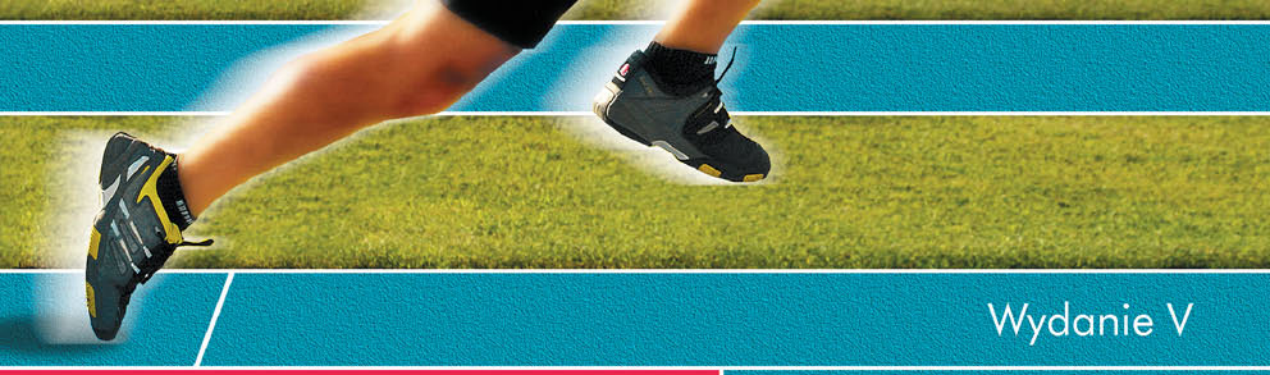


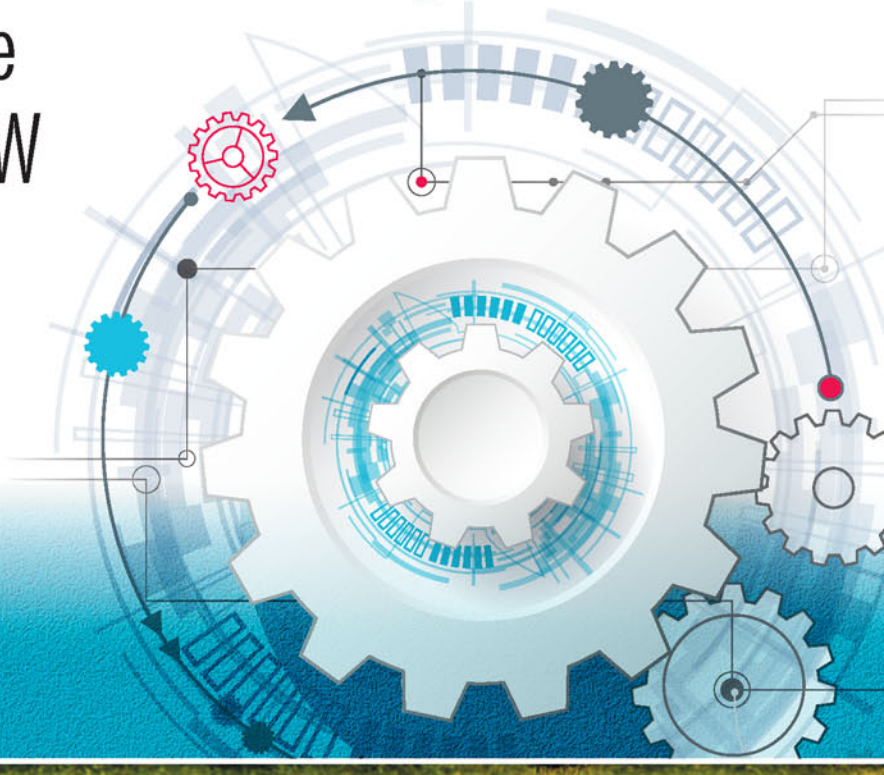
Wydanie V

Szybki start

Larry Ullman

PHP i MySQL

Dynamiczne strony WWW



Helion

Tytuł oryginału: PHP and MySQL for Dynamic Web Sites: Visual QuickPro Guide (5th Edition)

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-4466-2

Authorized translation from the English language edition, entitled: VISUAL QUICKPRO GUIDE: PHP AND MYSQL FOR DYNAMIC WEB SITES, Fifth Edition; ISBN 0134301846; by Larry Ullman; published by Pearson Education, Inc, publishing as Peachpit Press. Copyright © 2018 by Larry Ullman.

HELION SA
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/phmys5.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/phmys5
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

9

PHP i MySQL

Teraz, gdy dysponujesz już podstawowymi wiadomościami na temat PHP, SQL-a i MySQL-a, pora użyć tych technologii razem. Silna integracja PHP z MySQL-em jest jednym z głównych powodów ich popularności wśród programistów. Będziesz zaskoczony, jak łatwo można tworzyć aplikacje przy użyciu tych technologii.

W tym rozdziale wykorzystam bazę *sitename* (utworzoną w rozdziale 5., „Wprowadzenie do SQL-a") i opracuję w PHP interfejs umożliwiający wykonywanie operacji na tabeli `users`. Informacje i przykłady zawarte w tym rozdziale dadzą Ci podstawy do tworzenia wszelkiego rodzaju aplikacji PHP-MySQL, ponieważ pewne uniwersalne reguły są w każdym przypadku takie same.

Zanim przystąpisz do lektury tego rozdziału, powinieneś dobrze opanować materiał przedstawiony w pierwszych ośmiu. Dotyczy to również poznania technik usuwania i obsługi błędów przedstawionych w poprzednim rozdziale. Pamiętaj, że w celu wykonywania przykładów przedstawionych w tym rozdziale będziesz potrzebował działającego serwera internetowego obsługującego PHP, a także dostępu do serwera MySQL.

W tym rozdziale

Modyfikacja szablonu

Ponieważ wszystkie strony, jakie utworzę w tym i w kolejnym rozdziale, będą częścią tej samej aplikacji internetowej, warto poświęcić trochę czasu na opracowanie dla nich odpowiedniego szablonu. Zamiast tworzyć go od zera, wykorzystam szablon przygotowany w rozdziale 3., „Tworzenie dynamicznych stron WWW", wprowadzając jedynie drobne modyfikacje w łączach nagłówka.

Aby utworzyć plik nagłówkowy:

1. Otwórz plik *naglowek.html* (listing 3.2) w edytorze tekstów lub IDE.
2. Zmień listę łączy w poniższy sposób (listing 8.1).

```
<ul class="nav navbar-nav">
  <li class="active"><a href=
    "index.php">Strona domowa</a>
  </li>
  <li><a href="rejestracja.php">
    Zarejestruj się</a></li>
  <li><a href=
    "pokaz_uzytkownikow.php">
    Użytkownicy</a></li>
  <li><a href=
    "zmiana_hasla.php">Zmień
    hasło</a></li>
</ul>
```

Wszystkie przykłady zamieszczone w tym rozdziale będą używać stron rejestracji, wyświetlania zarejestrowanych użytkowników i zmiany hasła. Natomiast łącza do stron kalendarza i kalkulatora można usunąć.

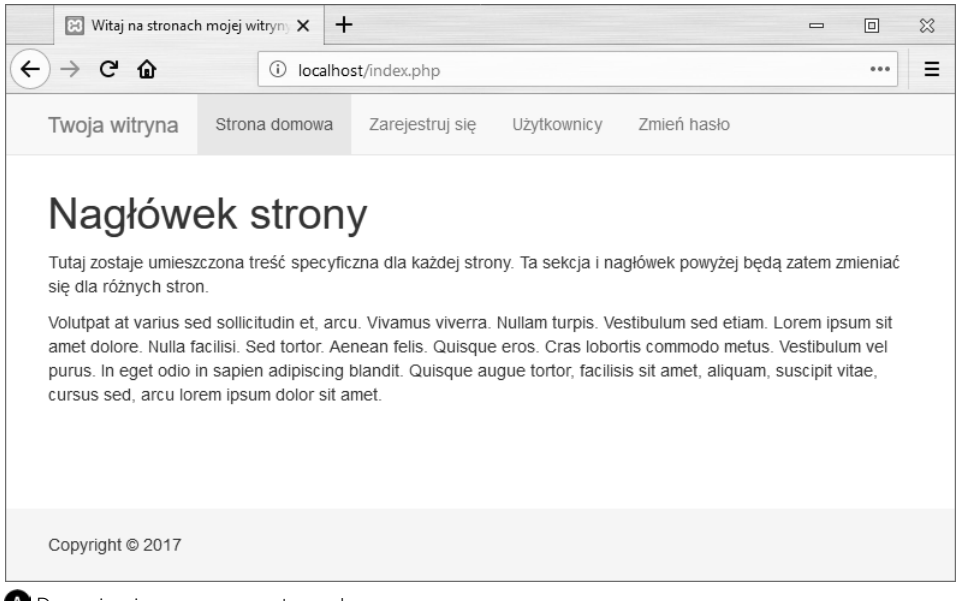
Listing 9.1. Plik nagłówka stosowany w szablonie stron i zawierający nowe łącza nawigacyjne

```
 1 <!DOCTYPE html>
 2 <html lang="pl">
 3 <head>
 4 <meta charset="utf-8">
 5 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 6 <meta name="viewport" content="width=device-width, initial-scale=1">
 7 <title><?php echo $page_title; ?></title>
 8 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/
     3.3.7/css/bootstrap.min.css" integrity="sha384-
     BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
     crossorigin="anonymous">
 9 <link href="css/sticky-footer-navbar.css" rel="stylesheet">
10 </head>
11 <body>
12 <nav class="navbar navbar-default navbar-fixed-top">
13   <div class="container">
14     <div class="navbar-header"><a class="navbar-brand" href="#">
         Twoja witryna</a></div>
15     <div id="navbar" class="collapse navbar-collapse">
16     <ul class="nav navbar-nav">
17       <li class="active"><a href="index.php">Strona domowa</a></li>
18       <li><a href="rejestracja.php">Zarejestruj się</a></li>
19       <li><a href="pokaz_uzytkownikow.php">Użytkownicy</a></li>
20       <li><a href="zmiana_hasla.php">Zmień hasło</a></li>
21     </ul>
22     </div>
23   </div>
24 </nav>
25 <div class="container">
26 <!-- Listing 9.1 - naglowek.html -->
```

3. Zapisz plik jako *naglowek.html*.
4. Załaduj nowy plik nagłówka na serwer internetowy do tego samego katalogu *includes*, w którym znajdzie się też plik *stopka.html* (listing 3.3) oraz *layout.css* (dostępny w przykładach do pobrania z *ftp://ftp.helion.pl/przyklady/phmys5.zip*).
5. Przetestuj nowy plik nagłówka, uruchamiając w przeglądarce skrypt *index.php* (patrz A).

Wskazówki

- Aby obejrzeć strukturę tworzonej witryny, zajrzyj do ramki „Zarządzanie dokumentami" w następnym podrozdziale.
- Nazwy plików pełniących rolę szablonów mogą mieć dowolne rozszerzenia, w tym także *.inc* i *.php*.



A Dynamicznie generowana strona domowa

Nawiązywanie połączenia z serwerem MySQL

Zanim będziesz mógł odwoływać się do MySQL-a, musisz połączyć się z serwerem. Umożliwia to funkcja `mysqli_connect()`:

```
$dbc = mysqli_connect (host, użytkownik,
➥hasło, nazwa_bazy);
```

To, jak powinny wyglądać pierwsze trzy argumenty przekazywane do funkcji (nazwa hosta, nazwa użytkownika i hasło), zależy od uprawnień, jakie przydzielono poszczególnym użytkownikom na serwerze MySQL. Więcej informacji na ten temat znajdziesz w dodatku A, „Instalacja". Najczęściej parametrowi *host* nadawać będziesz wartość *localhost*, choć nie zawsze (podanie innej nazwy hosta pozwala połączyć się z bazą obsługiwaną na innym serwerze).

Czwartym z argumentów jest nazwa bazy danych. Określenie wartości tego argumentu stanowi odpowiednik wykonania polecenia `USE nazwa_bazy` za pomocą klienta mysql.

Jeżeli uda się nawiązać połączenie, do bazy MySQL będzie można odwoływać się za pośrednictwem zmiennej `$dbc` (oczywiście jeśli chcesz, możesz jej nadać dowolną inną nazwę). Będziesz ją stosować jako pierwszy argument większości funkcji PHP współpracujących z MySQL-em.

Jeśli pojawi się problem z nawiązaniem połączenia, możesz wywołać funkcję `mysqli_connect_error()`, zwracającą komunikat o błędzie połączenia. Funkcja ta nie wymaga żadnego argumentu, zatem wywołasz ją w poniższy sposób:

```
mysqli_connect_error();
```

Po nawiązaniu połączenia z bazą danych musisz jeszcze ustawić odpowiednie kodowanie; użyj do tego funkcji `mysqli_set_charset()`:

```
mysqli_set_charset($dbc, 'utf8');
```

Wartość określająca kodowanie — czyli drugi argument wywołania funkcji — powinna odpowiadać kodowaniu zastosowanemu w skrypcie PHP oraz uporządkowaniu alfabetycznemu używanemu w bazie danych (patrz rozdział 6., „Projektowanie baz danych"); jeśli tego nie zrobisz, dane mogą być przesyłane z wykorzystaniem domyślnego kodowania, co może wywołać problemy.

Aby rozpocząć korzystanie z MySQL-a w PHP, stworzę specjalny skrypt, który będzie nawiązywać połączenie z serwerem MySQL. Następnie wszystkie pozostałe skrypty wymagające tego połączenia będą dołączać ten plik.

Aby nawiązać połączenie i wybrać bazę danych:

1. W edytorze tekstów lub IDE utwórz nowy dokument PHP, któremu nadasz nazwę *mysqli_connect.php* (listing 9.2).

   ```
   <?php # Listing 9.2 - mysqli_connect.php
   ```

   Plik ten będzie dołączany przez inne skrypty PHP, zatem nie musi zawierać kodu HTML.

2. Umieść nazwę hosta, nazwę użytkownika, hasło i nazwę bazy danych w stałych.

   ```
   DEFINE ('DB_USER', 'uzytkownik');
   DEFINE ('DB_PASSWORD', 'haslo');
   DEFINE ('DB_HOST', 'localhost');
   DEFINE ('DB_NAME', 'sitename');
   ```

   Ze względów bezpieczeństwa wolę umieszczać tego typu dane w stałych (dzięki temu nie można ich zmienić), ale nie jest to wymagane. Ogólnie rzecz biorąc, przypisanie tych wartości do jakichś zmiennych pozwala oddzielić parametry konfiguracyjne od funkcji, które je wykorzystują, ale to również nie jest obowiązkowe.

