PHP i MySQL

OD NOWICJUSZA DO WOJOWNIKA NINJA

KEVIN YANK

sitepoint

Helion

NAJLEPSZY PRZEWODNIK DLA ODKRYWCÓW PHP!

Tytuł oryginału: PHP & MySQL: Novice to Ninja

Tłumaczenie: Paweł Koronkiewicz (wstęp, rozdz. 1 - 9),
Tomasz Walczak (rozdz. 10 - 12, dodatki)

ISBN: 978-83-246-7110-6

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

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

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/phmnow
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Rozdział 7

System zarządzania treścią (CMS)

Tym, co odróżnia stronę WWW przeznaczoną do wyświetlania informacji z bazy danych od witryny, której funkcjonowanie w całości opiera się na informacjach z serwera bazy (ang. *database-driven website*), jest **system zarządzania treścią** (ang. *content management system*, *CMS*). System taki przyjmuje postać zbioru stron dostępnych jedynie użytkownikom uprawnionym do wprowadzania zmian. Jest on interfejsem administracyjnym bazy danych, który pozwala przeglądać i zmieniać przechowywane informacje bez konieczności operowania instrukcjami SQL.

Początki systemu CMS zbudowaliśmy w końcowej części rozdziału 4., kiedy dodaliśmy formularze pozwalające dodawać i usuwać dowcipy oraz przycisk *Usuń*. Choć są to mechanizmy niewątpliwie ciekawe, nie powinny one raczej znaleźć się na stronie wyświetlanej wszystkim gościom witryny. Wskazane byłoby pewne zabezpieczenie przed obraźliwymi treściami, a już na pewno przed swobodnym usuwaniem zawartości bazy.

Przesunięcie takich mechanizmów do wydzielonej grupy stron administracyjnych, do których dostęp jest wyraźnie ograniczony, pozwala zmniejszyć ryzyko ujawnienia danych poufnych. Jest to zarazem znaczne ułatwienie przy zarządzaniu treścią — nie trzeba uciekać się do samodzielnego pisania kwerend SQL. W tym rozdziale rozszerzymy możliwości systemu zarządzającego bazą dowcipów o obsługę elementów wprowadzonych do bazy w rozdziale 5. Mówiąc prościej, dodamy mechanizm pozwalający administratorowi witryny zarządzać autorami i kategoriami oraz przypisywać je do dowcipów.

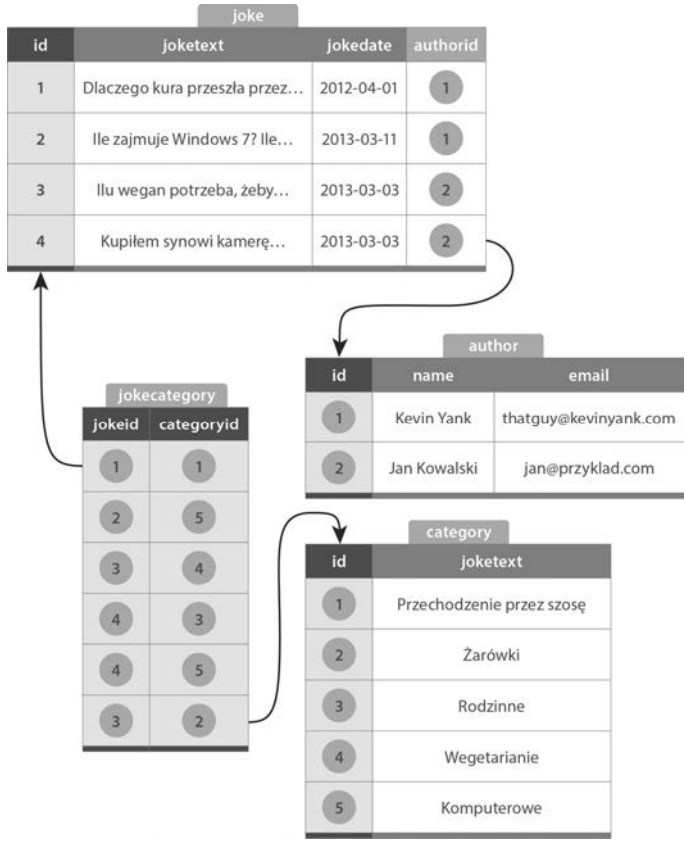
Strony administracyjne zawsze chroni pewnego rodzaju system kontroli dostępu. Jedną z możliwości implementacji takiego systemu jest skonfigurowanie serwera WWW w taki sposób, by chronił odpowiednie pliki PHP, wymagając podania nazwy użytkownika i hasła. Na serwerach Apache służą do tego pliki *.htaccess* przechowujące listy autoryzowanych użytkowników.

Inną metodą ochrony stron administracyjnych jest wykorzystanie możliwości samego PHP. Jest to rozwiązanie bardziej elastyczne i prowadzące do bardziej eleganckiego rezultatu. Wymaga ono jednak nieco więcej pracy. Opiszę je w rozdziale 9.

Na razie skoncentrujmy się na przygotowaniu stron systemu zarządzania treścią.

Strona główna

Pod koniec rozdziału 5. baza danych zawierała tabele dla obiektów trzech typów: dowcipów, autorów i kategorii dowcipów. Jej konstrukcję ilustruje rysunek 7.1. Zwróć uwagę, że pozostajemy przy początkowym założeniu, że każdy autor ma tylko jeden adres e-mail.



Rysunek 7.1. Struktura bazy ijdb przewiduje trzy rodzaje obiektów

Jeżeli musisz odtworzyć strukturę i podstawowe dane tabel od podstaw, możesz skorzystać z poniższej sekwencji kwerend SQL:

kod w pliku chapter7/sql/ijdb.sql

```
CREATE TABLE joke (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    joketext TEXT,
    jokedate DATE NOT NULL,
    authorid INT
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

CREATE TABLE author (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    email VARCHAR(255)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

CREATE TABLE category (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

CREATE TABLE jokecategory (
    jokeid INT NOT NULL,
    categoryid INT NOT NULL,
    PRIMARY KEY (jokeid, categoryid)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

# Przykładowe dane
# Określamy 'id', aby było zgodne w odwołaniach z innych tabel

INSERT INTO author (id, name, email) VALUES
(1, 'Kevin Yank', 'thatguy@kevinyank.com'),
(2, 'Jan Kowalski', 'jan@przyklad.com');

INSERT INTO joke (id, joketext, jokedate, authorid) VALUES
(1, 'Dlaczego kura przeszła przez szosę? Żeby dostać się na drugą stronę.',
  '2012-04-01', 1),
(2, 'Ile zajmuje Windows 7? Ile znajdzie, tyle zajmie...', '2012-04-01', 1),

 (3, 'Kupiłem synowi kamerę internetową. Jedną stronę pokoju ma teraz
  posprzątaną...', '2012-04-01', 2),
(4, 'Ilu wegan potrzeba, żeby zmienić żarówkę? Dwóch. Jeden wkręca, drugi
  czyta skład.', '2012-04-01', 2);

INSERT INTO category (id, name) VALUES
(1, 'Przechodzenie przez szosę'),
(2, 'Żarówki'),
(3, 'Rodzinne'),
(4, 'Wegetarianie'),
(5, 'Komputerowe');

INSERT INTO jokecategory (jokeid, categoryid) VALUES
(1, 1), (2, 5), (3, 4), (4, 3), (4, 5), (3, 2);
```

Strona główna systemu zarządzania treścią będzie zawierać łącza do stron, które pozwalają zarządzać każdym z trzech rodzajów obiektów. Poniższy HTML generuje stronę widoczną na rysunku 7.2.

Rysunek 7.2. Strona główna systemu CMS zawiera trzy łącza

kod w pliku chapter7/admin/index.html

```
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <title>System CMS Bazy Dowcipów</title>
  </head>
  <body>
    <h1>System Zarządzania Dowcipami</h1>
    <ul>
      <li><a href="jokes/">Zarządzanie dowcipami</a></li>
      <li><a href="authors/">Zarządzanie autorami</a></li>
      <li><a href="categories/">Zarządzanie kategoriami dowcipów</a></li>
    </ul>
  </body>
</html>
```

Każde z tych łączy kieruje do innego podkatalogu: *jokes*, *authors* lub *categories*. Każdy z tych podkatalogów będzie zawierał skrypt kontrolera oraz wymagane przezeń szablony — strony służące do zarządzania obiektami bazy.

Zarządzanie autorami

Rozpocznijmy od kodu, który zapewni możliwości dodawania i usuwania wpisów autorów oraz, dodatkowo, modyfikowania ich. Całość tego kodu będzie przechowywana w podkatalogu *authors*.

Podstawową informacją, którą powinniśmy wyświetlić administratorowi, aby mógł zarządzać autorami, powinna być lista zapisanych w bazie autorów. Potrzebny do tego kod nie różni się istotnie od używanego do wyświetlania listy zapisanych dowcipów. Ponieważ zamierzamy zapewnić możliwość usuwania i modyfikowania wpisów w tabeli autorów, dodamy odpowiednie przyciski. Podobnie jak dodany w rozdziale 4. przycisk *Usuń*, będą wysyłać identyfikator autora, co zapewni kontrolerowi informację o tym, na którym wierszu tabeli wykonać operację. Przycisk dodawania nowego wpisu będzie nosił nazwę *Dodaj nowego autora*. Wyświetlany przezeń formularz będzie podobny do wyświetlanego przy dodawaniu dowcipów w rozdziale 4.

Oto kod kontrolera:

kod w pliku chapter7/admin/authors/index.php (fragment)

```
// Wyświetlanie listy autorów
include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

try
{
  $result = $pdo->query('SELECT id, name FROM author');
}
catch (PDOException $e)
{
  $error = 'Błąd bazy danych w trakcie pobierania listy autorów!';
  include 'error.html.php';
  exit();
}

foreach ($result as $row)
{
  $authors[] = array('id' => $row['id'], 'name' => $row['name']);
}

include 'authors.html.php';
```

Działanie tego kodu powinno być już dla Ciebie oczywiste. Zwróć uwagę, że do nawiązywania połączenia z bazą danych wykorzystywany jest współużytkowany plik *db.inc.php* przechowywany w katalogu *includes* serwera WWW.

