helion.pl

Idź do

- Spis treści
- Przykładowy rozdział

Katalog książek

- Katalog online
- Zamów drukowany katalog

Twój koszyk

- Dodaj do koszyka

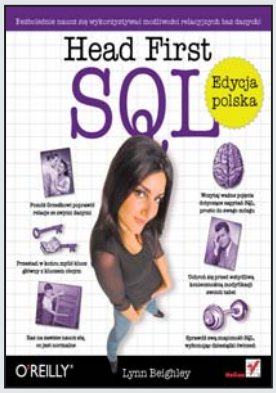
Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

Czytelnia

- Fragmenty książek online

Head First SQL. Edycja polska



Autor: Lynn Beighley
Tłumaczenie: Piotr Rajca
ISBN: 978-83-246-1445-5
Tytuł oryginału: Head First SQL: Your Brain on SQL – A Learner
Format: 200x230, stron: 592

Czy jesteś w stanie szybko powiedzieć, ile posiadasz książek? Jaki autor jest najbardziej popularny na Twojej półce? Jeżeli miałbyś bazę danych swoich książek, a baza ta obsługiwałaby język zapytań SQL, mógłbyś błyskawicznie udzielić odpowiedzi na te pytania. W przeciwnym razie... no cóż, zabierze Ci to o wiele więcej czasu. I czy będziesz pewien poprawności odpowiedzi?

Czym jest język SQL? To potężne narzędzie, którego opanowanie pozwoli Ci na sprawne poruszanie się po bazie danych. A za pomocą odpowiednio sformułowanych instrukcji będziesz mógł manipulować danymi, zarządzać kontami użytkowników i generować praktyczne raporty.

Oto innowacyjny podręcznik „Head First SQL. Edycja polska", w którym autorzy – wykorzystujący najnowsze, skuteczne techniki nauki – szybko przekażą ci całą niezbędną wiedzę o tym przydatnym języku. Nauczysz się tworzyć tabele, dodawać do nich dane oraz pobierać je. Dowiesz się, w jaki sposób ograniczać zbiór wybieranych danych za pomocą odpowiednich warunków. Po przeczytaniu tej książki bez trudu przedstawisz dane w odpowiedniej kolejności i zakresie oraz zaczniesz swobodnie używać podzapytań, łączyć dane z różnych tabel, a także zapewniać im bezpieczeństwo.

- Tworzenie bazy danych i tabel (CREATE)
- Pobieranie danych (SELECT)
- Usuwanie (DELETE) i aktualizowanie danych (UPDATE)
- Wykorzystywanie kluczy i indeksów
- Sortowanie danych
- Operacje na danych – sumy, średnie, elementy maksymalne i minimalne
- Pobieranie unikalnych danych
- Sposób pobierania określonej liczby wierszy
- Stosowanie podzapytań
- Zapewnianie spójności danych
- Ograniczanie dostępu do zgromadzonych danych
- Zapewnianie bezpieczeństwa bazie danych

Bezboleśnie naucz się wykorzystywać możliwości relacyjnych baz danych!

Kontakt

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

Spis treści (skrócony)

	Wprowadzenie	25
1	Dane i tabele: *Na wszystko znajdzie się odpowiednie miejsce*	37
2	Polecenie SELECT: *Pobieranie podarowanych danych*	87
3	DELETE i UPDATE: *Są szanse, że wszystko będzie w porządku*	153
4	Projektowanie dobrych tabel: *Po co być normalnym?*	193
5	Polecenie ALTER: *Korygowanie przeszłości*	231
6	Zaawansowane zastosowanie polecenia SELECT: *Nowy sposób spojrzenia na dane*	267
7	Projektowanie baz danych składających się z wielu tabel: *Wyrastamy z naszych starych tabel*	311
8	Złączenia i operacje na wielu tabelach: *Czy nie możemy się wszyscy dogadać?*	373
9	Podzapytania: *Zapytania w zapytaniach*	409
10	Złączenia zewnętrzne, złączenia zwrotne oraz unie: *Nowe manewry*	447
11	Ograniczenia, widoki i transakcje: *Zbyt wielu kucharzy psuje bazę danych*	483
12	Bezpieczeństwo: *Zabezpieczanie swych dóbr*	521
	Dodatek A *Pozostałości*	551
	Dodatek B *Instalacja MySQL-a*	569
	Dodatek C *Przypomnienie narzędzi*	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 jeżdżenie 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
Metapoznanie: myslenie o myśleniu	29
Oto co możesz zrobić, aby zmusić swój mózg do posłuszeństwa	31
Przeczytaj to	32
Nasi wspaniali recenzenci	34
Podziękowania	35