W trakcie pisania skryptu zmień te cztery wartości na takie, które są odpowiednie dla Twojej bazy danych, lub użyj nazwy użytkownika i hasła, które otrzymałeś dla bazy danych używanej na wynajmowanym serwerze. Możesz również skorzystać ze wskazówek zamieszczonych w dodatku A, stworzyć użytkownika mającego dostęp do bazy *sitename* i wstawić jego parametry do powyższego kodu. Obojętnie, które rozwiązanie zastosujesz, pamiętaj, aby użyć wartości, co do których jesteś absolutnie pewien, że będą działać z używanym przez Ciebie serwerem.

3. Połącz się z MySQL-em.

```
$dbc = @mysqli_connect (DB_HOST, DB_USER,
➥DB_PASSWORD, DB_NAME) OR
➥die('Brak połączenia z MySQL: ' .
➥mysqli_connect_error() );
```

Jeżeli tylko funkcji `mysqli_connect()` uda się nawiązać połączenie z MySQL-em, zwróci ona identyfikator zasobu reprezentujący otworzone połączenie. Zostanie on przypisany do zmiennej `$dbc`, dzięki której, wywołując funkcje dotyczące MySQL-a, będę mógł precyzyjnie określić, o które połączenie mi chodzi.

Wywołanie funkcji zostało poprzedzone operatorem wyłączającym komunikaty o błędach (`@`). Dzięki temu komunikat o ewentualnym błędzie nie zostanie wyświetlony w przeglądarce (co w tym konkretnym przypadku jest wskazane, ponieważ błąd zostanie obsłużony za pomocą klauzuli `OR die()`).

Ciąg dalszy na następnej stronie.

Listing 9.2. Skrypt mysqli_connect.php będzie wykorzystywany przez wszystkie pozostałe skrypty aplikacji. Skrypt ten nawiązuje połączenie z serwerem MySQL, wybiera bazę danych i ustawia kodowanie

```
<?php # Listing 8.2 - mysqli_connect.php

// Plik zawiera informacje potrzebne do uzyskania dostępu do bazy danych.
// Tworzy połączenie z serwerem MySQL, wybiera bazę danych
// i ustawia kodowanie.

// Tworzymy stałe z informacjami potrzebnymi do nawiązania połączenia z bazą.
define ('DB_USER', 'uzytkownik');
define ('DB_PASSWORD', 'haslo');
define ('DB_HOST', 'localhost');
define ('DB_NAME', 'sitename');

// nawiązujemy połączenie
$dbc = @mysqli_connect (DB_HOST, DB_USER, DB_PASSWORD, DB_NAME)
   OR die ('Brak połączenia z serwerem MySQL: ' . mysqli_connect_error() );

// określamy kodowanie
mysqli_set_charset($dbc, 'utf8');
```

Jeśli funkcja `mysqli_connect()` nie zdoła zwrócić identyfikatora zasobu, to wykonana zostanie klauzula `OR die()` (ponieważ pierwsza część warunku będzie mieć wartość `FALSE`, zatem konieczne będzie wykonanie drugiej).
Jak wyjaśniłem w poprzednim rozdziale, funkcja `die()` kończy działanie skryptu. Parametrem tej funkcji może być łańcuch, który zostanie wyświetlony przez przeglądarkę. W naszym przykładzie łańcuch ten jest kombinacją komunikatu *Brak połączenia z serwerem MySQL* i szczegółowej informacji o błędzie MySQL (patrz A). Rozwiązanie takie ułatwia usuwanie błędów pojawiających się w trakcie tworzenia witryny.

4. Ustaw kodowanie:

```
mysqli_set_charset($dbc, 'utf8');
```

Ostatnią operacją wykonywaną w tym skrypcie jest ustawienie sposobu kodowania, który będzie używany w całej przyszłej komunikacji z serwerem MySQL.

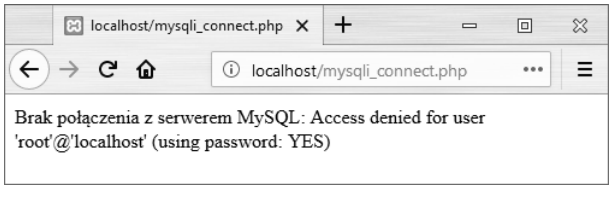
5. Zapisz plik pod nazwą *mysqli_connect.php*.

Ponieważ plik ten zawiera poufne informacje związane z dostępem do bazy danych, zastosowałem w jego przypadku rozszerzenie *.php*. Dzięki temu nawet jeśli użytkownik wykona go w swojej przeglądarce, to nie zobaczy rzeczywistej zawartości strony.

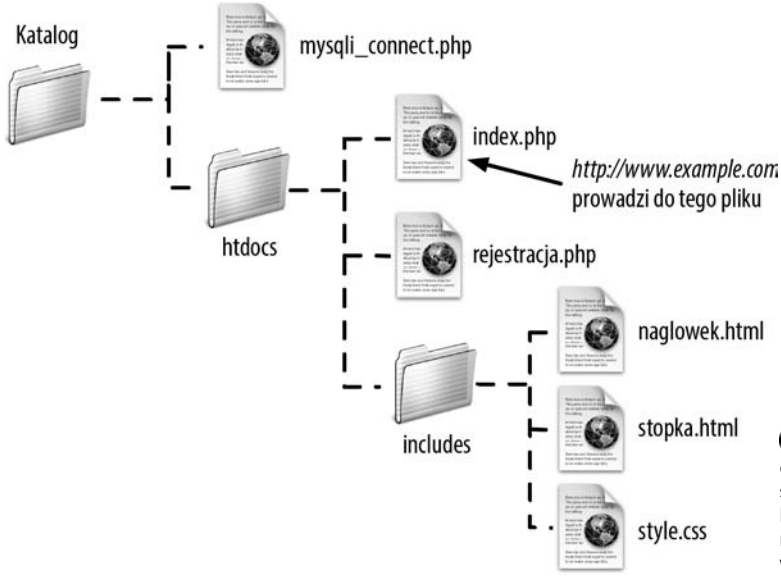
Być może zwróciłeś także uwagę, że w tym skrypcie brakuje zamykającego znacznika PHP, czyli `?>`. To dopuszczalne (o ile tylko skrypt kończy się blokiem kodu PHP) i korzystne ze względów, które wyjaśnię w kolejnych rozdziałach.

6. Wgraj plik na serwer, najlepiej do katalogu leżącego poza strukturą katalogów zawierających strony WWW (patrz B).

Ponieważ plik ten zawiera poufne informacje, to powinien być przechowywany w bezpiecznym miejscu. Najlepiej, jeśli umieścisz go w jednym z katalogów powyżej struktury katalogów zawierających strony WWW.



A Jeżeli w trakcie wykonywania skryptu pojawi się problem z nawiązaniem połączenia z serwerem MySQL, wyświetlony zostanie odpowiedni komunikat, a wykonanie skryptu zostanie zakończone



B Wizualna reprezentacja dokumentów znajdujących się na serwerze WWW. Plik mysqli_connect.php nie jest przechowywany w katalogu głównym (htdocs)

C Jeżeli skrypt nawiązujący połączenie z bazą danych działa prawidłowo, powinien zwracać pustą stronę WWW (ponieważ nie generuje on w ogóle kodu HTML)

Dzięki temu nie będzie on dostępny z przeglądarki internetowej. Patrz ramka „Zarządzanie dokumentami".

7. Umieść tymczasowo kopię tego pliku w katalogu przeznaczonym na strony WWW i przetestuj skrypt w przeglądarce internetowej (patrz C).

   Aby móc przetestować skrypt, będziesz chciał umieścić jego kopię w takim miejscu na serwerze, do którego można się odwoływać za pośrednictwem przeglądarki internetowej. Jeżeli skrypt działa prawidłowo, zobaczysz pustą stronę WWW (patrz C). Jeżeli natomiast ujrzysz komunikat *Access denied...* lub podobny (patrz A), będzie to oznaczało, że wprowadzona przez Ciebie nazwa użytkownika, nazwa hosta lub hasło są nieprawidłowe albo że nie masz wystarczających uprawnień, aby połączyć się z daną bazą.

   Usuń tymczasową kopię pliku z katalogu publicznego.

Zarządzanie dokumentami

O planowaniu struktury witryny mówiłem już w rozdziale 3., gdy tworzyłem pierwszą aplikację internetową. Teraz, gdy zacznę wykorzystywać skrypt nawiązujący połączenie z bazą, zagadnienie to nabierze jeszcze większego znaczenia.

Gdyby informacje potrzebne do połączenia się z bazą danych (nazwa użytkownika, hasło, nazwa hosta i nazwa bazy danych) wpadły w niepowołane ręce, mogłyby zostać użyte do wykradzenia informacji lub zniszczenia całej zawartości bazy. Dlatego też nie ma żadnej przesady w chronieniu pliku *mysqli_connect.php*.

Najważniejszym zaleceniem odnośnie do bezpieczeństwa jest przechowywanie tego rodzaju plików poza katalogiem przeznaczonym na strony internetowe. Jeżeli na przykład folder *htdocs* widoczny na rysunku B jest właśnie takim katalogiem (innymi słowy, prowadzi do niego adres podobny do *www.example.com*), to przechowywanie skryptu *mysqli_ connect.php* poza poddrzewem tego katalogu gwarantuje, że nikt nigdy nie dostanie się do tego pliku za pomocą przeglądarki. Co prawda, z założenia kod źródłowy PHP nie jest nigdy przesyłany bezpośrednio do przeglądarki (tylko generowana przez niego strona), ale ostrożności nigdy za wiele. Zatem jeśli nie masz uprawnień, aby umieszczać pliki poza katalogiem *htdocs*, nie oznacza to jeszcze końca świata, ale jest mniej bezpieczne.

Zalecam także stosowanie dla tego rodzaju skryptów rozszerzenia *.php*. Prawidłowo skonfigurowany serwer po prostu wykona ten kod, a nie będzie wyświetlał go na ekranie. Natomiast przy bezpośrednim odwołaniu do pliku *.inc* jego zawartość zostałaby wyświetlona w przeglądarce.

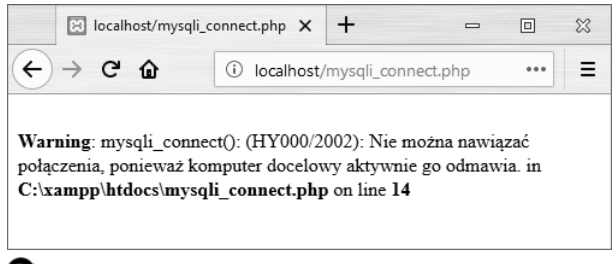
Wskazówki

- W Twoich skryptach PHP powinny zadziałać te same wartości, których używałeś w rozdziale 5. do zalogowania się do monitora mysql.
- Jeżeli otrzymasz komunikat o błędzie mówiący, że `mysqli_connect()` jest niezdefiniowaną funkcją, będzie to oznaczało, że obsługa MySQL-a nie została wkompilowana w Twoją wersję PHP. Informacje na temat instalacji znajdziesz w dodatku A.
- Jeżeli po uruchomieniu skryptu zobaczysz komunikat o błędzie *Can't connect...* (patrz D), to prawdopodobnie serwer MySQL nie został uruchomiony.
- Na rysunku E przedstawiona została sytuacja wystąpienia błędu, gdy wywołanie funkcji `mysqli_connect()` nie zostało poprzedzone operatorem `@`.
- Jeśli nawiązując połączenie z serwerem MySQL, nie chcesz od razu wybrać bazy danych, możesz pominąć ostatni argument wywołania funkcji `mysqli_connect()`:

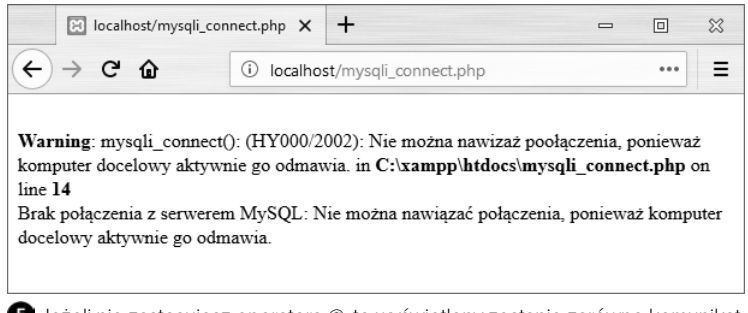
```
$dbc = mysqli_connect (host,
➥użytkownik, hasło);
```

Następnie, gdy będzie to konieczne, wybierz bazę danych za pomocą wywołania

```
mysqli_select_db($dbc, nazwa_bazy);
```



D Innym powodem, dla którego PHP nie połączy się z serwerem MySQL (oprócz nieprawidłowych: nazwy użytkownika, hasła lub informacji o serwerze), może być to, że ten ostatni nie został w ogóle uruchomiony



E Jeżeli nie zastosujesz operatora @, to wyświetlony zostanie zarówno komunikat o błędzie PHP, jak i komunikat przekazany funkcji die()

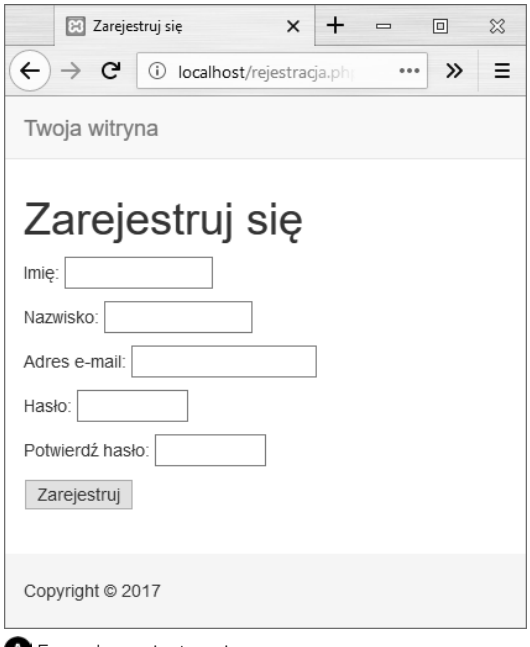
Wykonywanie prostych zapytań

Gdy uda Ci się już połączyć z wybraną bazą danych, możesz rozpocząć wykonywanie zapytań. Mogą to być proste operacje typu *wstawienie*, *aktualizacja* lub *usunięcie rekordu*, ale również złączenia zwracające wiele rekordów naraz. Niezależnie od typu polecenia SQL wykonuje się zawsze przy użyciu funkcji `mysqli_query()`:

```
$wynik = mysqli_query(dbc, zapytanie);
```

Pierwszym argumentem tej funkcji jest połączenie z serwerem MySQL, a drugim samo zapytanie. W przypadku większych skryptów PHP zazwyczaj zapisuję zapytanie w odrębnej zmiennej, o nazwie `$query` lub po prostu `$q`. Wtedy wykonanie zapytania będzie wyglądać jak poniżej:

```
$wynik = mysqli($dbc, $q);
```



A Formularz rejestracyjny

W przypadku prostych zapytań, które nie zwracają rekordów, takich jak `INSERT`, `UPDATE`, `DELETE` itp., w zmiennej `$wynik` zapisywana jest wartość `TRUE` lub `FALSE`, w zależności od tego, czy wykonanie zapytania przebiegło pomyślnie. Pamiętaj, że „pomyślne wykonanie" oznacza brak błędów podczas realizacji polecenia; nie oznacza to wcale, że zapytanie zwróciło oczekiwane wyniki — to będziesz musiał sprawdzić.

