O'REILLY®

Zawiera informacje na temat HTML5, CSS3 i jQuery

Wydanie IV

PHP, MySQL i JavaScript

Wprowadzenie

PRZEWODNIK TWÓRCY STRON I APLIKACJI SIECIOWYCH!

Helion

Robin Nixon

Tytuł oryginału: Learning PHP, MySQL & JavaScript, 4th Edition

Tłumaczenie: Piotr Cieślak

ISBN: 978-83-283-0842-8

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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/phmyj4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/phmyj4.zip

Printed in Poland.

Spis treści

ROZDZIAŁ 10.

Korzystanie z MySQL za pośrednictwem PHP

Jeśli zapoznałeś się z poprzednimi rozdziałami, to powinieneś już sprawnie posługiwać się MySQL i PHP. W tym rozdziale dowiesz się, w jaki sposób połączyć te dwie technologie za pomocą wbudowanych funkcji PHP umożliwiających bezpośredni dostęp do MySQL.

Tworzenie zapytań do bazy MySQL za pośrednictwem PHP

Zasadniczym powodem stosowania PHP jako interfejsu dla MySQL jest możliwość sformatowania wyników zwróconych przez zapytania SQL w sposób pozwalający na wyświetlenie ich na stronie internetowej. Jeśli możesz się zalogować do MySQL za pomocą nazwy użytkownika i hasła, to możesz to zrobić także poprzez PHP.

Różnica polega na tym, że zamiast korzystać z wiersza poleceń MySQL do wprowadzania instrukcji i wyświetlania wyników, będziesz tworzył zapytania w postaci łańcuchów znaków, przekazywane do MySQL. Wynik zwrócony przez bazę nie zostanie sformatowany tak jak w przypadku pracy z wierszem poleceń, lecz będzie miał postać pewnej struktury, którą PHP potrafi zinterpretować. Za pomocą instrukcji PHP można odczytywać dane z tej struktury i wyświetlać je na stronie WWW.

W tym rozdziale z poprzednich wydań książki opisywałem stary sposób dostępu do bazy danych MySQL, wykorzystujący funkcję `mysql`; dopiero w kolejnym rozdziale omawiałem nowsze rozszerzenie o nazwie `mysqli`. Ale jak to mówią, czas nie stoi w miejscu; liczba systemów ze starszymi wersjami kodu i oprogramowania powinna być już relatywnie mała, w tym wydaniu postanowiłem więc od razu przystąpić do omawiania nowszego rozszerzenia — które obecnie stało się już praktycznie standardem.

Proces

Proces komunikacji z MySQL za pomocą PHP wygląda tak:

1. nawiązanie połączenia z MySQL i wybór bazy danych,

2. utworzenie zapytania,
3. wykonanie zapytania,
4. pozyskanie rezultatów i wyświetlenie ich na stronie internetowej,
5. powtórzenie kroków od 2. do 4. aż do uzyskania wszystkich potrzebnych danych,
6. rozłączenie z MySQL.

Już za chwilę zajmiemy się tymi punktami krok po kroku, przede wszystkim jednak powinieneś skonfigurować procedurę logowania w bezpieczny sposób, by utrudnić postronnym osobom przechwycenie danych dostępowych do bazy.

Tworzenie pliku logowania

Większość stron internetowych napisanych w PHP składa się z wielu skryptów wymagających dostępu do MySQL, a tym samym — loginu oraz hasła. Mając to na uwadze, dobrze jest utworzyć osobny plik zawierający te informacje i dołączać go do plików strony za każdym razem, gdy będzie to konieczne. Przykład 10.1 przedstawia tego rodzaju plik, który nazwałem *login.php*.

Przykład 10.1. Plik login.php

```
<?php // login.php
  $hn = 'localhost';
  $db = 'publications';
  $un = 'użytkownik';
  $pw = 'hasło';
?>
```

Przepisz podany przykład, zastępując tymczasowe wartości, takie jak *użytkownik* i *hasło*, rzeczywistymi parametrami dostępowymi do bazy MySQL, a następnie zapisz plik w folderze z projektem, który skonfigurowałeś w rozdziale 2. Już za chwilę z niego skorzystamy.

Nazwa hosta `localhost` powinna być poprawna, jeśli pracujesz na MySQL zainstalowanym na lokalnym komputerze, a jeżeli wykonywałeś ćwiczenia opisane w poprzednich rozdziałach, to na serwerze powinna się znajdować baza danych o nazwie *publications*.

Znaczniki `<?php` oraz `?>` są bardzo ważne zwłaszcza w przypadku pliku *login.php* przedstawionego w przykładzie 10.1, bo wiersze pomiędzy nimi mogą być zinterpretowane *wyłącznie* jako kod PHP. Jeśli byś je pominął, a ktoś odwołałby się do tego pliku bezpośrednio na Twojej stronie internetowej, to zostałby on po prostu wyświetlony w przeglądarce i zdradził Twoje sekrety. Dzięki zastosowaniu znaczników intruz zobaczy tylko pustą stronę. Tak przygotowany plik będzie można poprawnie dołączać do innych plików PHP.

Zmienna `$hn` informuje PHP, do jakiego serwera ma się odwołać przy połączeniu z bazą danych. To konieczne, gdyż za pośrednictwem PHP możesz nawiązać połączenie z dowolną bazą danych MySQL dostępną z poziomu Twojego systemu, w tym także z serwerami w internecie. Na potrzeby przykładów opisanych w tym rozdziale użyjemy jednak lokalnego serwera i zamiast podawać pełną nazwę domeny, taką jak *mysql.mojserwer.com*, wpiszemy po prostu `localhost` (lub adres IP `127.0.0.1`).

W zmiennej `$db` podajemy nazwę bazy, której będziemy używać — w tym przypadku będzie to baza *publications*, którą zapewne utworzyłeś w rozdziale 8., bądź baza udostępniona przez administratora serwera (wówczas musisz oczywiście odpowiednio zmodyfikować plik *login.php*).

Kolejną zaletą umieszczenia danych logowania w jednym miejscu jest łatwość aktualizacji projektu przy zmianie hasła: możesz to robić tak często, jak tylko masz ochotę, bo niezależnie od liczby plików PHP odwołujących się do MySQL będziesz musiał wprowadzić poprawkę tylko w jednym miejscu.

Nawiązywanie połączenia z MySQL

Po zapisaniu pliku *login.php* można go dołączyć za pomocą dyrektywy `require_once` do dowolnego pliku z kodem PHP, który będzie wymagał dostępu do bazy danych. Takie rozwiązanie jest o tyle lepsze od użycia instrukcji `include`, że nieodnalezienie pliku z danymi logowania spowoduje wyświetlenie komunikatu o krytycznym błędzie. A wierz mi, brak pliku z danymi logowania do bazy *jest* krytycznym błędem.

Ponadto zastosowanie dyrektywy `require_once` zamiast `require` oznacza, że plik zostanie wczytany tylko wtedy, jeśli nie został dołączony już wcześniej, co pozwala zapobiec marnowaniu zasobów na niepotrzebne odwołania do dysku twardego. Przykład 10.2 przedstawia niezbędny fragment kodu.

Przykład 10.2. Nawiązywanie połączenia z serwerem MySQL za pośrednictwem mysqli

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
?>
```

W tym przykładzie został utworzony nowy obiekt o nazwie `$conn` poprzez wywołanie metody `mysqli`, do której zostały przekazane argumenty zaczerpnięte z pliku *login.php*. Za obsługę błędów odpowiada odwołanie do właściwości `$conn->connect_error`.

Funkcja `die` oddaje nieocenione usługi podczas programowania w PHP, ale na serwerze produkcyjnym warto przygotować specjalne komunikaty o błędach, które będą znacznie przyjaźniejsze w odbiorze. W takich przypadkach nie przerywa się działania programu PHP, ale odpowiednio formatuje komunikat i wyświetla go po zwykłym zakończeniu pracy programu, na przykład tak:

```
function mysql_fatal_error($msg)
{
    $msg2 = mysql_error();
    echo <<< _END
Niestety nie udało się zrealizować zadania.
Komunikat błędu:

    <p>$msg: $msg2</p>

Kliknij przycisk Wstecz w przeglądarce i spróbuj ponownie.
W razie dalszych problemów prosimy o wysłanie
<a href="mailto:admin@serwer.com">maila do administratora</a>.
Dziękujemy.
_END;
}
```

Operator `->` oznacza, że element po jego prawej stronie jest właściwością albo metodą obiektu znajdującego się po stronie lewej. Gdyby w tym przypadku właściwość `connect_error` miała

jakąś wartość, oznaczałoby to wystąpienie błędu. Wywołujemy więc funkcję `die`, która wyświetli tę wartość — zawiera ona informacje o napotkanym błędzie.

Obiekt `$conn` będzie używany w kolejnych przykładach do uzyskania dostępu do bazy MySQL.

Konstruowanie i wykonywanie zapytania

Aby wysłać zapytanie do MySQL przy użyciu PHP, wystarczy użyć metody `query` obiektu połączenia. Przykład 10.3 ilustruje, jak to zrobić.

Przykład 10.3. Wysyłanie zapytania do bazy za pośrednictwem mysqli

