PHP. Programowanie

Autor: Leon Atkinson
Tłumaczenie: Jarosław Dobrzański
ISBN: 83-7197-967-3
Tytuł oryginału: Core PHP Programming, 2E
Format: B5, stron: 674

Książka „PHP. Programowanie." to praktyczny przewodnik po PHP4 dla programistów stron internetowych. Jeden z najlepszych programistów PHP — Leon Atkinson, uczy wszystkiego, co potrzebujesz, by stworzyć dobrą i szybką aplikację sieciową. Dokładnie opisuje składnię PHP i kluczowe elementy języka. Atkinson przedstawia także najważniejsze funkcje PHP, w tym funkcje wejścia-wyjścia, przetwarzania danych, matematyczne, daty, czasu, konfiguracji, funkcje umożliwiające współpracę z bazami danych, funkcje graficzne i sieciowe. Prezentuje on również działanie PHP na przykładach realizujących sortowanie, przeszukiwanie, analizę łańcuchów i inne zadania. Opisane zostały także praktyczne metody tworzenia i diagnostyki programów w PHP4.

Książka PHP. Programowanie zawiera:

- Jasny i szczegółowy opis składni i funkcji PHP
- Dokładny opis integracji PHP z bazami danych
- Techniki tworzenia i optymalizacji skryptów, pod kątem ich wydajności i łatwej rozbudowy
- Praktyczne techniki diagnostyczne, ułatwiające znalezienie i poprawienie błędów

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

Spis treści

17

Integracja z bazami danych

W tym rozdziale:

- Tworzenie tabel HTML z zapytań SQL.
- Śledzenie odwiedzających za pomocą identyfikatorów sesji.
- Przechowywanie danych w bazie.
- Warstwy abstrakcyjne baz danych.

PHP ściśle współpracuje z wieloma bazami danych. Jeżeli wewnętrzna współpraca z określoną bazą nie istnieje, zawsze można skorzystać z ODBC, który jest standardem dla zewnętrznych sterowników baz danych. Regularnie pojawiają się moduły, umożliwiające współpracę z nowymi bazami danych. Programiści PHP twierdzą: „Dajcie nam do testowania swój produkt, a stworzymy kod obsługujący go w PHP".

MySQL jest niewątpliwie najbardziej popularną bazą danych wśród programistów PHP. Poza tym, że jest darmowa, nadaje się do zastosowań sieciowych z uwagi na dużą szybkość. W przykładach tu opisanych zakładam posiadanie bazy danych MySQL. Można ją pobrać z witryny MySQL pod adresem <*http://www.mysql.com*> i zainstalować w swoim systemie lub pokusić się o dostosowanie opisanych tu przykładów do współpracy z inną bazą.

Większość relacyjnych baz danych używa strukturalnego języka zapytań (SQL). Jest to język czwartej generacji, co oznacza, że przypomina bardziej tekst w języku angielskim niż kod źródłowy w PHP. Omówienie samego SQL wykracza poza ramy tej książki. W przypadku znikomej wiedzy na ten temat, radzę przeglądnąć listę materiałów wymienioną w dokumentacji na stronie domowej MySQL: <*http://www.mysql.com/doc.html*>. Alternatywą może być książka, taka jak *Hands-On SQL* Roberta Grotha i Davida Gerbera, wydana przez Prentice Hall.

Tworzenie tabel HTML z zapytań SQL

Jednym z prostszych zadań, jakie można wykonać za pomocą bazy danych i PHP, jest pobranie danych z bazy i wyświetlenie ich w tabeli HTML. Tabela taka może zawierać np. listę towarów na sprzedaż, listę projektów, listę serwerów DNS i ich czasy reakcji. Na potrzeby przedstawionego tu przykładu wykorzystany zostanie pierwszy z tych przykładów. Załóżmy, że supermarket chce zamieścić na stronie internetowej listę towarów na sprzedaż. W tym celu trzeba stworzyć stronę, która wyświetla zawartość bazy danych. Zostanie tu zastosowana baza `test`, tworzona podczas instalacji MySQL. Skrypt PHP, przeglądający katalog produktów, będzie znajdował się na tym samym komputerze, na którym zainstalowany jest serwer bazy danych.

Pierwszym krokiem jest utworzenie tabeli. Listing 17.1 pokazuje kod SQL, tworzący prostą tabelę, składającą się z trzech kolumn. Tabela nazywa się `katalog`. Zawiera ona kolumnę, zwaną `ID`, która zawiera wartości całkowite, maksymalnie jedenastocyfrowe. Kolumna ta nie może być pusta i nowym wierszom automatycznie będą przyporządkowywane kolejne wartości. Ostatni wiersz definicji określa `ID` jako klucz podstawowy. Powoduje to, że wartości w tej kolumnie są traktowane jako indeksy i uniemożliwia dublowanie się kluczy. Pozostałe dwie kolumny to `Name` i `Price`.

Listing 17.1. *Tworzenie tabeli „katalog”*

```
CREATE TABLE katalog
(
        ID INT(11) NOT NULL AUTO_INCREMENT,
        Nazwa VARCHAR(32),
        cena FLOAT(6,2),
        PRIMARY KEY (ID)
);
```

`Name` to łańcuch znaków o zmiennej długości, maksymalnie do 32 znaków. `Price` to liczba zmiennoprzecinkowa z sześcioma cyframi przed i dwoma po przecinku. Format ten nadaje się do zapisu cen.