Kontroler wykorzystuje do wyświetlania listy autorów następujący szablon:

kod w pliku chapter7/admin/authors/authors.html.php

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?> ❶
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <title>Zarządzanie autorami</title>
  </head>
  <body>
    <h1>Zarządzanie autorami</h1>
    <p><a href="?add">Dodaj nowego autora</a></p> ❷
    <ul>
      <?php foreach ($authors as $author): ?>
        <li>
          <form action="" method="post"> ❸
            <div>
              <?php htmlout($author['name']); ?> ❹
              <input type="hidden" name="id" value="<?php
                  echo $author['id']; ?>">
              <input type="submit" name="action" value="Edycja"> ❺
              <input type="submit" name="action" value="Usuń">
            </div>
```

```
        </form>
      </li>
    <?php endforeach; ?>
  </ul>
  <p><a href="..">Powrót do strony głównej CMS</a></p>
 </body>
</html>
```

Również ten kod powinien wyglądać znajomo. Zwróćmy uwagę na kilka ważniejszych miejsc:

❶ Szablon korzysta ze współużytkowanego pliku, który przygotowaliśmy wcześniej w rozdziale 6. Ułatwia to wypisywanie na stronie wartości wymagających wywołania `htmlspe↪cialchars`.

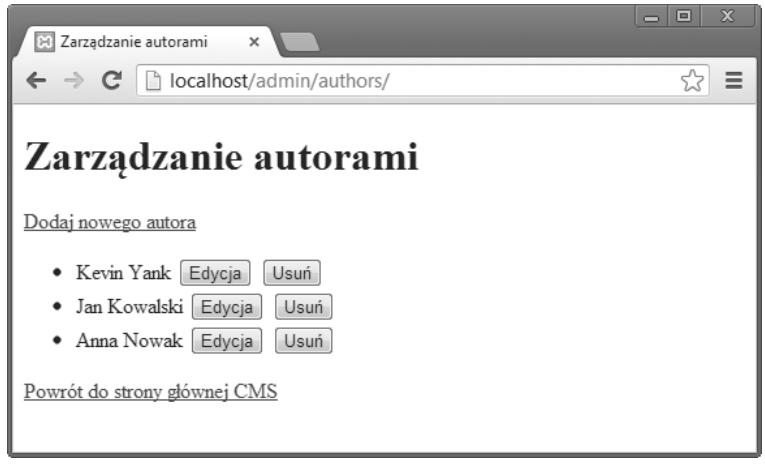
❷ Łącze wysyłające do kontrolera ciąg kwerendy URL `?add`. Służy on jako informacja o tym, że użytkownik zamierza dodać nowego autora.

❸ Pusty atrybut `action`. Po wysłaniu formularz staje się przekazywanym kontrolerowi żądaniem zmiany lub usunięcia danych autora. W rozdziale 4. wykorzystywaliśmy ciąg `?deletejoke` w atrybucie `action`, wskazując w ten sposób zamiar usunięcia wpisu. Ponieważ użytkownik ma teraz możliwość wykonywania dwóch operacji, użyjemy innej metody, aby poznać dokonany wybór.

❹ Używamy funkcji `htmlout` z pliku *helpers.inc.php*, aby bezpiecznie wypisać imię i nazwisko autora.

❺ Ten formularz ma dwa przyciski wysyłania: jeden do edycji i jeden do usuwania autora. Opisujemy je jednakowym atrybutem `name` (o brzmieniu `action`). Umożliwi to kontrolerowi ustalenie, który przycisk został kliknięty, poprzez sprawdzenie powiązanej z tą nazwą wartości (`$_POST['action']`).

Rysunek 7.3 przedstawia generowaną z użyciem tego szablonu listę autorów.



Rysunek 7.3. Zarządzanie informacjami o autorach rozpoczyna się od wyświetlenia głównej strony interfejsu Zarządzanie autorami

Usuwanie autorów

Gdy użytkownik kliknie jeden z przycisków *Delete*, nasz kontroler powinien usuwać odpowiadającego mu autora z bazy danych. Do wybierania autora do usunięcia służy przesyłana wraz z formularzem wartość `id`.

Jak widzieliśmy wcześniej, sama operacja usuwania jest niezwykle prosta. Jednak w tym przypadku należy liczyć się z pewnym utrudnieniem — tabela `joke` ma kolumnę `authorid`, która wskazuje autora odpowiedzialnego za wpisanie dowcipu do bazy. Po usunięciu wpisu autora musimy usunąć odwołania do niego z innych tabel. Jeżeli tego zaniedbamy, baza zawierać będzie dowcipy przypisane nieistniejącym już autorom.

Mamy do wyboru trzy możliwości:

- Uniemożliwić użytkownikom usuwanie autorów powiązanych z przechowywanymi w bazie dowcipami.
- Usuwać dowcipy autora jednocześnie z usuwaniem jego wpisu w bazie.
- Zmieniać wartość kolumny `authorid` dowcipów usuwanego autora na `NULL`, sygnalizując w ten sposób brak autora wpisu.

Podejmując tego rodzaju działania ukierunkowane na zachowanie poprawnego układu relacji w bazie, chronimy tak zwaną **spójność odwołań** lub **integralność odwołań** bazy (ang. *referential integrity*). MySQL, podobnie jak większość serwerów baz danych, posiada mechanizm **ograniczenia typu „klucz obcy”** (ang. *foreign key constraint*), który pozwala utrzymywać spójność odwołań automatycznie. Konfigurując takie ograniczenie, można nakazać serwerowi podejmowanie jednego z trzech wymienionych działań.

Automatycznymi ograniczeniami tego rodzaju zajmiemy się w rozdziale 10. Nie będziemy ich tu stosować. Gdybyśmy tak zrobili, oznaczałoby to definiowanie części mechanizmów CMS w kodzie PHP, a części — w architekturze bazy danych. Podejście takie prowadziłoby do sytuacji, w której przy jakiejkolwiek zmianie zasad usuwania autorów (na przykład wprowadzaniu reguły, że nie można usunąć autora, dopóki baza zawiera jego dowcipy) konieczne byłoby pamiętanie o wprowadzaniu korekt w dwóch miejscach. Zachowamy więc całą logikę operacji usuwania w kodzie PHP — ułatwi to pracę z kodem każdemu, kto będzie miał z nim do czynienia w przyszłości (łącznie z Tobą samym!).

Ponieważ większość autorów życzy sobie uznania ich wkładu przy każdym wyświetlanym dowcipie, decydujemy się na wybór opcji drugiej: usuwania wszystkich powiązanych dowcipów wraz z autorem. Zaoszczędzi to między innymi problemu obsługi dowcipów z wartością `NULL` w kolumnie `authorid` podczas wyświetlania zawartości bazy.

Ponieważ posuwamy się do usuwania dowcipów, pojawia się kolejna komplikacja. Dowcipy są przypisywane do kategorii, a informacje o przypisaniach przechowuje tabela `jokecategory`. Usunięciu dowcipu musi więc towarzyszyć usunięcie powiązanych z nim wpisów w tabeli łączącej.

Ogólnie rzecz biorąc, prosta z pozoru operacja usuwania autora rozrasta się do trzech niezbędnych czynności: usunięcia autora, usunięcia jego dowcipów i usunięcia przypisań tych dowcipów do kategorii.

Jak można oczekiwać, wymagany kod jest dość długi. Spróbuj prześledzić jego elementy i samodzielnie odtworzyć w wyobraźni jego działanie:

kod w pliku chapter7/admin/authors/index.php (fragment)

```
if (isset($_POST['action']) and $_POST['action'] == 'Usuń')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  // Pobierz dowcipy autora
  try
  {
    $sql = 'SELECT id FROM joke WHERE authorid = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy pobieraniu listy dowcipów do usunięcia.';
    include 'error.html.php';
    exit();
  }

  $result = $s->fetchAll();

  // Usuń przypisania dowcipów do kategorii
  try
  {
    $sql = 'DELETE FROM jokecategory WHERE jokeid = :id';
    $s = $pdo->prepare($sql);

    // Dla każdego dowcipu
    foreach ($result as $row)
    {
      $jokeId = $row['id'];
      $s->bindValue(':id', $jokeId);
      $s->execute();
    }
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy usuwaniu wpisów kategorii dla dowcipu.';
    include 'error.html.php';
    exit();
  }

  // Usuń dowcipy autora
  try
  {
    $sql = 'DELETE FROM joke WHERE authorid = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
```

```
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy usuwaniu dowcipów autora.';
    include 'error.html.php';
    exit();
  }

  // Usuń autora
  try
  {
    $sql = 'DELETE FROM author WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Error deleting author.';
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}
```

Choć większość z tego kodu powinna wydawać Ci się znajoma, jest tu kilka nowych elementów (zostały one wyróżnione tłustym drukiem).

Pierwszą nowością jest inicjująca dalsze działania instrukcja `if`:

kod w pliku chapter7/admin/authors/index.php (fragment)

```
if (isset($_POST['action']) and $_POST['action'] == 'Usuń')
```

Jak pisałem, użytkownik przekazuje żądanie usunięcia autora, klikając umieszczony przy jego nazwisku przycisk *Usuń*. Ponieważ atrybut `name` tego przycisku to `action`, możemy wykryć jego kliknięcie, sprawdzając, czy wartość `$_POST['action']` została ustawiona. Jeżeli tak, ustalamy, czy jest to wartość `'Usuń'`.

Następny nowy element to instrukcja:

kod w pliku chapter7/admin/authors/index.php (fragment)

```
$result = $s->fetchAll();
```

W tym miejscu skryptu wykonujemy kwerendę `SELECT`. Pobiera ona wszystkie dowcipy usuwanego autora. Dysponując nimi, będziemy mogli uruchomić sekwencję kwerend `DELETE`, po jednej dla każdego dowcipu autora, które usuną przypisania tego dowcipu do kategorii. Później wykorzystamy uzyskany zbiór wyników do usuwania samych dowcipów. Wymaga to jednak dodatkowych objaśnień.

Gdy wcześniej wykonywaliśmy kwerendy `SELECT`, używaliśmy warunku w pętli `while` lub `foreach` do pobierania kolejnych wierszy wyników:

```
while ($row = $result->fetch())
foreach ($result as $row)
```

Gdy korzystamy z wyników kwerendy w taki sposób, PHP w istocie pobiera kolejny wiersz za każdym razem, gdy żąda tego pętla, po czym gdy rozpoczyna się przetwarzanie następnego, „zapomina" poprzednie dane. Ponieważ w pamięci nigdy nie jest przechowywana całość zbioru wyników, pozwala to ekonomicznie wykorzystać zasoby serwera WWW.

W większości sytuacji programista nie musi przejmować się tym, jak w rzeczywistości PHP pracuje z rekordami bazy danych. Jednak od czasu do czasu może pojawić się potrzeba przesłania do serwera MySQL kolejnej kwerendy, niezależnej od postępów pracy z przetwarzaniem wyników wcześniejszej.

Z taką sytuacją mamy do czynienia tutaj: dopiero co wykonaliśmy kwerendę *Zapisz* pobierającą dowcipy określonego autora. W trakcie pracy z tą listą chcemy wykonać kwerendę `DELETE` dla każdego z nich. Jednak z punktu widzenia MySQL wciąż przetwarzamy wyniki kwerendy `SELECT`. Nie możemy po prostu przerwać tego i rozpocząć uruchamiania kwerend usuwających! Próba takiej operacji doprowadziłaby do zgłoszenia błędu.

To właśnie miejsce dla metody `fetchAll()`. Użycie jej dla naszej przygotowanej instrukcji `$s` doprowadzi do pobrania całego zbioru wyników kwerendy i zapisania ich w tablicy PHP o nazwie `$result`:

kod w pliku chapter7/admin/authors/index.php (fragment)

```
$result = $s->fetchAll();
```

Możemy teraz przetwarzać tablicę w pętli `foreach` w podobny sposób jak wcześniej obiekt `PDOStatement` i pracować z każdym wierszem niezależnie. Różnica polega na tym, że teraz PHP dysponuje wszystkimi wynikami od razu, co pozwala przesłać do serwera MySQL kolejne kwerendy.

To właśnie robi kolejny nowy fragment kodu:

kod w pliku chapter7/admin/authors/index.php (fragment)

```
// Usuń przypisania dowcipów do kategorii
try
{
  $sql = 'DELETE FROM jokecategory WHERE jokeid = :id';
  $s = $pdo->prepare($sql);

  // Dla każdego dowcipu
  foreach ($result as $row)
  {
    $jokeId = $row['id'];
    $s->bindValue(':id', $jokeId);
    $s->execute();
  }
}
```

Kod ten zapewnia uruchomienie kwerendy `DELETE` usuwającej wpisy w tabeli `jokecategory` dla każdego z dowcipów w bazie danych. Zwróć uwagę, że nie rozpoczynamy pisania kodu od zdefiniowania pętli `foreach` — najpierw przygotowujemy w bazie instrukcję SQL.

Widzimy tu drugą istotną zaletę korzystania z przygotowanych instrukcji SQL (poznałeś je, czytając rozdział 4.)[1]. Po przygotowaniu instrukcji można korzystać z niej wiele razy, przypisując jedynie zmieniające się wartości. W tym przypadku potrzebujemy sekwencji niemal identycznych kwerend `DELETE` — jedyna różnica to identyfikator dowcipu w klauzuli `WHERE`. Użycie przygotowanej instrukcji oszczędza MySQL pracy z wielokrotną interpretacją polecenia i przygotowywaniem planu jej wykonania. Kod SQL zostaje odczytany i przeanalizowany raz. Prowadzi to do określenia planu jego wykonania. Plan ten jest realizowany dla kolejnych wartości `id` przekazywanych przez serwer WWW.

Spójrz jeszcze raz na kod — po tych wyjaśnieniach powinien być bardziej zrozumiały. Najpierw przygotowywana jest instrukcja SQL. Następnie pętla `foreach` pracuje ze zbiorem wyników wcześniejszej kwerendy `SELECT`. Przygotowana instrukcja `DELETE` jest wywoływana jednokrotnie dla każdego dowcipu, po użyciu metody `bindValue` do przypisania symbolowi wieloznacznemu `:id` wartości identyfikatora.

Nie czuj się zażenowany, jeżeli pełne zrozumienie działania tego kodu wymaga od Ciebie dłuższego namysłu. Jest to najbardziej złożony fragment PHP, który znajdziesz w tej książce!

Gdy wiesz już dokładnie, jak działa nowy kod, przejdź do akcji i spróbuj naprawdę usunąć jednego z autorów. Użyj phpMyAdmin do weryfikacji zmian w tablicach dowcipów i przypisań do kategorii. Zdefiniowane kategorie powinny pozostać w bazie nawet, gdy nie są do nich przypisane żadne dowcipy.

Potwierdzanie operacji usuwania

Podejmij teraz próbę samodzielnej pracy i rozbuduj nową stronę o żądanie potwierdzenia operacji usuwania.

Jako punkt wyjścia możesz wykorzystać wersję z archiwum kodu książki. Zmodyfikuj kontroler tak, aby reagował na kliknięcie przycisku *Usuń* wyświetleniem kolejnego szablonu, tym razem z prośbą o potwierdzenie zamiaru wykonania operacji. Gdy użytkownik prześle *ten* formularz, powinno to spowodować wywołanie kodu kontrolera, który faktycznie usuwa dane. Zwróć uwagę, że nowy formularz musi przesyłać ukryte pole z identyfikatorem autora do usunięcia.

Dodawanie i zmienianie informacji o autorach

Implementacja łącza *Dodaj nowego autora* na początku strony może opierać się na kodzie obsługującym łącze *Dodaj nowy dowcip* z rozdziału 4. Zamiast żądać od użytkownika podania tekstu dowcipu, tym razem formularz powinien zawierać pola do wpisania imienia i nazwiska oraz adresu e-mail.

[1] Dla przypomnienia, pierwszą znaczącą zaletą przygotowanych instrukcji jest możliwość korzystania z symboli wieloznacznych, którym następnie przypisywana jest wartość. Pozwala to przypisywać wartości wykorzystywane w instrukcjach bazy danych bez obaw o „wstrzyknięty" SQL.

Jednak strona zarządzania autorami zawiera nową, pokrewną funkcję: modyfikacji danych autorów zapisanych wcześniej. Ponieważ zarówno dodawanie, jak i modyfikowanie rekordów wymaga podobnych formularzy, możemy upiec dwie pieczenie na jednym ogniu. Oto kod szablonu formularza, który posłuży zarówno do dodawania, jak i edycji autorów:

kod w pliku chapter7/admin/authors/form.html.php

```
 <?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <title><?php htmlout($pageTitle); ?></title>
  </head>
  <body>
    <h1><?php htmlout($pageTitle); ?></h1>
    <form action="?<?php htmlout($action); ?>" method="post">
      <div>
        <label for="name">Imię i nazwisko: <input type="text" name="name"
            id="name" value="<?php htmlout($name); ?>"></label>
      </div>
      <div>
        <label for="email">Adres e-mail: <input type="text" name="email"
            id="email" value="<?php htmlout($email); ?>"></label>
      </div>
      <div>
        <input type="hidden" name="id" value="<?php
            htmlout($id); ?>">
        <input type="submit" value="<?php htmlout($button); ?>">
      </div>
    </form>
  </body>
</html>
```

W treści tej strony wykorzystywanych jest sześć zmiennych PHP:

$pageTitle	określa treść tytułu i nagłówka najwyższego poziomu (<h1>)
$action	określa wartość przekazywaną ciągiem kwerendy URL przy wysyłaniu formularza
$name	określa początkową wartość pola formularza do wpisania imienia i nazwiska autora
$email	określa początkową wartość pola formularza do wpisania adresu e-mail autora
$id	określa wartość ukrytego pola formularza zawierającego identyfikator autora w bazie danych
$button	określa treść etykiety przycisku wysyłania formularza

Jest to zbiór zmiennych, które pozwalają nam wykorzystywać formularz dwojako: do dodawania wpisów i do modyfikowania wcześniejszych. Tabela 7.1 pokazuje wartości przypisywane zmiennym w każdej z tych sytuacji.

Tabela 7.1. Wartości zmiennych dla uniwersalnego formularza autorów

Zmienna	Wartość przy dodawaniu wpisu	Wartość przy zmienianiu wpisu
`$pageTitle`	`'Nowy autor'`	`'Edycja autora'`
`$action`	`'addform'`	`'editform'`
`$name`	`''` (ciąg pusty)	imię i nazwisko
`$email`	`''` (ciąg pusty)	adres e-mail
`$id`	`''` (ciąg pusty)	ID autora
`$button`	`'Dodaj autora'`	`'Aktualizuj autora'`

Oto kod kontrolera, który ładuje formularz w trybie dodawania autora po kliknięciu przycisku *Dodaj nowego autora*:

kod w pliku chapter7/admin/authors/index.php (fragment)

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

if (isset($_GET['add']))
{
  $pageTitle = 'Nowy autor';
  $action = 'addform';
  $name = '';
  $email = '';
  $id = '';
  $button = 'Dodaj autora';

  include 'form.html.php';
  exit();
}
```

Gdy autor prześle formularz w tym trybie, możesz to wykryć, sprawdzając wartość `$_GET['addform']`:

kod w pliku chapter7/admin/authors/index.php (fragment)

```
if (isset($_GET['addform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'INSERT INTO author SET
        name = :name,
        email = :email';
    $s = $pdo->prepare($sql);
    $s->bindValue(':name', $_POST['name']);
    $s->bindValue(':email', $_POST['email']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy dodawaniu nowego autora.';
    include 'error.html.php';
    exit();
```

```
  }

  header('Location: .');
  exit();
}
```