```
<?php
  $query  = "SELECT * FROM classics";
  $result = $conn->query($query);
  if (!$result) die($conn->error);
?>
```

W tym przypadku zmiennej `$query` został przypisany łańcuch tekstowy zawierający zapytanie do wykonania. Zmienna ta została następnie przekazana do metody `query` obiektu `$conn`. Z kolei metoda ta zwraca rezultat, który trafia do obiektu `$result`. Jeśli wartość `$result` wynosi `FALSE`, to znaczy, że wystąpił jakiś błąd, którego opis został zapisany we właściwości `error` obiektu połączenia. W takim przypadku wywołana zostaje funkcja `die`, która ten błąd wyświetli.

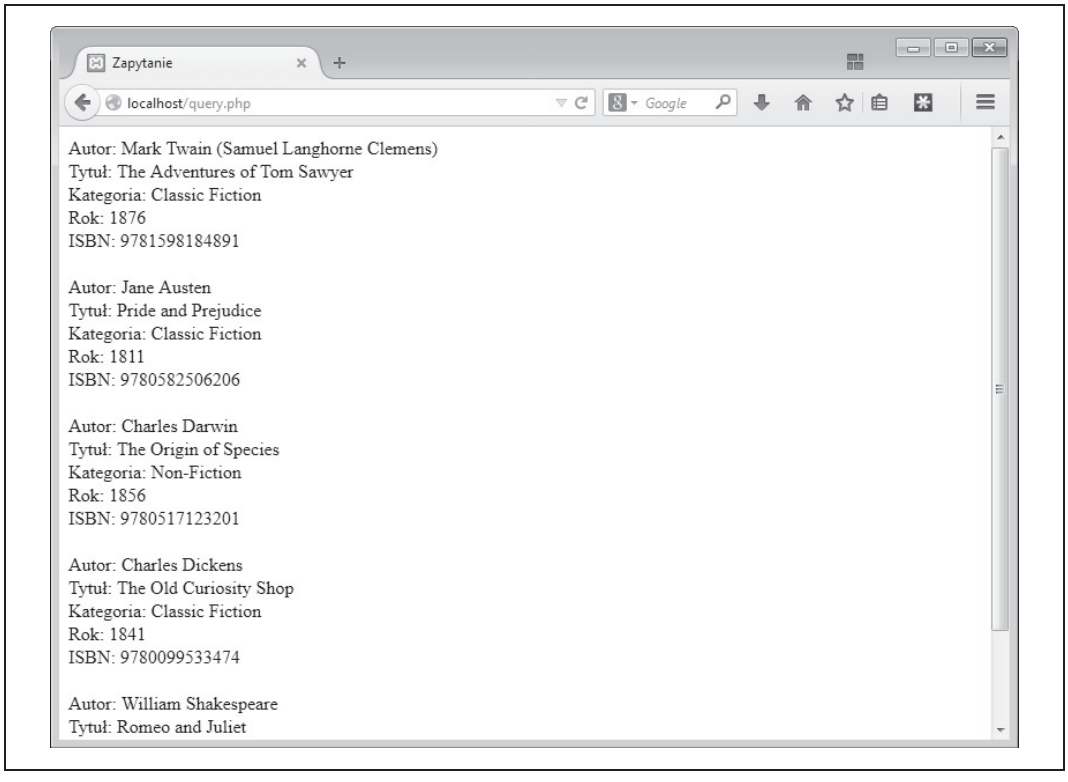
Wszystkie dane zwrócone przez MySQL trafią do obiektu `$result` w postaci, w której można je łatwo przetwarzać.

Pobieranie rezultatu

Po zwróceniu rezultatu do obiektu `$result` możesz pobrać z niego potrzebne dane po kolei przy użyciu metody `fetch_assoc` tego obiektu. Przykład 10.4 stanowi połączenie i rozszerzenie dotychczasowych przykładów. Jest to już kompletny program, który możesz przepisać i uruchomić, aby pozyskać z bazy potrzebne dane (rysunek 10.1). Zalecam zapisanie go pod nazwą *query.php* w tym samym folderze co plik *login.php* (albo wykorzystanie pliku pobranego z darmowego archiwum przykładów, dostępnego na stronie *lpmj.net*).

Przykład 10.4. Zwracanie zawartości poszczególnych komórek

```
<?php // query.php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  $query  = "SELECT * FROM classics";
  $result = $conn->query($query);
  if (!$result) die($conn->error);
  $rows = $result->num_rows;
  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $result->data_seek($j);
    echo 'Autor: '     . $result->fetch_assoc()['author']   . '<br>';
    $result->data_seek($j);
    echo 'Tytuł: '     . $result->fetch_assoc()['title']    . '<br>';
    $result->data_seek($j);
    echo 'Kategoria: ' . $result->fetch_assoc()['category'] . '<br>';
    $result->data_seek($j);
    echo 'Rok: '       . $result->fetch_assoc()['year']     . '<br>';
```



Rysunek 10.1. Rezultat działania programu query.php z przykładu 10.4

```
    $result->data_seek($j);
    echo 'ISBN: '      . $result->fetch_assoc()['isbn']     . '<br><br>';
  }
  $result->close();
  $conn->close();
?>
```

W opisanym przykładzie w celu odwołania się do odpowiedniego wiersza danych przy każdej iteracji pętli użyliśmy metody `data_seek` obiektu `$result`. Następnie wykorzystaliśmy metodę `fetch_assoc` do pobrania wartości przechowywanych w poszczególnych komórkach i wreszcie wartości te wyświetliliśmy przy użyciu instrukcji `echo`.

Zapewne zgodzisz się ze mną, że takie wyszukiwanie danych jest dość uciążliwe i powinna być jakaś skuteczniejsza metoda na osiągnięcie podobnego efektu. Taki sposób rzeczywiście istnieje i polega na pobieraniu całych wierszy danych naraz.

W rozdziale 9. przeczytałeś o pierwszej, drugiej i trzeciej postaci normalnej bazy danych i być może zwróciłeś uwagę na fakt, że tabela *classics* nie spełnia warunków tych postaci, ponieważ zawiera ona informacje o autorze i o książce. Wynika to z faktu, że tabela *classics* została utworzona jeszcze przed omówieniem kwestii normalizacji. Jednak na potrzeby zilustrowania metod dostępu do MySQL z poziomu PHP ta tabela zupełnie wystarczy i pozwoli nam uniknąć żmudnego wpisywania kolejnych danych — proponuję więc się nią posłużyć.

Pobieranie wiersza danych

Aby pobrać cały wiersz danych naraz, należy zastąpić pętlę `for` z przykładu 10.4 taką, jaka została wyróżniona pogrubieniem w przykładzie 10.5. Po wykonaniu tej zmiany powinieneś uzyskać taki sam efekt, jaki został pokazany wcześniej na rysunku 10.1. Tak zmodyfikowany plik możesz zapisać pod nazwą *fetchrow.php*.

Przykład 10.5. Odczytywanie zawartości kolejnych wierszy

```
<?php //fetchrow.php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  $query  = "SELECT * FROM classics";
  $result = $conn->query($query);
  if (!$result) die($conn->error);
  $rows = $result->num_rows;
  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $result->data_seek($j);
    $row = $result->fetch_array(MYSQLI_ASSOC);
    echo 'Autor: '     . $row['author']   . '<br>';
    echo 'Tytuł: '     . $row['title']    . '<br>';
    echo 'Kategoria: ' . $row['category'] . '<br>';
    echo 'Rok: '       . $row['year']     . '<br>';
    echo 'ISBN: '      . $row['isbn']     . '<br><br>';
  }
  $result->close();
  $conn->close();
?>
```

W tak zmodyfikowanym przykładzie mamy do czynienia z pięciokrotnie mniejszą liczbą odwołań do obiektu `$result` (w porównaniu z poprzednim kodem), a w ramach każdej iteracji pętli następuje tylko jedno takie odwołanie, ponieważ przy użyciu metody `fetch_array` są pobierane całe wiersze kodu. Metoda ta zwraca cały wiersz danych w postaci tablicy, która w naszym programie jest następnie przypisywana do zmiennej `$row`.

Zależnie od przekazanych do niej wartości metoda `fetch_array` może zwrócić trzy rodzaje tablic:

`MYSQLI_NUM`
: Tablica numeryczna. Poszczególne kolumny pojawiają się w tablicy zgodnie z kolejnością, w jakiej zostały utworzone w tabeli (z uwzględnieniem późniejszych zmian). W naszym przypadku na zerowej pozycji tablicy znajduje się kolumna *Author*, na pierwszej kolumna *Title* i tak dalej.

`MYSQLI_ASSOC`
: Tablica asocjacyjna. Każdy klucz stanowi nazwę kolumny. Ponieważ w tym przypadku do danych trzeba się odwoływać za pośrednictwem nazwy kolumny (a nie numeru indeksu), warto korzystać z tego wariantu zawsze, gdy to możliwe, aby ułatwić sobie debugowanie programu, a innym programistom — interpretację kodu.

`MYSQLI_BOTH`
: Tablica asocjacyjna i numeryczna.

Tablice asocjacyjne są zwykle bardziej praktyczne od numerycznych, gdyż umożliwiają odwoływanie się do kolumn za pomocą nazw, na przykład `$row['author']`, dzięki czemu nie trzeba

zapamiętywać, na którym miejscu w tabeli znajduje się potrzebna kolumna. W przykładowym skrypcie została użyta tablica asocjacyjna, o czym świadczy argument `MYSQLI_ASSOC`.

Zamykanie połączenia

Po zakończeniu wykonywania skryptu PHP zwolni pamięć zaalokowaną na potrzeby obiektów, więc w przypadku niewielkich programów na ogół nie trzeba się troszczyć o samodzielne zarządzanie pamięcią. Jednak w przypadku większej liczby rezultatów albo dużych porcji danych dobrze jest zwolnić pamięć, której program już nie potrzebuje, aby uniknąć problemów z jego dalszym działaniem.

Jest to szczególnie istotne na często odwiedzanych stronach, gdyż ilość pamięci zajmowanej w trakcie sesji może raptownie rosnąć. Zwróć uwagę na metody `close` dla obiektów `$result` oraz `$conn`, które w poprzednich skryptach były wywoływane w sytuacji, gdy dany obiekt przestał już być potrzebny.

