Zaostrz ołówek



Ktoś ciągle przydziela Elizie nieodpowiednie uprawnienia. Napisz prawidłowe polecenia REVOKE, dzięki którym ponownie będzie ona dysponować wyłącznie uprawnieniem do wykonywania poleceń SELECT.

```
GRANT SELECT, INSERT, DELETE ON lokalizacja TO eliza;
```

.....

```
GRANT ALL ON kloni_informacje TO eliza;
```

.....

```
GRANT SELECT, INSERT ON aktywnosci TO eliza;
```

.....

```
GRANT DELETE, SELECT ON info_lokalizacja TO eliza  
WITH GRANT OPTION;
```

.....

```
GRANT INSERT(lokalizacja), DELETE ON lokalizacja TO eliza;
```

.....

Zaostrz ołówek



Rozwiążanie

Ktoś ciągle przydziela Elizie nieodpowiednie uprawnienia. Napisz prawidłowe polecenia REVOKE, dzięki którym ponownie będzie ona dysponować wyłącznie uprawnieniem do wykonywania poleceń SELECT.

GRANT SELECT, INSERT, DELETE ON lokalizacja TO eliza;

REVOKE INSERT, UPDATE, DELETE ON lokalizacja FROM eliza;

GRANT ALL ON kloni_informacje TO eliza;

REVOKE INSERT, UPDATE, DELETE ON kloni_informacje FROM eliza;

GRANT SELECT, INSERT ON aktywnosci TO eliza;

REVOKE INSERT ON aktywnosci FROM eliza;

GRANT DELETE, SELECT ON info_lokalizacja TO eliza
WITH GRANT OPTION;

REVOKE DELETE ON info_lokalizacja FROM eliza CASCADE;

GRANT INSERT(lokalizacja), DELETE ON lokalizacja TO eliza;

REVOKE INSERT(lokalizacja), DELETE ON lokalizacja FROM eliza;

Wygląda na to, że także tutaj moglibyśmy użyć polecenia GRANT, by upewnić się, że Eliza wciąż będzie mogła wykonywać polecenia SELECT.

I lepiej upewnijmy się, że Eliza nie przydzieli swoich uprawnień jakimś innym użytkownikom.

Chcemy pozostawić Elizie uprawnienie do wykonywania poleceń SELECT; właśnie dlatego nie odbieramy jej wszystkich uprawnień.

Ten sam efekt można by uzyskać w nieco inny sposób — odebrać Elizie wszystkie uprawnienia, a następnie przydzielić jej ponownie tylko to, które chcemy, by miało.

.....
Nie istniejąca
grupie pytania

P: Wciąż myślę o poleceniach GRANT, w których są określane nazwy kolumn. Co się dzieje, kiedy przydzielimy użytkownikowi uprawnienie do wykonywania poleceń INSERT, ale tylko na jednej kolumnie tabeli?

O: To dobre pytanie. W praktyce takie uprawnienie jest raczej nieprzydatne. Jeśli możesz określić wartość tylko jednej kolumny, to i tak nie jesteś w stanie dodać do tabeli nowego wiersza. Takie rozwiązanie może zatem działać tylko w jednym przypadku — kiedy tabela będzie się składać wyłącznie z jednej kolumny — tej, do której użytkownik ma uprawnienia.

P: Czy są także inne, równie bezużyteczne wersje polecenia GRANT?

O: Niemal wszystkie uprawnienia odwołujące się do poszczególnych kolumn są raczej nieprzydatne, chyba że dotyczą polecenia SELECT.

P: Założmy, że chcę dodać nowego użytkownika i przydzielić mu uprawnienia do wykonywania poleczeń SELECT we wszystkich tabelach we wszystkich bazach danych. Czy można to zrobić w jakiś szybki i prosty sposób?

O: Podobnie jak wszystkie polecenia przedstawiane w tym rozdziale, także i to zależy od używanego systemu zarządzania relacyjnymi bazami danych. W MySQL-u takie globalne uprawnienia można nadawać w następujący sposób:

```
GRANT SELECT ON *.*  
TO eliza;
```

Pierwsza gwiazdka oznacza wszystkie bazy danych, a druga wszystkie tabele.

P: Czy opcja CASCADE jest domyślnie stosowana w poleceniach REVOKE?

O: Zazwyczaj tak. Faktycznie opcja ta jest domyślna, ale trzeba powtórzyć jeszcze raz: wszystko zależy od używanego systemu zarządzania bazami danych.

P: Co się stanie, jeśli spróbuję odebrać użytkownikowi uprawnienie, którego w ogóle nie posiada?

O: Zostanie wyświetlony komunikat o błędzie informujący o braku uprawnień.

P: A co się stanie, jeśli dwie odrębne osoby przydzielą Ziewaczowi z poprzedniego przykładu jakieś uprawnienia, a następnie administrator odbierze mu jedno z nich?

O: Dopiero wtedy zaczną się prawdziwe problemy. W niektórych bazach danych, jeśli została zastosowana opcja CASCADE, to nie ma znaczenia, od kogo pochodzi odbierane uprawnienie, w innych natomiast ma. Także w tym przypadku będziesz zatem musiał przejrzeć dokumentację używanego oprogramowania.

P: Czy oprócz tabel i kolumn są jeszcze jakieś inne obiekty bazy danych, do których można przydzielać uprawnienia poleceniem GRANT i odbierać poleceniem REVOKE?

O: Dokładnie w taki sam sposób można używać tych polecień w celu przydzielania i odbierania praw dostępu do widoków, chyba że widok nie pozwala na aktualizację danych w tabelach. Jeśli widok nie pozwala na aktualizację, to nie możesz używać go do wykonywania poleczeń INSERT. Nie jest to możliwe, nawet jeśli miałbyś do tego uprawnienia. I podobnie jak w przypadku tabel możesz także przydzielać prawa dostępu do poszczególnych kolumn widoku.



A zatem gdybym chciała, żeby pięć różnych osób miało dokładnie te same uprawnienia, musiałabym podać ich nazwy na końcu polecenia GRANT, oddzielając je od siebie przecinkami?

Bez wątpienia takie rozwiązanie by zadziałało. I na pewno powinnaś zastosować to rozwiązanie w przypadkach, kiedy użytkowników nie jest wielu.

Jednak kiedy Twoja organizacja czy firma się rozwinię, zaczniesz zapewne używać „klas” użytkowników. Na przykład 10 pracowników będzie się zajmować wyłącznie wprowadzaniem danych do bazy. Takie osoby będą musiały wykonywać polecenia INSERT i SELECT wyłącznie na ściśle określonej grupie tabel. Możesz jednak mieć także trzech zaawansowanych użytkowników, którzy mogą wszystko, oraz wielu użytkowników, którzy mogą wykonywać wyłącznie polecenia SELECT. Możesz nawet korzystać z innych programów oraz aplikacji internetowych, które będą się łączyć z bazą danych i korzystać z niej na różne sposoby.

Dlaczego dzielenie się jest złe



Chwileczkę. Skoro napisaliście, że można utworzyć klasy użytkowników, to dlaczego nie można by ustanowić po jednym użytkowniku dla każdej z tych klas, a następnie podać jego nazwę i hasło wybranym osobom?

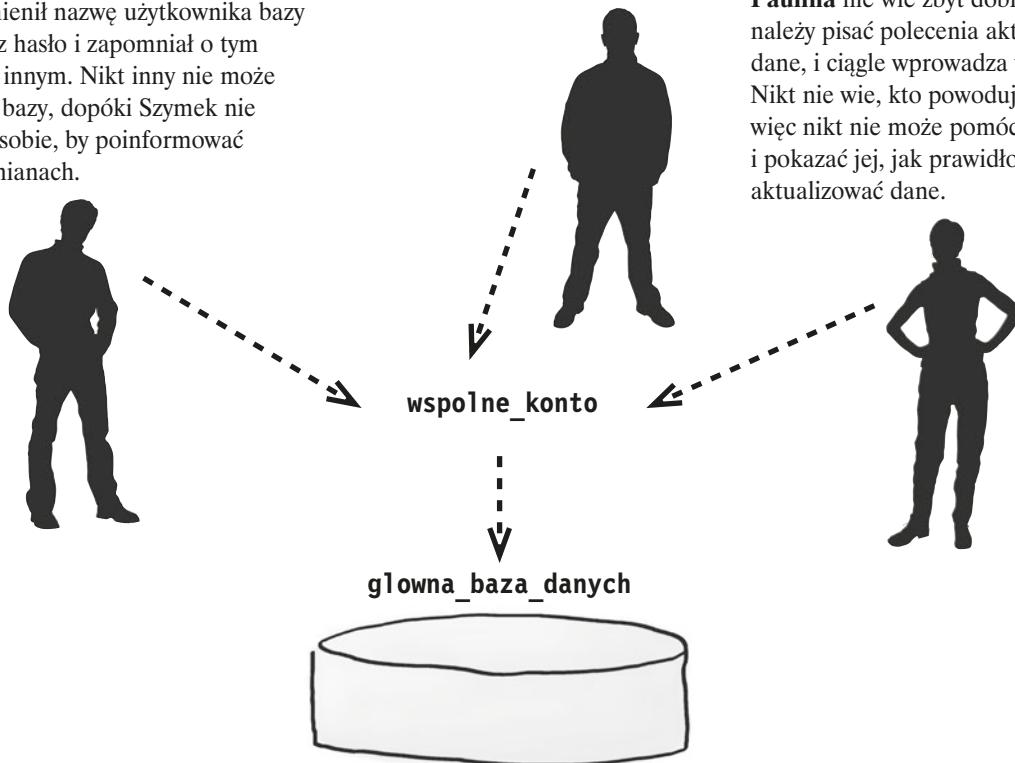
Współużytkowane konta przysparzają problemów

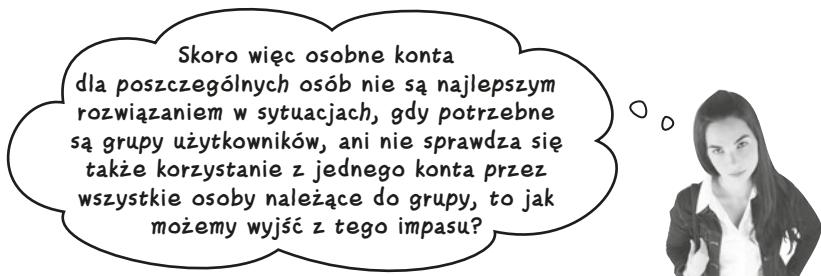
Choć niektóre firmy z powodzeniem stosują technikę polegającą na obsłudze bazy danych przy wykorzystaniu jednego konta użytkownika, to jednak nie jest to szczególnie bezpieczne rozwiązanie. Poniżej przedstawiliśmy przykład pokazujący, co mogłyby pójść źle:

Radek musi dysponować pełnymi uprawnieniami do wykonywania wszystkich możliwych operacji na całej zawartości bazy danych, inaczej nie będzie w stanie wykonywać swych obowiązków. To z kolei zwiększa zagrożenie, że inne osoby korzystające z bazy, które gorzej znają język SQL, mogą niechcący uszkodzić lub zniszczyć bazę.

Szymek zmienił nazwę użytkownika bazy danych oraz hasło i zapomniał o tym powiedzieć innym. Nikt inny nie może korzystać z bazy, dopóki Szymek nie przypomni sobie, by poinformować innych o zmianach.

Paulina nie wie zbyt dobrze, jak należy pisać polecenia aktualizujące dane, i ciągle wprowadza w nich błędy. Nikt nie wie, kto powoduje problemy, więc nikt nie może pomóc Paulinie i pokazać jej, jak prawidłowo należy aktualizować dane.





Potrzebujemy mechanizmu pozwalającego na przydzielanie niezbędnych uprawnień całej grupie, a jednocześnie pozwalającego na tworzenie indywidualnych kont dla poszczególnych użytkowników.

Potrzebujemy tak zwanych **ról**. Rola to sposób pozwalający na zgrupowanie wielu indywidualnych uprawnień i stosowanie ich jako jednej grupy. Taka rola staje się obiektem w bazie danych, który w razie zmiany bazy możesz modyfikować i dostosowywać do nowych potrzeb, bez konieczności wprowadzania zmian w uprawnieniach poszczególnych użytkowników.

A utworzenie roli jest wyjątkowo proste:

CREATE ROLE wprowadzanie_danych;

Nazwa tworzonej roli.



Uwaga!

W MySQL-u nie ma ról.

Role są czymś, co być może pojawi się w przyszłych wersjach MySQL-a. Na razie jednak będziesz musiał przypisywać potrzebne uprawnienia poszczególnym użytkownikom.

Aby przypisać roli wybrane uprawnienie, wystarczy podać jej nazwę w poleceniu GRANT, dokładnie tak samo, jak wcześniej podawaliśmy nazwę użytkownika.

GRANT SELECT, INSERT ON jakas_tabela T0 wprowadzanie_danych;

Przypisując uprawnienia, zamiast nazwy użytkownika możemy także podać nazwę roli.

A zatem utworzyliśmy rolę i określiliśmy jej uprawnienia.

Teraz musimy przypisać tę rolę jakiemuś użytkownikowi.

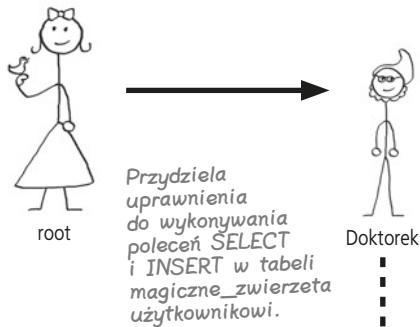
Zabawa z rolami

Stosowanie roli

Przed utworzeniem roli mogliśmy przydzielać użytkownikom zajmującym się wpisywaniem danych niezbędne uprawnienia, korzystając z polecenia GRANT, na przykład takiego jak poniższe:

```
GRANT SELECT, INSERT  
ON magiczne_zwierzeta  
TO doktorek;
```

Stary sposób

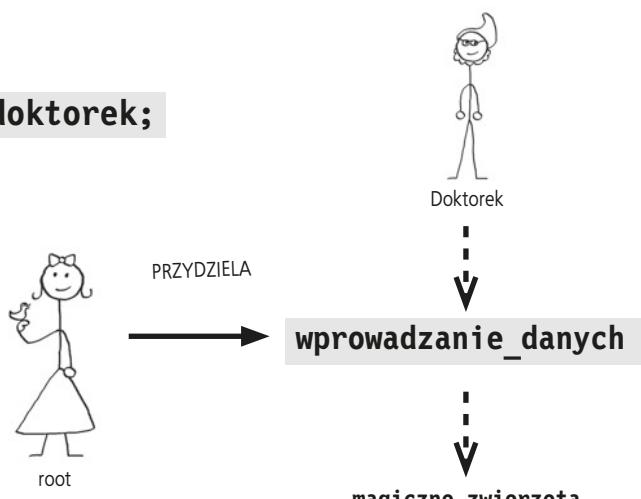


id_zwierzenia	typ	spiewa	tanczy
1	blekitny ptak	T	T
2	borsuk	T	T
3	jeleń	N	N
4	wiewiórka	T	T

Teraz wystarczy, że w poleceniu GRANT zastąpimy uprawnienia naszą nową rolą i przypiszemy ją użytkownikowi Doktorek. Poszczególnych uprawnień nie musimy już podawać, gdyż są zapisane w roli wprowadzanie_danych:

```
GRANT wprowadzanie_danych TO doktorek;
```

Nazwa roli zastępuje nazwę tabeli oraz uprawnienia.



id_zwierzenia	typ	spiewa	tanczy
1	blekitny ptak	T	T
2	borsuk	T	T
3	jeleń	N	N
4	wiewiórka	T	T

Usuwanie roli

Kiedy rola nie będzie Ci już potrzebna, nie będzie powodu, by ją dłużej przechowywać. W każdej chwili możesz usunąć rolę, używając do tego celu polecenia DROP o następującej postaci:

```
DROP ROLE wprowadzanie_danych;
```

Nie istnieja
grupie pytania

P: A jeśli chciałbym przydzielić uprawnienia do wszystkich tabel w bazie? Czy muszę podawać nazwę każdej z nich?