Gdy użytkownik kliknie przycisk *Edycja* na liście autorów, używamy tego samego formularza, ale tym razem pobieramy z bazy informacje o autorze:

kod w pliku chapter7/admin/authors/index.php (fragment)

```
if (isset($_POST['action']) and $_POST['action'] == 'Edycja')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'SELECT id, name, email FROM author WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy pobieraniu danych autora.';
    include 'error.html.php';
    exit();
  }

  $row = $s->fetch();

  $pageTitle = 'Edycja autora';
  $action = 'editform';
  $name = $row['name'];
  $email = $row['email'];
  $id = $row['id'];
  $button = 'Aktualizuj autora';

  include 'form.html.php';
  exit();
}
```

Formularz przesłany w tym trybie wykrywasz, sprawdzając wartość `$_GET['editform']`. Kod przetwarzający przesłane dane jest podobny do stosowanego przy dodawaniu autora, ale miejsce kwerendy `INSERT` zajmuje kwerenda `UPDATE`.

kod w pliku chapter7/admin/authors/index.php (fragment)

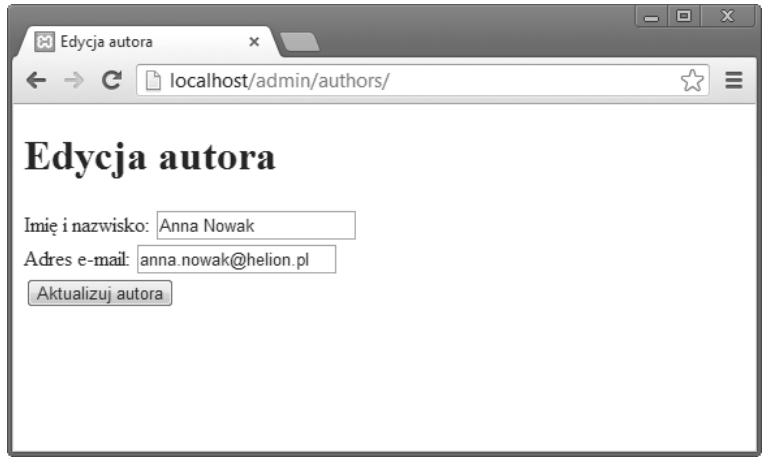
```
if (isset($_GET['editform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'UPDATE author SET
        name = :name,
        email = :email
        WHERE id = :id';
```

```
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->bindValue(':name', $_POST['name']);
    $s->bindValue(':email', $_POST['email']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy aktualizowaniu danych autora.';
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}
```

To wszystko! Wypróbuj zaktualizowany system zarządzania autorami z nowym, uniwersalnym szablonem widocznym na rysunku 7.4. Upewnij się, że działają funkcje dodawania, zmieniania i usuwania danych autorów. Jeżeli pojawią się komunikaty błędów, przejrzyj kod i sprawdź, czy nie pomyliłeś się przy jego wpisywaniu. W razie problemów użyj gotowego kodu z archiwum przykładów i porównaj go z własnym.



Rysunek 7.4. Miejmy nadzieję, że nowy autor wyśle coś ciekawego…

Zarządzanie kategoriami

Role listy autorów i listy kategorii dowcipów są w bazie danych dość podobne. Obie są przechowywane w odrębnych tabelach i obie służą do łączenia dowcipów w grupy. Oznacza to, że kod obsługujący kategorie może być bardzo podobny do kodu pracującego z autorami. Jest tylko jedna większa różnica.

Przy usuwaniu kategorii nie powinniśmy jednocześnie usuwać wszystkich przypisanych do niej dowcipów. Często należą one do innych kategorii. Możemy sprawdzać każdy kolejny dowcip i jego

powiązania z kategoriami i usuwać tylko te, które nie mają żadnych przypisań. Jest to jednak dość skomplikowany i czasochłonny proces. Możemy zdecydować, że dowcipy nieprzypisane do kategorii pozostają w bazie. W pewnych sytuacjach spowoduje to, że nie będą wyświetlane (przy pewnych konfiguracjach mechanizmu wyświetlania zawartości tabeli dowcipów), pozostaną jednak w bazie, oczekując na przypisanie kategorii w przyszłości.

Usuwanie kategorii pociąga za sobą jedynie usunięcie powiązanych z nią wpisów w tabeli `jokecategory`:

kod w pliku chapter7/admin/categories/index.php (fragment)

```
  // Usuń powiązania dowcipów z tą kategorią
  try
  {
    $sql = 'DELETE FROM jokecategory WHERE categoryid = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy usuwaniu powiązań dowcipów z usuwaną kategorią.';
    include 'error.html.php';
    exit();
  }

  // Usuń kategorię
  try
  {
    $sql = 'DELETE FROM category WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy usuwaniu kategorii.';
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}
```

Poza tym jednym szczegółem zarządzanie kategoriami nie różni się od zarządzania autorami. Poniżej przedstawiony jest kod wszystkich czterech niezbędnych plików. Wykorzystuje on współużytkowane pliki dołączane z rozdziału 6.: *db.inc.php*, *magicquotes.inc.php* i *helpers.inc.php*.

kod w pliku chapter7/admin/categories/index.php

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

if (isset($_GET['add']))
{
```

```
  $pageTitle = 'Nowa kategoria';
  $action = 'addform';
  $name = '';
  $id = '';
  $button = 'Dodaj kategorię';

  include 'form.html.php';
  exit();
}

if (isset($_GET['addform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'INSERT INTO category SET
        name = :name';
    $s = $pdo->prepare($sql);
    $s->bindValue(':name', $_POST['name']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy dodawaniu kategorii.';
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}

if (isset($_POST['action']) and $_POST['action'] == 'Edycja')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'SELECT id, name FROM category WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy pobieraniu informacji o kategorii.';
    include 'error.html.php';
    exit();
  }

  $row = $s->fetch();

  $pageTitle = 'Edycja kategorii';
  $action = 'editform';
  $name = $row['name'];
  $id = $row['id'];
  $button = 'Aktualizuj kategorię';
```

```
  include 'form.html.php';
  exit();
}

if (isset($_GET['editform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'UPDATE category SET
        name = :name
        WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->bindValue(':name', $_POST['name']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy aktualizowaniu informacji o kategorii.';
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}

if (isset($_POST['action']) and $_POST['action'] == 'Usuń')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  // Usuń powiązania dowcipów z tą kategorią
  try
  {
    $sql = 'DELETE FROM jokecategory WHERE categoryid = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy usuwaniu powiązań dowcipów z usuwaną kategorią.';
    include 'error.html.php';
    exit();
  }

  // Usuń kategorię
  try
  {
    $sql = 'DELETE FROM category WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy usuwaniu kategorii.';
    include 'error.html.php';
    exit();
```

```
  }

  header('Location: .');
  exit();
}

// Wyświetl listę kategorii
include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

try
{
  $result = $pdo->query('SELECT id, name FROM category');
}
catch (PDOException $e)
{
  $error = 'Błąd przy pobieraniu listy kategorii!';
  include 'error.html.php';
  exit();
}

foreach ($result as $row)
{
  $categories[] = array('id' => $row['id'], 'name' => $row['name']);
}

include 'categories.html.php';
```

kod w pliku chapter7/admin/categories/categories.html.php

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <title>Zarządzanie kategoriami</title>
  </head>
  <body>
    <h1>Zarządzanie kategoriami</h1>
    <p><a href="?add">Dodaj nową kategorię</a></p>
    <ul>
      <?php foreach ($categories as $category): ?>
        <li>
          <form action="" method="post">
            <div>
              <?php htmlout($category['name']); ?>
              <input type="hidden" name="id" value="<?php
                  echo $category['id']; ?>">
              <input type="submit" name="action" value="Edycja">
              <input type="submit" name="action" value="Usuń">
            </div>
          </form>
        </li>
      <?php endforeach; ?>
    </ul>
    <p><a href="..">Powrót do strony głównej CMS</a></p>
  </body>
</html>
```

kod w pliku chapter7/admin/categories/form.html.php

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <title><?php htmlout($pageTitle); ?></title>
  </head>
  <body>
    <h1><?php htmlout($pageTitle); ?></h1>
    <form action="?<?php htmlout($action); ?>" method="post">
      <div>
        <label for="name">Nazwa kategorii: <input type="text" name="name"
            id="name" value="<?php htmlout($name); ?>"></label>
      </div>
      <div>
        <input type="hidden" name="id" value="<?php
            htmlout($id); ?>">
        <input type="submit" value="<?php htmlout($button); ?>">
      </div>
    </form>
  </body>
</html>
```

kod w pliku chapter7/admin/categories/error.html.php

```
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <title>Błąd skryptu</title>
  </head>
  <body>
    <p>
      <?php echo $error; ?>
    </p>
  </body>
</html>
```

Zarządzanie dowcipami

Poza dodawaniem, usuwaniem i modyfikowaniem dowcipów w bazie musimy mieć możliwość przypisywania im kategorii i autorów. Co więcej, dowcipów będzie zapewne znacznie więcej niż autorów czy kategorii. Próba wyświetlenia ich pełnej listy, jak to robiliśmy przy autorach i kategoriach, mogłaby doprowadzić do utworzenia mało przyjaznej strony, na której znalezienie jednego konkretnego dowcipu byłoby niemal niemożliwe. Niezbędne jest zatem zapewnienie nieco bardziej wyszukanej metody przeglądania zgromadzonych zbiorów.

Wyszukiwanie dowcipów

Często się zapewne zdarzy, że będziemy znali kategorię, autora lub część tekstu dowcipu. Zapewnijmy więc możliwość wyszukiwania wpisów na podstawie tego rodzaju kryteriów. Powinniśmy uzyskać w ten sposób prostą wyszukiwarkę.

Formularz, który pyta administratora o znane mu informacje o poszukiwanym dowcipie, powinien zapewniać gotowe listy kategorii i autorów. Rozpocznijmy pisanie kodu od części kontrolera, która przygotuje te dane.

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
// Wyświetl formularz wyszukiwania
include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

try
{
  $result = $pdo->query('SELECT id, name FROM author');
}
catch (PDOException $e)
{
  $error = 'Błąd przy pobieraniu listy autorów!';
  include 'error.html.php';
  exit();
}

foreach ($result as $row)
{
  $authors[] = array('id' => $row['id'], 'name' => $row['name']);
}

try
{
  $result = $pdo->query('SELECT id, name FROM category');
}
catch (PDOException $e)
{
  $error = 'Błąd przy pobieraniu listy kategorii!';
  include 'error.html.php';
  exit();
}

foreach ($result as $row)
{
  $categories[] = array('id' => $row['id'], 'name' => $row['name']);
}

include 'searchform.html.php';
```