```
$result->close();
$conn->close();
```

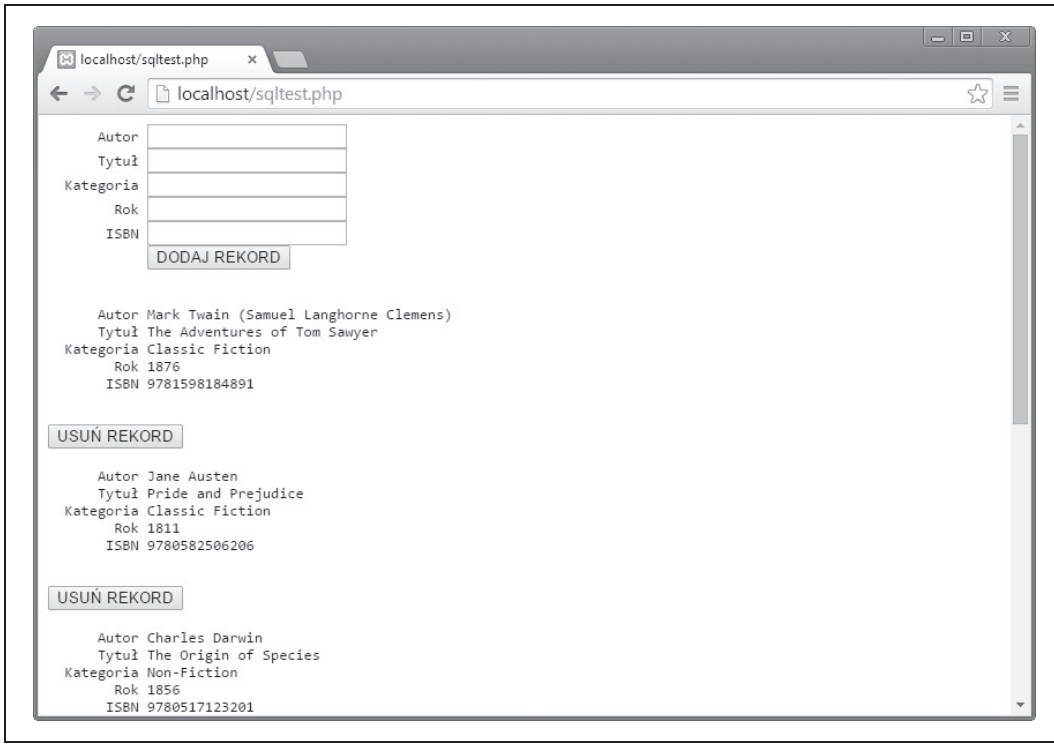
Najlepiej byłoby zamknąć każdy obiekt z wynikami zapytania, gdy przestanie być używany, a następnie zamknąć obiekt połączenia w chwili, gdy komunikacja z serwerem MySQL stanie się zbędna. Takie rozwiązanie gwarantuje zwolnienie zasobów do systemu tak szybko, jak to możliwe, to zaś przekłada się na płynne działanie MySQL i eliminuje wszelkie wątpliwości co do tego, czy PHP sam zwróci zajętą pamięć, zanim będzie ponownie potrzebna.

Praktyczny przykład

Pora na napisanie pierwszego programu, który za pomocą PHP będzie umieszczał dane w tabeli MySQL i usuwał je stamtąd. Sugeruję, abyś przykład 10.6 zapisał w roboczym folderze z dokumentami WWW pod nazwą *sqltest.php*. Przykład działania programu został zilustrowany na rysunku 10.2.

Przykład 10.6. Wstawianie i usuwanie danych przy użyciu programu sqltest.php

```
<?php // sqltest.php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  if (isset($_POST['delete']) && isset($_POST['isbn']))
  {
    $isbn   = get_post($conn, 'isbn');
    $query  = "DELETE FROM classics WHERE isbn='$isbn'";
    $result = $conn->query($query);
    if (!$result) echo "Instrukcja DELETE nie powiodła się: $query<br>" .
      $conn->error . "<br><br>";
  }
  if (isset($_POST['author'])   &&
      isset($_POST['title'])    &&
      isset($_POST['category']) &&
      isset($_POST['year'])     &&
      isset($_POST['isbn']))
  {
    $author   = get_post($conn, 'author');
```



Rysunek 10.2. Rezultat działania programu sqltest.php z przykładu 10.8

```
    $title    = get_post($conn, 'title');
    $category = get_post($conn, 'category');
    $year     = get_post($conn, 'year');
    $isbn     = get_post($conn, 'isbn');
    $query    = "INSERT INTO classics VALUES" .
      "('$author', '$title', '$category', '$year', '$isbn')";
    $result   = $conn->query($query);
    if (!$result) echo "Instrukcja INSERT nie powiodła się: $query<br>" .
      $conn->error . "<br><br>";
  }
  echo <<<_END
  <form action="sqltest.php" method="post"><pre>
      Autor <input type="text" name="author">
      Tytuł <input type="text" name="title">
  Kategoria <input type="text" name="category">
        Rok <input type="text" name="year">
       ISBN <input type="text" name="isbn">
            <input type="submit" value="DODAJ REKORD">
  </pre></form>
_END;
  $query  = "SELECT * FROM classics";  $result = $conn->query($query);
  if (!$result) die ("Brak dostępu do bazy danych: " . $conn->error);
  $rows = $result->num_rows;
  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $result->data_seek($j);
    $row = $result->fetch_array(MYSQLI_NUM);
```

```
    echo <<<_END
  <pre>
      Autor $row[0]
      Tytuł $row[1]
  Kategoria $row[2]
        Rok $row[3]
       ISBN $row[4]
  </pre>
  <form action="sqltest.php" method="post">
  <input type="hidden" name="delete" value="yes">
  <input type="hidden" name="isbn" value="$row[4]">
  <input type="submit" value="USUŃ REKORD"></form>
_END;
  }
  $result->close();
  $conn->close();
  function get_post($conn, $var)
  {
    return $conn->real_escape_string($_POST[$var]);
  }
?>
```

W przykładzie 10.6 został użyty typowy formularz HTML. W rozdziale 11. zapoznasz się ze szczegółowymi informacjami na temat formularzy — tymczasem przyjmij, że uznałem ich znajomość za coś oczywistego, i skup się przede wszystkim na komunikacji z bazą danych.

Składający się z ponad 80 linii program może sprawiać wrażenie skomplikowanego, ale nie obawiaj się — większość kodu poznałeś już w przykładzie 10.5, a jego działanie jest stosunkowo proste.

Najpierw program sprawdza, czy użytkownik podjął jakieś działania, a następnie — zgodnie z nimi — umieszcza w tabeli *classics* w bazie *publications* nowy rekord albo usuwa jeden z istniejących. Niezależnie od akcji podjętej przez użytkownika program wyświetla w przeglądarce wszystkie aktualne wiersze tabeli. Przeanalizujmy działanie programu.

Pierwsza część nowego kodu rozpoczyna się od wywołania funkcji `isset`, która sprawdza, czy wypełnione zostały wszystkie pola umożliwiające dodanie nowego rekordu lub usunięcie go. Jeśli tak, to w każdym z wierszy w ramach instrukcji `if` jest wywoływana funkcja `get_post`, która została zdefiniowana na końcu programu. Funkcja ta pełni prostą, ale bardzo ważną funkcję: zbiera i przesyła dane z przeglądarki.

Tablica $_POST

W jednym z poprzednich rozdziałów wspomniałem, że dane wprowadzone przez użytkownika w przeglądarce są wysyłane za pośrednictwem metody `GET` albo `POST`. Na ogół lepsza jest metoda `POST`, której tutaj użyliśmy (ponieważ pozwala uniknąć wyświetlania niepożądanych danych na pasku adresu przeglądarki). Serwer WWW gromadzi wszystkie dane podane przez użytkownika (nawet jeśli mowa o formularzu składającym się ze stu pól) i umieszcza je w tablicy o nazwie `$_POST`.

Tablica `$_POST` ma charakter asocjacyjny — z tego rodzaju tablicami zetknąłeś się już w rozdziale 6. W zależności od tego, czy formularz został skonfigurowany z użyciem metody `POST`,

czy GET, dane z niego trafią do tablicy asocjacyjnej $_POST albo $_GET. Informacje zawarte w obydwu mogą być odczytywane w identyczny sposób.

Każdemu polu formularza jest przypisywany element tablicy o nazwie zgodnej z nazwą tego pola. To oznacza, że jeśli w formularzu znajduje się pole o nazwie isbn, w tablicy $_POST pojawi się element, którego klucz będzie nosił nazwę isbn. Program w PHP może odczytać zawartość takiego pola poprzez odwołanie w postaci $_POST['isbn'] albo $_POST["isbn"] (w tym przypadku pojedynczy i podwójny cudzysłów będzie miał ten sam efekt).

Jeśli składnia $_POST wydaje Ci się skomplikowana, możesz po prostu skorzystać z rozwiązania przedstawionego w przykładzie 10.6: skopiuj dane wprowadzone przez użytkownika do innych zmiennych, a potem możesz zapomnieć o tablicy $_POST. To rozwiązanie jest często stosowane w programach PHP — na samym początku pobiera się dane z wszystkich pól tablicy $_POST, a potem ją ignoruje.

Nie ma powodu, by modyfikować elementy tablicy $_POST. Jej rola ogranicza się do wymiany informacji między przeglądarką a programem i z tego względu lepiej najpierw przenieść zawarte w niej dane do własnych zmiennych.

Wróćmy do wspomnianej funkcji get_post: zauważ, że każda informacja jest najpierw poddawana działaniu funkcji real_escape_string w celu usunięcia dowolnych znaków, które haker mógłby wykorzystać do włamania się do bazy lub zmodyfikowania jej zawartości.