O: Nie, możesz użyć następującej składni:

```
GRANT SELECT, INSERT, DELETE
ON lista_grzesia.*
TO jozek;
```

A zatem wystarczy, że podasz nazwę bazy danych i gwiazdkę — w ten sposób przypiszesz podane uprawnienia do wszystkich tabel w bazie danych.

P: A jeśli rola została przypisana jakiemuś użytkownikowi, to czy można ją usunąć?

O: Można usuwać używane role. Musisz jednak bardzo uważać, usuwając role, by nie odciąć użytkowników od uprawnień, które są im potrzebne.

P: Czy to oznacza, że jeśli użytkownik ma jakąś rolę, która następnie zostanie usunięta, to użytkownik straci uprawnienia skojarzone z tą rolą?

O: Dokładnie tak. To odpowiednik sytuacji, w której przydzieliłeśbyś użytkownikowi uprawnienia ręcznie, a następnie odebrał je, używając polecenia REVOKE. Jednak w tym przypadku zamiast zmienić uprawnienia pojedynczego użytkownika, odbierzesz je wszystkim użytkownikom, którym została przypisana rola.

P: Czy użytkownik może w danej chwili posiadać więcej niż jedną rolę?

O: Owszem. Należy tylko upewnić się, że pomiędzy poszczególnymi rolami nie występują konflikty uprawnień, gdyż w przeciwnym razie możesz sobie przysporzyć wielu problemów. Pamiętaj, że odebranie konkretnego uprawnienia ma wyższy priorytet niż przydzielenie tego uprawnienia.

Zaostrz ołówek



Usuwanie uprawnień przydzielonych przy użyciu roli

Usuwanie uprawnień przydzielonych przy wykorzystaniu roli działa bardzo podobnie, jak usuwanie uprawnień przypisanych przy użyciu polecenia GRANT. Sprawdź, czy **bez zagłówka do wcześniejszej części rozdziału** będziesz potrafił napisać polecenie, które usunie przypisanie roli wprowadzanie_danych użytkownikowi Doktorek.

Zaostrz ołówek



Rozwiążanie

Usuwanie uprawnień przydzielonych przy wykorzystaniu roli działa bardzo podobnie, jak usuwanie uprawnień przypisanych przy użyciu polecenia GRANT. Sprawdź, czy **bez zagłdania do wcześniejszej części rozdziału** będziesz potrafił napisać polecenie, które usunie przypisanie roli wprowadzanie_danych użytkownikowi Doktorek.

REVOKE wprowadzanie_danych FROM doktorek;

Stosowanie roli z klauzulą WITH ADMIN OPTION

Analogicznie do przedstawionej wcześniej postaci polecenia GRANT, w której można zastosować klauzulę WITH GRANT OPTION, także w przypadku stosowania ról można w tym poleceniu zastosować opcjonalną klauzulę — WITH ADMIN OPTION.

GRANT wprowadzanie_danych TO doktorek WITH ADMIN OPTION;

Klauzula WITH ADMIN OPTION zapewnia użytkownikowi Doktorek możliwość przypisywania tej samej roli (w naszym przypadku jest to rola wprowadzanie_danych) innym użytkownikom bazy.

GRANT wprowadzanie_danych TO szczesciarz;

Teraz użytkownik Doktorek będzie posiadał uprawnienia administracyjne i będzie mógł przypisać role wprowadzanie_danych użytkownikowi Szczęściaż dokładnie w taki sam sposób, w jaki ktoś przypisał ją jemu.

Także w przypadku stosowania ról można stosować w poleceniu REVOKE znane nam już słowa kluczowe CASCADE oraz RESTRICT. Zobaczmy, jak one działają:

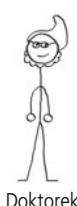
Usuwanie roli z zastosowaniem słowa kluczowego CASCADE

W przypadku zastosowania słowa kluczowego CASCADE polecenie REVOKE odbiera rolę nie tylko wskazanemu użytkownikowi, lecz także wszystkim innym, którym rola ta została przydzielona przez wskazanego użytkownika.

REVOKE wprowadzanie_danych FROM doktorek CASCADE;



Usuwa przypisaną użytkownikowi Doktorek rolę wprowadzanie_danych



Szczęściaż traci uprawnienia przypisane mu przez Doktorka.



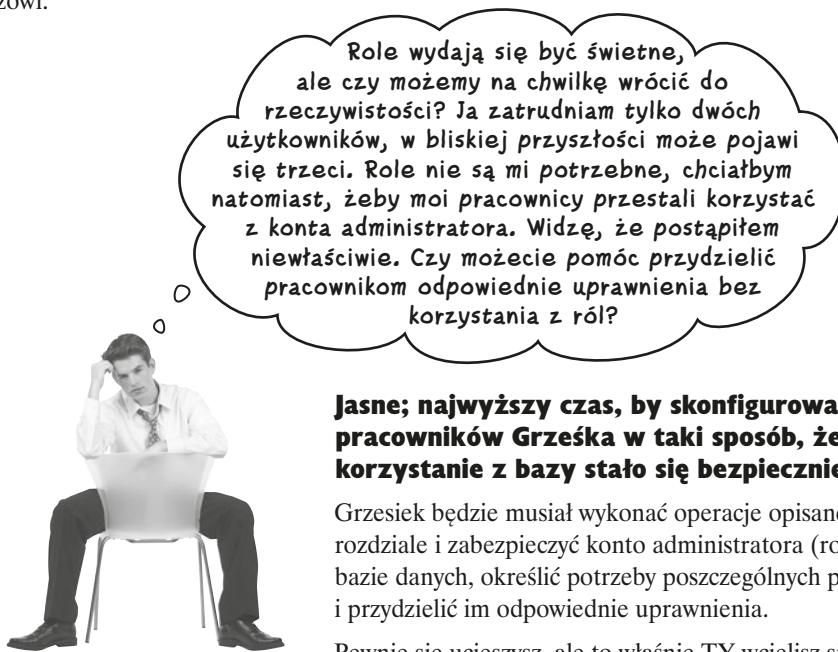
W razie zastosowania słowa kluczowego CASCADE polecenie REVOKE odbierze uprawnienia wskazanemu użytkownikowi, jak również wszystkim innym użytkownikom, którym wskazany użytkownik je przydzielił.

Usuwanie ról z wykorzystaniem słowa kluczowego RESTRICT

Zastosowanie słowa kluczowego RESTRICT w poleceniu REVOKE spowoduje wystąpienie błędu, jeśli wskazany użytkownik przydzielił te same uprawnienia komu innemu.



A zatem w powyższym przykładzie obaj użytkownicy, Doktorek i Szczęściarz, zachowają swoje uprawnienia, a użytkownik root zobaczy komunikat o błędzie. Polecenie nie zostało wykonane i pojawił się błąd, gdyż wykonanie polecenia spowodowałoby odebranie uprawnień także Szczęściarzowi.



Pewnie się ucieszysz, ale to właśnie TY wcielisz się w postać Grześka...

Bądź GRZEŚKIEM



Twoim zadaniem jest po raz ostatni wcielić się w postać Grześka i poprawić uprawnienia użytkowników w bazie danych, tak by pracownicy nie byli w stanie przypadkowo zniszczyć danych.

Przeczytaj podane poniżej opisy zadań poszczególnych pracowników i podaj polecenia GRANT, które zapewnią im dostęp do niezbędnych informacji, a jednocześnie nie pozwolą na dostęp do danych, na których nie powinni operować.



Franek: „Jestem odpowiedzialny za dopasowywanie osób do ofert pracy. Nigdy niczego nie zapisuję w bazie, jednak usuwam z niej rekordy, kiedy znajdę dopasowanie lub gdy oferta zostanie zrealizowana. Czasami muszę poszukać informacji kontaktowych w tabeli moje_kontakty”.

Kuba: „Zajmuję się wpisywaniem wszystkich nowych danych do całej bazy. Teraz, kiedy już nie mogę przypadkowo zapisać wartości X w polu płci, naprawdę sprawnie zapisuję dane. Jestem także odpowiedzialny za aktualizację danych. Uczę się także, jak usuwać dane z bazy, jednak Grzesiek kazał mi tego nie robić. Oczywiście nie wie o tym, ale...”

Józek: „Właśnie zostałem zatrudniony przez Grześka do obsługi procesu dopasowywania osób i ofert pracy. Grzesiek chciałby zintegrować swoją bazę danych z witryną WWW. Jestem bardziej programistą niż specem od baz danych i SQL-a, jednak potrafię pisać proste polecenia SELECT. Nie wykonuję natomiast polecień INSERT. I nie używam Windows. Ech... nie... to tylko taki kiepski żart”.

Przyjrzyj się strukturze bazy danych lista_grzesia i napisz polecenia GRANT, które ochronią ją przed całkowitym zniszczeniem.

Napisz polecenie, które określi hasło użytkownika administratora, określonego także jako „root”.

.....

Napisz trzy polecenia, które utworzą konta użytkowników bazy danych dla każdego pracownika firmy Grześka.

.....

.....

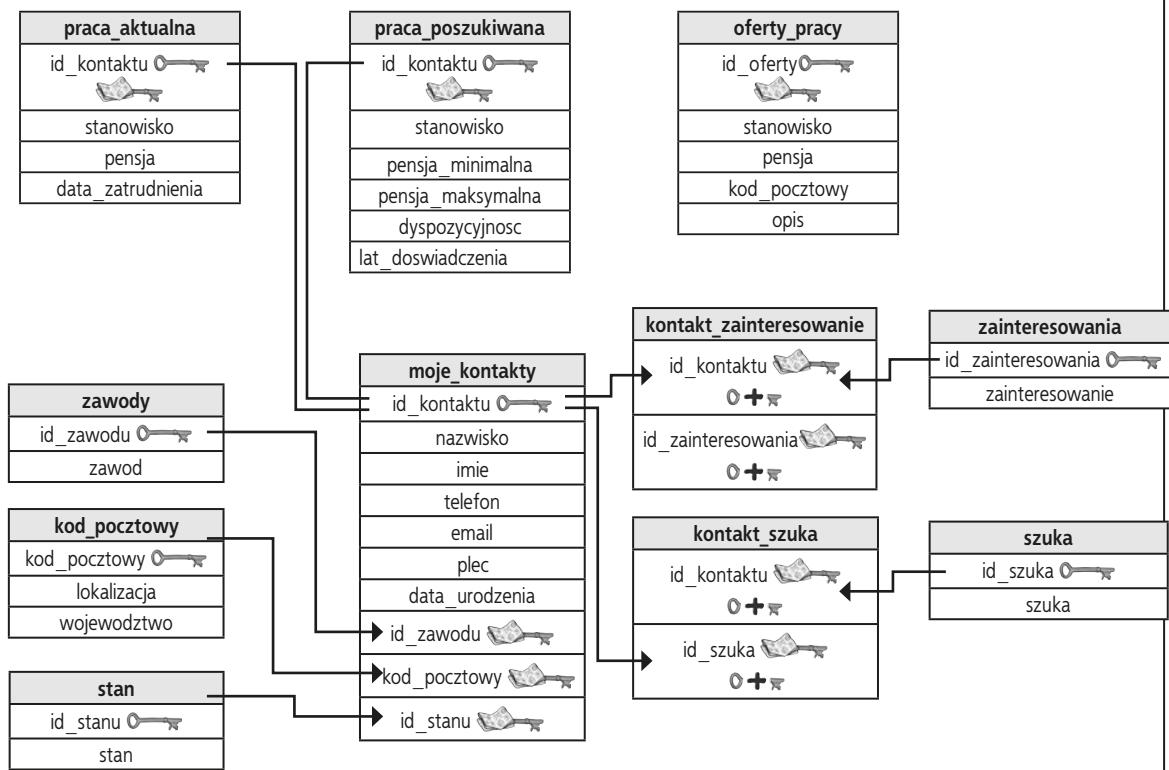
.....

Dla każdego pracownika napisz polecenie GRANT, które przydzieli mu niezbędne uprawnienia.

.....

.....

.....



Bądź GRZEŚKIEM. Rozwiążanie



Twoim zadaniem jest po raz ostatni wcielić się w postać Grześka i poprawić uprawnienia użytkowników w bazie danych, tak by pracownicy nie byli w stanie przypadkowo zniszczyć danych.

Przeczytaj podane poniżej opisy zadań poszczególnych pracowników i podaj polecenia GRANT, które zapewnią im dostęp do niezbędnych informacji, a jednocześnie nie pozwolą na dostęp do danych, na których nie powinni operować.

Napisz polecenie, które określi hasło użytkownika administratora, określonego także jako „root”.

```
SET PASSWORD FOR root@localhost = PASSWORD('gr3Grulz');
```

Napisz trzy polecenia, które utworzą konta użytkowników bazy danych dla każdego pracownika firmy Grześka.

```
CREATE USER franek IDENTIFIED BY 'j0bM4teH';  
CREATE USER kuba IDENTIFIED BY 'NOmor3Xs';  
CREATE USER jozek IDENTIFIED BY 'e3LeCTD00d';
```

Nie przejmuj się, jeśli wymyślone przez Ciebie hasła będą inne. Jeśli tylko poszczególne elementy polecenia zapisacie w odpowiedniej kolejności, to wszystko będzie w porządku!

Dla każdego pracownika napisz polecenie GRANT, które przydzieli mu niezbędne uprawnienia.

```
GRANT DELETE ON oferty_pracy TO franek;  
GRANT SELECT ON moje_kontakty TO franek;
```

Franek musi mieć możliwość usuwania ofert z listy oraz pobierania danych z tabeli moje_kontakty.

```
GRANT SELECT, INSERT ON lista_grzesia.* TO kuba;
```

Kuba musi mieć uprawnienia do wykonywania poleceń SELECT i INSERT we wszystkich tabelach bazy danych lista_grzesia. Na razie nie zapewniamy mu możliwości zabawy polecienniem DELETE.

```
GRANT SELECT ON moje_kontakty, zawody, kod_pocztowy, stan,  
kontakt_zainteresowanie, zainteresowania, kontakt_szuka, szuka TO jozek;
```

Z kolei Józek musi mieć możliwość pobierania danych ze wszystkich oryginalnych tabel bazy, oprócz tabel związanych z zatrudnieniem i ofertami pracy.

Łączenie poleceń CREATE USER i GRANT



Zanim sobie pójdziecie,
chciałem zapytać, czy
moglibyśmy spróbować połączyć
polecenia CREATE USER oraz
GRANT i wykonać je jako jedno
polecenie?

Owszem, możesz... To całkiem proste, wystarczy połączyć fragmenty dwóch znanych Ci już poleceń.

Poniżej przedstawiliśmy polecenia CREATE USER oraz GRANT, których użyliśmy do utworzenia użytkownika o imieniu Eliza i określenia jego uprawnień:

```
CREATE USER eliza  
IDENTIFIED BY 'c13v3rp4s5w0rd';
```

```
GRANT SELECT ON  
klowni_informacje  
TO eliza;
```

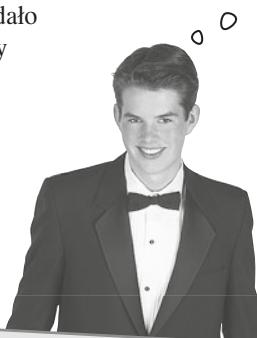
Możemy połączyć te polecenia, odrzucając przy tym słowa kluczowe CREATE USER. Ponieważ użytkownik *eliza* musi istnieć, zanim będziemy mu mogli przypisać jakiekolwiek uprawnienia, zatem system zarządzania relacyjną bazą danych w pierwszej kolejności sprawdzi, czy użytkownik istnieje, a w razie potrzeby automatycznie go utworzy.

```
GRANT SELECT ON  
klowni_informacje  
TO eliza  
IDENTIFIED BY 'c13v3rp4s5w0rd';
```

Lista Grześka stała się usługą globalną!

Dzięki Waszej pomocy Grzesiek już bardzo dobrze posługuje się językiem SQL, a dzięki dokształcaniu swoich pracowników — Kuby, Franka i Józka — Grześkowi udało się rozszerzyć działalność swojej firmy także na reklamy lokalne i fora dyskusyjne.