Przedstawiony kod buduje dwie tablice, z których będzie korzystał szablon *searchform.html.php*: `$authors` i `$categories`. Posłużą one do wyświetlenia w formularzu list rozwijanych:

kod w pliku chapter7/admin/jokes/searchform.html.php

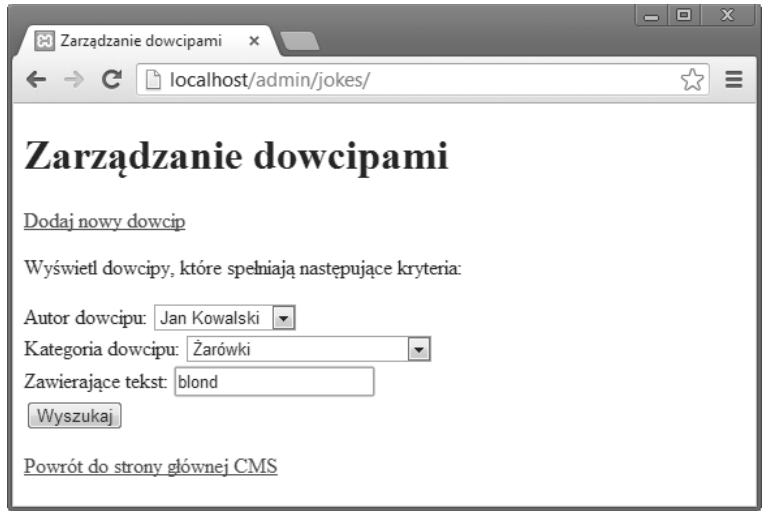
```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
```

```
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <title>Zarządzanie dowcipami</title>
  </head>
  <body>
    <h1>Zarządzanie dowcipami</h1>
    <p><a href="?add">Dodaj nowy dowcip</a></p>
    <form action="" method="get">
      <p>Wyświetl dowcipy, które spełniają następujące kryteria:</p>
      <div>
        <label for="author">Autor dowcipu:</label>
        <select name="author" id="author">
          <option value="">Dowolny autor</option>
          <?php foreach ($authors as $author): ?>
            <option value="<?php htmlout($author['id']); ?>"><?php
                htmlout($author['name']); ?></option>
          <?php endforeach; ?>
        </select>
      </div>
      <div>
        <label for="category">Kategoria dowcipu:</label>
        <select name="category" id="category">
          <option value="">Dowolna kategoria</option>
          <?php foreach ($categories as $category): ?>
            <option value="<?php htmlout($category['id']); ?>"><?php
                htmlout($category['name']); ?></option>
          <?php endforeach; ?>
        </select>
      </div>
      <div>
        <label for="text">Zawierające tekst:</label>
        <input type="text" name="text" id="text">
      </div>
      <div>
        <input type="hidden" name="action" value="search">
        <input type="submit" value="Wyszukaj">
      </div>
    </form>
    <p><a href="..">Powrót do strony głównej CMS</a></p>
  </body>
</html>
```

Jak widać, elementy `option` list `select` budujemy przy użyciu pętli PHP `foreach`. Wartością opcji (`value`) jest identyfikator autora lub kategorii. Z kolei ich etykiety tekstowe to imię i nazwisko autora lub nazwa kategorii. Obie listy rozwijane zaczynają się od opcji bez wartości. Pozwala to wyłączyć odpowiadające jej pole z kryteriów wyszukiwania.

Zwróć uwagę, że atrybut `method` formularza ma wartość `get`. Pozwala to zapisać wyniki wyszukiwania jako zakładkę przeglądarki — przesyłane wartości są przekazywane jako ciąg kwerendy URL. Jest to wskazane dla większości formularzy wyszukiwania. Gotowy formularz widać na rysunku 7.5.

Zadaniem kontrolera będzie teraz wykorzystanie przesyłanych z tym formularzem wartości do zbudowania listy dowcipów odpowiadającej określonym w nim kryteriom. Posłuży do tego, oczywi-



Rysunek 7.5. Coś z klasyki

ście, kwerenda `SELECT`, jednak jej konstrukcja będzie silnie uzależniona od dokonanego wyboru kryteriów. Ponieważ budowanie tej instrukcji jest dość złożonym procesem, prześledźmy odpowiedzialny za to kod wiersz po wierszu.

Rozpoczynamy od zdefiniowania kilku ciągów znakowych, których połączenie doprowadzi do utworzenia kwerendy `SELECT` odpowiadającej za wyszukiwanie w sytuacji, gdy żadne kryteria nie zostały w formularzu określone:

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
if (isset($_GET['action']) and $_GET['action'] == 'search')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  // Podstawowa instrukcja SELECT
  $select = 'SELECT id, joketext';
  $from   = ' FROM joke';
  $where  = ' WHERE TRUE';
```

Nieco zaskakująca może być klauzula `WHERE`. Dążymy do tego, aby mieć możliwość dalszej rozbudowy tej podstawowej formy instrukcji `SELECT` na podstawie wybranych w formularzu kryteriów. Kryteria te zdecydują o elementach dołączanych do klauzul `FROM` i `WHERE`. Gdy jednak żadne kryteria nie zostaną wybrane (administrator chce wyświetlić wszystkie dowcipy), klauzula `WHERE` nie będzie potrzebna! Ponieważ trudno dodać cokolwiek do klauzuli, której nie ma, pożądane jest utworzenie zapisu, który będzie równie neutralny, jak jego brak. Ponieważ wartość `TRUE` to zawsze „prawda”, klauzula `WHERE TRUE` jest dokładnie tym, czego potrzebujemy[2].

[2] Najkrótsza postać takiej klauzuli `WHERE` to `WHERE 1` — w języku SQL każda liczba dodatnia ma wartość logiczną „prawda”. Jeżeli wydaje Ci się to bardziej przejrzyste, możesz wprowadzić zmianę w kodzie.

Naszym kolejnym krokiem jest sprawdzenie kolejno wszystkich ograniczeń wyszukiwania (według autora, według kategorii i na podstawie wyszukiwania ciągu znaków), które mogły zostać określone w formularzu. Będzie to prowadziło do rozbudowy różnych części kwerendy SQL. Rozpoczniemy od autora. Pusta opcja w formularzu ma wartość `""`. Jeżeli zatem wartość pola formularza, dostępna jako `$_GET['author']`, jest różna od `''` (ciąg pusty), oznacza to, że autor został wybrany i kwerenda musi zostać zmodyfikowana:

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
$placeholders = array();

if ($_GET['author'] != '') // Autor został wybrany
{
  $where .= " AND authorid = :authorid";
  $placeholders[':authorid'] = $_GET['author'];
}
```

Jak widzieliśmy już wcześniej, do dołączania nowego ciągu na końcu zdefiniowanego wcześniej używamy **operatora dołączania** (ang. *append operator*) `.=`. W tym przypadku dodajemy do klauzuli `WHERE` warunek stanowiący, że wartość `authorid` w tabeli `joke` musi być zgodna z wartością symbolu zastępczego, `:authorid`. Nie możemy jeszcze użyć metody `bindValue`, bo jest to metoda obiektu reprezentującego przygotowaną instrukcję, a taką utworzymy dopiero później. Na tym etapie elementy kwerendy pozostają jeszcze rozdzielone pomiędzy trzy ciągi znakowe, `$select`, `$from` i `$where`. Dopóki nie zostaną one połączone w całość i przekazane MySQL, wartości symboli zastępczych będziemy przechowywać w zmiennej tablicowej PHP `$placeholders`. Nazwy symboli zastępczych służą w niej jako indeksy elementów.

Przechodzimy do kategorii dowcipu:

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
if ($_GET['category'] != '') // Kategoria została wybrana
{
  $from  .= ' INNER JOIN jokecategory ON id = jokeid';
  $where .= " AND categoryid = :categoryid";
  $placeholders[':categoryid'] = $_GET['category'];
}
```

Ponieważ przypisania dowcipów do kategorii są przechowywane w tabeli `jokecategory`, musimy dodać ją do kwerendy i utworzyć złączenie tabel. Dodajemy w tym celu `INNER JOIN jokecategory ON id = jokeid` na końcu zmiennej `$from`. Zapewnia to złączenie dwóch tabel na podstawie zgodności kolumny `id` w tabeli `joke` z kolumną `jokeid` w tabeli `jokecategory`.

Dysponując złączeniem dwóch tabel, możemy użyć warunku określonego przy przesyłaniu formularza — przynależności dowcipu do pewnej kategorii. Do zmiennej `$where` musi zostać dodany warunek zgodności kolumny `categoryid` w tabeli `jokecategory` z podaną przez użytkownika kategorią (`:categoryid`). Przypisywaną symbolowi zastępczemu wartość `$_GET['category']` ponownie zapisujemy w tablicy `$placeholders`.

Wyszukiwanie ciągów znakowych jest stosunkowo proste dzięki operatorowi SQL `LIKE`, który poznaliśmy w rozdziale 2.:

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
if ($_GET['text'] != '') // Podano tekst do wyszukania
{
  $where .= " AND joketext LIKE :joketext";
  $placeholders[':joketext'] = '%' . $_GET['text'] . '%';
}
```

Aby uzyskać pożądany symbol zastępczy, dodajemy teraz do wartości `$_GET['text']` znaki procentu (`%`). Przypomnijmy, że znaki te służą jako symbole wieloznaczne. Efektem będzie wyszukiwanie tekstu, który zawiera ciąg `$_GET['text']`. Znaki `%` reprezentują dowolny ciąg przed i po `$_GET['text']` w przeszukiwanym polu.

Gdy wszystkie elementy kwerendy SQL są gotowe, możemy je połączyć, a uzyskaną instrukcję wykorzystać do pobierania danych.