Kolejnym krokiem jest umieszczenie elementów w tabeli. Jako że jest to tylko demonstracja, wprowadzimy tu parę przykładowych produktów, jakie można nabyć w supermarkecie wraz z wymyślonymi cenami. W tym celu użyta zostanie instrukcja `INSERT`. Listing 17.2 jest przykładem opisanej procedury.

Listing 17.2. *Wprowadzanie danych do tabeli „katalog”*

```
INSERT INTO katalog (Nazwa, Cena) VALUES ('Szczoteczka', 1.79);
INSERT INTO katalog (Nazwa, Cena) VALUES ('Grzebien', 0.95);
INSERT INTO katalog (Nazwa, Cena) VALUES ('Pasta do zebow', 5.39);
INSERT INTO katalog (Nazwa, Cena) VALUES ('Nic dentystyczna', 3.50);
INSERT INTO katalog (Nazwa, Cena) VALUES ('Szampon', 2.50);
INSERT INTO katalog (Nazwa, Cena) VALUES ('Odzywka', 3.15);
INSERT INTO katalog (Nazwa, Cena) VALUES ('Dezodorant', 1.50);
INSERT INTO katalog (Nazwa, Cena) VALUES ('Zel', 6.25);
INSERT INTO katalog (Nazwa, Cena) VALUES ('Ostrza do golarki', 2.99);
INSERT INTO katalog (Nazwa, Cena) VALUES ('Szczotka', 1.15);
```

Każda instrukcja SQL kończy się średnikiem, podobnie jak w PHP. Informujemy serwer MySQL, że chcemy wstawić wiersz do tabeli `katalog` i że podamy tylko nazwę produktu i cenę. Jako że pomijamy tu `ID`, MySQL stworzy go sam. Wynika to ze zdefiniowania tej kolumny jako `AUTO_INCREMENT`. Słowo kluczowe `VALUES` informuje serwer, że za chwilę zostaną przesłane wartości opisane w poprzednim poleceniu. Wartości przesyłane są w apostrofach, zgodnie ze standardem przyjętym w SQL.

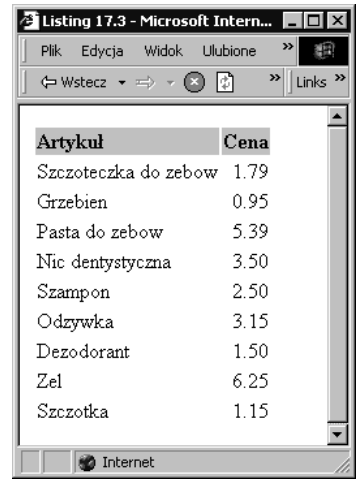
Aby sprawdzić, czy operacja się powiodła, listing 17.1 wyświetla wartości, jakie otrzymalibyśmy, wybierając wszystko z tabeli `katalog` z poziomu klienta MySQL. Realizuje to polecenie:

```
SELECT * FROM katalog;
```

po wydaniu którego otrzymujemy:

```
ID      Nazwa                   Cena
1       Szczoteczka             1.79
2       Grzebien                0.95
3       Pasta do zebow          5.39
4       Nic dentystyczna        3.50
5       Szampon                 2.50
6       Odzywka                 3.15
7       Dezodorant              1.50
8       Zel                     6.25
9       Ostrza do golarki       2.99
10      Szczotka                1.15
10      rows in set (0.01 sec)
```

Ostatnim krokiem jest napisanie skryptu PHP, który pobiera zawartość tabeli i przekształca ją w tabelę HTML. Listing 17.3 przedstawia kod PHP, pobierający nazwy oraz ceny i wyświetlający je w tabeli HTML. Na początku należy połączyć się z serwerem bazy danych. Realizuje to funkcja `mysql_pconnect`. Pobiera ona nazwę hosta, nazwę użytkownika i hasło. Zwykle tworzę w moich bazach MySQL użytkownika o nazwie `httpd` bez hasła. Ograniczam również tego użytkownika tylko do połączeń z lokalnego serwera. Jego nazwa jest taka sama, jak nazwa użytkownika UNIX, który wykonuje skrypty, czyli serwera sieciowego. Jeżeli wynajmujemy miejsce na serwerze, możemy mieć przyporządkowane inne nazwy użytkownika i bazy danych, co wymaga zmiany odpowiednich argumentów w przykładzie.



Rysunek 17.1. *Tworzenie tabeli HTML z wynikami zapytania*

Listing 17.3. *Tworzenie tabeli HTML z wynikami zapytania*

```
<?
        //połączenie z serwerem i sprawdzenie, czy się powiodło
        if(!($dbLink = mysql_pconnect("localhost", "httpd", "")))
        {
                print("Nie można połączyć się z bazą!<BR>\n");
                print("Zakończenie skryptu!<BR>\n");
                exit();
        }

        //wybór bazy danych i sprawdzenie rezultatów operacji
        if(!mysql_select_db("test", $dbLink))
        {
                print("Nie można wybrać bazy test!<BR>\n");
                print("Zakończenie skryptu!<BR>\n");
                exit();
        }

        // pobranie całej zawartości tabeli
        $Query = "SELECT Nazwa, Cena ";
        $Query .= "FROM Katalog ";
        if(!($dbResult = mysql_query($Query, $dbLink)))
        {
                print("Nie można wykonać zapytania!<BR>\n");
                print("komunikat MySQL: " . mysql_error() . "<BR>\n");
                print("Zapytanie: $Query<BR>\n");
                exit();
        }

        //początek tabeli
        print("<TABLE BORDER=\"0\">\n");

        //utworzenie wiersza nagłówków
        print("<TR>\n");
        print("<TD BGCOLOR=\"#CCCCCC\"><B>Artykuł</B></TD>\n");
        print("<TD BGCOLOR=\"#CCCCCC\"><B>Cena</B></TD>\n");
        print("</TR>\n");

        // pobranie każdego z wierszy
        while($dbRow = mysql_fetch_object($dbResult))
        {
                print("<TR>\n");

                print("<TD>$dbRow->Nazwa</TD>\n");
                print("<TD ALIGN=\"right\">$dbRow->Cena</TD>\n");

                print("</TR>\n");
        }

        //koniec tabeli
        print("</TABLE>\n");
?>
```