A wiecie, co z tego wszystkiego jest najlepsze? Usługi Grześka odniosły tak ogromny sukces w Bazodanowie, że obecnie ponad 500 miast na całym świecie posiada własne filie firmy Grześka, a sam Grzesiek znalazł się na pierwszych stronach gazet!



*Dziękuję wam. Nigdy bym tego nie osiągnął bez waszej pomocy!
Hej, chciałbym otworzyć filię w waszym mieście, jesteście może zainteresowani?*

TYGODNIK BAZODETEKTYWISTYCZNY

Ogromny wzrost popularności Listy Grzesia.

FILIE I FORA

Przyjaciele i rodzina twierdzą, że sława i popularność wcale nie zmieniły Grześka.

Autor: Radek Wierszyński

ETATOWY DETEKTYW BAZODANOWY

BAZODANOWO: Lokalny przedsiębiorca Grzesiek doczekał się swoich pięciu minut. Jego internetowa baza danych rozrosła się od miejsca do przechowywania adresów znajomych zapisywanych wcześniej na karteczkach, poprzez prostą tabelę, bazę danych składającą się z wielu tabel, aż do globalnej usługi zajmującej się dobieraniem osób do dostępnych ofert pracy. A to jeszcze nie cały zakres działalności firmy Grześka.



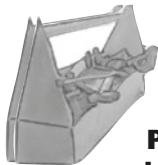
Czy w twoim mieście też już jest filia Listy Grzesia? Jeśli nie, to jest to jedynie kwestią czasu, tak uważają wszyscy lokalni analitycy.

Czy już coś wiesz o filii Listy Grzesia w swoim mieście?



Korzystaj z SQL-a w swoich własnych projektach, bo także Ty możesz odnieść taki sukces jak Grzesiek.

Byliśmy szczęśliwi, mogąc gościć Cię w Bazodanowie. Jest nam przykro, że musimy się pożegnać, jednak nie ma nic lepszego od skorzystania ze zdobytej wiedzy i wykorzystania jej do tworzenia własnych baz danych... poza tym mamy pewność, że także w Twojej okolicy są klowni, których można śledzić, albo pączki oczekujące na sklasyfikowanie i cenę, albo Listy [tu wpisz swoje imię], które oczekują na swego twórcę. W pozostałej części książki czeka na Ciebie jeszcze kilka perełek, no i nie zapominajmy o indeksie... a potem... potem nie pozostaje już nic innego, jak tylko zastosować te wszystkie pomysły i idee w praktyce.



Przybornik SQL

**Gratulacje, właśnie skończyłeś
lekturę rozdziału dwunastego!
Poświęć chwilkę na przypomnienie
sobie zasad i sposobów zapewniania
bezpieczeństwa baz danych, które
przedstawiliśmy w tym rozdziale. Kompletną
listę porad znajdziesz w dodatku C,
zamieszczonym na końcu książki.**

REVOKE

Tego polecenia SQL możesz
używać, by odbierać
użytkownikom uprawnienia.

WITH ADMIN OPTION

Ta klauzula pozwala wszystkim
użytkownikom, którym została
przypisana rola, przypisywać
tę rolę innym użytkownikom.

CREATE USER

Polecenie używane przez
systemy zarządzania
relacyjnymi bazami danych,
które pozwala na utworzenie
użytkownika i określenie
jego hasła.

WITH GRANT OPTION

Pozwala użytkownikowi
przydzielać innym te same
uprawnienia, które on posiada.

GRANT

Pozwala w bardzo precyzyjny
sposób określić na podstawie
uprawnień przydzielanych
poszczególnym użytkownikom,
jakie operacje będą oni mogli
wykonywać na poszczególnych
tabelach oraz ich kolumnach.

Rola

Rola jest grupą uprawnień.
Role pozwalają grupować
wybrane uprawnienia
i przypisywać je dowolnej
liczbie użytkowników.

Dodatek A Pozostałości



Dziesięć najważniejszych zagadnień (których nie opisaliśmy wcześniej)



Nawet po tym wszystkim jest jeszcze coś więcej. Jest jeszcze dosłownie kilka rzeczy, o których, jak sądzimy, powinieneś wiedzieć. Nie czulibyśmy się w porządku, gdybyśmy je całkowicie zignorowali i nie poświęcili im choćby krótkiej wzmianki. A zatem, nim będziesz mógł odłożyć tę książkę na półkę, przeczytaj o tych **drobnych, lecz ważnych sprawach**.

Poza tym kiedy skończysz lekturę tego dodatku, pozostaną Ci jeszcze dwa następne... oraz indeks... no i może jeszcze jakieś reklamy... a potem już naprawdę skończysz. Obiecujemy!

Nr 1. Znajdź i zainstaluj graficzny program do obsługi używanego systemu zarządzania bazami danych

Choć umiejętność wykonywania poleceń SQL bezpośrednio z poziomu konsoli jest bardzo ważna, to jednak obecnie już doskonale wiesz, co i jak chcesz zrobić. Zasługujesz zatem na prostszy sposób tworzenia tabel i przeglądania ich zawartości.

Praktycznie dla każdego systemu zarządzania relacyjnymi bazami danych istnieje oprogramowanie z graficznym interfejsem użytkownika. Poniżej przedstawiliśmy krótkie informacje na temat programów tego typu współpracujących z bazą MySQL.

Programy MySQL Tools

Pobierając oprogramowanie serwera bazy MySQL, możesz także pobrać grupę programów MySQL Tools, z których najważniejszym jest MySQL Administrator. Możesz je pobrać bezpośrednio ze strony o następującym adresie:

<http://dev.mysql.com/downloads/gui-tools/5.0.htm>

Programy te są dostępne w wersjach dla systemu Windows, Mac OS X oraz Linux. Program MySQL Administrator pozwala na bardzo proste przeglądanie, tworzenie i modyfikowanie baz danych oraz tabel.

Na pewno spodoba Ci się także program MySQL Query Browser. Pozwala on na wpisywanie zapytań i wygodne przeglądanie ich wyników w graficznym środowisku programu, a nie w oknie konsoli.

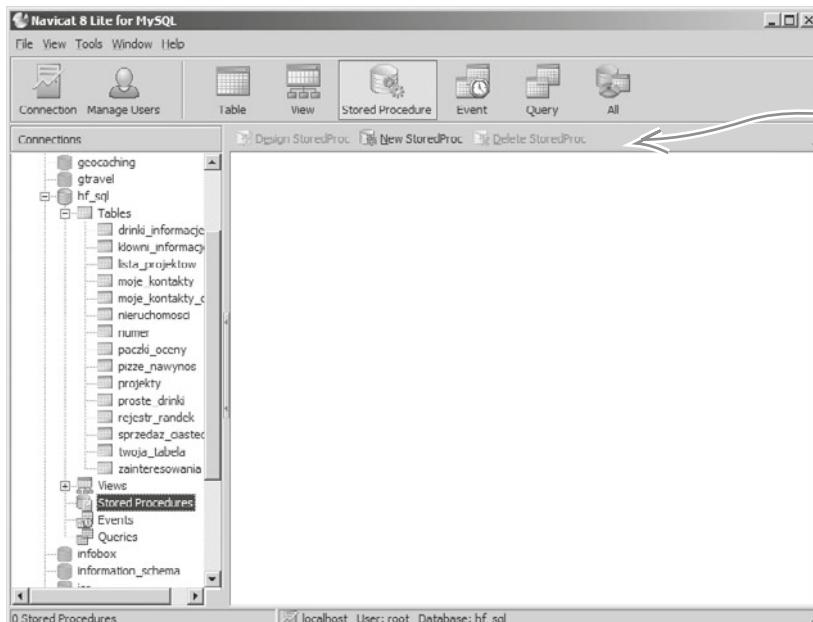
Tutaj wpisz swoje zapytanie.

The screenshot shows the MySQL Query Browser interface. At the top, there's a menu bar with Polish labels: Plik, Edycja, Widok, Zapytanie, Skrypt, Narzędzia, Okno, MySQL Enterprise, and Pomoc. Below the menu is a toolbar with icons for Back, Forward, Refresh, and buttons for Wykonaj (Execute) and Stop. A query window contains the command: `SELECT imie, wyglad FROM klowni_informacje GROUP BY imie`. An arrow points from this command to the execute button. The main area is titled "Wynik 1" and displays a table with two columns: "imie" and "wyglad". The table lists various names with their corresponding descriptions. A bracket on the left side groups the table area, with the text "Tu są wyświetlane wyniki zapytania." (Here the query results are displayed.) positioned next to it. At the bottom of the window, there's a status bar with the text "10 rows fetched in 0.1477s (0.0110s)" and several navigation buttons: Edycja, Zapisz zmiany, Discard Changes, Pierwszy, Ostatni, and Znajdź.

imie	wyglad
Balbina	K, całka pomarańczowo i w cekinach
Eklerka	K, czerwone włosy, zielona sukienka, ogromne stopy
Gonzo	M, w przebraniu kobiety, kostium w plamki
Klarebela	K, różowe włosy, ogromny kwiat, niebieska sukienka
Pan Hobo	M, cygaro, czarne włosy, niewielki kapelusz
Pan Pimpus	M, pomarańczowe włosy, niebieski garnitur, ogromne stopy
Pan Smyk	M, zielono-fioletowy kostium, szpiczasty nos
Pani Smyk	K, żółta koszula, dzierwone czerwone spodnie
Skuter	M, niebieskie włosy, czerwony garnitur, ogromny nos
Zippo	K, pomarańczowy garnitur, dziurawe spodnie

Inne programy z graficznym interfejsem użytkownika

Dostępnych jest także kilka innych programów z graficznym interfejsem użytkownika, przeznaczonych do obsługi baz danych MySQL. Sam będziesz musiał zdecydować, który z nich najbardziej Ci się podoba. Istnieje także znacznie więcej podobnych programów, o których tu nawet nie wspominaliśmy — bez trudu znajdziesz je za pomocą wyszukiwarki.



Używając tych przycisków, bez najmniejszych problemów możesz wyświetlić strukturę tabeli, przeszukać jej zawartość, bądź zmodyfikować kolumny.

Jeśli potrzebna Ci będzie internetowa aplikacja do zarządzania bazą, powinieneś zainteresować się programem phpMyAdmin. Doskonale sprawdza się on w sytuacjach, gdy korzystasz z konta na serwerze dostawcy internetowego i chcesz zarządzać bazą danych MySQL działającą na tym samym serwerze. Nie jest to jednak najlepsze rozwiązanie w przypadku, gdy serwer MySQL działa na Twoim lokalnym komputerze. Więcej informacji na temat tego produktu znajdziesz na stronie:

<http://www.phpmyadmin.net/>

Poniżej przedstawiliśmy jeszcze kilka popularnych programów do obsługi baz danych MySQL i zarządzania bazami. Niektóre z nich są przeznaczone wyłącznie dla użytkowników komputerów PC; najlepiej zrobisz, jeśli zajrzesz na ich strony, przeczytasz informacje o najnowszych wersjach i sam ocenisz, który z nich najlepiej spełni Twoje wymagania i potrzeby:

Navicat udostępnia 30-dniową wersję testową, którą znajdziesz na stronie:

<http://www.navicat.com/>

Z kolei program SQLyog udostępnia ogólnie dostępną wersję Community Edition; znajdziesz ją na stronie:

<http://www.webyog.com/en/>

Nr 2. Słowa zastrzeżone i znaki specjalne

W języku SQL występuje całkiem dużo słów zastrzeżonych. Najlepiej nie stosować ich w nazwach swoich baz danych, tabel i kolumn. Chociaż mogłoby Ci przyjść do głowy, by nadać nowej tabeli nazwę „select”, to jednak znacznie lepiej zrobisz, jeśli wymyślisz dla niej jakąś bardziej opisową nazwę, a słowa „select” w ogóle nie użyjesz. Jeśli jednak będziesz musiał zastosować jedno z zastrzeżonych słów kluczowych, to dodaj do niego jakieś inne słowo lub znak podkreślenia, by ułatwić systemowi zarządzania bazami danych zrozumienie polecenia. Aby ułatwić Ci życie i przyszłe tworzenie baz danych, na następnej stronie zamieściliśmy pełną listę słów, których raczej nie powinieneś stosować w nazwach tabel i kolumn.

Aby dodatkowo utrudnić całe zagadnienie, musisz wiedzieć, że MySQL posiada listę słów, które na razie nie są zastrzeżone, lecz mogą się takimi stać w przyszłości. Nie będziemy podawać ich tutaj, lecz możesz je znaleźć w podręczniku dotyczącym korzystania z bazy MySQL, który zdecydowanie powinieneś kupić po zakończeniu lektury tej książki.

Znaki specjalne

Poniżej zamieszczona została lista większości znaków specjalnych używanych przez MySQL, wraz z ich znaczeniami. Podobnie jak w przypadku słów zastrzeżonych, także tych znaków specjalnych raczej nie należy używać w nazwach tabel i kolumn. Wyjątkiem od tej reguły jest znak podkreślenia, który powinieneś stosować i do czego zresztą gorąco Cię zachęcamy. Ogólnie rzecz biorąc, najlepiej jest tworzyć nazwy tabel i kolumn wyłącznie w oparciu o litery i znak podkreślenia. Nawet stosowanie cyfr nie jest najlepszym rozwiązaniem, no chyba że użyta liczba sama w sobie niesie informację o przeznaczeniu i zawartości tabeli lub kolumny.

*	Użyty w poleceniu SELECT zwraca wszystkie kolumny tabeli.
()	Pozwalają na grupowanie wyrażeń, określanie kolejności, w jakiej będą wykonywane operacje matematyczne, oraz na wywoływanie funkcji. Oprócz tego wewnętrz nich umieszcza się podzapytania.
;	Kończy każde polecenie SQL.
,	Rozdziela elementy list. Stosuje się je między innymi w poleceniach SELECT oraz klauzulach IN.
.	Używany w odwołaniach do kolumn tabel oraz w liczbach z miejscami dziesiętnymi.
-	To znak wieloznaczny reprezentujący dowolny znak w klauzulach LIKE.
%	Kolejny znak specjalny używany w klauzulach LIKE; może zastępować dowolne łańcuchy znaków.
!	Wykrzyknik oznacza NOT — czyli negację. Jest on stosowany wraz z operatorami logicznymi w klauzulach WHERE.
'	W języku SQL pomiędzy apostrofami zapisywane są łańcuchy znaków.
"	Dokładnie w taki sam sposób możesz używać znaków cudzysłowu, choć uważamy, że zastosowanie apostrofów jest lepszym rozwiązaniem.
\	Ten znak pozwala Ci umieścić znak apostrofu w łańcuchu zapisywanym w polu tekstowym.
+	Oprócz stosowania tego znaku w celu dodawania liczb, można go także używać do łączenia łańcuchów znaków.

A oto krótkie przypomnienie operatorów arytmetycznych:

+	Dodawanie	-	Odejmowanie	*	Ten znak, użyty pomiędzy dwiema wartościami, oznacza mnożenie	/	Dzielenie
---	-----------	---	-------------	---	---	---	-----------

A oto operatory porównania:

>	Większy niż	!>	Nie większy	>=	Większy lub równy
<	Mniejszy niż	!<	Nie mniejszy	<=	Mniejszy lub równy
=	Równy	<>	Różny	!=	Różny

&		^	~
Te operatory nie zostały opisane w niniejszej książce. Zajrzyj do dokumentacji bazy danych, by dowiedzieć się, jakie mają znaczenie.			

Słowa zarezerwowane

Ogólnie rzecz biorąc, warto przejrzeć tę listę, gdy planujesz nadawać tabelom lub kolumnom jednowyrazowe nazwy. W ten sposób będziesz mieć pewność, że nie użyjesz żadnego ze słów zastrzeżonych.

