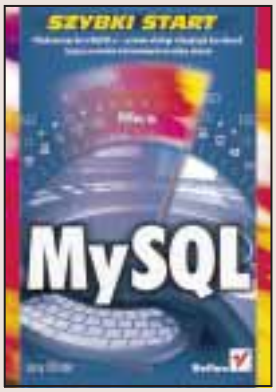
MySQL. Szybki start



Autor: Larry Ullman
Tłumaczenie: Marek Pałczyński
ISBN: 83-7361-040-5
Tytuł oryginału: MySQL VQG
Format: B5, stron: 336

Książka „MySQL. Szybki start" to przystępne wprowadzenie dla osób, które chcą w krótkim czasie poznać MySQL — jeden z najpopularniejszych systemów bazodanowych. Do jego zalet należą: szerokie rozpowszechnienie, duża wydajność i prostota obsługi. Jeśli chcesz stworzyć swoją pierwszą bazę danych, MySQL idealnie się do tego nadaje. Chociaż jest to produkt darmowy, pod wieloma względami nie ustępuje znacznie droższym aplikacjom komercyjnym.

„MySQL. Szybki start" to same konkrety; nie znajdziesz tu zbędnych teoretycznych rozważań i dygresji. Każdy podrozdział przedstawia sposób, w jaki należy rozwiązać dany problem programistyczny. Jednocześnie książka ta stanowi kompletny przewodnik po wszystkich ważnych dla programisty zagadnieniach. Nie zabrakło tu również informacji na temat korzystania z MySQL z poziomu języków programowania takich jak Perl, Java, czy PHP.

Dzięki tej książce:

- Zainstalujesz MySQL w różnych systemach operacyjnych
- Uruchomisz serwer MySQL i dowiesz się, z jakich programów klienckich korzystać
- Zaprojektujesz wydajną bazę danych
- Poznasz język SQL
- Zaznajomisz się ze specyficznymi funkcjami dostępnymi w MySQL
- Nauczysz się pisać aplikacje Javy, Perla i PHP wykorzystujące MySQL
- Poznasz podstawy administrowania serwerem bazodanowym

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

Spis treści

Projektowanie bazy danych

3

W pracy z systemem zarządzania relacyjną bazą danych, takim jak MySQL, pierwszym etapem procesu tworzenia i wykorzystywania bazy polega na zdefiniowaniu jej struktury. Projektowanie bazy danych, inaczej **modelowanie danych**, ma zasadnicze znaczenie dla pomyślnego i długotrwałego zarządzania informacjami. Wykorzystanie procesu zwanego **normalizacją** umożliwia całkowite wyeliminowanie redundancji oraz innego typu problemów, które mogłyby naruszyć integralność danych.

Omówione w niniejszym rozdziale techniki pomogą w zapewnieniu projektowanej bazie danych realności jej wykonania, wysokiej jakości i niezawodności. Zaprezentowany przykład — obsługa transakcji handlowych, w tym przechowywanie faktur i zapis wydatków — będzie wykorzystywany także w kolejnych rozdziałach niniejszej książki, a przedstawione zasady normalizacji znajdują zastosowanie w każdej nowo tworzonej bazie danych.

Normalizacja

Zagadnieniem normalizacji jako pierwszy zajął się — we wczesnych latach 70. — pracownik naukowy firmy IBM, E.F. Codd (opracował on również podstawy teorii relacyjnych baz danych). Relacyjna baza danych jest jedynie zbiorem danych, ułożonych w określony sposób. Dr Codd opracował natomiast szereg zasad, zwanych **postaciami normalnymi**, które ułatwiają zdefiniowanie wspomnianego ułożenia. W niniejszym rozdziale przedstawiono charakterystykę trzech postaci normalnych, które zazwyczaj są wystarczające dla większości projektów baz danych.

Przed rozpoczęciem normalizowania bazy danych konieczne jest określenie przeznaczenia budowanej aplikacji. Oznacza to, że niezbędne jest wnikliwe przeanalizowanie tego zagadnienia z klientem lub we własnym zakresie, gdyż sposób operowania danymi determinuje proces ich modelowania. Dlatego podczas lektury niniejszego rozdziału Czytelnik powinien zamiast oprogramowania MySQL korzystać z kartki i ołówka (dla jasności, projektowanie bazy danych jest niezbędnym etapem tworzenia wszystkich relacyjnych baz danych, nie tylko w MySQL).

Publikacje związane z bazami danych opierają się zazwyczaj na przykładach zbiorów muzycznych czy książkowych (drugi ich rodzaj był wykorzystywany przez Autora w innej książce: *Po prostu PHP. Techniki zaawansowane,* Helion 2002 (*PHP Advanced for the World Wide Web: Visual QuickPro Guide*)). Podczas lektury niniejszej książki Czytelnik zapozna się ze sposobem tworzenia bazy danych typu finansowo-księgowego. Zasadniczym jej zadaniem będzie rejestrowanie faktur i wydatków, choć może być w prosty sposób zmodyfikowana, tak by przechowywać informacje innego rodzaju, np. dane o czasie pracy nad projektem itp. W tabeli 3.1 zestawiono wstępną listę danych, jakie powinny być gromadzone w omawianej, przykładowej bazie danych.

Tabela 3.1. *Bazując na założonym sposobie wykorzystywania bazy danych opracowano listę wszystkich niezbędnych informacji, które powinny być w niej przechowywane*

Baza danych finansów	
Pozycja	**Przykład**
Numer faktury	1
Data wystawienia faktury	2002-04-20
Wartość faktury	30,28 USD
Opis faktury	Projekt HTML
Termin płatności	2002-05-11
Informacje o kliencie	Acme Industries, 100 Main Street, Anytown, NY 11111, (800) 555-1234
Wydatek	100 USD
Kategoria wydatku i opis	Opłaty za utrzymywanie serwisu internetowego *www.DMCinsights.com*
Data zapłaty	2002-01-26

Wskazówki

- Jednym z najlepszych sposobów określenia, jakiego typu informacje powinny być przechowywane w bazie danych, jest ustalenie rodzajów zadawanych pytań i informacji, jakie powinny się znaleźć w odpowiedziach.
- Choć niniejszy rozdział prezentuje sposób manualnego projektowania bazy danych, należy pamiętać, że istnieją gotowe aplikacje, przeznaczone do tego celu. Ich lista znajduje się w dodatku C *Źródła informacji*.