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 zaczniesz 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 połowach. Tabela Marka zawiera kolumny pozwalające na zapisanie łacińskiej nazwy gatunku ryby, nazwy polskiej, wagi złowionej ryby oraz miejsca dokonania połowu. Nie zawiera jednak kolumn pozwalających na zapianie imienia i nazwiska osoby, która ustanowiła rekord.

polowy_informacje

Ta tabela ma jedynie cztery kolumny. Porównaj ją z tabelą rekordowe_polowy przedstawioną na następnej stronie.

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

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 połowu.

Także ta tabela służy do rejestrowania rekordowych połowów wędkarskich, jednak zawiera niemal dwukrotnie więcej kolumn.

rekordowe_polowy

imie	nazwisko	nazwa	miejsce	wojewodztwo	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	płotka	Charzykowy	PM	0,65 kg	1.8.1965
Ignacy	Wikorczyk	łuskot	Czaplinek	ZP	1,10 kg	31.9.1988
Krzysztof	Dubała	węgorz	Swornegacie	PM	1,45 kg	12.8.1973
Paweł	Wronek	szczupak	Karwica	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 połowy dokonane w województwie podkarpackim.

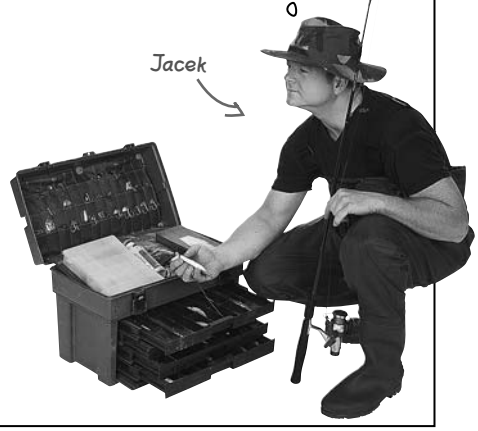
.....

.....

.....

.....

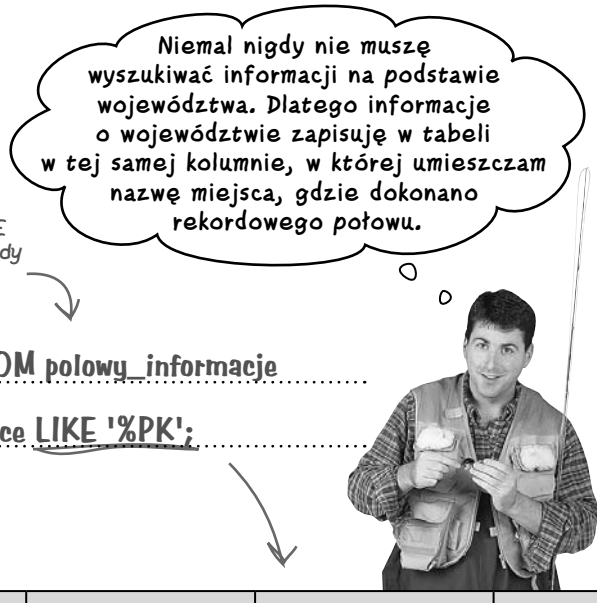
Piszę artykuły dla magazynu „Weekend z wędką". Muszę znać imiona i nazwiska wędkarzy, którzy ustanowili rekordowe połowy, daty tych połowów oraz ich miejsca.