A	ABSOLUTE ACTION ADD ADMIN AFTER AGGREGATE ALIAS ALL ALLOCATE ALTER ANY ARE ARRAY AS ASC ASSERTION AT AUTHORIZATION
B	BEFORE BEGIN BINARY BIT BLOB BOOLEAN BOTH BREADTH BY
C	CALL CASCADE CASCADED CASE CAST CATALOG CHAR CHARACTER CHECK CLASS CLOB CLOSE COLLATE COLLATION COLUMN COMMIT COMPLETION CONNECT CONNECTION CONSTRAINT CONSTRAINTS CONSTRUCTOR CONTINUE CORRESPONDING CREATE CROSS CUBE CURRENT CURRENT_DATE CURRENT_PATH CURRENT_ROLE CURRENT_TIME CURRENT_TIMESTAMP CURRENT_USER CURSOR CYCLE
D	DATA DATE DAY DEALLOCATE DEC DECIMAL DECLARE DEFAULT DEFERRABLE DEFERRED DELETE DEPTH DEREF DESC DESCRIBE DESCRIPTOR DESTROY DESTRUCTOR DETERMINISTIC DICTIONARY DIAGNOSTICS DISCONNECT DISTINCT DOMAIN DOUBLE DROP DYNAMIC
E	EACH ELSE END END_EXEC EQUALS ESCAPE EVERY EXCEPT EXCEPTION EXEC EXECUTE EXTERNAL
F	FALSE FETCH FIRST FLOAT FOR FOREIGN FOUND FROM FREE FULL FUNCTION
G	GENERAL GET GLOBAL GO GOTO GRANT GROUP GROUPING
H	HAVING HOST HOUR
I	IDENTITY IGNORE IMMEDIATE IN INDICATOR INITIALIZE INITIALLY INNER INOUT INPUT INSERT INT INTEGER INTERSECT INTERVAL INTO IS ISOLATION ITERATE
J	JOIN
K	KEY
L	LANGUAGE LARGE LAST LATERAL LEADING LEFT LESS LEVEL LIKE LIMIT LOCAL LOCALTIME LOCALTIMESTAMP LOCATOR
M	MAP MATCH MINUTE MODIFIES MODIFY MODULE MONTH
N	NAMES NATIONAL NATURAL NCHAR NCLOB NEW NEXT NO NONE NOT NULL NUMERIC
O	OBJECT OF OFF OLD ON ONLY OPEN OPERATION OPTION OR ORDER ORDINALITY OUT OUTER OUTPUT
P	PAD PARAMETER PARAMETERS PARTIAL PATH POSTFIX PRECISION PREFIX PREORDER PREPARE PRESERVE PRIMARY PRIOR PRIVILEGES PROCEDURE PUBLIC
Q	
R	READ READS REAL RECURSIVE REF REFERENCES REFERENCING RELATIVE RESTRICT RESULT RETURN RETURNS REVOKE RIGHT ROLE ROLLBACK ROLLUP ROUTINE ROW ROWS
S	SAVEPOINT SCHEMA SCROLL SCOPE SEARCH SECOND SECTION SELECT SEQUENCE SESSION SESSION_USER SET SETS SIZE SMALLINT SOME SPACE SPECIFIC SPECIFICTYPE SQL SQLEXCEPTION SQLSTATE SQLWARNING START STATE STATEMENT STATIC STRUCTURE SYSTEM_USER
T	TABLE TEMPORARY TERMINATE THAN THEN TIME TIMESTAMP TIMEZONE_HOUR TIMEZONE_MINUTE TO TRAILING TRANSACTION TRANSLATION TREAT TRIGGER TRUE
U	UNDER UNION UNIQUE UNKNOWN UNNEST UPDATE USAGE USER USING
V	VALUE VALUES VARCHAR VARIABLE VARYING VIEW
W	WHEN WHENEVER WHERE WITH WITHOUT WORK WRITE
X	
Y	YEAR
Z	ZONE

Nr 3. ALL, ANY oraz SOME

W języku SQL istnieją trzy słowa kluczowe, które są niezwykle przydatne podczas stosowania podzapytań. Są to: ALL, ANY oraz SOME. Można ich używać wraz z operatorami porównania oraz zbiorami wartości. Zanim szczegółowo się nimi zajmiemy, cofnijmy się na chwilę do rozdziału 9. i przypomnijmy informacje o operatorze IN:

```
SELECT nazwa, ocena FROM ranking_restauracji
WHERE ocena IN
(SELECT ocena FROM ranking_restauracji
WHERE ocena > 3 AND ocena < 9);
```

To zapytanie zwraca nazwy i oceny wszystkich restauracji, których ocena jest taka sama jak jedna z wartości zwróconych przez podzapytanie umieszczone w nawiasach. W naszym przypadku zapytanie zwróci nazwy *Coś na żąb* oraz *Ryby i nie tylko*.

Stosowanie ALL

A teraz przeanalizujmy następujące zapytanie:

```
SELECT nazwa, ocena FROM ranking_restauracji
WHERE ocena > ALL
(SELECT ocena FROM ranking_restauracji
WHERE ocena > 3 AND ocena < 9);
```

Tym razem mamy zamiar pobrać nazwy wszystkich restauracji, których ocena jest wyższa od wszystkich ocen zwróconych jako wyniki podzapytania. W naszym przykładzie jedynym wynikiem całego zapytania będzie nazwa *U Artura*.

Dla odmiany w poniższym zapytaniu zastosowano operator <:

```
SELECT nazwa, ocena FROM ranking_restauracji
WHERE ocena < ALL
(SELECT ocena FROM ranking_restauracji
WHERE ocena > 3 AND ocena < 9);
```

Ze słowem kluczowym ALL można także używać operatorów porównania \geq oraz \leq . Poniższe zapytanie zwróci nazwy *Coś na żąb* oraz *U Artura*. Zwraca ono restauracje, które mają oceny wyższe od tych w zbiorze zwróconym przez zapytanie lub równe największej wartości z tego zbioru (czyli wartości 7):

```
SELECT nazwa, ocena FROM ranking_restauracji
WHERE ocena >= ALL
(SELECT ocena FROM ranking_restauracji
WHERE ocena > 3 AND ocena < 9);
```



To podzapytanie zwraca wszystkie oceny z zakresu od 3 do 9. W naszym przykładzie będą to oceny o wartościach 7 i 5.

Większy niż ALL
zwraca wszystkie
wartości większe
od największej
wartości w zbiorze.

Mniejszy niż ALL
zwraca wszystkie
wartości mniejsze
od najmniejszej
wartości w zbiorze.

Dopasowane zostaną wszystkie wartości większe od tych ze zwroconego zbioru bądź równe jego największej wartości.

Stosowanie ANY

Warunek wykorzystujący słowo kluczowe ANY zwróci prawdę logiczną, jeśli spełni go dowolna wartość ze zbioru. Przeanalizuj następujący przykład:

```
SELECT nazwa, ocena FROM ranking_restauracji  
WHERE ocena > ANY  
(SELECT ocena FROM ranking_restauracji  
WHERE ocena > 3 AND ocena < 9);
```

Należy go interpretować w następujący sposób: należy odczytać i zwrócić wszystkie wiersze, w których ocena jest większa od którejkolwiek z wartości ze zbioru (5, 7). Restauracja *Coś na żąb* ma ocenę 7, która jest większa od 5; a zatem zostanie zwrócona. Zwracana jest także restauracja *U Artura*, której ocena wynosi 9.

Stosowanie SOME

W standardowej składni języka SQL SOME oznacza dokładnie to samo co ANY i tak właśnie jest w MySQL-u. Sprawdź w dokumentacji używanej bazy danych, jakie jest znaczenie tego słowa kluczowego oraz czy możesz je stosować.

Większy niż ANY
znajduje wszystkie
wartości większe
od najmniejszej
wartości ze zbioru.

Mniejszy niż ANY
znajduje wszystkie
wartości mniejsze
od największej
wartości zbioru.

Nr 4. Dodatkowe informacje o typach danych

Już znasz najczęściej stosowane typy danych, jest jednak kilka szczegółów, które pozwolą Ci w jeszcze większym stopniu dopracować kolumny w tworzonych tabelach. Przyjrzyjmy się nieco dokładniej kilku nowym typom danych oraz kilku typom, które już znasz i które stosowałeś w tej książce.

BOOLEAN

W kolumnach tego typu można przechowywać wartości logiczne „prawda” (`true`) lub „fałsz” (`false`), ewentualnie kolumna taka może zawierać `NULL`. Ten typ danych doskonale nadaje się do tworzenia kolumn, które mogą zawierać wartości logiczne „prawda” lub „fałsz”. W niewidocznym dla nas sposobie system zarządzania relacyjną bazą danych zapisuje w tabeli wartość 1, odpowiadającą logicznej „prawdzie”, oraz wartość 0, odpowiadającą logicznemu „fałszowi”. Równie dobrze zamiast wartości `true` i `false` w kolumnie tego typu możesz zapisywać wartości 1 i 0.

INT

Typu INT często używaliśmy w tej książce. W kolumnach typu INT można przechowywać liczby całkowite z zakresu **od 0 do 4294967295**. Ten zakres jest dostępny tylko wtedy, gdy chcesz używać liczb dodatnich, w takim przypadku mówimy o **liczbach całkowitych bez znaku**.

Jeśli w kolumnie chcesz przechowywać dodatnie i ujemne liczby całkowite, to musisz zażądać użycia liczb ze znakiem. W takim przypadku będziesz miał dostępne liczby z zakresu od -2147483648 do 2147483647. Aby utworzyć kolumnę typu INT pozwalającą na zapisywanie liczb ze znakiem, musisz tego jawnie zażądać od systemu zarządzania relacyjnymi bazami danych, używając zapisu o postaci:

INT(SIGNED)

Inne typy liczb całkowitych

Już znasz typ INT, jednak dostępne zakresy liczb całkowitych poszerzają dwa kolejne typy: SMALLINT oraz BIGINT. Różnią się one maksymalną wartością, jaką można zapisać w kolumnie.

Zakres liczb, jaki można zapisywać w kolumnach tych typów, zmienia się w zależności od używanego systemu zarządzania relacyjnymi bazami danych. W przypadku MySQL-a wartości te wyglądają następująco:

	Liczby ze znakiem	Liczby bez znaku
TINYINT	od -32768 do 32767	od 0 do 65535
MEDIUMINT	od -9223372036854775808 do 9223372036854775807	od 0 do 18446744073709551615

W MySQL-u ilość dostępnych typów liczb całkowitych jest jeszcze większa — do przedstawionych powyżej dodano dwa dodatkowe: TINYINT oraz MEDIUMINT:

	Liczby ze znakiem	Liczby bez znaku
SMALLINT	od -128 do 127	od 0 do 255
BIGINT	od -8388608 do 8388607	od 0 do 16777215

Typy dat i czasu

Poniżej opisaliśmy pokrótce format, w jakim MySQL zapisuje daty i godziny:

DATE	RRRR-MM-DD
DATETIME	RRRR-MM-DD GG:MM:SS
TIMESTAMP	RRRRMMDDGGMMSS
TIME	GG:MM:SS

rozne_daty	
	pewna_data
	2007-08-25 22:10:00
	1925-01-01 02:05:00

Pobierając poleceniem SELECT wartości dat lub czasu, można określić, w jakiej postaci baza danych je zwróci. W różnych systemach zarządzania relacyjnymi bazami danych dostępne są różne funkcje formatujące. Poniżej przedstawiliśmy przykład funkcji DATE_FORMAT() stosowanej w bazie MySQL.

Założmy, że dysponujemy kolumną o nazwie `pewna_data`:

```
SELECT DATE_FORMAT(pewna_data, '%M %Y') FROM rozne_daty;
```

Wyrażenia %M oraz %Y przekazują funkcji informacje o tym, w jaki sposób daty mają zostać sformatowane. Oto jak powinny wyglądać wyniki powyższego zapytania:

rozne_daty	
	pewna_data
	August 2007
	January 1925

Niestety, w tym dodatku nie mamy wystarczająco dużo miejsca, by przedstawić wszystkie dostępne opcje i funkcje związane z formatowaniem dat — jest ich bowiem tak dużo. Jednak dzięki nim będziesz w stanie pobrać z wartości dat i czasu wszystkie potrzebne Ci informacje i to bez wyświetlania innych, niepotrzebnych danych.

Nr 5. Tabele tymczasowe

W tej książce utworzyliśmy naprawdę dużo tabel. Zawsze gdy tworzymy tabelę, system zarządzania relacyjną bazą danych zapisuje jej strukturę. Za każdym razem, gdy wstawiamy nowy wiersz do tabeli, jego dane są zapisywane. Tabela, jak i umieszczone w niej dane są zapisywane na stałe. Kiedy zakończymy sesję z bazą danych i zamknijemy okno terminalu lub program graficzny, dane, na których operowaliśmy, wciąż istnieją w bazie. I będą w niej istniały aż do momentu, gdy je usuniemy; podobnie tabela będzie istnieć tak długo, aż ją usuniemy.

SQL udostępnia jednak także i inny typ tabel, tak zwane *tabele tymczasowe*. Tabele te istnieją od momentu utworzenia, aż do chwili usunięcia *bądź też do momentu zakończenia sesji z bazą danych*. Poprzez „sesję” rozumiemy czas, w którym jesteś zalogowany na serwerze bazy danych i używasz okna konsoli lub graficznego programu do obsługi bazy. Oczywiście taką tymczasową tabelę można także jawnie usunąć poleceniem DROP.

Dlaczego tabele tymczasowe mogą być przydatne:

- Można ich używać do przechowywania wyników pośrednich — na przykład w takiej tabeli można zapisać wyniki obliczeń matematycznych, które będą używane później podczas trwania tej samej sesji, lecz nie w kolejnej sesji.
- Dzięki nim można zapamiętać zawartość tabeli w konkretnej chwili.
- Czy pamiętasz moment transformacji, kiedy przekształcaliśmy oryginalną bazę Grześka na bazę składającą się z wielu tabel? W takich momentach można utworzyć tabele tymczasowe, by ułatwić sobie restrukturyzację danych, mając jednocześnie pewność, że zostaną one usunięte po zakończeniu sesji.
- Kiedy w końcu zaczniesz używać SQL-a w jakimś języku programowania, to z powodzeniem możesz zastosować tabele tymczasowe do gromadzenia danych częściowych, a dopiero później, w trwałej tabeli, zapisać wyniki ostateczne.

Tworzenie tabeli tymczasowej

Składnia polecenia służącego do tworzenia tabel tymczasowych jest całkiem prosta — do zwyczajnego polecenia CREATE wystarczy dodać słowo kluczowe TEMPORARY:

```
CREATE TEMPORARY TABLE moja_tabela_tymczasowa  
(  
    jakieś_id INT,  
    jakas_wartosc VARCHAR(50)  
);
```

Jedynym elementem polecenia, jaki musimy dodać, by utworzyć tabelę tymczasową, jest słowo kluczowe TEMPORARY.



Uwaga!

W poszczególnych systemach zarządzania relacyjnymi bazami danych występują bardzo duże różnice w składni poleceń służących do tworzenia tabel tymczasowych.

Nie zapomnij zatem sprawdzić postaci polecień w dokumentacji używanej bazy danych.

Uproszczony sposób tworzenia tabel tymczasowych

Poniżej pokazaliśmy, jak można utworzyć tabelę tymczasową i jednocześnie zapisać w niej wyniki wykonania zapytania:

```
CREATE TEMPORARY TABLE moja_tabela_tymczasowa AS  
SELECT * FROM moja_tabela_trwala;
```

Za słowem kluczowym AS można umieścić dowolne polecenie SELECT.

Nr 6. Rzutowanie typów

Czasami zdarza się, że dane są zapisane w tabeli w kolumnie jednego typu, a chcesz, by w wynikach zapytania pojawiły się dane innego typu. SQL udostępnia funkcję o nazwie **CAST()**, która pozwala skonwertować dane do wskazanego typu.

Poniżej przedstawiliśmy składnię tej funkcji:

CAST(kolumna AS TYP)

Gdzie TYP może przybierać następujące wartości:

CHAR()
DATE
DATETIME
DECIMAL
SIGNED [INTEGER]
TIME
UNSIGNED [INTEGER]

Kilka sytuacji, w których funkcja **CAST()** może Ci się przydać:

Konwersja łańcucha znaków zawierającego datę do typu DATE:

SELECT CAST('2005-01-01' AS DATE);

← Łącuch znaków '2005-01-01' zostanie przekształcony na wartość typu DATE.