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
  try
  {
    $sql = $select . $from . $where;
    $s = $pdo->prepare($sql);
    $s->execute($placeholders);
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy pobieraniu dowcipów.';
    include 'error.html.php';
    exit();
  }

  foreach ($s as $row)
  {
    $jokes[] = array('id' => $row['id'], 'text' => $row['joketext']);
  }

  include 'jokes.html.php';
  exit();
}
```

Zwróć uwagę na wyróżniony wiersz. Ponieważ wartości symboli zastępczych zostały zapisane w zmiennej tablicowej PHP `$placeholders`, wykorzystujemy bardzo poręczną cechę metody `execute` — zamiast wielokrotnie wywoływać metodę `bindValue`, możemy przy uruchamianiu kwerendy przekazać wszystkie wartości symboli zastępczych, wskazując po prostu tablicę.

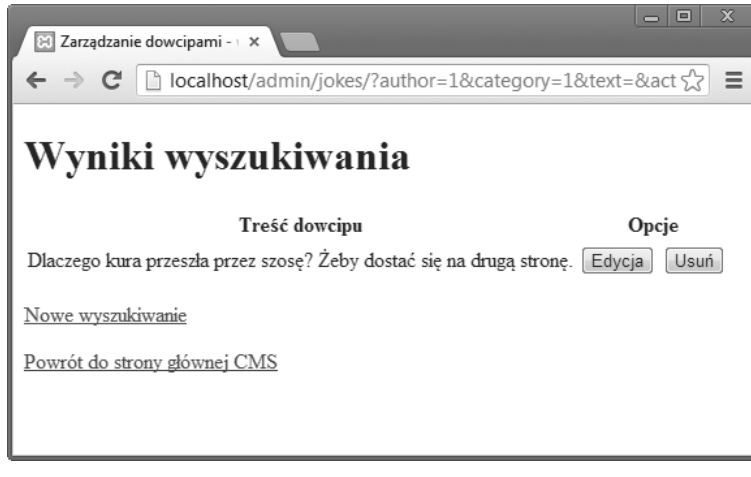
Szablon do wyświetlania dowcipów będzie zawierał przyciski *Edycja* i *Usuń* dla każdego z dowcipów. Aby zachować porządek na stronie, użyjemy tabeli HTML:

kod w pliku chapter7/admin/jokes/jokes.html.php

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
```

```
    <title>Zarządzanie dowcipami - wyniki wyszukiwania</title>
  </head>
  <body>
    <h1>Wyniki wyszukiwania</h1>
    <?php if (isset($jokes)): ?>
      <table>
        <tr><th>Treść dowcipu</th><th>Opcje</th></tr>
        <?php foreach ($jokes as $joke): ?>
        <tr>
          <td><?php htmlout($joke['text']); ?></td>
          <td>
            <form action="?" method="post">
              <div>
                <input type="hidden" name="id" value="<?php
                    htmlout($joke['id']); ?>">
                <input type="submit" name="action" value="Edycja">
                <input type="submit" name="action" value="Usuń">
              </div>
            </form>
          </td>
        </tr>
        <?php endforeach; ?>
      </table>
    <?php endif; ?>
    <p><a href="?">Nowe wyszukiwanie</a></p>
    <p><a href="..">Powrót do strony głównej CMS</a></p>
  </body>
</html>
```

Wyniki wyszukiwania widać na rysunku 7.6.



Rysunek 7.6. Dowcip znaleziony

Brak wyników

Jeżeli czujesz się na siłach, spróbuj rozbudować szablon o kod zapewniający ładną obsługę sytuacji, gdy żaden z dowcipów nie spełnia określonych w formularzu kryteriów. W obecnej formie w przypadku braku wyników szablon pozostawia po prostu puste miejsce.

Dodawanie i zmienianie dowcipów

Na początku formularza wyszukiwania dowcipów umieściliśmy łącze do wprowadzania nowych:

kod w pliku chapter7/admin/jokes/searchform.html.php (fragment)

```
<p><a href="?add">Dodaj nowy dowcip</a></p>
```

Zaimplementujmy teraz tę funkcję. Kod będzie bardzo podobny do używanego przy dodawaniu autorów i kategorii, jednak poza wpisaniem samej treści dowcipu musimy zapewnić administratorowi strony przypisanie autora i kategorii.

Podobnie jak przy autorach i kategoriach, użyjemy tego samego formularza do dodawania nowych wpisów i edycji wcześniejszych. Prześledźmy najważniejsze elementy tego formularza. Rozpoczynamy od standardowego obszaru do wpisywania tekstu. W przypadku edycji dowcipu umieszczamy w tym polu jego treść (`$text`):

kod w pliku chapter7/admin/jokes/form.html.php (fragment)

```
<div>
  <label for="text">Wpisz tekst dowcipu:</label>
  <textarea id="text" name="text" rows="3" cols="40"><?php
      htmlout($text); ?></textarea>
</div>
```

Następnie zapewniamy możliwość wyboru autora:

kod w pliku chapter7/admin/jokes/form.html.php (fragment)

```
<div>
  <label for="author">Autor:</label>
  <select name="author" id="author">
    <option value="">Wybierz autora</option>
    <?php foreach ($authors as $author): ?>
      <option value="<?php htmlout($author['id']); ?>"<?php
          if ($author['id'] == $authorid)
          {
            echo ' selected';
          }
          ?>><?php htmlout($author['name']); ?></option>
    <?php endforeach; ?>
  </select>
</div>
```

Lista rozwijana jest już znajomym elementem, z którym miałeś do czynienia między innym w formularzu wyszukiwania dowcipów. Ważną różnicą jest jednak to, że chcemy mieć wpływ na początkowy wybór na tej liście, gdy formularz jest wykorzystywany do edycji (a nie do wprowadzania nowego dowcipu). Wyróżniony fragment kodu wstawia w znaczniku `<option>` atrybut `selected`, jeżeli dodawany do listy rozwijanej autor (`$author['id']`) to autor wyświetlanego dowcipu (`$authorid`).

Kolejnym elementem jest wybór kategorii, do których dowcip powinien zostać przypisany. Lista rozwijana nie jest w tym przypadku rozwiązaniem, ponieważ istnieje możliwość wyboru *wielu* kategorii

dla tego samego dowcipu. Użyjemy zatem grupy pól wyboru (`<input type="checkbox">`) — po jednym dla każdej kategorii. Ponieważ w chwili pisania kodu nie znamy ich liczby, wybór wartości dla ich atrybutu `name` staje się pewnym wyzwaniem.

Odpowiedzią jest użycie jednej zmiennej dla wszystkich pól wyboru. Wszystkie one będą miały tę samą nazwę. Aby powiązać wiele wartości z jedną nazwą zmiennej, użyjemy *tablicy*. Przypomnijmy z rozdziału 3., że tablica to pojedyncza zmienna, która może przechowywać wiele wartości. Aby przekazać element formularza jako element zmiennej tablicowej, wystarczy dodać parę nawiasów kwadratowych na końcu wartości atrybutu `name` (w tym przypadku wartością tą będzie `categories[]`).

Lista z możliwością wielokrotnego wyboru

Inną możliwością przekazania tablicy jest użycie znacznika `<select multiple="multiple">`. W tym przypadku również nadajemy atrybutowi `name` wartość zakończoną nawiasami kwadratowymi. Formularz przekazuje następnie tablicę wszystkich wartości `option` wybranych przez użytkownika z listy.

Możesz spróbować samodzielnych eksperymentów z modyfikacją formularza w taki sposób, aby wyświetlał kategorie jako listę elementów `option`. Pamiętaj jednak, że wielu użytkowników nie domyśli się w takiej sytuacji, że istnieje możliwość wyboru więcej niż jednej opcji po wciśnięciu klawisza *Ctrl* (lub *Command* na komputerach macintosh).

Ponieważ wszystkie pola wyboru mają taką samą nazwę, potrzebujemy innego sposobu identyfikowania tych, które zostały wybrane. Umożliwią to przypisane im różne wartości. Wartościami tymi będą identyfikatory kategorii z bazy danych. Pozwoli to formularzowi przekazać tablicę wartości `id` dla wszystkich kategorii, z którymi dany dowcip powinien zostać powiązany.

Ponownie, aby umożliwić edycję wpisanego wcześniej dowcipu, zapewnimy kod generujący odpowiednik atrybutu `selected` dla kategorii, do których żart został już przypisany. Będzie to sygnalizowane przez kontroler ustawieniem wartości `$category['selected']` na `TRUE`:

kod w pliku chapter7/admin/jokes/form.html.php (fragment)

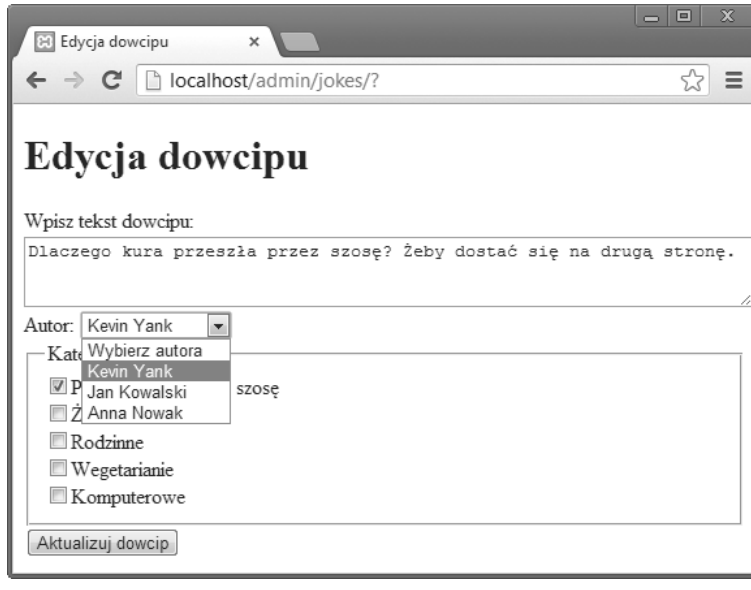
```
<fieldset>
  <legend>Kategorie:</legend>
  <?php foreach ($categories as $category): ?>
    <div><label for="category<?php htmlout($category['id']);
        ?>"><input type="checkbox" name="categories[]"
        id="category<?php htmlout($category['id']); ?>"
        value="<?php htmlout($category['id']); ?>"<?php
        if ($category['selected'])
        {
          echo ' checked';
        }
        ?>><?php htmlout($category['name']); ?></label></div>
  <?php endforeach; ?>
</fieldset>
```