Klucze

Klucze stanowią tę część danych, która pomaga w identyfikowaniu danego wiersza informacji w tabeli (inną nazwą wiersza jest **rekord**). Istnieją dwa rodzaje kluczy, którymi będziemy operować: **klucze główne** i **klucze obce**. Klucz główny jest unikatowym identyfikatorem i musi spełniać określone reguły. Musi:

- zawsze posiadać jakąś wartość (nie może mieć wartości `NULL`);
- przechowywać stałą wartość (nigdy niezmienianą);
- posiadać niepowtarzalną wartość w każdym wierszu tabeli.

Najlepszym przykładem klucza głównego, który można spotkać w życiu codziennym, jest numer PESEL. Idea polega na tym, że każdy obywatel otrzymuje niepowtarzalny numer PESEL, który nie podlega żadnym zmianom. PESEL jest sztuczną konstrukcją, służącą do identyfikowania osób. Łatwo się można przekonać, że takie sztuczne narzucenie klucza głównego każdej tabeli stanowi najefektywniejszy sposób jej projektowania.

Drugim rodzajem kluczy są klucze obce. Stanowią one reprezentację kluczy Tabeli A w Tabeli B. Załóżmy, że dysponujemy bazą danych filmów, gdzie istnieją tabele **film** i **reżyser**. Klucz publiczny tabeli **reżyser** mógłby wówczas być odwzorowany w tabeli **film** jako klucz obcy. Idea ta stanie się łatwiejsza do zrozumienia po przeanalizowaniu założeń procesu normalizacji.

Aktualnie, oficjalnie MySQL obsługuje klucze obce tylko w przypadku wykorzystania tabel typu InnoDB (więcej informacji na temat typów tabel zawarto w rozdziale 11. *MySQL dla zaawansowanych*), w przeciwnym razie je ignoruje. Dlatego też występowanie kluczy obcych w MySQL ma raczej charakter teoretyczny a nie użytkowy, co powinno ulec zmianie w kolejnych wersjach oprogramowania.

W przedstawionej postaci baza danych *finanse* stanowi pojedynczą tabelę. Rozpoczęcie procesu normalizacji wymaga ustalenia przynajmniej jednego klucza głównego (klucze obce będą określane w kolejnych etapach).

Aby określić klucz główny:

1. Wyszukaj dowolne pole, które spełnia założenia klucza głównego.

   W przedstawionym przykładzie jedynymi danymi, które zawsze są niepowtarzalne, posiadają wartość i ich wartość nie może być zmieniana, są **numery faktur**. Pole to zostanie oznaczone jako klucz główny (ang. *primary key*), do czego posłuży symbol (*PK*) (rysunek 3.1).

2. Jeśli nie istnieje naturalny klucz główny, należy go wymyślić.

   Czasami jest konieczne utworzenie klucza głównego, gdyż nie istnieje wśród danych pole nadające się do tego celu. Nawet w przypadku stosowania identyfikatorów PESEL czy oznaczania książek numerem ISBN (ang. *International Standardized Book Number* — standardowy znormalizowany numer książki), które spełniają podane kryteria, dobrym rozwiązaniem jest utworzenie dodatkowego pola, przeznaczonego jednoznacznie na klucz główny.

Wskazówki

- MySQL dopuszcza użycie tylko jednego klucza głównego w tabeli, choć można go oprzeć na wielu kolumnach (zagadnienie to wykracza tematycznie poza zakres niniejszej książki).
- MySQL osiąga największą wydajność w przypadku, gdy kluczem głównym jest wartość ze zbioru liczb całkowitych. Jest to kolejny powód, dla którego identyfikatory ISBN, zawierające znaki myślnika, niezbyt dobrze nadają się do pełnienia funkcji klucza głównego.

Rysunek 3.1. *Pierwszy krok w procesie normalizacji bazy danych polega na określeniu klucza głównego — numeru faktury*

Relacje

Mówiąc o relacjach w bazie danych, mamy na myśli zależności danych jednej tabeli od danych drugiej. Relacje pomiędzy dwoma tabelami mogą mieć postać **jeden-do-jeden**, **jeden-do-wielu** lub **wiele-do-wielu**.

Zależność typu **jeden-do-jeden** występuje wtedy, gdy jedna i tylko jedna pozycja w tabeli A odnosi się do jednej i tylko jednej pozycji w tabeli B (np. każdy obywatel Polski posiada jeden identyfikator PESEL, a każdy identyfikator PESEL jest przypisany tylko do jednego obywatela. Żaden obywatel nie może posiadać dwóch numerów PESEL i żaden numer PESEL nie może być przydzielony dwóm obywatelom).

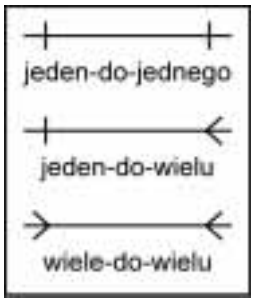
Relacja **jeden-do-wielu** ma miejsce wówczas, gdy jedna i tylko jedna pozycja w tabeli A dotyczy wielu pozycji w tabeli B. Określenie **kobieta** lub **mężczyzna** odnosi się do wielu osób, ale każda z osób może być tylko mężczyzną lub kobietą. Relacja jeden-do-wielu jest najczęściej spotykaną relacją pomiędzy tabelami bazy danych.