```
function get_post($conn, $var)
{
  return $conn->real_escape_string($_POST[$var]);
}
```

Usuwanie rekordu

Przed sprawdzeniem, czy użytkownik wpisał nowe dane do wprowadzenia do bazy, program weryfikuje wartość zmiennej $_POST['delete']. Jeśli taka wartość istnieje, to znaczy, że użytkownik kliknął przycisk *USUŃ REKORD*, aby usunąć jeden z istniejących rekordów z bazy. W takim przypadku przekazywana jest ponadto wartość zmiennej $isbn.

Jak zapewne pamiętasz, numer ISBN jednoznacznie identyfikuje każdy rekord. Formularz HTML uwzględnia ten numer w zawartym w zmiennej $query zapytaniu DELETE FROM, które jest następnie przekazywane do metody query obiektu $conn, skąd trafia do MySQL.

Jeśli zmienna $_POST['delete'] nie ma wartości (co oznacza, że nie trzeba usuwać danych z bazy), sprawdzana jest zawartość zmiennej $_POST['author'] i pozostałych. Jeśli wszystkim została przypisana wartość, to do zmiennej $query trafia zapytanie w postaci INSERT INTO z pięcioma danymi do umieszczenia w bazie. Zmienna ta jest następnie przekazywana do metody query, która może zwrócić wartość TRUE albo FALSE. W przypadku wartości FALSE generowany jest komunikat błędu, który trafia do właściwości error obiektu $conn i może zostać wyświetlony na przykład tak:

```
if (!$result) echo "Instrukcja INSERT nie powiodła się: $query<br>" .
  $conn->error . "<br><br>";
```

Wyświetlanie formularza

Następna część kodu odpowiada za wyświetlenie niewielkiego formularza, widocznego w górnej części rysunku 10.2. Zapewne pamiętasz z poprzednich rozdziałów konstrukcję `echo <<<_END…_END`, która powoduje wyświetlenie w przeglądarce wszystkiego, co znajduje się między znacznikami `_END`.

Zamiast używać instrukcji `echo`, można byłoby przerwać program PHP za pomocą znacznika `?>`, umieścić w pliku niezbędny fragment dokumentu HTML, a potem wstawić znacznik `<?`, aby wrócić do kodu PHP. Obrana metoda zależy tylko od preferencji programisty, ja jednak zawsze zalecam zrealizowanie całej operacji w kodzie PHP. Powody ku temu mam następujące:

- Umieszczanie tylko kodu PHP w plikach *.php* jest korzystne podczas debugowania (także przez innych programistów). Nie ma wówczas potrzeby szukać wydzielonych fragmentów HTML.
- Jeśli chciałbyś wyświetlić wartość zmiennej PHP w obrębie kodu HTML, możesz po prostu wpisać jej nazwę. Jeśli zamknąłbyś blok kodu PHP, musiałbyś najpierw zainicjować nowy blok, wyświetlić zmienną, a potem wrócić do HTML.

Fragment kodu HTML z formularzem po prostu przekierowuje jego działanie na plik *sqltest.php*. To oznacza, że po wysłaniu formularza zawartość jego pól trafia z powrotem do pliku *sqltest.php*, czyli naszego programu. Formularz został ponadto skonfigurowany tak, by pola były wysyłane metodą `POST`, a nie `GET`. Ten wybór jest podyktowany faktem, że w przypadku metody `GET` dane są dołączane do przesyłanego adresu URL, co może powodować mały bałagan w pasku adresu przeglądarki. Poza tym metoda `GET` umożliwia łatwą ingerencję w przesyłane dane, a tym samym pozwala na podejmowanie prób włamania się na serwer. Z tego względu zawsze gdy tylko jest to możliwe, lepiej używać metody `POST`, która ma zarazem tę zaletę, że ukrywa wysyłane dane.

Poniżej pól formularza HTML znajduje się przycisk wysyłania danych z napisem *DODAJ REKORD*. Zwróć uwagę na zastosowanie znaczników `<pre>` i `</pre>`, które zostały użyte w celu wymuszenia fontu o stałej szerokości znaków, co pozwala na eleganckie wyrównanie poszczególnych wpisów w formularzu. Ponadto w obrębie znaczników `<pre>` są uwzględniane znaki końca linii.

Wysyłanie zapytań do bazy danych

Następnie kod wraca do znajomego „terytorium” z przykładu 10.5 — do bazy MySQL jest wysyłane zapytanie o wszystkie rekordy z tabeli *classics*:

```
$query  = "SELECT * FROM classics";
$result = $conn->query($query);
```

W dalszej kolejności zmiennej `$rows` jest przypisywana wartość odzwierciedlająca liczbę wierszy w tabeli:

```
$rows = $result->num_rows;
```

Na podstawie wartości zmiennej $rows jest inicjalizowana pętla for, która służy do wyświetlania zawartości poszczególnych wierszy. W każdej iteracji tej pętli wywoływana jest metoda data_seek obiektu $result, która odczytuje porcję interesujących nas danych:

```
$result->data_seek($j);
```

Następnie tablica $row jest wypełniana poszczególnymi wierszami danych przy użyciu metody fetch_array obiektu $result. Do metody jest przekazywana stała MYSQLI_NUM, która wymusza zwrócenie tablicy numerycznej (a nie asocjacyjnej):

```
$row = $result->fetch_array(MYSQLI_NUM);
```

Po umieszczeniu danych w tablicy $row ich wyświetlenie jest już proste. Użyłem w tym celu konstrukcji heredoc z instrukcjami echo i z zastosowaniem znaczników <pre>, które umożliwiają estetyczne wyrównanie poszczególnych wierszy.

Po każdym wyświetlonym rekordzie mamy do czynienia z kolejnym formularzem, który również przekierowuje użytkownika do pliku *sqltest.php* (czyli do tego samego programu). Ten formularz zawiera dwa ukryte pola: delete oraz isbn. Pole delete ma wartość yes, zaś wartość pola isbn jest określana na podstawie zawartości pola tablicy $row[4], w którym jest przechowywany numer ISBN dla danego rekordu.

Formularz kończy przycisk wysyłania danych z napisem *USUŃ REKORD*. Następnie nawias klamrowy zamyka pętlę, która będzie powtarzana aż do wyświetlenia wszystkich rekordów. Gdy tak się stanie, wywoływane są metody close obiektów $result oraz $conn w celu zamknięcia połączenia i zwolnienia zasobów dla PHP.

```
$result->close();
$conn->close();
```

Wreszcie na końcu kodu znajduje się definicja funkcji get_post, której już wcześniej się przyglądaliśmy. I to już koniec struktury naszego pierwszego programu PHP umożliwiającego komunikację z bazą danych. Przekonajmy się, co on potrafi.

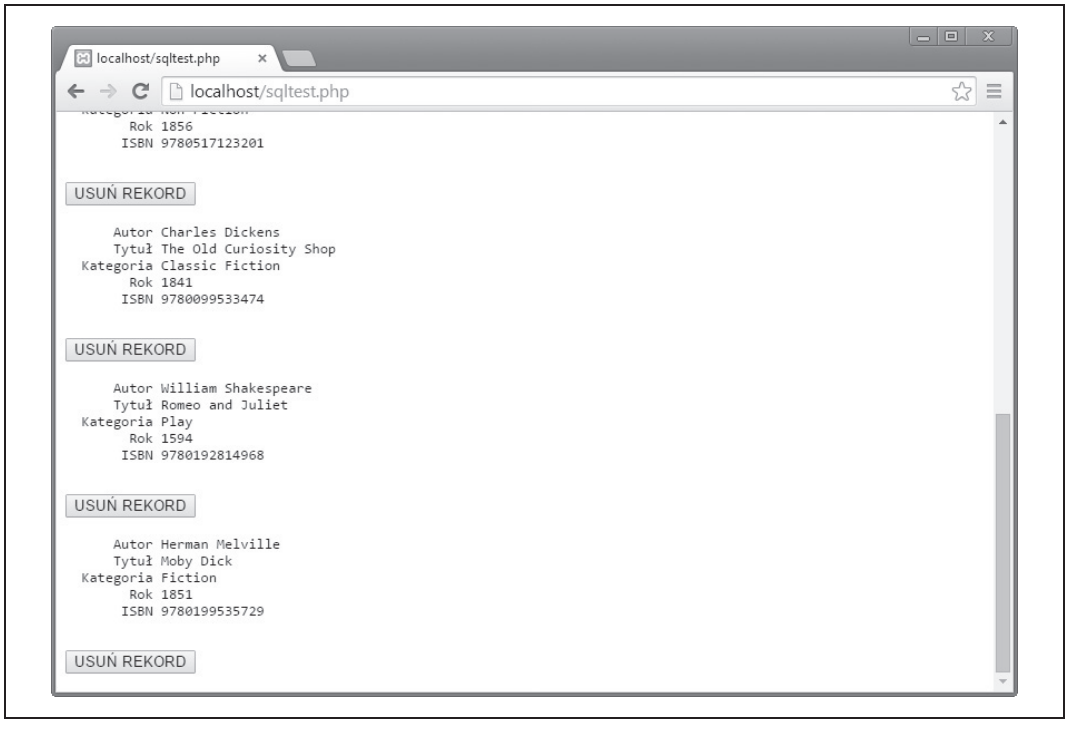
Po wprowadzeniu kodu (i poprawieniu ewentualnych pomyłek) spróbuj wprowadzić następujące informacje do kolejnych pól formularza, aby dodać do bazy nowy rekord poświęcony książce *Moby Dick*:

```
Herman Melville
Moby Dick
Fiction
1851
9780199535729
```

Działanie programu

Po wysłaniu powyższych danych do bazy za pomocą przycisku *DODAJ REKORD* przewiń zawartość strony w dół, aby wyświetlić nowy rekord. Efekt powinien wyglądać podobnie jak na rysunku 10.3.

Przekonajmy się teraz, jak działa usuwanie danych z bazy, na podstawie przykładowego rekordu, który wpiszemy tylko w tym celu. Wprowadź cyfrę 1 w każdym z pięciu pól z informacjami o nowej książce i kliknij przycisk *DODAJ REKORD*. Jeśli teraz przewiniesz zawartość okna w dół, zobaczysz nowy wpis składający się z samych jedynek. Oczywiście taki rekord nie jest nam w tabeli do niczego potrzebny, kliknij więc znajdujący się pod nim przycisk *USUŃ*



Rysunek 10.3. Po dodaniu do bazy książki „Moby Dick"

REKORD i ponownie przewiń zawartość okna w dół, aby się przekonać, że rekord rzeczywiście został usunięty.

Przy założeniu, że wszystko przebiegło zgodnie z planem, za pomocą opisanego programu możesz dodawać i usuwać rekordy z bazy. Wykonaj kilka tego typu operacji, ale zostaw w tabeli dotychczasowe informacje o książkach (w tym rekord z książką *Moby Dick*), bo będziemy ich jeszcze potrzebować. Spróbuj dwukrotnie dodać rekord zawierający same jedynki — przy drugiej próbie pojawi się komunikat błędu wynikający z faktu, że w bazie istnieje już książka o ISBN wynoszącym 1.

MySQL w praktyce

Masz już wszystkie wiadomości niezbędne do zapoznania się z kilkoma praktycznymi metodami komunikacji z bazą danych za pośrednictwem PHP, umożliwiającymi między innymi tworzenie i usuwanie tabel, wstawianie, aktualizowanie i kasowanie rekordów oraz zabezpieczanie bazy danych oraz strony internetowej przed złośliwymi atakami. W kolejnych przykładach przyjąłem, że dysponujesz plikiem *login.php*, o którym wspominałem wcześniej w tym rozdziale.

Tworzenie tabeli

Przypuśćmy, że pracujesz w ogrodzie zoologicznym i chciałbyś utworzyć bazę danych z informacjami o wszystkich gatunkach kotowatych, które w nim zamieszkują. Wiesz, że istnieje

dziewięć *rodzin* kotów — lwy, tygrysy, jaguary, pantery, pumy, gepardy, rysie, karakale i koty domowe. Aby je ująć w bazie, potrzebna będzie osobna kolumna. Każdy kot ma jakieś *imię*, które powinno trafić do kolejnej kolumny; powinieneś też zebrać informacje o ich *wieku* — czyli potrzebna będzie jeszcze jedna. Oczywiście na dalszym etapie rozwoju bazy niezbędne okazałyby się jeszcze inne kolumny, na przykład z informacjami dotyczącymi wymogów żywieniowych, szczepień itp., ale na razie nie są nam potrzebne. Ponadto każde zwierzę będzie wymagało unikatowego identyfikatora, to zaś oznacza, że trzeba będzie utworzyć jeszcze jedną kolumnę o nazwie *id*.

Przykład 10.7 przedstawia kod tabeli MySQL, w której można przechowywać tego rodzaju dane. Główne zapytanie zostało wyróżnione pogrubieniem.

Przykład 10.7. Tworzenie tabeli o nazwie koty

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);

  $query = "CREATE TABLE koty (
    id SMALLINT NOT NULL AUTO_INCREMENT,
    rodzina VARCHAR(32) NOT NULL,
    imie VARCHAR(32) NOT NULL,
    wiek TINYINT NOT NULL,
    PRIMARY KEY (id)
  )";

  $result = $conn->query($query);
  if (!$result) die ("Brak dostępu do bazy danych: " . $conn->error);
?>
```

Jak widać, zapytanie do bazy MySQL w PHP wygląda bardzo podobnie jak wówczas, gdy wpisuje się je z poziomu wiersza poleceń; różnica polega przede wszystkim na braku kończącego je średnika — ten znak jest bowiem zbędny w przypadku wysyłania zapytań za pośrednictwem PHP.

Wyświetlanie informacji o tabeli

Oto przydatny program, który umożliwia weryfikację poprawności tabeli utworzonej za pomocą przeglądarki bez uciekania się do wiersza poleceń MySQL. Jego działanie opiera się na instrukcji `DESCRIBE koty` i polega na wyświetleniu tabeli HTML z czterema nagłówkami — *Kolumna*, *Typ*, *Null* oraz *Klucz*. Pod nagłówkami są wymienione wszystkie kolumny tabeli. Aby użyć tego programu w odniesieniu do innej tabeli, wystarczy zastąpić nazwę `koty` w zapytaniu nazwą żądanej tabeli (przykład 10.8).

Przykład 10.8. Wyświetlanie informacji o tabeli koty

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  $query  = "DESCRIBE koty";
  $result = $conn->query($query);
  if (!$result) die ("Brak dostępu do bazy danych: " . $conn->error);
  $rows = $result->num_rows;
```

```
  echo "<table><tr><th>Kolumna</th><th>Typ</th><th>Null</th><th>Klucz</th></tr>";
  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $result->data_seek($j);
    $row = $result->fetch_array(MYSQLI_NUM);
    echo "<tr>";
    for ($k = 0 ; $k < 4 ; ++$k) echo "<td>$row[$k]</td>";
    echo "</tr>";
  }
  echo "</table>";
?>
```

Efekt działania powyższego programu powinien wyglądać tak:

```
Kolumna Typ         Null Klucz
id      smallint(6) NO   PRI
rodzina varchar(32) NO
imie    varchar(32) NO
wiek    tinyint(4)  NO
```

Usuwanie tabeli

Usuwanie tabeli jest bardzo proste, a co za tym idzie — bardzo niebezpieczne, trzeba więc zachować ostrożność. Przykład 10.9 przedstawia kod, który to umożliwia. Proponuję jednak, żebyś chwilę poczekał z jego wypróbowaniem, aż zapoznasz się z kolejnymi przykładami — uruchomienie tego programu spowoduje bowiem usunięcie tabeli *koty* i trzeba będzie ją ponownie utworzyć za pomocą kodu z przykładu 10.7.

Przykład 10.9. Usuwanie tabeli koty

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  $query  = "DROP TABLE koty";
  $result = $conn->query($query);
  if (!$result) die ("Brak dostępu do bazy danych: " . $conn->error);
?>
```

Dodawanie danych

Dodajmy trochę danych do naszej tabeli, korzystając z kodu przykładu 10.10.

Przykład 10.10. Dodawanie danych do tabeli koty

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  $query  = "INSERT INTO koty VALUES(NULL, 'Lew', 'Leon', 4)";
  $result = $conn->query($query);
  if (!$result) die ("Brak dostępu do bazy danych: " . $conn->error);
?>
```

W analogiczny sposób można dodać informacje o dwóch innych kotach. Aby to zrobić, zmień zawartość zmiennej `$query` zgodnie z poniższymi propozycjami i za każdym razem ponownie uruchom program w przeglądarce:

```
$query = "INSERT INTO koty VALUES(NULL, 'Puma', 'Pazur', 2)";
$query = "INSERT INTO koty VALUES(NULL, 'Gepard', 'Splinter', 3)";
```

Zwróciłeś uwagę na parametr `NULL` przekazywany na początku listy danych? Jego obecność wynika z faktu, że kolumna *id* jest typu `AUTO_INCREMENT`, więc to MySQL decyduje, jaką wartość w niej wpisać, zgodnie z kolejnym dostępnym numerem. Jeśli przekażemy w zapytaniu wartość `NULL`, parametr ten zostanie zignorowany, co załatwi sprawę.

Oczywiście najskuteczniejszym sposobem na umieszczenie danych w bazie MySQL jest utworzenie tablicy i wstawienie danych w postaci jednego zapytania.

Odczytywanie danych

Ponieważ w tabeli *koty* powinny już być jakieś dane, za pomocą kodu z przykładu 10.11 możemy sprawdzić, czy zostały one poprawnie wprowadzone.

Przykład 10.11. Odczytywanie wierszy z tabeli koty

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  $query  = "SELECT * FROM koty";
  $result = $conn->query($query);
  if (!$result) die ("Brak dostępu do bazy danych: " . $conn->error);
  $rows = $result->num_rows;
  echo "<table><tr><th>Id</th><th>Rodzina</th><th>Imię</th><th>Wiek</th></tr>";
  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $result->data_seek($j);
    $row = $result->fetch_array(MYSQLI_NUM);
    echo "<tr>";
    for ($k = 0 ; $k < 4 ; ++$k) echo "<td>$row[$k]</td>";
    echo "</tr>";
  }
  echo "</table>";
?>
```

Ten kod powoduje po prostu wysłanie zapytania MySQL w postaci `SELECT * FROM koty` i wyświetla wszystkie zwrócone przez nie wiersze. Rezultat jest następujący:

```
Id Rodzina Imię      Wiek
1  Lew     Leon      4
2  Puma    Pazur     2
3  Gepard  Splinter  3
```

Jak widać, wartość w kolumnie *id* była automatycznie zwiększana.