Poza tymi szczegółami, formularz będzie działał podobnie jak inne formularze dodawania i edycji zbudowane w tym rozdziale. Oto jego pełny kod:

kod w pliku chapter7/admin/jokes/form.html.php

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <title><?php htmlout($pageTitle); ?></title>
    <style type="text/css">
    textarea {
      display: block;
      width: 100%;
    }
    </style>
  </head>
  <body>
    <h1><?php htmlout($pageTitle); ?></h1>
    <form action="?<?php htmlout($action); ?>" method="post">
      <div>
        <label for="text">Wpisz tekst dowcipu:</label>
        <textarea id="text" name="text" rows="3" cols="40"><?php
            htmlout($text); ?></textarea>
      </div>
      <div>
        <label for="author">Autor:</label>
        <select name="author" id="author">
          <option value="">Wybierz autora</option>
          <?php foreach ($authors as $author): ?>
            <option value="<?php htmlout($author['id']); ?>"<?php
                if ($author['id'] == $authorid)
                {
                  echo ' selected';
                }
                ?>><?php htmlout($author['name']); ?></option>
          <?php endforeach; ?>
        </select>
      </div>
      <fieldset>
        <legend>Kategorie:</legend>
        <?php foreach ($categories as $category): ?>
          <div><label for="category<?php htmlout($category['id']);
              ?>"><input type="checkbox" name="categories[]"
              id="category<?php htmlout($category['id']); ?>"
              value="<?php htmlout($category['id']); ?>"<?php
              if ($category['selected'])
              {
                echo ' checked';
              }
              ?>><?php htmlout($category['name']); ?></label></div>
        <?php endforeach; ?>
      </fieldset>
      <div>
        <input type="hidden" name="id" value="<?php
            htmlout($id); ?>">
        <input type="submit" value="<?php htmlout($button); ?>">
      </div>
    </form>
  </body>
</html>
```

Rysunek 7.7 przedstawia formularz w przeglądarce.



Rysunek 7.7. Edycja dowcipu

Przejdźmy teraz do kontrolera i kodu odpowiedzialnego za wyświetlanie formularza oraz obsługę przesyłanych danych.

Gdy użytkownik klika łącze *Dodaj nowy dowcip*, kontroler ma wyświetlić formularz, w którym wszystkie pola są puste. Żaden element tego kodu nie powinien być nowością. Nie śpiesz się, dokładnie się z nim zapoznaj i sprawdź, czy wszystko rozumiesz. Jeżeli masz wątpliwości co do przeznaczenia zmiennych, wyszukaj je w szablonie (gdzie ich rola powinna być wyraźnie widoczna).

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

if (isset($_GET['add']))
{
  $pageTitle = 'Nowy dowcip';
  $action = 'addform';
  $text = '';
  $authorid = '';
  $id = '';
  $button = 'Dodaj dowcip';

  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  // Buduj listę autorów
  try
  {
    $result = $pdo->query('SELECT id, name FROM author');
  }
```

```
  catch (PDOException $e)
  {
    $error = 'Błąd przy pobieraniu listy autorów.';
    include 'error.html.php';
    exit();
  }

  foreach ($result as $row)
  {
    $authors[] = array('id' => $row['id'], 'name' => $row['name']);
  }

  // Buduj listę kategorii
  try
  {
    $result = $pdo->query('SELECT id, name FROM category');
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy pobieraniu listy kategorii.';
    include 'error.html.php';
    exit();
  }

  foreach ($result as $row)
  {
    $categories[] = array(
        'id' => $row['id'],
        'name' => $row['name'],
        'selected' => FALSE);
  }

  include 'form.html.php';
  exit();
}
```

Zauważ, że przypisujemy wartość `FALSE` każdemu elementowi `'selected'` zapisywanemu w tablicach przechowywanych w tablicy `$categories`. Zapewnia to, że żadne z pól wyboru kategorii nie będzie domyślnie zaznaczone.

Gdy użytkownik klika przycisk *Edycja* obok dowcipu, kontroler ładuje formularz z danymi z bazy. Ogólna struktura kodu pozostaje jednak podobna, jak przy generowaniu pustego formularza:

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
if (isset($_POST['action']) and $_POST['action'] == 'Edycja')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'SELECT id, joketext, authorid FROM joke WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
```

```
  $error = 'Błąd przy pobieraniu informacji o dowcipie.';
  include 'error.html.php';
  exit();
}
$row = $s->fetch();

$pageTitle = 'Edycja dowcipu';
$action = 'editform';
$text = $row['joketext'];
$authorid = $row['authorid'];
$id = $row['id'];
$button = 'Aktualizuj dowcip';

// Buduj listę autorów
try
{
  $result = $pdo->query('SELECT id, name FROM author');
}
catch (PDOException $e)
{
  $error = 'Błąd przy pobieraniu listy autorów.';
  include 'error.html.php';
  exit();
}

foreach ($result as $row)
{
  $authors[] = array('id' => $row['id'], 'name' => $row['name']);
}

// Pobierz listę kategorii zawierających ten dowcip
try
{
  $sql = 'SELECT categoryid FROM jokecategory WHERE jokeid = :id'; ❶
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $id);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Błąd przy pobieraniu listy wybranych kategorii.';
  include 'error.html.php';
  exit();
}

foreach ($s as $row)
{
  $selectedCategories[] = $row['categoryid']; ❷
}

// Buduj listę wszystkich kategorii
try
{
  $result = $pdo->query('SELECT id, name FROM category');
}
catch (PDOException $e)
{
  $error = 'Błąd przy pobieraniu listy kategorii.';
  include 'error.html.php';
```

```
    exit();
  }

  foreach ($result as $row)
  {
    $categories[] = array(
        'id' => $row['id'],
        'name' => $row['name'],
        'selected' => in_array($row['id'], $selectedCategories)); ❸
  }

  include 'form.html.php';
  exit();
}
```

Poza pobieraniem informacji o dowcipie (identyfikator, treść i identyfikator autora) kod ten pobiera listę kategorii, do których dany dowcip należy:

❶ Kwerenda `SELECT` jest dość prosta, ponieważ pobiera jedynie rekordy z tabeli łączącej `joke ↪category`. Potrzebne są wyłącznie identyfikatory kategorii przypisanych edytowanemu dowcipowi.

❷ Pętla `foreach` zapisuje uzyskane identyfikatory kategorii w zmiennej tablicowej o nazwie `$selectedCategories`.

❸ Tutaj pojawia się pewna komplikacja. Podczas budowania listy *wszystkich* kategorii, które mają na formularzu przyjąć postać pól wyboru, sprawdzamy identyfikator każdej z nich, ustalając, czy znalazła się ona w tablicy `$selectedCategories`. Znacznie ułatwia to standardowa funkcja `in_array`. Uzyskaną wartość (`TRUE` lub `FALSE`) przypisujemy elementowi `'selected'` tablicy reprezentującej daną kategorię. Wartość ta zostaje następnie wykorzystana w szablonie formularza (na co zwracałem uwagę już wcześniej).

Na tym kończy się kod generujący formularz w obu trybach, wprowadzania i edycji danych. Dalsza część to obsługa formularza po jego przesłaniu.

Ponieważ po raz pierwszy będziemy mieli teraz do czynienia z przesyłaniem do serwera tablicy danych (listy zaznaczonych pól wyboru), w kodzie, który pracuje z formularzem, pojawi się kilka nowości. Początek jest stosunkowo prosty — dodajemy dowcip do tablicy `joke`. Ponieważ wymagane jest podanie autora, upewniamy się, że element `$_POST['author']` zawiera wartość. Uniemożliwi to administratorowi wybranie zamiast autora pozycji *Wybierz autora* (której odpowiada ciąg pusty, `""`).

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
if (isset($_GET['addform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  if ($_POST['author'] == '')
  {
    $error = 'Musisz wybrać autora dowcipu.
        Kliknij &lsquo;Wstecz&rsquo; i spróbuj ponownie.';
    include 'error.html.php';
    exit();
  }
```

```
try
{
  $sql = 'INSERT INTO joke SET
      joketext = :joketext,
      jokedate = CURDATE(),
      authorid = :authorid';
  $s = $pdo->prepare($sql);
  $s->bindValue(':joketext', $_POST['text']);
  $s->bindValue(':authorid', $_POST['author']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Błąd przy dodawaniu dowcipu.';
  include 'error.html.php';
  exit();
}

$jokeid = $pdo->lastInsertId();
```

W ostatnim wierszu pojawia się metoda, której jeszcze nie używaliśmy: `lastInsertId`. Zwraca ona liczbę przypisaną ostatniemu wpisowi, w którym wykorzystano mechanizm MySQL `AUTO_INCREMENT`. Innymi słowy, zwraca ona `id` wstawionego właśnie dowcipu. Identyfikator ten będzie za chwilę potrzebny.

Spodziewam się, że nie masz jeszcze klarownej idei tego, jak powinien być napisany kod wstawiający rekordy do tabeli `jokecategory` na podstawie zaznaczonych pól wyboru. Podstawowym problemem jest to, że we wcześniejszych przykładach nie pokazałem, jak wartość pola wyboru trafia do zmiennej PHP. Dodatkowa trudność polega na tym, że w tym przypadku pozyskane dane mają trafić do zmiennej tablicowej.

Typowe pole wyboru przekazuje wartość do zmiennej PHP wtedy, gdy jest zaznaczone. Gdy nie jest, wartość nie zostaje przypisana. Pola wyboru bez określonych wartości przekazują wartość `'on'` (lub żadną, gdy nie są zaznaczone). W tym przypadku pola wyboru mają wartości odpowiadające identyfikatorom kategorii.

Fakt, że dane z naszych pól wyboru trafiają do tablicy, jest w rzeczywistości ułatwieniem. W wyniku przesłania formularza otrzymamy jeden z dwóch wyników:

- tablicę identyfikatorów kategorii, do których należy dowcip lub
- nic (jeżeli żadne z pól wyboru nie zostało zaznaczone).