Zaostrz ołówek

Rozwiązanie

Dla każdej z tabel napisz zapytanie, które pobierze informacje o rekordowych połowach dokonanych na terenie województwa podkarpackiego.



Musimy zastosować operator LIKE i odszukać interesujące nas rekordy na podstawie pola zawierającego połączoną nazwę miejscowości i oznaczenie województwa.

```
SELECT * FROM polowy_informacje
WHERE miejsce LIKE '%PK';
```

nazwa	nazwa_gatunkowa	miejsce	waga
pstrąg	O. Clarki	Mrzygłód, PK	1,20 kg

To zapytanie odwołuje się bezpośrednio do kolumny „wojewodztwo".

```
SELECT * FROM rekordowe_polowy
WHERE województwo = 'PK';
```

imie	nazwisko	nazwa	miejsce	wojewodztwo	waga	data
Zenon	Krawczyk	pstrąg	Mrzygłód	PK	1,20 kg	23.6.1978

Nie ma niemądrych pytań

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 połowów, 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 powinienem przechowywać w kolumnach bardzo małe fragmenty informacji?

O: Niekoniecznie. Analizując tabele Jacka i Marka, mogłeś już zauważyć, 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 pomyśl, 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.

WYTĘŻ UMYSŁ

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 odlotową 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 korzystał z tabeli?

3. **Na podstawie tej listy podziel informacje o zagadnieniu na elementy, których następnie będziesz mógł użyć, określając organizację tabeli.**

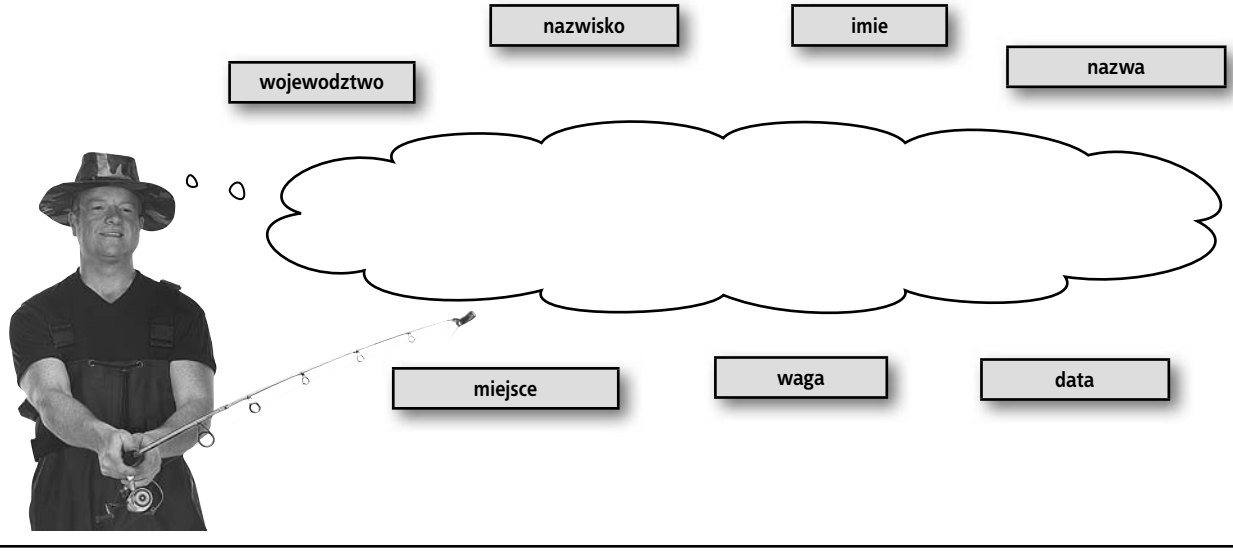
Jak będzie Ci najłatwiej przeszukiwać i pobierać dane z tabeli?

* 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 ichtiolog Marek chce przeszukiwać i pobierać dane z tabeli, potrafisz określić, jakie powinny być jej kolumny? Wpisz nazwy kolumn w pustych prostokątach na poniższym rysunku.