W końcu, zależność **wiele-do-wielu** występuje wtedy, gdy wiele pozycji w tabeli A może odnosić się do wielu pozycji w tabeli B. Przykładowo, album płytowy może zawierać piosenki wykonywane przez wielu artystów, a artyści mogą być autorami wielu albumów płytowych. Projektując bazę danych należy unikać relacji wiele-do-wielu, gdyż są one przyczyną powstawania redundancji danych i problemów związanych z ich integralnością.

Współzależność relacji i kluczy polega na tym, że klucz w jednej tabeli odnosi się zazwyczaj do pola danych w innej, o czym już wspominano. Zrozumienie zasad posługiwania się unikatowymi identyfikatorami oraz relacjami pozwala na przystąpienie do normalizowania bazy danych.

Wskazówki

- Modelowanie bazy danych opiera się na pewnej konwencji reprezentowania jej struktury. Zostanie ona zaprezentowana za pomocą szeregu rysunków, zaprezentowanych w niniejszym rozdziale. Symbole wspomnianych rodzajów relacji pokazano na rysunku 3.2.
- Wynikiem procesu projektowania bazy danych jest diagram element-związek (ang. *entity reletionship diagram*), uwzględniający reprezentację obramowanych tabel i symboli przedstawionych na rysunku 3.2.
- Słowo „relacyjny" w skrócie RDBMS pochodzi od tabel, które z technicznego punktu widzenia nazywają się **relacjami**.



Rysunek 3.2. *Prezentowane oznaczenia reprezentują relacje pomiędzy tabelami w procesie modelowania bazy danych*

Rysunek 3.3. *Zgodnie z założeniami pierwszej postaci normalnej dwie kolumny, które nie spełniały jej kryteriów, zostały podzielone na większą liczbę pól*

Pierwsza postać normalna

Pierwsza postać normalna bazy danych zakłada, że każda kolumna może zawierać tylko jedną wartość (kolumna jest wtedy czasami nazywana **atomową**). Tabela, która posiada jedno pole przeznaczone do przechowywania danych adresowych nie spełnia tego kryterium, gdyż przechowuje w pojedynczym polu pięć różnych danych — ulicę, miasto, województwo, kod pocztowy i czasami kraj. Podobnie, niezgodne z założeniem byłoby przechowywanie w jednym polu imienia i nazwiska osoby (choć niektórzy mogliby twierdzić, że imię i nazwisko stanowią całość, będąc elementem niepodzielnym).

W ramach prowadzonego procesu normalizacji zostanie dokonane sprawdzenie istniejącej struktury pod kątem zgodności z pierwszą postacią normalną.

Aby uczynić bazę danych zgodną z pierwszą postacią normalną:

1. Wyszukuj pola, które zawierają więcej niż jedną informację.

   Analizując tabelę 3.1 można zauważyć, że z kryteriami pierwszej postaci normalnej nie są zgodne dwie kolumny: *Informacje o kliencie* oraz *Kategoria wydatku i opis*. Pola dotyczące dat zawierają, co prawda, informacje o dniu, miesiącu i roku, jednak ich dalsze rozdzielanie nie byłoby uzasadnione.

2. Dokonaj podziału pól wybranych w pierwszym kroku (rysunek 3.3).

   Rozwiązanie problemu polega na wydzieleniu z *Informacji o kliencie* pól: *Nazwisko klienta*, *Ulica klienta*, *Stan klienta*, *Kod pocztowy klienta* i *Telefon klienta*, a następnie z *Kategorii wydatków i opisu* pól: *Kategoria wydatku*, *Opis wydatku*.

3. Upewnij się, że wszystkie utworzone w kroku 2. pola spełniają kryteria pierwszej postaci normalnej.

Druga postać normalna

Baza danych ma drugą postać normalną, jeżeli spełnia założenia pierwszej postaci normalnej (normalizacja przebiega stopniowo), a każda kolumna w tabeli, która nie jest kluczem, pozostaje w relacji tylko z kluczem głównym. Najbardziej oczywisty wniosek, jaki płynie z powyższego założenia jest taki, że baza danych nie ma drugiej postaci normalnej, jeżeli kilka rekordów tabeli posiada dokładnie taką samą wartość w danej kolumnie. Przykładowo, zamieszczenie listy producentów muzycznych wraz z nazwami ich albumów, spowodowałoby występowanie tych samych wartości w ramach jednej tabeli albumów.

Przyglądając się bazie danych *finansów* (rysunek 3.3) można zauważyć szereg niezgodności. Na początek, informacje na temat klienta nie zawsze będą się odnosiły tylko do jednej faktury (klient może dokonać kilku transakcji). Po drugie, informacje o wydatkach nie wiążą się z fakturami.

Przekształcenie bazy danych do drugiej postaci normalnej wymaga przeniesienia wymienionych kolumn do oddzielnych tabel, gdzie każda z wartości będzie zamieszczona tylko raz. W rzeczywistości normalizacja może być rozumiana jako proces tworzenia coraz większej liczby tabel, aż do momentu usunięcia wszystkich możliwych redundancji danych.

Rysunek 3.4. *Normalizacja bazy danych wymaga przeniesienia informacji nadmiarowych — takich jak dane klienta i informacje o wydatkach — do oddzielnych tabel*

Aby uczynić bazę danych zgodną z drugą postacią normalną:

1. Wyszukaj wszystkie pola, które nie zależą bezpośrednio od klucza głównego.

   Jak już wspomniano, informacjami, które nie dotyczą bezpośrednio konkretnych faktur, są informacje o kliencie oraz dane na temat wydatków.

2. Utwórz nowe tabele (rysunek 3.4).

   Najbardziej racjonalnym sposobem modyfikacji istniejącej struktury będzie utworzenie oddzielnych tabel *Klienci*, *Faktury* i *Wydatki*. Zgodnie z przyjętymi w książce regułami wizualizacji, reprezentacja bazy danych składa się z obramowanych tabel. W każdym bloku tabeli znajduje się nagłówek, który zawiera nazwę tabeli oraz cześć składającą się z nazw wszystkich kolumn.