Jeżeli zaś chodzi o bardziej złożone zapytania, zwracające rekordy (z których najważniejsze są zapytania `SELECT`), `$wynik` przechowuje wszystkie zwrócone rekordy lub wartość `FALSE` — jeśli pojawił się jakiś błąd. Zatem, aby sprawdzić, czy zapytanie zakończyło się powodzeniem, możesz zastosować instrukcję warunkową:

```
$wynik = mysqli($dbc, $q);
if ($wynik) { // Zapytanie wykonane!
```

Jeśli natomiast wykonanie zapytanie zakończyło się niepowodzeniem, musiał wystąpić jakiś błąd MySQL. Aby się o nim dowiedzieć, wywołaj funkcję `mysqli_error()`:

```
echo mysqli_error($dbc);
```

Jedynym argumentem tej funkcji jest połączenie z bazą danych.

Po zakończeniu wszystkich operacji na bazie danych można wykonać ostatnią, opcjonalną czynność, to znaczy zamknąć połączenie z serwerem:

```
mysqli_close($dbc);
```

Wywołanie tej funkcji nie jest konieczne, ponieważ PHP automatycznie zamknie połączenie, kończąc wykonywanie skryptu. Jednak dobra praktyka programistyczna nakazuje użyć tej funkcji i jawnie zakończyć połączenie.

Aby zademonstrować omawiany proces, napiszę kolejny skrypt rejestracji użytkownika. Będzie on wyświetlać odpowiedni formularz (patrz A), obsługiwać go, przeprowadzać weryfikację wprowadzonych danych i umieszczać je w tabeli `users` bazy *sitename*.

Ciąg dalszy na następnej stronie.

Muszę ostrzec, że przedstawiony tu skrypt jest (w zależności od używanej wersji PHP oraz stosowanych ustawień) niebezpieczny, co wyjaśnię w dalszej części rozdziału.

Aby wykonywać proste zapytania:

1. W edytorze tekstów lub IDE utwórz nowy skrypt PHP (listing 8.3), któremu nadasz nazwę *rejestracja.php*.

```
<?php # Listing 9.3 - rejestracja.php
$page_title = 'Zarejestruj się!';
include ('includes/naglowek.html');
```

   Skrypt ten wykorzystuje podstawowe koncepcje omówione w rozdziale 3., takie jak dołączanie plików, wyświetlanie i obsługa formularza przez tę samą stronę oraz zapamiętywanie wprowadzonych danych.

2. Utwórz wyrażenie warunkowe obsługujące dane przekazane z formularza i zainicjalizuj tablicę `$errors`.

```
if ($_SERVER['REQUEST_METHOD'] ==
    'POST') {
   $errors = [];
```

   Ten skrypt będzie wyświetlał i obsługiwał formularz HTML. Powyższa instrukcja warunkowa sprawdza rodzaj użytego żądania i na jego podstawie określa sposób obsługi formularza (także to rozwiązanie opisałem dokładniej w rozdziale 3.). Zmiennej `$errors` będę używał do gromadzenia komunikatów o błędach (po jednym dla każdego elementu formularza, który nie został poprawnie wypełniony).

3. Sprawdź imię.

```
if (empty($_POST['first_name'])) {
      $errors[] = 'Zapomniałeś
      ↪podać swoje imię!';
} else {
      $fn = trim($_POST['first_name']);
}
```

   Podobnie jak w rozdziale 3., sprawdzam za pomocą funkcji `empty()`, czy pole zostało wypełnione. Jeśli pole nie zostało wypełnione, to do tablicy `$errors` zostanie dodany komunikat o błędzie. W przeciwnym wypadku w zmiennej `$fn` znajdzie się wartość wprowadzona przez użytkownika po usunięciu zbędnych odstępów. Dzięki zastosowaniu tej zmiennej, której nazwa jest oczywiście skrótem od `first_name`, będzie mi znacznie łatwiej sformułować później zapytanie SQL.

4. Sprawdź nazwisko i adres e-mail.

```
if (empty($_POST['last_name'])) {
      $errors[] = 'Zapomniałeś podać
      ↪swoje nazwisko! ';
} else {
      $ln = trim($_POST['last_name']);
}
if (empty($_POST['email'])) {
      $errors[] = 'Zapomniałeś podać
      ↪swój adres e-mail!';
} else {
      $e = trim($_POST['email']);
}
```

   Powyższe wiersze kodu są identyczne jak użyte do sprawdzenia pola zawierającego imię. Jeśli kontrola zawartości pól przebiegnie pomyślnie, to w obu wypadkach zostaną utworzone nowe zmienne.

Ciąg dalszy na stronie 305.

Listing 9.3. Skrypt rejestracyjny dodaje do bazy danych nowy rekord, wykonując polecenie INSERT

```
<?php # Listing 9.3 - rejestracja.php
// Ten skrypt wykonuje polecenie INSERT, by dodać wiersz do tabeli użytkowników.

$page_title = 'Zarejestruj się';
include ('includes/naglowek.html');

// Sprawdzamy, czy formularz został wysłany
if ($_SERVER['REQUEST_METHOD'] == 'POST') {

  $errors = array(); // inicjalizujemy tablicę błędów

  // sprawdzamy imię
  if (empty($_POST['first_name'])) {
    $errors[] = 'Zapomniałeś podać swoje imię!';
  } else {
    $fn = trim($_POST['first_name']);
  }

  // sprawdzamy nazwisko
  if (empty($_POST['last_name'])) {
    $errors[] = 'Zapomniałeś podać swoje nazwisko!';
  } else {
    $ln = trim($_POST['last_name']);
  }

  // sprawdzamy adres e-mail
  if (empty($_POST['email'])) {
    $errors[] = 'Zapomniałeś podać swój adres e-mail!';
  } else {
    $e = trim($_POST['email']);
  }

  // sprawdzamy, czy użytkownik wprowadził hasło i czy w obu polach jest ono takie samo
  if (!empty($_POST['pass1'])) {
    if ($_POST['pass1'] != $_POST['pass2']) {
      $errors[] = 'Za drugim razem wpisałeś inne hasło!';
    } else {
      $p = trim($_POST['pass1']);
    }
  } else {
    $errors[] = 'Zapomniałeś wprowadzić hasło!';
  }

  if (empty($errors)) { // jeśli formularz poprawnie wypełniony...

    // rejestrujemy użytkownika w bazie danych

```

Ciąg dalszy na następnej stronie.

Listing 9.3. Ciąg dalszy

```
48     require('../mysqli_connect.php'); // nawiązujemy połączenie z bazą danych
49
50     // tworzymy zapytanie
51     $q = "INSERT INTO users (first_name, last_name, email, pass,
         registration_date) VALUES ('$fn', '$ln', '$e', SHA2('$p',512), NOW() )";
52     $r = @mysqli_query ($dbc, $q); // wykonujemy zapytanie
53     if ($r) { // jeśli poprawnie wykonane...
54
55       // wyświetlamy komunikat
56       echo '<h1>Dziękujemy!</h1>
57       <p>Zostałeś zarejestrowany. W rozdziale 12. będziesz mógł się zalogować!
         </p><p><br /></p>';
58
59     } else { // jeśli zapytanie nie zostało poprawnie wykonane...
60
61       // wyświetlamy komunikat dla użytkownika
62       echo '<h1>Błąd systemu</h1>
63       <p class="error">Nie zostałeś zarejestrowany z powodu awarii
         naszego systemu.  Przepraszamy za kłopot.</p>';
64
65       // wyświetlamy komunikat uruchomieniowy
66       echo '<p>' . mysqli_error($dbc) . '<br><br>Zapytanie: ' . $q .
         '</p>';
67
68     } // koniec instrukcji if ($r).
69
70     mysqli_close($dbc); // zamykamy połączenie z bazą danych
71
72     // dołączamy stopkę i kończymy działanie skryptu
73     include ('includes/stopka.html');
74     exit();
75
76   } else { // wyświetlamy błędy
77
78     echo '<h1>Błąd!</h1>
79     <p class="error">Wystąpiły następujące błędy:<br>';
80     foreach ($errors as $msg) { // wyświetlamy każdy komunikat o błędzie
81       echo " - $msg<br />\n";
82     }
83     echo '</p><p>Spróbuj jeszcze raz.</p><p><br></p>';
84
85   } // koniec instrukcji if (empty($errors))
86
87 } // koniec głównej instrukcji warunkowej (wysłania formularza)
88 ?>
89 <h1>Zarejestruj się</h1>
90 <form action="rejestracja.php" method="post">
91   <p>Imię: <input type="text" name="first_name" size="15" maxlength="20"
     value="<?php if (isset($_POST['first_name'])) echo $_POST['first_name']; ?>"
     ></p>
92   <p>Nazwisko: <input type="text" name="last_name" size="15" maxlength="40"
     value="<?php if (isset($_POST['last_name'])) echo $_POST['last_name']; ?>"
     ></p>
```

Ciąg dalszy na następnej stronie.

5. Sprawdź hasło.

```
if (empty($_POST['pass1'])) {
   if ($_POST['pass1'] ==
     $_POST['pass2']) {
     $errors[] = 'Za drugim razem
     ↪wpisałeś inne hasło!';
   } else {
      $p = trim($_POST['pass1']);
   }
} else {
    $errors[] = 'Zapomniałeś
    ↪wprowadzić hasło!';
}
```

Aby zweryfikować hasło, sprawdzam najpierw, czy w odpowiednie pole formularza wprowadzono w ogóle jakąś wartość. Następnie badam, czy wartość `$pass1` jest taka sama jak `$pass2` (a więc, czy użytkownik wprowadził dwukrotnie to samo hasło).

6. Sprawdź, czy można zarejestrować użytkownika

```
if (empty($errors)) {
```

Jeżeli wprowadzone przez użytkownika dane spełniają wszystkie warunki, główny warunek skryptu również jest spełniony i można bezpiecznie kontynuować. Jeśli nie, powinien zostać wyświetlony odpowiedni komunikat o błędzie (patrz punkt 11.), a użytkownik powinien otrzymać jeszcze jedną szansę na zarejestrowanie się.

7. Dodaj użytkownika do bazy danych.

```
require_once('../mysqli_connect.php');
```

Pierwsza linijka kodu wstawia do skryptu zawartość pliku *mysqli_connect.php*, tworząc tym samym połączenie z MySQL-em i wybierając bazę danych. Możliwe, że będziesz musiał wprowadzić inną ścieżkę do pliku, zgodną z jego lokalizacją na Twoim serwerze (w obecnej postaci instrukcja zakłada, że plik *mysqli_connect.php* znajduje się w katalogu nadrzędnym względem katalogu zawierającego wykonywany skrypt).

8. Dodaj użytkownika do bazy danych.

```
$q = "INSERT INTO users (first_name,
↪last_name, email, pass,
↪registration_date) VALUES ('$fn',
↪'$ln', '$e', SHA1('$p'), NOW() )";
$r = @mysqli_query ($dbc, $q);
```

Samo zapytanie SQL jest bardzo podobne do tych, które przedstawiłem w rozdziale 5. Funkcja `SHA2()` wykorzystywana jest do szyfrowania hasła, a `NOW()` ustawia czas rejestracji na bieżącą chwilę. (W rozdziale 13., „Zabezpieczenia”, przedstawię metodę szyfrowania i porównywania haseł na potrzeby rejestracji, której można używać w skryptach PHP).

Ciąg dalszy na następnej stronie.

Listing 9.3. Ciąg dalszy

```
93   <p>Adres e-mail: <input type="text" name="email" size="20" maxlength="80"
     value="<?php if (isset($_POST['email'])) echo $_POST['email']; ?>"  > </p>
94   <p>Hasło: <input type="password" name="pass1" size="10" maxlength="20"
     value="<?php if (isset($_POST['pass1'])) echo $_POST['pass1']; ?>" ></p>
95   <p>Potwierdź hasło: <input type="password" name="pass2" size="10"
     maxlength="20" value="<?php if (isset($_POST['pass2']))
     echo $_POST['pass2']; ?>" ></p>
96   <p><input type="submit" name="submit" value="Zarejestruj" ></p>
97 </form>
98 <?php include ('includes/stopka.html'); ?>
```

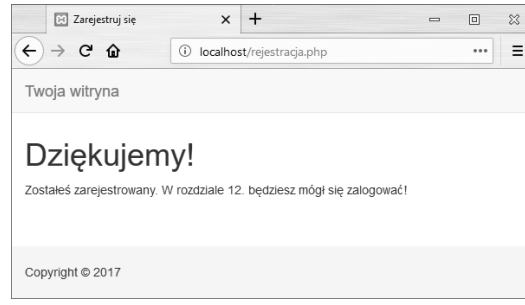
Po przypisaniu zapytania do zmiennej wykonuję je za pośrednictwem funkcji `mysqli_query()`. Przesyła ona polecenia języka SQL na serwer MySQL. Podobnie jak w skrypcie *mysqli_connect.php*, wywołanie funkcji `mysqli_query()` zostało poprzedzone operatorem `@`, aby zapobiec wyświetlaniu komunikatów o błędach w przeglądarce. Jeśli wystąpi błąd, to zostanie on obsłużony w następnym punkcie.

9. Wyświetl informację, czy rejestracja przebiegła pomyślnie.

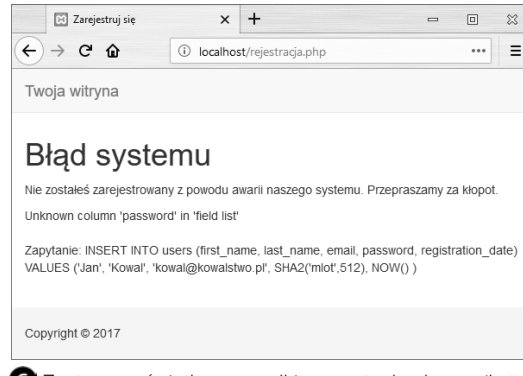
```
if ($r) {
  echo '<h1>Dziękujemy!</h1>
  <p>Zostałeś zarejestrowany.
  W rozdziale 12. będziesz mógł
  się zalogować!</p><p>
  <br></p>';
} else {
  echo '<h1>Błąd systemu</h1>
  <p class="error">Nie zostałeś
  zarejestrowany z powodu awarii
  naszego systemu.  Przepraszamy
  za kłopot.</p>';
  echo '<p>' . mysqli_error($dbc)
  . '<br><br>Zapytanie: '
  . $q . '</p>';
}
```

Zmienna `$r`, do której przypisywana jest wartość zwracana przez zapytanie `mysqli_query()`, może zostać użyta w wyrażeniu warunkowym, które sprawdzi, czy polecenie SQL zostało wykonane pomyślnie.