Jeżeli połączenie się powiodło, zwracany jest identyfikator łącza z MySQL. Jak widać, w tym samym wierszu następuje nawiązanie połączenia i testowanie, czy operacja się powiodła.

Identyfikatory łączy są zawsze większe od zera, a zero zwracane jest wtedy, gdy nie można nawiązać połączenia. Dlatego też testowanie wystąpienia wartości `FALSE` umożliwia wykrycie nieudanego połączenia. W takim przypadku następuje po prostu opuszczenie skryptu.

Funkcja, za pomocą której łączymy się z bazą, to `mysql_pconnect`. Przeglądając opisy funkcji MySQL w rozdziale 13., „Funkcje współpracujące z bazami danych", można znaleźć również inną funkcję: `mysql_connect`. Funkcje te działają identycznie w ramach skryptu, ale `mysql_pconnect` nawiązuje trwałe połączenie z serwerem.

Większość funkcji współpracujących z bazami danych udostępnia możliwość tworzenia trwałych połączeń — czyli takich, które nie zamykają się z chwilą zakończenia skryptu. Jeżeli ten sam proces sieciowy uruchomi później inny skrypt, który łączy się z tym samym serwerem bazy danych, wykorzystane będzie istniejące już połączenie. Dzięki temu można oszczędzić trochę czasu. W praktyce oszczędności te nie są duże i wynikają ze sposobu, w jaki serwer Apache 1.3x używa procesów potomnych, zamiast wątków. Procesy te obsługują pewną liczbę wywołań, po czym są zastępowane przez nowe procesy. Po zakończeniu procesu zakończone zostaje również trwałe połączenie.

Rzeczywisty zysk czasu, wynikający ze stosowania trwałych połączeń, pojawia się dopiero przy dużych obciążeniach — w takich chwilach mogą one okazać się bardzo pomocne. Dlatego też domyślnie stosuję `mysql_pconnect`. W chwili pisania tej książki zbliża się moment pojawienia się Apache 2.0. Zapowiadane jest tam stosowanie wielowątkowego podejścia, co pozwoli zapewne w pełni wykorzystać zalety trwałych połączeń.

Następnym krokiem jest wybór bazy danych. Tutaj została wybrana baza `store`. Po poinformowaniu PHP, z której bazy należy korzystać, otrzymamy wszystkie wiersze z tabeli `katalog`. Realizuje to funkcja `mysql_query`. Wykonuje ona zapytanie na określonym łączu i zwraca identyfikator wyniku. Identyfikator ten zostanie użyty do pobrania wyników zapytania.

Zanim rozpoczniemy pobieranie danych ze zbioru wyników, należy rozpocząć budowanie tabeli HTML. Jak można się domyślić, w tym celu użyty został znacznik otwierający tabelę. Stworzony został wiersz nagłówków, z szarym tłem, a reszta właściwości tablicy pozostała domyślna.

Po wyświetleniu wiersza nagłówków można pobrać każdy z wierszy ze zbioru wyników. Najszybciej działającą funkcją jest `mysql_fetch_object`. Każda z kolumn wyniku staje się właściwością zwróconego obiektu. Nazwy kolumn stają się nazwami właściwości. Można również zastosować funkcje `mysql_fetch_row` lub `mysql_fetch_array`, które są równie efektywne. W większości przypadków stosowanie obiektów jest dla mnie bardziej czytelne. Należy unikać stosowania `mysql_result`, ponieważ funkcja ta wykonuje czasochłonną procedurę przeszukiwania tablicy dwuwymiarowej.

Gdy nie ma już więcej wierszy do pobrania, funkcja zwraca `FALSE`. Aby wykorzystać to zachowanie, pobieranie pojedynczego wiersza jest realizowane w pętli `while`. Wiersz w tabeli HTML jest tworzony poprzez wyświetlenie właściwości obiektu w odpowiednich jej polach. Po wyświetleniu wszystkich wierszy tabela jest zamykana. Nie trzeba w tym miejscu zamykać połączenia z bazą danych, ponieważ PHP robi to automatycznie w chwili zakończenia skryptu.

Opisany przykład jest niezwykle prosty, ale dotyka większości istotnych kwestii dotyczących współpracy z bazami danych. Ponieważ każdy z wierszy jest wyświetlany w pętli, wszystkie

wyglądają tak samo. Jeżeli dane ulegną zmianie, nie trzeba wprowadzać zmian w kodzie, który przekształca je w HTML. Można dowolnie zmieniać zawartość bazy danych.

Dobrym przykładem zastosowania tej techniki jest losowy generator nazw marek handlowych: *<http://www.leonatkinson.com/random/>*, który tworzy losowe nazwy z tabeli wyrazów przechowywanej w bazie MySQL, do której każdy może dodawać nowe słowa. Każde odświeżenie strony generuje kolejne dziesięć nazw.