Konwersja liczby całkowitej na wartość typu DECIMAL:

SELECT CAST(2 AS DECIMAL);

← Liczba całkowita 2 staje się wartością 2.00 typu DECIMAL.

Funkcji **CAST()** można także używać na liście wartości w poleceniach **INSERT** oraz na liście kolumn w poleceniach **SELECT**.

Oto sytuacje, w których nie można używać funkcji **CAST()**:

- Konwersja wartości typu DECIMAL na liczbę całkowitą.
- Konwersja wartości typów TIME, DATE, DATETIME oraz CHAR do wartości typu DECIMAL lub INTEGER.

Jednak funkcji tej można używać na liście wartości w poleceniach **INSERT** oraz na liście kolumn w poleceniach **SELECT**.

Nr 7. Kim jesteś? Która jest godzina?

Może się zdarzyć, że w używanej przez Ciebie bazie danych zostanie utworzonych więcej niż jedno konto użytkownika i każde z nich będzie mieć inne uprawnienia. Jeśli w takim przypadku będziesz chciał się dowiedzieć, jakiego konta aktualnie używasz, możesz to zrobić za pomocą następującego polecenia:

```
SELECT CURRENT_USER;
```

Polecenie to zwróci także informacje o aktualnie używanym komputerze. Jeśli system zarządzania relacyjnymi bazami danych został zainstalowany na tym samym komputerze, którego używasz do pracy, oraz jeśli używasz konta administratora (użytkownika root), to wykonanie powyższego polecenia zwróci następujące wyniki:

```
root@localhost
```

Poniżej przedstawiliśmy polecenia, które pozwolą Ci uzyskać informacje o aktualnej godzinie i czasie:

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT CURRENT_DATE;
+-----+
| CURRENT_DATE |
+-----+
| 2009-01-13 |
+-----+
1 row in set (0.01 sec)

mysql>
```

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT CURRENT_TIME;
+-----+
| CURRENT_TIME |
+-----+
| 13:06:46 |
+-----+
1 row in set (0.01 sec)

mysql>
```

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> SELECT CURRENT_USER;
+-----+
| CURRENT_USER |
+-----+
| root@localhost |
+-----+
1 row in set (0.00 sec)

mysql>
```

Nr 8. Przydatne funkcje matematyczne

Poniżej zamieściliśmy krótkie zestawienie funkcji, których można używać do wykonywania różnego rodzaju operacji na wartościach liczbowych.

Niektóre z nich poznałeś już we wcześniejszej części książki:

Funkcja	Działanie	
ABS(x)	Zwraca wartość bezwzględną liczby x.	
	zapytanie	wyniki
	SELECT ABS(-23);	23
ACOS(x)	Zwraca arcus cosinus liczby x.	
	SELECT ACOS(0);	1.5707963267949
ASIN(x)	Zwraca arcus sinus liczby x.	
	SELECT ASIN(0.1);	0.10016742116156
ATAN(x,y)	Zwraca arcus tangens liczb x i y.	
	SELECT ATAN(-2,2);	-0.78539816339745
CEIL(x)	Zwraca najmniejszą liczbę całkowitą, większą lub równą wartości x. Zwrócona wartość będzie typu BIGINT.	
	SELECT CEIL(1.32);	2
	Zwraca cosinus liczby x.	
COS(x)	SELECT COS(1);	0.54030230586814
COT(x)	Zwraca cotangens liczby x.	
	SELECT COT(12);	-1.5726734063977
EXP(x)	Zwraca wartość liczby e podniesionej do potęgi x.	
	SELECT EXP(-2);	0.13533528323661
FLOOR(x)	Zwraca największą liczbę całkowitą mniejszą lub równą wartości x.	
	SELECT FLOOR(1.32);	1
FORMAT(x,y)	Formatuje wartość x do postaci łańcucha znaków przedstawiającego liczbę zaokrągloną do y miejsc po przecinku.	
	SELECT FORMAT(3452100.50,2);	3,452,100.50
LN(x)	Zwraca logarytm naturalny liczby x.	
	SELECT LN(2);	0.69314718055995
LOG(x) oraz LOG(x,y)	Zwraca logarytm naturalny liczby x bądź też w przypadku wersji z dwoma argumentami zwraca logarytm o podstawie y z liczby x.	
	SELECT LOG(2);	0.69314718055995
	SELECT LOG(2, 65536);	16

→ dokończenie na następnej stronie

Nr 8. Przydatne funkcje matematyczne (ciąg dalszy)

Funkcja	Działanie	
MOD(x,y)	Zwraca część ułamkową z dzielenia liczby x przez y.	
	zapytanie	wyniki
	SELECT MOD(249,10);	9
PI()	Zwraca wartość liczby pi.	
	SELECT PI();	3.141593
POWER(x,y)	Zwraca wartość liczby x podniesionej do potęgi y.	
	SELECT POW(3,2);	9
RADIANS(x)	Zwraca wartość x skonwertowaną ze stopni na radiany.	
	SELECT RADIANS(45);	0.78539816339745
RAND()	Zwraca losową wartość zmienoprzecinkową.	
	SELECT RAND();	0.82362382686237
ROUND(x)	Zwraca wartość x zaokrągloną do najbliższej liczby całkowitej.	
	SELECT ROUND(1.34);	1
	SELECT ROUND(-1.34);	-1
ROUND(x,y)	Zwraca wartość x zaokrągloną do y miejsc po przecinku.	
	SELECT ROUND(1.456, 1);	1.5
	SELECT ROUND(1.456, 0);	1
	SELECT ROUND(28.367, -1);	30
SIGN(x)	Zwraca wartość 1, jeśli liczba x jest dodatnia, lub -1, jeśli jest ujemna.	
	SELECT SIGN(-23);	-1;
SIN(x)	Zwraca sinus liczby x.	
	SELECT SIN(PI());	1.2246063538224e-16
SQRT(x)	Oblicza pierwiastek kwadratowy liczby x.	
	SELECT SQRT(100);	10
TAN(x)	Oblicza tangens liczby x.	
	SELECT TAN(PI());	-1.2246063538224e-16
TRUNCATE(x,y)	Zwraca wartość x skróconą do y miejsc po przecinku dziesiętnym.	
	SELECT TRUNCATE(8.923,1);	8.9

Nr 9. Indeksowanie dla poprawy szybkości działania zapytań

Już wiesz wszystko na temat klucza głównego oraz klucza obcego. Te typy indeksów doskonale nadają się do łączenia ze sobą tabel oraz do wymuszania integralności danych. Jednak można także tworzyć indeksy dotyczące wskazanych kolumn, aby przyspieszyć wykonywanie zapytań.

Kiedy warunki z klauzuli WHERE operują na kolumnie, dla której nie utworzono indeksu, system zarządzania relacyjnymi bazami danych zaczyna od samego początku takiej kolumny, a następnie analizuje jej kolejne wiersze. Jeśli Twoja tabela jest naprawdę olbrzymia, a mówiąc „olbrzymia” mamy ma myślis co najmniej 4 miliony rekordów, to wykonanie takiej operacji może potrwać zauważalną chwilę.

Z kolei w przypadku utworzenia indeksu na kolumnie system zarządzania relacyjnymi bazami danych przechowuje dodatkowe informacje o tej kolumnie, które są w stanie niesamowicie przyspieszyć wyszukiwanie. Te dodatkowe informacje są przechowywane w dodatkowej, niewidocznej tabeli, a system zarządzania bazami danych zapisuje je w określonej kolejności, która pozwala na szybsze przeszukiwanie. Oczywiście ten zysk szybkości wymaga pewnego kompromisu — jest nim utrata miejsca na dysku. A zatem musisz się dobrze zastanowić, zanim zindeksujesz kolumny — warto zindeksować jedynie te kolumny, które będziesz często przeszukiwać.

Poniżej przedstawiliśmy kod polecenia ALTER, umożliwiający dodanie indeksu na kolumnie:

```
ALTER TABLE moje_kontakty  
ADD INDEX (nazwisko);
```

Oczywiście teoria dotycząca indeksów jest znacznie bardziej rozbudowana, ale tak by to wyglądało w najprostszym ujęciu.

Nr 10. Dwuminutowy kurs PHP i MySQL-a

Zanim skończymy, chcieliśmy przedstawić bardzo krótki przykład ilustrujący, jak PHP i MySQL mogą ze sobą współpracować, by ułatwić Ci opublikowanie Twoich danych w internecie. Poniższy przykład stanowi jedynie przedsmak tego, co można robić, korzystając z obu tych technologii, i niewątpliwie powinieneś poszukać dodatkowych informacji na ten temat.

Przedstawiając poniższy przykład, zakładamy, że dysponujesz już podstawową znajomością języka PHP. Wiemy także, że doskonale potrafisz pisać zapytania SQL. Poniższy skrypt nawiązuje połączenie z bazą danych `lista_grzesia` i pobiera wszystkie imiona i nazwiska z tabeli `moje_kontakty`. Kod PHP odczytuje wszystkie wiersze z bazy danych i zapisuje je w tabeli. Ostatni fragment skryptu wyświetla je na stronie WWW:

```
<?php

$conn = mysql_connect("localhost", "grzesiek", "gr3gzpAs");
if (!$conn)
{
    die('Nie udało się nawiązać połączenia: ' . mysql_error());
}

mysql_select_db("lista_grzesia", $conn);

$result = mysql_query("SELECT imie, nazwisko FROM moje_kontakty");

while($row = mysql_fetch_array($result))
{
    echo $row['nazwisko'] . " " . $row['imie'];
    echo "<br />";
}

mysql_close($conn);
?>
```

Zapiszemy ten kod na serwerze WWW, w pliku o nazwie `listagreska.php`.

Dokładniejsza analiza kodu

```
<?php
```

Ten wiersz informuje serwer WWW, że zaczyna się kod PHP.

```
$conn = mysql_connect("localhost", "grzesiek", "gr3gzpAs");
```

Aby nawiązać połączenie z bazą danych `lista_grzesia`, musimy poinformować serwer WWW, gdzie znajduje się system zarządzania relacyjnymi bazami danych, jaka jest nasza nazwa użytkownika oraz hasło dostępu. Informacje te, zapisane w postaciłańcuchów znaków, przekazujemy w wywołaniu funkcji, której wyniki wykonania zapisujemy następnie w zmiennej `$conn`. Funkcja PHP `mysql_connect()` sprawdza, czy jest w stanie nawiązać połączenie ze wskazanym serwerem bazy danych.

```
if (!$conn)
{
    die('Nie udało się nawiązać połączenia: ' . mysql_error());
}
```

Jeśli nie uda się nawiązać połączenia, kod wyświetli komunikat informujący nas o przyczynach problemów i na tym działanie skryptu się zakończy.

```
mysql_select_db("lista_grzesia", $conn);
```

No dobrze, zatem udało się nawiązać połączenie z serwerem bazy danych. Teraz musimy poinformować PHP, jaka baza danych nas interesuje. Oczywiście chcemy skorzystać z naszej ulubionej bazy `lista_grzesia`.

```
$result = mysql_query("SELECT imie, nazwisko FROM moje_kontakty");
```

A zatem jesteśmy połączeni z serwerem i wybraliśmy już bazę, nie wykonaliśmy jednak jeszcze żadnego zapytania. Piszymy więc zapytanie i korzystamy z funkcji `mysql_query()`, żeby przesłać je do bazy danych. Wszystkie zwrócone wiersze są zapisywane w tablicy o nazwie `$result`.

```
while($row = mysql_fetch_array($result))
{
```

Teraz korzystamy z instrukcji PHP, by pobrać te wszystkie wiersze z tablicy i wyświetlić na stronie WWW. Używamy w tym celu pętli `while`, która pobiera kolejne wiersze, aż dotrze do samego końca tablicy.

```
echo $row['nazwisko'] . " " . $row['imie'];
echo "<br />";
}
```

Te dwie instrukcje PHP wyświetlają na stronie WWW imię i nazwisko pobrane z bazy. Pomiędzy poszczególnymi wierszami tekstu umieszczałyśmy znacznik `
`.

```
mysql_close($conn);
```

Kiedy zakończymy wypisywanie imion i nazwisk z bazy, możemy zamknąć połączenie z systemem zarządzania relacyjnymi bazami danych. Operacja ta jest odpowiednikiem wylogowania się w oknie konsoli.