W drugim z tych przypadków nic nie musimy robić — żadna kategoria nie została wybrana, więc nie ma czego dodawać do tabeli `jokecategory`. Jeżeli natomiast tablica z identyfikatorami kategorii istnieje, używamy pętli `foreach`, aby wygenerować kwerendę `INSERT` dla każdego z nich. Używamy oczywiście przygotowanej instrukcji SQL:

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
if (isset($_POST['categories']))
{
  try
  {
```

```
      $sql = 'INSERT INTO jokecategory SET
          jokeid = :jokeid,
          categoryid = :categoryid';
      $s = $pdo->prepare($sql);

      foreach ($_POST['categories'] as $categoryid)
      {
        $s->bindValue(':jokeid', $jokeid);
        $s->bindValue(':categoryid', $categoryid);
        $s->execute();
      }
    }
    catch (PDOException $e)
    {
      $error = 'Błąd przy wiązaniu dowcipu z kategoriami.';
      include 'error.html.php';
      exit();
    }
  }

  header('Location: .');
  exit();
}
```

Zwróć uwagę na sposób pracy ze zmienną `$jokeid`, którą pozyskaliśmy dzięki wywołaniu metody `lastInsertId`.

Na tym kończy się temat dodawania dowcipów. Kod przetwarzania formularza przy modyfikowaniu wcześniejszych wpisów jest, jak można oczekiwać, podobny, choć zwracają uwagę dwie istotne różnice:

- Miejsce kwerendy `INSERT` wstawiającej dowcip do tabeli `joke` zajmuje kwerenda `UPDATE` (aktualizująca dane wpisu).
- Przed wstawieniem danych do tabeli `jokecategory` wszystkie wcześniejsze wpisy dotyczące modyfikowanego dowcipu zostają z niej usunięte.

Kolejny listing przedstawia całość kodu edycji. Uważnie go przeczytaj i upewnij się, że dokładnie rozumiesz działanie poszczególnych fragmentów.

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
if (isset($_GET['editform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  if ($_POST['author'] == '')
  {
    $error = 'Musisz wybrać autora tego dowcipu.
        Kliknij &lsquo;Wstecz&rsquo; i spróbuj ponownie.';
    include 'error.html.php';
    exit();
  }

  try
  {
    $sql = 'UPDATE joke SET
```

```
        joketext = :joketext,
        authorid = :authorid
        WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->bindValue(':joketext', $_POST['text']);
    $s->bindValue(':authorid', $_POST['author']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy aktualizowaniu dowcipu.';
    include 'error.html.php';
    exit();
  }

  try
  {
    $sql = 'DELETE FROM jokecategory WHERE jokeid = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy usuwaniu przestarzałych wpisów kategorii.';
    include 'error.html.php';
    exit();
  }

  if (isset($_POST['categories']))
  {
    try
    {
      $sql = 'INSERT INTO jokecategory SET
          jokeid = :jokeid,
          categoryid = :categoryid';
      $s = $pdo->prepare($sql);

      foreach ($_POST['categories'] as $categoryid)
      {
        $s->bindValue(':jokeid', $_POST['id']);
        $s->bindValue(':categoryid', $categoryid);
        $s->execute();
      }
    }
    catch (PDOException $e)
    {
      $error = 'Błąd przy wiązaniu dowcipu z kategoriami.';
      include 'error.html.php';
      exit();
    }
  }

  header('Location: .');
  exit();
}
```

Usuwanie dowcipów

Ostatnim elementem do zaimplementowania jest mechanizm usuwania dowcipów, czyli operacja inicjowana kliknięciem wyświetlanego obok żartów przycisku *Usuń*. Odpowiedzialny za to kod podąża ścieżką wytyczoną przy usuwaniu autorów i kategorii. Zawiera on tylko kilka drobnych dostosowań, z których najbardziej znaczącym jest usuwanie powiązanych z dowcipem wpisów w tabeli `jokecategory`.

Poniżej przedstawiona jest całość kodu. Nie ma w nim żadnych elementów, które nie byłyby wykorzystywane już wcześniej. Prześledź jego budowę i upewnij się, że wszystko dobrze rozumiesz.

kod w pliku chapter7/admin/jokes/index.php (fragment)

```
if (isset($_POST['action']) and $_POST['action'] == 'Usuń')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  // Usuń przypisania tego dowcipu do kategorii
  try
  {
    $sql = 'DELETE FROM jokecategory WHERE jokeid = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy usuwaniu przypisań dowcipu do kategorii.';
    include 'error.html.php';
    exit();
  }

  // Usuń dowcip
  try
  {
    $sql = 'DELETE FROM joke WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Błąd przy usuwaniu dowcipu.';
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}
```

Podsumowanie

Wciąż można wskazać kilka elementów, których naszemu prostemu systemowi zarządzania treścią brakuje. Nie przewiduje on na przykład wyświetlania listy dowcipów, które nie należą do *żadnej* kategorii. Byłoby to zapewne pożądane po wprowadzeniu do bazy znacznej liczby różnorodnych żartów. Brakuje też możliwości określania kryteriów sortowania wyświetlanych list. Aby zaimplementować takie mechanizmy, będziesz musiał poszerzyć nieco wiedzę z zakresu języka SQL. Zajmiemy się tym w rozdziale 11.

Kilka pominiętych drobiazgów

Jeżeli dokładnie przejrzysz archiwum kodu dla tego rozdziału, zauważysz, że zmieniłem też stronę listy dowcipów (w folderze *joke*). Usunąłem z niej łącza służące do dodawania i usuwania wpisów. Wcześniejsza implementacja tych mechanizmów nie była dostosowana do wprowadzonej w tym rozdziale rozbudowanej struktury bazy.

W rozdziale 9., w punkcie „Wyzwanie dla Ciebie — moderacja dowcipów", zaproponuję Ci samodzielne znalezienie sposobu eleganckiej obsługi dowcipów nadsyłanych przez użytkowników.

Jeżeli pominąć te drobne szczegóły, masz już przed sobą gotowy system, który pozwala osobie nieznającej SQL ani nawet zasad pracy z bazami danych z łatwością wykonywać podstawowe czynności administracyjne. W połączeniu ze stronami przeznaczonymi dla zwykłych użytkowników jest to rozwiązanie, które pozwala rozbudowywać i zmieniać treść witryny praktycznie bez żadnej wiedzy. Jeżeli wydaje Ci się, że mogłoby to zainteresować wiele firm chcących zaistnieć w sieci, to jest to znak, że dobrze zrozumiałeś cel tworzenia systemu zarządzania treścią.

Jest tylko jeden aspekt naszej przykładowej witryny, który wymusza na użytkownikach posiadanie jakiejkolwiek wiedzy (poza umiejętnością korzystania z przeglądarki WWW): formatowanie treści. Gdybyśmy mieli zezwolić administratorom na stosowanie we wpisywanych dowcipach oznaczeń formatowania, moglibyśmy zaprosić ich do wpisywania w formularzu *Nowy dowcip* elementów kodu HTML. Tak wzbogacona treść musiałaby być następnie wyświetlana bez żadnych modyfikacji (a nie przy użyciu funkcji `htmlout`).

Nie byłoby to jednak dobre rozwiązanie. Po pierwsze, zmuszałoby do wykluczenia możliwości nadsyłania dowcipów przez dowolne osoby. Wyświetlanie nieprzetworzonego tekstu bez pośrednictwa funkcji `htmlspecialchars` byłoby poważną luką w zabezpieczeniach.

Po drugie, jak pisałem na początku tej książki, jedną z najcenniejszych cech witryn opartych na bazach danych jest to, że można rozbudowywać ich treść bez żadnej wiedzy technicznej, w tym znajomości HTML. Wymaganie kodów HTML do podzielenia dowcipu na akapity lub wyróżnienia kilku wyrazów kursywą przekreślałoby zatem znaczną część korzyści uzyskiwanych z użycia PHP i MySQL.

W rozdziale 8. pokażę, jak wykorzystać mechanizmy PHP w celu ułatwienia użytkownikom formatowania treści bez znajomości HTML. Wrócimy też do formularza przesyłania nowego dowcipu oraz poznamy sposób bezpiecznego odbierania zawartości przekazywanej przez nieznane osoby.

Skorowidz

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

X

Z

Język PHP wciąż zyskuje na popularności. Kolejne wersje, ciągłe ulepszenia sprawiają, że w niektórych obszarach zaczyna on rywalizować z weteranami na rynku języków programowania. Ten trend widać również w ofertach pracy. Programiści biegle znający PHP są wręcz rozchwytywani na rynku! Warto dołączyć do tego grona. Jeśli chcesz tworzyć zaawansowane aplikacje i poznać PHP na wylot, trafiłeś na właściwą książkę!

Dzięki niej już wkrótce staniesz się prawdziwym ninja programowania w PHP. Przeprowadzi Cię ona przez wszystkie etapy poznawania języka: skonfigurujesz serwer WWW i MySQL oraz PHP. Następnie zaprojektujesz swoją pierwszą bazę i podłączysz się do niej, żeby zapisać i pobrać dane. W kolejnych rozdziałach przekonasz się, jak wyrażenia regularne mogą Ci pomóc w codziennej pracy, dlaczego należy korzystać z sesji oraz co możesz zapisać w *ciasteczkach*.

PHP i MySQL. Od nowicjusza do wojownika ninja jest genialnym przewodnikiem po PHP i MySQL. Pozwoli Ci przeobrazić się z laika w profesjonalistę.

ZAINWESTUJ W NOWĄ WIEDZĘ O JĘZYKU PHP!

Błyskawicznie opanuj:

- konfigurację serwera WWW i bazy danych MySQL
- pobieranie danych z bazy i zapisywanie ich do bazy
- sposoby wykorzystania sesji i plików cookies
- niuanse języka PHP

sitepoint

sięgnij po WIĘCEJ

KOD KORZYŚCI

Nr katalogowy: 14970

Helion

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

Informatyka w najlepszym wydaniu

Cena 59,00 zł

ISBN 978-83-246-7110-6

9 788324 671106