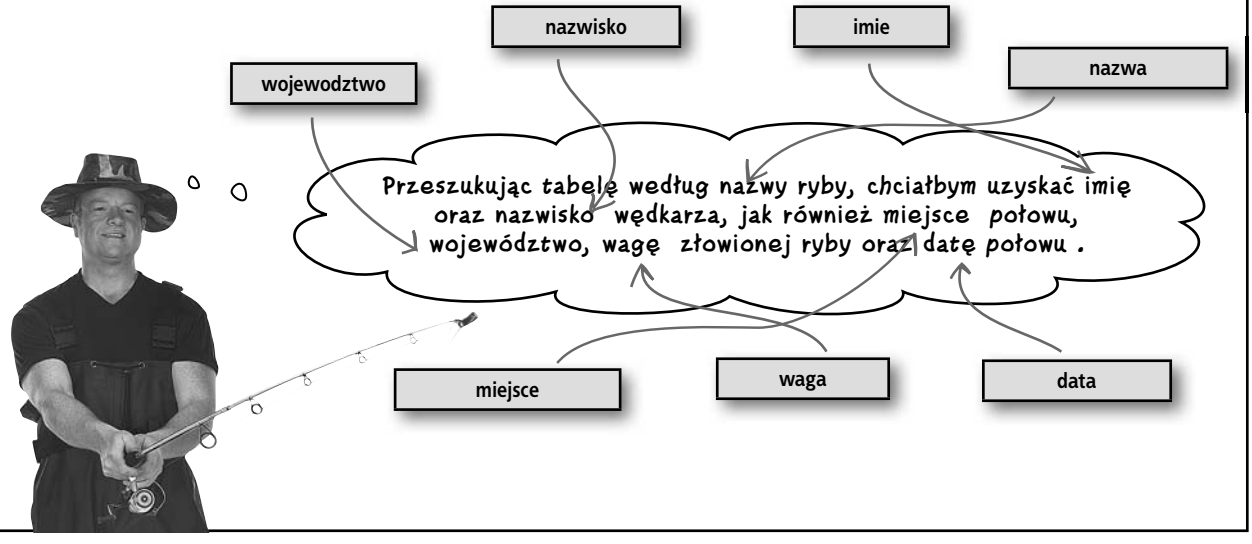
Twoja kolej. Napisz analogiczne zdanie dotyczące Jacka — autora artykułów dla magazynu „Weekend z wędką", który korzysta z bazy, by notować w niej szczegółowe informacje na potrzeby swoich artykułów. Następnie narysuj strzałki prowadzące od nazw kolumn do miejsca w zdaniu, w którym Jacek nawiązuje do danej kolumny.



Rozwiązanie ćwiczenia

Czy na podstawie przedstawionego poniżej zdania określającego, w jaki sposób ichtiolog 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, miesięcy 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 „miejsce" 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***.

WYTĘŻ UMYSŁ

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 zostałoby 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 Zwycieстwa 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 pizze_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.

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
+--------------+---------------------+-----------------+---------+
| 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 = 'Warszawsk
a';
+--------+---------------+
| 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 klownów, 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 ma niemądrych pytań

P: Czyż atomy nie są bardzo małe? Czy nie powinienem 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

Rozwiązanie

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.

Zbyt wiele kolumn do podawania studentów!

nauczyciel	student1	student2	student3
Pani Martini	Janek	Romek	Kasia
Pan Grog	Sonia	Tymon	Julia

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 klownach, strona 155 .....

Tabela drinków, ze strony 93 .....

Tabela połowów 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 ucieszy 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.

Rozwiązanie 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 klownów w każdej z kolumn tej tabeli przechowywany jest tylko jeden element informacji.

Tabela z informacjami o klownach, strona 155 Tabela nie jest atomowa. W niektórych rekordach w kolumnie „aktywnosci" zapisywanych jest więcej czynności niż jedna, co jest sprzeczne z regułą 1.