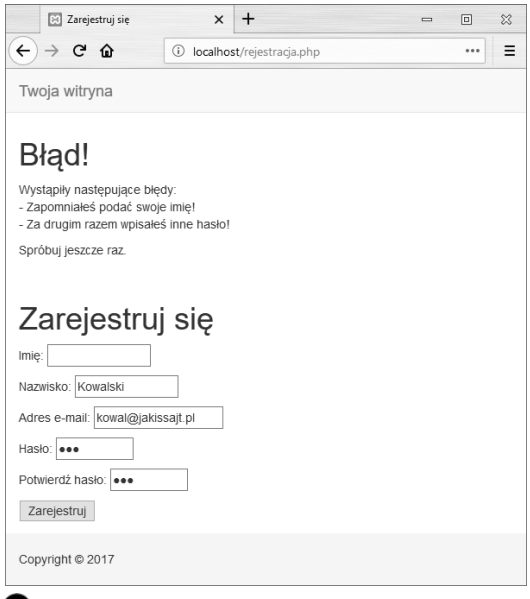
Jeżeli zmienna `$r` ma wartość `TRUE`, wyświetlany jest komunikat z podziękowaniem za rejestrację (patrz B). Jeżeli zmienna `$r` ma wartość `FALSE`, wyświetlone zostają komunikaty o błędach. Dla potrzeb procesu usuwania błędów komunikat składa się z wyników zwróconych przez funkcję `mysqli_error()` oraz tekstu wykonanego zapytania (patrz C). Takich informacji lepiej jednak nie wyświetlać na witrynie produkcyjnej.



B Jeżeli operacja zarejestrowania użytkownika w bazie danych przebiegnie pomyślnie, zostanie wyświetlony ten komunikat



C Zostaną wyświetlone wszelkie ewentualne komunikaty o błędach wygenerowane przez zapytanie oraz samo zapytanie



D Każdy błąd wykryty podczas weryfikacji danych jest raportowany, a użytkownik może ponownie wypełnić formularz

10. Zamknij połączenie z bazą danych i dokończ szablon HTML.

```
mysqli_close();
include('includes/stopka.html');
exit();
```

Zamknięcie połączenia nie jest wymagane, ale stanowi przykład dobrej praktyki programistycznej. Następnie dołączona zostaje stopka, a wykonanie skryptu kończy się (przez wywołanie funkcji `exit()`). Gdybym pominął te dwa ostatnie wiersze kodu, formularz rejestracji zostałby ponownie wyświetlony (co nie jest konieczne, jeśli rejestracja zakończyła się pomyślnie).

11. Zamknij główną instrukcję warunkową i wyświetl ewentualne komunikaty o błędach.

```
  } else { // wyświetlamy błędy
    echo '<h1>Błąd!</h1>
    <p class="error">Wystąpiły
      następujące błędy:<br>';
    foreach ($errors as $msg) {
      // wyświetlamy każdy komunikat o błędzie
      echo " - $msg<br>\n";
    }
    echo '</p><p>Spróbuj jeszcze
    raz.</p><p><br></p>';
  } // koniec instrukcji if (empty($errors))
} // koniec głównej instrukcji warunkowej
  (wysłania formularza)
```

Klauzula `else` zostaje wykonana, jeśli wystąpiły błędy. Wszystkie komunikaty o błędach zostają wyświetlone za pomocą pętli `foreach` (patrz D).

Ostatni nawias klamrowy zamyka główne wyrażenie warunkowe związane z wysłaniem danych formularza. Wyrażenie to ma prostą postać if, a nie `if-else`, aby formularz zawierał zawsze dane wcześniej wprowadzone (patrz również rozdział 3.).

Ciąg dalszy na następnej stronie.

12. Zamknij kod PHP i rozpocznij formularz HTML.

```
?>
<h1>Zarejestruj się</h1>
<form action="register.php"
  method="post">
  <p>Imię: <input type="text"
    name="first_name" size="15"
    maxlength="20" value="<?php if
    (isset($_POST['first_name']))
    echo $_POST['first_name']; ?>"
    /></p>
  <p>Nazwisko: <input type="text"
    name="last_name" size="15"
    maxlength="40" value="<?php if
   (isset($_POST['last_name']))
   echo $_POST['last_name']; ?>"
   /></p>
```

Formularz ten jest bardzo prosty, zawiera po jednym polu tekstowym dla każdej kolumny tabeli `users` (z wyjątkiem kolumn `user_id` oraz `registration_date`, które są wypełniane automatycznie). Dane wprowadzone w poszczególnych polach są zapisywane dzięki zastosowaniu kodu w postaci

```
value="<?php if
  (isset($_POST['first_name']))
echo
  $_POST['first_name']; ?>"
```

Zalecam stosowanie takich samych nazw pól formularza jak odpowiadających im kolumn bazy danych. Dobrze będzie, jeśli pola formularza będą mieć również taką samą długość jak kolumny bazy danych. Oba te rozwiązania pozwalają ograniczyć możliwość pomyłek.

13. Dokończ formularz HTML.

```
  <p>Adres e-mail: <input type="text"
    name="email" size="20"
    maxlength="80" value="<?php if
    (isset($_POST['email']))
     echo $_POST['email']; ?>" /></p>
  <p>Hasło: <input type="password"
    name="pass1" size="10"
    maxlength="20" value="<?php if
    (isset($_POST['pass1']))
    echo $_POST['pass1']; ?>" ></p>
  <p>Potwierdź hasło: <input
    type="password" name="pass2"
    size="10" maxlength="20"
    value="<?php if
    (isset($_POST['pass2']))
    echo $_POST['pass2']; ?>"></p>
  <p><input type="submit"
    name="submit" value="Zarejestruj"
    /></p>
</form>
```

Ta część formularza jest podobna do stworzonej w punkcie 12., lecz dodatkowo został w niej umieszczony przycisk przesyłający formularz.

W przypadku wprowadzania haseł nie muszę się stosować do sugestii dotyczącej ograniczania długości pola za pomocą atrybutu `maxlength` (patrz punkt 12.), ponieważ hasła są szyfrowane za pomocą funkcji `SHA2()`, która zawsze zwraca łańcuch o stałej długości. A ponieważ są dwa pola służące do podania hasła, zatem oba nie mogą jednocześnie mieć tej samej nazwy co kolumna w bazie danych.

14. Zakończ stronę stopką HTML.

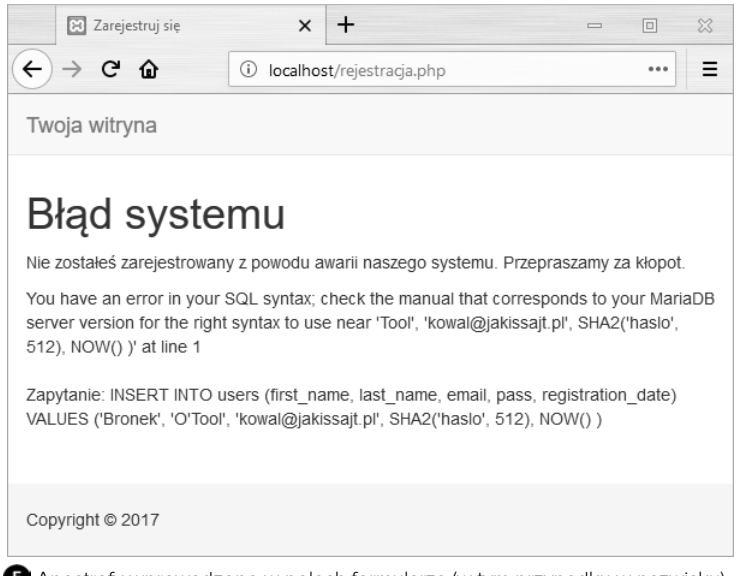
```
<?php
include ('includes/stopka.html');
?>
```

15. Zapisz plik pod nazwą *rejestracja.php*, wgraj go na serwer i przetestuj w przeglądarce internetowej.

    Zwróć uwagę, że jeśli w którymś polu formularza wprowadzisz znak apostrofu, to prawdopodobnie spowoduje to błędy podczas wykonywania polecenia SQL (patrz E). Sposób rozwiązania tego problemu zostanie pokazany w podrozdziale „Bezpieczeństwo zapytań".

Wskazówki

- Po wywołaniu skryptu możesz zawsze sprawdzić, czy zadziałał on jak trzeba, wywołując monitor mysql lub aplikację phpMyAdmin i oglądając zmiany wprowadzone w tabeli.
- Gdy korzystasz z PHP, nie powinieneś kończyć zapytań SQL średnikiem, tak jak robiłeś to w monitorze mysql. Jest to niegroźny, choć często popełniany błąd w pracy z MySQL-em. Jednak w przypadku innych baz danych, takich jak na przykład Oracle, uniemożliwia on wykonanie zapytania.
- Funkcja `mysqli_query()` zwraca wartość `TRUE`, jeżeli udało się wykonać jakieś zapytanie na bazie danych. Nie oznacza to jeszcze, że efekty tej operacji są zgodne z Twoimi oczekiwaniami. W kolejnych skryptach zobaczysz, jak można bardziej precyzyjnie określać efekty wykonania zapytania.
- Nie musisz koniecznie tworzyć zmiennej `$q` tak jak ja. Możesz wstawić treść zapytania bezpośrednio do funkcji `mysqli_query()`. Jednak w miarę jak Twoje zapytania będą się stawać coraz bardziej złożone, dojdziesz do momentu, w którym użycie zmiennej będzie jedynym rozwiązaniem.
- Za pomocą funkcji `mysqli_query()` można wykonać praktycznie każde zapytanie, które działa w monitorze mysql.
- Za pomocą funkcji `mysqli_multi_query()` można wykonywać wiele zapytań za jednym razem. Jednak składnia jest wtedy dość skomplikowana, zwłaszcza gdy zapytania zwracają wyniki.



E Apostrofy wprowadzone w polach formularza (w tym przypadku w nazwisku) spowodują konflikt z apostrofami używanymi do wyodrębniania wartości w zapytaniach SQL

Odczytywanie wyników zapytania

W poprzednim podrozdziale pokazałem Ci, jak można wykonywać proste zapytania na bazie danych MySQL. Przez *proste zapytania* rozumiem tu te, które zaczynają się od słów `INSERT`, `UPDATE`, `DELETE` lub `ALTER`. Mają one wszystkie jedną wspólną cechę — nie zwracają żadnych danych, a jedynie informację, czy udało się je wykonać. Z kolei zapytania `SELECT` generują informacje (na przykład rekordy), które muszą być obsłużone przez funkcje PHP.

Podstawowym narzędziem, które to umożliwia, jest funkcja `mysqli_fetch_array()`. Pobiera ona zmienną, w której zapisano wynik zapytania (ja nadaję jej nazwę `$r`), i zwraca po jednym wierszu danych naraz w formacie tablicowym. Funkcję tę wykorzystuje się w pętli, uzyskując za każdym razem dostęp do kolejnego rekordu. Podstawowa konstrukcja językowa umożliwiająca odczytanie każdego rekordu zwróconego przez zapytanie wygląda tak:

```
while ($row = mysqli_fetch_array($r)) {
   // Zrób coś z $row.
}
```

Praktycznie zawsze będziesz używać pętli `while`, aby pobrać wyniki zapytania `SELECT`.

Funkcja `mysqli_fetch_array()` pobiera też opcjonalny parametr określający rodzaj zwracanej tablicy (asocjacyjna, indeksowana lub dwuformatowa). Tablica asocjacyjna pozwala odwoływać się do wartości przechowywanych w kolumnach po ich nazwach, podczas gdy tablica indeksowana wymaga stosowania oznaczeń liczbowych (gdzie `0` oznacza pierwszą kolumnę). Każdy parametr jest definiowany przez jedną ze stałych wymienionych w tabeli 9.1, przy czym domyślnie używany jest `MYSQLI_BOTH`. Zazwyczaj korzystam z ustawienia `MYSQLI_NUM`, które jest nieco szybsze (i wymaga mniej pamięci) od pozostałych. Inni programiści wolą `MYSQLI_ASSOC`, ponieważ jest ono bardziej precyzyjne i działa nawet po zmianie struktury tabeli (`$rekord['kolumna']` zamiast `$rekord[3]`).

Gdy korzystasz z funkcji `mysqli_fetch _array()`, możesz zwolnić zasoby, które nie są Ci już potrzebne:

```
mysqli_free_result($r);
```

To wyrażenie zwalnia pamięć zajmowaną przez zmienną `$r`. Podobnie jak `mysqli_close()` jest ono opcjonalne, ponieważ PHP automatycznie zwalnia wszystkie zasoby po dojściu do końca skryptu. Jednak stosowanie go jest dobrą praktyką programistyczną.

Aby pokazać Ci, jak należy obsługiwać wyniki zapytania, utworzę prosty skrypt wyświetlający listę aktualnie zarejestrowanych użytkowników.