Śledzenie odwiedzających za pomocą identyfikatorów sesji

Witryny sieciowe powoli przekształcają się w aplikacje sieciowe. Pojawia się tu problem zachowywania stanu. Podczas przechodzenia ze strony na stronę aplikacja musi pamiętać, kim jest użytkownik. Sieć jest anonimowa: przeglądarka łączy się z serwerem, pobiera parę plików i zamyka połączenie. Pięć minut później, gdy ponownie połączymy się ze stroną, proces się powtarza. Mimo że informacje o logowaniu są przechowywane, serwer nas nie pamięta. Wszelkie dane, jakie podaliśmy o sobie trzy strony wcześniej, nawet jeżeli zostały zapisane, nie będą z nami kojarzone.

Wyobraźmy sobie kreator zamówień pizzy: pierwszy ekran pyta się, ile pizz chcemy zamówić; następnie przechodzimy do ekranów odpowiadających każdej z zamawianych pizz i wybieramy jej zawartość i rodzaj ciasta; na koniec kolejna strona pyta o nasze nazwisko i telefon, aby przesłać nasze zamówienie do najbliższej pizzerii realizującej dowóz. Jednym ze sposobów rozwiązania tego problemu jest stopniowe przekazywanie dotychczas gromadzonych informacji z każdym następnym przesłaniem danych z formularza użytkownika. W miarę przechodzenia ze strony na stronę liczba danych będzie rosła. Przekazujemy serwerowi wielokrotnie częściowe dane o zamówieniu — metoda ta działa, ale wiąże się z przesyłaniem wielu niepotrzebnych informacji.

Stosując bazę danych i identyfikator sesji, można gromadzić informacje w miarę ich przekazywania. Identyfikator staje się kluczem do informacji. Z chwilą gdy skrypt pobiera identyfikator, przypomina sobie, co działo się wcześniej.

Inną kwestią jest sposób pobrania identyfikatora. Istnieją tu dwie możliwości. Pierwszą jest przekazywanie identyfikatora jako zmiennej w każdym łączu i w każdym formularzu. W przypadku formularza można to łatwo zrealizować za pomocą zmiennej ukrytej. W przypadku łącza należy wstawić znak zapytania i definicję zmiennej. Jeżeli założymy, że identyfikator sesji jest przechowywany w zmiennej `session`, jego przesłanie na następną stronę mogłoby mieć następującą formę:

```
print("A HREF=\"page2.php3?session=$session\">next</A>");
```

Metoda ta działa z wszystkim przeglądarkami, również z Lynx.

Drugim sposobem jest użycie *cookies*. Podobnie jak zmienne formularzy *GET* i *POST*, *cookies* są zamieniane przez PHP na zmienne. Można więc utworzyć *cookie* o nazwie `session`.

Różnica polegać będzie na tym, że *cookies* muszą być przesyłane w nagłówkach i należy wysłać je, zanim przeglądarka otrzyma jakąkolwiek treść HTML. Warto w tym miejscu zajrzeć do opisu funkcji `setcookie` w rozdziale 8., „Funkcje wejścia-wyjścia". Bardziej złożona strategia polega na próbie zastosowania *cookies* i w przypadku niepowodzenia oparciu się na zmiennych *GET*.

Obydwie metody są szeroko stosowane w Internecie. Wystarczy wejść w jakąkolwiek witrynę sklepu internetowego. Na potrzeby przykładu opisana tu zostanie strategia polegająca na zastosowaniu zmiennych *GET*. Pierwszym krokiem jest stworzenie tabeli przechowującej identyfikatory sesji. Listing 17.4 przedstawia kod SQL, tworzący prostą tabelę `sesja` w bazie danych MySQL.

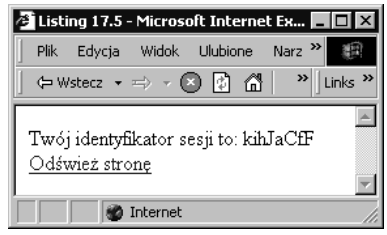
Listing 17.4. *Tworzenie tabeli przechowującej sesje*

```
CREATE TABLE sesja
(
        ID VARCHAR(8) NOT NULL,
        OstatniaOperacja DATETIME,
        PRIMARY KEY (ID)
);
```

Kluczami w tabeli są 8-znakowe łańcuchy. Po każdym przejściu użytkownika na inną stronę odświeżana jest zawartość kolumny `OstatniaOperacja`. W ten sposób możemy pozbyć się wszelkich sesji, które wyglądają na nieużywane. Każda wizyta na stronie spowoduje usuwanie sesji bezczynnych przez 30 minut. Następnym krokiem jest sprawdzanie, czy użytkownik posiada identyfikator sesji. Jeżeli nie posiada, zostaje on utworzony. Jeżeli posiada, należy sprawdzić, czy jest on prawidłowy.

Po pierwszym załadowaniu skryptu z listingu 17.5 tworzy on identyfikator sesji. Każde kliknięcie przycisku *Odśwież* w przeglądarce spowoduje sprawdzenie sesji przez skrypt. Jeżeli identyfikator sesji nie znajduje się w tabeli `sesja`, wówczas zostanie odrzucony i utworzony będzie nowy. Można spróbować przesłać nieprawidłowy identyfikator, usuwając jakiś znak w pasku adresu przeglądarki.