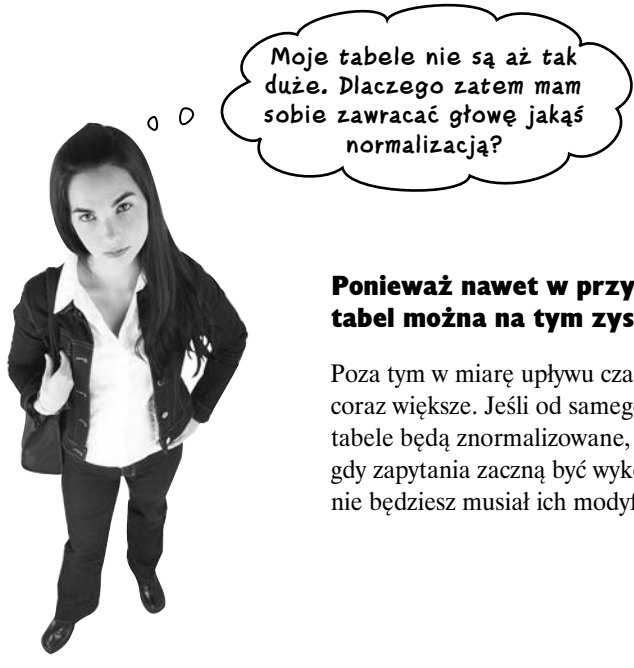
Tabela drinków, ze strony 93 Tabela nie jest atomowa. Zawiera ona więcej niż jedną kolumnę „składnik", co jest sprzeczne z regułą 2.

Tabela połowów 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.

2. Dzięki mniejszej ilości informacji w bazie wszelkie zapytania będą wykonywane szybciej.



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 klownów? Śledzenie poczynań klownów stało się ogólnokrajowym szaleństwem i nasza stara tabela może nie sprostać zwiększonym wymaganiom, ponieważ kolumny `wyglad` i `aktywnosci` 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!

klowni_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	wchodzenie do malutkiego samochodu

Zaostrz ołówek

Spróbujmy tak zmodyfikować tabelę `klowni_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`.

Odpowiedzi znajdziesz na stronie 229

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 zaspokoić 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).

WYTĘŻ UMYSŁ

Jak sądzisz, jakiego typu kolumny będą się nadawały do tworzenia klucza głównego?

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.

Kluczem głównym tabeli nazywamy kolumnę, która sprawia, że każdy wiersz tabeli jest unikalny.

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 założyć, 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.

Obejrzyj to!

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 właścicieli.

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.

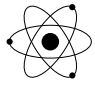
Jeśli wartość kolumny klucza głównego nie zostanie określona w momencie dodawania rekordu, to ryzykujemy, że pojawi się w niej wartość NULL, co może doprowadzić do powtórzenia się tego samego rekordu w tabeli i naruszenia zasad pierwszej postaci normalnej.

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.



WYTĘŻ UMYSŁ

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 ma niemądrych pytań

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 narzucał 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`.

WYTĘŻ UMYSŁ

Czy pamiętasz, jak można dodawać kolumny do istniejących tabel?

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 poleceniem 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.