Rysunek 3.5. *Każda tabela bazy danych powinna posiadać własny klucz główny, niezależnie od tego, czy jest to pole nadmiarowe — jak Klient ID — czy pole znaczące — jak Numer faktury*

Rysunek 3.6. *Wchodzące w skład tabeli Wydatki pole Kategoria wydatku powinno zostać wydzielone w postaci odrębnej tabeli*

Rysunek 3.7. *Nowym kluczom głównym zostały przypisane odpowiednie klucze obce (FK — ang. foreign key) oraz określony został charakter relacji (w obu przypadkach jeden-do-wielu)*

3. Przydziel lub utwórz nowe klucze główne (rysunek 3.5).

Do zapewnienia, że wszystkie nowo utworzone tabele posiadają klucz główny, służy technika opisana w wcześniejszej części tego rozdziału. Z uwagi na fakt, że zarówno tabela *Klienci*, jak i *Wydatki* nie posiada właściwego, unikatowego identyfikatora, konieczne było sztuczne jego utworzenie — *Klient ID* oraz *Wydatek ID*. Nazwa klienta byłaby być może niepowtarzalną wartością, a tym samym mogłaby być kluczem głównym, niemniej korzystniej jest zawsze posłużyć się w tym celu wartościami ze zbioru liczb całkowitych.

4. Powtórz kroki 1 – 3.

Z uwagi na fakt, że zostały utworzone nowe tabele z nowymi kluczami, należy się upewnić, że w efekcie wykonania tej operacji powstała baza zgodna z drugą postacią normalną. W prezentowanym przykładzie (rysunek 3.5) pozostaje jeden istotny problem — pole *Kategoria wydatku* może się odnosić do wielu rodzajów wydatków. Dlatego też zostanie utworzona nowa tabela o nazwie *Kategorie wydatków* (rysunek 3.6).

5. Utwórz niezbędne klucze obce, które określą właściwe relacje (rysunek 3.7).

Ostatnim krokiem na drodze do uzyskania zgodności z drugą postacią normalną jest uwzględnienie kluczy obcych i relacji, które będą określały sposób wzajemnego powiązania danych i tabel. Należy pamiętać, że klucz główny jednej tabeli stanowi z reguły klucz obcy innej tabeli. Jeżeli okaże się, że klucz główny danej tabeli nie funkcjonuje jako klucz obcy w innej, prawdopodobnie coś zostało pominięte (choć nie zawsze jest to prawdą).

Wskazówka

- Innym sposobem sprawdzenia zgodności z drugą postacią normalną jest przeanalizowanie relacji pomiędzy tabelami. Najlepszym rozwiązaniem jest utworzenie zależności typu jeden-do-wielu. Tablice, w których występują relacje wiele-do-wielu prawdopodobnie wymagają przekształcenia.

Trzecia postać normalna

Baza danych ma trzecią postać normalną, jeżeli jest zgodna z drugą postacią normalną i jeżeli każda kolumna, która nie jest kluczem, pozostaje niezależna od innych kolumn niebędących kluczami. Innymi słowy, wszystkie pola tabeli, które nie są kluczami, powinny być wzajemnie niezależne.

W przypadku poprawnego zrealizowania dwóch pierwszych kroków normalizacyjnych może się okazać, że wprowadzanie zmian na tym etapie nie będzie konieczne. Z drugiej strony, jeżeli w ramach wspomnianych działań zostało dokonanych wiele zmian, krok ten dostarcza możliwości ostatecznego sprawdzenia projektu. Załóżmy przykładowo, że do każdej faktury przypisywane jest nazwisko osoby odpowiedzialnej za kontakt oraz adres jej poczty elektronicznej (rysunek 3.8). Rzecz w tym, że wspomniane informacje odnoszą się do klienta, a nie do faktury, tym samym baza danych nie spełnia kryteriów trzeciej postaci normalnej. Prawidłowe rozwiązanie polega na dodaniu tych pól do tabeli *Klienci* (rysunek 3.9).

Rysunek 3.8. *Modyfikowanie struktury bazy danych może wprowadzać nieco zmieszania do projektu, który przestaje spełniać kryteria normalizacji ze względu na pola: osoba kontaktowa i adres e-mail*

Rysunek 3.9. *Prawidłowe umiejscowienie danych z pól: osoba kontaktowa i adres e-mail wymaga przeniesienia ich do tabeli Klienci*

Wskazówki

- Po zakończeniu szkicowania bazy danych na kartce dobrym pomysłem jest przeniesienie go do arkusza kalkulacyjnego, który odwzoruje powstały projekt (można też posłużyć się specjalnie utworzonym do tego celu narzędziem). Taki dokument mógłby służyć jako dobre źródło informacji dla projektantów witryny internetowej, a także jako dokument przekazywany klientowi po zakończeniu projektu.
- Normalizacja stanie się jeszcze czynnością o jeszcze większym znaczeniu z chwilą, gdy MySQL zacznie wymuszać stosowanie kluczy obcych (w wersjach 4.1 i późniejszych).

Zaniechanie normalizacji

Mimo iż zapewnienie, że baza danych spełnia kryteria trzeciej postaci normalnej zapewnia jej stabilność i trwałość, nie istnieje konieczność normalizowania każdej bazy danych, z jaką będziemy pracować. Jednak zanim podważymy słuszność tej metody, należy pamiętać, że takie działanie może mieć poważne konsekwencje w dłuższej perspektywie.