```
?>
```

I kończymy skrypt PHP.

Dodatek B Instalacja MySQL-a

Spróbuj to zrobić sam



Czy ktokolwiek wiedział,
że tam na dole jest cały,
działający system zarządzania
relacyjnymi bazami danych?
Być może już nigdy nie wyjdę
tu na górę.

Cała zdobyta przez Ciebie wiedza i umiejętności nie na wiele się zdadzą, jeśli nie wykorzystasz ich w praktyce. W tym dodatku znajdziesz instrukcje dotyczące instalacji swojego własnego serwera MySQL, którego będziesz mógł używać do zabawy i pracy.

Zacznij działać i to szybko!

Ponieważ samo posiadanie książki o języku SQL, bez możliwości samodzielnego pisania i wykonywania zapytań, nie jest ani twórcze, ani zabawne, zatem w tym dodatku znajdziesz krótkie instrukcje dotyczące instalacji serwera MySQL na komputerach PC z systemem Windows.

Informacje zamieszczone w niniejszym dodatku dotyczą systemów operacyjnych Windows 2000, XP oraz Windows Server 2003 oraz innych, 32-bitowych wersji systemu operacyjnego Windows.

A teraz opiszemy proces pobierania i instalacji serwera MySQL. Oficjalnie oprogramowanie systemu zarządzania relacyjnymi bazami danych MySQL nosi obecnie nazwę **MySQL Community Server**.

Instrukcje i rozwiązywanie problemów

Na kolejnych kilku stronach zamieściliśmy listę czynności, jakie należy wykonać, by zainstalować serwer MySQL w systemie Windows. Dodatek ten *nie ma* bynajmniej zastąpić doskonałych instrukcji dostępnych na witrynie MySQL, dlatego też gorąco Cię zachęcamy, byś **zajrzał na tę witrynę i przeczytał zamieszczone tam informacje o instalacji serwera!** Szczegółowe informacje o instalacji i rozwiązywaniu ewentualnych problemów możesz znaleźć na stronie:

 *Pobierz wersję 5.0 lub nowszą!*

<http://dev.mysql.com/doc/refman/5.0/en/windows-installation.html>

Na pewno spodoba Ci się także program MySQL Query Browser, o którym wspominaliśmy na stronach 552-553. Pozwala on na wygodne wpisywanie i wykonywanie zapytań oraz przeglądanie ich wyników w wygodnym graficznym interfejsie użytkownika, a nie w tekstowym oknie konsoli.

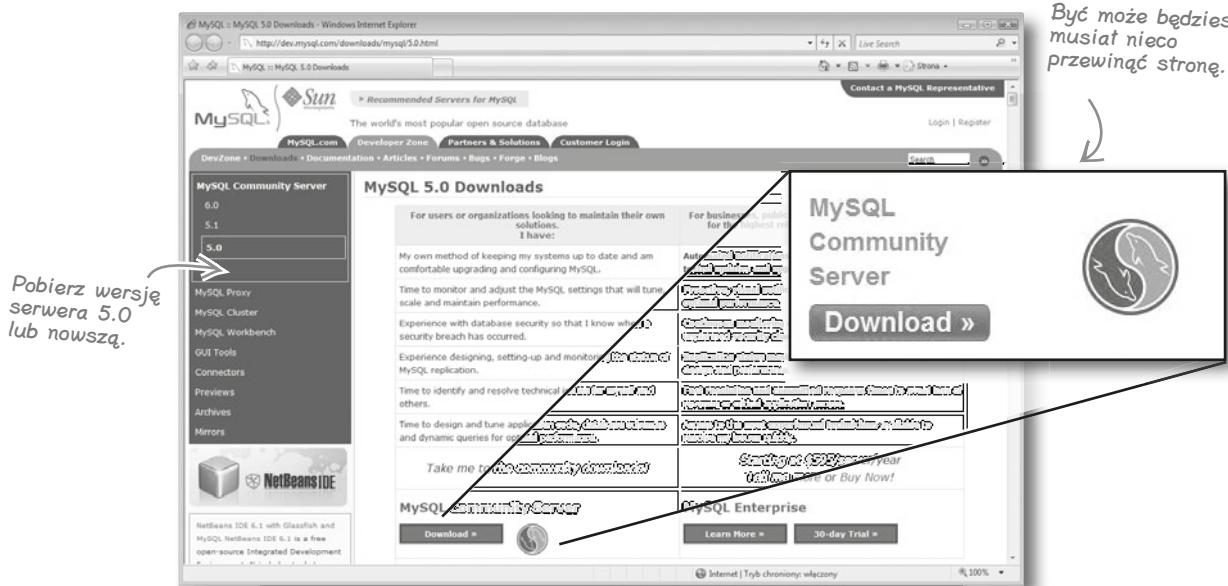
Proces instalacji MySQL-a w systemie Windows

- 1** W przeglądarce wyświetl stronę

<http://dev.mysql.com/downloads/mysql/5.0.html>

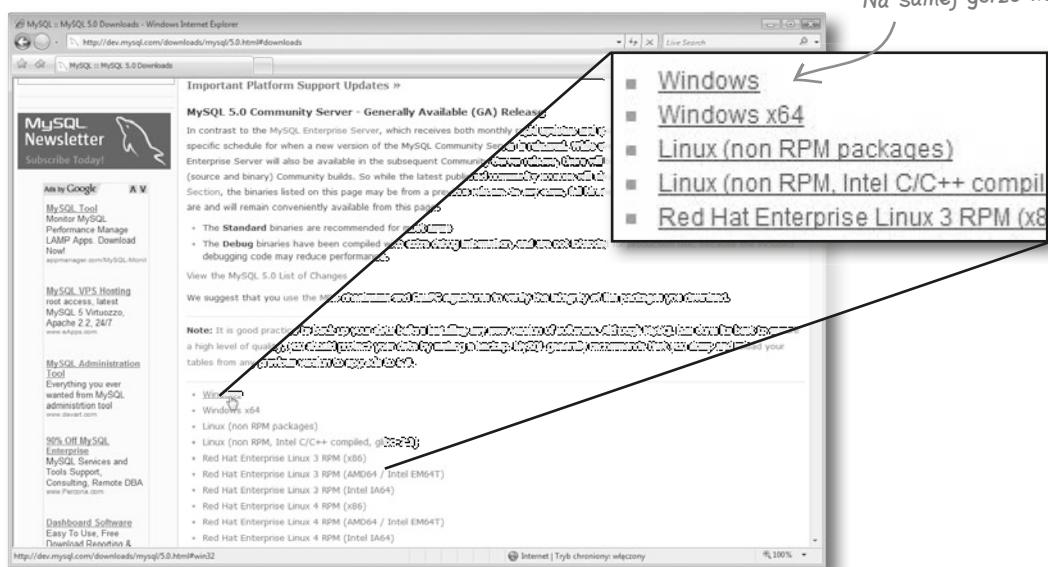
i kliknij przycisk **Download** poniżej napisu MySQL Community Server.

Być może będziesz musiać nieco przewijać stronę.



- 2** Z listy wersji wybierz opcję **Windows**.

Na samej górze listy!

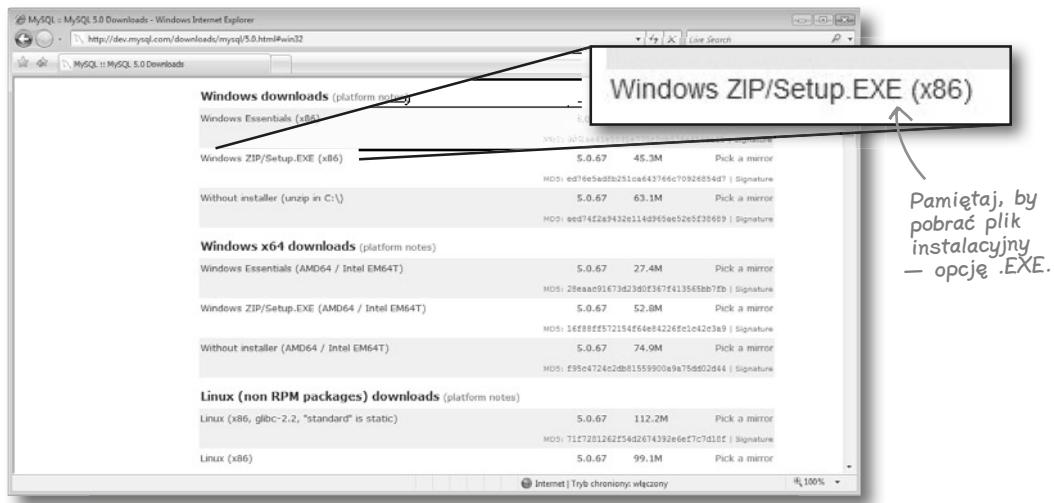


Instalacja MySQL-a w systemie Windows

Pobierz program instalacyjny

3

Sugerujemy, by w dziale plików do pobrania przeznaczonych dla systemu Windows skorzystać z opcji Windows ZIP/Setup.EXE. Kryje się pod nią program instalacyjny, który w ogromnym stopniu ułatwia instalację serwera. Aby pobrać program, kliknij łącze **Pick a Mirror**.



4

W oknie przeglądarki zostanie wyświetlona lista serwerów, z których możesz pobrać plik instalacyjny. Wybierz serwer, który jest położony najbliżej Ciebie.

5

Kiedy program instalacyjny zostanie w całości pobrany, kliknij go dwukrotnie, by go uruchomić. Na ekranie pojawi się kreator **Setup Wizard**, który przeprowadzi Cię przez proces instalacji serwera. Kliknij przycisk **Next**.



Po dwukrotnym kliknięciu pobranego pliku na ekranie pojawi się okno dialogowe kreatora instalacji — **Setup Wizard**. Kliknij w nim przycisk **Next**.

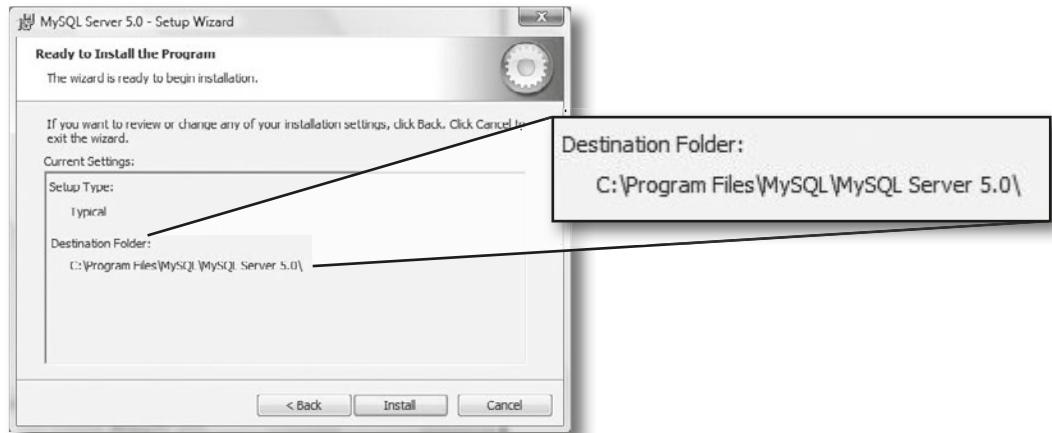
Wybierz katalog docelowy

- 6 Zostaniesz poproszony o wybór typu instalacji: *Typical* (typowa), *Complete* (pełna) lub *Custom* (niestandardowa). Do naszych celów doskonale nadaje się instalacja typowa — **Typical**.

Choć możesz zmienić katalog w komputerze, w którym zostanie zainstalowany serwer bazy danych, to jednak sugerujemy, byś nie zmieniał domyślnej lokalizacji:

C:\Program Files\MySQL\MySQL Server 5.0

Następnie kliknij przycisk **Next**.



Kliknij przycisk **Install** i poczekaj na zakończenie!

- 7 Na ekranie pojawi się okno dialogowe *Ready to Install* (gotowy do instalacji), w którym będzie wyświetlony katalog, w jakim zostanie zainstalowany serwer MySQL. Jeśli ta lokalizacja Ci odpowiada, kliknij przycisk **Install**. W przeciwnym razie kliknij przycisk **Back**, **Change**, następnie wybierz katalog i wróć do tego okna dialogowego.

Kliknij przycisk **Install**.

Dodatek C Przypomnienie narzędzi



Wszystkie nowe narzędzia SQL



Twoje nowe narzędzia SQL
Nauczyłeś się mnie, skarbie!

Kolejne narzędzia SQL
Nas też się nauczyłeś!

Jeszcze następne narzędzia SQL
Niesamowite – poznasz nas
wszystkie!

W tym dodatku, w jednym jedynym miejscu, zostały po raz pierwszy zebrane wszystkie podpowiedzi dotyczące SQL-a... ale będą tu tylko przez jedną noc (żartujemy)! Znajdziesz tu podsumowanie wszystkich porad i informacji o SQL-u, jakie zamieściliśmy w całej książce. Poświeć chwilę, by przejrzeć listę, i rozkoszuj się poczuciem dumy — już znasz wszystkie te rady i informacje.

Symbole

= <> < > <= >=

Mamy do dyspozycji całą grupę operatorów porównania.

Rozdział 2.

A

ALTER z klauzulą CHANGE

Pozwala zmienić zarówno nazwę, jak i typ danych istniejącej kolumny tabeli.

Rozdział 5.

ALTER z klauzulą MODIFY

Pozwala zmienić jedynie typ danych istniejącej kolumny tabeli.

Rozdział 5.

ALTER z klauzulą ADD

Pozwala dodać do tabeli kolumnę w wybranym miejscu.

Rozdział 5.

ALTER z klauzulą DROP

Pozwala usunąć kolumnę tabeli.

Rozdział 5.

ALTER TABLE

Pozwala zmienić nazwę tabeli oraz całą jej strukturę, pozostawiając jednocześnie zapisane w niej dane.

Rozdział 5.

AND oraz OR

Dzięki operatorom AND i OR możesz łączyć warunki logiczne umieszczane w klauzuli WHERE i precyzyjniej wybierać dane z tabeli.

Rozdział 2.

AUTO_INCREMENT

Jeśli to stowarzyszenie kluczowe zostanie użyte w deklaracji kolumny, to podczas wykonywania każdego polecenia INSERT w kolumnie tej zostanie zapisana unikalna liczba całkowita.

Rozdział 4.

AVG

Zwraca średnią wyliczoną na podstawie wartości kolumny liczbowej.

Rozdział 6.

B

BETWEEN

Pozwala wybrać zakres wartości.

Rozdział 2.

C

CHECK - OGRANICZENIE SPRAWDZAJĄCE

Używaj tego ograniczenia, by zezwolić na zapisywanie w kolumnie wyłącznie ścisłe określonych wartości.

Rozdział 11.

CHECK OPTION

Korzystaj z tej klauzuli podczas tworzenia widoków umożliwiających aktualizację danych, by zagwarantować, że dane podawane w poleceniach INSERT i UPDATE nie zostaną zapisane w bazie, jeśli nie spełnią warunku podanego w klauzuli WHERE.

Rozdział 11.

COUNT

Potrafi zwrócić informację o ilości wierszy, jakie może zwrócić zapytanie SELECT, bez konieczności pobierania tych danych. COUNT zwraca jedną liczbę całkowitą.

Rozdział 6.

CREATE TABLE

Rozpoczyna proces tworzenia tabeli, którego kontynuacja będzie dodatkowo wymagała znajomości NAZW KOLUMN oraz TYPÓW DANYCH. Powinieneś opracować strukturę bazy, analizując informacje, które będą zapisywane w jej tabelach.

Rozdział 1.

CREATE TABLE AS

Możesz skorzystać z tego polecenia, by utworzyć tabelę na podstawie wyników zapytania.

Rozdział 10.

CREATE USER

Polecenie używane przez niektóre systemy zarządzania relacyjnymi bazami

danych do tworzenia użytkowników i określania ich haseł.

Rozdział 12.

CROSS JOIN

Zwraca każdy wiersz z jednej tabeli połączony z każdym wierszem z drugiej tabeli. To złączenie ma także kilka innych nazw, takich jak: iloczyn kartezjański bądź też złączenie bez złączenia.

Rozdział 8.

D

DANE ATOMOWE

Dane w tabeli są atomowe, jeśli podzieliłeś je na najmniejsze logiczne części, jakie są Ci potrzebne.

Rozdział 4.

DANE ATOMOWE – REGUŁA 1.

Dane atomowe nie mogą w jednej kolumnie zawierać kilku elementów danych tego samego typu.

Rozdział 4.

DANE ATOMOWE – REGUŁA 2.

Jeśli dane w tabeli mają być atomowe, to tabela ta nie może zawierać kilku kolumn z informacjami tego samego typu.

Rozdział 4.

DELETE

To narzędzie służące do usuwania wierszy danych z tabel. Można używać tego polecenia wraz z klauzulą WHERE,

by bardzo precyjnie wybrać usuwane rekordy.

Rozdział 3.

DISTINCT

Zwraca każdą unikalną wartość tabeli tylko jeden raz — bez powtórzeń.

Rozdział 6.

DROP TABLE

Polecenie pozwala usuwać tabele, jeśli pojawiły się w nich jakiś błąd. Należy pamiętać, że polecenie to można bezpiecznie wykonywać wyłącznie w sytuacjach, kiedy jeszcze nie zapisaliśmy niczego w bazie, postugując się przy tym poleceniem INSERT.

Rozdział 1.

Druga postać normalna (2NF)

Aby tabela była w drugiej postaci normalnej, musi być w pierwszej postaci normalnej i nie może zawierać żadnych częściowych zależności funkcjonalnych.

Rozdział 7.

E

EXCEPT

Stosuj to słowo kluczowe, by zwracać wartości, które są w pierwszym podzapytaniu, LECZ NIE MA ich w drugim.

Rozdział 10.

F

funkcje łańcuchowe pozwalają modyfikować kopie zawartości kolumn łańcuchowych w ramach wykonywanego zapytania. Oryginalne wartości nie zostają w żaden sposób zmodyfikowane.

Rozdział 5.

G

GRANT

To polecenie pozwala przypisywać użytkownikom uprawnienia określające, jakie operacje będą mogli wykonywać na poszczególnych tabelach i kolumnach.

Rozdział 12.

GROUP BY

Scala dane na podstawie wskazanej kolumny.

Rozdział 6.

I

INNER JOIN

Dowolne złączenie, które łączy rekordy z dwóch tabel, używając przy tym pewnego warunku.

Rozdział 8.

INTERSECT

Zwraca tylko te wartości, które są zarówno w wynikach jednego zapytania, JAK I drugiego.

Rozdział 10.

IS NULL

Używaj tego operatora, by tworzyć warunki, które sprawdzają występowanie tej nieznośnej wartości NULL.

Rozdział 2.

J

Jeden-do-jednego

Jednemu wierszowi z tabeli nadzędnej odpowiada dokładnie jeden wiersz z tabeli podrzędnej.

Rozdział 7.

Jeden-do-wielu

Jednemu wierszowi w pierwszej tabeli może odpowiadać wiele wierszy w drugiej tabeli, jednak każdemu wierszowi z drugiej tabeli może odpowiadać tylko jeden wiersz z pierwszej.

Rozdział 7.

K

KLUCZ GŁÓWNY

Kolumna lub grupa kolumn, które w unikalny sposób identyfikują wiersze tabeli.

Rozdział 4.

KLUCZ OBCY

Kolumna tabeli, która odwołuje się do klucza głównego innej tabeli.

Rozdział 7.

KLUCZ OBCY ODWOŁUJĄCY SIĘ DO TEJ SAMEJ TABELI

To klucz obcy umieszczony w tej samej tabeli co klucz główny, lecz używany do innych celów.

Rozdział 10.

Klucz złożony

To klucz główny składający się z wielu kolumn, które łącznie tworzą złożoną i jednocześnie unikalną wartość klucza.

Rozdział 7.

L

LEWOSTRONNE ZŁĄCZENIE ZEWNĘTRZNE

LEWOSTRONNE ZŁĄCZENIE

ZEWNĘTRZNE pobiera wszystkie wiersze z lewej tabeli i jedynie pasujące do nich wiersze z prawej tabeli.

Rozdział 10.

LIKE i znaki % oraz _

Stosuj operator LIKE ze znakami wieloznaczonymi, by poszukiwać podanych fragmentówłańcuchów znaków.

Rozdział 2.

LIMIT

Pozwala określić, jak wiele wierszy ma zwrócić zapytanie, oraz od którego wiersza należy rozpocząć zwracanie wyników.

Rozdział 6.

M

MAX i MIN

Pierwsza z tych funkcji zwraca największą wartość w kolumnie, a druga — wartość najmniejszą.

Rozdział 6.

N

NOT

Operator NOT pozwala zanegować wyniki i uzyskać wartość przeciwną.