Aktualizowanie danych

Zmiana danych znajdujących się w tabeli również jest dość prosta. Być może zwróciłeś uwagę na pisownię imienia geparda? Zamiast Splinter powinno być Sprinter. Wprowadźmy więc odpowiednią poprawkę, jak w przykładzie 10.12.

Przykład 10.12. Zmiana imienia geparda ze Splinter na Sprinter

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  $query  = "UPDATE koty SET imie='Sprinter' WHERE imie='Splinter'";
  $result = $conn->query($query);
  if (!$result) die ("Brak dostępu do bazy danych: " . $conn->error);
?>
```

Jeśli teraz ponownie uruchomiłbyś przykład 10.11, to okazałoby się, że rezultat jest następujący:

```
Id Rodzina Imię      Wiek
1 Lew      Leon      4
2 Puma     Pazur     2
3 Gepard   Sprinter  3
```

Usuwanie danych

Puma Pazur została przeniesiona do innego zoo, powinniśmy więc usunąć ją z bazy danych (przykład 10.13).

Przykład 10.13. Usuwanie pumy o imieniu Pazur z tabeli koty

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  $query  = "DELETE FROM koty WHERE imie='Pazur'";
  $result = $conn->query($query);
  if (!$result) die ("Brak dostępu do bazy danych: " . $conn->error);
?>
```

Ten przykład bazuje na standardowym zapytaniu `DELETE FROM`, a gdy po jego wykonaniu raz jeszcze uruchomisz program z przykładu 10.11, okaże się, że jeden wiersz tabeli rzeczywiście został usunięty:

```
Id Rodzina Imię      Wiek
1 Lew      Leon      4
3 Gepard   Sprinter  3
```

Zastosowanie opcji AUTO_INCREMENT

W przypadku opcji `AUTO_INCREMENT` nie wiadomo, jaka konkretnie wartość została umieszczona w kolumnie po dodaniu do tabeli nowego wiersza. Jeśli chcesz się dowiedzieć, musisz zapytać o to MySQL za pomocą funkcji `mysql_insert_id` albo użyć właściwości `insert_id` obiektu połączenia. Takie sytuacje jak powyższa zdarzają się często, na przykład przy przetwarzaniu zamówienia, po dodaniu nowego klienta do tabeli *Klienci*; można potem odwołać się do nowo utworzonego identyfikatora *IdKlienta* podczas umieszczania danych o jego zakupach w tabeli *Zakupy*.

Przykład 10.14 opiera się na zmodyfikowanym kodzie przykładu 10.10. Zmiany powodują wyświetlenie omawianej wartości po każdym wstawieniu nowego wiersza.

Przykład 10.14. Dodawanie danych do tabeli koty i wyświetlanie identyfikatora nowego wiersza

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  $query  = "INSERT INTO koty VALUES(NULL, 'Ryś', 'Rysiek', 5)";
  $result = $conn->query($query);
  if (!$result) die ("Brak dostępu do bazy danych: " . $conn->error);
  echo "Identyfikator nowego wiersza to: " . $conn->insert_id;
?>
```

Zawartość tabeli powinna teraz wyglądać tak jak poniżej (zwróć uwagę, że użyty wcześniej identyfikator 2 *nie został* ponownie wykorzystany, gdyż w pewnych sytuacjach mogłoby to prowadzić do komplikacji):

```
Id Rodzina Imię      Wiek
1 Lew      Leon      4
3 Gepard   Sprinter  3
4 Ryś      Rysiek    5
```

Zastosowanie identyfikatorów wstawionych wierszy

Bardzo często zdarza się, że dane trzeba wstawić w kilku tabelach: po dodaniu książki trzeba wprowadzić informacje o jej autorze, po utworzeniu konta nowego klienta należy dodać dane o jego zakupie itp. W takich przypadkach, jeśli użyta została kolumna z automatyczną inkrementacją, należy zachować identyfikator wstawionego wiersza, aby posłużyć się nim w innej, powiązanej tabeli.

Przypuśćmy na przykład, że koty z naszego zoo mogą być „wirtualnie adoptowane" przez osoby spoza ogrodu, które będą organizowały zbiórki wspomagające ich utrzymanie. W takim przypadku po umieszczeniu nowego kota w tabeli *koty* powinniśmy utworzyć klucz umożliwiający powiązanie zwierzęcia z jego opiekunem. Kod umożliwiający przeprowadzenie tej operacji jest podobny do kodu z przykładu 10.14. Różnicą jest to, że zwrócony identyfikator nowego wiersza trafia do zmiennej o nazwie `$insertID`, która jest następnie wykorzystywana jako element kolejnego zapytania:

```
$query    = "INSERT INTO koty VALUES(NULL, 'Ryś', 'Rysiek', 5)";
$result   = $conn->query($query);
$insertID = $conn->insert_id;

$query    = "INSERT INTO sponsorzy VALUES($insertID, 'Anna', 'Kowalska')";
$result   = $conn->query($query);
```

W rezultacie kot jest powiązany ze swoim opiekunem poprzez unikatowy identyfikator, utworzony automatycznie przy użyciu opcji `AUTO_INCREMENT`.

Zastosowanie blokad

Bezpieczna procedura łączenia tabel za pośrednictwem identyfikatora wstawionego wiersza polega na zastosowaniu blokad (albo transakcji, zgodnie ze wskazówkami podanymi w rozdziale 9.). Takie rozwiązanie może wprawdzie wydłużyć czas operacji w przypadku dużej liczby użytkowników zapisujących dane w tej samej tabeli, ale na ogół warto się z tym pogodzić. Procedura wygląda następująco:

1. Zablokowanie pierwszej tabeli (np. *koty*).
2. Wstawienie danych do pierwszej tabeli.
3. Pobranie unikatowego identyfikatora nowego wiersza z pierwszej tabeli przy użyciu właściwości `insert_id`.
4. Odblokowanie pierwszej tabeli.
5. Wstawienie danych do drugiej tabeli.

Blokadę można bezpiecznie zdjąć przed wpisaniem danych do drugiej tabeli, gdyż identyfikator został już pobrany i zapisany w zmiennej programu. Zamiast blokowania tabel można się posłużyć transakcjami, ale to rozwiązanie jeszcze bardziej spowalnia działanie serwera MySQL.

Wykonywanie zapytań pomocniczych

Na razie wystarczy kocich spraw. Aby przeanalizować trochę bardziej skomplikowane zapytania, musimy bowiem wrócić do tabel *customers* oraz *classics* utworzonych w rozdziale 8. W tabeli *customers* znajdują się informacje o klientach, tabela *classics* zawiera informacje o kilku książkach. Obie tabele mają wspólną kolumnę z numerami ISBN, o nazwie *isbn*, której możemy użyć do utworzenia dodatkowych zapytań.

Na przykład w celu wyświetlenia wszystkich klientów wraz z tytułami i autorami kupionych przez nich książek można użyć kodu podanego w przykładzie 10.15.

Przykład 10.15. Wykonywanie zapytań pomocniczych

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);

  $query  = "SELECT * FROM customers";
  $result = $conn->query($query);
  if (!$result) die ("Brak dostępu do bazy danych: " . $conn->error);

  $rows = $result->num_rows;

  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $result->data_seek($j);
    $row = $result->fetch_array(MYSQLI_NUM);
    echo "$row[0] zakupił(a) ISBN $row[1]:<br>";
    $subquery  = "SELECT * FROM classics WHERE isbn='$row[1]'";
    $subresult = $conn->query($subquery);
    if (!$subresult) die ("Brak dostępu do bazy danych: " . $conn->error);
    $subrow = $subresult->fetch_array(MYSQLI_NUM);
    echo "  '$subrow[1]' autora $subrow[0]<br>";
  }
?>
```

Ten program wykonuje zapytanie pomocnicze do tabeli *customers* i wyszukuje wszystkich klientów, a następnie na podstawie numeru ISBN zakupionych przez nich książek wykonuje kolejne zapytanie do tabeli *classics*, aby odszukać tytuły i autorów tych pozycji. Rezultat działania powyższego kodu jest następujący:

```
Joe Bloggs zakupił(a) ISBN 9780099533474:
'The Old Curiosity Shop' autora Charles Dickens
Mary Smith zakupił(a) ISBN 9780582506206:
 'Pride and Prejudice' autora Jane Austen
Jack Wilson zakupił(a) ISBN 9780517123201:
 'The Origin of Species' autora Charles Darwin
```

W tym konkretnym przypadku — choć nie byłaby to ilustracja możliwości zapytań pomocniczych — można byłoby uzyskać te same informacje za pomocą zapytania typu `NATURAL JOIN` (rozdział 8.), na przykład takiego:

```
SELECT name,isbn,title,author FROM customers
 NATURAL JOIN classics;
```

Zapobieganie próbom ataków

Brak weryfikacji danych wprowadzanych przez użytkowników do bazy MySQL niesie ze sobą duże ryzyko, z którego nie zawsze zdajemy sobie sprawę. Przypuśćmy, że mamy następujący prosty kawałek kodu służący do weryfikacji użytkowników:

```
$user  = $_POST['user'];
$pass  = $_POST['pass'];
$query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";
```

Na pierwszy rzut oka taki kod nie budzi najmniejszych zastrzeżeń. Jeśli na przykład użytkownik wprowadzi wartości `jankowalski` oraz `haslo123`, które trafią do zmiennych `$user` i `$pass`, to całe zapytanie przekazane do bazy MySQL będzie wyglądało następująco:

```
SELECT * FROM users WHERE user='jankowalski' AND pass='haslo123'
```

Tu akurat rzeczywiście nic złego się nie stanie, ale co, jeśli do zmiennej `$user` trafiłby następujący łańcuch znaków (a zmienna `$pass` pozostałaby pusta)?

```
admin' #
```