Dwoma zasadniczymi przyczynami zaniechania normalizacji są wygoda i wydajność. Mniejszą liczbą tabel łatwiej się manipuluje i łatwiej jest też wtedy zrozumieć strukturę bazy danych. Co więcej, ze względu na ich złożoną naturę, znormalizowane bazy danych stają się zazwyczaj wolniejsze w trakcie uaktualniania, pobierania i modyfikowania danych. Normalizacja oznacza wybranie większego poziomu integralności danych i skalowalności kosztem prostoty i szybkości działania. Z drugiej strony, istnieje wiele sposobów poprawienia wydajności baz danych, podczas gdy metod przeciwdziałania utracie danych, wynikającej z zastosowania wadliwego projektu, jest niewiele.

Typy danych MySQL

Po zdefiniowaniu wszystkich wymaganych tabel i kolumn konieczne jest określenie typu danych przechowywanych w każdym z pól. Podczas tworzenia bazy danych (co zostanie przedstawione w następnym rozdziale) będzie wymagane określenie rodzaju informacji, które będą przechowywane w każdym z pól. Niemal każda baza danych opiera się na trzech ich kategoriach:

- tekst;
- liczby;
- data i czas.

W każdej z wymienionych grup wyróżnia się kilka odmian typów danych, z których pewne są charakterystyczne jedynie dla MySQL. Właściwy wybór typu dla danej kolumny wpływa nie tylko na rodzaj informacji, jakie mogą być w niej gromadzone oraz na sposób ich przechowywania, ale również na całkowitą wydajność bazy danych. Większość dostępnych w MySQL typów danych została zestawiona w tabeli 3.2. Zamieszczono tam także informacje o rozmiarze oraz krótki opis każdego z nich.

Wiele typów pozwala na określenie opcjonalnego atrybutu `Długość` (nawiasy kwadratowe — `[]` — oznaczają, że pomiędzy nimi można wstawić parametr opcjonalny, tymczasem nawiasy okrągłe odpowiadają argumentom obowiązkowym). Typy liczbowe mogą być definiowane jako `UNSIGNED` — co ogranicza wartości kolumny do liczb dodatnich i zera — lub `ZEROFILL`, co oznacza, że wolne miejsce zostanie wypełnione zerami (typy `ZEROFILL` są jednocześnie typami `UNSIGNED`). Z kolei z poszczególnymi typami dat są związane różne sposoby ich wykorzystania. Opis zagadnienia znajduje się w podręczniku dostępnym pod adresem *www.mysql.com/doc/D/A/DATETIME.html*. Zazwyczaj jednak są stosowane podstawowe pola typu `TIME` i `DATE`, nie ma więc potrzeby analizowania ich zawiłości. Omówienia wymagają także dwa rozszerzenia typu `TEXT` — `ENUM` i `SET`. Oba pozwalają na zdefiniowanie w trakcie tworzenia tabeli serii akceptowalnych wartości.

Pole typu ENUM może zawierać tylko jedną z kilku tysięcy możliwych wartości, podczas gdy pole SET może się składać z kilku wartości, przy czym ich liczba nie może przekraczać 64. Z typami ENUM i SET wiążą się dwa problemy — nie są one obsługiwane przez inne bazy danych, a ich użycie jest niezgodne z zasadami normalizacji.

Tabela 3.2. *Większość typów kolumn dostępnych w bazach danych MySQL*

Typy danych MySQL		
Typ	**Rozmiar**	**Opis**
CHAR[Długość]	Liczba bajtów	Pole o stałej długości; długość: 0 – 255 znaków.
VARCHAR(Długość)	Długość ciągu + 1 bajt	Pole o stałej długości; długość: 0 – 255 znaków.
TINYTEXT	Długość ciągu + 1 bajt	Ciąg tekstowy o maksymalnej długości 255 znaków.
TEXT	Długość ciągu + 2 bajty	Ciąg tekstowy o maksymalnej długości 65.536 znaków.
MEDIUMTEXT	Długość ciągu + 3 bajty	Ciąg tekstowy o maksymalnej długości 16.777.215 znaków.
LONGTEXT	Długość ciągu + 4 bajty	Ciąg tekstowy o maksymalnej długości 4.294.967.295 znaków.
TINYINT[Długość]	1 bajt	Liczba z zakresu od –128 do 128 lub 0 do 255 jeżeli jest typu UNSIGNED.
SMALLINT[Długość]	2 bajty	Liczba z zakresu od –32768 do 32768 lub 0 do 65535 jeżeli jest typu UNSIGNED.
MEDIUMINT[Długość]	3 bajty	Liczba z zakresu od –8.388.608 do 8.388.607 lub 0 do 16.777.215 jeżeli jest typu UNSIGNED.
INT[Długość]	4 bajty	Liczba z zakresu od –2.147.483.648 do 2.147.483.647 lub 0 do 4.294.967.295 jeżeli jest typu UNSIGNED.
BIGINT[Długość]	8 bajtów	Liczba z zakresu od –9.223.372.036.854.775.808 do 9.223.372.036.854.775.807 lub 0 do 18.446.744.073.709.551.615 jeżeli jest typu UNSIGNED.
FLOAT	4 bajty	Mała wartość zmiennoprzecinkowa.
DOUBLE[Długość,Pozycje]	8 bajtów	Duża wartość zmiennoprzecinkowa.
DECIMAL[Długość,Pozycje]	Długość + 1 lub Długość + 2 bajty	Wartość typu DOUBLE zapisana jako ciąg tekstowy, dla którego możliwe jest ustalenie liczby pozycji po przecinku.
DATE	3 bajty	Data w formacie: RRRR-MM-DD.
DATETIME	8 bajtów	Data i czas w formacie: RRRR-MM-DD GG:MM:SS.
TIMESTAMP	4 bajty	Znacznik czasowy w formacie: GG:MM:SS.
TIME	3 bajty	Znacznik czasowy w formacie GG:MM:SS.
ENUM	1 lub 2 bajty	Wyliczenie, które pozwala na to, by każda kolumna posiadała jedną z kilku możliwych wartości.
SET	1,2,3,4 lub 8 bajtów	Typ podobny do ENUM z tą różnicą, że może posiadać więcej niż jedną dopuszczalną wartość.