Rozdział 2.

NULL oraz NOT NULL

Powinieneś wiedzieć, w których kolumnach nie należy zapisywać wartości NULL, by ułatwić sobie w ten sposób sortowanie i wyszukiwanie danych. Podczas tworzenia tabeli takie kolumny należy zdefiniować jako NOT NULL.

Rozdział 1.

O

ORDER BY

Sortuje wyniki alfabetycznie na podstawie wskazanej kolumny.

Rozdział 6.

P

PIERWSZA POSTAĆ NORMALNA (1NF)

Każdy wiersz tabeli musi zawierać dane

atomowe, a oprócz tego musi zawierać unikalny identyfikator.

Rozdział 4.

Podzapytania nieskorelowane

Podzapytanie, które jest niezależne i nie odwołuje się w żaden sposób do zapytania zewnętrznego.

Rozdział 9.

Podzapytanie

Zapytanie umieszczone wewnętrz innego zapytania. Inne, często stosowane określenie podzapytań to „zapytania wewnętrzne”.

Rozdział 9.

Poprzedzanie znakami ' i \

Apostrofy umieszczone w tekście należy poprzedzać dodatkowym znakiem apostrofu lub ukośnika.

Rozdział 2.

PRAWOSTRONNE ZŁĄCZENIE ZEWNĘTRZNE

PRAWOSTRONNE ZŁĄCZENIE

ZEWNĘTRZNE pobiera wszystkie wiersze z prawej tabeli i dopasowuje je do wierszy z lewej tabeli.

Rozdział 10.

Przechodnia zależność funkcjonalna

Występuje, gdy jakakolwiek kolumna nie-będąca kluczem jest powiązana z innymi kolumnami, które także nie są kluczem.

Rozdział 7.

S

Schemat

Opis danych przechowywanych w bazie danych, wszelkich innych obiektów powiązanych z nimi oraz ich wzajemnych zależności.

Rozdział 7.

SELECT *

Używaj polecenia o takiej postaci, by pobrać wszystkie kolumny tabeli.

Rozdział 2.

SET

To słowo kluczowe jest stosowane w poleceniach UPDATE i służy do zmiany wartości konkretnej, istniejącej kolumny.

Rozdział 3.

SHOW CREATE TABLE

Skorzystaj z tego polecenia, by poznać prawidłową składnię stującą do utworzenia istniejącej tabeli.

Rozdział 4.

SUM

Sumuje wszystkie wartości w kolumnie liczbowej.

Rozdział 6.

T

Trzecia postać normalna (3NF)

Tabela musi być w drugiej postaci normalnej i nie mieć żadnych przechodnich zależności funkcjonalnych.

Rozdział 7.

U

UNION oraz UNION ALL

Polecenie UNION łączy wyniki dwóch lub większej liczby zapytań w jednej tabeli wynikowej, używając przy tym kolumn podanych na liście kolumn polecenia SELECT. Polecenie UNION ukrywa powielające się wartości, natomiast polecenie UNION ALL wyświetla wszystkie wiersze, nawet powtarzające się.

Rozdział 10.

UPDATE

To polecenie pozwala aktualizować wartość istniejącej kolumny lub kolumn. Można w nim zastosować klauzulę WHERE.

Rozdział 3.

USE DATABASE

Pozwala wybrać bazę danych, na jakiej chcemy działać.

Rozdział 1.

W

WIDOKI

Widoków można używać, by móc potraktować wyniki wykonania zapytania jako tabelę. Doskonale nadają się do zmiany bardzo złożonych zapytań w bardzo proste.

Rozdział 11.

WIDOKI, NIE POZWAJAJĄCE NA AKTUALIZACJĘ

Widoki, których nie można używać w poleceniach INSERT oraz UPDATE.

Rozdział 11.

WIDOKI POZWAŁAJĄCE NA AKTUALIZACJĘ

To widoki, które pozwalają na zmianę danych w tabelach, na których operują. Muszą one zawierać wszystkie kolumny zdefiniowane jako NOT NULL z tabeli lub tabel, na których operują.

Rozdział 11.

Wiele-do-wielu

Dwie tabele są połączone przez dodatkową tabelę łączącą, dzięki czemu wiele wierszy z jednej tabeli może odpowiadać wielu wierszom z drugiej tabeli i na odwrót.

Rozdział 7.

WITH GRANT OPTION

Pozwala użytkownikowi przekazywać innym te same uprawnienia, którymi on dysponuje.

Rozdział 12.

Z

Zapytanie wewnętrzne

Zapytanie umieszczone wewnętrznie innego zapytania. Określone także jako podzapytanie.

Rozdział 9.

Zapytanie zewnętrzne

Zapytanie zawierające zapytanie wewnętrzne nazywane także podzapytaniem.

Rozdział 9.

ZŁĄCZENIE NATURALNE

Złączenie wewnętrzne, w którym została pominięta klauzula ON. Działa ono wyłącznie w przypadkach, gdy obie łączone tabele posiadają kolumnę o identycznej nazwie.

Rozdział 8.

ZŁĄCZENIE PRZECINKOWE

Złączenia przecinkowe są tym samym co ZŁĄCZENIA KRZYŻOWE, z tą jedną różnicą, że do utworzenia złączenia stosowany jest przecinek, a nie stowa kluczowe CROSS JOIN.

Rozdział 8.

ZŁĄCZENIE RÓWNOŚCIOWE I NIERÓWNOŚCIOWE

Oba są rodzajami złączeń wewnętrznych. Pierwsze z nich zwraca wiersze, które są równe, a drugie – wiersze, które są różne.

Rozdział 8.

ZŁĄCZENIE ZWROTNE

Pozwala ono na pobieranie danych z jednej tabeli w taki sposób, jak gdyby w bazie istniały dwie tabele o identycznej zawartości.

Rozdział 10.

Skorowidz

% , 136
*, 92
_ , 136
< , 121, 150
<?php ?> , 567
<= , 121, 150
<> , 120, 145, 150
= , 120, 150
=> , 121, 150
> , 120, 121, 150
1NF , 210, 211, 228, 350, 372
2NF , 359, 361, 364, 372
3NF , 366, 372

A

ABS() , 563
ACID , 511
ACOS() , 563
ADD , 266
ADD COLUMN , 226, 253
ADD CONSTRAINT , 489
ADD INDEX , 565
ADD PRIMARY KEY , 253
administrator , 525
AFTER , 236
aktualizacja danych , 183, 188, 190, 504
aliasy , 373, 383, 384
ALL , 556
ALTER , 225, 227, 231, 244, 270, 320
ALTER TABLE , 232, 233, 266, 280
 ADD COLUMN , 233
 ADD CONSTRAINT , 489
 ADD INDEX , 565
 ADD PRIMARY KEY , 233
 CHANGE COLUMN , 244, 245
 DROP COLUMN , 250
 DROP PRIMARY KEY , 254

MODIFY COLUMN , 246
RENAME , 239
TYPE , 514
ALTER USER , 525
AND , 114, 122, 129, 130, 139, 145, 150
ANY , 556, 557
apostrofy , 54, 97, 100, 101
AS , 383, 384
ASC , 292, 293
ASIN() , 563
ATAN() , 563
atomicity , 511
atomowe dane , 201, 202
AUTO_INCREMENT , 222, 224, 227,
 228, 234, 254
automatyczne inkrementowanie , 222
AVG() , 298, 299, 305, 310

B

baza danych , 40, 44
 schemat , 326
 tabele , 46
 tworzenie , 53
BDB , 514
BEFORE , 236
BETWEEN , 140, 150
bezpieczeństwo , 521
BIGINT , 558
BLOB , 60, 61, 98
błąd składni SQL , 100
błędy , 522, 523
BOOLEAN , 558

C

CASCADE , 534, 537, 542
CASE , 273, 274, 279

CAST() , 561
CREATE VIEW , 493
CEIL() , 563
CHANGE , 244, 266
CHANGE COLUMN , 245
CHAR , 60, 98, 125
CHARACTER , 60
CHECK , 338, 488, 490, 504, 505
CHECK CONSTRAINT , 503
CHECK OPTION , 503
chronologiczne pobieranie danych , 160
close() , 566
COMMIT , 512, 517
consistency , 511
CONSTRAINT , 337
COS() , 563
COT() , 563
COUNT() , 301, 302, 305, 310
CREATE , 54, 382
CREATE DATABASE , 53, 54, 63, 86
CREATE ROLE , 539
CREATE TABLE , 55, 56, 57, 81, 86,
 217, 219, 224, 255, 382, 489
 AUTO_INCREMENT , 222
 CONSTRAINT , 337
 PRIMARY KEY , 220
CREATE TABLE AS , 471
CREATE TEMPORARY TABLE , 560
CREATE USER , 526, 547
CREATE VIEW , 493
CROSS JOIN , 389, 404, 408
cudzysłowy , 54, 101
CURRENT_DATE , 562
CURRENT_TIME , 562
CURRENT_USER , 562
czas , 60, 559, 562
częściowa zależność funkcjonalna , 355

D

dane, 38
atomowe, 201, 202, 228
tekstowe, 125
DATE, 58, 60, 98, 559
DATE_FORMAT(), 559
DATETIME, 60, 62, 559
daty, 60, 559
DEC, 60, 61, 96, 98
DECIMAL, 60
DEFAULT, 84
definiowanie danych, 38
DELETE, 153, 163, 165, 169, 192, 501
 FROM, 179
 nieprecyzyjne polecenia, 178
 reguły polecenia, 166
 WHERE, 164, 166, 167
DESC, 63, 64, 86, 292, 293
DESCRIBE, 218, 238, 327
diagram tabeli, 327
die(), 567
DISTINCT, 302, 305, 499
dodawanie
 indeks, 565
 klucz główny, 226, 383
 kolumny, 67, 234
 połączenia do diagramów tabel, 330
 użytkownicy, 526
dopasowywanie danych, 270
doprowadzanie tabeli do 1NF, 350
doprowadzanie tabeli do 2NF, 360
doprowadzanie tabeli do 3NF, 366
DROP, 68
DROP COLUMN, 250, 254, 280
DROP PRIMARY KEY, 254
DROP ROLE, 540
DROP TABLE, 68, 86

E

ELSE, 278
END, 274, 275
EXCEPT, 472
EXISTS, 440
EXP(), 563

F

false, 558
fałsz, 558
FIRST, 236
first normal form, 211
FLOOR(), 563
FORMAT(), 563
formatowanie danych różnych typów, 98
FROM, 79, 179, 495
FULL OUTER JOIN, 457
funkcje, 297
 matematyczne, 563
 obsługa łańcuchów znaków, 258

G

graficzna reprezentacja tabel, 327
graficzne programy do obsługi baz danych, 552
GRANT, 527, 528, 531, 537, 540, 544, 546, 547
 WITH ADMIN OPTION, 542
GRANT ALL, 531
GRANT OPTION, 533
GROUP BY, 298, 310, 499
grupowanie wartości, 298

H

hasła dostępu, 525
HAVING, 499

I

IDENTIFIED BY, 526
IN, 143, 415, 433
indeksy, 224, 565
informacje o kolumnach, 221
INNER JOIN, 393, 394, 396, 400, 406, 408
InnoDB, 514
INSERT, 70, 73, 77, 86, 169, 382, 383, 501
 dane zawierające apostrofy, 102
INSERT INTO, 70, 73, 74, 75
instalacja MySQL, 569, 570
 program instalacyjny, 572
 Windows, 571
INT, 60, 61, 98, 558
INTEGER, 60
integralność danych, 346, 488
integralność odwołań, 335, 338
INTERSECT, 472
IS NULL, 133, 150
isolation, 511
izolacja, 511

J

jeden-do-jednego, 339, 340, 411
jeden-do-wielu, 341, 345
język PHP, 566

K

kategoryzowanie danych, 40
klauzula CHECK OPTION, 503

klient bazy danych, 54
 klucz główny, 211, 212, 228, 333, 334, 462
 dodawanie do tabeli, 226
 tworzenie, 214
 usuwanie, 254
 wartości NULL, 213
 klucz nadrędny, 334
 klucz naturalny, 214
 klucz obcy, 333, 334, 372
 odwołanie do tej samej tabeli, 462
 ograniczenia, 335
 zastosowanie, 336
 klucz złożony, 352, 372
 kod
 PHP, 567
 SQL, 57
 kolejność kolumn, 254
 kolejność przetwarzania kolumn, 272
 kolumny, 43, 46, 49, 50
 aliasy, 384
 bez wartości, 78
 częściowa zależność funkcjonalna, 355
 dane atomowe, 204, 206
 klucz obcy, 333
 kontrola wartości NULL, 81
 liczba wierszy, 301
 modyfikacja, 242
 niezależne, 355
 przechodnia zależność funkcjonalna, 356
 usuwanie, 249
 wartości domyślne, 84
 zależne, 355
 zależność funkcjonalna, 353
 zmiana typu danych, 245
 komórki, 418
 konto administratora, 525
 konto użytkownika, 524

kontrola wewnętrznych wartości
 NULL, 81
 konwencje formatowania danych różnych typów, 98
 kopiowanie polecen SQL, 111

L

LAST, 236
 LEFT OUTER JOIN, 450, 454
 LEFT(), 258
 LENGTH(), 259
 lewostronne złączenia zewnętrzne, 449, 450, 457
 wartości NULL, 451
 lewy ukośnik, 101, 102
 liczba wierszy w kolumnie, 301
 liczby całkowite bez znaku, 558
 LIKE, 135, 150, 197, 320
 %, 136
 _, 136
 łańcuch o dowolnej długości, 136
 nieznany znak, 136
 znaki wieloznaczne, 135
 LIMIT, 306, 307, 310, 426, 478
 LN(), 563
 localhost, 525
 LOG(), 563
 LOWER(), 259
 LTRIM(), 259

T

łańcuch znaków o zmiennej długości, 55
 łączenie polecen SQL, 264
 łączenie tabel, 330, 333
 kolumny o unikalnych wartościach, 333
 łączenie tabeli z nią samą, 458, 463
 łączenie zapytań, 114

M

MAX(), 300, 305, 310, 424
 mechanizm składowania, 514
 MEDIUMINT, 558
 metapoznanie, 29
 MIN(), 300, 305, 310
 mniejszość, 121
 mniejszy lub równy, 121
 MOD(), 564
 MODIFY, 246, 266
 MODIFY COLUMN, 253
 modyfikacja
 dane, 180, 270
 kolumny, 242
 tabele, 227, 237
 tabele do 2NF, 360
 mózg, 27
 MySQL, 53, 566
 instalacja, 570
 rozwiązywanie problemów, 570
 MySQL Administrator, 54, 552
 MySQL Query Browser, 54, 552, 570
 MySQL Tools, 552
 mysql_close(), 567
 mysql_connect(), 566, 567
 mysql_error(), 566
 mysql_fetch_array(), 566, 567
 mysql_query(), 566, 567
 mysql_select_db(), 566, 567

N

najmniejsza wartość w kolumnie, 300
 największa wartość w kolumnie, 300
 NATURAL JOIN, 398, 408, 428
 naturalne złączenia zewnętrzne, 398, 426
 nazwy, 54
 kolumny, 221
 korelacje, 438

tabele, 48
nazwy zastępcze, 406
kolumny, 384, 385, 406, 423
tabele, 406
Niepodzielność, 511
nieskorelowane podzapytania, 429
noncorrelated subqueries, 429