Rysunek 17.2. *Sprawdzanie identyfikatora sesji*



Listing 17.5. *Sprawdzanie identyfikatora sesji*

```
<?
        /*
        ** Demonstracja zastosowania identyfikatorów sesji
        */

        function SessionID($length=8)
```

```
{
        // Ustanowienie zakresu dopuszczalnych znaków
        $Pool = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        $Pool .= "abcdefghijklmnopqrstuvwxyz";
        $lastChar = strlen($Pool) - 1;

        for($i = 0; $i < $length; $i++)
        {
                $sid .= $Pool[mt_rand(0, $lastChar)];
        }

        return($sid);
}

//Inicjalizacja generatora liczb losowych
mt_srand(time());

//połączenie z serwerem i kontrola powodzenia operacji
if(!($dbLink = mysql_pconnect("localhost", "freetrade", "")))
{
        print("Nie można połączyć się z bazą danych!<BR>\n");
        print("Zakończenie skryptu!<BR>\n");
        exit();
}

//wybór bazy danych i kontrola powodzenia operacji
if(!mysql_select_db("test", $dbLink))
{
        print("Nie można wybrać bazy test!<BR>\n");
        print("Zakończenie skryptu!<BR>\n");
        exit();
}

//usunięcie wszystkich starych sesji
$Query = "DELETE FROM sesja ";
$Query .= "WHERE OstatniaOperacja < '";
$Query .= date("Y-m-d H:i:s", (time()-10800));
$Query .= "'";
if(!($dbResult = mysql_query($Query, $dbLink)))
{
        //nie można wykonać zapytania
        print("Nie można usunąć starych sesji!<BR>\n");
        print("Komunikat MySQL: " . mysql_error() . "<BR>\n");
        exit();
}

//sprawdzenie sesji
if(isset($session))
{
        //jeżeli mamy sesję, następuje jej sprawdzenie
        $Query = "SELECT * ";
        $Query .= "FROM sesja ";
        $Query .= "WHERE ID='" . addslashes($session) . "' ";

        if(!($dbResult = mysql_query($Query, $dbLink)))
```

```
                {
                        //nie można wykonać zapytania
                        print("nie można wykonać zapytania do tabeli sesja!<BR>\n");
                        print("Komunikat MySQL: " . mysql_error() . "<BR>\n");
                        exit();
                }

                //jeżeli zwrócony został jakiś wiersz, udało się odnaleźć sesję
                if(mysql_numrows($dbResult))
                {
                        //sesja istnieje, odświeżenie czasu ostatniej operacji
                        $Query = "UPDATE sesja ";
                        $Query .= "SET OstatniaOperacja = now() ";
                        $Query .= "WHERE ID='$session' ";
                        if(!($dbResult = mysql_query($Query, $dbLink)))
                        {
                                //nie można wykonać zapytania
                                print("Nie można odświeżyć tabeli sesja!<BR>\n");
                                print("komunikat MySQL: " . mysql_error() . "<BR>\n");
                                exit();
                        }
                }
                else
                {
                        //zły identyfikator
                        print("Nieznany identyfikator sesji ($session)!<BR>\n");
                        $session = "";
                }
        }

        //jeżeli nie było sesji, należy ją utworzyć
        if($session == "")
        {
                //nie było sesji - tworzona jest nowa
                $session = SessionID(8);

                //umieszczenie sesji w bazie danych
                $Query = "INSERT INTO sesja ";
                $Query .= "VALUES ('$session', now()) ";
                if(!($dbResult = mysql_query($Query, $dbLink)))
                {
                        //nie można wykonać zapytania
                        print("Nie można wstawić elementu do tabeli sesja!<BR>\n");
                        print("Komunikat MySQL: " . mysql_error() . "<BR>\n");
                        exit();
                }
        }

        print("Twój identyfikator sesji to: $session<BR>\n");
        print("<A HREF=\"$PHP_SELF?session=$session\">");
        print("Odśwież stronę");
        print("</A><BR>\n");
?>
```

Kolejnym logicznym krokiem jest dodanie kolejnej tabeli, przechowującej informacje o osobie przeglądającej witrynę. Jedna z kolumn powinna przechowywać identyfikator sesji z tabeli `sesja`. Dopisanie tego kodu może być dobrym ćwiczeniem.

Przechowywanie danych w bazie

Informacja przechowywana w bazie danych nie ogranicza się do krótkich łańcuchów, jak 32-znakowa nazwa towaru z listingu 17.3. Można też tworzyć duże, 64-kilobajtowe obiekty, które pomieszczą nawet kompletną stronę internetową. Zaletą jest fakt, że strony zawsze istnieją w bardzo uporządkowanym otoczeniu. Można je określić numerem, który wystarcza do ustalenia zależności między stronami. Minusem jest niemożność załadowania strony istniejącej w bazie danych do swojego ulubionego edytora. Należy tu rozważyć zyski i straty — dla większości stron nie ma potrzeby przechowywania każdego ich elementu w bazie danych.

Przykładem sytuacji, kiedy umieszczenie zawartości strony w bazie danych jest zasadne, jest forum internetowe. System taki przechowuje wiadomości, które, oprócz tego, że są stronami, posiadają charakterystyki, takie jak: tytuł, data utworzenia i autor. Taka struktura może być w prosty sposób odwzorowana w tabeli bazy danych. Co więcej, ponieważ każdej wiadomości można nadać jednoznaczny identyfikator, można organizować wiadomości w formie drzewa zależności. Użytkownik zostawia wiadomość, która rozpocznie wątek w dyskusji i spowoduje pojawienie się nowych wiadomości do niego nawiązujących. Wiadomości można wyświetlać w hierarchii wątków, co ułatwia ich przeglądanie.