Aby wybrać typ danych:

1. Określ czy kolumna będzie przechowywała dane tekstowe, liczbowe czy też daty.

   Zazwyczaj jest to prosty i oczywisty etap. Do przechowywania wartości liczbowych, takich jak kody pocztowe czy sumy pieniężne, które będą przechowywane wraz z dodatkowymi znakami (jak znak myślnika czy oznaczenie waluty), używa się pól tekstowych, choć zapisanie ich jako wartości liczbowych daje lepsze rezultaty. Problem formatowania może być rozwiązywany w innymi miejscu.

2. Wybierz dla danej kolumny odpowiedni typ z danej kategorii.

   Mając na uwadze wysoką wydajność bazy danych, warto pamiętać, że:

   - pola o stałej długości (jak `CHAR`) są zazwyczaj szybciej przetwarzane niż pola o zmiennej długości (jak `VARCHAR`), choć z drugiej strony zajmują więcej przestrzeni dyskowej. Więcej informacji na ten temat zamieszczono we wskazówce;
   - rozmiar każdego z pól powinien być ograniczony do najmniejszej możliwej wartości, którą można wyznaczyć określając największą możliwą wartość wprowadzaną do danego pola. Przykładowo, jeżeli największa wartość pola *Klient ID* będzie rzędu setek, to dla danej kolumny powinno się wybrać trzycyfrowy typ `SMALLINT` bez znaku (`UNSIGNED`) — pozwoli on na wprowadzenie wartości od 0 do 999.

     Należy pamiętać, że wprowadzenie pięcioznakowego ciągu tekstowego do pola typu `CHAR(2)` spowoduje obcięcie trzech ostatnich znaków. Ta prawidłowość znajduje zastosowanie we wszystkich typach o określonej długości (`CHAR`, `VARCHAR`, `INT` itp.).

3. Ustal maksymalną długość kolumn tekstowych lub liczbowych oraz dołącz, jeśli to konieczne, inne atrybuty (jak np. `UNSIGNED`) (tabela 3.3).

   Zamiast rozpisywania się o sposobach i przyczynach takiego, a nie innego zdefiniowania wszystkich 21 przykładowych kolumn, wszystkie ich własności zestawiono w tabeli 3.3. Niektórzy programiści mogą mieć odmienne propozycje. Najistotniejsze jest jednak, aby dostosować każdy typ do rozmiarów przechowywanych informacji, zamiast korzystać zawsze z podstawowych (nieefektywnych) typów `TEXT` i `INT`.

CHAR a VARCHAR

Co do przewagi któregokolwiek z tych dwóch podobnych do siebie typów wciąż trwają dyskusje. Oba przechowują ciągi tekstowe i mogą być definiowane z podaniem maksymalnej jego długości. Podstawowa różnica polega na tym, że jakiekolwiek dane zapisane jako `CHAR` zawsze będą zapisywane jako ciąg tekstowy o długości określonej dla danej kolumny (wypełnienie znakami spacji). Z kolei długość ciągów tekstowych typu `VARCHAR` jest równa długości przechowywanego ciągu danych.

Wynika z tego, że:

- **kolumny `VARCHAR` zajmują mniej miejsca na dysku;**
- **kolumny `CHAR` są przetwarzane szybciej niż `VARCHAR`, o ile nie są stosowane typy tabel InnoDB (więcej informacji na ten temat zamieszczono w rozdziale 11. *MySQL dla zaawansowanych*).**

Trzeba przyznać, że w większości przypadków różnica w wielkości zajmowanego miejsca na dysku oraz w szybkości pomiędzy oboma typami jest niezauważalna, przez co rozważanie tego problemu nie ma szczególnego znaczenia.

Istnieje jeszcze jedna, mniej istotna różnica pomiędzy omawianymi typami danych — MySQL usuwa nadmiarowe znaki spacji z kolumn `CHAR` podczas pobierania danych a z kolumn `VARCHAR` podczas ich wstawiania.

Wskazówki

- Wiele z nazw typów posiada synonimy, np. `INT` — `INTEGER`, `DEC` — `DECIMAL` itd.
- Pole typu `TIMESTAMP` jest uaktualniane automatycznie podczas wykonywania polecenia `INSERT` czy `UPDATE`, nawet jeżeli dla danego pola nie określono żadnej wartości. W przypadku, gdy tabela zawiera więcej pól typu `TIMESTAMP`, podczas realizacji polecenia `INSERT` lub `UPDATE` uaktualniane jest tylko pierwsze z nich.
- Dostępny jest również typ `BLOB`, będący odmianą typu `TEXT`, który pozwala na przechowywanie w tabeli plików binarnych. Przykład użycia zostanie zaprezentowany w rozdziale 9. *Techniki programowania baz danych*.

Tabela 3.3. *Dobór optymalnego typu danych dla każdego z pól jest często zaniedbywaną czynnością*

Baza danych finansów		
Nazwa kolumny	**Tabela**	**Typ kolumny**
Numer faktury	Faktury	`SMALLINT(4) UNSIGNED`
Klient ID	Faktury	`SMALLINT(3) UNSIGNED`
Data wystawienia faktury	Faktury	`date`
Wartość faktury	Faktury	`decimal(10,2) unsigned`
Opis faktury	Faktury	`tinytext`
Klient ID	Klienci	`smallint(3) unsigned`
Nazwa klienta	Klienci	`varchar(40)`
Ulica klienta	Klienci	`varchar(80)`
Miasto klienta	Klienci	`varchar(30)`
Stan klienta	Klienci	`CHAR(2)`
Kod pocztowy klienta	Klienci	`mediumint(5) unsigned`
Telefon klienta	Klienci	`varchar(14)`
Osoba kontaktowa	Klienci	`varchar(40)`
Adres e-mail kontaktowy	Klienci	`varchar(60)`
Wydatek ID	Wydatki	`smallint(4) unsigned`
Kategoria wydatku ID	Wydatki	`tinyint(3) unsigned`
Wartość wydatku	Wydatki	`decimal(10,2) unsigned`
Opis wydatku	Wydatki	`tinytext`
Data zapłaty	Wydatki	`date`
Kategoria wydatku ID	Kategorie wydatków	`tinyint(3) unsigned`
Kategoria wydatku	Kategorie wydatków	`varchar(30)`

Wartości domyślne i NULL

Zgodnie z przedstawionymi wcześniej informacjami, definiując typy danych jest możliwe dołączanie atrybutów, takich jak UNSIGNED czy ZEROFILL. Istnieją jeszcze dwie wartości, z których jedna informuje, czy w kolumnie dopuszcza się wartości NULL, a druga wskazuje domyślną wartość danego pola.