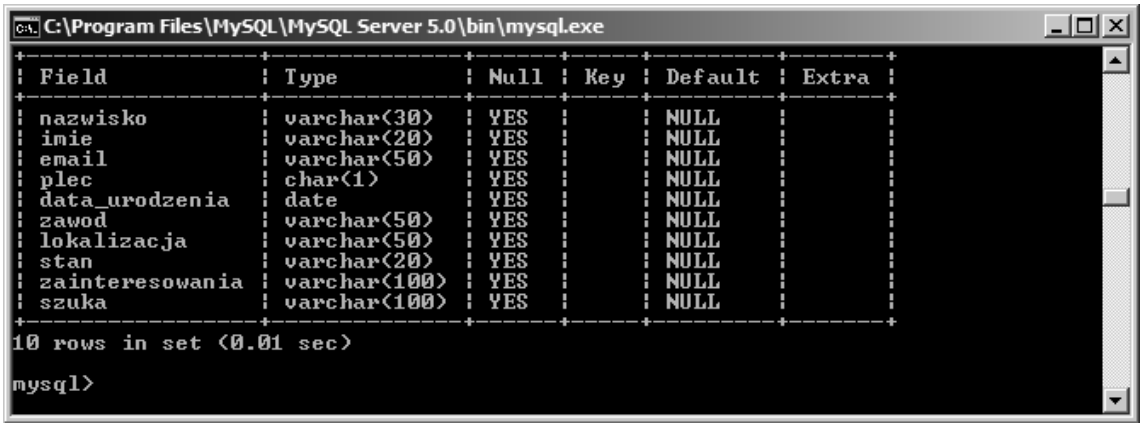
Czy tworząc tabelę, możemy poprawić atomowość tych kolumn?

WYTĘŻ UMYSŁ

A co zrobić, gdybyśmy nie mieli nigdzie zanotowanego oryginalnego polecenia `CREATE TABLE` użytego do utworzenia tabeli? Czy istnieje jakiś sposób dotarcia do jego kodu?

Pokażcie mi moją ~~kasę~~ tabelę

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:



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, określanymi jako lewy apostrof, są zapisywane nazwy kolumn oraz nazwa tabeli. Znaki lewego apostrofu są używane w wynikach generowanych przez polecenie SHOW CREATE TABLE.

Jeśli jawnie nie zażądamy 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.

```
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
```

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`.

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 pominąć wartość zapisywaną w tej kolumnie, to nie moglibyśmy zagwarantować, że wszystkie wiersze tabeli będą identyfikowane w jednoznaczny, unikalny sposób.

```
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)
)
```

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.

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 ma niemądrych pytań

P: Zatem mówicie, że KLUCZ GŁÓWNY nie może przybierać wartości NULL. Co jeszcze zapewnia jego unikalność?

O: Najprościej rzecz 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 jakiejś 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 niepożądane osoby).

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;

Powyższe 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 niewytłumaczalnych powodów chcielibyśmy nadać kolumnie nazwę `select`. Poniższa deklaracja kolumny jest jednak nieprawidłowa:

select varchar(50)

Zamiast niej musielibyśmy 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 którą 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 poleceń.

twoja_tabela

id	imie	nazwisko

Rozwiązanie ćwiczenia

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.

twoja_tabela

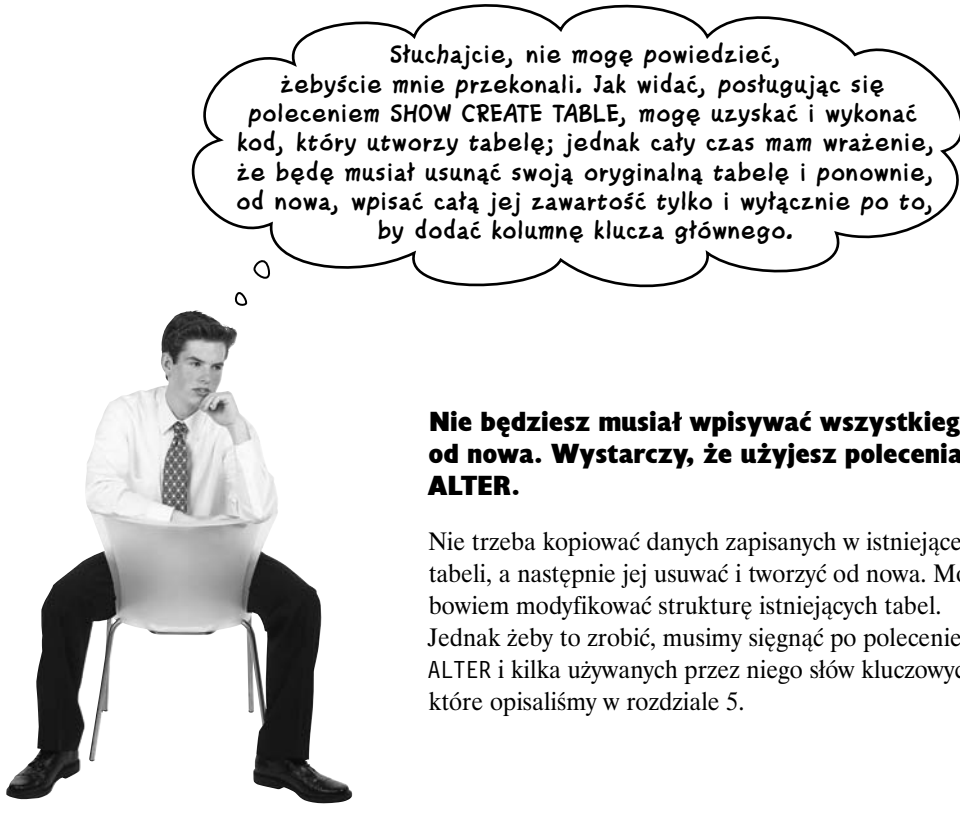
id	imie	nazwisko
1	Marysia	Krawczyk
2	Bartek	Krawczyk
3	Cecylia	Krawczyk
99	Piotr	d Krawczyk