Jak w każdym systemie opartym na bazie danych, pierwszy krok polega na stworzeniu tabeli. W listingu 17.6 utworzona zostaje tabela do przechowywania wiadomości. Każda wiadomość ma tytuł, nazwę nadawcy, czas wysłania, wiadomość nadrzędną i treść. Identyfikator wiadomości nadrzędnej może wynosić zero, co oznacza, że wiadomość inicjuje nowy wątek. Treścią wiadomości nie musi być zwykły tekst, może nią być również kod HTML. Tym sposobem użytkownicy mają możliwość tworzenia w bazie stron za pomocą własnych przeglądarek.

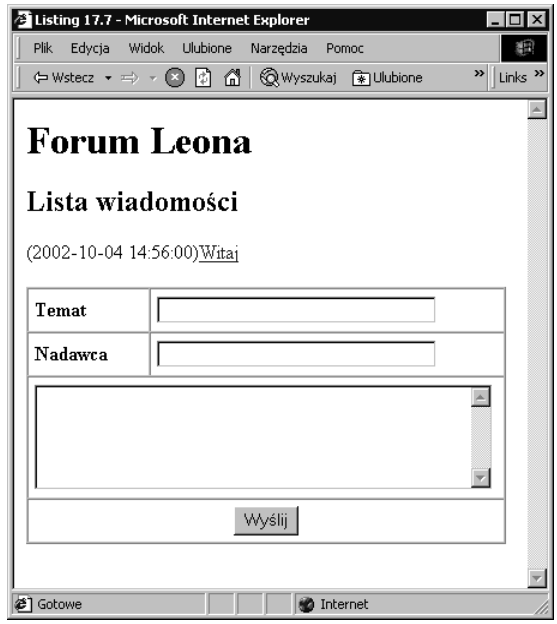
Listing 17.6. *Tworzenie tabeli przechowującej wiadomości*

```
CREATE TABLE Wiadomosc
(
        ID INT NOT NULL AUTO_INCREMENT,
        Temat VARCHAR(64),
        Nadawca VARCHAR(64),
        Stworzono DATETIME,
        Przodek INT,
        Tresc BLOB,
        PRIMARY KEY(ID)
);
```

Skrypt z listingu 17.7 ma dwa tryby pracy: listowanie tytułów wiadomości i przeglądanie pojedynczej wiadomości. Jeżeli zmienna `messageID` jest pusta, wówczas ukazana zostaje lista wszystkich wiadomości w systemie, zorganizowanych według wątków. Realizuje to funkcja `showMessages`. W tym miejscu warto powrócić do fragmentu rozdziału 4., „Funkcje”, gdzie opisana jest rekurencja. Funkcja `showMessage` korzysta z rekurencji przy przeglądaniu każdej

z gałęzi drzewa wiadomości. Na początku pobiera listę wszystkich wiadomości, które nie mają wiadomości nadrzędnych. Są to początki wątków. Po ukazaniu wszystkich wiadomości inicjujących wątki `showMessage` jest wywoływana dla danego wątku. Proces powtarza się do momentu odnalezienia wiadomości, która nie posiada wiadomości potomnych. Do wyświetlenia tytułów wiadomości wykorzystano znaczniki `UL`. Wcięcia pomagają odwzorować na ekranie hierarchię wiadomości.

Rysunek 17.3. *Lista wiadomości forum*



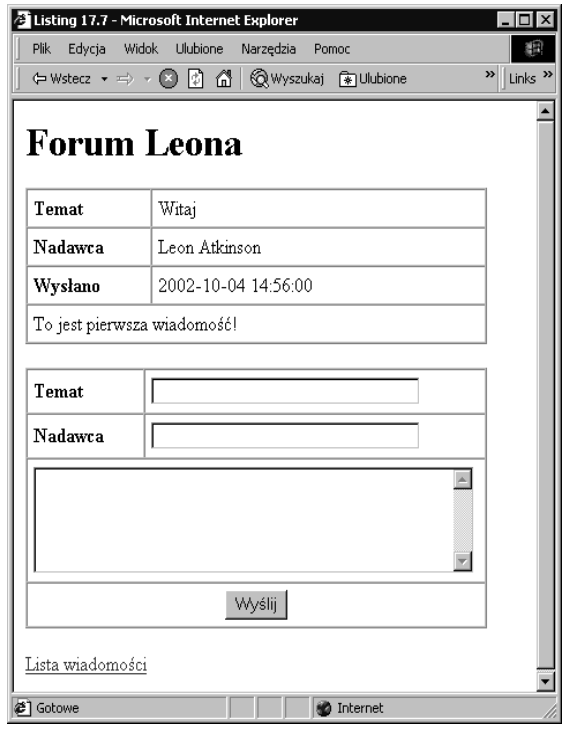
Listing 17.7. *Proste forum*

```
<?
        print("<H1>Forum Leona</H1>\n");

        //połączenie z serwerem i sprawdzenie, czy się powiodło
        if(!($dbLink = mysql_pconnect("localhost", "httpd", "")))
        {
                print("Nie można połączyć się z bazą danych!<BR>\n");
                print("Zakończenie skryptu!<BR>\n");
                exit();
        }

        //wybór bazy danych i sprawdzanie powodzenia operacji
        if(!mysql_select_db("test", $dbLink))
        {
                print("Nie można wybrać bazy test!<BR>\n");
                print("Aborting!<BR>\n");
                exit();
        }

        /*
        ** rekurencyjna funkcja wyświetlająca
        ** wszystkie wiadomości z danego zakresu
        */
```