Przyjrzyjmy się zapytaniu, które w tej sytuacji zostałoby wysłane do serwera MySQL:

```
SELECT * FROM users WHERE user='admin' #' AND pass=''
```

Dostrzegasz problem? W MySQL symbol `#` oznacza początek komentarza. W efekcie użytkownik zostałby zalogowany jako *admin* (jeśli takie konto rzeczywiście istnieje) bez konieczności wpisywania hasła. Przyjrzyj się jeszcze raz temu zapytaniu; pogrubieniem została oznaczona tylko ta część, która rzeczywiście zostałaby wykonana; resztę serwer by pominął:

```
SELECT * FROM users WHERE user='admin' #' AND pass=''
```

Można byłoby jednak mówić o dużym szczęściu, gdyby złośliwy użytkownik ograniczył się do czegoś takiego. W tej sytuacji mógłbyś bowiem uruchomić swoją aplikację i naprawić szkody wyrządzone przez kogoś, kto zaloguje się jako *admin*. Ale co by było, gdyby kod programu powodował usunięcie użytkownika z bazy? Kod mógłby wyglądać na przykład tak:

```
$user = $_POST['user'];
$pass = $_POST['pass'];
$query = "DELETE FROM users WHERE user='$user' AND pass='$pass'";
```

I ponownie: taki kod nie wydaje się podejrzany, ale co by się stało, gdyby ktoś wpisał do zmiennej `$user` poniższy ciąg znaków?

```
cokolwiek' OR 1=1 $
```

Do serwera MySQL trafiłoby następujące zapytanie:

```
DELETE FROM users WHERE user='cokolwiek' OR 1=1 # AND pass=''
```

Ups… to zapytanie SQL zawsze będzie zwracało wartość TRUE, a w rezultacie cała baza użytkowników zostałaby usunięta! W jaki sposób zabezpieczyć się przed tego typu atakami?

Działania prewencyjne

Przede wszystkim nie należy polegać na wbudowanym mechanizmie PHP o nazwie *magic quotes* (dosł. magiczne cudzysłowy), który automatycznie poprzedza pojedyncze i podwójne cudzysłowy modyfikatorem —lewym ukośnikiem (\). Dlaczego? Ponieważ ten mechanizm można wyłączyć; wielu programistów postępuje w ten sposób, aby zastąpić go własnymi zabezpieczeniami. Nie ma więc żadnej gwarancji, że właśnie tak się nie stało na serwerze, na którym pracujesz. Co więcej, w PHP 5.3.0 mechanizm ten został uznany za przestarzały, a z PHP 6.0.0 już go usunięto.

Zamiast tego we wszystkich odwołaniach do MySQL powinieneś używać funkcji real_escape_string. Przykład 10.16 ilustruje działanie tej funkcji, która „oczyszcza" łańcuch znaków wprowadzony przez użytkownika z modyfikatorów i niepożądanych znaków.

Przykład 10.16. Prawidłowe „oczyszczanie" danych użytkownika przed wysłaniem do serwera MySQL

```
<?php
  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Funkcja get_magic_quotes_gpc zwraca wartość TRUE, jeśli mechanizm *magic quotes* jest aktywny. W takim przypadku ukośniki dodane do łańcucha znaków zostają usunięte, bo w przeciwnym razie metoda real_escape_string spowoduje dodanie kolejnych modyfikatorów i doprowadzi do przekłamań w łańcuchu znaków. W przykładzie 10.17 została zdefiniowana funkcja mysql_fix_string, wykorzystująca opisane rozwiązanie. Funkcji tej można bez przeszkód użyć w kodzie innych, własnych programów.

Przykład 10.17. Bezpieczny dostęp do MySQL z użyciem danych użytkownika

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);
  $user  = mysql_fix_string($conn, $_POST['user']);
  $pass  = mysql_fix_string($conn, $_POST['pass']);
  $query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";

 // itd.

  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Rola opisanych zabezpieczeń ostatnio trochę zmalała, a to ze względu na znacznie łatwiejszą i bezpieczniejszą metodę dostępu do MySQL, która pozwala uniknąć ich stosowania. Mam na myśli używanie elementów zastępczych, o których przeczytasz za chwilę.

Zastosowanie elementów zastępczych

Przygotowanie wyrażeń z elementami zastępczymi umożliwia przesłanie do bazy „czystych" danych, dzięki czemu można uniknąć niepożądanej interpretacji danych przesłanych przez użytkownika (lub innych) jako instrukcji MySQL (co stanowi potencjalną furtkę do włamania).

Trik polega na uprzednim zdefiniowaniu wyrażenia, które ma być wykonane przez MySQL, i zastąpieniu wszystkich jego elementów, które będą zawierały dane, zwykłymi znakami zapytania.

W czystym kodzie MySQL tak spreparowane wyrażenie wygląda podobnie jak w przykładzie 10.18.

Przykład 10.18. Elementy zastępcze w MySQL

```
PREPARE statement FROM "INSERT INTO classics VALUES(?,?,?,?,?)";

SET @author   = "Emily Brontë",
    @title    = "Wuthering Heights",
    @category = "Classic Fiction",
    @year     = "1847",
    @isbn     = "9780553212587";

EXECUTE statement USING @author,@title,@category,@year,@isbn;
DEALLOCATE PREPARE statement;
```

Obsługa tego rodzaju wyrażeń jest dość kłopotliwa, ale na szczęście w sukurs przychodzi nam rozszerzenie `mysqli`, a konkretnie jego metoda o nazwie `prepare`, którą wywołuje się następująco:

```
$stmt = $conn->prepare('INSERT INTO classics VALUES(?,?,?,?,?)');
```

Obiekt o nazwie `$stmt` (lub dowolnej innej, jaką wybierzesz) zwracany przez tę metodę jest następnie używany do przesyłania na serwer danych w miejsce znaków zapytania. Najpierw jednak należy powiązać zmienne PHP z poszczególnymi znakami zapytania (czyli elementami zastępczymi) w kolejności ich wymienienia:

```
$stmt->bind_param('sssss', $author, $title, $category, $year, $isbn);
```

Pierwszy argument metody `bind_param` to ciąg znaków odzwierciedlający typy kolejnych argumentów. W tym przypadku składa się z pięciu znaków `s`, które odpowiadają łańcuchom tekstowym, ale można w nim podać dowolną kombinację typów według następujących zasad:

- `i` — liczba całkowita,
- `d` — liczba zmiennoprzecinkowa podwójnej precyzji,
- `s` — łańcuch znaków,
- `b` — obiekt binarny (tzw. *blob*, zostanie przesłany w pakietach).

Po powiązaniu zmiennych z elementami gotowego wyrażenia należy zapełnić te zmienne danymi, które zostaną przekazane do MySQL:

```
$author   = 'Emily Brontë';
$title    = 'Wuthering Heights';
$category = 'Classic Fiction';
$year     = '1847';
$isbn     = '9780553212587';
```

Na tym etapie PHP dysponuje wszystkimi informacjami niezbędnymi do wykonania przygotowanego wyrażenia. Wobec tego możemy użyć poniższej instrukcji, która odwołuje się do metody `execute` obiektu `$stmt` utworzonego wcześniej:

```
$stmt->execute();
```

Zanim przystąpimy do dalszych operacji, sprawdźmy, czy instrukcja została wykonana poprawnie. Można tego dokonać przez zweryfikowanie właściwości `affected_rows` wyrażenia `$stmt`:

```
printf("%d Wiersz wstawiony.\n", $stmt->affected_rows);
```

W przypadku przykładu podanego niżej instrukcja ta powinna poinformować o pomyślnym wstawieniu jednego wiersza.

Po udanym wykonaniu instrukcji (albo rozprawieniu się z ewentualnymi błędami) można zamknąć obiekt `$stmt`:

```
$stmt->close();
```

Na koniec można też zamknąć obiekt `$conn` (przy założeniu, że nie będziesz go już używał):

```
$conn->close();
```

Owocem wszystkich wykonanych działań jest przykład 10.19:

Przykład 10.19. Praktyczne zastosowanie przygotowanego wyrażenia

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die($conn->connect_error);

  $stmt = $conn->prepare('INSERT INTO classics VALUES(?,?,?,?,?)');
  $stmt->bind_param('sssss', $author, $title, $category, $year, $isbn);

  $author   = 'Emily Brontë';
  $title    = 'Wuthering Heights';
  $category = 'Classic Fiction';
  $year     = '1847';
  $isbn     = '9780553212587';

  $stmt->execute();
  printf("%d Wiersz wstawiony.\n", $stmt->affected_rows);
  $stmt->close();
  $conn->close();
?>
```

Każde użycie predefiniowanych wyrażeń zamiast zwykłych to jedna okazja dla potencjalnych hakerów mniej, warto więc poświęcić trochę czasu na opanowanie techniki ich stosowania.

Zapobieganie przekazywaniu niepożądanych danych przez HTML

Istnieje jeszcze jedna metoda podstawiania niepożądanych danych, przed którą warto się zabezpieczyć — przy czym tym razem chodzi nie tyle o bezpieczeństwo strony internetowej,

ale o prywatność i ochronę praw jej użytkowników. Mowa o ataku typu *cross-site scripting* (w skrócie XSS).