Aby odczytać wyniki zapytania:

1. W edytorze tekstów lub IDE utwórz nowy dokument PHP (listing 9.4).

```
<?php # Skrypt 9.4 - pokaz_uzytkownikow.php
$page_title = 'Wykaz obecnych
  użytkownikow';
include ('includes/naglowek.html');
echo '<h1>Zarejestrowani użytkownicy</h1>'
```

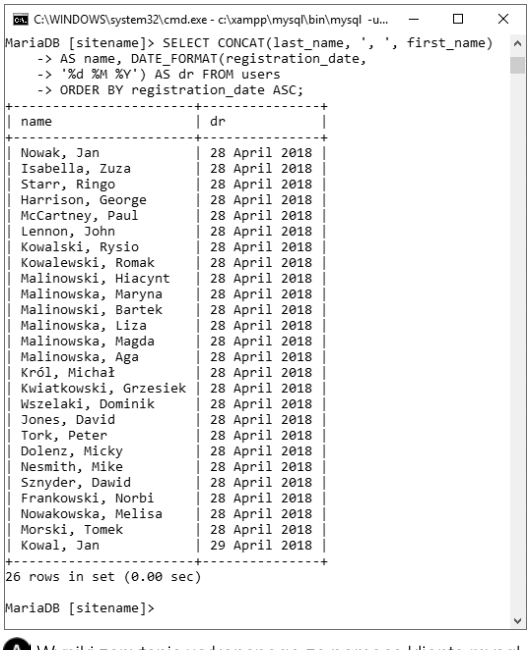
Tabela 9.1. Stałe używane w funkcji mysql_fetch_array()

Stała	Przykład
`MYSQL_ASSOC`	`$rzad['kolumna']`
`MYSQL_NUM`	`$rzad[0]`
`MYSQL_BOTH`	`$rzad[0]` lub `$rzad['kolumna']`

2. Połącz się z bazą danych i wykonaj na niej zapytanie.

```
require('../mysqli_connect.php');
$q = "SELECT CONCAT(last_name, ',  ',
➥first_name) AS name,
➥DATE_FORMAT(registration_date, '%d  %M %Y')
➥AS dr FROM users ORDER BY
➥registration_date ASC";
$r = @mysqli_query ($dbc, $q);
```

Pokazane tu zapytanie zwróci dwie kolumny (patrz A): nazwiska użytkowników (w formacie *Nazwisko*, *Imię*) i daty ich rejestracji (w formacie *Miesiąc DD*, *RRRR*). Ponieważ obie kolumny zostały sformatowane przy użyciu funkcji MySQL-a, to zwróconym wynikom nadałem aliasy (`name` i `dr`). Jeśli składnia wydaje Ci się skomplikowana, to wróć do lektury rozdziału 5.



```
C:\WINDOWS\system32\cmd.exe - c:\xampp\mysql\bin\mysql -u...
MariaDB [sitename]> SELECT CONCAT(last_name, ', ', first_name)
    -> AS name, DATE_FORMAT(registration_date,
    -> '%d %M %Y') AS dr FROM users
    -> ORDER BY registration_date ASC;
+------------------------+--------------+
| name                   | dr           |
+------------------------+--------------+
| Nowak, Jan             | 28 April 2018 |
| Isabella, Zuza         | 28 April 2018 |
| Starr, Ringo           | 28 April 2018 |
| Harrison, George       | 28 April 2018 |
| McCartney, Paul        | 28 April 2018 |
| Lennon, John           | 28 April 2018 |
| Kowalski, Rysio        | 28 April 2018 |
| Kowalewski, Romak      | 28 April 2018 |
| Malinowski, Hiacynt    | 28 April 2018 |
| Malinowska, Maryna     | 28 April 2018 |
| Malinowski, Bartek     | 28 April 2018 |
| Malinowska, Liza       | 28 April 2018 |
| Malinowska, Magda      | 28 April 2018 |
| Malinowska, Aga        | 28 April 2018 |
| Król, Michał           | 28 April 2018 |
| Kwiatkowski, Grzesiek  | 28 April 2018 |
| Wszelaki, Dominik      | 28 April 2018 |
| Jones, David           | 28 April 2018 |
| Tork, Peter            | 28 April 2018 |
| Dolenz, Micky          | 28 April 2018 |
| Nesmith, Mike          | 28 April 2018 |
| Sznyder, Dawid         | 28 April 2018 |
| Frankowski, Norbi      | 28 April 2018 |
| Nowakowska, Melisa     | 28 April 2018 |
| Morski, Tomek          | 28 April 2018 |
| Kowal, Jan             | 29 April 2018 |
+------------------------+--------------+
26 rows in set (0.00 sec)

MariaDB [sitename]>
```

A Wyniki zapytania wykonanego za pomocą klienta mysql

3. Wyświetl wyniki zapytania.

```
if ($r) {
  echo '<table width="60%">
  <thead>
  <tr>
    <th align="left">Nazwisko</th>
    <th align="left">Data
      rejestracji</th>
  </tr>
  </thead>
  <tbody>
';
```

Jeśli zmienna `$r` będzie zawierać wartość `TRUE`, będzie to oznaczać, że zapytanie zostało wykonane poprawnie i można wyświetlić wyniki. Wyświetlanie wyników zaczniemy od rozpoczęcia tabeli HTML i wyświetlenia jej nagłówka.

4. Pobierz i wyświetl każdy z pobranych rekordów:

```
while ($row =
  mysqli_fetch_array($r,
  MYSQLI_ASSOC)) {
  echo '<tr><td align="left">'
  . $row['name'] . '</td><td
  align="left">' . $row['dr']
  . '</td></tr>
';
}
```

Następnie w pętli pobierzemy każdy z rekordów przy użyciu funkcji `mysqli_fetch_array()` i wyświetlimy go na stronie. Zwróć uwagę, że wewnątrz pętli `while` odwołuję się do każdej zwróconej wartości, używając właściwego aliasu: `$row['name']` i `$row['dr']`. Nie mógłbym odwołać się do nich jako `$row['first_name']` lub `$row['date_registered']`, ponieważ nie zostały zwrócone pola o takich nazwach (patrz A).

Ciąg dalszy na stronie 313.

Listing 9.4. Skrypt pokaz_uzytkownikow.php wykonuje statyczne zapytanie na bazie danych i wyświetla wszystkie zwrócone rekordy

```
<?php # Listing 9.4 - pokaz_uzytkownikow.php
// Ten listing odczytuje wszystkie rekordy z tabeli users.

$page_title = 'Zobacz zarejestrowanych użytkowników';
include ('includes/naglowek.html');

// wyświetlamy nagłówek strony
echo '<h1>Zarejestrowani użytkownicy</h1>';

require_once ('../mysqli_connect.php'); // nawiązujemy połączenie z bazą danych

// tworzymy zapytanie
$q = "SELECT CONCAT(last_name, ', ', first_name) AS name, DATE_FORMAT(registration_date,
'%d %M %Y') AS dr FROM users ORDER BY registration_date ASC";
$r = @mysqli_query ($dbc, $q); // wykonujemy zapytanie

if ($r) { // jeżeli zapytanie zostało wykonane poprawnie, pokazujemy każdy rekord

  // nagłówek tabeli
  echo '<table width="60%">
  <thead>
  <tr>
    <th align="left">Nazwisko</th>
    <th align="left">Data rejestracji</th>
  </tr>
  </thead>
  <tbody>
';

  // Pobieramy i wyświetlamy wszystkie rekordy
  while ($row = mysqli_fetch_array($r, MYSQLI_ASSOC)) {
    echo '<tr><td align="left">' . $row['name'] . '</td><td align="left">'.
    $row['dr'] . '</td></tr>
    ';
  }

  echo '</tbody></table>'; // zamykamy tabelę

  mysqli_free_result ($r); // zwalniamy zasoby

} else { // jeżeli zapytanie nie zostało wykonane pomyślnie...

  // wyświetlamy komunikat dla użytkownika
  echo '<p class="error">Nie możemy wyświetlić listy użytkowników
  ze względu na awarię systemu.  Przepraszamy za kłopot.</p>';

  // wyświetlamy uruchomieniowy
  echo '<p>' . mysqli_error($dbc) . '<br><br>Zapytanie: ' . $q . '</p>';

} // koniec instrukcji if ($r).

mysqli_close($dbc); // zamykamy połączenie z bazą danych

include ('includes/stopka.html');
?>
```

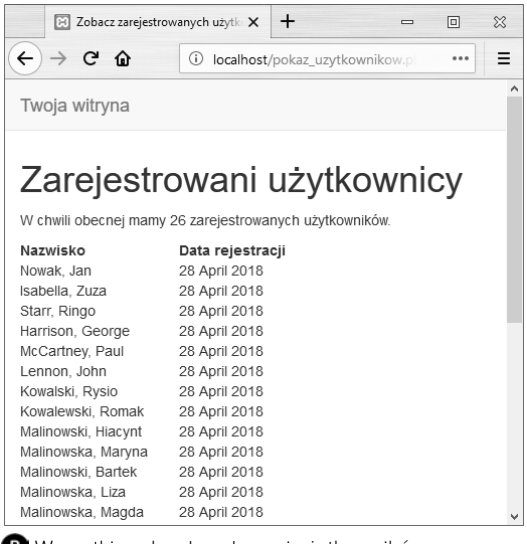
5. Zakończ tabelę HTML i zwolnij zasoby zajęte przez zapytanie.

```
echo '</tbody></table>';
mysqli_free_result($r);
```

Jeszcze raz powtórzę, że wywoływanie tej funkcji jest opcjonalne, ale warto się na to zdecydować.

6. Dokończ główną instrukcję warunkową.

```
} else {
  echo '<p class="error">Nie możemy
    wyświetlić listy użytkowników ze
    względu na awarię systemu.
    Przepraszamy za kłopot.</p>';
  echo '<p>' . mysqli_error($dbc)
   . '<br><br>Zapytanie: ' . $q
   . '</p>';
}  // koniec instrukcji if ($r).
```



B Wszystkie rekordy z danymi użytkowników są odczytywane z bazy danych

Podobnie jak w poprzednim przykładzie mamy tutaj dwa komunikaty. Pierwszy jest ogólnym komunikatem, takim jakie wyświetla się użytkownikom działającej witryny. Natomiast drugi jest bardziej szczegółowy, zawiera zarówno informacje o błędzie MySQL jak i tekst zapytania. Ten komunikat przewidziany jest jako pomoc dla programisty w procesie usuwania błędów.

7. Zamknij połączenie z bazą danych i dokończ stronę.

```
mysqli_close($dbc);
include ('includes/stopka.html');
?>
```

8. Zapisz plik pod nazwą *pokaz_uzytkownikow.php*, wgraj go na serwer i przetestuj w przeglądarce internetowej (patrz B).

Wskazówki

- Funkcja `mysqli_fetch_row()` jest odpowiednikiem `mysqli_fetch_array($r, MYSQLI_NUM)`.
- Funkcja `mysqli_fetch_assoc()` jest odpowiednikiem `mysqli_fetch_array ($r, MYSQLI_ASSOC)`.
- Gdy odczytujesz rekordy z bazy danych, stosując tablice asocjacyjne, musisz wprowadzać nazwy kolumn lub aliasy dokładnie tak, jak zostały one zdefiniowane w bazie lub zapytaniu. Innymi słowy, wielkie i małe litery w kluczach są rozróżniane.
- Jeżeli znajdziesz się w sytuacji, w której będziesz musiał wykonać drugie zapytanie w ramach pętli `while`, pamiętaj, aby użyć w nim innych nazw zmiennych (`$r2` i `$row2` zamiast `$r` i `$row`). W przeciwnym wypadku powstaną błędy logiczne.
- Początkujący programiści PHP często mają problemy z pobieraniem wyników zapytania. Zapamiętaj, że najpierw należy wykonać zapytanie, używając funkcji `mysqli_query()`, a następnie zastosować funkcję `mysqli_fetch_array()` do pobrania pojedynczego rekordu. Jeśli wynik zapytania zawiera wiele rekordów, to użyj pętli `while`.

Bezpieczeństwo zapytań

Zapewnienie bazom danych bezpieczeństwa sprowadza się w PHP do trzech zagadnień:

- Ochrony informacji potrzebnych do uzyskania dostępu do bazy.
- Nieujawniania zbyt wielu informacji o bazie danych.
- Zachowania ostrożności przy wstawianiu danych do bazy, zwłaszcza wprowadzonych przez użytkownika.

Pierwszy z tych celów możesz osiągnąć, umieszczając skrypt odpowiedzialny za połączenie z serwerem MySQL poza katalogiem przechowującym strony WWW (patrz C w podrozdziale „Nawiązywanie połączenia z serwerem MySQL" na początku tego rozdziału). W ten sposób nie będzie można podejrzeć jego zawartości za pomocą przeglądarki. Zresztą już o tym wspominałem. Drugi cel możemy osiągnąć, nie pokazując użytkownikowi komunikatów o błędach PHP ani treści zapytań (w dotychczasowych przykładach wyświetlałem te informacje w celach uruchomieniowych; nie praktykuj tego nigdy na witrynach produkcyjnych).

Jeżeli zaś chodzi o trzeci cel, to istnieje kilka kroków, które warto podjąć, wychodząc z założenia, że nie należy ufać danym wprowadzonym przez użytkownika. Po pierwsze, sprawdź, czy zostały wprowadzone dane i czy są one odpowiedniego typu (liczba, łańcuch itp.). Po drugie, możesz korzystać z rozszerzenia Filter (przedstawionego w rozdziale 13.) lub wyrażeń regularnych (przedstawionych w rozdziale 14., „Wyrażenia regularne Perl"), aby upewnić się, że dane wprowadzone przez użytkownika mają taką postać, jak powinny. Po trzecie, możesz rzutować wartości na określone typy, aby zagwarantować, że są liczbami. Czwarty krok polega na stosowaniu funkcji `mysqli_real_escape_string()`, która poprzedza ukośnikiem problematyczne znaki:

```
$dane = mysqli_real_escape_string($dbc, dane);
```

Aby przekonać się, dlaczego stosowanie tej funkcji jest niezbędne, spójrz na rysunek E w podrozdziale „Wykonywanie prostych zapytań" wcześniej w tym rozdziale. Umieszczenie znaku apostrofu w polu nazwiska użytkownika sprawiło, że polecenie SQL stało się nieprawidłowe:

```
INSERT INTO users (first_name,
 last_name, email, pass,
 registration_date) VALUES ('Peter',
'O'Toole', 'petey@example.net',
 SHA2('aPass8', 512), NOW() )
```

W tym przypadku to prawidłowe dane wpisane przez użytkownika doprowadziły do wygenerowania nieprawidłowego polecenia SQL. Gdyby jednak skrypt PHP na to pozwalał, to wrogo nastawiony użytkownik mógłby celowo wpisywać problematyczne znaki — na przykład apostrof — aby włamać się do bazy danych lub ją uszkodzić. Dlatego, ze względów bezpieczeństwa, funkcja ta powinna być stosowana dla wszystkich wartości pochodzących z pól tekstowych formularzy. Jej zastosowanie zademonstruję teraz na przykładzie skryptu rejestracji (listing 9.3).

Aby użyć funkcji mysqli_real_escape_string():

1. Jeśli jeszcze tego nie zrobiłeś, to otwórz plik *rejestracja.php* (listing 9.3) w edytorze lub IDE.
2. Przenieś wiersz dołączający plik *mysqli_connect.php* (wiersz 48. na listingu 9.3) zaraz za główną instrukcję warunkową `if` (listing 9.5).

   Ponieważ funkcja `mysqli_real_escape_string()` wymaga połączenia z bazą danych, to przed jej wywołaniem musi zostać wykonany skrypt *mysqli_connect.php*.

   Zmodyfikuj procedury weryfikujące dane tak, aby wykorzystywały tę funkcję. W tym celu zastąp każde wyrażenie `$var = trim($_POST['var'])` wyrażeniem `$var = mysqli_real_escape_data($dbc, trim($_POST['var']))`.

```
$fn = mysqli_real_escape_string
($dbc,trim($_POST['first_name']));
$ln = mysqli_real_escape_string
($dbc, trim($_POST['last_name']));
$e = mysqli_real_escape_string
($dbc, trim($_POST['email']));
$p = mysqli_real_escape_string ($dbc,
trim($_POST['pass1']));
```

Zamiast po prostu przypisać wprowadzone wartości do poszczególnych zmiennych (`$fn`, `$ln` itd.), przepuszczam je najpierw przez funkcję `mysqli_real_escape_data()`. Nadal stosuję najpierw funkcję `trim()`, aby pozbyć się niepotrzebnych odstępów.

Ciąg dalszy na stronie 317.