Rysunek 17.4. *Wiadomość z forum*

```
function showMessages($parentID)
{
        global $dbLink;

        $dateToUse = Date("U");

        print("<UL>\n");

        $Query = "SELECT ID, Temat, Stworzono ";
        $Query .= "FROM Wiadomosc ";
        $Query .= "WHERE Parent=$parentID ";
        $Query .= "ORDER BY Stworzono ";

        if(!($dbResult = mysql_query($Query, $dbLink)))
        {
                //nie można wykonać zapytania
                print("Nie można wykonać zapytania do tabeli Wiadomość!<BR>\n");
                print("Komunikat MySQL: " . mysql_error() . "<BR>\n");
                exit();
        }

        while($row = mysql_fetch_object($dbResult))
        {
                //ukazanie tematu wiadomości w formie odnośnika do jej treści
                print("<LI>($row->Stworzono) <A HREF=\"");
                print("$PHP_SELF?messageID=$row->ID\">");
                print("$row->Temat</A><BR>\n");
```

```
                //ukazanie wiadomości potomnych
                showMessages($row->ID);
        }

        print("</UL>\n");
}


/*
** wyświetlenie formularza do dodawania wiadomości
** z danym identyfikatorem przodka
*/
function postForm($parentID, $useTitle)
{
        print("<FORM ACTION=\"$PHP_SELF\" METHOD=\"post\">\n");

        print("<INPUT TYPE=\"hidden\" NAME=\"inputParent\" ");
        print("VALUE=\"$parentID\">\n");

        print("<INPUT TYPE=\"hidden\" NAME=\"ACTION\" ");
        print("VALUE=\"POST\">\n");

        print("<TABLE BORDER=\"1\" CELLSPACING=\"0\" ");
        print("CELLPADDING=\"5\" WIDTH=\"400\">\n");

        print("<TR>\n");

        print("<TD WIDTH=\"100\"><B>Temat</B></TD>\n");

        print("<TD WIDTH=\"300\">");
        print("<INPUT TYPE=\"text\" NAME=\"inputTitle\" ");
        print("SIZE=\"35\" MAXLENGTH=\"64\" VALUE=\"$useTitle\">");
        print("</TD>\n");

        print("</TR>\n");

        print("<TR>\n");

        print("<TD WIDTH=\"100\"><B>Nadawca</B></TD>\n");

        print("<TD WIDTH=\"300\">");
        print("<INPUT TYPE=\"text\" NAME=\"inputPoster\" ");
        print("SIZE=\"35\" MAXLENGTH=\"64\">");
        print("</TD>\n");

        print("</TR>\n");

        print("<TR>\n");

        print("<TD COLSPAN=\"2\" WIDTH=\"400\">");
        print("<TEXTAREA NAME=\"inputBody\" ");
        print("COLS=\"45\" ROWS=\"5\"></TEXTAREA>");
        print("</TD>\n");

        print("</TR>\n");

        print("<TR>\n");
```

```
        print("<TD COLSPAN=\"2\" WIDTH=\"400\" ALIGN=\"middle\">");
        print("<INPUT TYPE=\"submit\" VALUE=\"Wyślij\">");
        print("</TD>\n");

        print("</TR>\n");

        print("</TABLE>\n");
        print("</FORM>\n");
}

/*
** wykonuje operacje
*/
if($ACTION != "")
{
        if($ACTION == "POST")
        {
                $Query = "INSERT INTO Wiadomosc ";
                $Query .= "VALUES(0,";
                $Query .= "'" . addslashes($inputTitle) . "', ";
                $Query .= "'" . addslashes($inputPoster) . "', ";
                $Query .= "now(), $inputParent, ";
                $Query .= "'" . addslashes($inputBody) . "')";

                if(!($dbResult = mysql_query($Query, $dbLink)))
                {
                        //nie można wykonać zapytania
                        print("Nie można wstawić elementu do tabeli
                        ↳Wiadomość!<BR>\n");
                        print("Komunikat MySQL: " . mysql_error() . "<BR>\n");
                        exit();
                }
        }

}


/*
** Wyświetlenie wiadomości lub listy wiadomości
*/
if($messageID > 0)
{
        $Query = "SELECT ID, Temat, Nadawca, Stworzono, Przodek, Tresc ";
        $Query .= "FROM Wiadomosc ";
        $Query .= "WHERE ID=$messageID ";

        if(!($dbResult = mysql_query($Query, $dbLink)))
        {
                //nie można wykonać zapytania
                print("Nie można wykonać zapytania do tabeli Wiadomość!<BR>\n");
                print("Komunikat MySQL: " . mysql_error() . "<BR>\n");
                exit();
        }

        if($row = mysql_fetch_object($dbResult))
        {
                print("<TABLE BORDER=\"1\" CELLSPACING=\"0\" ");
                print("CELLPADDING=\"5\" WIDTH=\"400\">\n");
```

```
                print("<TR>");
                print("<TD WIDTH=\"100\"><B>Temat</B></TD>");
                print("<TD WIDTH=\"300\">$row->Title</TD>");
                print("</TR>\n");

                print("<TR>");
                print("<TD WIDTH=\"100\"><B>Nadawca</B></TD>");
                print("<TD WIDTH=\"300\">$row->Poster</TD>");
                print("</TR>\n");

                print("<TR>");
                print("<TD WIDTH=\"100\"><B>Wyslano</B></TD>");
                print("<TD WIDTH=\"300\">$row->Stworzono</TD>");
                print("</TR>\n");

                print("<TR>");
                print("<TD COLSPAN=\"2\" WIDTH=\"400\">");
                print("$row->Tresc");
                print("</TD>");
                print("</TR>\n");

                print("</TABLE>\n");

                postForm($row->ID, "RE: $row->Temat");

        }


        print("<A HREF=\"$PHP_SELF\">Lista wiadomości</A><BR>\n");

    }
    else
    {
        print("<H2>Lista wiadomości</H2>\n");

        // pobranie całej listy
        showMessages(0);

        postForm(0, "");

    }
?>
```