W przypadku programowania lub tworzenia bazy danych użycie NULL jest jednoznaczne z poinformowaniem, że dane pole nie przechowuje żadnej wartości (lub wartość jest nieznana). Rozwiązaniem idealnym byłoby oczywiście przypisanie każdemu rekordowi bazy danych pewnej konkretnej wartości. W rzeczywistości jednak takie sytuacje zdarzają się rzadko. Dołączając do deklaracji typu ciąg NOT NULL możliwe jest wymuszenie takiego ograniczenia na danym polu. Przykładowo, deklaracja klucza głównego mogłaby wyglądać następująco:

```
klient_id SMALLINT(3) UNSIGNED NOT NULL
```


Tabela 3.4. *W wyniku dalszych prac nad projektem bazy danych niektóre kolumny zostały uzupełnione o wartości NOT NULL i DEFAULT*

Baza danych finansów		
Nazwa kolumny	**Tabela**	**Typ kolumny**
Numer faktury	Faktury	SMALLINT(4) UNSIGNED NOT NULL DEFAULT 0
Klient ID	Faktury	SMALLINT(3) UNSIGNED
Data wystawienia faktury	Faktury	date NOT NULL
Wartość faktury	Faktury	decimal(10,2) unsigned NOT NULL
Opis faktury	Faktury	tinytext
Klient ID	Klienci	smallint(3) unsigned NOT NULL DEFAULT 0
Nazwa klienta	Klienci	varchar(40) NOT NULL
Ulica klienta	Klienci	varchar(80)
Miasto klienta	Klienci	varchar(30)
Stan klienta	Klienci	char(2)
Kod pocztowy klienta	Klienci	mediumint(5) unsigned
Telefon klienta	Klienci	varchar(14)
Osoba kontaktowa	Klienci	varchar(40)
Adres e-mail kontaktowy	Klienci	varchar(60)
Wydatek ID	Wydatki	smallint(4) unsigned NOT NULL DEFAULT 0
Kategoria wydatku ID	Wydatki	tinyint(3) unsigned
Wartość wydatku	Wydatki	decimal(10,2) unsigned NOT NULL
Opis wydatku	Wydatki	tinytext
Data zapłaty	Wydatki	date
Kategoria wydatku ID	Kategorie wydatków	tinyint(3) unsigned
Kategoria wydatku	Kategorie wydatków	varchar(30)

Podczas budowania tabeli możliwe jest także przypisywanie wartości domyślnych. W sytuacjach, gdy duża część rekordów posiadać będzie tę samą zawartość, wcześniejsze ustalenie wartości domyślnych pozwoli na wyeliminowanie konieczności wprowadzania pewnych danych w trakcie uzupełniania bazy danych, o ile oczywiście nie odbiegają one od wartości standardowych. Przykładem może być:

```
plec ENUM('M','K') DEFAULT 'K'
```

Obie techniki zostały uwzględnione w tabeli 3.4.

Wskazówki

- Zgodnie ze sztuką projektowania bazy danych oraz zasadami funkcjonowania MySQL klucze główne nie mogą zawierać wartości `NULL`.
- Jeżeli kolumna `ENUM` zostanie określona jako `NOT NULL`, wówczas pierwsza dopuszczalna wartość stanie się wartością domyślną.
- Istotnym jest, aby mieć świadomość, że `NULL` nie jest wartością równoznaczną z zerem, pustym ciągiem (`""`) czy znakiem spacji (`" "`).

Indeksy

Indeksy składają się na szczególny system, wykorzystywany do poprawienia całościowej wydajności bazy danych. Ustalając indeksy w ramach tabeli wskazuje się kolumny, które są w danej tabeli ważniejsze od innych kolumn tej samej tabeli (definicja dla laików). W rzeczywistości, do przechowywania indeksów i efektywnego nimi zarządzania MySQL tworzy oddzielne pliki.

MySQL pozwala na utworzenie maksymalnie 32 indeksów dla jednej tabeli, a każdy z nich może obejmować do 16 kolumn. Wykorzystanie indeksów wielokolumnowych nie musi się wydawać takie oczywiste, jednak stają się użyteczne w przypadku częstego przeszukiwania grupy tych samych kolumn (np. zawierających dane na temat imienia, nazwiska, miasta i województwa).

Z drugiej strony, w stosowaniu indeksów wskazany jest umiar. Zwiększają one co prawda szybkość odczytu danych z bazy, ale spowalniają proces ich modyfikacji (z uwagi na fakt, że zmiany muszą być odwzorowane także w indeksach). Najlepszym zastosowaniem dla indeksów jest użycie ich w kolumnach, które:

- są często wykorzystywane w części `WHERE` zapytań;
- są często wykorzystywane w części `ORDER BY` zapytań;
- cechują się różnorodnością wartości (kolumny, w których wartości powtarzają się wielokrotnie nie powinny być indeksowane).

W celu zwiększenia wydajności kolumna klucza głównego jest indeksowana automatycznie przez MySQL.