Wygląda na to, że zgubiliśmy Jankę; zapewne dlatego, że próbowaliśmy jej przypisać indeks, który został już użyty w rekordzie Marysi. Marysia, Marysia, Marysia!

Nie ma niemądrych 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.



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).

Oto nasze nowe polecenie SQL — ALTER.

A to kod, który doda do tabeli nową kolumnę. Czy nie wygląda znajomo?

FIRST to słowo, które nakazuje, by nowa kolumna została dodana jako pierwsza kolumna tabeli. Choć nie ma to większego znaczenia, to jednak umieszczanie klucza głównego jako pierwszej kolumny tabeli jest dobrym pomysłem.

```
ALTER TABLE moje_kontakty
ADD COLUMN id_kontaktu INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (id_kontaktu);
```

Działanie słów ADD COLUMN jest zgodne z ich znaczeniem w języku angielskim — dodają 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.

WYTĘŻ UMYSŁ

Czy sądzisz, że powyższe 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` poleceniem `USE` i wykonaj polecenie:

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.

```
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>
```

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.

```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
+-------------+-------------+-----------+-----------------------------------+
| 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 | Mikulski    | Janek     | mikunek@wp.pl                     |
|           9 | Tkacz       | Martyna   | mptkacz@o2.eu                     |
|          10 | Maliniak    | Konrad    | kon_malinka@op.pl                 |
|          11 | Piekarska   | Kasia     | mysia_kasia@o2.eu                 |
|          12 | [illegible] | [illegible] | [illegible]                       |
```

Kiedy w przyszłości będziemy dodawali nowy rekord, to w jego kolumnie id_kontaktu zostanie zapisana wartość o jeden większa od największej wartości zapisanej w całej 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

Dane w tabeli są atomowe, jeśli zostały już podzielone na najmniejsze elementy, które są Ci potrzebne.

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.

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.

KLUCZ GŁÓWNY

Kolumna lub zbiór kolumn, które w unikalny sposób identyfikują wiersze tabeli.

AUTO_INCREMENT

Zastosowanie tego słowa kluczowego w deklaracji tabeli sprawi, że w momencie wykonywania poleceń INSERT w danej kolumnie będą się automatycznie i magicznie pojawiały unikalne liczby całkowite.

Zaostrz ołówek

Rozwiązanie

Spróbujmy tak zmodyfikować tabelę `klowni_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, musiałbyś wydzielić te wszystkie aktywności i umieścić je w osobnych kolumnach, i podobnie zrobić z informacjami o wyglądzie.

Dodatkowe punkty możesz sobie przyznać, jeśli chciałeś rozbić adres i umieścić te informacje w takich kolumnach jak adres, miejscowość i województwo.