Z punktu widzenia efektywności, zastosowanie rekurencji nie jest optymalne. Każdy wątek powoduje kolejne wywołanie `showMessage`, z czym wiąże się następne zapytanie do bazy danych. Istnieje sposób na jednokrotne przesłanie zapytania i przeglądanie drzewa przechowywanego w pamięci — realizację tego pomysłu pozostawiam jako ćwiczenie dla Czytelnika.

Gdy użytkownik kliknie tytuł wiadomości, strona zostaje ponownie załadowana z ustawionym `messageID`. Powoduje to, że skrypt przełącza się na tryb wyświetlania pojedynczych wiadomości. Pola wiadomości zostają wyświetlone w tabeli. Jeżeli wiadomość zawiera jakikolwiek kod HTML, zostanie on zinterpretowany przez przeglądarkę, ponieważ nie następuje tu żadna filtracja znaczników. Ograniczenie to znajduje zastosowanie w kodzie dodającym nowe wiadomości.

W każdym z dwóch trybów pracy ukazywany jest formularz do dodawania wiadomości. Jeżeli wiadomość została dodana w chwili wyświetlania listy, zostanie uznana za początek nowego wątku. Jeżeli zaś wiadomość dodana została podczas przeglądania innej wiadomości, uznana zostanie za odpowiedź na tę wiadomość i nowa wiadomość będzie potomkiem przeglądanej wiadomości.

Przedstawione tu forum jest proste, ale zawiera podstawowe elementy funkcjonalne takiego systemu. Bardziej wyrafinowanym rozwiązaniem byłoby umożliwienie dodawania wiadomości tylko przez uwierzytelnionych użytkowników lub nieudostępnianie wiadomości na forum do momentu ich akceptacji przez moderatora. Struktury tej można użyć do budowy dowolnej aplikacji, która zarządza danymi przesłanymi przez użytkownika, chociażby księgi gości. Jeżeli szukamy bardziej wyrafinowanej implementacji forum internetowego, można zajrzeć do witryny projektu Phorum Briana Moona pod adresem <*http://www.phorum.org*>.

Warstwy abstrakcyjne baz danych

Wyobraźmy sobie, że podczas tworzenia aplikacji sieciowej, wykorzystującej MySQL, zostaliśmy poproszeni o zmianę bazy danych na Oracle. Każda z funkcji PHP jest inna dla tych dwóch baz i musielibyśmy zmienić każdą z nich. Dodatkowo Oracle i MySQL używają nieco odmiennego SQL, co wymagałoby zapewne zmiany większości zapytań. Rozwiązaniem tego problemu jest zastosowanie warstwy abstrakcyjnej. Pozwala to na oddzielenie sposobu działania aplikacji od kodu komunikującego się z bazą danych. Pojedyncza funkcja wywołuje tu odpowiednią funkcję w zależności od typu bazy, z jaką się komunikujemy.

Jedną z najbardziej popularnych warstw abstrakcyjnych baz danych stanowi fragment biblioteki PHP Base Library <*http://phplib.netuse.de*>. Biblioteka ta zawiera również kod do zarządzania sesjami. Inną warstwą abstrakcyjną jest Metabase, dostępna pod adresem: <*http://phpclasses.upperdesign.com*>.

Niezależnie od warstw abstrakcyjnych, niezgodności między bazami powodują stałe powstawanie nowych rozwiązań. MySQL używa specjalnego kwalifikatora do opisu kolumn, zwanego `AUTO_INCREMENT`. Powoduje on automatyczne wypełnianie kolumn liczbami całkowitymi w kolejności rosnącej. W Oracle można to w przybliżeniu z realizować za pomocą sekwencji (*SEQUENCE*) i wyzwalacza (*TRIGGER*). Wszystkie te różnice trudno systematycznie godzić. W 1999 r. Scott Ambler zaproponował rozwiązanie w swojej publikacji *The Design of a Robust Persistence Layer for Relational Databases*, <*http://www.ambysoft.com/persistenceLayer.html*>. Zawarł tam dogłębną analizę problemu, wraz ze szczegółowymi rozwiązaniami, jednak nie oddaję im sprawiedliwości w kontekście tego rozdziału.

Warstwy abstrakcyjne udostępniają pewność kosztem pewnej utraty efektywności. Część rozwiązań poprawiających efektywność w danej bazie danych musi zostać pominięta. Warstwa abstrakcyjna udostępnia zwykle podstawowy zestaw funkcji bazy danych. Zysk polega tu na uniezależnieniu się od jednego typu bazy.