W MySQL wyróżnia się trzy typy indeksów: `INDEX`, `UNIQUE` (narzucający konieczność wprowadzania unikatowej wartości w każdym wierszu) oraz `PRIMARY KEY` (będący szczególną postacią indeksu `UNIQUE`). Propozycje indeksów dla bazy danych finansów zestawiono w tabeli 3.5.

Tabela 3.5. *W celu zwiększenia wydajności baza danych została wzbogacona o kilka (choć nie jest ich zbyt wiele) indeksów, które pozwolą jej na efektywniejsze pobieranie przechowywanych informacji*

Indeksy bazy danych finansów	
Kolumna	**Typ indeksu**
Numer faktury	`PRIMARY KEY`
Klient ID	`PRIMARY KEY`
Wydatek ID	`PRIMARY KEY`
Kategoria wydatku ID	`PRIMARY KEY`
Data wystawienia faktury	`INDEX`
Nazwa klienta	`INDEX` (lub `UNIQUE`)

Ostatnim atrybutem kolumny, który często wykorzystuje się w połączeniu z indeksem, jest AUTO_INCREMENT. Definicja pola zawierającego tę własność wygląda następująco:

```
klient_id SMALLINT(3) UNSIGNED NOT NULL
AUTO_INCREMENT
```

Określa ona obowiązek wprowadzania w danym polu kolejnej logicznej wartości z danej serii. Jeśli kolumna jest kolumną przechowującą wartości typu liczb całkowitych, to w trakcie dodawania do tabeli nowego rekordu w danym polu zostanie wprowadzona kolejna liczba całkowita.

Wskazówki

- AUTO_INCREMENT w MySQL jest odpowiednikiem sekwencji w Oracle.
- Indeksy stosowane w kolumnach o zmiennej długości cechuje mniejsza wydajność. Ogólnie, stosowanie pól, których długość nie jest stała, spowalnia pracę MySQL.
- W trakcie tworzenia indeksów można im nadawać nazwy (zobacz rozdział 4. *SQL*). Jeżeli jednak nie zostaną one określone, nazwą indeksu stanie się nazwa kolumny, której dotyczy.

Końcowy etap projektu

Ostatnim etapem projektowania bazy danych jest zastosowanie odpowiedniej konwencji nazewnictwa. Co prawda, MySQL nie narzuca zasad nazywania baz danych, tabel czy kolumn, istnieją jednak pewne sprawdzone reguły, których należy przestrzegać (niektóre z nich są nawet obowiązkowe):

- używanie znaków alfanumerycznych;
- ograniczenie maksymalnej długości nazw do 64 znaków (jest to wymóg MySQL);
- używanie znaków podkreślenia (_) w celu rozdzielania wyrazów;
- korzystanie tylko z małych liter (choć nie jest to rzecz obowiązkowa);
- używanie liczby mnogiej w oznaczaniu tabel i pojedynczej w definiowaniu kolumn;
- dołączanie `id` (lub `ID`) do nazw kolumn kluczy głównych i obcych;
- umieszczanie kluczy głównych w początkowej części tabeli, a w dalszej kolejności kluczy obcych;
- nazwy pól powinny mieć charakter opisowy;
- nazwy pól, z wyjątkiem kluczy, powinny być unikatowe w obrębie wszystkich tabel.

Zamieszczone powyżej reguły mają jedynie charakter zalecenia, ich przestrzeganie, poza koniecznością posługiwania się znakami alfanumerycznymi bez znaków spacji, nie jest zatem obowiązkowe. Część programistów preferuje używanie wielkich liter do rozdzielania wyrazów (zamiast znaku podkreślenia). Inni z kolei uwzględniają w nazwie kolumny jej typ. Najistotniejszym jest jednak to, by przestrzegać ustalonej konwencji.

Ostateczny projekt bazy danych przedstawiono w tabeli 3.6. Sposób jej utworzenia zostanie zaprezentowany w następnym rozdziale.

Wskazówki

- Systemy Unix, w przeciwieństwie do systemów Windows, rozróżniają w nazwach baz danych i tabel wielkość liter. W nazwach kolumn wielkość znaków jest zawsze rozróżniana.
- Drobiazgowe przestrzeganie określonych zasad projektowania baz danych pozwala na ograniczenie liczby błędów, które mogą się pojawić w trakcie programowania interfejsu bazy danych, o czym będzie mowa w rozdziałach 6., 7. i 8.

Tabela 3.6. *Ostatni etap projektu polega na zastosowaniu odpowiedniej konwencji nazewnictwa*

Baza danych finansów		
Nazwa kolumny	**Tabela**	**Typ kolumny**
faktura_id	faktury	SMALLINT(4) UNSIGNED NOT NULL DEFAULT 0
klient_id	faktury	SMALLINT(3) UNSIGNED
data_faktury	faktury	date NOT NULL
wartosc_faktury	faktury	decimal(10,2) unsigned NOT NULL
opis_faktury	faktury	tinytext
klient_id	klienci	smallint(3) unsigned NOT NULL DEFAULT 0
nazwa_klienta	klienci	varchar(40) NOT NULL
ulica_klienta	klienci	varchar(80)
miasto_klienta	klienci	varchar(30)
stan_klienta	klienci	char(2)
kod_pocztowy_klienta	klienci	mediumint(5) unsigned
telefon_klienta	klienci	varchar(14)
osoba_kontaktowa	klienci	varchar(40)
email_kontaktowy	klienci	varchar(60)
wydatek_id	wydatki	smallint(4) unsigned NOT NULL DEFAULT 0
kategoria_wydatku_id	wydatki	tinyint(3) unsigned
wartosc_wydatku	wydatki	decimal(10,2) unsigned NOT NULL
opis_wydatku	wydatki	tinytext
data_zaplaty	wydatki	date
kategoria_wydatku_id	kategorie_wydatkow	tinyint(3) unsigned
kategoria_wydatku	kategorie_wydatkow	varchar(30)