normalizacja, 193, 208, 239, 349, 377
 1NF, 210, 350
 2NF, 360
 3NF, 366
 zalety, 209
NOT, 145, 150
NOT EXISTS, 439, 440
NOT IN, 144, 145, 433
NOT NULL, 81, 82, 83, 86, 220, 225,
 504
NULL, 79, 80, 86
numery PESEL, 212, 221

O

obsługa łańcuchów znaków, 258
odnajdywanie
 dane tekstowe, 125
 wartości liczbowe, 117, 122
odporność, 511
odwracanie kolejności sortowania, 292
oglądanie widoków, 494
ograniczanie wyników zapytania, 107
 liczba wyników, 306
ograniczenia, 334, 335, 337, 486
 CHECK, 338, 488, 504
 CHECK CONSTRAINT, 503
 sprawdzające, 488
 UNIQUE, 338
określanie zwracanych kolumn, 107
ON, 396, 405
operacje na łańcuchach, 378
operacje na wielu tabelach, 373

operatorы
 AND, 114, 122, 130, 139
 BETWEEN, 140
 EXISTS, 440
 IN, 143
 IS NULL, 133
 LIKE, 135, 197
 mniejszość, 121
 mniejszy lub równy, 121
 NOT, 144, 145
 NOT EXISTS, 439, 440
 OR, 127, 128, 130
 porównania, 118, 120, 122
 równość, 120
 różny, 120
 warunkowe, 131
 większość, 119, 121
 większy lub równy, 121
opis tabeli, 63
optymalny sposób realizacji
 zapytania, 430
OR, 127, 128, 129, 130, 145, 150
ORDER BY, 285, 286, 289, 290, 296,
 302, 310, 405, 478
 ASC, 293
 DESC, 293
wiele kolumn, 289, 290
ostatni rekord, 162
ostatnie znaki łańcucha, 258
otrzeżenia, 221
oznaczanie apostrofów, 102, 104

P

PASSWORD(), 525, 546
pełnełączenia zewnętrzne, 457
PESEL, 212, 221
PHP, 101, 566
phpMyAdmin, 553
PI(), 564

pierwsza postać normalna, 211, 215,
 228, 349, 350
pobieranie
 dane, 79, 87, 88
 fragment łańcucha, 258
 kolumny, 106
 unikalne wartości, 303
 wartości liczbowe, 122
 wartości należące do zbioru
 wartości, 143
 zakres, 139
podzapytania, 409, 416, 444, 473, 556
 ALL, 556
 ANY, 556, 557
 EXISTS, 440
 IN, 433
 NATURAL JOIN, 428
 nazwy zastępcze, 438
 nieskorelowane, 429, 433
 NOT EXISTS, 439, 440
 NOT IN, 433
 reguły, 421, 422
 SELECT, 427
 skorelowane, 438, 479
 SOME, 556, 557
 tworzenie, 424
 zaginieździanie podzapytań, 442
 zamiana na złączenie, 474
 zapytania wewnętrzne, 416, 417, 418
 zapytania zewnętrzne, 416, 417, 422
 złączenia naturalne, 428
 złączenia zwrotne, 479
 zwracanie wielu wartości, 433
pola, 49
polecenia, 53
 ALTER, 225, 227, 231, 244, 270
 ALTER TABLE, 232
 CHANGE, 244
 COMMIT, 512
 CREATE, 54

- CREATE DATABASE, 53, 54
 CREATE ROLE, 539
 CREATE TABLE, 55, 56, 217,
 382
 CREATE TEMPORARY
 TABLE, 560
 CREATE USER, 526, 547
 CREATE VIEW, 493
 DELETE, 153, 163, 169
 DESC, 63, 64, 86
 DESCRIBE, 218, 238, 327
 DROP, 68
 DROP ROLE, 540
 DROP TABLE, 68, 86
 DROP VIEW, 505
 EXCEPT, 472
 GRANT, 527, 528, 537, 547
 GRANT ALL, 531
 INSERT, 70, 73, 86, 169
 INSERT INTO, 70
 INTERSECT, 472
 pomijanie nazw kolumn, 77
 pomijanie niektórych kolumn, 77
 REVOKE, 532, 537
 ROLLBACK, 512
 SELECT, 79, 87, 383
 SHOW, 221
 SHOW CREATE TABLE, 218,
 221
 SHOW WARNINGS, 221
 START TRANSACTION, 512
 UNION, 466
 UNION ALL, 470
 UPDATE, 153, 180, 263
 USE, 54
 zmiana kolejności kolumn, 77
 połączenie z bazą danych, 567
 porównania, 118, 120
- postaci normalne
 1NF, 210, 211, 350
 2NF, 360
 3NF, 366
 POWER(), 564
 powiązane informacje, 48
 powtarzanie danych, 346
 prawda, 558
 prawostronne złączenia zewnętrzne,
 456, 457
 primary key, 211
 PRIMARY KEY, 220, 224, 337
 problemy wprowadzania danych, 522
 projekt tabeli, 162, 315
 projektowanie bazy danych, 311
 1NF, 350
 2NF, 360
 3NF, 366
 graficzna reprezentacja tabel, 327
 integralność odwołań, 335
 klucz obcy, 334
 ograniczenia, 335
 schemat bazy danych, 326
 tabele, 325
 zależności pomiędzy tabelami, 339
 projektowanie tabel, 193
 przechodnia zależność funkcjonalna, 356
 przejrzystość kodu, 219
 przekształcanie tabeli w dwie tabele, 328
 przydzielanie uprawnień, 527, 528
- R
 RADIANS(), 564
 RAND(), 564
 RDBMS, 53, 198
 reguły klucza głównego, 212
 reguły kolejności, 287, 288
 rejestrowanie poleceń SQL, 512
- rekordy, 49
 relacje, 197, 339
 jeden-do-jednego, 339, 340
 jeden-do-wielu, 341
 wiele-do-wielu, 342, 346
 Relational Database Management
 System, 53, 198
 RENAME, 239, 253
 RESTRICT, 534, 543
 REVERSE(), 259
 REVOKE, 532, 537
 CASCADE, 542
 RESTRICT, 543
 RIGHT OUTER JOIN, 456
 RIGHT(), 258, 263
 role, 539, 541
 stosowanie, 540
 usuwanie, 540
 usuwanie uprawnień, 541
 ROLLBACK, 512, 517
 root, 525
 ROUND(), 564
 rozdzielenie wartości, 378
 rozwiązywanie problemu apostrofów, 102
 równościowe złączenia wewnętrzne, 394
 równość, 120
 różnociowe złączenia wewnętrzne, 397
 RTRIM(), 259
 rzutowanie typów, 561
- S
 samoprzylepne karteczki, 38
 schemat bazy danych, 326, 372
 schemat przepływu sterowania, 44
 SECOND, 236
 SELECT, 79, 87, 261, 376, 381, 382,
 383, 479
 *, 88, 92, 150

- aliasy, 383, 384
AND, 114
AS, 383, 384
BETWEEN, 140
COUNT(), 302
CROSS JOIN, 389, 404
FROM, 79, 88
FULL OUTER JOIN, 457
INNER JOIN, 393, 394, 396, 400
kryteria wyboru, 114
LEFT OUTER JOIN, 450
LIKE, 135
LIMIT, 306, 307
NATURAL JOIN, 398
nazwy zastępcze kolumn, 385
ograniczanie liczby wyników, 306
ograniczanie wyników, 107
określanie kolumn, 107
ORDER BY, 285, 286
organizowanie danych zwracanych, 282
pobieranie kolumn, 106
pobieranie unikalnych wartości, 303
podzapytania, 427
RIGHT OUTER JOIN, 456
UNION, 466
UNION ALL, 470
WHERE, 91, 97
widoki, 494
wybór wartości, 91
wyszukiwanie konkretnych danych, 105
złączenia zewnętrzne, 448
złączenia zwrotne, 465
znaki przestankowe, 99
zwracanie wszystkich kolumn, 92
serwer bazy danych, 53
SET, 180, 181, 192, 264
SET PASSWORD FOR, 525
SHOW, 221
SHOW COLUMNS FROM, 221
SHOW CREATE DATABASE, 221
SHOW CREATE TABLE, 218, 219, 221, 225, 228, 514
SHOW INDEX FROM, 221
SHOW WARNINGS, 221
SIGN(), 564
SIGNED, 558
SIN(), 564
skorelowane podzapytania, 438
słowa kluczowe, 54
słowa zastrzeżone, 554, 555
SMALLINT, 558
SOME, 556, 557
sortowanie w oparciu o kilka kolumn, 290
sortowanie według jednej kolumny, 286
sposób korzystania z danych, 197
spójność, 511
SQL, 44, 53
SQLyog, 553
SQRT(), 564
START TRANSACTION, 512, 517
stosowanie
 dane atomowe, 204
 role, 540
 widoki w poleceniach SELECT, 494
 złączenia wewnętrzne, 412
SUBSTR(), 379
SUBSTRING(), 259
SUBSTRING_INDEX(), 258, 318, 378
SUM(), 297, 298, 310
sumowanie, 297
syntetyczny klucz główny, 214
system zarządzania relacyjną bazą
 danych, 53
sztuczny klucz główny, 366
- Ś**
średnia, 299
średnik, 55
- T**
tabele, 43, 46, 47, 54
 1NF, 350
 2NF, 360
 3NF, 366
 dane atomowe, 204
 diagram, 327
 dodawanie klucza głównego, 226
 dodawanie kolumn, 67, 234
 doprowadzanie do 1NF, 350
 klucz główny, 212
 klucz obcy, 336
 klucz złożony, 352
 kolumny, 46, 49
 łączące, 345
 modyfikacja, 237
 nadzędne, 334
 normalizacja, 208, 239
 opis, 63
 pola, 49
 projekt, 162
 rekordy, 49
 tworzenie, 50, 55
usuwanie, 68
usuwanie klucza głównego, 254
wiersze, 46, 49
wirtualne, 496
wstawianie danych, 70
zależności pomiędzy tabelami, 339
zmiana nazwy, 239
zmiana strukturalna, 243
związki, 198
tabele tymczasowe, 560
tworzenie, 560
TAN(), 564
tekst, 55, 125
TEMPORARY, 560
test ACID, 511

THEN, 274, 275
 THIRD, 236
 TIME, 98, 559
 TIMESTAMP, 60, 62, 559
 TINYINT, 558
 transakcje, 506, 510, 515
 ACID, 511
 BDB, 514
 COMMIT, 512
 InnoDB, 514
 mechanizm składowania, 514
 MySQL, 514
 ROLLBACK, 512
 wycofanie, 512
 zarządzanie, 512
 zatwierdzanie, 512
 true, 558
 TRUNCATE(), 564
 trzecia postać normalna, 215, 366
 tworzenie
 baza danych, 53
 klucz główny, 214
 klucz obcy, 335
 podzapytania, 424
 role, 539
 schemat bazy danych, 326
 tabela łącząca, 346
 tabele, 50, 55
 tabele na podstawie wyników
 polecenia UNION, 471
 tabele tymczasowe, 560
 tabele z kluczem głównym, 220
 tabele z kluczem obcym, 337
 tabele znormalizowane, 208
 użytkownicy, 526
 widoki, 492, 493
 typy danych, 55, 59, 60, 558
 apostrofy, 101
 BIGINT, 558

BLOB, 60
 BOOLEAN, 558
 CHAR, 60, 125
 CHARACTER, 60
 DATE, 58, 60, 559
 DATETIME, 60, 559
 DEC, 60
 DECIMAL, 60
 INT, 60, 558
 INTEGER, 60
 MEDIUMINT, 558
 rzutowanie, 561
 SMALLINT, 558
 TIME, 559
 TIMESTAMP, 60, 559
 TINYINT, 558
 VARCHAR, 55, 60, 125
 wybór, 61

U

unia, 447, 467
 unikalne wartości, 303, 333
 unikalność rekordów tabeli, 212
 unikanie częściowej zależności
 funkcjonalnej, 359
 UNION, 466, 467
 ograniczenia, 468
 reguły stosowania, 469
 tworzenie tabeli na podstawie
 wyników, 471
 UNION ALL, 470
 UNIQUE, 338
 UPDATE, 153, 180, 182, 190, 192,
 261, 263, 271, 378, 502
 aktualizacja wielu rekordów, 190
 CASE, 273
 reguły stosowania, 181
 SET, 180, 181, 264
 WHERE, 181
 UPPER(), 259

uprawnienia, 526, 528, 541
 GRANT OPTION, 533
 precyzyjne usuwanie, 534
 przydzielanie, 527, 528
 REVOKE, 532
 usuwanie, 532
 USE, 54, 63, 86
 usuwanie, 153, 174
 klucz główny, 254
 kolumny, 249
 rekordy, 163
 role, 540
 tabele, 68
 uprawnienia, 532, 534
 uprawnienia przydzielone przy
 użyciu roli, 541
 widoki, 505
 użytkownicy, 522, 524
 przydzielanie uprawnień, 528
 tworzenie kont, 526
 uprawnienia, 526
 usuwanie uprawnień, 532

V

VALUES, 70, 73, 101
 VARCHAR, 55, 60, 67, 98, 125

W

wartości domyślne, 84
 wartości NULL, 79, 80, 86, 219, 305,
 486
 wyszukiwanie, 133
 wartości skalarne, 418
 WHEN, 274, 275
 WHERE, 91, 96, 97, 164, 167, 181,
 278, 405, 503
 widoki, 492, 496
 aktualizacja danych, 499, 504

- CHECK OPTION, 503
stosowanie w poleceniach
 SELECT, 494
tworzenie, 493
usuwanie, 505
usuwanie danych, 499
wstawianie danych, 499
zasada działania, 495
wiele-do-wielu, 342, 345, 346
wielkość liter poleceń, 54
wieloznaczność, 135
wiersze, 43, 46, 49
większość, 121
większy lub równy, 121
WITH ADMIN OPTION, 542
WITH GRANT OPTION, 531
wklejanie poleceń SQL, 111
wpisywanie danych bez ograniczeń,
 486
współużytkowanie konta, 538
wstawianie danych, 70
 niekompletne rekordy, 78
wybór bazy danych, 54
wybór typu danych, 61
wycofanie transakcji, 512
wykonanie zapytania, 44
wymuszanie integralności odwołań,
 338
wyniki kartezjańskie, 388
- wypełnianie tabel, 375
wyszukiwanie
 konkretnie dane, 105
 wartości NULL, 133
 wartości różnych typów, 98
wyświetlanie
 informacje o kolumnach, 221
 kod polecenia, 218
- Z**
- zabezpieczanie konta administratora,
 525
zagospodarowanie podzapytania, 442
zależności, 339
 jeden-do-jednego, 340, 411
 jeden-do-wielu, 341
 wiele-do-wielu, 342, 346
zależność funkcjonalna, 353
zamiana podzapytania na złączenie, 474
zapobieganie błędom, 523
zapytania, 44, 414
 podzapytania, 416, 417
 wewnętrzne, 416, 418
 zewnętrzne, 416
zarządzanie transakcjami, 512
zatwierdzanie transakcji, 512
złączenia, 373, 405, 473
 kartezjańskie, 387, 388
 krzyżowe, 388, 389
- naturalne, 398, 411
równościowe, 394
różnościowe, 397
wewnętrzne, 387, 389, 393, 412, 426
złączenia zewnętrzne, 389, 447, 448
 LEFT OUTER JOIN, 450
lewostronne, 449, 450
pełne, 457
prawostronne, 456, 457
wielokrotne dopasowania, 455
złączenia zwrotne, 447, 465
podzapytania, 479
- zmiana
 dane, 153
kolejność kolumn, 77, 254
kolejność sortowania, 293
mechanizm składowania, 514
nazwa tabeli, 239
typ danych kolumny, 245
wartość kolumny we wszystkich
 wierszach, 263
wielkość liter, 259
- znaki
 przestankowe, 99
 specjalne, 101, 554
wieloznaczne, 135
zarezerwowane, 92
związki, 198
zwracanie informacji z wielu tabel, 466