Listing 9.5. Skrypt rejestracja.php przepuszcza teraz wszystkie wprowadzone dane przez funkcję mysqli_real_escape_string(), aby zapewnić, że będzie można je bezpiecznie zastosować w poleceniu SQL

```
<?php # Listing 9.3 - rejestracja.php #2
// Ten skrypt wykonuje polecenie INSERT, by dodać wiersz do tabeli użytkowników.

$page_title = 'Zarejestruj się';
include ('includes/naglowek.html');

// Sprawdzamy, czy formularz został wysłany
if ($_SERVER['REQUEST_METHOD'] == 'POST') {

  require('../mysqli_connect.php'); // nawiązujemy połączenie z bazą danych

  $errors = array(); // inicjalizujemy tablicę błędów

  // sprawdzamy imię
  if (empty($_POST['first_name'])) {
    $errors[] = 'Zapomniałeś podać swoje imię!';
  } else {
    $fn = mysqli_real_escape_string($dbc, trim($_POST['first_name']));
  }

  // sprawdzamy nazwisko
  if (empty($_POST['last_name'])) {
    $errors[] = 'Zapomniałeś podać swoje nazwisko!';
```

Ciąg dalszy na następnej stronie.

Listing 9.5. Ciąg dalszy

```
24  } else {
25    $ln = mysqli_real_escape_string($dbc, trim($_POST['last_name']));
26  }
27
28  // sprawdzamy adres e-mail
29  if (empty($_POST['email'])) {
30    $errors[] = 'Zapomniałeś podać swój adres e-mail!';
31  } else {
32    $e = mysqli_real_escape_string($dbc, trim($_POST['email']));
33  }
34
35  // sprawdzamy, czy użytkownik wprowadził hasło i czy w obu polach jest ono takie samo
36  if (!empty($_POST['pass1'])) {
37    if ($_POST['pass1'] != $_POST['pass2']) {
38      $errors[] = 'Za drugim razem wpisałeś inne hasło!';
39    } else {
40      $p = mysqli_real_escape_string($dbc, trim($_POST['pass1']));
41    }
42  } else {
43    $errors[] = 'Zapomniałeś wprowadzić hasło!';
44  }
45
46  if (empty($errors)) { // jeśli formularz poprawnie wypełniony...
47
48    // rejestrujemy użytkownika w bazie danych
49
50    // tworzymy zapytanie
51    $q = "INSERT INTO users (first_name, last_name, email, pass,
        registration_date) VALUES ('$fn', '$ln', '$e', SHA1('$p'), NOW() )";
52    $r = @mysqli_query ($dbc, $q); // wykonujemy zapytanie
53    if ($r) { // jeśli poprawnie wykonane...
54
55      // wyświetlamy komunikat
56      echo '<h1>Dziękujemy!</h1>
57      <p>Zostałeś zarejestrowany. W rozdziale 12. będziesz mógł się
        zalogować!</p><p><br></p>';
58
59    } else { // jeśli zapytanie nie zostało poprawnie wykonane...
60
61      // wyświetlamy komunikat dla użytkownika
62      echo '<h1>Błąd systemu</h1>
63      <p class="error">Nie zostałeś zarejestrowany z powodu awarii
         naszego systemu.  Przepraszamy za kłopot.</p>';
64
65      // wyświetlamy komunikat uruchomieniowy
66      echo '<p>' . mysqli_error($dbc) . '<br><br>Zapytanie: ' . $q . '</p>';
67
68    } // koniec instrukcji if ($r).
69
```

Ciąg dalszy na następnej stronie.

3. Dodaj drugie wywołanie `mysqli_close()` przed końcem głównej instrukcji warunkowej.
`mysqli_close($dbc);`

   Ponieważ pierwszą operacją tego wyrażenia jest nawiązanie połączenia z bazą danych, to ostatnią, konsekwentnie, powinno być jego zamknięcie. Nadal połączenie powinno być również zamykane przed dołączeniem stopki i zakończeniem skryptu (wiersze 73. i 74.).

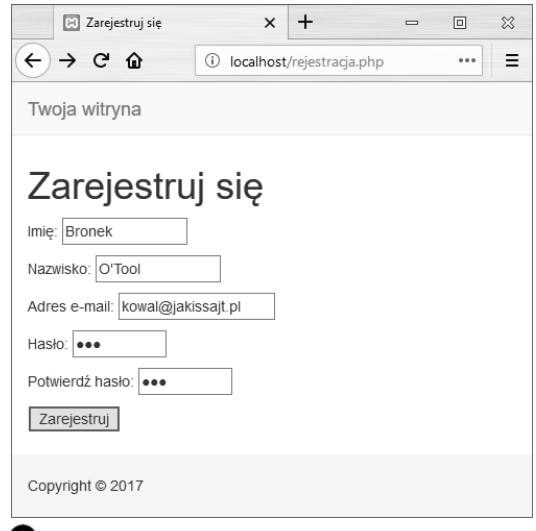
Listing 9.5. Ciąg dalszy

```
 70     mysqli_close($dbc); // zamykamy połączenie z bazą danych
 71
 72     // dołączamy stopkę i kończymy działanie skryptu.
 73     include ('includes/stopka.html');
 74     exit();
 75
 76   } else { // wyświetlamy błędy
 77
 78     echo '<h1>Błąd!</h1>
 79     <p class="error">Wystąpiły następujące błędy:<br>';
 80     foreach ($errors as $msg) { // wyświetlamy każdy komunikat o błędzie
 81       echo " - $msg<br>\n";
 82     }
 83     echo '</p><p>Spróbuj jeszcze raz.</p><p><br></p>';
 84
 85   } // koniec instrukcji if (empty($errors))
 86
 87   mysqli_close($dbc); // zamykamy połączenie z bazą danych
 88
 89 } // koniec głównej instrukcji warunkowej (wysłania formularza)
 90 ?>
 91 <h1>Zarejestruj się</h1>
 92 <form action="rejestracja.php" method="post">
 93   <p>Imię: <input type="text" name="first_name" size="15" maxlength="20"
        value="<?php if (isset($_POST['first_name'])) echo $_POST['first_name'];
        ?>" ></p>
 94   <p>Nazwisko: <input type="text" name="last_name" size="15" maxlength="40"
        value="<?php if (isset($_POST['last_name'])) echo $_POST['last_name'];
        ?>" ></p>
 95   <p>Adres e-mail: <input type="text" name="email" size="20" maxlength="80"
        value="<?php if (isset($_POST['email'])) echo $_POST['email']; ?>" ></p>
 96   <p>Hasło: <input type="password" name="pass1" size="10" maxlength="20"
        value="<?php if (isset($_POST['pass1'])) echo $_POST['pass1']; ?>" ></p>
 97   <p>Potwierdź hasło: <input type="password" name="pass2" size="10"
        maxlength="20" value="<?php if (isset($_POST['pass2']))
        echo $_POST['pass2']; ?>" ></p>
 98   <p><input type="submit" name="submit" value="Zarejestruj" ></p>
 99 </form>
100 <?php include ('includes/stopka.html'); ?>
```

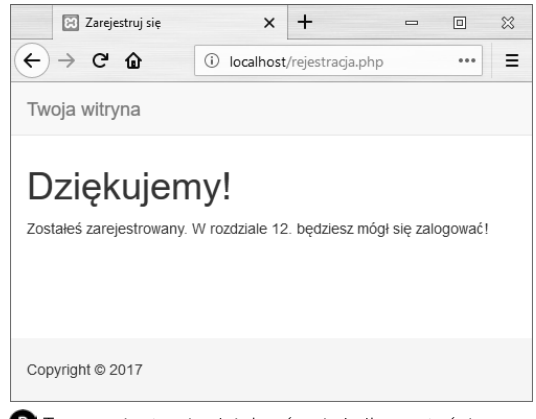
4. Zapisz plik pod nazwą *rejestracja.php*, wgraj go na serwer i przetestuj w przeglądarce (patrz A i B).

Wskazówki

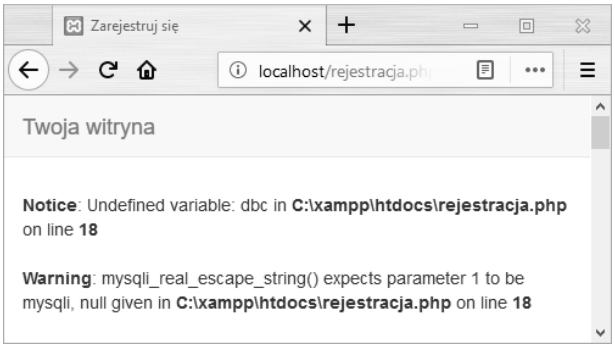
- Funkcja `mysqli_real_escape_string()` zabezpiecza łańcuchy zgodnie z regułami używanego języka (np. uporządkowaniem alfabetycznym), co daje jej niewątpliwą przewagę nad alternatywnymi rozwiązaniami.
- Jeżeli widzisz takie komunikaty jak na rysunku C, oznacza to, że funkcja `mysqli_real_escape_string()` nie może uzyskać dostępu do bazy danych (ze względu na brak połączenia reprezentowanego przez `$dbc`).
- Jeśli przejrzysz zawartość bazy (używając klienta mysql lub aplikacji takiej jak phpMyAdmin), to nie zobaczysz problematycznych znaków, które podczas zapisu zostały poprzedzone odwrotnymi ukośnikami. To prawidłowe działanie. Odwrotne ukośniki zabezpieczają polecenie SQL przed problematycznymi znakami, choć same nie są zapisywane w bazie danych.



A Wartości zawierające apostrofy nie spowodują błędu zapytania INSERT, jeśli zastosujesz do nich funkcję mysqli_real_escape_string()



B Teraz rejestracja działa również dla wartości zawierających problematyczne znaki i jest bezpieczniejsza



C Ponieważ funkcja mysqli_real_escape_string() potrzebuje aktywnego połączenia z bazą danych, niewłaściwe korzystanie z niej (przed dołączeniem skryptu nawiązującego połączenie) może prowadzić do wystąpienia kolejnych błędów

Zliczanie zwróconych rekordów

Następną w kolejności funkcją, którą należałoby omówić, jest `mysqli_num_rows()`. Zwraca ona liczbę rekordów odczytanych przez zapytanie `SELECT`, a pobiera zmienną przechowującą wyniki zapytania.

```
$num = mysqli_num_rows($r);
```

Funkcja ta jest wyjątkowo przydatna w wielu zastosowaniach. Jej użycie jest konieczne, jeśli chcesz wyświetlić wynik zapytania na wielu stronach (przykład znajdziesz w następnym rozdziale). Warto również, abyś zastosował ją, zanim spróbujesz pobrać wyniki zapytania w pętli `while` (ponieważ nie ma sensu pobieranie wyników, jeśli zapytanie nie zwróciło żadnych; wtedy próba pobrania wyników może spowodować błędy). Teraz zmodyfikuję skrypt *pokaz_uzytkownikow.php* w taki sposób, aby wyświetlał on całkowitą liczbę zarejestrowanych użytkowników.

Aby zmodyfikować plik pokaz_uzytkownikow.php:

1. Jeśli jeszcze tego nie zrobiłeś, to otwórz plik *pokaz_uzytkownikow.php* w edytorze tekstów lub IDE (patrz listing 9.4).
2. Zaraz przed instrukcją warunkową `if ($r)` wprowadź następującą instrukcję (listing 9.6):

   ```
   $num = mysqli_num_rows($r);
   ```

   Przypisze ona liczbę zwróconych rekordów do zmiennej `$num`.
3. Zmień pierwotne wyrażenie warunkowe `$r` na `if ($num > 0) {`

   Dotychczasowe wyrażenie warunkowe bazowało na tym, czy udało się wykonać zapytanie, a nie na tym, czy zwrócono jakiekolwiek rekordy. Teraz będzie ono bardziej precyzyjne.
4. Zanim utworzysz tabelę HTML, wyświetl liczbę zarejestrowanych użytkowników.

   ```
   echo "<p>W chwili obecnej mamy $num
   ↪zarejestrowanych użytkownikow.</p>\n";
   ```

Listing 9.6. Skrypt pokaz_uzytkownikow.php będzie teraz wyświetlał całkowitą liczbę zarejestrowanych użytkowników dzięki zastosowaniu funkcji mysqli_num_rows()

```
<?php # Listing 9.6 - pokaz_uzytkownikow.php #2
// Ten listing odczytuje wszystkie rekordy z tabeli users.

$page_title = 'Zobacz zarejestrowanych użytkowników';
include ('includes/naglowek.html');

// wyświetlamy nagłówek strony
echo '<h1>Zarejestrowani użytkownicy</h1>';

require_once ('../mysqli_connect.php'); // nawiązujemy połączenie z bazą danych

// tworzymy zapytanie
$q = "SELECT CONCAT(last_name, ', ', first_name) AS name,
    DATE_FORMAT(registration_date, '%d %M %Y') AS dr FROM users ORDER BY
    registration_date ASC";
$r = @mysqli_query ($dbc, $q); // wykonujemy zapytanie

// określamy liczbę zwróconych rekordów
$num = mysqli_num_rows($r);

```

Ciąg dalszy na następnej stronie.

Listing 9.6. Ciąg dalszy

```
19 if ($num > 0) { // jeżeli zapytanie zwróciło jakieś rekordy, to je wyświetlamy
20
21   // wyświetlamy liczbę użytkowników
22   echo "<p>W chwili obecnej mamy $num zarejestrowanych użytkowników.</p>\n";
23
24   // wyświetlamy nagłówek tabeli
25   echo '<table width="60%">
26   <thead>
27   <tr>
28     <th align="left">Nazwisko</th>
29     <th align="left">Data rejestracji</th>
30   </tr>
31   </thead>
32   <tbody>
33 ';
34
35   // pobieramy i wyświetlamy wszystkie rekordy
36   while ($row = mysqli_fetch_array($r, MYSQLI_ASSOC)) {
37     echo '<tr><td align="left">' . $row['name'] . '</td><td align="left">'
       . $row['dr'] . '</td></tr>
38     ';
39   }
40
41   echo '</tbody></table>'; // zamykamy tabelę
42
43   mysqli_free_result ($r); // zwalniamy zasoby
44
45 } else { // jeżeli zapytanie nie zostało wykonane pomyślnie...
46
47   echo '<p class="error">Aktualnie nie ma żadnych zarejestrowanych
       użytkowników.</p>';
48
49 }
50
51 mysqli_close($dbc); // zamykamy połączenie z bazą danych
52
53 include ('includes/stopka.html');
54 ?>
```

Modyfikacja skryptu rejestracja.php

W skrypcie *rejestracja.php* możesz zastosować funkcję `mysqli_num_rows()`, aby zapobiec wielokrotnemu zarejestrowaniu się użytkownika o tym samym adresie e-mail. Chociaż indeks `UNIQUE` założony na kolumnie przechowującej adres e-mail zapobiegnie takiej sytuacji na poziomie bazy danych, to jednak próba wstawienia kolejnego rekordu o takiej samej wartości tej kolumny spowoduje błąd MySQL-a. Aby uniknąć tego już na poziomie skryptu PHP, wystarczy najpierw wykonać dodatkowe zapytanie `SELECT`, aby sprawdzić, czy adres wprowadzony przez użytkownika nie jest już zarejestrowany w bazie. Zapytanie to będzie mieć całkiem prostą postać:

```
SELECT user_id FROM users WHERE email='$e'
```

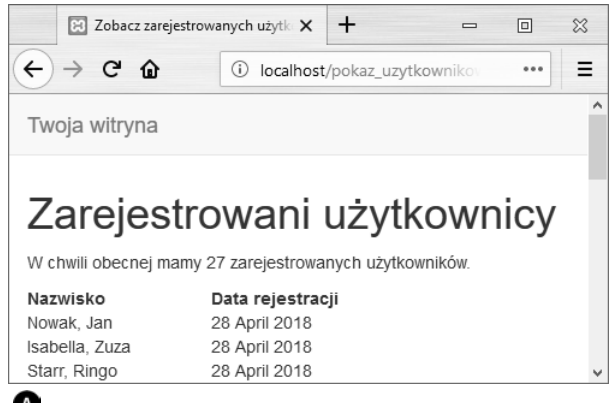
Takie zapytanie powinieneś wykonać (używając w tym celu funkcji `mysqli_query()`), a następnie wywołać funkcję `mysqli_num_rows()`. Jeśli wywołanie tej drugiej funkcji zwróci wartość `0`, będziesz miał pewność, że adres e-mail nie jest zarejestrowany w bazie i można wykonać polecenie `INSERT`.