Do tego rodzaju ataku może dojść wówczas, gdy zezwoli się użytkownikowi na wpisywanie kodu HTML (a częściej JavaScript), który zostanie następnie wyświetlony na stronie. Bardzo często wykorzystuje się w tym celu formularz komentarzy. Atak na ogół polega na podjęciu przez hakera próby skonstruowania takiego kodu, który odczytywałby zawartość ciasteczek użytkowników strony, przechwytywał zapisywane w nich loginy i ewentualnie hasła bądź inne informacje. Co gorsza, w ten sposób można też opracować atak mający na celu pobranie złośliwego oprogramowania (trojana) przez zwykłych użytkowników.

Zapobieganie takim działaniom jest na szczęście proste i polega na zastosowaniu funkcji `htmlentities`, która usuwa z kodu HTML znaki specjalne znaczników i zamienia je na łańcuchy tekstowe, tzw. encje HTML. W tej postaci są one przez przeglądarkę wyświetlane jako zwykłe znaki, ale nie będą traktowane jako elementy kodu. Weźmy na przykład następujący kod HTML:

```
<script src='http://x.com/hack.js'>
</script><script>hack();</script>
```

Załóżmy, że ten kod powoduje uruchomienie programu w JavaScripcie, który podejmuje złośliwe działania. Jeśli jednak najpierw „przepuścimy" ten kod przez funkcję `htmlentities`, otrzymamy następujący, zupełnie nieszkodliwy łańcuch znaków:

```
&lt;script src='http://x.com/hack.js'&gt;
&lt;/script&gt;&lt;script&gt;hack();&lt;/script&gt;
```

Jeśli zamierzasz uwzględnić w programie możliwość wyświetlania informacji wprowadzonych przez użytkownika — czy to od razu, czy po zapisaniu ich w bazie danych — najpierw powinieneś oczyścić te informacje przy użyciu funkcji `htmlentities`. Najwygodniej będzie napisać w tym celu własną funkcję, taką jak w przykładzie 10.20, która oczyszcza kod SQL i zapobiega atakom XSS.

Przykład 10.20. Funkcje zapobiegające przemycaniu złośliwego kodu w zapytaniach SQL i atakom XSS

```
<?php
  function mysql_entities_fix_string($conn, $string)
  {
    return htmlentities(mysql_fix_string($conn, $string));
  }
  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Funkcja `mysql_entities_fix_string` najpierw wywołuje funkcję `mysql_fix_string`, następnie przekazuje rezultat jej działania do funkcji `htmlentities` i wreszcie zwraca w pełni oczyszczony łańcuch znaków. Aby móc skorzystać z dowolnej z tych funkcji, musisz dysponować obiektem otwartego połączenia z bazą MySQL.

Przykład 10.21 stanowi udoskonaloną („nie do złamania") wersję przykładu 10.17.

Przykład 10.21. Sposób na bezpieczny dostęp do MySQL i uniknięcie ataków typu XSS

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
```

```
  if ($conn->connect_error) die($conn->connect_error);
  $user  = mysql_entities_fix_string($conn, $_POST['user']);
  $pass  = mysql_entities_fix_string($conn, $_POST['pass']);
  $query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";
  // itd.
  function mysql_entities_fix_string($conn, $string)
  {
    return htmlentities(mysql_fix_string($conn, $string));
  }
  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Proceduralny wariant zastosowania mysqli

Jeśli wolisz, możesz skorzystać z alternatywnego zestawu funkcji, umożliwiającego obsługę `mysqli` w postaci proceduralnej (a nie obiektowej).

Zamiast tworzenia obiektu `$connection` w taki sposób:

```
$conn = new mysqli($hn, $un, $pw, $db);
```

możesz użyć następującego kodu:

```
$link = mysqli_connect($hn, $un, $pw, $db);
```

Aby sprawdzić poprawność połączenia i je obsłużyć, możesz wykorzystać następującą instrukcję:

```
if (mysqli_connect_errno()) die(mysqli_connect_error());
```

Z kolei w celu wysłania zapytania do MySQL należy użyć następującego kodu:

```
$result = mysqli_query($link, "SELECT * FROM classics");
```

Po jego wykonaniu rezultat zapytania trafi do zmiennej `$result`. Liczbę zwróconych wierszy można sprawdzić następująco:

```
$rows = mysqli_num_rows($result));
```

Do zmiennej `$rows` trafia całkowita liczba pobranych wierszy. Konkretne dane możesz odczytać wierszami za pomocą następującej instrukcji, która zwraca tablicę numeryczną:

```
$row = mysqli_fetch_array($result, MYSQLI_NUM);
```

W tym przypadku pole `$row[0]` będzie zawierało pierwszą kolumnę danych, pole `$row[1]` drugą kolumnę i tak dalej. Zgodnie z wcześniejszym opisem poszczególne wiersze danych mogą być zwrócone w postaci tablic asocjacyjnych lub obu typów tablic — w zależności od wartości drugiego argumentu.

Jeśli chciałbyś sprawdzić identyfikator operacji wstawiania, możesz w tym celu użyć funkcji `mysqli_insert_id` w następujący sposób:

```
$insertID = mysqli_insert_id($result);
```

Zastosowanie znaków modyfikujących w przypadku proceduralnej obsługi `mysqli` jest bardzo proste i można to zrobić na przykład tak:

```
$escaped = mysqli_real_escape_string($link, $val);
```

Przygotowanie zapytania przedstawia się następująco:

```
$stmt = mysqli_prepare($link, 'INSERT INTO classics VALUES(?,?,?,?,?)');
```

Aby powiązać zmienne z konkretnymi znakami zastępczymi w zapytaniu, należy postąpić tak:

```
mysqli_stmt_bind_param($stmt, 'sssss', $author, $title, $category, $year, $isbn);
```

Z kolei aby wykonać tak przygotowane zapytanie po podstawieniu zmiennych, trzeba użyć następującej instrukcji:

```
mysqli_stmt_execute($stmt);
```

Zakończenie pracy z danym zapytaniem wygląda tak:

```
mysqli_stmt_close($stmt);
```

A tak zamyka się połączenie z bazą MySQL:

```
mysqli_close($link);
```

Szczegółowe informacje na temat stosowania elementów zastępczych (proceduralnie lub obiektowo) znajdziesz pod adresem *http://tinyurl.com/mysqlistmt*. Z kolei omówienie innych aspektów zastosowania rozszerzenia `mysqli` zostało opisane na stronie *http://tinyurl.com/usingmysqli*.

Poznałeś różne sposoby odwoływania się do baz MySQL za pomocą PHP, a to stanowi solidny fundament pod wiedzę, jaką przyswoisz w następnym rozdziale. Przeczytasz w nim bowiem o tworzeniu wygodnych w obsłudze formularzy i wykorzystywaniu danych przekazywanych za ich pośrednictwem.

Pytania

1. Jak można połączyć się z bazą MySQL za pomocą `mysqli`?
2. Jak wysłać do bazy MySQL zapytanie za pomocą `mysqli`?
3. W jaki sposób uzyskać dostęp do łańcucha tekstowego z komunikatem błędu, jeśli taki wystąpił przy korzystaniu z `mysqli`?
4. W jaki sposób sprawdzić liczbę wierszy zwróconych przez zapytanie `mysqli`?
5. Jak odczytać konkretny wiersz danych spośród całego rezultatu zwróconego przez `mysqli`?
6. Jakiej metody `mysqli` należy użyć, aby oczyścić dane wprowadzone przez użytkownika i w ten sposób uniknąć „przemycenia” niepożądanego kodu?
7. Jakie skutki uboczne może mieć niezamknięcie obiektów utworzonych za pomocą metod `mysqli`?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 10.”.

Skorowidz

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

X

Z

Ź

Ż

O'REILLY®

Kompendium wiedzy dla twórcy stron i aplikacji sieciowych!

PHP wraz z bazą danych MySQL oraz językiem JavaScript to potężne trio, dzięki któremu możesz zbudować aplikację internetową dowolnej wielkości. Jeżeli do tego dołożysz możliwości najnowszej wersji języka HTML (oznaczonej cyfrą 5), CSS3 oraz bibliotekę jQuery, nic nie będzie w stanie zatrzymać Twojej kreatywności.

Jeżeli chcesz opanować te narzędzia i stworzyć atrakcyjną oraz funkcjonalną aplikację internetową, trafiłeś na idealną książkę. Znajdziesz tu niezbędną wiedzę o języku PHP, bazie danych MySQL, HTML5, CSS3, JavaScripcie i jQuery. Już od pierwszych stron będziesz poznawać składnię i konstrukcje języka programowania PHP, techniki programowania obiektowego oraz praktyczne porady związane z używaniem PHP. Następnie uzupełnisz wiedzę na temat bazy danych MySQL. Dowiesz się, jak tworzyć zapytania SQL oraz w jaki sposób wykorzystać dane zawarte w bazie z poziomu PHP. Po opanowaniu „strony serwerowej" przejdziesz do nauki technik tworzenia interaktywnych stron WWW. Zobaczysz, jak używać języka JavaScript, jakie nowości zawiera HTML5 oraz jak wielki potencjał kryją w sobie CSS3 i jQuery. Książka ta jest ciekawą lekturą dla pasjonatów chcących tworzyć własne, zaawansowane aplikacje.

Dzięki tej książce:

- przygotujesz środowisko pracy
- poznasz język PHP oraz jego możliwości
- wykorzystasz możliwości bazy danych MySQL
- zbudujesz aplikację internetową

Robin Nixon – publicysta, autor setek artykułów poświęconych technologiom komputerowym, związany z branżą IT od wczesnych lat 80. XX wieku. Stworzył liczne strony WWW przy użyciu narzędzi open source. Jest specjalistą w zakresie PHP, MySQL, JavaScriptu i HTML.

Helion

32799 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

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

ISBN 978-83-283-0842-8

9 788328 308428

Informatyka w najlepszym wydaniu

cena: 99,00 zł