5. Zmień klauzulę `else` głównej instrukcji warunkowej

   ```
   echo '<p class="error">Aktualnie nie ma
   ↪żadnych zarejestrowanych użytkowników.</p>';
   ```

   Pierwotne wyrażenie warunkowe opierało się tylko na tym, czy zapytanie zostało prawidłowo wykonane, czy nie. Teraz zakładamy, że zapytanie zostało wystarczająco przetestowane i poprzedni komunikat o błędach nie jest już potrzebny. Obecny komunikat o błędzie informuje jedynie, że nie zostały zwrócone żadne rekordy.

6. Zapisz plik pod nazwą *pokaz_uzytkownikow.php*, wgraj go na serwer i przetestuj w przeglądarce internetowej (patrz A).



A Na górze strony jest teraz wyświetlana liczba zarejestrowanych użytkowników

Aktualizacja rekordów w PHP

Ostatnim zagadnieniem, które opiszę w tym rozdziale, jest uaktualnianie rekordów baz danych z poziomu skryptów PHP. Wymaga to użycia zapytania `UPDATE`, którego poprawne wykonanie można potwierdzić, wywołując funkcję `mysqli_affected_rows()`.

Podczas gdy funkcja `mysqli_num_rows()` zwraca liczbę rekordów wygenerowanych przez zapytanie `SELECT`, `mysqli_affected_rows()` zwraca liczbę rekordów, na które miało wpływ wykonanie zapytania `INSERT`, `UPDATE` lub `DELETE`. Używa się jej następująco:

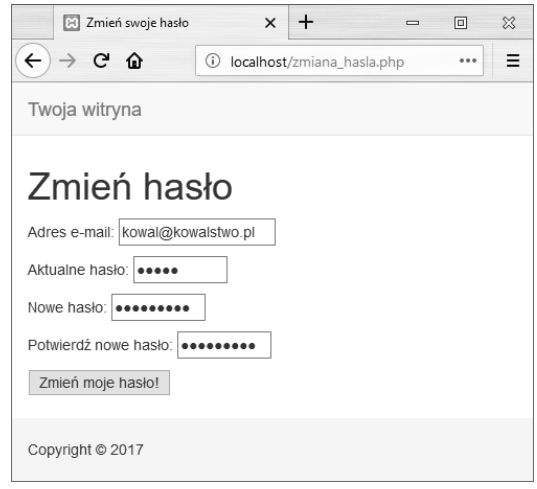
```
$num = mysqli_affected_rows($dbc);
```

Jedynym argumentem pobieranym przez tę funkcję jest identyfikator połączenia z bazą danych (`$dbc`). Nie potrzebuje ona natomiast wyników zwróconych przez ostatnie zapytanie (`$r`).

Utworzę teraz przykładowy skrypt, który pozwala zarejestrowanym użytkownikom zmienić swoje hasło. Zademonstruję tym samym dwie ważne techniki:

- Porównywanie nazwy użytkownika i hasła wprowadzonych przez użytkownika z wartościami przechowywanymi w bazie danych (kluczowe zagadnienie w systemie logowania).
- Uaktualnianie rekordów bazy danych przy odwoływaniu się do nich za pomocą klucza głównego.

Podobnie jak w przykładzie rejestracji użytkownika, również i ten skrypt PHP będzie wyświetlać formularz (patrz A) i obsługiwać go.



A Formularz służący do zmiany hasła

Aby uaktualniać rekordy w PHP:

1. W edytorze tekstów lub IDE utwórz nowy skrypt PHP, któremu nadasz nazwę *zmiana_hasla.php* (listing 9.7).

```
<?php # Skrypt 9.7 - zmiana_hasla.php
$page_title = 'Zmień swoje hasło';
include ('includes/naglowek.html');
```

2. Utwórz główną instrukcję warunkową.

```
if ($_SERVER['REQUEST_METHOD'] ==
  'POST') {
```

Ponieważ ta strona jednocześnie wyświetla i obsługuje formularz, zastosuję moje standardowe rozwiązanie do sprawdzania, czy formularz został przesłany, czy nie.

3. Dołącz skrypt nawiązujący połączenie z bazą danych i utwórz tablicę przechowującą komunikaty o błędach.

```
require('../mysqli_connect.php');
$errors = [];
```

Na początku skrypt będzie przypominał skrypt obsługujący formularz rejestracji.

4. Sprawdź wprowadzone dane.

```
if (empty($_POST['email'])) {
  $errors[] = 'Zapomniałeś
  wprowadzić adres e-mail!';
} else {
  $e = mysqli_real_escape_string
  ($dbc, trim($_POST['email']));
}
if (empty($_POST['pass'])) {
  $errors[] = 'Zapomniałeś
  wprowadzić dotychczasowe
  hasło!';
} else {
  $p = mysqli_real_escape_string(
  $dbc, trim($_POST['pass']));
}
```

Wszystkie zachodzące tu procesy są dokładnie takie same jak w skrypcie *rejestracja.php*. Formularz (patrz A) będzie miał cztery pola wejściowe: adres e-mail, obecne hasło, nowe hasło i pole pozwalające potwierdzić nowe hasło. Dane, które przejdą kontrolę poprawności, zostaną pozbawione zbędnych odstępów za pomocą funkcji `trim()` i przepuszczone przez funkcję `mysqli_real_escape_string()`, co zapewni ich bezpieczne użycie w zapytaniach SQL.

Ciąg dalszy na stronie 325.

Listing 9.7. Skrypt zmiana_hasla.php wykonuje polecenie UPDATE i używa funkcji mysqli_affected_rows(), aby potwierdzić wprowadzenie zmian

```
 1 <?php # Listing 9.7 - zmiana_hasla.php
 2 // Ten skrypt pozwala użytkownikowi zmienić hasło.
 3
 4 $page_title = 'Zmień swoje hasło';
 5 include ('includes/naglowek.html');
 6
 7 // Sprawdzamy, czy formularz został wysłany…
 8 if ($_SERVER['REQUEST_METHOD'] == 'POST') {
 9
10   require('../mysqli_connect.php'); // nawiązujemy połączenie z bazą danych
11
12   $errors = []; // inicjujemy tablicę błędów
13
14   // sprawdzamy adres e-mail
```

Ciąg dalszy na następnej stronie.

Listing 9.7. Ciąg dalszy

```
15  if (empty($_POST['email'])) {
16    $errors[] = 'Zapomniałeś podać swój adres e-mail!';
17  } else {
18    $e = mysqli_real_escape_string($dbc, trim($_POST['email']));
19  }
20
21  // sprawdzamy, czy użytkownik wprowadził dotychczasowe hasło
22  if (empty($_POST['pass'])) {
23    $errors[] = 'Zapomniałeś wprowadzić dotychczasowe hasło!';
24  } else {
25    $p = mysqli_real_escape_string($dbc, trim($_POST['pass']));
26  }
27
28  // sprawdzamy, czy użytkownik wprowadził w obu polach takie samo hasło
29  if (!empty($_POST['pass1'])) {
30    if ($_POST['pass1'] != $_POST['pass2']) {
31      $errors[] = 'Za drugim razem wpisałeś inne hasło!';
32    } else {
33      $np = mysqli_real_escape_string($dbc, trim($_POST['pass1']));
34    }
35  } else {
36    $errors[] = 'Zapomniałeś wprowadzić nowe hasło!';
37  }
38
39  if (empty($errors)) { // jeśli w formularzu nie było błędów…
40
41    // sprawdzamy, czy została wprowadzona poprawna kombinacja e-mail/hasło
42    $q = "SELECT user_id FROM users WHERE (email='$e' AND pass=SHA2('$p',
        512) )";
43    $r = @mysqli_query($dbc, $q);
44    $num = @mysqli_num_rows($r);
45    if ($num == 1) { // poprawna
46
47      // pobieramy id użytkownika
48      $row = mysqli_fetch_array($r, MYSQLI_NUM);
49
50      // tworzymy polecenie UPDATE
51      $q = "UPDATE users SET pass=SHA2('$np', 512) WHERE user_id=$row[0]";
52      $r = @mysqli_query($dbc, $q);
53
54      if (mysqli_affected_rows($dbc) == 1) { // jeżeli polecenie zostało wykonane poprawnie…
55
56        // wyświetlamy komunikat
57        echo '<h1>Dziękujemy!</h1>
58        <p>Hasło zostało zmienione. W rozdziale 12. będziesz mógł się
          zalogować!</p><p><br></p>';
59
60      } else { // jeżeli polecenie nie zostało wykonane poprawnie…
61
```

Ciąg dalszy na następnej stronie.

5. Sprawdź nowe hasło.

```
if (empty($_POST['pass1'])) {
   if ($_POST['pass1'] !=
     $_POST['pass2']) {
     $errors[] = 'Za drugim razem
     wpisałeś inne hasło!';
   }
   else {
     $np = mysqli_real_escape_string
    ($dbc, trim($_POST['pass1']));
   }
} else {
   $errors[] = 'Zapomniałeś wprowadzić
   nowe hasło!';
}
```

Również tutaj wszystko przebiega podobnie jak w skrypcie rejestracji z tą różnicą, że nowe hasło zostaje przypisane zmiennej `$np` (ponieważ `$p` reprezentuje bieżące hasło).

6. Jeżeli wszystkie dane przeszły pomyślnie weryfikację, odczytaj identyfikator użytkownika.

```
if (empty($errors)) {
  $q = "SELECT user_id FROM users
  WHERE (email='$e' AND pass=SHA2
   ('$p', 512) )";
  $r = @mysqli_query ($dbc, $q);
  $num = @mysqli_num_rows ($r);
```

Pierwsze zapytanie zwraca tylko zawartość pola `user_id` rekordu pasującego do podanego adresu e-mail i hasła (patrz B). Aby porównać wprowadzone hasło z tym przechowywanym w bazie danych, szyfruję je za pomocą funkcji `SHA2()`. Jeżeli użytkownik jest zarejestrowany w bazie i wpisał poprawny adres e-mail i hasło, zostanie zwrócona dokładnie jedna kolumna z jednego rekordu (ponieważ adres e-mail jest unikatowy). Na zakończenie rekord w postaci tablicy (jednoelementowej) zostanie przypisany do zmiennej `$row`.

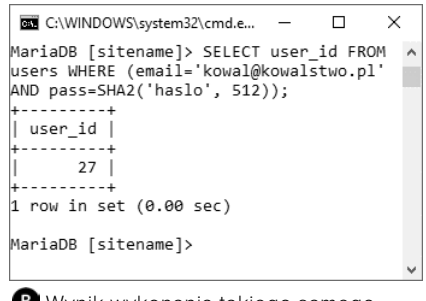
Jeśli ta część skryptu nie działa u Ciebie poprawnie, zastosuj standardowe metody uruchamiania i usuwania błędów: usuń operator `@`, aby sprawdzić, czy wystąpiły jakieś błędy; użyj funkcji `mysqli_error()` do raportowania błędów MySQL-a; wykonaj zapytania, używając innego interfejsu MySQL-a (jak na rysunku B).

7. Uaktualnij bazę danych.

```
$q = "UPDATE users
  SET pass=SHA2('$np', 512)
  WHERE user_id=$row[0]";
$re = @mysqli_query ($dbc, $q);
```

To zapytanie zmieni istniejące hasło na nowe w tym rekordzie, w którym zawartość kolumny `user_id` jest równa liczbie odczytanej w poprzednim zapytaniu.

Ciąg dalszy na stronie 327.



B Wynik wykonania takiego samego zapytania SELECT jak w skrypcie, ale za pomocą klienta mysql

Listing 9.7. Ciąg dalszy

```
62          // wyświetlamy komunikat dla użytkownika
63          echo '<h1>Błąd systemu</h1>
64          <p class="error">Hasło nie zostało zmienione z powodu awarii naszego
            systemu. Przepraszamy za kłopot.</p>';
65
66          // wyświetlamy komunikat uruchomieniowy
67          echo '<p>' . mysqli_error($dbc) . '<br><br>Zapytanie: ' . $q
            . '</p>';
68
69        }
70
71        mysqli_close($dbc); // zamykamy połączenie z bazą danych
72
73        // dołączamy stopkę i kończymy skrypt (aby nie wyświetlił formularza)
74        include ('includes/stopka.html');
75        exit();
76
77      } else { // użytkownik wpisał nieprawidłową kombinację e-mail/hasło.
78        echo '<h1>Błąd!</h1>
79        <p class="error">Podałeś niewłaściwy adres e-mail bądź hasło.</p>';
80      }
81
82    } else { // wyświetlamy komunikaty o błędach
83
84      echo '<h1>Błąd!</h1>
85      <p class="error">Wystąpiły następujące błędy:<br>';
86      foreach ($errors as $msg) { // wyświetlamy każdy komunikat o błędzie
87        echo " - $msg<br>\n";
88      }
89      echo '</p><p>Spróbuj jeszcze raz.</p><p><br></p>';
90
91    } // koniec instrukcji warunkowej if (empty($errors))
92
93    mysqli_close($dbc); // zamykamy połączenie z bazą danych
94
95 } // koniec głównej instrukcji warunkowej (wysłania formularza)
96 ?>
97 <h1>Zmień hasło</h1>
98 <form action="zmiana_hasla.php" method="post">
99    <p>Adres e-mail: <input type="email" name="email" size="20" maxlength="60"
         value="<?php if (isset($_POST['email'])) echo $_POST['email']; ?>" ></p>
100   <p>Aktualne hasło: <input type="password" name="pass" size="10"
         maxlength="20" value="<?php if (isset($_POST['pass']))
         echo $_POST['pass']; ?>" ></p>
101   <p>Nowe hasło: <input type="password" name="pass1" size="10" maxlength="20"
         value="<?php if (isset($_POST['pass1'])) echo $_POST['pass1']; ?>" ></p>
102   <p>Potwierdź nowe hasło: <input type="password" name="pass2" size="10"
         maxlength="20" value="<?php if (isset($_POST['pass2']))
         echo $_POST['pass2']; ?>" ></p>
103   <p><input type="submit" name="submit" value="Zmień moje hasło!"></p>
104 </form>
105 <?php include ('includes/stopka.html'); ?>
```

8. Sprawdź wyniki zapytania.

```
if (mysqli_affected_rows($dbc) == 1) {
  echo '<h1>Dziękujemy!</h1>
  <p>Hasło zostało zmienione.
  W rozdziale 12. będziesz mógł się
  zalogować!</p><p><br /></p>';
} else {
  echo '<h1>Błąd systemu</h1>
  <p class="error">Hasło nie zostało
  zmienione z powodu awarii naszego
  systemu. Przepraszamy za kłopot.
  </p>';
  echo '<p>' . mysqli_error($dbc)
  . '<br /><br />Zapytanie: ' . $q
  . '</p>';
}
```

Ta część skryptu również przypomina skrypt *rejestracja.php*. W przypadku gdy funkcja `mysqli_affected_rows()` zwraca liczbę 1, rekord jest uaktualniany i wyświetlany jest komunikat. Jeżeli zwróci ona inną wartość, zostanie wyświetlony komunikat o błędzie.

9. Dołącz stopkę i zakończ działanie skryptu.

```
include ('includes/stopka.html');
exit();
```

W tym momencie zapytanie `UPDATE` zostało już wykonane i zakończyło się sukcesem lub nie (ze względu na błąd systemu). W obu przypadkach nie trzeba ponownie wyświetlać formularza i dlatego dołączona zostaje stopka, a wywołanie funkcji `exit()` kończy działanie skryptu. Wcześniej jednak, dla porządku, zamykamy połączenie z bazą danych.

10. Dokończ instrukcję warunkową `if ($num == 1)`:

```
} else {
  echo '<h1>Błąd!</h1>
  <p class="error">Podałeś
  niewłaściwy adres e-mail
  bądź hasło.</p>';
}
```

Jeśli funkcja `mysqli_num_rows()` zwróci wartość różną od 1, oznacza to, że podany adres e-mail i hasło nie zgadzają się z zapisanymi w bazie. W takiej sytuacji zostanie wyświetlony stosowny komunikat o błędzie oraz formularz, tak by użytkownik mógł wpisać poprawne informacje.

11. Wyświetl ewentualne komunikaty o błędach.

```
} else {
  echo '<h1>Błąd!</h1>
  <p class="error">Pojawiły się
  następujące błędy:<br />';
  foreach ($errors as $msg) {
    echo " - $msg<br />\n";
  }
  echo '</p><p>Spróbuj jeszcze
  raz.</p><p><br></p>';

}
```

Powyższa klauzula `else` zostanie wykonana, gdy tablica `$errors` nie będzie pusta (co oznacza, że dane formularza nie przeszły pomyślnie wszystkich testów). Podobnie jak w przypadku strony rejestracji, błędy zostają wyświetlone.

12. Zamknij połączenie z bazą danych i zakończ kod PHP.

```
  mysqli_close($dbc);
}
?>
```

Ciąg dalszy na następnej stronie.

13. Wyświetl formularz.

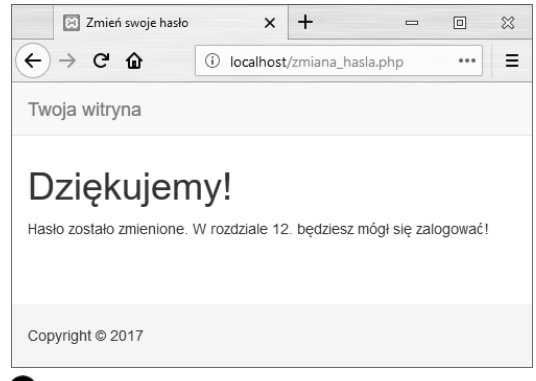
```
<h1>Zmień hasło</h1>
<form action="zmiana_hasla.php"
method="post">
  <p>Adres e-mail: <input
    type="email" name="email"
    size="20" maxlength="60"
    value="<?php if
    (isset($_POST['email']))
    echo $_POST['email']; ?>" >
    </p>
  <p>Aktualne hasło: <input
    type="password" name="pass"
    size="10" maxlength="20"
    value="<?php if
    (isset($_POST['pass']))
    echo $_POST['pass']; ?>" ></p>
  <p>Nowe hasło: <input
    type="password" name="pass1"
    size="10" maxlength="20"
    value="<?php if
    (isset($_POST['pass1']))
    echo $_POST['pass1']; ?>" >
    </p>
  <p>Potwierdź nowe hasło: <input
    type="password" name="pass2"
    size="10" maxlength="20"
    value="<?php if
    (isset($_POST['pass2']))
    echo $_POST['pass2']; ?>" >
    </p>
  <p><input type="submit"
    name="submit" value="Zmień
    moje hasło!"></p>
</form>
```

Formularz zawiera trzy pola służące do podawania hasła — aktualnego, nowego i jego potwierdzenia — a także adresu e-mail. Wprowadzony adres e-mail jest zachowywany w formularzu, co oczywiście nie jest wskazane w przypadku haseł. Wszystkie wartości są przekazywane i wyświetlane po ponownym przesłaniu formularza.

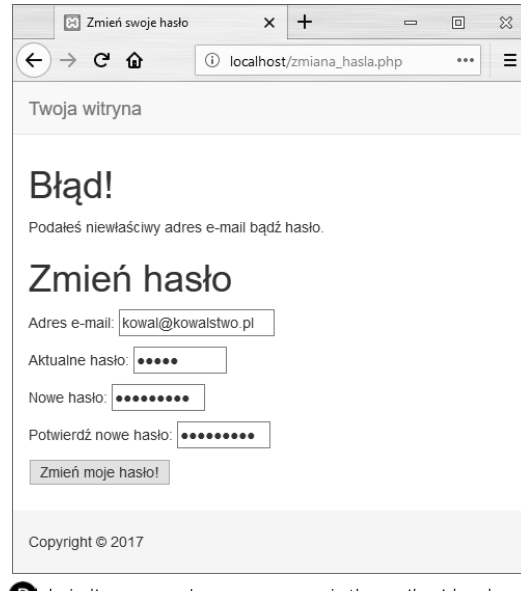
14. Dołącz plik stopki.

```
<?php
include ('includes/stopka.html');
?>
```

15. Zapisz plik pod nazwą *zmiana_hasla.php*, wgraj go na serwer i przetestuj w przeglądarce internetowej (patrz C i D).



C Hasło w bazie danych zostało zmienione



D Jeżeli wprowadzona nazwa użytkownika i hasło nie zgadzają się z informacjami przechowywanymi w bazie danych, hasło nie zostanie uaktualnione

Wskazówki

- Jeżeli usuniesz wszystkie rekordy występujące w tabeli za pomocą polecenia `TRUNCATE` *`nazwatabeli`*, funkcja `mysqli_affected_rows()` zwróci wartość `0`. Nawet wtedy, gdy zapytanie zostało wykonane prawidłowo i rekordy zostały usunięte.
- Jeżeli wykonujesz zapytanie `UPDATE`, które nie zmienia faktycznej wartości przechowywanej w kolumnie (na przykład zastępujesz bieżące hasło jego wierną kopią), `mysqli_affected_row()` zwróci wartość `0`.
- Użyte tu wyrażenie warunkowe `mysqli_` ↪`affected_rows()` można by też zastosować (i pewnie wypadałoby) w skrypcie *rejestracja.php*, ponieważ jest ono dokładniejsze niż `if ($r)`.

Podsumowanie i kontynuacja

Podsumowanie

- Której wersji PHP używasz? Której wersji MySQL-a używasz? Czy używane wersje PHP i MySQL-a pozwalają na korzystanie z ulepszonej wersji rozszerzenia MySQL?
- Jaka jest najważniejsza sekwencja czynności służących do debugowania problemów PHP-MySQL (opisana szczegółowo w rozdziale 8., „Obsługa i usuwanie błędów").
- Jakiej nazwy komputera, nazwy użytkownika i hasła używasz do nawiązywania połączenia z serwerem MySQL?
- Jaki kod PHP pozwala na nawiązanie połączenia z serwerem MySQL, wybranie bazy danych i ustawienie kodowania?
- Jakiego kodowania używasz? Dlaczego skrypt PHP podczas zapisywania tekstu w bazie musi używać tego samego kodowania, które jest używane do interakcji z serwerem MySQL?
- Dlaczego preferowane jest umieszczanie skryptu *mysql_connect.php* poza drzewem katalogu głównego serwera WWW? Co to jest za katalog?
- Dlaczego na witrynach produkcyjnych nie należy wyświetlać komunikatów o błędach MySQL ani o wykonywanych zapytaniach?
- Jaki kod jest niemal zawsze używany do obsługi wyników zwracanych przez polecenia `SELECT`? Jakiej składni można by użyć, jeśli polecenie `SELECT` zwróci tylko jeden wiersz wyników?
- Dlaczego ważne jest stosowanie funkcji `mysqli_real_escape_string()`?
- Po jakich poleceniach SQL można używać funkcji `mysqli_num_rows()`?
- Po jakich poleceniach SQL można używać funkcji `mysqli_affected_rows()`?

Kontynuacja

- Jeśli nie pamiętasz, jak działa stosowany przeze mnie system szablonów lub jak należy używać funkcji `include()`, to zajrzyj do rozdziału 3.
- Skorzystaj z informacji podanych w rozdziale 8., aby dodać do przedstawionych tu skryptów własną funkcję obsługi błędów.
- Zmień sposób użycia funkcji `mysqli_num_rows()` w pliku *pokaz_uzytkownikow.php*, tak by była ona wywoływana wyłącznie w przypadku, gdy wykonanie zapytania zwróci wartość `TRUE`.
- Użyj funkcji `mysqli_num_rows()` w skrypcie *rejestracja.php*, zgodnie z sugestiami podanymi w ramce „Modyfikacja skryptu rejestracja.php".
- Użyj funkcji `mysqli_affected_rows()` w skrypcie *rejestracja.php*, aby upewnić się, że polecenie `INSERT` zostało prawidłowo wykonane.
- Jeśli chcesz, to napisz skrypty korzystające z bazy danych ***`banking`***. Prosty projekt mógłbyś zacząć od przygotowania skryptów umożliwiających wyświetlenie listy klientów, wyświetlenie wszystkich kont (użyj złączenia, by wyświetlać także personalia klientów) oraz formularza pozwalającego na modyfikowanie stanu konta.

Skorowidz

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

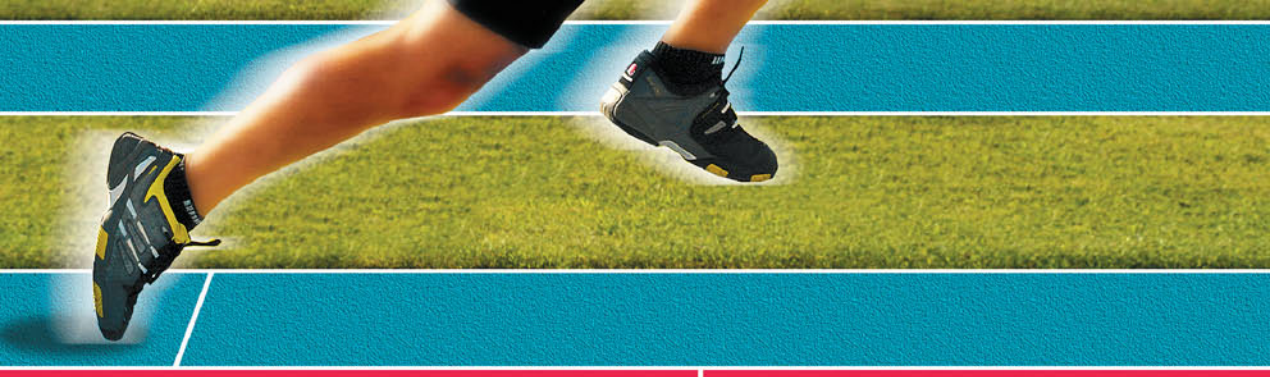
W

X

Z

Ż

Notatki



Szybki start

PHP i MySQL — wydajne, niezawodne, stabilne działanie!

Język PHP i serwer bazy danych MySQL stały się nieformalnym standardem tworzenia dynamicznych witryn, które korzystają z baz danych. Mimo że istnieje wiele innych konkurencyjnych technologii, rzesza programistów aplikacji internetowych decyduje się właśnie na to rozwiązanie. Otwarte źródła, dostępność, coraz szersze możliwości, elastyczność i szybkość, konsekwentny rozwój — to tylko kilka zalet PHP i MySQL. Projektanci dynamicznych stron WWW wysoce sobie cenią te zalety, o czym świadczy choćby stale wzrastająca liczba witryn WWW, które napisano z wykorzystaniem tych technologii.

Ta książka jest doskonałym poradnikiem dla projektantów dynamicznych stron WWW. Szczególnie przydatna okaże się dla osób dopiero nabierających wprawy w programowaniu. W przejrzysty sposób opisano podstawy języków PHP oraz SQL. Przedstawiono zasady poprawnego konfigurowania serwerów PHP i MySQL. Uwzględniono zagadnienia dotyczące obsługi i usuwania błędów, a także przeprowadzania testów aplikacji. Szeroko potraktowano kwestie bezpieczeństwa aplikacji. Znalazło się tu również sporo informacji o integracji PHP z takimi technologiami jak JavaScript, jQuery, Perl czy Ajax. Przy tym wszystkim książka jest napisana w sposób bardzo przystępny, a liczne przykłady znakomicie ułatwiają zrozumienie prezentowanych treści.

Najważniejsze zagadnienia przedstawione w książce:

- Podstawy PHP i MySQL
- Zaawansowane zagadnienia i tworzenie zoptymalizowanego kodu
- Ukryte pola, stronicowanie wyników i inne techniki programistyczne
- Korzystanie z ciasteczek i sesje
- Zabezpieczenia aplikacji

Larry Ullman jest prezesem spółki Digital Media and Communications, firmy specjalizującej się w technologiach informacyjnych. Jest też autorem świetnych książek, programistą, trenerem, mówcą i konsultantem. Programuje dynamiczne strony WWW i zajmuje się bazami danych. Posiada rzadki dar tłumaczenia złożonych tajników programowania w prosty, zrozumiały i bardzo użyteczny sposób. Mimo to uparcie twierdzi, że z całą pewnością nie jest geekiem.

ISBN 978-83-283-4466-2
9 788328 344662
Cena: 99,00 zł